

An Introduction to Python Programming

Author: Ian D Chivers

Date: Tuesday, 25th April, 2023

Naming trivia - Python's name is derived from Monty Python, whom Python's creator Guido van Rossum obviously had watched. Monty Python references appear in Python code and culture, and spam and eggs are used instead of the traditional foo and bah used in other languages. Visit

https://en.wikipedia.org/wiki/Monty_Python

for more information.

History

I ended up writing these notes because of hearing about Python, and how good it was for numeric and scientific computing. Here is a comment about Python from an enthusiast:

I would recommend the Python route as it is the numerical programming language du jour. It competes well with the numerical IDEs like Matlab (with interaction and graphics) and also with the traditional compiled languages. Just about every scientific discipline has a set of Python applications, tools and/or libraries, many of which are bundled with some of the science-orientated Linux distributions. As it comes with its own package management, it is very easy for people to pull down all the libraries they need without too much head-scratching - John Pelan (Head of Scientific Computing at the Wellcome Centre, University College London).

The examples have (in the main) been run on both Windows and Linux.

The original notes were written in 2015. They have been updated several times in response to comments made by delegates and also as Python has evolved.

December 2017

Updated to reflect changes that have been made to the multiprocessing library.

Added a brief coverage of QT Creator.

March 2018, with feedback from Nick and Poppy.

Semi colons removed amongst other changes.

Added cartopy example as a replacement for the basemap plots.

April 2018.

Minor improvements.

Added C++ and Java programs to the performance comparison chapter.

July 2018.

Updated to correct Acrobat navigation.

September 2018.

Added some additional material on Anaconda on Windows.

September 2018.

Updated the installation details to have more information about how to run some of the examples.

Added coverage of Spyder.

Added some additional matplotlib and cartopy examples.

Added details of various parts of the Python apis used in the examples.

Updated chapter 27 to have summary timing figures for several compilers.

Add a chapter on installing and calling the Nag library.

Corrected page size issues across the postscript print file and Adobe Acrobat Distiller.

Also corrected page tag issues.

Continued:

January 2019.

Added the Monty Python details.

February 2019.

Corrections and updates.

March 2019.

Updated the version information to bring up to date with the latest Python distributions.

Added an example summary chapter.

April 2019.

Added additional examples in the following chapters:

I/O, csv, internal writes;

Graphics and matplotlib.

New examples in the SQL chapter based on the Met Office historic data records.

April 2019.

Corrected two d numpy array example, chapter 6.

Added OO Met Office example, chapter 10.

Added notes about several Unix commands in chapter 11.

Added genfromtxt example, chapter 11.

Added example in the sql chapter using genfromtxt and numpy arrays that calculates monthly averages.

May 2019.

Added two examples to the string chapter.

Added 4 examples on Pandas working with the Met Office station data to the Scipy chapter.

Updated the multiprocessing pool examples to give the overall parallel time including pool creation and pool closure figures.

Updated examples on web site to match latest notes.

Added details of the Windows subsystem for linux offering from Microsoft. It is easy to add Python to the Ubuntu distribution.

Added details of using Microsoft Visual Studio Community Edition for Python development. Visual Studio is Python aware. There is coverage of Visual Studio 2017 and 2019.

Added details of Intel Python.

Added details of Windows Hyper-V and Python.

Added references and citations for Cartopy and the map data used.

Continued:

September 2019.

Added two new graphics plots.

Updated the run time table in chapter 27.

January 2020

Updated sample numbering. Added extra material on handling Excel stored data

April 2020

Corrected blank pages issue.

September 2020

Updated the timing figures comparing Python to compiled languages.

April 2023

Updated to bring in line with current systems.

Corrected some of the source files.

Generated updated example summary table.

Updated the Python source file examples to reflect the current notes contents.

Table of Contents

1 Overview	13
1.1 Aims.....	13
1.2 History.....	13
1.3 Use.....	14
1.4 Assumptions.....	16
1.5 Web resources.....	16
1.6 Downloading and installing the software.....	16
1.7 Windows.....	17
1.7.1 Windows and anaconda.....	17
1.7.1.1 Accessing Anaconda and Python on Windows.....	34
1.7.2 Windows - cygwin python version.....	36
1.7.3 Windows - Python download.....	41
1.7.4 Windows and Microsoft Visual Studio.....	41
1.7.4.1 Visual Studio Community Edition 2019.....	43
1.7.5 Windows subsystem for Linux and Python install.....	44
1.7.6 Windows Hyper-V manager.....	45
1.8 Linux.....	47
1.8.1 Python and openSuSe.....	47
1.8.2 Python, openSuSe and an anaconda installation.....	47
1.9 Intel Python for Windows, Linux and Mac.....	47
1.10 Mapping with Python - basemap.....	47
1.11 Mapping with Python - Cartopy.....	48
1.12 Python on line documentation.....	49
1.12.1 Published books and on line electronic manuscripts.....	61
1.13 Download and installation summary.....	63
1.13.1 Summary of systems setups.....	63
1.14 Course Details.....	66
1.15 Problems.....	66
2 An Introduction to Python	67
2.1 Example 1 - Hello World.....	67
2.2 Example 2 - Simple text I/O using Python style strings.....	67
2.3 Example 3 - Simple numeric I/O.....	68
2.4 Running the examples using jupyter qtconsole.....	68
2.5 Using spyder.....	71
2.6 Problems.....	72
3 Python base types, operators and expressions	73
3.1 Built-in Types.....	73
3.2 Python symbols.....	73
3.2.1 Operators.....	73
3.2.2 Delimiters and other characters.....	74
3.3 Numeric Types — int, float, complex.....	74
3.4 Iterator Types.....	75
3.5 Sequence Types.....	75
3.6 Text Sequence Type - str.....	75
3.7 Binary sequence types - bytes, bytearray, memoryview.....	76
3.8 Set types - set, frozenset.....	76
3.9 Mapping types - dict.....	76
3.10 Context manager types.....	76
3.11 Other types.....	76
3.12 Problems.....	76
4 Arithmetic	77
4.1 Example 1 - assignment and division.....	77
4.2 Example 2 - division with integers.....	78
4.3 Example 3 - time taken to reach the earth from the Sun.....	78
4.4 Example 4 - converting from Fahrenheit to centigrade.....	79
4.5 Example 5 - converting from Centigrade to Fahrenheit.....	79
4.6 Example 6 - numbers getting too large - overflow.....	79

4.7	Example 7 - numbers getting too small - underflow	79
4.8	Example 8 - subtraction of two similar values.....	80
4.9	Example 9 - summation.....	80
4.10	Absolute and relative errors	81
4.11	Problems.....	81
4.12	Bibliography	82
5	Arrays using the array module.....	84
5.1	Array methods	84
5.2	Array size known at compile time	87
5.2.1	Example 1 - array and conventional for loop syntax.....	87
5.2.2	Example 2 - using the len function to determine the size of array	88
5.3	Array size known at run time.....	88
5.3.1	Example 3 - reading in the array size.....	88
5.4	Summary	89
5.5	Problems.....	89
6	Arrays using the Numpy module	91
6.1	Documentation.....	94
6.2	Creating arrays	95
6.3	Simple 1 and 2 d array examples	96
6.3.1	Example 1 - simple rainfall example.....	96
6.3.2	Example 2 - variant of one using len intrinsic function.....	96
6.3.3	Example 3 - setting the size at run time.....	96
6.3.4	Example 4 - two d array using numpy.zeros method.....	97
6.3.5	Example 5 - two d array using numpy.array() method	97
6.3.6	Example 6 - two d array and the numpy.sum() method.....	97
6.4	Simple 1 and 2 d array slicing.....	100
6.4.1	Example 7 - simple one d slicing.....	100
6.4.2	Example 8 - two d slicing.....	100
6.4.3	Example 9 - arithmetic and slicing.....	101
6.5	Miscellaneous examples: aggregate, reshape, copies and views.....	102
6.5.1	Example 10 - Aggregate usage	102
6.5.2	Example 11 - Shape manipulation.....	102
6.5.3	Example 12 - Copies or views	103
6.6	Numpy documentation.....	104
6.7	Problems.....	104
7	Text in Python: Strings	105
7.1	Introduction	105
7.2	String Methods	105
7.3	String example 1 - initialisation, len and find methods.....	110
7.4	String example 2 - concatenation and split method	110
7.5	String example 3 - split variant.....	112
7.6	String example 4 - reading from an external file.....	112
7.7	String example 5 - reading data from a file and calculating sum and average rainfall values	115
7.8	String example 6 - simple variant of the previous example using the .format option	117
7.9	Character data in Python.....	118
7.10	String example 7 - the ASCII character set	118
7.11	Unicode.....	119
7.12	String example 8 - Unicode characters	120
7.13	Example 9 - another unicode example.....	121
7.14	Problems.....	123
8	Control Structures - compound statements	125
8.1	Compound statements.....	125
8.2	The if statement.....	126
8.3	The while statement	126
8.4	The for statement	126
8.5	The try statement	127

8.6	The with statement	129
8.7	The pass statement	130
8.8	Example 1 - the if statement	130
8.9	Example 2 - the while statement	131
8.10	Example 3 - the for loop with arrays	131
8.11	Example 4 - the for loop with lists and enumerate	132
8.12	Example 5 - the for in statement	133
8.13	Example 6 - try and except	133
8.14	Additional material	133
8.15	Problems	133
8.16	Bibliography	135
9	Functions	136
9.1	Example 1 - a bigger function	136
9.2	Example 2 - a swap function	137
9.3	Example 3 - another swap	137
9.4	Example 4 - yet another swap	138
9.5	Example 5 - recursive functions	138
9.6	Example 6 - simple factorial variant, reading the value in	138
9.7	Intrinsic maths functions	139
9.7.1	math — Mathematical functions	139
9.7.1.1	Number-theoretic and representation functions	139
9.7.1.2	Power and logarithmic functions	141
9.7.1.3	Trigonometric functions	142
9.7.1.4	Angular conversion	142
9.7.1.5	Hyperbolic functions	142
9.7.1.6	Special functions	142
9.7.1.7	Constants	142
9.8	Example 7 - testing out the maths functions	143
9.9	Example 8 - math module sin function	145
9.10	Example 9 - math module using numpy arrays	146
9.11	Example 10 - math module using a pi shortcut	146
9.12	Fibonacci implementations	147
9.13	Example 11 - Using generators	148
9.14	Example 12 - Iterative	148
9.15	Example 13 - Recursive	148
9.16	Functional programming in Python	148
9.17	Example 14 - generating prime numbers	149
9.18	Example 15 - list and lambda usage	149
9.19	Example 16 - functional example	150
9.20	Example 17 - functional example	151
9.21	Example 18 - functional example variant using the array module	152
9.22	Example 19 - functional variant using the numpy module	152
9.23	Problems	153
10	Object oriented programming and classes in Python	154
10.1	Example 1 - base shape class	154
10.2	Example 2 - variation using modules	155
10.3	Example 3 - a circle derived class	156
10.4	Example 4 - test program for the shape and circle classes	157
10.5	Example 5 - polymorphism and dynamic binding	158
10.6	Example 6 - data structuring using the Met Office data	159
10.7	Problems	161
11	IO	162
11.1	Example 1 - reading from a file using substrings	162
11.2	Example 2 - reading the same file using the split() method	164
11.3	Example 3 - internet file read	165
11.4	Example 4 - variation on the internet file read where we save the file	166
11.5	Example 5 - reading all of the station data files with timing	167
11.6	Example 6 - Writing to a set of files names generated within Python	170

11.7	Example 7 - Copying a file and replacing missing values	170
11.8	Example 8 - creating an SQL file	170
11.9	Example 9 - Creating a csv file	171
11.10	Example 10 - CSV files and the csv module	172
11.11	Example 11 - CSV usage and data extraction	174
11.12	Example 12 - reading a met office file using the csv module	175
11.13	Example 13 - reading data using the genfromtxt method	176
11.14	Example 14 - Writing a CSV file.....	178
11.15	Example 15 - write large array as text file, element by element, with timing 179	
11.16	Example 16 - write large array as binary file , element by element, with timing 180	
11.17	Example 17 - write large array as binary file , whole array, with timing	181
11.18	Example 18 - listing subdirectories.....	182
11.19	Example 19 - listing all Python files	182
11.20	Background i/o technical information.....	183
11.21	Text I/O.....	183
11.22	Binary I/O.....	183
11.23	Raw I/O.....	183
11.24	Performance	183
11.24.1	Binary I/O	183
11.24.2	Text I/O.....	184
11.24.3	Multi-threading.....	184
11.24.4	Reentrancy	184
11.25	Problems.....	184
12	An Introduction to Algorithms and the Big O notation.....	185
12.1	Basic background	185
12.1.1	Brief explanation.....	186
12.2	Quicksort and insertion sort comparison	187
12.3	Basic array and linked list performance	187
12.4	Bibliography	187
12.5	Problems.....	188
13	Sequence types, Iterators and Lists	189
13.1	Iterator types.....	189
13.2	Example 1 - Simple iterator usage	189
13.3	Sequence types	190
13.3.1	Common Sequence Operations.....	190
13.3.2	Immutable Sequence Types	191
13.3.3	Mutable Sequence Types	191
13.4	Lists	192
13.5	Example 2 - list type initialisation and simple for in statement	193
13.6	Example 3 - list type and various sequence methods	193
13.7	Example 4 - list assignment versus copy() method	194
13.8	List comprehensions.....	195
13.9	Example 5 - simple list comprehension.....	195
13.10	Example 6 - more list comprehensions	196
13.11	Example 7 - more list comprehensions	196
13.12	Example 8 - even more list comprehensions	197
13.13	Tuples	197
13.14	Example 9 - simple tuple usage	198
13.15	Ranges	199
13.16	Example 10 - simple range usage.....	199
13.17	Problems.....	199
14	Set types	200
14.1	Set Types	200
14.2	Example 1 - simple set usage	202
14.3	Example 2 - simple dictionary	203
14.4	Problems.....	204

15 Mapping types	205
15.1 Mapping types	205
15.2 Example 1 - simple dict usage	205
15.3 Example 2 - dict view usage	205
15.4 Problems.....	206
16 Operator overloading.....	207
16.1 Introduction	207
16.2 Example 1 - simple operator overloading.....	207
16.3 Problems.....	208
17 Decimals, fractions, random numbers	209
17.1 Introduction	209
17.2 The Decimal module.....	209
17.3 Example 1 - using getcontext()	210
17.4 Function availability	210
17.5 Example 2 - values for the maths constants e and pi.....	212
17.6 Example 3 - summation using float and decimal	213
17.7 The Fraction module.....	214
17.8 Example 4 - simple fraction usage	214
17.9 The Random module	215
17.10 Example 5 - simple random usage.....	216
17.11 Problems.....	217
18 Databases and sqlite	218
18.1 Introduction to database management systems.....	218
18.2 SQL based systems and Python.....	218
18.2.1 Microsoft SQL Server.....	218
18.2.2 DB API 2.0 Drivers.....	218
18.3 SQLite.....	220
18.4 On line documentation at W3 Schools	221
18.5 SQL examples	221
18.5.1 Example 1 - Database creation	222
18.5.2 Example 2 - Table creation.....	222
18.5.3 Example 3 - loading the earthqk table.....	226
18.5.4 Example 4 - loading the regions table.....	227
18.5.5 Example 5 - loading the tsunami table	228
18.5.6 Example 6 - Querying the tables	228
18.6 Using SQLite from the command line	228
18.7 Creating a database of the Met Office data.....	228
18.7.1 Example 7 - creating the database.....	228
18.7.2 Example 8 - creating a table for one of the sites	229
18.7.3 Example 9 - loading data into the table.....	229
18.7.4 Example 10 - simple table query	230
18.7.5 Example 11 - computing averages	230
18.7.6 Example 12 - Finding the wettest month and displaying the year, month and rainfall	231
18.7.7 Example 13 - Finding the wettest months and displaying the year, month and rainfall	232
18.8 Example 14 - doing monthly average calculations using the genfromtxt ex- ample in the IO chapter	232
18.9 Problems.....	234
19 Regular expressions and pattern matching	243
19.1 Metacharacters	244
19.2 Example 1 - UK post codes	244
19.3 Problems.....	245
19.4 Bibliography	245
20 Built in exceptions	247
20.1 Exception hierarchy	247
20.2 Problems.....	249
21 Concurrent execution - threading.....	250

21.1	Thread based parallelism - the threading package	250
21.2	Example 1 - Serial solution	250
21.3	Example 2 - Multi-threaded solution.....	252
21.4	Problems.....	253
22	Concurrent execution - multi processing	255
22.1	Introduction	255
22.2	Process based parallelism - the multiprocessing package	255
22.3	Contexts and start methods¶.....	255
22.4	Example 1 - Simple multi-processing on a 6 core system.....	256
22.5	Example 2 - Simple variant for an 8 core system.....	258
22.6	Differences between the two version	260
22.7	Sample runs	260
22.8	Summary timing table.....	262
22.9	Problems.....	263
23	Modules	264
23.1	Introduction	264
23.2	Introduction to modules	270
23.3	Example 1 - simple module usage	270
23.4	More on Modules.....	271
23.4.1	Note:.....	272
23.5	Executing modules as scripts.....	272
23.6	The Module Search Path.....	272
23.6.1	Note:.....	272
23.7	“Compiled” Python files	273
23.8	Standard Modules.....	273
23.9	The dir() Function	274
23.10	Packages	276
23.11	Importing * From a Package	277
23.12	Intra-package References	278
23.13	Packages in Multiple Directories	279
23.14	Summary	279
23.15	Problems.....	279
24	SciPy and Pandas	280
24.1	Introduction	280
24.2	Documentation.....	280
24.3	Tutorials	280
24.4	Reference material	281
24.5	Pandas.....	281
24.5.1	Example 1 - Basic Pandas syntax.....	283
24.5.2	Example 2 - Calculating overall averages	284
24.5.3	Example 3 - Calculating minimum and maximum values.....	285
24.5.4	Example 4 - Using the groupby method.....	286
24.6	Summary	289
24.7	Problems.....	289
25	Windows programming in Python	290
25.1	Introduction to Windows programming.....	290
25.2	Tkinter.....	290
25.3	Example 1 - simple test program included with Tkinter distribution.....	290
25.4	Example 2 - Hello world version 1	291
25.5	Example 3 - Hello world variant 1	292
25.6	Example 4 - Hello world variant 2	293
25.7	Example 5 - Hello world version 2	293
25.8	Example 6 - Hello world version 3	294
25.9	The remaining examples	296
25.10	Example 7 - simple button example.....	296
25.11	Example 8 - Button and message example	297
25.12	Example 9 - Button, message and entry example	298
25.13	Example 10 - Button, entry and text widget example	300

25.14	Tkinter on line examples and resources	301
25.15	Other options	302
25.15.1	QT Creator	302
25.16	Problems	303
26	Graphics plotting in Python using matplotlib	304
26.1	Graphics plotting with matplotlib	304
26.2	The jupyter qtconsole on Windows	305
26.3	Example 1 - Simple trigonometric plot	307
26.4	Example 2 - Enhanced trigonometric plot	316
26.5	Example 3 - adding a legend, matplotlib defaults	317
26.6	Example 4 - adding a legend with manual positioning	318
26.7	Example 5 - Bar charts	319
26.8	Example 6 - bar chart with standard deviations	321
26.9	Example 7 - bar chart with 4 frequencies	322
26.10	Example 8 - bar chart with 10 frequencies	325
26.11	Example 9 - Mapping with Python 2.x and basemap	328
26.12	Mapping with Python 3 and Cartopy	333
26.12.1	Example 10 - tsunami plot using cartopy	334
26.12.2	Example 11 - shifting the center of the map	341
26.12.3	Example 12 - mapping using UK postcodes	345
26.13	Bibliography	349
26.13.1	Python	349
26.13.2	Cartopy	349
26.13.3	Map data	349
26.13.4	UNEP	349
26.14	Problems	349
27	Python performance versus other programming languages	351
27.1	Introduction	351
27.2	Example 1 - Python solution	351
27.3	Example 2 - Fortran solution	352
27.4	Example 3 - C++ solution	353
27.5	Example 4 - Java solution	355
27.6	Summary	356
27.7	Problems	357
28	Calling the Nag library from Python	358
28.1	Introduction	358
28.2	Example 1 - testing the Nag library calls	359
28.3	Example 2 - testing the Python random number generators	360
28.4	Example 3 - Python native timing	360
28.5	Example 4 - Nag timing	362
28.6	Problems	364
29	Functional programming background	365
29.1	Introduction	365
29.2	Background	365
29.3	History	366
29.4	Concepts	367
29.4.1	First-class and higher-order functions	367
29.4.2	Pure functions	367
29.4.3	Recursion	368
29.4.4	Strict versus non-strict evaluation	368
29.4.5	Type systems	369
29.4.6	Referential Transparency	369
29.4.7	Functional programming in non-functional languages	369
29.5	Comparison to imperative programming	370
29.5.1	Simulating state	370
29.5.2	Efficiency issues	371
29.5.3	Coding styles	371
29.5.3.1	Version 1 – With Generators	371
29.5.3.2	Version 2 – Iterative	372

29.5.3.3	Version 3 – Recursive	372
29.5.3.4	Haskell	372
29.5.3.5	Erlang	372
29.5.3.6	Elixir	373
29.5.3.7	Lisp	373
29.5.3.8	D	373
29.5.3.9	R	374
29.6	Use in industry	374
29.7	In education	375
30	SQL background	376
30.1	SQL background	376
30.1.1	History	377
30.1.2	SQL online documentation	378
30.1.3	Design	378
30.1.4	Syntax	378
30.1.5	Language elements	378
30.1.6	Operators	379
30.1.7	Queries	380
30.1.8	Subqueries	382
30.1.9	Inline View	382
30.1.10	Null or three-valued logic (3VL)	382
30.1.11	Data manipulation	384
30.1.12	Transaction controls	384
30.1.13	Data definition	385
30.1.14	Data types	386
30.1.14.1	Character strings	386
30.1.14.2	Bit strings	386
30.1.14.3	Numbers	386
30.1.14.4	Temporal (date/time)	386
30.1.15	Data control	387
30.1.16	Procedural extensions	387
30.1.17	Interoperability and standardization	388
30.1.18	Alternatives	391
30.1.19	Distributed SQL processing	392
30.1.20	See also	392
30.1.21	Notes	392
30.2	My Bibliography	395
31	Example summary	397
31.1	Introduction	397
31.2	Chapter notes	402

‘The first thing we do, let’s kill all the language lawyers.’

Henry VI, part II

1 Overview

1.1 Aims

The aim of the notes is to provide an introduction to the Python language. The following is taken from the Python wiki.

Python is a great object-oriented, interpreted, and interactive programming language. It is often compared (favourably of course) to Lisp, Tcl, Perl, Ruby, C#, Visual Basic, Visual Fox Pro, Scheme or Java... and it's much more fun.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as. New built-in modules are easily written in C or C++ (or other languages, depending on the chosen). Python is also usable as an extension language for that need easy-to-use scripting or automation interfaces.

The main Python site is.

<https://www.python.org>

The [downloads] tab has details of the versions that are available.

<https://www.python.org/downloads/>

and the [documentation] tab has details of the documentation.

<https://docs.python.org/3>

The [faq] tab

<https://docs.python.org/3/faq/index.html>

has the following entries

- General Python

- Programming

- Design and history

- Library and extension

- Extending/Embedding

- Python on Windows

- Graphical User Interface

- Why is Python installed on my computer

These sites provide a really good starting point, with the benefit that they don't cost anything except your time.

1.2 History

Here is the Wikipedia entry.

Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the Amoeba

operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, (benevolent dictator for life - BDFL).

Here is some background taken from the Python site.

Here's a very brief summary of what started it all, written by Guido van Rossum:

I had extensive experience with implementing an interpreted language in the ABC group at CWI, and from working with this group I had learned a lot about language design. This is the origin of many Python features, including the use of indentation for statement grouping and the inclusion of very-high-level data types (although the details are all different in Python).

I had a number of gripes about the ABC language, but also liked many of its features. It was impossible to extend the ABC language (or its implementation) to remedy my complaints – in fact its lack of extensibility was one of its biggest problems. I had some experience with using Modula-2+ and talked with the designers of Modula-3 and read the Modula-3 report. Modula-3 is the origin of the syntax and semantics used for exceptions, and some other Python features.

I was working in the Amoeba distributed operating system group at CWI. We needed a better way to do system administration than by writing either C programs or Bourne shell scripts, since Amoeba had its own system call interface which wasn't easily accessible from the Bourne shell. My experience with error handling in Amoeba made me acutely aware of the importance of exceptions as a programming language feature.

It occurred to me that a scripting language with a syntax like ABC but with access to the Amoeba system calls would fill the need. I realized that it would be foolish to write an Amoeba-specific language, so I decided that I needed a language that was generally extensible.

During the 1989 Christmas holidays, I had a lot of time on my hand, so I decided to give it a try. During the next year, while still mostly working on it in my own time, Python was used in the Amoeba project with increasing success, and the feedback from colleagues made me add many early improvements.

In February 1991, after just over a year of development, I decided to post to USENET. The rest is in the Misc/HISTORY file

Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed.

Python 3.0 (also called Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008[31] after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6 and 2.7.

1.3 Use

This is taken from the wikipedia entry. Numbers in [] brackets refer to wikipedia references.

Main article: List of Python software

Since 2003, Python has consistently ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of September 2015, it is in the fifth position.[91] It was ranked as Programming Language of the Year for the year 2007 and 2010.[19] It is the third most popu-

lar language whose grammatical syntax is not predominantly based on C, e.g. C++, Objective-C (note, C# and Java only have partial syntactic similarity to C, such as the use of curly braces, and are closer in similarity to each other than C).

An empirical study found scripting languages (such as Python) more productive than conventional languages (such as C and Java) for a programming problem involving string manipulation and search in a dictionary. Memory consumption was often "better than Java and not much worse than C or C++".[92]

Large organizations that make use of Python include Google,[93] Yahoo!,[94] CERN,[95] NASA,[96] and some smaller ones like ILM,[97] and ITA.[98]

Python can serve as a scripting language for web applications, e.g., via `mod_wsgi` for the Apache web server.[99] With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web application frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjamas and IronPython can be used to develop the client-side of Ajax-based applications. SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.

Libraries like NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing,[100][101] with specialized libraries such as BioPython and Astropy providing domain-specific functionality. Sage is a mathematical software with a "notebook" programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

Python has been successfully embedded in a number of software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP,[102] Inkscape, Scribus and Paint Shop Pro,[103] and musical notation program or scorewriter capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS.[104] It has also been used in several video games,[105][106] and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.[107]

Python has also been used in artificial intelligence tasks.[108][109][110][111] As a scripting language with module architecture, simple syntax and rich text processing tools, Python is often used for natural language processing tasks.[112]

Many operating systems include Python as a standard component; the language ships with most Linux distributions, AmigaOS 4, FreeBSD, NetBSD, OpenBSD and OS X, and can be used from the terminal. A number of Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system, Portage.

Python has also seen extensive use in the information security industry, including in exploit development.[113][114]

Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python.[115]

The Raspberry Pi single-board computer project has adopted Python as its principal user programming language.

LibreOffice includes Python and intends to replace Java with Python. Python Scripting Provider is a core feature[116] since Version 4.0 from 7 February 2013.

We use Python 3 in these notes.

1.4 Assumptions

It is assumed that the reader is familiar with using a computer system, in particular using an editor and working with files.

1.5 Web resources

A copy of these notes can be found at:

<http://www.rhymneyconsulting.co.uk/python/>

The main Python site is

<https://www.python.org/>

which is where you should start.

The Anaconda version of Python is available from a number of sites including:

<https://www.anaconda.com/>

and

<https://repo.continuum.io/>

1.6 Downloading and installing the software

I used openSuSe Linux and Windows implementations of Python writing these notes. There are a number of options available to do Python programming on Windows and Linux, and these are given below

- download and install anaconda from the anaconda site for Windows;

- download and install anaconda from the anaconda site for Linux;

- download and install anaconda from the continuum.io site for Windows;

- download and install anaconda from the continuum.io site for Linux;

- download and install cygwin for Windows, which has Python packages available;

- use the native Python installation that comes with your Linux distribution. I use openSuSe Linux and this can be installed with Python or added using the Yast systems tool;

- download and install Python for Windows from the Python.org site;

- download and build from source for Linux from the Python.org site;

- install the Windows subsystem for Linux on a Windows platform;

download and install Microsofts Hyper V manager and install a linux distribution;

There is a coverage of each option in the sections that follow.

1.7 Windows

We will look at

anaconda download and install;

cygwin download and install;

python download and install;

using Microsoft Visual Studio as a development platform;

using the Windows Subsystem for Linux for development;

installing Hyper V manager and a linux distribution for Python development;

We will also look at the cartopy and basemap modules for mapping.

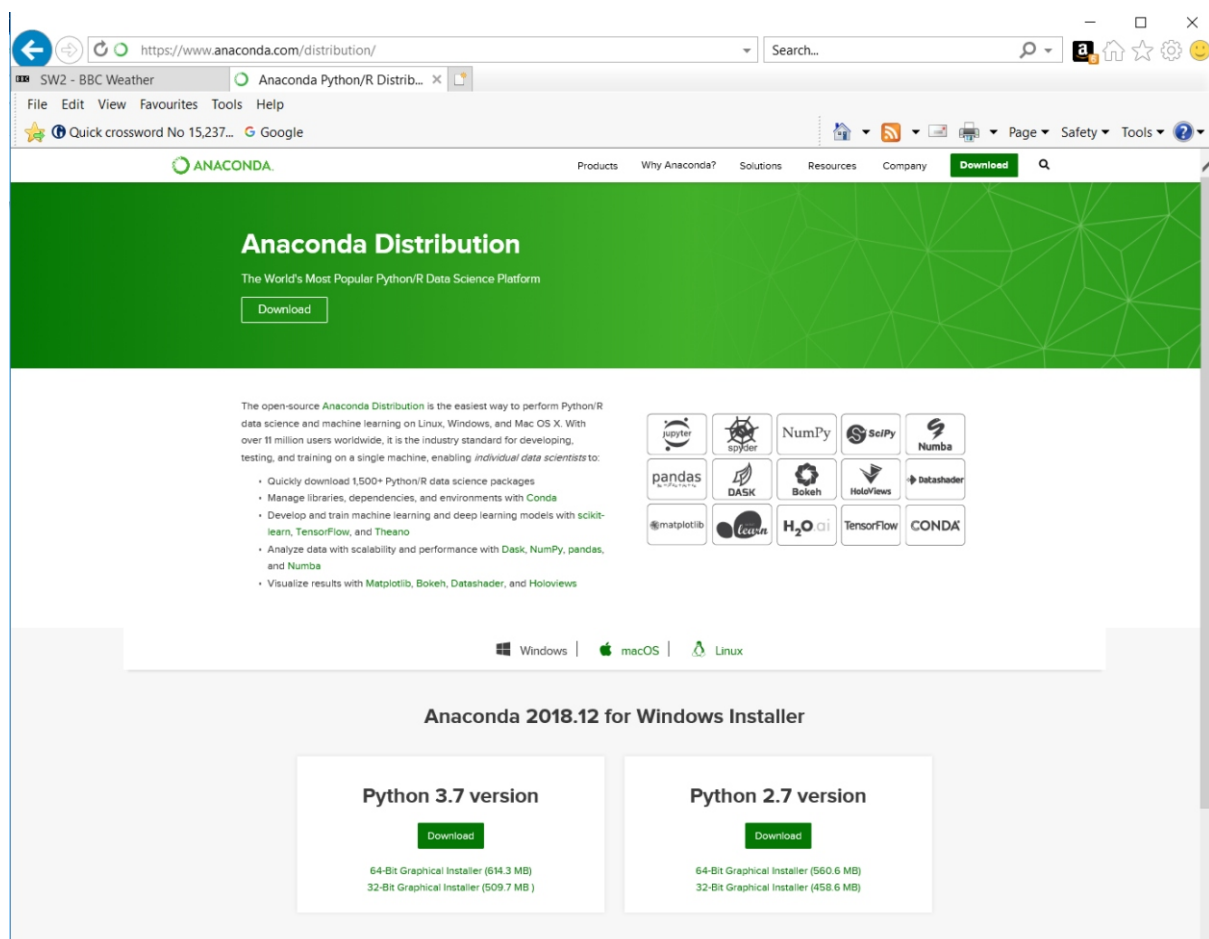
1.7.1 Windows and anaconda

If you visit

<https://www.anaconda.com/>

you will see a download [tab].

Here is a screen shot of the download page.



A Windows Anaconda installation is done by download the installer, and running the .exe, following the instructions on the screen.

One of the aims of the project is to simplify package management and deployment associated with Python. Conda is its package management system.

Visit

<https://docs.anaconda.com/anaconda>

for documentation.

Here is an extract from that page.

Anaconda® is a package manager, an environment manager, a Python/R data science distribution, and a collection of over 1,500+ open source packages. Anaconda is free and easy to install, and it offers free community support.

Get the Anaconda Cheat Sheet and then download Anaconda.

Want to install conda and use conda to install just the packages you need? Get Miniconda.

Anaconda Navigator or conda?

After you install Anaconda or Miniconda, if you prefer a desktop graphical user interface (GUI) then use Navigator. If you prefer to use Anaconda prompt (or Terminal on Linux or macOS), then use that and conda. You can also switch between them.

You can install, remove or update any Anaconda package with a few clicks in Navigator, or with a single conda command in Anaconda Prompt (Terminal on Linux or macOS).

To try Navigator, after installing Anaconda, click the Navigator icon on your operating system's program menu, or in Anaconda prompt (or Terminal on Linux or macOS), run the command `anaconda-navigator`.

To try conda, after installing Anaconda or Miniconda, take the 30-minute conda test drive and download a conda cheat sheet.

Packages available in Anaconda

Over 200 packages are automatically installed with Anaconda.

Over 2000 additional open source packages (including R) can be individually installed from the Anaconda repository with the conda install command.

Thousands of other packages are available from Anaconda Cloud.

You can download other packages using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in some cases they can work together. However, the preference should be to install the conda package if it is available.

You can also make your own custom packages using the conda build command, and you can share them with others

by uploading them to Anaconda Cloud, PyPi or other repositories.

Previous versions

Previous versions of Anaconda are available in the archive. For a list of packages included in each previous version, see Old package lists.

Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it does not matter which one you download, because you can create new environments that include any version of Python packaged with conda. See Managing Python with conda.

The above information was taken in March 2019.

Here is the package list for Anaconda 2.4.1. Notes for column 2 and 4 in the table are given after the table. As can be seen there are a wide variety of packages, covering a lot of areas.

Name	V	Summary / License	I
abstract-rendering (Linux) (Mac)	0.5.1	Rendering as binning process / 3-clause BSD	Y
alabaster	0.7.6	configurable sidebar-enabled Sphinx theme / BSD	Y
anaconda-build	0.13.2	Anaconda build client library / proprietary - Continuum Analytics, Inc.	Y
anaconda-client	1.2.1	anaconda.org command line client library / BSD	Y
ansi2html	1.1.0	Convert text with ANSI color codes to HTML. / GPLv3+	Y
appnope (Mac)	0.1.0	Disable App Nap on OS X 10.9 / BSD	Y
appscript (Mac)	1.0.1	Control AppleScriptable applications from Python / Public-Domain	Y
argcomplete	1.0.0	Bash tab completion for argparse. Tab complete all the things! / Apache	Y
astroid	1.3.4	abstract syntax tree for Python with inference support. / LGPL	Y
astropy	1.0.6	Community-developed python astronomy tools / BSD	Y
azure	1.0.2	easy to access the Microsoft Azure components / Apache License 2.0	Y
babel	2.1.1	Internationalization utilities / BSD	Y
basemap (Linux) (Mac)	1.0.7	Plot data on map projections with matplotlib / PSF	Y
bcolz	0.12.0	columnar and compressed data containers. / BSD	Y
beautifulsoup4	4.4.1	screen-scraping library / MIT	Y
biopython	1.66	Freely available tools for computational molecular biology / BSD-like	Y
bitarray	0.8.1	efficient arrays of booleans -- C extension / PSF	Y

Name	V	Summary / License	I
blaze	0.8.3	NumPy and Pandas interface to Big Data / BSD	Y
blaze-core	0.8.3	Blaze is the next generation of NumPy / BSD	Y
blist	1.3.6	a replacement type with better performance for modifying large lists / BSD	Y
blockspring	0.1.13	Blockspring api wrapper for Python / MIT	Y
blosc (Windows)	1.7.0	Blosc data compressor / MIT	Y
bokeh	0.10.0	statistical and novel interactive HTML plots for Python / BSD	Y
boost	1.57.0	Boost provides free peer-reviewed portable C++ source libraries. / Boost license	Y
boto	2.38.0	Amazon Web Services Library / MIT	Y
bottleneck	1.0.0	Fast NumPy array functions written in Cython. / Simplified BSD	Y
bsdiff4	1.1.4	binary diff and patch using the BSDIFF4-format / BSD	Y
btrees	4.1.4	scalable persistent object containers / ZPL 2.1	Y
bz2file	0.98	read and write bzip2-compressed files / Apache License, Version 2.0	Y
bzip2 (Windows)	1.0.6	high-quality data compressor / BSD	Y
cachecontrol	0.11.5	httplib2 caching for requests / Apache	Y
ctypes	1.2.1	Foreign Function Interface for Python calling C code / MIT	Y
chameleon	2.22	Fast HTML/XML Template Compiler. / BSD-like	Y
cherrypy	3.8.0	object-oriented HTTP framework / BSD	Y
chest	0.2.3	a dictionary that spills to disk / BSD	Y
chrpath (Linux)	0.13	Tool to edit the rpath in ELF binaries / GPLv2	Y
click	4.1	A simple wrapper around optparse for powerful command line utilities. / BSD	Y
cligj	0.2.0	Click params for GeoJSON CLI. / MIT	Y
cloudpickle	0.1.1	Extended pickling support for Python objects / as-is	Y
clyent	1.2.0	Command line client Library for windows and posix / BSD	Y
cmake (Linux) (Mac)	3.3.1	CMake is an extensible, open-source system that manages the build process / 3-clause BSD	Y
colorama	0.3.3	Cross-platform colored terminal text. / BSD	Y
comtypes (Windows)	1.1.2	pure Python COM package / MIT	Y

Name	V	Summary / License	I
conda	3.18.8	cross-platform, Python-agnostic binary package manager / BSD	Y
conda-api	1.1.0	A light weight conda interface library / BSD	Y
conda-build	1.18.2	Commands and tools for building conda packages / BSD	Y
conda-env	2.4.5	provides a unified interface to dealing with Conda environments / BSD	Y
configobj	5.0.6	Config file reading, writing and validation / BSD	Y
contextlib2	0.4.0	backports and enhancements for the contextlib module / PSF	Y
coverage	4	Code coverage measurement for Python / BSD	Y
cryptacular	1.4.1	password hashing framework with bcrypt and pbkdf2 / MIT	Y
cryptography	1.0.2	provides cryptographic recipes and primitives to Python developers / Apache	Y
cssselect (Linux) (Mac)	0.9.1	cssselect parses CSS3 Selectors and translates them to XPath 1.0 / BSD	Y
csvkit	0.9.1	utilities for working with CSV, the king of tabular file formats / MIT	Y
cubes	1.0.1	a light-weight Python OLAP framework for data warehouses / MIT	Y
curl	7.45.0	tool and library for transferring data with URL syntax / MIT/X derivate	Y
cvxopt (Linux) (Mac)	1.1.7	CVXOPT is a free software package for convex optimization / GPL	Y
cycler	0.9.0	Composable style cycles / BSD	Y
cymem (Linux) (Mac)	1.3	Manage calls to calloc/free through Cython / MIT	Y
cython	0.23.4	The Cython compiler for writing C extensions for the Python language / Apache	Y
cytoolz	0.7.4	Cython implementation of Toolz, high performance functional utilities / BSD	Y
dask	0.7.3	Task scheduling and blocked algorithms for parallel processing / BSD	Y
datashape	0.4.7	A data description language / BSD	Y
datrie	0.7	Super-fast, efficiently stored Trie for Python / LGPLv2	Y
dbf	0.96.0 03	reading/writing dBase, FoxPro, and Visual FoxPro .dbf files / BSD	Y

Name	V	Summary / License	I
decorator	4.0.4	Better living through Python with decorators / BSD	Y
dill	0.2.4	Serialize all of python (almost) / 3-clause BSD	Y
django	1.8.4	Web framework that encourages rapid development / BSD	Y
docopt	0.6.2	Pythonic argument parser, that will make you smile / MIT	Y
docutils	0.12	Utilities for general- and special-purpose documentation / Public-Domain, PSF, 2-clause BSD, GPLv3	Y
drmaa (Linux) (Mac)	0.7.6	python DRMAA library / BSD	Y
dynd-python	0.7.0	Python exposure of DyND / BSD	Y
ecdsa	0.13	ECDSA cryptographic signature library (pure python) / MIT	Y
ephem (Linux) (Mac)	3.7.6. 0	Compute positions of the planets and stars / LGPL	Y
execnet	1.3.0	rapid multi-Python deployment / MIT	Y
fastcache	1.0.2	C implementation of Python 3 functools.lru_cache / MIT	Y
feedparser	5.2.1	Universal feed parser / MIT	Y
flake8	2.3.0	the modular source code checker: pep8, pyflakes and co / MIT	Y
flask	0.10.1	A microframework based on Werkzeug, Jinja2 and good intentions / BSD	Y
flask-login	0.2.11	User session management for Flask / MIT	Y
flask-wtf	0.11	Simple integration of Flask and WTForms / BSD	Y
fontconfig (Linux)	2.11.1	Fontconfig is a library for configuring and customizing font access / BSD	Y
freeglut (Linux)	2.8.1	a completely OpenSourced alternative to the OpenGL Utility Toolkit library. / MIT	Y
freetype	2.5.5	A Free, High-Quality, and Portable Font Engine / FreeType License	Y
funcsigs	0.4	Python function signatures from PEP362 for Python 2.6, 2.7 and 3.2+ / Apache	Y
future	0.15.2	Clean single-source support for Python 3 and 2 / MIT	Y
futures	3.0.3	Backport of the concurrent.futures package from Python 3.2 / BSD	Y

Name	V	Summary / License	I
gensim	0.12.2	Python framework for fast Vector Space Modelling / LGPL	Y
geos	3.4.2	GEOS (Geometry Engine - Open Source) is a C++ port of the Java Topology Suite (JTS). / LGPL	Y
glueviz	0.6.0	Multidimensional data visualization across files / MIT	Y
graphviz	2.38.0	Open Source graph visualization software. / EPL	Y
greenlet	0.4.9	lightweight in-process concurrent programming / MIT	Y
gridmap (Linux) (Mac)	0.13.0	map Python functions onto a cluster using a grid engine / GPL3	Y
gunicorn (Linux) (Mac)	19.1.0	WSGI HTTP Server for UNIX / MIT	Y
h5py	2.5.0	Read and write HDF5 files from Python. / 3-clause BSD	Y
hdf4	4.2.11	/ BSD-style	Y
hdf5	1.8.15.1	HDF5 is a data model, library, and file format for storing and managing data / BSD-like	Y
heapdict	1.0.0	a heap with decrease-key and increase-key operations / BSD	Y
holoviews	1.3.2	composable, declarative data structures for building complex visualizations / BSD	Y
html5lib	0.999	HTML parser based on the WHATWG HTML specification / MIT	Y
icu (Linux) (Mac)	54.1	/ MIT	Y
idna	2	Internationalized Domain Names in Applications / BSD	Y
iopro	1.7.2	python interface for databases, NoSQL stores, Amazon S3, and large data files / proprietary - Continuum Analytics, Inc.	Y
ipykernel	4.1.1	IPython Kernel for Jupyter / BSD	Y
ipyparallel	4.1.0	Jupyter Qt Console / BSD	Y
ipython	4.0.1	Productive Interactive Computing / BSD	Y
ipython_genutils	0.1.0	vestigial utilities from IPython / BSD	Y
ipywidgets	4.1.0	IPython Static Widgets / BSD	Y
itsdangerous	0.24	Various helpers to pass trusted data to untrusted environments and back. / BSD	Y
jbig (Linux) (Mac)	2.1	implementation of the JBIG1 data compression standard / GPL2	Y

Name	V	Summary / License	I
jdcal	1	Julian dates from proleptic Gregorian and Julian calendars. / BSD	Y
jedi	0.9.0	/ MIT	Y
jinja2	2.8	An easy to use stand-alone template engine written in pure python. / BSD	Y
joblib	0.8.4	using Python functions as pipeline jobs / BSD	Y
jpeg	8d	read/write jpeg COM, EXIF, IPTC metadata / Custom free software license	Y
jsonschema	2.4.0	An implementation of JSON Schema validation for Python / MIT	Y
jupyter	1.0.0	Jupyter metapackage / BSD	Y
jupyter_client	4.1.1	Jupyter protocol implementation and client libraries / BSD	Y
jupyter_console	4.0.3	Jupyter terminal console / BSD	Y
jupyter_core	4.0.6	base package on which Jupyter projects rely / BSD	Y
kealib	1.4.5	The KEA format provides an implementation of the GDAL specification within the the HDF5 file format. / MIT	Y
lancet	0.9.0	launch jobs, organize the output, and dissect the results / BSD	Y
launcher (Mac) (Windows)	1.0.0	Anaconda's application launcher / proprietary - Continuum Analytics, Inc.	Y
ldap3	0.9.8.4	A strictly RFC 4511 conforming LDAP V3 pure Python client. / LGPLv3	Y
libdynd	0.7.0	C++ dynamic ndarray library / BSD	Y
libffi (Linux)	3.0.13	A portable foreign-function interface library / MIT	Y
libgfortran (Linux)	1	GNU Fortran runtime library / GPL3	Y
libnetcdf	4.3.3.1	libraries and data formats that support array-oriented scientific data / MIT	Y
libpng	1.6.17	libpng is the official PNG reference library / libpng license	Y
libsodium (Linux) (Windows)	1.0.3	a modern software library for encryption, signatures, password hashing, etc. / MIT	Y
libtiff	4.0.6	tiff image library / BSD-like	Y
libxml2 (Linux) (Mac)	2.9.2	The XML C parser and toolkit of Gnome / MIT	Y

Name	V	Summary / License	I
libxslt (Linux) (Mac)	1.1.28	Libxslt is the XSLT C library developed for the GNOME project / MIT	Y
lighttpd (Linux) (Mac)	1.4.36	light web server (httpd) / BSD	Y
line_profiler	1	Line-by-line profiler / BSD	Y
llvmlite	0.8.0	lightweight wrapper around basic LLVM functionality / BSD	Y
locket	0.2.0	File based locks / BSD	Y
lockfile (Linux) (Mac)	0.10.2	Platform-independent file locking module / MIT	Y
logilab-common	1.0.2	collection of low-level Python packages and modules used by Logilab projects / LGPL	Y
lxml	3.4.4	XML processing library combining libxml2/libxslt with the ElementTree API / BSD	Y
mako	1.0.3	templating language / MIT	Y
markdown	2.6.2	Python implementation of Markdown / BSD	Y
markdown2	2.3.0	Python implementation of Markdown / BSD	Y
markupsafe	0.23	Implements a XML/HTML/XHTML Markup safe string for Python / BSD	Y
mathjax	2.2	A JavaScript display engine for mathematics that works in all browsers / Apache	Y
matplotlib	1.5.0	Python plotting package / PSF-based	Y
mccabe	0.3	McCabe checker, plugin for flake8 / MIT	Y
mdp	3.3	a Python data processing framework. / BSD	Y
meld3	1.0.2	an HTML/XML templating engine / BSD-derived	Y
menuinst (Windows)	1.3.1	cross platform install of menu items / BSD	Y
mingw (Windows)	4.7	GCC-like development environment for native Windows / Public-Domain	Y
mistune	0.7.1	The fastest markdown parser in pure Python with renderer feature / BSD	Y
mock	1.3.0	A Python mocking and patching library for testing / BSD	Y
mpi4py (Linux)	1.3.1	MPI for Python / BSD	Y
mpich2 (Linux)	1.4.1p1	a high performance widely portable implementation of the MPI standard / mpich license	Y

Name	V	Summary / License	I
mpmath	0.19	Python library for arbitrary-precision floating-point arithmetic / BSD	Y
msgpack-python	0.4.6	MessagePack is an efficient binary serialization format / Apache	Y
msvc_runtime (Windows)	1.0.1	Bundles of the MSVC runtime for your Python / Proprietary	Y
multimethods	1.0.0	A simple python multidispatch. / MIT	Y
multipliedispatch	0.4.8	Multiple dispatch / BSD	Y
murmurhash (Linux) (Mac)	0.24	Cython .pxd files for some of the MurmurHash 2 and 3 hash functions / Public-Domain	Y
mysql-connector-python	2.0.3	MySQL driver written in Python / GPL2	Y
nano (Linux) (Mac)	2.4.1	An enhanced clone of the Pico text editor / GPL2	Y
natsort	4.0.3	Sort lists naturally / MIT	Y
nbconvert	4.0.0	converts notebooks to various other formats via Jinja templates / BSD	Y
nbformat	4.0.1	the base implementation of the Jupyter Notebook format / BSD	Y
ncurses (Linux) (Mac)	5.9	free software emulation of curses in System V Release 4.0, and more / ncurses license	Y
netcdf4	1.1.9	python/numpy interface to netCDF library / MIT	Y
networkx	1.1	Python package for creating and manipulating graphs and networks / BSD	Y
nltk	3.1	Natural Language Toolkit / Apache	Y
node-webkit (Mac) (Windows)	0.10.1	calls Node.js modules from DOM and enables a new way of writing applications / MIT	Y
nose	1.3.7	nose extends unittest to make testing easier / LGPL	Y
notebook	4.0.6	a web-based notebook environment for interactive computing / BSD	Y
numba	0.22.1	compiling Python code using LLVM / BSD	Y
numexpr	2.4.4	Fast numerical expression evaluator for NumPy / MIT	Y
numpy	1.9.3	array processing for numbers, strings, records, and objects. / BSD	Y
numpydoc	0.5	Sphinx extension to support docstrings in Numpy format / BSD	Y
odo	0.3.4	Data Migration for Blaze Project / BSD	Y

Name	V	Summary / License	I
openblas (Linux)	0.2.14	optimized BLAS library based on GotoBLAS2 / BSD	Y
openpyxl	2.2.6	A Python library to read/write Excel 2007 xlsx/xlsm files / MIT	Y
openssl	1.0.2d	OpenSSL is an open-source implementation of the SSL and TLS protocols / Apache-style	Y
pandas	0.17.1	Powerful data structures for data analysis, time series, and statistics / BSD	Y
pandas-datareader	0.2.0	Data readers extracted from the pandas codebase, should be compatible with recent pandas versions / BSD License	Y
param	1.3.2	declarative Python programming using Parameters / BSD	Y
paramiko	1.15.3	SSH2 protocol library / LGPL	Y
partd	0.3.2	Appendable key-value byte store / BSD	Y
passlib	1.6.5	comprehensive password hashing framework supporting over 30 schemes / BSD	Y
pastedeploy	1.5.2	Load, configure, and compose WSGI applications and servers / MIT	Y
patch (Windows)	2.5.9	Native Win32 versions of common unix tools / GPL	Y
patchelf (Linux)	0.6	a small utility to modify the dynamic linker and RPATH of ELF executables / GPL3	Y
path.py	8.1.2	module wrapper for os.path / MIT	Y
patsy	0.4.0	a library for describing statistical models and building design matrices. / BSD	Y
pbkdf2	1.3	the password-based key derivation function, PBKDF2 / MIT	Y
pbr	1.3.0	Python Build Reasonableness / Apache	Y
pep8	1.6.2	Python style guide checker / MIT	Y
persistent	4.1.1	translucent persistent objects / ZPL 2.1	Y
pexpect (Linux) (Mac)	3.3	Pexpect allows easy control of interactive console appli- cations / ISC	Y
pickleshare	0.5	tiny shelve-like database with concurrency support / MIT	Y
pillow	3.0.0	Python Imaging Library (Fork) / PIL license	Y
pip	7.1.2	PyPA recommended tool for installing Python packages / MIT	Y

Name	V	Summary / License	I
plac	0.9.1	The smartest command line arguments parser in the world / BSD	Y
ply	3.8	Python Lex & Yacc / BSD	Y
preshed (Linux) (Mac)	0.44	Cython hash table that trusts the keys are pre-hashed / MIT	Y
proj4	4.9.1	PROJ.4 Cartographic Projections library / MIT	Y
psutil	3.3.0	cross-platform process and system utilities module for Python / BSD	Y
psycopg2 (Linux) (Mac)	2.6.1	Python-PostgreSQL Database Adapter / LGPL, BSD-like, ZPL	Y
ptyprocess (Linux) (Mac)	0.5	Run a subprocess in a pseudo terminal / ISC	Y
py	1.4.30	library with cross-python path, ini-parsing, io, code, log facilities / MIT	Y
pyasn1	0.1.9	Offline IP address to Autonomous System Number lookup module / BSD	Y
pycosat	0.6.1	bindings to picosat (a SAT solver) / MIT	Y
pycparser	2.14	C parser in Python / BSD	Y
pycrypto	2.6.1	Cryptographic modules for Python. / Public-Domain	Y
pycurl	7.19.5 .1	PycURL -- cURL library module for Python / LGPL, MIT	Y
pyflakes	1.0.0	passive checker of Python programs / MIT	Y
pygments	2.0.2	Pygments is a syntax highlighting package written in Python / BSD	Y
pylint	1.4.2	python code static checker / GPL	Y
pymc	2.3.6	Markov Chain Monte Carlo sampling toolkit / Academic Free License	Y
pymongo	3.0.3	Python driver for MongoDB / Apache	Y
pymysql	0.6.7	Pure-Python MySQL Driver / MIT	Y
pyodbc	3.0.10	DB API Module for ODBC / MIT	Y
pyopengl	3.1.1a 1	Standard OpenGL bindings for Python / BSD	Y
pyopengl -accelerate	3.1.1a 1	Standard OpenGL bindings for Python / BSD	Y
pyopenssl	0.15.1	Python wrapper module around the OpenSSL library / Apache	Y
yparsing	2.0.3	Python parsing module / MIT	Y

Name	V	Summary / License	I
pyproj	1.9.4	Python interface to PROJ4 library for cartographic transformations. / MIT	Y
pyqt	4.11.4	PyQt is a Python binding of the cross-platform GUI toolkit Qt / Commercial, GPLv2, GPLv3	Y
pyramid	1.5.7	The Pyramid Web Framework, a Pylons project / BSD	Y
pyramid_chameleon	0.3	bindings for the Chameleon templating system for Pyramid / BSD	Y
pyramid_debugtoolbar	2.4.1	interactive HTML debugger for Pyramid application development / BSD	Y
pyramid_jinja2	2.5	Jinja2 template bindings for the Pyramid web framework / BSD	Y
pyramid_mako	1.0.2	Mako template bindings for the Pyramid web framework / BSD	Y
pyramid_tm	0.12	allows Pyramid requests to join the active transaction / BSD	Y
pyreadline (Windows)	2.1	A python implementation of GNU readline. / BSD	Y
pyserial	2.7	Python Serial Port Extension / PSF	Y
pysnmp (Linux) (Mac)	4.2.5	A pure-Python SNMPv1/v2c/v3 library / BSD	Y
pystan (Linux) (Mac)	2.8.0.0	PyStan provides an interface to Stan / GPL3	Y
pytables	3.2.2	brings together Python, HDF5 and NumPy to easily handle large amounts of data / BSD	Y
pytest	2.8.1	simple powerful testing with Python / MIT	Y
pytest-cache	1	pytest plugin with mechanisms for caching across test runs / MIT	Y
pytest-pep8	1.0.6	pytest plugin to check PEP8 requirements / MIT	Y
python	3.5.1	general purpose programming language / PSF	Y
python-dateutil	2.4.2	Extensions to the standard Python datetime module / BSD	Y
pytz	2015.7	World timezone definitions, modern and historical / MIT	Y
pywget	2.2	pure python download utility / Public-Domain	Y
pywin32 (Windows)	219	Python extensions for Windows / PSF	Y
pyyaml	3.11	YAML parser and emitter for Python / MIT	Y

Name	V	Summary / License	I
pyzmq	14.7.0	zeromq bindings for Python / LGPL and BSD	Y
qt	4.8.7	Qt is a cross-platform application and UI framework / LGPL	Y
qtconsole	4.1.1	Jupyter Qt Console / BSD	Y
quandl	2.8.9	Package for Quandl API access / MIT	Y
queuelib	1.4.2	Collection of persistent (disk-based) queues / BSD	Y
readline (Linux) (Mac)	6.2	line-editing for programs with a command-line interface / GPL3	Y
redis (Linux) (Mac)	2.6.9	Redis is an open source, BSD licensed, advanced key-value cache and store / 3-clause BSD	Y
redis-py (Linux) (Mac)	2.10.3	Redis Python Client / MIT	Y
reportlab	3.2.0	The ReportLab Toolkit / BSD	Y
repoze.lru	0.6	A tiny LRU cache implementation and decorator / BSD	Y
requests	2.8.1	Python HTTP for Humans / Apache	Y
rope	0.9.4	a python refactoring library / GPL	Y
routes	2.2	Routing Recognition and Generation Tools / MIT	Y
runipy	0.1.3	Run IPython notebooks from the command line / BSD	Y
sas7bdat	2.0.6	sas7bdat file reader for Python / MIT	Y
scikit-bio (Linux) (Mac)	0.4.0	Data structures, algorithms and educational resources for bioinformatics. / BSD	Y
scikit-image	0.11.3	Image processing routines for SciPy / 3-clause BSD	Y
scikit-learn	0.17	A set of python modules for machine learning and data mining / 3-clause BSD	Y
scikit-rf	0.14.1	Object Oriented Microwave Engineering / new BSD	Y
scipy	0.16.0	Scientific Library for Python / BSD	Y
seaborn	0.6.0	statistical data visualization / BSD	Y
semantic_version	2.4.2	A library implementing the 'SemVer' scheme. / BSD	Y
setuptools	18.5	Easily download, build, install, upgrade, and uninstall Python packages / PSF or ZPL	Y
setuptools_scm	1.9.0	the blessed package to manage your versions by scm tags / BSD	Y
sh (Linux) (Mac)	1.11	full-fledged subprocess replacement for Python / MIT	Y
shapely (Linux) (Mac)	1.5.11	Geometric objects, predicates, and operations / BSD	Y

Name	V	Summary / License	I
simplegeneric	0.8.1	lets you define simple single-dispatch generic functions / ZPL 2.1	Y
sip	4.16.9	/ GPL3	Y
six	1.10.0	Python 2 and 3 compatibility utilities / MIT	Y
snowballstemmer	1.2.0	provides 16 stemmer algorithms generated from Snowball algorithms / BSD	Y
sockjs-tornado	1.0.1	SockJS python server implementation on top of Tornado framework / MIT	Y
spacy (Linux) (Mac)	0.99	Industrial-strength NLP / MIT	Y
sphinx	1.3.1	Python documentation generator / BSD	Y
sphinx_rtd_theme	0.1.7	ReadTheDocs.org theme for Sphinx / BSD	Y
spyder	2.3.8	Scientific PYthon Development EnviRonment / MIT	Y
sqlalchemy	1.0.9	Database Abstraction Library / MIT	Y
sqlite (Linux) (Mac)	3.8.4.1	self-contained, zero-configuration, SQL database engine / Public-Domain	Y
sqlparse	0.1.16	A non-validating SQL parser module for Python / BSD	Y
statsmodels	0.6.1	Statistical computations and models for use with SciPy / 3-clause BSD	Y
stripe	1.25.0	Stripe python bindings. / MIT	Y
sympy	0.7.6.1	SymPy is a Python library for symbolic mathematics / 3-clause BSD	Y
terminado (Linux) (Mac)	0.5	Terminals served to term.js using Tornado websockets / BSD	Y
text-unidecode (Linux) (Mac)	1	the most basic Text Unidecode port / Artistic License	Y
theano (Linux)	0.7.0	Optimizing compiler for evaluating mathematical expressions on CPUs and GPUs / BSD	Y
thinc (Linux) (Mac)	4.0.0	Learn sparse linear models / Commercial, GPLv2	Y
tk	8.5.18	dynamic programming language with GUI elements / BSD-like	Y
toolz	0.7.4	List processing tools and functional utilities / BSD	Y
tornado	4.3	a Python web framework and asynchronous networking library / Apache	Y
traitlets	4.0.0	configuration system for Python applications / BSD	Y
transaction	1.4.4	transaction management for Python / ZPL 2.1	Y

Name	V	Summary / License	I
translationstring	1.3	Utility library for i18n relied on by various Repoze and Pyramid packages / BSD	Y
twisted	15.4.0	Twisted is an event-driven networking engine for Python / MIT	Y
ujson	1.33	Ultra fast JSON encoder and decoder for Python / BSD	Y
unicodcsv	0.14.1	The unicodcsv file reads and decodes byte strings for you / BSD	Y
unidecode	0.4.17	ASCII transliterations of Unicode text / GPL2	Y
unixodbc (Linux)	2.3.4	unixODBC is an open source project that implements the ODBC API / LGPLv2	Y
unxutils (Windows)	14.04.03	ports of common GNU utilities to native Win32 / GPL3	Y
util-linux (Linux)	2.21	Util-linux is a suite of essential utilities for any Linux system / GPL2	Y
venusian	1	A library for deferring decorator actions / BSD	Y
virtualenv	13.0.1	Virtual Python Environment builder / MIT	Y
w3lib	1.12.0	Library of web-related functions / BSD	Y
waitress	0.8.9	production-quality WSGI server with very acceptable performance / ZPL 2.1	Y
webob	1.4.1	WSGI request and response object / MIT	Y
webtest	2.0.18	helper to test WSGI applications / MIT	Y
werkzeug	0.11.2	The Swiss Army knife of Python web development / BSD	Y
wheel	0.26.0	built-package format for Python / MIT	Y
whoosh	2.7.0	Fast, pure-Python full text indexing, search, and spell checking library / BSD	Y
workerpool	0.9.4	Module for distributing jobs to a pool of worker threads / MIT	Y
wtforms	2.0.2	A flexible forms validation and rendering library for Python / BSD	Y
xerces-c	3.1.2	Xerces-C++ is a validating XML parser written in a portable subset of C++ / Apache 2.0	Y
xlrd	0.9.4	Library for developers to extract data from Microsoft Excel spreadsheet files / BSD	Y
xlsxwriter	0.7.7	A Python module for creating Excel XLSX files / BSD	Y
xlwings (Mac) (Windows)	0.5.0	Make Excel fly. Interact with Excel from Python and vice versa / 3-clause BSD	Y

Name	V	Summary / License	I
xlwt	1.0.0	writing data and formatting information to Excel files / BSD	Y
xray	0.6.1	N-D labeled arrays and datasets in Python / Apache	Y
xz (Linux) (Mac)	5.0.5	data compression software with high compression ratio / Public-Domain, GPL	Y
yaml (Linux) (Mac)	0.1.6	a human friendly data serialization standard for all programming languages / MIT	Y
yt (Linux) (Mac)	3.2.2	An analysis and visualization toolkit for Astrophysical simulations / BSD	Y
zeromq	4.1.3	a messaging system, or "message-oriented middleware" / LGPL	Y
zlib	1.2.8	massively spiffy yet delicately unobtrusive compression library / zlib	Y
zope.deprecation	4.1.2	Zope Deprecation Infrastructure / ZPL 2.1	Y
zope.interface	4.1.3	Interfaces for Python / ZPL 2.1	Y
zope.sqlalchemy	0.7.6	minimal Zope/SQLAlchemy transaction integration / ZPL 2.1	Y

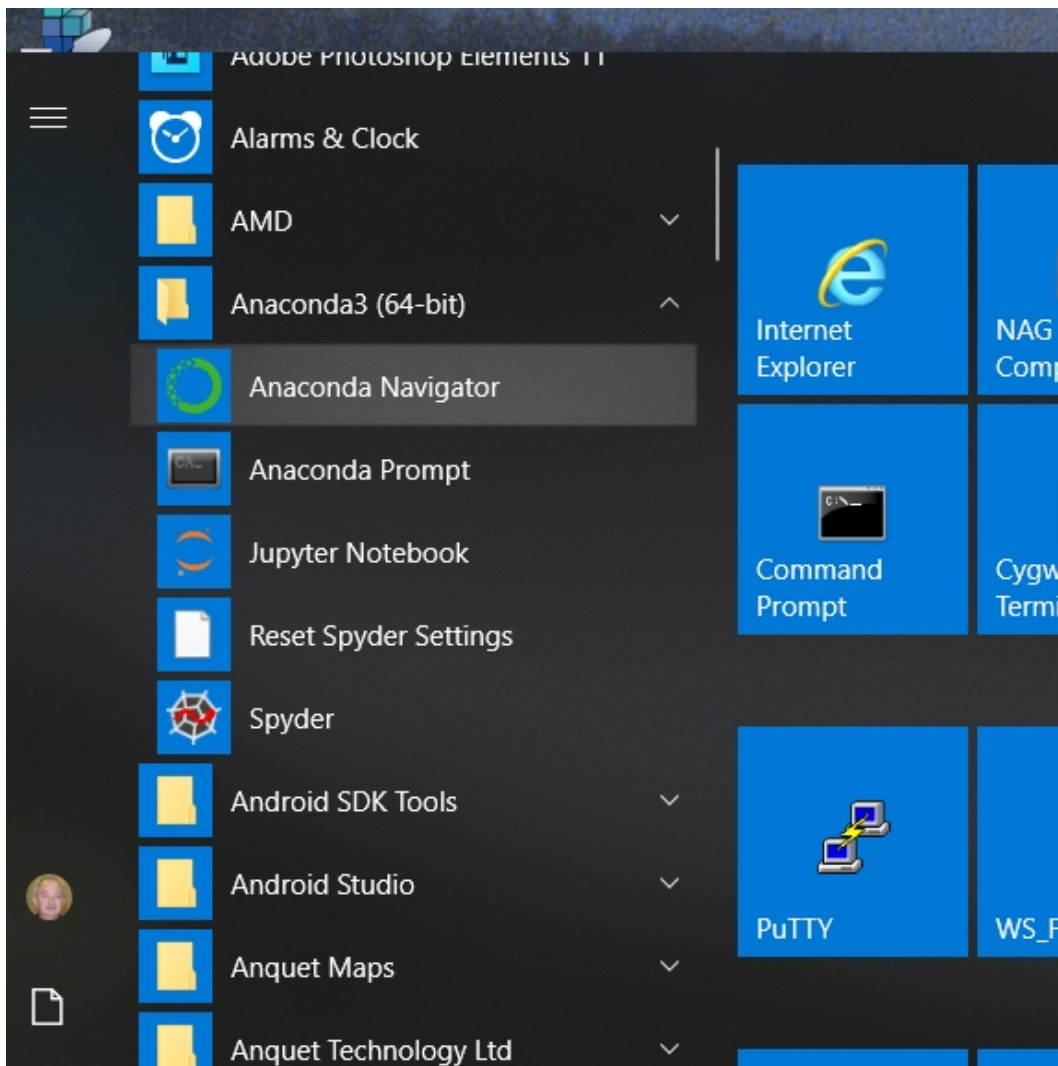
Table notes

Column 2 - V = Version

Column 4 - In installer

1.7.1.1 Accessing Anaconda and Python on Windows

By installing Anaconda on Windows you get access to Anaconda from the start menu. I recommend installing from a command prompt with administrator rights. Here is a screen shot.



As you can see you get the choice of

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, macOS and Linux.

Anaconda prompt

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simu-

lation, statistical modeling, data visualization, machine learning, and much more.

Reset Spyder settings

Spyder

Spyder is the Scientific PYthon Development EnviRonment: a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features and a numerical computing environment thanks to the support of IPython (enhanced interactive Python interpreter) and popular Python libraries such as NumPy (linear algebra), SciPy (signal and image processing) or matplotlib (interactive 2D/3D plotting).

Spyder may also be used as a library providing powerful console-related widgets for your PyQt-based applications – for example, it may be used to integrate a debugging console directly in the layout of your graphical user interface.

The second option allows simple compile and run from the command line.

To run the graphics examples you will need to install matplotlib and to run the mapping examples you will need to install cartopy.

From a python administrator command prompt run the following commands:

```
conda install matplotlib
```

followed by

```
conda install -c conda-forge cartopy
```

and this will enable you to run the graphics examples.

Note that the process based parallel examples won't run correctly with this version of Python under Windows. You need to use the cygwin version.

Here are some useful commands.

```
conda update conda - update the package manager
```

```
conda list conda - display version information
```

```
conda update anaconda - update anaconda
```

```
conda update --all - update all components
```

```
conda info - basic information
```

You will need to run the update commands as administrator.

Here is the output from running the conda info command on one of my systems.

```
active environment : base
active env location : C:\ProgramData\Anaconda3
shell level : 1
user config file : C:\Users\ian\.condarc
populated config files : C:\Users\ian\.condarc
conda version : 4.4.10
conda-build version : 3.4.1
python version : 3.6.4.final.0
```

```

base environment : C:\ProgramData\Anaconda3 (read
only)
channel URLs : https://repo.continuum.io/pkgs/main/win-64
               https://repo.continuum.io/pkgs/main/noarch
               https://repo.continuum.io/pkgs/free/win-64
               https://repo.continuum.io/pkgs/free/noarch
               https://repo.continuum.io/pkgs/r/win-64
               https://repo.continuum.io/pkgs/r/noarch
               https://repo.continuum.io/pkgs/pro/win-64
               https://repo.continuum.io/pkgs/pro/noarch
               https://repo.continuum.io/pkgs/msys2/win-64
               https://repo.continuum.io/pkgs/msys2/noarch
package cache : C:\ProgramData\Anaconda3\pkgs
                C:\Users\ian\AppData\Local\conda\conda\pkgs
envs directories : C:\Users\ian\AppData\Local\conda\conda\envs
                  C:\ProgramData\Anaconda3\envs
                  C:\Users\ian\.conda\envs
platform : win-64
user-agent : conda/4.4.10 requests/2.18.4
CPython/3.6.4 Windows/10 Windows/10.0.17763
administrator : False
netrc file : None
offline mode : False

```

Try this out on your system.

1.7.2 Windows - cygwin python version

Visit

<https://www.cygwin.com/>

The parallel programming examples under Windows require the installation of the Cygwin version of Python.

Here is the cygwin home page.

The screenshot shows the Cygwin website homepage. The browser address bar displays <https://cygwin.com/>. The page layout includes a left-hand navigation menu with categories like 'Cygwin', 'Cygwin/X', 'Community', 'Documentation', 'Contributing', and 'Related Sites'. The main content area features the 'Cygwin' logo and the tagline 'Get that [Linux](#) feeling - on Windows'. Below the logo is the heading 'This is the home of the Cygwin project' followed by a 'What...' section. This section is divided into two columns: '...is it?' and '...isn't it?'. The '...is it?' column lists features such as a large collection of GNU and Open Source tools and a DLL (cygwin1.dll) providing substantial POSIX API functionality. The '...isn't it?' column lists limitations, including the need to rebuild applications from source and the requirement to build applications from source to take advantage of Cygwin's functionality. A section titled 'Current Cygwin DLL version' states that the most recent version is 2.4.0 and provides instructions for installation using `setup-x86.exe` (32-bit) or `setup-x86_64.exe` (64-bit). A 'Commercial Support for Cygwin' section is also visible at the bottom of the page.

I use the 64 bit version. The setup program can be used to do a new install or update an existing version. When you run the setup program you get the following list of packages to install.

- accessibility
- admin
- archive
- audio
- base
- database
- debug
- devel
- doc
- editors
- games
- GNOME

- graphics
- interpreters
- KDE
- libs
- lua
- LXDE
- Mail
- mate
- math
- misc
- net
- OCaml
- Office
- perl
- PHP
- publishing
- python
- Ruby
- Scheme
- Science
- Security
- Shells
- Sugar
- System
- tcl
- text
- utils
- video
- web
- X11
- Xfce

The following list has the Python packages first and the rest indented.

- accessibility
- admin
- archive
- audio
- base
- database
- debug

```
devel
doc
  editors
  games
GNOME
graphics
interpreters
KDE
libs
  lua
  LXDE
  Mail
mate
math
  misc
net
  OCaml
  Office
  perl
  PHP
  publishing
python
  Ruby
  Scheme
  Science
  Security
  Shells
  Sugar
  System
  tcl
text
utils
  video
  web
  X11
  Xfce
```

Make sure you install all of the Python packages. If you already have cygwin installed a `cygcheck -c`

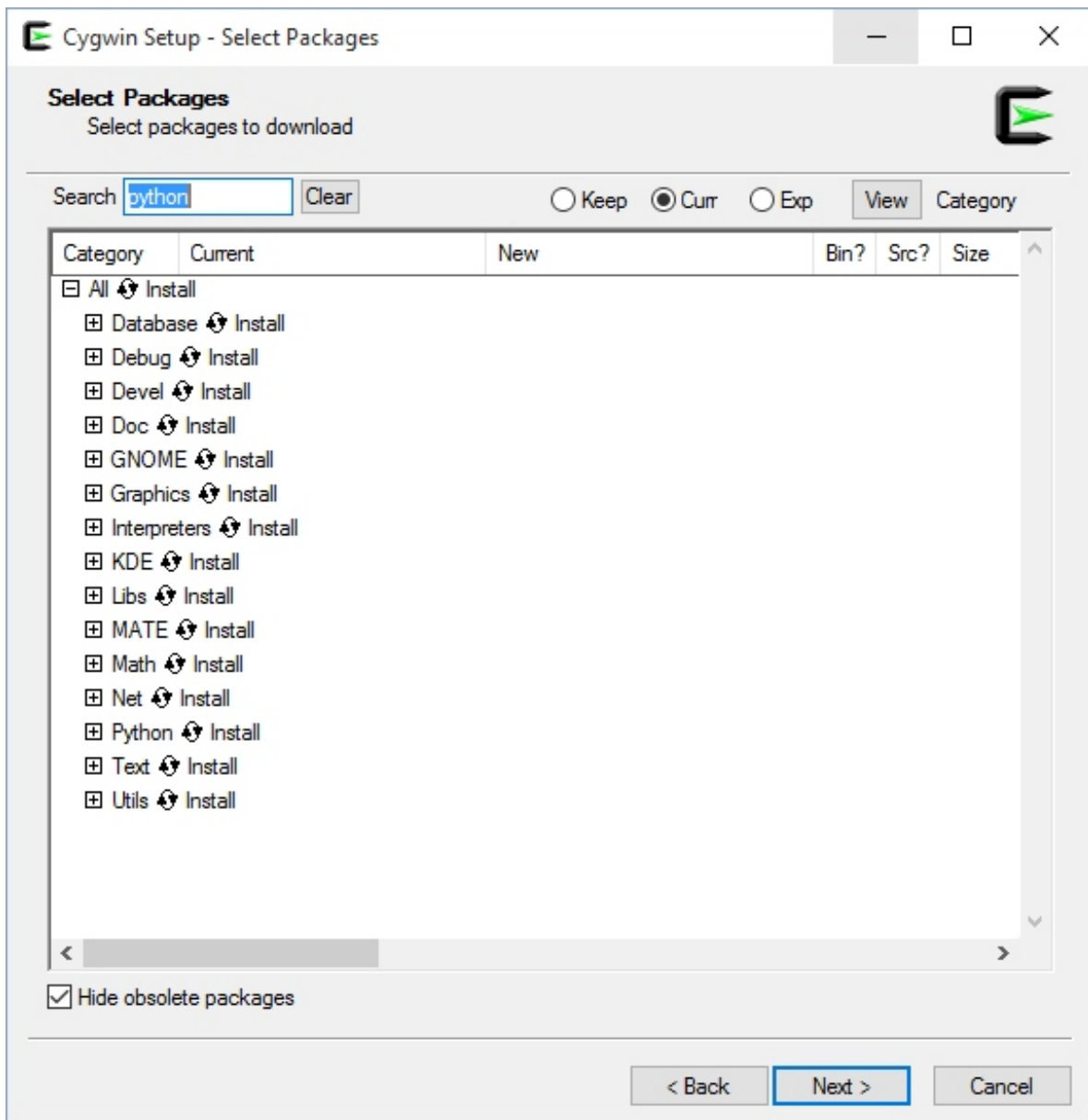
will provide details of installed packages.

The following commands

```
cygcheck -c > installed.txt
$ cat installed.txt | grep python | wc
    236      708   18644
```

shows 236 installed Python components on one cygwin installation.

Here is a screen shot of part of a cygwin install.



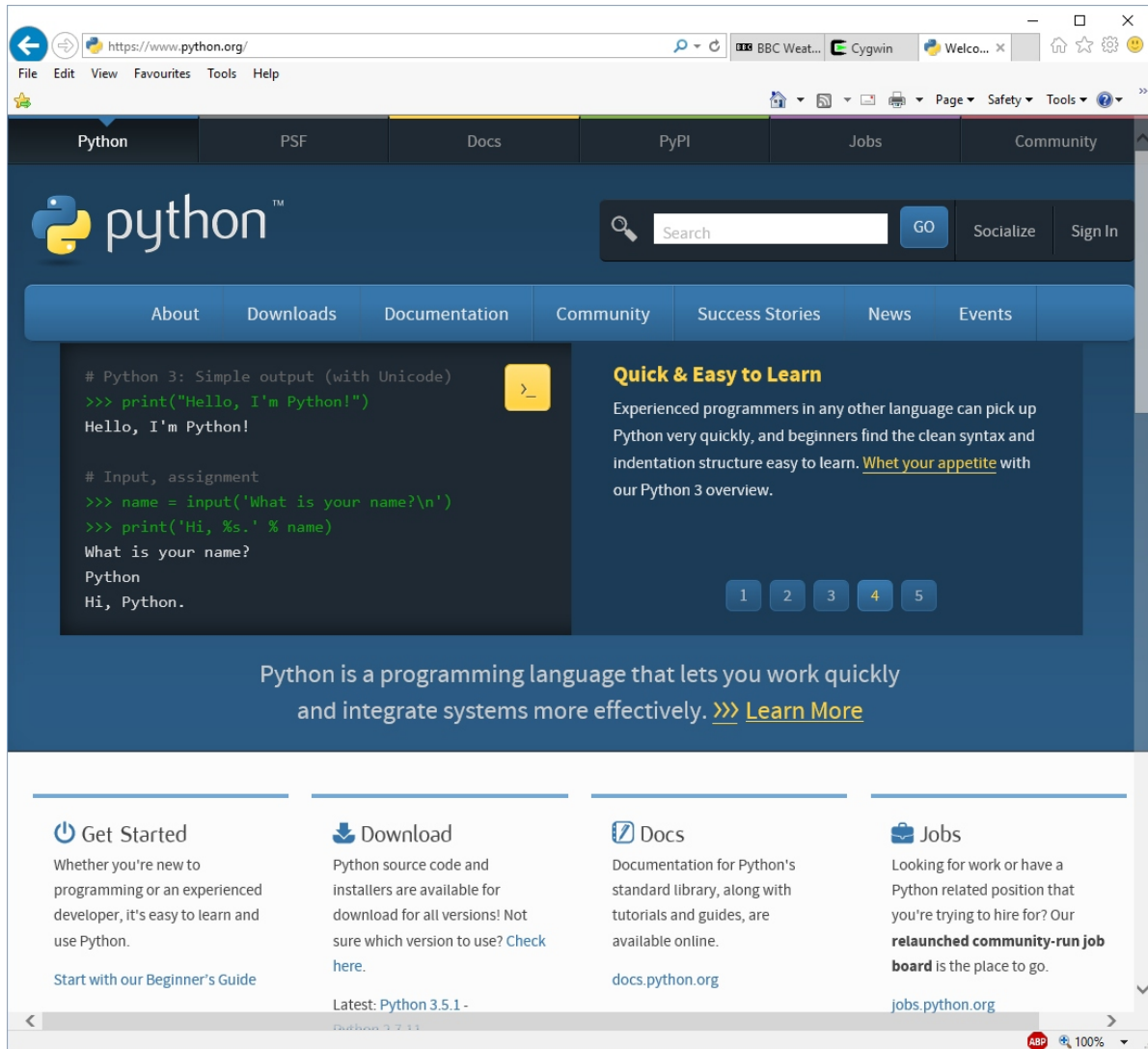
Note the python keyword in the search box. I normally split the install into two stages
first a download
and secondly
install from local directory
in case anything goes wrong with the download.

1.7.3 Windows - Python download

Visit

<https://www.python.org/downloads/>

to download the software. Here is a screenshot of their home page.



I downloaded the 3.5.x version.

1.7.4 Windows and Microsoft Visual Studio

Another option is to use Microsoft Visual Studio. On my laptop I have

Microsoft Visual Studio Community 2017, 15.6.4

installed. There is an option to create Python projects from within Visual Studio. There are the following options:

Global default or an auto detected environment

Anaconda 5.0.0

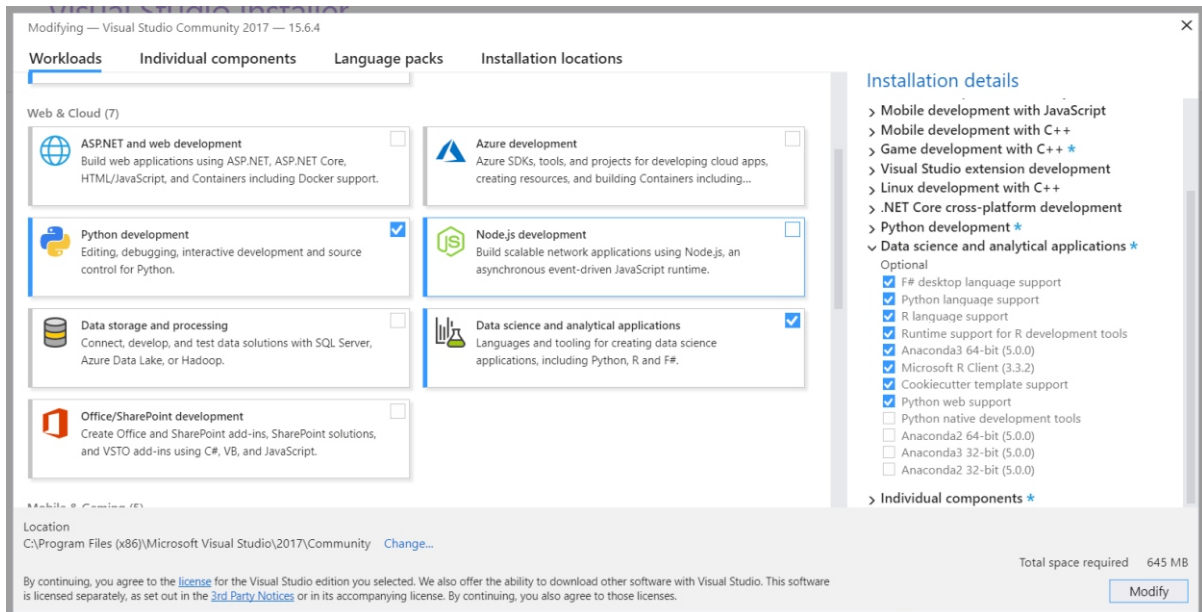
Python 3.6 (64 bit)

The IDE is Python aware and the code is colour coded. I set things up to point to my default example directory

`c:\documnt\python\examples`

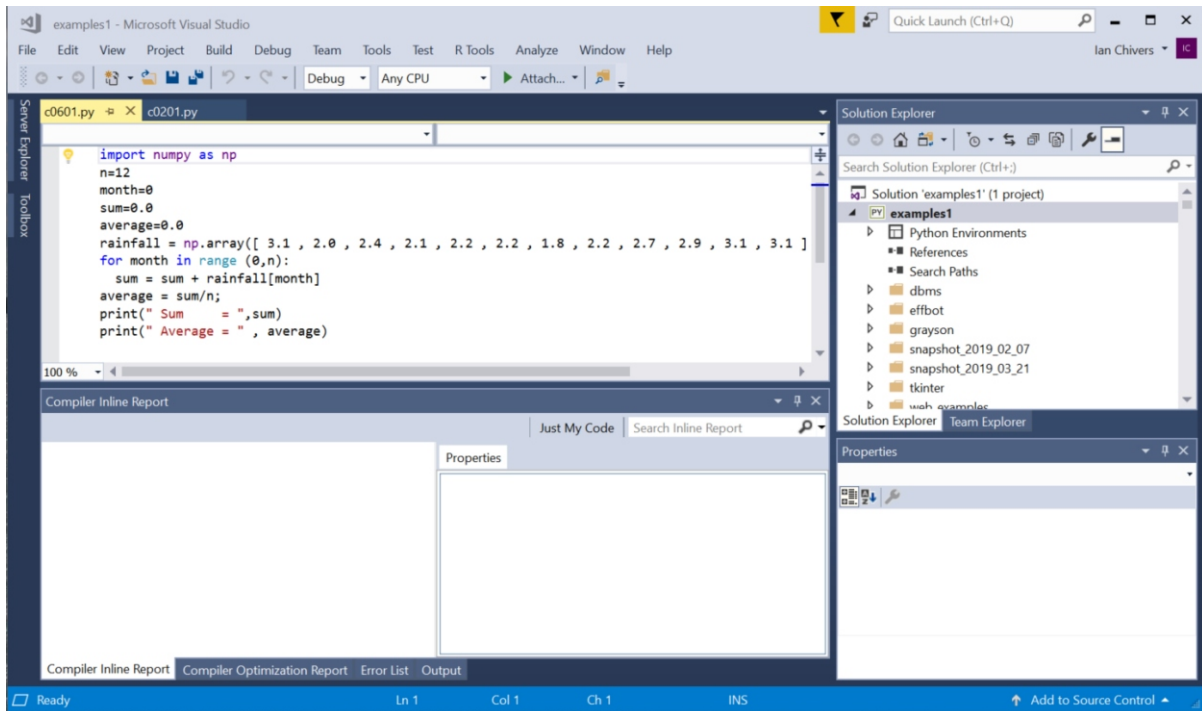
and all of the examples appear in one of the Windows. They can be clicked on and run individually.

On one of my desk top systems I needed to install additional components. Here is a screen shot.



The download and install took several minutes.

Here is a screen shot of using Visual Studio for Python development.



As can be seen in the on line version of these notes Visual Studio is Python aware, and the Python code is colour coded.

When setting up a project there are several options for a version of Python. On one system I have there were the following options

- Global default
- Anaconda 4.1.1
- Anaconda 5.2.0
- Anaconda3 3.6
- Python 3.5
- Python 3.6

I chose Anaconda 3 3.6 for my numpy examples.

1.7.4.1 Visual Studio Community Edition 2019

Visual Studio 2019 has just been released, and I chose to install the following options

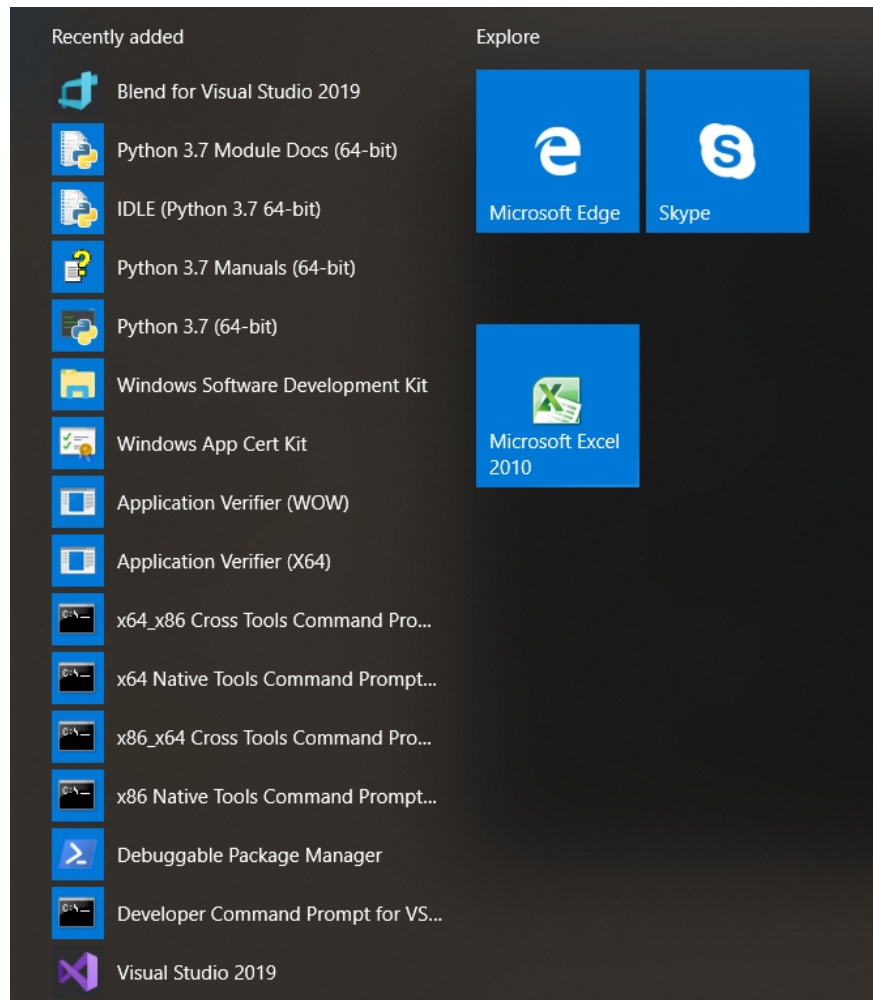
- Python
 - Python miniconda
 - Python web support
 - Python 3 64-bit (3.7.2)
 - Live Share
 - 2.26 GB
- Windows (3)
 - .NET desktop development
 - 5.23 GB

Desktop development

7.12 GB

9.89 GB Combined

Choosing the [Start] button brings up the following



and there are several Python options.

1.7.5 Windows subsystem for Linux and Python install

Windows Subsystem for Linux - The following has been taken from the Wikipedia entry.

https://en.wikipedia.org/wiki/Windows_Subsystem_for_Linux

Windows Subsystem for Linux (WSL) is a compatibility layer for running Linux binary executables (in ELF format) natively on Windows 10 and Windows Server 2019.

WSL provides a Linux-compatible kernel interface developed by Microsoft (containing no Linux kernel code), which can then run a GNU user space on top of it, such as that of Ubuntu, openSUSE, SUSE Linux Enterprise Server, Debian and Kali Linux. Such a user space might contain a Bash shell and command language, with native GNU/Linux command-line tools (sed, awk, etc.), programming language interpreters (Ruby, Python, etc.), and even graphical applications (using a X11 server at the host side).

Introduction and availability - When introduced with the Anniversary Update, only an Ubuntu image was available. The Fall Creators Update moved the installation process for Linux distributions to the Windows Store, and introduced SUSE images.

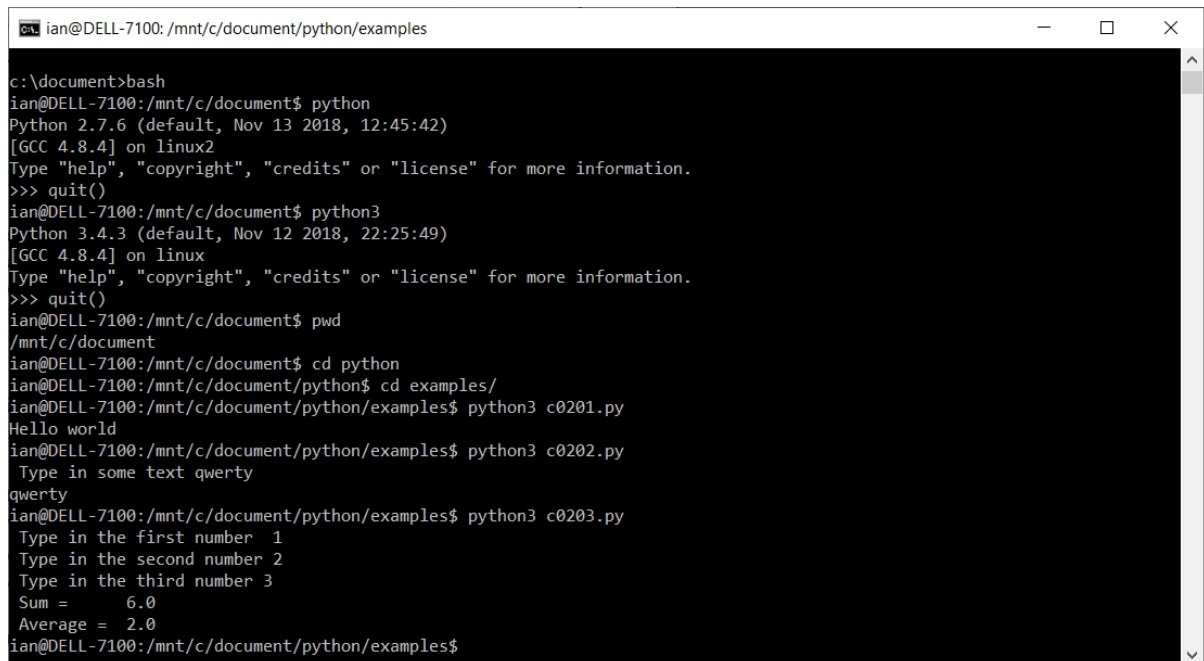
WSL is available only in 64 bit versions of Windows 10 from version 1607. It is also available in Windows Server 2019.

I recommend installing the Ubuntu version. This will make available complete Unix functionality on the Windows platform. I use vi, sed, diff etc on a regular basis whilst programming.

After starting bash you will need to run one or more of the following commands:

```
sudo apt install python-minimal
sudo apt install python3
```

Here is a screen shot of using WSL on one of my systems.

A screenshot of a terminal window titled 'ian@DELL-7100: /mnt/c/document/python/examples'. The terminal shows the following commands and output:

```
c:\document>bash
ian@DELL-7100:/mnt/c/document$ python
Python 2.7.6 (default, Nov 13 2018, 12:45:42)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
ian@DELL-7100:/mnt/c/document$ python3
Python 3.4.3 (default, Nov 12 2018, 22:25:49)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
ian@DELL-7100:/mnt/c/document$ pwd
/mnt/c/document
ian@DELL-7100:/mnt/c/document$ cd python
ian@DELL-7100:/mnt/c/document/python$ cd examples/
ian@DELL-7100:/mnt/c/document/python/examples$ python3 c0201.py
Hello world
ian@DELL-7100:/mnt/c/document/python/examples$ python3 c0202.py
Type in some text qwerty
qwerty
ian@DELL-7100:/mnt/c/document/python/examples$ python3 c0203.py
Type in the first number 1
Type in the second number 2
Type in the third number 3
Sum = 6.0
Average = 2.0
ian@DELL-7100:/mnt/c/document/python/examples$
```

I am compiling with Python3.

The following are some of the useful commands

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install
sudo apt-get full-upgrade
sudo apt-get install dos2unix
sudo apt-get install python3
```

when managing software.

1.7.6 Windows Hyper-V manager

Visit

<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v>

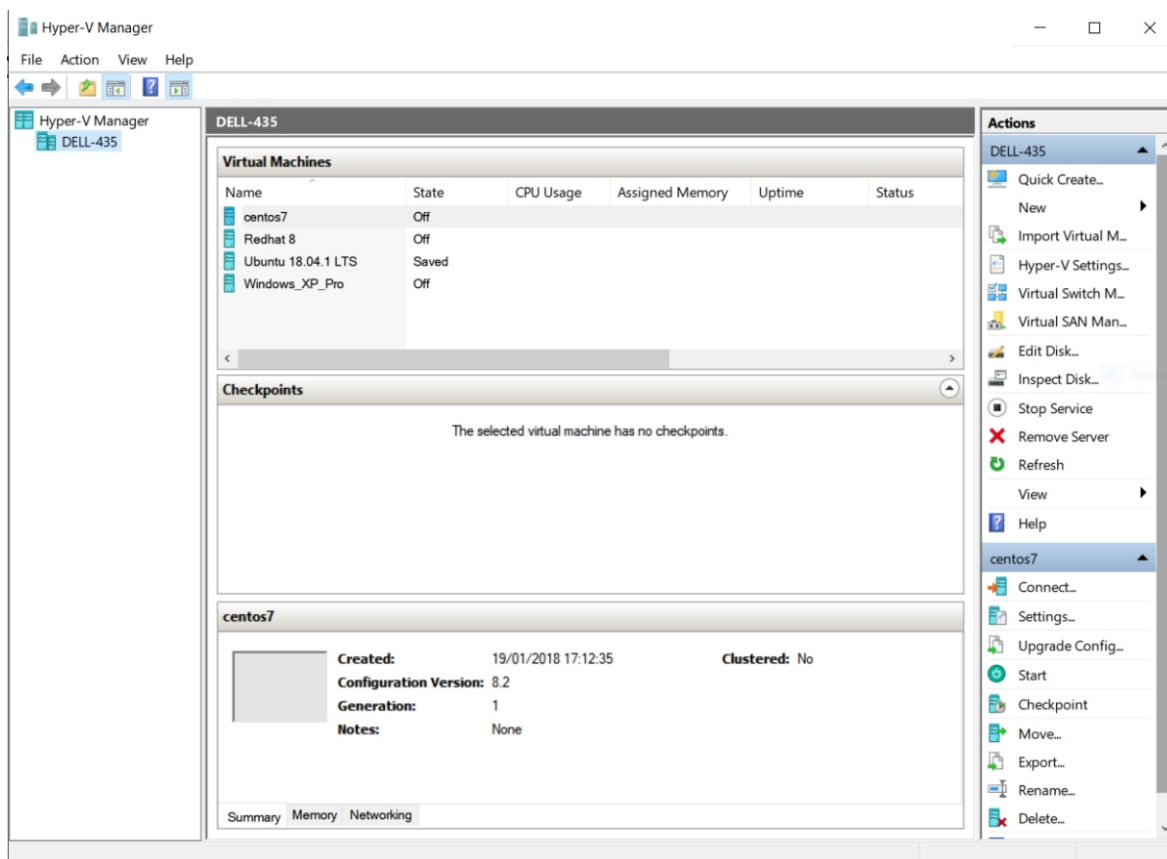
for details of how to enable and install Hyper-V manager.

Then visit

<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/quick-create-virtual-machine>

for details of how to install one or more virtual machines.

Here is a screen shot from one of my Windows 10 Pro systems that has Hyper-V manager installed.



As you can see I have the following operating systems installed

centos 7

Redhat 8

Ubuntu 18.x

Windows XP Pro

Python can be installed on the three Linux systems.

The following command

```
sudo apt -install python3
```

installs Python 3 on the Ubuntu system.

I then did

```
sudo apt update
```

```
sudo apt upgrade
```

to upgrade the base operating system, followed by

```
sudo apt install python3-numpy python3-scipy
python3-matplotlib
```

to install the rest of the Python environment.

1.8 Linux

In this section we will look at some of the options of using Python on a Linux platform. We will look at OpenSuSe Linux.

1.8.1 Python and openSuSe

I used openSuSe 13.1 writing these notes. Python was already installed. I normally do a fairly complete software install when installing openSuSe Linux as I use Linux for a wide range of purposes.

```
rpm -qa > rpm_list.txt
```

will give a complete list of all software installed. You can then use `grep` on the file to see what Python components are installed.

Here are some details from one of the openSuSe 13.1 systems that I use.

```
cat rpm_list.txt | grep python | wc
28094 lines
cat rpm_list.txt | grep python3 | wc
2145 lines
```

1.8.2 Python, openSuSe and an anaconda installation

I recommend doing an anaconda Python install. The first thing to do is download an appropriate version.

```
https://www.continuum.io/downloads
```

I downloaded the 64 bit version. I recommend logging on as root and installing in `/opt/anaconda3`

and adding

```
/opt/anaconda3/bin
```

to the PATH.

The download file was called

```
Anaconda3-2.4.1-Linux-x86_64.sh
```

and typing

```
bash Anaconda3-2.4.1-Linux-x86_64.sh
```

from the console started the installation process. Change the installation directory to that show above.

1.9 Intel Python for Windows, Linux and Mac

Intel make available Python for all three platforms. My Intel Fortran licence includes access to the software. I use the software on one of my systems.

1.10 Mapping with Python - basemap

Visit

```
http://matplotlib.org/basemap/
```

for details of how to do mapping with Python. Windows executables and Linux tar files are available.

If you do the Anaconda install above typing

```
conda install basemap
```

from the shell will install basemap, and all necessary software.

1.11 Mapping with Python - Cartopy

Visit

<https://scitools.org.uk/cartopy/docs/latest/>

for up to date information.

Here is an extract from that site.

Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.

Cartopy makes use of the powerful PROJ.4, NumPy and Shapely libraries and includes a programmatic interface built on top of Matplotlib for the creation of publication quality maps.

Key features of cartopy are its object oriented projection definitions, and its ability to transform points, lines, vectors, polygons and images between those projections.

You will find cartopy especially useful for large area / small scale data, where Cartesian assumptions of spherical data traditionally break down. If you've ever experienced a singularity at the pole or a cut-off at the dateline, it is likely you will appreciate cartopy's unique features!

The installation guide provides information on getting up and running. Cartopy's documentation is arranged in userguide form, with reference documentation available inline.

- Coordinate reference systems in Cartopy

- Cartopy projection list

- Using cartopy with matplotlib

- The cartopy Feature interface

- Understanding the transform and projection keywords

- Using the cartopy shapereader

- Cartopy developer interfaces

The outline link found above the cartopy logo on all pages can be used to quickly find the reference documentation for known classes or functions.

For those updating from an older version of cartopy, the what's new page outlines recent changes, new features, and future development plans.

Cartopy was originally developed at the UK Met Office to allow scientists to visualise their data on maps quickly, easily and most importantly, accurately. Cartopy has been made freely available under the terms of the GNU Lesser General Public License. It is suitable to be used in a variety of scientific fields and has an active development community.

Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.

Cartopy makes use of the powerful PROJ.4, numpy and shapely libraries and includes a programatic interface built on top of Matplotlib for the creation of publication quality maps.

Key features of cartopy are its object oriented projection definitions, and its ability to transform points, lines, vectors, polygons and images between those projections.

You will find cartopy especially useful for large area / small scale data, where Cartesian assumptions of spherical data traditionally break down. If you've ever experienced a singularity at the pole or a cut-off at the dateline, it is likely you will appreciate cartopy's unique features!

1.12 Python on line documentation

Visit

<https://docs.python.org/3/download.html>

Here is some information from that site.

Download Python 3.7.3rc1 Documentation

Last updated on: Mar 20, 2019.

To download an archive containing all the documents for this version of Python in one of various formats, follow one of links in this table. The numbers in the table are the size of the download files in megabytes.

Format

Packed as .zip

Packed as .tar.bz2

PDF (US-Letter paper size) Download (ca. 13 MiB) Download (ca. 13 MiB)

PDF (A4 paper size) Download (ca. 13 MiB) Download (ca. 13 MiB)

HTML Download (ca. 9 MiB) Download (ca. 6 MiB)

Plain Text Download (ca. 3 MiB) Download (ca. 2 MiB)

EPUB Download (ca. 5 MiB)

These archives contain all the content in the documentation.

HTML Help (.chm) files are made available in the "Windows" section on the Python download page.

Unpacking

Unix users should download the .tar.bz2 archives; these are bziped tar archives and can be handled in the usual way using tar and the bzip2 program. The InfoZIP unzip program can be used to handle the ZIP archives if desired. The .tar.bz2 archives provide the best compression and fastest download times.

Windows users can use the ZIP archives since those are customary on that platform. These are created on Unix using the InfoZIP zip program.

I recommend downloading the A4 pdf versions.

The library reference can be found at

<https://docs.python.org/3/library/index.html>

1. Introduction
2. Built-in Functions
3. Built-in Constants
 - 3.1. Constants added by the site module
4. Built-in Types
 - 4.1. Truth Value Testing
 - 4.2. Boolean Operations — and, or, not
 - 4.3. Comparisons
 - 4.4. Numeric Types — int, float, complex
 - 4.5. Iterator Types
 - 4.6. Sequence Types — list, tuple, range
 - 4.7. Text Sequence Type — str
 - 4.8. Binary Sequence Types — bytes, bytearray, memoryview
 - 4.9. Set Types — set, frozenset
 - 4.10. Mapping Types — dict
 - 4.11. Context Manager Types
 - 4.12. Other Built-in Types
 - 4.13. Special Attributes
5. Built-in Exceptions
 - 5.1. Base classes
 - 5.2. Concrete exceptions
 - 5.3. Warnings
 - 5.4. Exception hierarchy
6. Text Processing Services
 - 6.1. string — Common string operations
 - 6.2. re — Regular expression operations
 - 6.3. difflib — Helpers for computing deltas
 - 6.4. textwrap — Text wrapping and filling
 - 6.5. unicodedata — Unicode Database
 - 6.6. stringprep — Internet String Preparation
 - 6.7. readline — GNU readline interface
 - 6.8. rlcompleter — Completion function for GNU readline
7. Binary Data Services

- 7.1. struct — Interpret bytes as packed binary data
- 7.2. codecs — Codec registry and base classes

- 8. Data Types
 - 8.1. datetime — Basic date and time types
 - 8.2. calendar — General calendar-related functions
 - 8.3. collections — Container datatypes
 - 8.4. collections.abc — Abstract Base Classes for Containers
 - 8.5. heapq — Heap queue algorithm
 - 8.6. bisect — Array bisection algorithm
 - 8.7. array — Efficient arrays of numeric values
 - 8.8. weakref — Weak references
 - 8.9. types — Dynamic type creation and names for built-in types
 - 8.10. copy — Shallow and deep copy operations
 - 8.11. pprint — Data pretty printer
 - 8.12. reprlib — Alternate repr() implementation
 - 8.13. enum — Support for enumerations

- 9. Numeric and Mathematical Modules
 - 9.1. numbers — Numeric abstract base classes
 - 9.2. math — Mathematical functions
 - 9.3. cmath — Mathematical functions for complex numbers
 - 9.4. decimal — Decimal fixed point and floating point arithmetic
 - 9.5. fractions — Rational numbers
 - 9.6. random — Generate pseudo-random numbers
 - 9.7. statistics — Mathematical statistics functions

- 10. Functional Programming Modules
 - 10.1. itertools — Functions creating iterators for efficient looping
 - 10.2. functools — Higher-order functions and operations on callable objects
 - 10.3. operator — Standard operators as functions

- 11. File and Directory Access
 - 11.1. pathlib — Object-oriented filesystem paths
 - 11.2. os.path — Common pathname manipulations
 - 11.3. fileinput — Iterate over lines from multiple input streams
 - 11.4. stat — Interpreting stat() results

- 11.5. filecmp — File and Directory Comparisons
 - 11.6. tempfile — Generate temporary files and directories
 - 11.7. glob — Unix style pathname pattern expansion
 - 11.8. fnmatch — Unix filename pattern matching
 - 11.9. linecache — Random access to text lines
 - 11.10. shutil — High-level file operations
 - 11.11. macpath — Mac OS 9 path manipulation functions
-
- 12. Data Persistence
 - 12.1. pickle — Python object serialization
 - 12.2. copyreg — Register pickle support functions
 - 12.3. shelve — Python object persistence
 - 12.4. marshal — Internal Python object serialization
 - 12.5. dbm — Interfaces to Unix “databases”
 - 12.6. sqlite3 — DB-API 2.0 interface for SQLite databases
-
- 13. Data Compression and Archiving
 - 13.1. zlib — Compression compatible with gzip
 - 13.2. gzip — Support for gzip files
 - 13.3. bz2 — Support for bzip2 compression
 - 13.4. lzma — Compression using the LZMA algorithm
 - 13.5. zipfile — Work with ZIP archives
 - 13.6. tarfile — Read and write tar archive files
-
- 14. File Formats
 - 14.1. csv — CSV File Reading and Writing
 - 14.2. configparser — Configuration file parser
 - 14.3. netrc — netrc file processing
 - 14.4. xdrlib — Encode and decode XDR data
 - 14.5. plistlib — Generate and parse Mac OS X .plist files
-
- 15. Cryptographic Services
 - 15.1. hashlib — Secure hashes and message digests
 - 15.2. hmac — Keyed-Hashing for Message Authentication
-
- 16. Generic Operating System Services
 - 16.1. os — Miscellaneous operating system interfaces

- 16.2. `io` — Core tools for working with streams
 - 16.3. `time` — Time access and conversions
 - 16.4. `argparse` — Parser for command-line options, arguments and sub-commands
 - 16.5. `getopt` — C-style parser for command line options
 - 16.6. `logging` — Logging facility for Python
 - 16.7. `logging.config` — Logging configuration
 - 16.8. `logging.handlers` — Logging handlers
 - 16.9. `getpass` — Portable password input
 - 16.10. `curses` — Terminal handling for character-cell displays
 - 16.11. `curses.textpad` — Text input widget for curses programs
 - 16.12. `curses.ascii` — Utilities for ASCII characters
 - 16.13. `curses.panel` — A panel stack extension for curses
 - 16.14. `platform` — Access to underlying platform's identifying data
 - 16.15. `errno` — Standard errno system symbols
 - 16.16. `ctypes` — A foreign function library for Python
-
- 17. Concurrent Execution
 - 17.1. `threading` — Thread-based parallelism
 - 17.2. `multiprocessing` — Process-based parallelism
 - 17.3. The concurrent package
 - 17.4. `concurrent.futures` — Launching parallel tasks
 - 17.5. `subprocess` — Subprocess management
 - 17.6. `sched` — Event scheduler
 - 17.7. `queue` — A synchronized queue class
 - 17.8. `dummy_threading` — Drop-in replacement for the `threading` module
 - 17.9. `_thread` — Low-level threading API
 - 17.10. `_dummy_thread` — Drop-in replacement for the `_thread` module
-
- 18. Interprocess Communication and Networking
 - 18.1. `socket` — Low-level networking interface
 - 18.2. `ssl` — TLS/SSL wrapper for socket objects
 - 18.3. `select` — Waiting for I/O completion
 - 18.4. `selectors` — High-level I/O multiplexing
 - 18.5. `asyncio` — Asynchronous I/O, event loop, coroutines and tasks
 - 18.6. `asyncore` — Asynchronous socket handler
 - 18.7. `asynchat` — Asynchronous socket command/response handler

- 18.8. `signal` — Set handlers for asynchronous events
- 18.9. `mmap` — Memory-mapped file support

- 19. Internet Data Handling
 - 19.1. `email` — An email and MIME handling package
 - 19.2. `json` — JSON encoder and decoder
 - 19.3. `mailcap` — Mailcap file handling
 - 19.4. `mailbox` — Manipulate mailboxes in various formats
 - 19.5. `mimetypes` — Map filenames to MIME types
 - 19.6. `base64` — Base16, Base32, Base64, Base85 Data Encodings
 - 19.7. `binhex` — Encode and decode binhex4 files
 - 19.8. `binascii` — Convert between binary and ASCII
 - 19.9. `quopri` — Encode and decode MIME quoted-printable data
 - 19.10. `uu` — Encode and decode uuencode files

- 20. Structured Markup Processing Tools
 - 20.1. `html` — HyperText Markup Language support
 - 20.2. `html.parser` — Simple HTML and XHTML parser
 - 20.3. `html.entities` — Definitions of HTML general entities
 - 20.4. XML Processing Modules
 - 20.5. `xml.etree.ElementTree` — The ElementTree XML API
 - 20.6. `xml.dom` — The Document Object Model API
 - 20.7. `xml.dom.minidom` — Minimal DOM implementation
 - 20.8. `xml.dom.pulldom` — Support for building partial DOM trees
 - 20.9. `xml.sax` — Support for SAX2 parsers
 - 20.10. `xml.sax.handler` — Base classes for SAX handlers
 - 20.11. `xml.sax.saxutils` — SAX Utilities
 - 20.12. `xml.sax.xmlreader` — Interface for XML parsers
 - 20.13. `xml.parsers.expat` — Fast XML parsing using Expat

- 21. Internet Protocols and Support
 - 21.1. `webbrowser` — Convenient Web-browser controller
 - 21.2. `cgi` — Common Gateway Interface support
 - 21.3. `cgitb` — Traceback manager for CGI scripts
 - 21.4. `wsgiref` — WSGI Utilities and Reference Implementation
 - 21.5. `urllib` — URL handling modules
 - 21.6. `urllib.request` — Extensible library for opening URLs

- 21.7. urllib.response — Response classes used by urllib
- 21.8. urllib.parse — Parse URLs into components
- 21.9. urllib.error — Exception classes raised by urllib.request
- 21.10. urllib.robotparser — Parser for robots.txt
- 21.11. http — HTTP modules
- 21.12. http.client — HTTP protocol client
- 21.13. ftplib — FTP protocol client
- 21.14. poplib — POP3 protocol client
- 21.15. imaplib — IMAP4 protocol client
- 21.16. nntplib — NNTP protocol client
- 21.17. smtpplib — SMTP protocol client
- 21.18. smtpd — SMTP Server
- 21.19. telnetlib — Telnet client
- 21.20. uuid — UUID objects according to RFC 4122
- 21.21. socketserver — A framework for network servers
- 21.22. http.server — HTTP servers
- 21.23. http.cookies — HTTP state management
- 21.24. http.cookiejar — Cookie handling for HTTP clients
- 21.25. xmlrpc — XMLRPC server and client modules
- 21.26. xmlrpc.client — XML-RPC client access
- 21.27. xmlrpc.server — Basic XML-RPC servers
- 21.28. ipaddress — IPv4/IPv6 manipulation library

22. Multimedia Services

- 22.1. audioop — Manipulate raw audio data
- 22.2. aifc — Read and write AIFF and AIFC files
- 22.3. sunau — Read and write Sun AU files
- 22.4. wave — Read and write WAV files
- 22.5. chunk — Read IFF chunked data
- 22.6. colorsys — Conversions between color systems
- 22.7. imghdr — Determine the type of an image
- 22.8. sndhdr — Determine type of sound file
- 22.9. ossaudiodev — Access to OSS-compatible audio devices

23. Internationalization

- 23.1. gettext — Multilingual internationalization services
- 23.2. locale — Internationalization services

24. Program Frameworks

24.1. turtle — Turtle graphics

24.2. cmd — Support for line-oriented command interpreters

24.3. shlex — Simple lexical analysis

25. Graphical User Interfaces with Tk

25.1. tkinter — Python interface to Tcl/Tk

25.2. tkinter.ttk — Tk themed widgets

25.3. tkinter.tix — Extension widgets for Tk

25.4. tkinter.scrolledtext — Scrolled Text Widget

25.5. IDLE

25.6. Other Graphical User Interface Packages

26. Development Tools

26.1. typing — Support for type hints

26.2. pydoc — Documentation generator and online help system

26.3. doctest — Test interactive Python examples

26.4. unittest — Unit testing framework

26.5. unittest.mock — mock object library

26.6. unittest.mock — getting started

26.7. 2to3 - Automated Python 2 to 3 code translation

26.8. test — Regression tests package for Python

26.9. test.support — Utilities for the Python test suite

27. Debugging and Profiling

27.1. bdb — Debugger framework

27.2. faulthandler — Dump the Python traceback

27.3. pdb — The Python Debugger

27.4. The Python Profilers

27.5. timeit — Measure execution time of small code snippets

27.6. trace — Trace or track Python statement execution

27.7. tracemalloc — Trace memory allocations

28. Software Packaging and Distribution

28.1. distutils — Building and installing Python modules

28.2. ensurepip — Bootstrapping the pip installer

- 28.3. venv — Creation of virtual environments
- 28.4. zipapp — Manage executable python zip archives

- 29. Python Runtime Services
 - 29.1. sys — System-specific parameters and functions
 - 29.2. sysconfig — Provide access to Python's configuration information
 - 29.3. builtins — Built-in objects
 - 29.4. __main__ — Top-level script environment
 - 29.5. warnings — Warning control
 - 29.6. contextlib — Utilities for with-statement contexts
 - 29.7. abc — Abstract Base Classes
 - 29.8. atexit — Exit handlers
 - 29.9. traceback — Print or retrieve a stack traceback
 - 29.10. __future__ — Future statement definitions
 - 29.11. gc — Garbage Collector interface
 - 29.12. inspect — Inspect live objects
 - 29.13. site — Site-specific configuration hook
 - 29.14. fpectl — Floating point exception control

- 30. Custom Python Interpreters
 - 30.1. code — Interpreter base classes
 - 30.2. codeop — Compile Python code

- 31. Importing Modules
 - 31.1. zipimport — Import modules from Zip archives
 - 31.2. pkgutil — Package extension utility
 - 31.3. modulefinder — Find modules used by a script
 - 31.4. runpy — Locating and executing Python modules
 - 31.5. importlib — The implementation of import

- 32. Python Language Services
 - 32.1. parser — Access Python parse trees
 - 32.2. ast — Abstract Syntax Trees
 - 32.3. symtable — Access to the compiler's symbol tables
 - 32.4. symbol — Constants used with Python parse trees
 - 32.5. token — Constants used with Python parse trees
 - 32.6. keyword — Testing for Python keywords

- 32.7. tokenize — Tokenizer for Python source
- 32.8. tabnanny — Detection of ambiguous indentation
- 32.9. pycbr — Python class browser support
- 32.10. py_compile — Compile Python source files
- 32.11. compileall — Byte-compile Python libraries
- 32.12. dis — Disassembler for Python bytecode
- 32.13. pickletools — Tools for pickle developers

- 33. Miscellaneous Services
 - 33.1. formatter — Generic output formatting

- 34. MS Windows Specific Services
 - 34.1. msilib — Read and write Microsoft Installer files
 - 34.2. msvcrt – Useful routines from the MS VC++ runtime
 - 34.3. winreg – Windows registry access
 - 34.4. winsound — Sound-playing interface for Windows

- 35. Unix Specific Services
 - 35.1. posix — The most common POSIX system calls
 - 35.2. pwd — The password database
 - 35.3. spwd — The shadow password database
 - 35.4. grp — The group database
 - 35.5. crypt — Function to check Unix passwords
 - 35.6. termios — POSIX style tty control
 - 35.7. tty — Terminal control functions
 - 35.8. pty — Pseudo-terminal utilities
 - 35.9. fcntl — The fcntl and ioctl system calls
 - 35.10. pipes — Interface to shell pipelines
 - 35.11. resource — Resource usage information
 - 35.12. nis — Interface to Sun’s NIS (Yellow Pages)
 - 35.13. syslog — Unix syslog library routines

- 36. Superseded Modules
 - 36.1. optparse — Parser for command line options
 - 36.2. imp — Access the import internals

- 37. Undocumented Modules

37.1. Platform specific modules

The above combined with the language reference (which can be found below)

<https://docs.python.org/3/reference/index.html>

1. Introduction

1.1. Alternate Implementations

1.2. Notation

2. Lexical analysis

2.1. Line structure

2.2. Other tokens

2.3. Identifiers and keywords

2.4. Literals

2.5. Operators

2.6. Delimiters

3. Data model

3.1. Objects, values and types

3.2. The standard type hierarchy

3.3. Special method names

3.4. Coroutines

4. Execution model

4.1. Structure of a program

4.2. Naming and binding

4.3. Exceptions

5. The import system

5.1. importlib

5.2. Packages

5.3. Searching

5.4. Loading

5.5. The Path Based Finder

5.6. Replacing the standard import system

5.7. Special considerations for `__main__`

5.8. Open issues

5.9. References

- 6. Expressions
 - 6.1. Arithmetic conversions
 - 6.2. Atoms
 - 6.3. Primaries
 - 6.4. Await expression
 - 6.5. The power operator
 - 6.6. Unary arithmetic and bitwise operations
 - 6.7. Binary arithmetic operations
 - 6.8. Shifting operations
 - 6.9. Binary bitwise operations
 - 6.10. Comparisons
 - 6.11. Boolean operations
 - 6.12. Conditional expressions
 - 6.13. Lambdas
 - 6.14. Expression lists
 - 6.15. Evaluation order
 - 6.16. Operator precedence

- 7. Simple statements
 - 7.1. Expression statements
 - 7.2. Assignment statements
 - 7.3. The assert statement
 - 7.4. The pass statement
 - 7.5. The del statement
 - 7.6. The return statement
 - 7.7. The yield statement
 - 7.8. The raise statement
 - 7.9. The break statement
 - 7.10. The continue statement
 - 7.11. The import statement
 - 7.12. The global statement
 - 7.13. The nonlocal statement

- 8. Compound statements
 - 8.1. The if statement
 - 8.2. The while statement
 - 8.3. The for statement

- 8.4. The try statement
- 8.5. The with statement
- 8.6. Function definitions
- 8.7. Class definitions
- 8.8. Coroutines

9. Top-level components

- 9.1. Complete Python programs
- 9.2. File input
- 9.3. Interactive input
- 9.4. Expression input

10. Full Grammar specification

should provide you a good starting point.

1.12.1 Published books and on line electronic manuscripts

Here are details of some online material (html and pdf format) and books (printed and electronic, mainly pdf.).

Reference material

A down loadable zip file can be found at

<https://docs.python.org/3/download.html>

There are the following documents.

c-api.pdf	209
distributing.pdf	48
extending.pdf	102
faq.pdf	110
howto-argparse.pdf	12
howto-clinic.pdf	23
howto-cporting.pdf	8
howto-curses.pdf	8
howto-descriptor.pdf	7
howto-functional.pdf	19
howto-ipaddress.pdf	6
howto-logging-cookbook.pdf	37
howto-logging.pdf	16
howto-pyporting.pdf	6
howto-regex.pdf	17
howto-sockets.pdf	6
howto-sorting.pdf	5
howto-unicode.pdf	12

howto-urllib2.pdf	11
howto-webservers.pdf	11
installing.pdf	46
library.pdf	1,722
reference.pdf	141
tutorial.pdf	133
using.pdf	69
whatsnew.pdf	33

I have added a page count for each document.

Python Standard Library

The Python Standard Library by Example, Doug Hellman, Addison Wesley, ISBN 978-0-321-76734-9. Bonus is that a free e-version comes with it. 1302.

I have bought this. It is a good starting place as it has a lot of simple examples. Buying the book provided me with access to the ebook.

Algorithms and Data Structures

Kent Lee, Steve Hubbard, Data Structures and Algorithms with Python, Springer, ISSN 1863-7310 ISSN 2197-1781 (electronic) ISBN 978-3-319-13071-2 ISBN 978-3-319-13072-9 (eBook), 369.

I have the pdf version of this. Good coverage of algorithms and data structures in Python.

tkinter

Python and Tkinter Programming, John Grayson, Manning, paper and ebook 684 pages. I bought the paper version and that provided access to the ebook.

John Shipman, Tkinter 8.5 reference: A GUI for Python. Free download.

Stephen Freg, Thinking in Tkinter, 32. Free download.

Numpy

Numpy User Guide, 107. Free download.

Numpy Reference Manual, 1528. Free download.

matplotlib

matplotlib manual, 2,824 pages, pdf.

Miscellaneous

Scipy Lecture Notes, 367. Free download.

How to think like a Computer Scientist, Allen Downey, Jeffrey Elkner, Chris Meyers, Green Tea Press, Wellesley, Massachusetts, 288. Free download.

Amit Saha, Doing Math with Python: Use Programming to Explore Algebra, Statistics, Calculus, and More!, August 2015, 264 pp. ISBN: 978-1-59327-640-9

Chapter 1: Working with Numbers

Chapter 2: Visualizing Data with Graphs

Chapter 3: Describing Data with Statistics

Chapter 4: Algebra and Symbolic Math with SymPy

Chapter 5: Playing with Sets and Probability

Chapter 6: Drawing Geometric Shapes and Fractals

Chapter 7: Solving Calculus Problems

<https://www.nostarch.com/doingmathwithpython>

Python programming for Research, Volumes 1 and 2, UCL London. UCL have material available. Visit

<http://development.rc.ucl.ac.uk/training/introductory/>

<http://development.rc.ucl.ac.uk/training/engineering/>

1.13 Download and installation summary

So there are a lot of Python installed components. The table below provides details of what is installed on some of the systems I use.

1.13.1 Summary of systems setups

Here is a summary of the Python setups on the systems used in producing these notes from 2015 to 2020.

Operating system and hardware	Component	Dell 435 Desktop	Dell 7100 Desktop	PC Specialist Laptop
RAM - GB		24	16	16
Windows				
	cygwin			
	python	N/A	2.7.15	2.7.10
	python3	N/A	3.6.8	3.4.3
	python	N/A	N/A	N/A
	anaconda			
	python	3.6.8	3.6.8	
	basemap			
	cartopy			
	visual studio 2017		Installed	
	Intel Python			
Linux		openSuSe	openSuSe	openSuSe
		15.0	15.0	15.0
	python	2.7.14	2.7.14	2.7.14
	python3	3.6.5	3.6.5	3.6.5
	anaconda			

Not all programs in the notes run on all systems, with all combinations of Python versions.

Here are some timing comparisons for one program (c2701) on three different hardware platforms, on 2 operating systems, and with 3 Python versions on the Windows OS.

Operating system	Windows			Linux openSuSe 15 Standard
	10 cygwin	10 anaconda	10 Visual Studio 2017	
Dell 7100	11.63 14.68	14.31 15.56	12.94 16.00	14.27 14.00
Dell 435	15.01 18.54	20.41 12.50	23.59 15.05	14.33 12.23
Laptop	11.37 20.15	11.27 12.56	16.35 14.44	10.98 8.98

The top figure is for an array initialisation and the bottom for a summation.

Here are some figures taken from Ubuntu under Hyper-V.

Operating System	Hyper-V Ubuntu
Dell 435	12.39 14.09
Laptop	9.34 9.68

The Dell 7100 originally ran Windows Home and couldn't run Hyper-V manager.

The Dell 435 has been retired and the PC Specialist has died.

Here is a table summarising the current systems.

Operating system and hardware	Component	Dell 5820 desktop	Dell 5515 laptop	Dell 7100 desktop
RAM - GB		64	32	16
Windows				
	Anaconda	3.9.13 August 2022	3.9.12 April 2022	3.8.3 July 2020
	Intel	1	1	1
	Microsoft Visual Studio Python	2 17.3.5 3.9	2	2
Linux	Native	ubuntu	NA	openSuSe
	hyper-v	openSuSe 15.3 3 2.7.18		
	hyper-v	Redhat 9.1 3.9.13	Redhat 9.1 3.9.14	
	wsl - opensuse	3.10.8	3.10.9	
	wsl - ubuntu	3.8.10	3.10.6	

Notes

- 1 - Intel Part of the Intel AI Analytics Toolkit
- 2 - Microsoft Can be included and ran from Visual Studio.
- 3 - openSuSe `sudo zypper install base-python`
This installs the 2.7.x release.

1.14 Course Details

The course is organised as a mixture of lectures and practicals as the most effective way to learn a programming language is by using it. Practice is essential. Think about how you learn French, German, etc.

It is important for you to read the notes between the time tabled sessions and also try completing the examples and problems set.

1.15 Problems

You will need access to a system running Windows, Linux or Unix. You need access to either an IDE or simple command line access to the compiler.

‘Though this be madness, yet there is method in’t’

Shakespeare.

‘Plenty of practice’ he went on repeating, all the time that Alice was getting him on his feet again. ‘plenty of practice.’

The White Knight, Through the Looking Glass and What Alice Found There, Lewis Carroll.

2 An Introduction to Python

In this chapter we will look at some simple program examples, illustrating some of the syntax of Python programs. Python is an interpreted language so most programs can be typed in whilst running the interpreter.

2.1 Example 1 - Hello World

Here is an example of using the interpreter under Windows.

```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59)
[MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more in-
formation.
>>>
```

Here is the program.

```
print("Hello world")
```

You just type the program in at the interpreter prompt and press the [return] key.

```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59)
[MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more in-
formation.
>>> print("Hello world")
Hello world
>>>
```

`print()` is one of the built in functions in Python. In this case we are using `print` to print out some text in quotes "Hello world" to the screen.

You can also create Python program files and invoke Python from the command line. Here is an example of doing that.

```
$ python3 c0201.py
Hello world
```

Both methods achieve the same result.

2.2 Example 2 - Simple text I/O using Python style strings

This example uses the `input()` function in Python 3. Here is the program.

```
line = input(" Type in some text ")
print(line)
```

Here is an example of running this program.

```
$ python3 c0202.py
Type in some text My name is Ian
My name is Ian
```

The `input()` function returns a string which is assigned to the variable `line`. We then use the `print()` function to print out the value of the string variable `line` to the screen.

2.3 Example 3 - Simple numeric i/o

This example reads in 3 numbers and sums and averages them.

Here is the source.

```
x1 = float(input(" Type in the first number "))
x2 = float(input(" Type in the second number "))
x3 = float(input(" Type in the third number "))
sum = x1+x2+x3
average=sum/3.0
print(" Sum =      " , sum)
print(" Average = " , average)
```

Here is a sample run.

```
$ python3 c0203.py
Type in the first number 1
Type in the second number 2
Type in the third number 3
Sum =      6.0
Average =  2.0
```

In this example we read one number at a time. Interaction with the user is as strings. We then extract the number from what the user has type in and assign it to the variable on the left hand side of the `=`. We cast from a string type to a float type. Real numbers in Python are real (have a decimal point), which generally maps on to the IEEE double data type. We then calculate the sum and average and print out the results with some explanatory text.

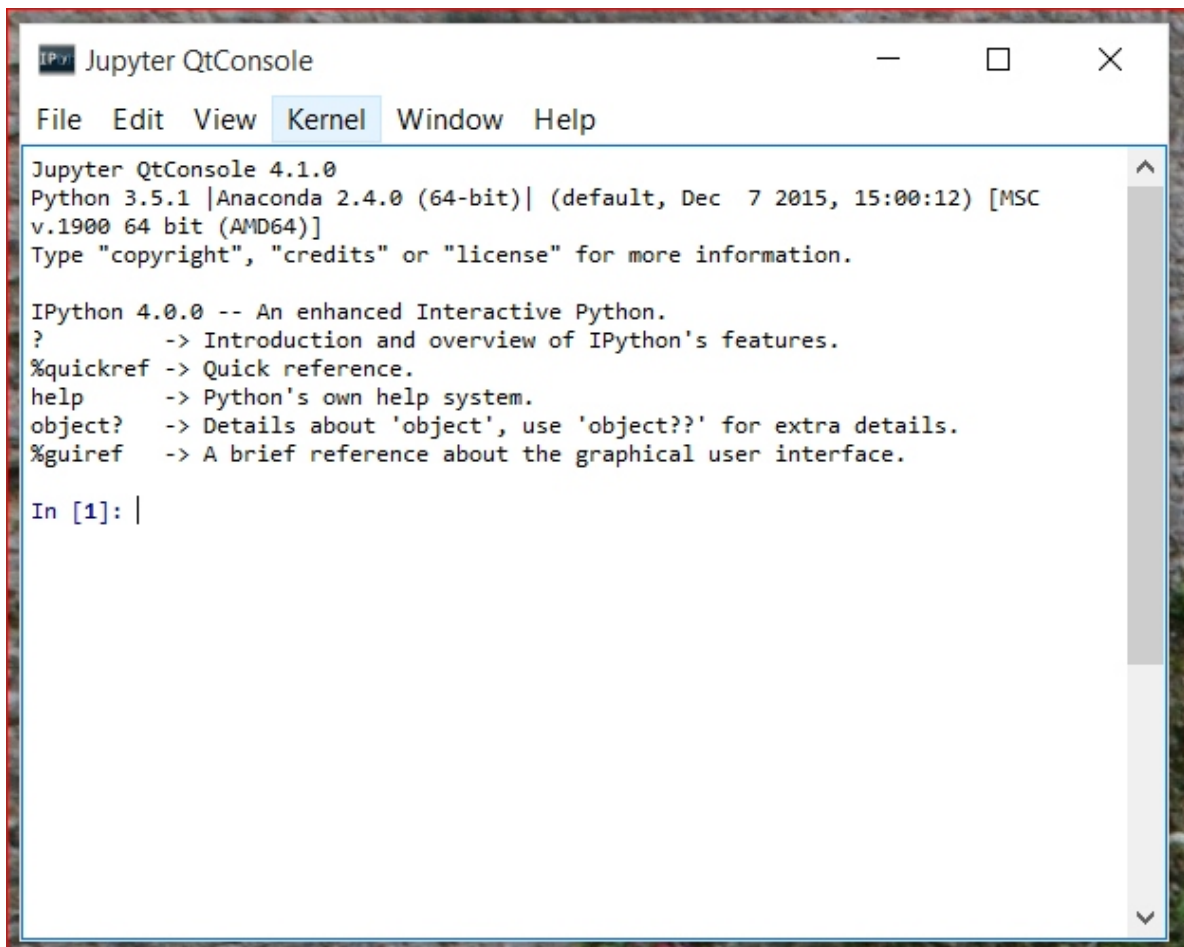
2.4 Running the examples using jupyter qtconsole

Here is an example of using the jupyter qtconsole on Windows. This is using a complete anaconda install on Windows.

Typing

```
jupyter qtconsole
```

from a Windows command prompt brings up the following Window.

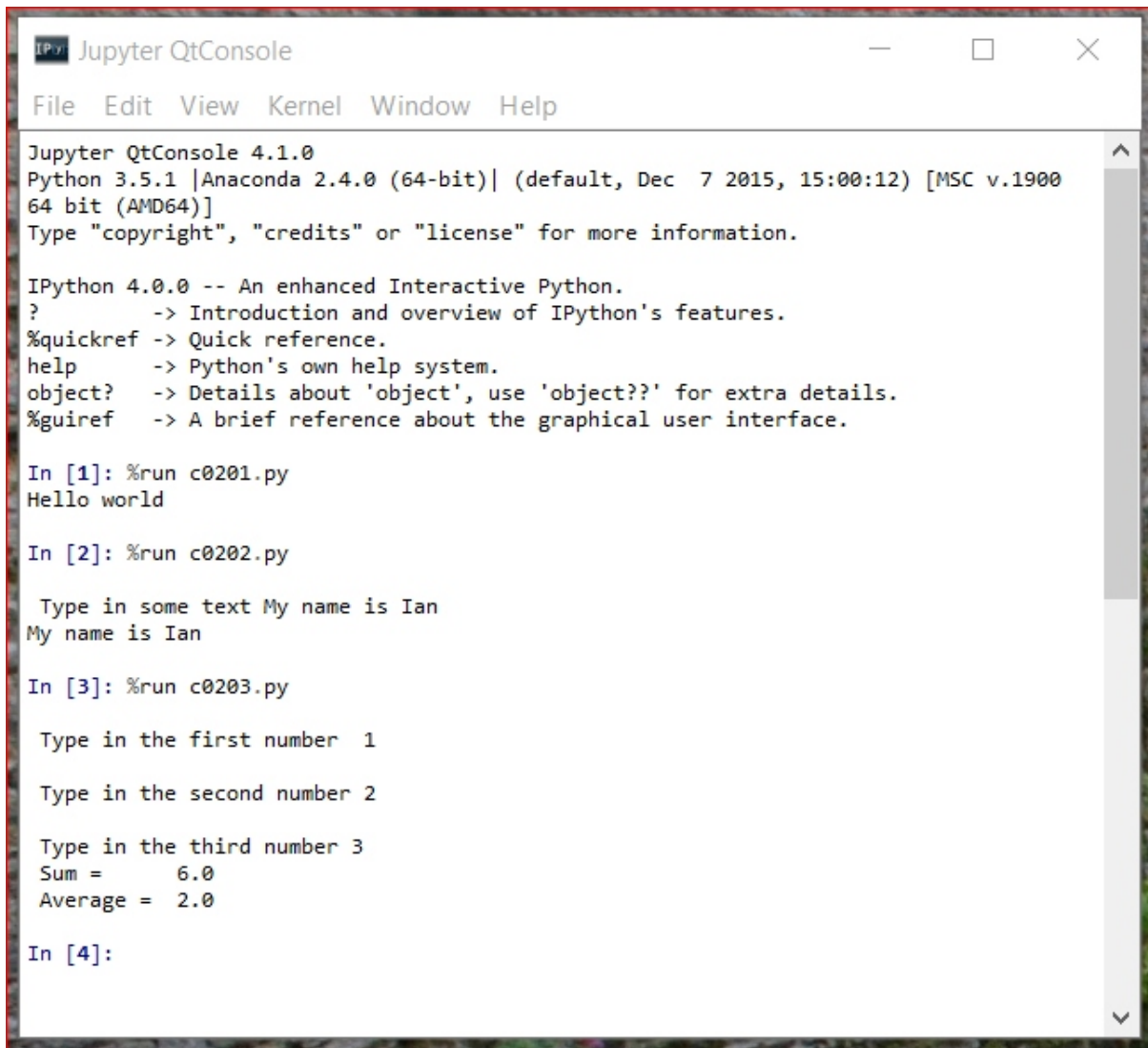


```
Jupyter QtConsole 4.1.0
Python 3.5.1 |Anaconda 2.4.0 (64-bit)| (default, Dec 7 2015, 15:00:12) [MSC
v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
%gui       -> A brief reference about the graphical user interface.

In [1]: |
```

Here is the Window after running the first three examples.



```
Jupyter QtConsole
File Edit View Kernel Window Help
Jupyter QtConsole 4.1.0
Python 3.5.1 |Anaconda 2.4.0 (64-bit)| (default, Dec 7 2015, 15:00:12) [MSC v.1900
64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
%gui       -> A brief reference about the graphical user interface.

In [1]: %run c0201.py
Hello world

In [2]: %run c0202.py

Type in some text My name is Ian
My name is Ian

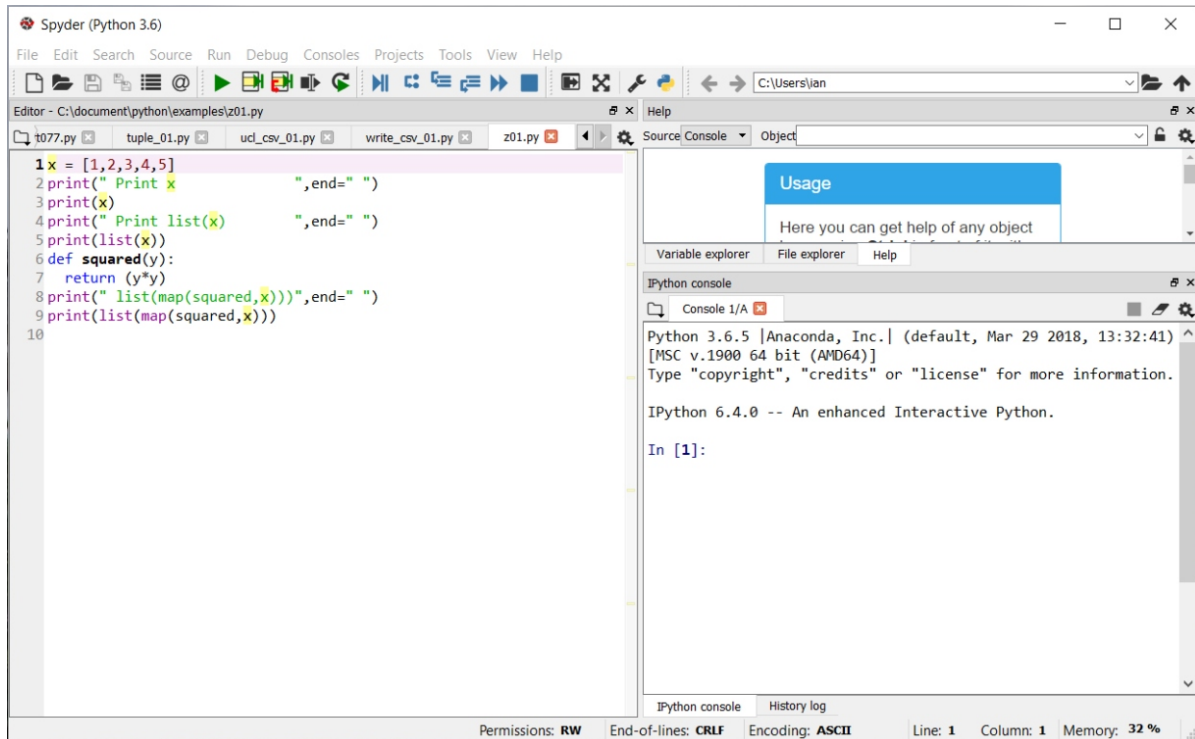
In [3]: %run c0203.py

Type in the first number 1
Type in the second number 2
Type in the third number 3
Sum =      6.0
Average =  2.0

In [4]:
```

2.5 Using spyder

An anaconda 3 install will also make spyder available. Goto to the start menu and Anaconda3 and you will see an entry for spyder. Clicking on this will bring up a screen shot similar to the one below.



I organise my Python programming in a directory and subdirectory structure as shown below:

```
c:\document\python\books
c:\document\python\documentation
c:\document\python\examples
c:\document\python\notes
```

I set the working directory in Spyder to

```
c:\document\python\examples
```

An

```
ls *.py
```

in the console Window provides output similar to that shown below.

```
23/12/2015 15:53          610 thread_10.py
23/12/2015 14:41          871 thread_11.py
14/12/2015 10:55           37 tk01.py
14/12/2015 10:56          104 tk02.py
14/12/2015 10:57          111 tk02_1.py
14/12/2015 10:57          107 tk02_2.py
14/12/2015 10:58          422 tk03.py
26/09/2018 20:04          535 tk04.py
14/12/2015 14:30       1,155 tkdoc01.py
27/12/2015 13:41          350 tk_button.py
27/12/2015 13:20          292 tk_button_01.py
27/12/2015 13:52          354 tk_entry.py
```

```
25/11/2015 13:18          4,331 tt077.py
22/12/2015 14:08          468 tuple_01.py
27/01/2016 10:00          537 ucl_csv_01.py
27/01/2016 19:57          962 write_csv_01.py
12/01/2016 12:02          228 z01.py
          359 File(s)          169,594 bytes
          0 Dir(s) 187,465,453,568 bytes free
```

I can now compile and run the examples in this directory from the console window.

The `ls` command is a Unix command, but is available from the Spyder console. The `pwd` command from the Spyder console shows your current directory.

You can also cut and paste examples from the notes into the editor window in Spyder.

We will be using a variety of methods throughout the notes to compile and run Python programs.

2.6 Problems

Try these examples out. Run them by

- typing them into the interpreter

- creating files and running them using files using the interpreter

- using the `qtconsole`

- using Spyder

3 Python base types, operators and expressions

This chapter looks at the fundamental data types in Python, and a number of rules that apply to their effective use. Quite a lot of technical material is introduced in this chapter, so don't panic if it does all make sense at first reading. We will look at examples throughout the notes that will hopefully clarify things! The information is taken from the on line reference manual.

3.1 Built-in Types

This chapter describes the standard types that are built into the interpreter. The principal built-in types are

- numerics,
- boolean types
- Iterator Types
- Sequence Types — list, tuple, range
- Text Sequence Type — str
- Binary Sequence Types — bytes, bytearray, memoryview
- Set Types — set, frozenset
- Mapping Types — dict
- Context Manager Types
- sequences,
- mappings,
- classes,
- instances and
- exceptions.

Some collection classes are mutable. The methods that add, subtract, or rearrange their members in place, and don't return a specific item, never return the collection instance itself but None.

Some operations are supported by several object types; in particular, practically all objects can be compared, tested for truth value, and converted to a string (with the repr() function or the slightly different str() function). The latter function is implicitly used when an object is written by the print() function.

3.2 Python symbols

The language reference makes the following distinctions.

3.2.1 Operators

The following tokens are operators:

+	-	*	**		
/	//	%	@		
<<	>>	&		^	~
<	>	<=	>=	==	!=

3.2.2 Delimiters and other characters

The following tokens serve as delimiters in the grammar:

```
(          )          [          ]          {          }
,          :          .          ;          @          =          ->
+=         -=         *=         /=         //=         %=         @=
&=         |=         ^=         >>=        <<=         **=
```

The period can also occur in floating-point and imaginary literals. A sequence of three periods has a special meaning as an ellipsis literal. The second half of the list, the augmented assignment operators, serve lexically as delimiters, but also perform an operation.

The following printing ASCII characters have special meaning as part of other tokens or are otherwise significant to the lexical analyzer:

```
'          "          #          \
```

The following printing ASCII characters are not used in Python. Their occurrence outside string literals and comments is an unconditional error:

```
$      ?      `
```

The comment character is the # symbol.

The line continuation character is the \ symbol.

3.3 Numeric Types — int, float, complex

There are three distinct numeric types:

- integers,
- floating point numbers, and
- complex numbers.

In addition, Booleans are a subtype of integers. The arithmetic chapter has several references to material on numerical analysis and IEEE arithmetic.

Integers have unlimited precision.

Floating point numbers are usually implemented using double in C; information about the precision and internal representation of floating point numbers for the machine on which your program is running is available in `sys.float_info`.

Complex numbers have a real and imaginary part, which are each a floating point number. To extract these parts from a complex number `z`, use `z.real` and `z.imag`. (The standard library includes additional numeric types, `fractions` that hold rationals, and `decimal` that hold floating-point numbers with user-definable precision.)

Numbers are created by numeric literals or as the result of built-in functions and operators. Unadorned integer literals (including hex, octal and binary numbers) yield integers. Numeric literals containing a decimal point or an exponent sign yield floating point numbers. Appending 'j' or 'J' to a numeric literal yields an imaginary number (a complex number with a zero real part) which you can add to an integer or float to get a complex number with real and imaginary parts.

Python fully supports mixed arithmetic: when a binary arithmetic operator has operands of different numeric types, the operand with the “narrower” type is widened to that of the other, where integer is narrower than floating point, which is narrower than complex. Comparisons between numbers of mixed type use the same rule. [2] The constructors `int()`, `float()`, and `complex()` can be used to produce numbers of a specific type.

All numeric types (except complex) support the following operations, sorted by ascending priority (all numeric operations have a higher priority than comparison operations):

Operation	Result
$x + y$	sum of x and y
$x - y$	difference of x and y
$x * y$	product of x and y
x / y	quotient of x and y
$x // y$	floored quotient of x and y (1)
$x \% y$	remainder of x / y (2)
$-x$	x negated
$+x$	x unchanged
<code>abs(x)</code>	absolute value or magnitude of x <code>abs()</code>
<code>int(x)</code>	x converted to integer (3)(6) <code>int()</code>
<code>float(x)</code>	x converted to floating point (4)(6) <code>float()</code>
<code>complex(re, im)</code>	a complex number with real part re , imaginary part im . im defaults to zero. (6) <code>complex()</code>
<code>c.conjugate()</code>	conjugate of the complex number c
<code>divmod(x, y)</code>	the pair $(x // y, x \% y)$ (2) <code>divmod()</code>
<code>pow(x, y)</code>	x to the power y (5) <code>pow()</code>
$x ** y$	x to the power y (5)

3.4 Iterator Types

Python supports a concept of iteration over containers. This is implemented using two distinct methods; these are used to allow user-defined classes to support iteration. Sequences, described below in more detail, always support the iteration methods.

3.5 Sequence Types

There are three basic sequence types: lists, tuples, and range objects.

3.6 Text Sequence Type - `str`

Textual data in Python is handled with `str` objects, or strings. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

Single quotes: 'allows embedded "double" quotes'

Double quotes: "allows embedded 'single' quotes".

Triple quoted: """Three single quotes""", """"Three double quotes""""

Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.

3.7 Binary sequence types - bytes, bytearray, memoryview

The core built-in types for manipulating binary data are bytes and bytearray. They are supported by memoryview which uses the buffer protocol to access the memory of other binary objects without needing to make a copy.

The array module supports efficient storage of basic data types like 32-bit integers and IEEE754 double-precision floating values.

3.8 Set types - set, frozenset

A set object is an unordered collection of distinct hashtable objects.

3.9 Mapping types - dict

A mapping object maps hashtable values to arbitrary objects.

3.10 Context manager types

Python's with statement supports the concept of a runtime context defined by a context manager. This is implemented using a pair of methods that allow user-defined classes to define a runtime context that is entered before the statement body is executed and exited when the statement ends.

3.11 Other types

Python also supports

- modules - the only special operation is attribute access.

- classes

- functions - function objects are created by function definition.

- methods - functions called using the attribute notation

- code objects - Code objects are a low-level detail of the CPython implementation. Each one represents a chunk of executable code that hasn't yet been bound into a function.

- type objects represent the various object types.

- null object - returned by functions that don't return a value.

- ellipsis object - used by slicing

- NotImplemented object this object is returned from comparisons and binary operations when they are asked to operate on objects they don't support.

See the library reference manual for more information.

We will look at some of these in the chapters that follow.

3.12 Problems

There are none in this chapter.

Taking Three as the subject to reason about —

A convenient number to state —

We add Seven, and Ten, and then multiply out

By One Thousand diminished by Eight.

The result we proceed to divide, as you see,

By Nine Hundred and Ninety and Two:

Then subtract Seventeen, and the answer must be

Exactly and perfectly true.

Lewis Carroll, *The Hunting of the Snark*

Round numbers are always false.

Samuel Johnson.

4 Arithmetic

This chapter looks at arithmetic in Python, and we will have a look at several examples in this chapter illustrating arithmetic in Python. We covered the operators in the previous chapter.

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

We will use some of these functions in examples in this chapter.

4.1 Example 1 - assignment and division

Here is the first example.

```
a = 1.5
b = 2.0
c = a/b
print(" a = " ,a)
print(" b = " ,b)
print(" c = " ,c)
```

Here is some sample output.

```
$ python3 c0401.py
a = 1.5
b = 2.0
c = 0.75
```

The result is as expected from our experience with mathematics.

4.2 Example 2 - division with integers

Here is the example.

```
i = 5
j = 2
k = 4
l = i/j*k
print(" i = ",i)
print(" j = ",j)
print(" k = ",k)
print(" l = ",l)
print(" type i = ",type( i))
print(" type j = ",type( j))
print(" type k = ",type( k))
print(" type l = ",type( l))
```

Here is the output.

```
$ python3 c0402.py
i = 5
j = 2
k = 4
l = 10.0
type i = <class 'int'>
type j = <class 'int'>
type k = <class 'int'>
type l = <class 'float'>
```

In this case even though i and j are integers the expression $i/j*k$ evaluates as 10.0 This is at variance with Fortran, C and C++.

4.3 Example 3 - time taken to reach the earth from the Sun.

Here is the program.

```
light_year    = 9.46 * 10**12
light_minute = light_year/(365.25*24.0*60.0)
light_second  = light_minute/60.0
distance      = 150.0 * 10.0**6
elapse        = distance/light_minute
minute        = int(elapse)
second        = int((elapse-minute)*60)
print(" Light takes " , minute , " minutes and " ,second ,
"seconds")
elapse        = distance/light_second
print(" or " , elapse , "seconds")
```

and here is the output.

```
$ python3 c0403.py
Light takes 8 minutes and 20 seconds
or 500.384778012685 seconds
```

and the output is as expected. Note that Python has an exponentiation operator, unlike the C family of languages (C, C++, Java, C#).

4.4 Example 4 - converting from Fahrenheit to centigrade.

Here is a simple program to convert from Fahrenheit to centigrade.

```
f = 75.0
c = 5/9*(f-32)
print(f , " Fahrenheit = ",c, " centigrade")
```

Here is the output.

```
$ python3 c0404.py
75.0 Fahrenheit = 23.888888888888889 centigrade
```

4.5 Example 5 - converting from Centigrade to Fahrenheit.

Here is a program to convert from centigrade to Fahrenheit.

```
c = 25
f = 32 + 9/5 * c
print(c , " centigrade = " , f , " Fahrenheit")
```

Here is the output.

```
$ python3 c0405.py
25 centigrade = 77.0 Fahrenheit
```

4.6 Example 6 - numbers getting too large - overflow

This program illustrates overflow in Python.

```
x = 10.0
for i in range(1,320):
    print(x)
    x=x*10
```

Here is the output. Lines have deleted to reduce the page count.

```
10.0
100.0
1000.0
10000.0
100000.0
1000000.0
...
1e+16
1e+17
...
1e+304
1e+305
9.999999999999999e+305
9.999999999999999e+306
9.999999999999998e+307
inf
inf
```

4.7 Example 7 - numbers getting too small - underflow

Here is the program.

```
x = 10.0
for i in range(1,330):
    print(x)
    x=x/10
```

Here is the output. Again lines have deleted to reduce the page count.

```
10.0
1.0
0.1
0.01
0.001
0.0001
1e-05
1.000000000000000002e-06
1.000000000000000002e-07
1.000000000000000002e-08
...
1.0000000000000000021e-305
1.0000000000000000021e-306
1.000000000000000002e-307
1.000000000000000002e-308
1e-309
1e-310
...
1e-320
1e-321
1e-322
1e-323
0.0
0.0
0.0
0.0
```

4.8 Example 8 - subtraction of two similar values

Here is the program.

```
x = 1.00000002
y = 1.00000001
z = x-y
print(" {0:2.18f} ".format(x))
print(" {0:2.18f} ".format(y))
print(" {0:2.18f} ".format(z))
```

Here is the output.

```
$ python3 c0408.py
1.000000020000000100
1.000000009999999939
0.000000010000000161
```

4.9 Example 9 - summation

Here is the program.


```

x1 = 1.0
x2 = 0.1
x3 = 0.01
x4 = 0.001
x5 = 0.0001
for i in range(1,10):
    x1=x1+1.0
for i in range(1,10):
    x2=x2+0.1
for i in range(1,10):
    x3=x3+0.01
for i in range(1,10):
    x4=x4+0.001
for i in range(1,10):
    x5=x5+0.0001
print(" {0:2.18f} ".format(x1))
print(" {0:2.18f} ".format(x2))
print(" {0:2.18f} ".format(x3))
print(" {0:2.18f} ".format(x4))
print(" {0:2.18f} ".format(x5))

```

Here is the output.

```

$ python3 c0409.py
10.00000000000000000000000000000000
0.9999999999999999889
0.099999999999999992
0.01000000000000000002
0.00100000000000000000

```

4.10 Absolute and relative errors

In mathematics if p is an approximation to p then the relative error is given by

$$\frac{|p - p|}{|p|}$$

and the absolute error is given by

$$|p - p|$$

We will use this information in some of the problems.

4.11 Problems

1. The period of a pendulum is given by

$$2 \sqrt{\text{length}/9.81}$$

Write a Python program to evaluate this for a length of 10m.

2. Calculate the relative and absolute error for the difference of the two variables in example 8. Subtracting 1.0000001 from 1.00000002 is 0.00000001.

2. In mathematics the following are all equal.

$$\begin{aligned}
 &x^2 - y^2 \\
 &(x + y)(x - y) \\
 &(x - y)(x + y)
 \end{aligned}$$

Test out the equality of these expressions in Python. Try the following values

$$x = 1.002, y = 1.001$$

$$x = 1.0002, y = 1.0001$$

$$x = 1.00002, y = 1.00001$$

3. For

$$p = 0.4e-4$$

and

$$p_{\text{approx}} = 0.41e-4$$

calculate the relative and absolute errors. Repeat for two more values of p and p_{approx} , multiplying by $1.0e5$ each time.

4. A geostationary satellite will be in orbit approximately 35,870 km from the earth. Calculate the round trip time to the satellite and back. This will be the minimum delay for satellite based broadband.

You can just change the distance in the example earlier in this chapter.

5. The Moon is approximately 384,400 km from the earth. What is the time delay here?

6. The following table gives the distances in 10^9 m from the Sun to the planets in the Solar system.

Mercury	57.9	Venus	108.2
Earth	149.6	Mars	227.9
Jupiter	778.3	Saturn	1427.0
Uranus	2869.6	Neptune	4496.6
Pluto	5900.0		

Use this information to find the greatest and least time taken to send a message from the Earth to the other planets. Assume that all orbits are in the same plane, and circular - if it was good enough for Copernicus it is good enough for us.

4.12 Bibliography

Some understanding of numerical analysis is essential for successful use of a programming language. As Froberg says ‘numerical analysis is a science – computation is an art.’ The following are some of the more accessible books available.

Burden R.L., Douglas Faires J., Numerical Analysis, Brooks/Cole, 2001.

Source code on a cd is available in C, Fortran, Maple, Mathematica, Matlab and Pascal. Good modern text.

Froberg C.E., Introduction to Numerical Analysis, Addison Wesley, 1969.

The short chapter on numerical computation is well worth a read, and it covers some of the problems of conversion between number bases, and some of the errors that are introduced when we compute numerically. The Samuel Johnson quote owes its inclusion to Froberg!

<http://grouper.ieee.org/groups/754/>

The working group website.

IEEE, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, Institute of Electrical and Electronic Engineers Inc.

The formal definition of IEEE 754.

Knuth D., *Seminumerical Algorithms*, Addison Wesley, 1969.

A more thorough and mathematical coverage than Wakerly. The chapter on positional number systems provides a very comprehensive historical coverage of the subject. As Knuth points out the floating point representation for numbers is very old, and is first documented around 1750 B.C. by Babylonian mathematicians. Very interesting and worthwhile reading.

Sun, *Numerical Computation Guide*, SunPro, 1993.

Very good coverage of the numeric formats for IEEE Standard 754 for Binary Floating Point Arithmetic. All SunPro compiler products support the features of the IEEE 754 standard .

Wakerly J.F., *Microcomputer Architecture and Programming*, Wiley, 1981.

The chapter on number systems and arithmetic is surprisingly easy. There is a coverage of positional number systems, octal and hexadecimal number system conversions, addition and subtraction of non-decimal numbers, representation of negative numbers, two's complement addition and subtraction, one's complement addition and subtraction, binary multiplication, binary division, bcd or binary coded decimal representation and fixed and floating point representations. There is also coverage of a number of specific hardware platforms, including DEC PDP-11, Motorola 68000, Zilog Z8000, TI 9900, Motorola 6809 and Intel 8086. A little old but quite interesting nevertheless.

Wikipedia. There is a Wikipedia entry for IEEE 854.

5 Arrays using the array module

Most programming languages have to include language features that provide ways of manipulating tabular data. This is done most commonly by using arrays. Here is a basic description of the array in Python.

<http://docs.python.org/library/array.html>

This module defines an object type which can compactly represent an array of basic values: characters, integers, floating point numbers. Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained. The type is specified at object creation time by using a type code, which is a single character. The information below is taken from 3.5.1 documentation. The following type codes are defined:

Type code	C Type	Python Type	Minimum size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2 (1))
'h'	signed short	int	2
'H'	unsigned short	int	2
'I'	signed int	int	2
'I'	unsigned int	long	2
'l'	signed long	int	4
'L'	unsigned long	long	4
'q'	signed long long	int	8(2)
'Q'	unsigned long long	int	8(2)
'f'	float	float	4
'd'	double	float	8

Note:

1. The 'u' typecode corresponds to Python's unicode character. On narrow Unicode builds this is 2-bytes, on wide builds this is 4-bytes. 'u' will be removed together with the rest of the Py_UNICODE API.

Deprecated since version 3.3, will be removed in version 4.0.

2. The 'q' and 'Q' type codes are available only if the platform C compiler used to build Python supports C long long, or, on Windows, __int64

5.1 Array methods

The module defines the following type:

```
class array.array(typecode[, initializer])
```

A new array whose items are restricted by typecode, and initialized from the optional initializer value, which must be a list, string, or iterable over elements of the appropriate type. Changed in version 2.4: Formerly, only lists or strings were accepted. If given a list or string, the initializer is passed to the new array's `fromlist()`, `fromstring()`, or `fromunicode()` method (see below) to add initial items to the array. Otherwise, the iterable initializer is passed to the `extend()` method.

`array.ArrayType`

Obsolete alias for `array`. Array objects support the ordinary sequence operations of indexing, slicing, concatenation, and multiplication. When using slice assignment, the assigned value must be an array object with the same type code; in all other cases, `TypeError` is raised. Array objects also implement the buffer interface, and may be used wherever buffer objects are supported. The following data items and methods are also supported:

`array.typecode`

The typecode character used to create the array.

`array.itemsize`

The length in bytes of one array item in the internal representation.

`array.append(x)`

Append a new item with value `x` to the end of the array.

`array.buffer_info()`

Return a tuple (address, length) giving the current memory address and the length in elements of the buffer used to hold array's contents. The size of the memory buffer in bytes can be computed as `array.buffer_info()[1] * array.itemsize`. This is occasionally useful when working with low-level (and inherently unsafe) I/O interfaces that require memory addresses, such as certain `ioctl()` operations. The returned numbers are valid as long as the array exists and no length-changing operations are applied to it.

Note: When using array objects from code written in C or C++ (the only way to effectively make use of this information), it makes more sense to use the buffer interface supported by array objects. This method is maintained for backward compatibility and should be avoided in new code. The buffer interface is documented in [Buffers and Memoryview Objects](#).

`array.byteswap()`

“Byteswap” all items of the array. This is only supported for values which are 1, 2, 4, or 8 bytes in size; for other types of values, `RuntimeError` is raised. It is useful when reading data from a file written on a machine with a different byte order.

`array.count(x)`

Return the number of occurrences of `x` in the array.

`array.extend(iterable)`

Append items from `iterable` to the end of the array. If `iterable` is another array, it must have exactly the same type code; if not, `TypeError` will be raised. If `iterable` is not an array, it must be iterable and its elements must be the right type to be appended to the array. Changed in version 2.4: Formerly, the argument could only be another array.

`array.fromfile(f, n)`

Read *n* items (as machine values) from the file object *f* and append them to the end of the array. If less than *n* items are available, `EOFError` is raised, but the items that were available are still inserted into the array. *f* must be a real built-in file object; something else with a `read()` method won't do.

`array.fromlist(list)`

Append items from the list. This is equivalent to `for x in list: a.append(x)` except that if there is a type error, the array is unchanged.

`array.fromstring(s)`

Appends items from the string, interpreting the string as an array of machine values (as if it had been read from a file using the `fromfile()` method).

`array.fromunicode(s)`

Extends this array with data from the given unicode string. The array must be a type `'u'` array; otherwise a `ValueError` is raised. Use `array.fromstring(unicodestring.encode(enc))` to append Unicode data to an array of some other type.

`array.index(x)`

Return the smallest *i* such that *i* is the index of the first occurrence of *x* in the array.

`array.insert(i, x)`

Insert a new item with value *x* in the array before position *i*. Negative values are treated as being relative to the end of the array.

`array.pop([i])`

Removes the item with the index *i* from the array and returns it. The optional argument defaults to `-1`, so that by default the last item is removed and returned.

`array.read(f, n)`

Deprecated since version 1.5.1: Use the `fromfile()` method.

Read *n* items (as machine values) from the file object *f* and append them to the end of the array. If less than *n* items are available, `EOFError` is raised, but the items that were available are still inserted into the array. *f* must be a real built-in file object; something else with a `read()` method won't do.

`array.remove(x)`

Remove the first occurrence of *x* from the array.

`array.reverse()`

Reverse the order of the items in the array.

`array.tofile(f)`

Write all items (as machine values) to the file object *f*.

`array.tolist()`

Convert the array to an ordinary list with the same items.

`array.tostring()`

Convert the array to an array of machine values and return the string representation (the same sequence of bytes that would be written to a file by the `tofile()` method.)

`array.tounicode()`

Convert the array to a unicode string. The array must be a type 'u' array; otherwise a ValueError is raised. Use `array.tostring().decode(enc)` to obtain a unicode string from an array of some other type.

`array.write(f)`

Deprecated since version 1.5.1: Use the `tofile()` method. Write all items (as machine values) to the file object `f`.

We will only look at a small number of examples in this chapter.

5.2 Array size known at compile time

5.2.1 Example 1 - array and conventional for loop syntax

```
import array
n=12
month=0
sum=0.0
average=0.0
rainfall = array.array('d' , [ 3.1 , 2.0 , 2.4 , 2.1 , 2.2
, 2.2 , 1.8 , 2.2 , 2.7 , 2.9 , 3.1 , 3.1 ] )
for month in range (0,n):
    sum = sum + rainfall[month]
average = sum/n
print(" Sum          = ",sum)
print(" Average = " , average)
```

The first thing we need to do is make the array module available, which we do with the `import` statement.

The next thing we do is set the size of the array. In the next example we will calculate this using one of the array methods.

We then declare and initialise one integer variable (`month`) and two float variables, `sum` and `average`.

We then declare the array `rainfall` and create it using an array constructor and provide initial values for the array.

A summation

$$\sum_1^n x$$

is often written in a programming language as a loop over an array. In Python we start at zero.

We then calculate the sum, looping over each element of the array, using Python's `for` loop construct.

We will look in more detail at the `for` statement in a later chapter. Here is some data taken from the monthly rainfall figures for London.

Month	Month as integer	Index	Rainfall value
January	1	0	3.1
February	2	1	2.0
March	3	2	2.4
April	4	3	2.1

Month	Month as integer	Index	Rainfall value
May	5	4	2.2
June	6	5	2.2
July	7	6	1.8
August	8	7	2.2
September	9	8	2.7
October	10	9	2.9
November	11	10	3.1
December	12	11	3.1

The measurements are in inches.

5.2.2 Example 2 - using the len function to determine the size of array

This example is a simple variant of the first.

```
import array
month=0
sum=0.0
average=0.0
rainfall = array.array('d' , [ 3.1 , 2.0 , 2.4 , 2.1 , 2.2
, 2.2 , 1.8 , 2.2 , 2.7 , 2.9 , 3.1 , 3.1 ] )
n = len(rainfall)
for month in range (0,n):
    sum = sum + rainfall[month]
average = sum/n
print(" Sum      = ",sum)
print(" Average = " , average)
```

We use the len function in this example.

5.3 Array size known at run time

In this example we input the array size at run time.

5.3.1 Example 3 - reading in the array size

Here is the example

```
import array
i = 0
sum = 0
n = int(input(" Type in the size of the array: "))
temp = [0]*n
x = array.array('L',temp)
for i in range (0,n):
    x[i]=i
    sum=sum+x[i]
print (" Sum of array elements is: ",sum)
```

The second argument to array.array must be iterable. We have used a temporary iterable object temp to set the array size dynamically at run time.

5.4 Summary

The array data type in this chapter is fine for one dimensional arrays. For mathematical multidimensional arrays Python provides NumPy. This is covered in the next chapter.

5.5 Problems

1. Compile and run the examples.
2. Convert the rainfall measurements to mm. What is the sum and average?
3. Visit

<http://www.metoffice.gov.uk/public/weather/climate-historic/#?tab=climateHistoric>

The following is an alphabetical list of sites.

aberporth
armagh
ballypatrick
bradford
braemar
camborne
cambridge
cardiff
chivenor
cwmystwyth
dunstaffnage
durham
eastbourne
eskdalemuir
heathrow
hurn
lerwick
leuchars
lowestoft
manston
nairn
newtonrigg
oxford
paisley
ringway
rossonwey
shawbury
sheffield

southampton
stornoway
suttonbonington
tiree
valley
waddington
whitby
wickairport
yeovilton

Chose a site and a year and replace the rainfall measurements in the rainfall example with your data.

Is your site and year wetter or dryer than London?

6 Arrays using the Numpy module

Here is the Numpy site.

<http://www.numpy.org/>

and here is their description about Numpy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Numpy is licensed under the BSD License, enabling reuse with few restrictions.

The following has been taken from the wikipedia entry.

Introduction

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open source and has many contributors.

Traits

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy address the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring (re)writing some code, mostly inner loops using NumPy. Thus any algorithm that can be expressed primarily as operations on arrays and matrices can run almost as quickly as the equivalent C code.[1]

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both

allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink; whereas NumPy is intrinsically integrated with Python, a more modern, complete, and open source programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

The ndarray data structure

The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory.[2] In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++. Cython and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.[2]

Limitations

NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.[3]

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include `scipy.weave`, `numexpr`[4] and `Numba`. [5] Cython is a static-compiling alternative to these.

History

The Python programming language was not initially designed for numerical computing, but attracted the attention

of the scientific/engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer/maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.[6] An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric,[6] also variously called Numerical Python extensions or NumPy.[7][8] Hugunin, a graduate student at MIT,[8]:10 joined CNRI to work on JPython in 1997[6] leading Paul Dubois of LLNL to take over as maintainer.[8]:10 Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.[8]:10

A new package called Numarray was written as a more flexible replacement for Numeric.[2] Like Numeric, it is now deprecated.[9] Numarray had faster operations for large arrays, but was slower than Numeric on small ones,[citation needed] so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.[10]

There was a desire to get Numeric into the Python standard library, but Guido van Rossum (the author of Python) was quite clear that the code was not maintainable in its state then.[when?][citation needed]

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.[2] This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy.

The release version 1.5.1 of NumPy is compatible with Python versions 2.4–2.7 and 3.1–3.2. Support for Python 3 was added in 1.5.0.[11] In 2011, PyPy started development on an implementation of the numpy API for PyPy.[12] It is not yet fully compatible with NumPy.[13]

References

1. "SciPy PerformancePython". Retrieved 2006-06-25.
2. Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux (2011). "The NumPy array: a structure for efficient numerical computation". *Computing in Science and Engineering (IEEE)*.

3. "Blaze 0.4.1 Documentation". Blaze. Retrieved 8 March 2014.
4. Francesc Alted. "numexpr". Retrieved 8 March 2014.
5. "Numba". Retrieved 8 March 2014.
6. Millman, K. Jarrod; Aivazis, Michael (2011). "Python for Scientists and Engineers". *Computing in Science and Engineering* 13 (2): 9–12.
7. Travis Oliphant (2007). "Python for Scientific Computing" (PDF). *Computing in Science and Engineering*.
8. David Ascher; Paul F. Dubois; Konrad Hinsen; Jim Hugunin; Travis Oliphant (1999). "Numerical Python" (PDF).
9. "Numarray Homepage". Retrieved 2006-06-24.
10. "NumPy Sourceforge Files". Retrieved 2008-03-24.
11. "NumPy 1.5.0 Release Notes". Retrieved 2011-04-29.
12. "PyPy Status Blog: Numpy funding and status update". Retrieved 2011-12-22.
13. "NumPyPy Status". Retrieved 2013-10-14.

We next have a look at the main numpy site.

6.1 Documentation

Here is an extract from the numpy user guide.

This guide is intended as an introductory overview of NumPy and explains how to install and make use of the most important features of NumPy. For detailed reference documentation of the functions and classes contained in the package, see the NumPy Reference.

Warning:

This "User Guide" is still a work in progress; some of the material is not organized, and several aspects of NumPy are not yet covered sufficient detail. We are an open source community continually working to improve the documentation and eagerly encourage interested parties to contribute. For information on how to do so, please visit the NumPy doc wiki.

More documentation for NumPy can be found on the numpy.org website.

Thanks!

Introduction ?What is NumPy?

Building and installing NumPy

How to find documentation

Numpy basics ?Data types

Array creation

I/O with Numpy

Indexing

Broadcasting

- Byte-swapping
- Structured arrays
- Subclassing ndarray
- Performance
- Miscellaneous ?IEEE 754 Floating Point Special Values
- How numpy handles numerical exceptions
- Examples
- Interfacing to C
- Interfacing to Fortran:
- Interfacing to C++:
- Methods vs. Functions
- Using Numpy C-API ?How to extend NumPy
- Using Python as glue
- Writing your own ufunc
- Beyond the Basics

We recommend using this site when working with numpy arrays.

6.2 Creating arrays

There are 5 general mechanisms for creating arrays:

- Conversion from other Python structures (e.g., lists, tuples)
- Intrinsic numpy array array creation objects (e.g., `arange`, `ones`, `zeros`, etc.)
- Reading arrays from disk, either from standard or custom formats
- Creating arrays from raw bytes through the use of strings or buffers
- Use of special library functions (e.g., `random`)

Numpy supports a much greater variety of numerical types than Python does.

Data type	Description
<code>bool_</code>	Boolean (True or False) stored as a byte
<code>int_</code>	Default integer type (same as C long; normally either <code>int64</code> or <code>int32</code>)
<code>intc</code>	Identical to C int (normally <code>int32</code> or <code>int64</code>)
<code>intp</code>	Integer used for indexing (same as C <code>ssize_t</code> ; normally either <code>int32</code> or <code>int64</code>)
<code>int8</code>	Byte (-128 to 127)
<code>int16</code>	Integer (-32768 to 32767)
<code>int32</code>	Integer (-2147483648 to 2147483647)
<code>int64</code>	Integer (-9223372036854775808 to 9223372036854775807)
<code>uint8</code>	Unsigned integer (0 to 255)
<code>uint16</code>	Unsigned integer (0 to 65535)
<code>uint32</code>	Unsigned integer (0 to 4294967295)
<code>uint64</code>	Unsigned integer (0 to 18446744073709551615)
<code>float_</code>	Shorthand for <code>float64</code> .
<code>float16</code>	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa

float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

We will use some of these methods in the examples below.

6.3 Simple 1 and 2 d array examples

The first set of examples are a simple rewrite of the examples in the previous chapter. We then move on to two dimensional examples.

6.3.1 Example 1 - simple rainfall example

```
import numpy as np
n=12
month=0
sum=0.0
average=0.0
rainfall = np.array([ 3.1 , 2.0 , 2.4 , 2.1 , 2.2 , 2.2 ,
1.8 , 2.2 , 2.7 , 2.9 , 3.1 , 3.1 ] )
for month in range (0,n):
    sum = sum + rainfall[month]
average = sum/n
print(" Sum      = ",sum)
print(" Average = " , average)
```

In this example we store the array size in the variable n. We then loop over the array to calculate the sum.

6.3.2 Example 2 - variant of one using len intrinsic function

```
import numpy as np
month=0
sum=0.0
average=0.0
rainfall = np.array([ 3.1 , 2.0 , 2.4 , 2.1 , 2.2 , 2.2 ,
1.8 , 2.2 , 2.7 , 2.9 , 3.1 , 3.1 ] )
n = len(rainfall)
for month in range (0,n):
    sum = sum + rainfall[month]
average = sum/n
print(" Sum      = ",sum)
print(" Average = " , average)
```

In this example we use the len() method to calculate the array size.

6.3.3 Example 3 - setting the size at run time

```
import numpy as np
i = 0
sum = 0
```



```
n = int(input(" Type in the size of the array: "))
x = np.empty([n],dtype=np.int32)
for i in range (0,n):
    x[i]=i
    sum=sum+x[i]
print (" Sum of array elements is: ",sum)
```

In this example we read in the array size at run time, and use the `numpy.empty` method to create an array of 32 bit integers.

6.3.4 Example 4 - two d array using `numpy.zeros` method

```
import numpy as np
nr = 3
nc = 3
x = np.zeros([nr,nc] , dtype=np.int32)
print("\n 3 by 3 matrix, using numpy.zeros() method \n\n")
print(x)
```

Here we zero use the `numpy.zeros()` method to create the array and initialise to zero.

6.3.5 Example 5 - two d array using `numpy.array()` method

```
import numpy as np
nr = 3
nc = 3
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
print("\n 3 by 3 matrix, initialisation using numpy.array()
method\n\n")
print(x)
```

This example uses the `numpy.array()` method.

6.3.6 Example 6 - two d array and the `numpy.sum()` method

In this example we illustrate the use of the `numpy.sum` method with a two d array. We also show whole array assignment in Python. We then compare with a C++ version to illustrate the power of Python.

```
import numpy as np
nr = 3
nc = 3
rsum = np.zeros([nr],dtype=np.int32)
csum = np.zeros([nc],dtype=np.int32)
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
csum=np.sum(x,axis=0)
rsum=np.sum(x,axis=1)
print("\n 3 * 3 matrix \n\n",x)
print("\n Row sum      = ",rsum)
print(" Column sum    = ",csum)
```

Here is the output.

```
3 * 3 matrix

[[1 2 3]
```

```
[4 5 6]
[7 8 9]]
```

```
Row sum      = [ 6 15 24]
Column sum   = [12 15 18]
```

Here is the C++ implementation.

```
#include <iostream>

using namespace std;

int main()
{
    const int nrows=3;
    const int ncols=3;

    int r;
    int c;

    int x[nrows][ncols] = {{1,2,3},
                           {4,5,6},
                           {7,8,9}};

    int rsum[nrows] = {0,0,0};
    int csum[ncols] = {0,0,0};

    cout << " Row sum      " ;

    for (r=0;r<nrows;r++)
    {
        for (c=0;c<ncols;c++)
        {
            rsum[r]=rsum[r]+x[r][c];
        }
        cout << rsum[r] << " " ;
    }

    cout << endl;
    cout << " Column sum " ;

    for (c=0;c<ncols;c++)
    {
        for (r=0;r<nrows;r++)
        {
            csum[c]=csum[c]+x[r][c];
        }
        cout << csum[c] << " ";
    }

    cout << endl;
```

```

cout << "\n 3 * 3 matrix plus row and column sums\n\n" ;

for (r=0;r<nrows;r++)
{
    for (c=0;c<ncols;c++)
    {
        cout.width(2);
        cout << x[r][c] << " ";
    }
    cout << rsum[r] << endl;
}

for (c=0;c<ncols;c++)
    cout << csum[c] << " ";
cout << endl;

return(0);
}

```

Here is the C++ output.

```

c:\document\python\examples>c0606
Row sum      6 15 24
Column sum 12 15 18

3 * 3 matrix plus row and column sums

1  2  3  6
4  5  6 15
7  8  9 24
12 15 18

```

The Python version is obviously a lot simpler.

We have a `numpy.sum()` method which enables us to do the summation without having to use loops.

We also have whole array assignment as the `numpy.sum()` method can return an array.

One of the problems is to modify the Python version to produce the same output as the C++ version.

Here are two variants of this showing breaking a line at a white space point and using explicit continuation symbols.

White space.

```

import numpy as np
nr = 3
nc = 3
rsum = np.zeros([nr],dtype=np.int32)
csum = np.zeros([nc],dtype=np.int32)
x = np.array([[1,2,3] ,
              [4,5,6] ,
              [7,8,9]])

```

```

csum=np.sum(x,axis=0)
rsum=np.sum(x,axis=1)
print("\n 3 * 3 matrix \n\n",x)
print("\n Row sum      = ",rsum)
print(" Column sum    = ",csum)
Explicit continuation
import numpy as np
nr = 3
nc = 3
rsum = np.zeros([nr],dtype=np.int32)
csum = np.zeros([nc],dtype=np.int32)
x = np.array([[1,2,3] , \
              [4,5,6] , \
              [7,8,9]])
csum=np.sum(x,axis=0)
rsum=np.sum(x,axis=1)
print("\n 3 * 3 matrix \n\n",x)
print("\n Row sum      = ",rsum)
print(" Column sum    = ",csum)

```

6.4 Simple 1 and 2 d array slicing

Numpy arrays support slicing. The basic slice syntax is start:end:increment, where start, end and increment are all integers. We will look several examples to illustrate slicing.

6.4.1 Example 7 - simple one d slicing

Here is the program.

```

import numpy as np
x=np.array([0,1,2,3,4,5,6,7,8,9,10])
even = np.empty([6],dtype=np.int32)
odd  = np.empty([5],dtype=np.int32)
even=x[0:11:2]
odd =x[1:11:2]
print(even)
print(odd)

```

Here is the output.

```

$ python3 c0607.py
[ 0  2  4  6  8 10]
[ 1  3  5  7  9]

```

The slice syntax is 0:11:2 in the even example, and 1:11:2 in the odd example.

6.4.2 Example 8 - two d slicing

Here is the program.

```

import numpy as np
nr = 3
nc = 3
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
col      = np.zeros([nr],dtype=np.int32)
row      = np.zeros([nc],dtype=np.int32)
diagonal = np.zeros([nc],dtype=np.int32)

```

```
col=x[0:nr,1]
row=x[1,0:nc]
diagonal=np.diagonal(x)
print("\n 3 * 3 matrix \n\n",x)
print("\n Column    = ",col)
print("\n Row        = ",row)
print("\n Diagonal = ",diagonal)
```

Here is the output.

```
$ python3 c0608.py
```

```
3 * 3 matrix
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
Column    =  [2 5 8]
```

```
Row        =  [4 5 6]
```

```
Diagonal =  [1 5 9]
```

`x[0:nr,1]` selects column 1, and `x[1,0:nc]` selects row 1. There is a diagonal method to extract the diagonal elements of the 3*3 matrix.

6.4.3 Example 9 - arithmetic and slicing

Here is the example.

```
import numpy as np
nr = 3
nc = 3
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
print("\n x before \n\n",x)
x[0:nr,1]=x[0:nr,1]*3
print("\n x after \n\n",x)
```

Here is the output.

```
$ python3 c0609.py
```

```
x before
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
x after
```

```
[[ 1  6  3]
 [ 4 15  6]]
```

```
[ 7 24  9]]
```

and we have multiplied the whole middle column by 3.

6.5 Miscellaneous examples: aggregate, reshape, copies and views

The next example look at some of the built in aggregate functions available in Numpy.

6.5.1 Example 10 - Aggregate usage

Here is the source code.

```
import numpy as np
nr = 3
nc = 3
rsum = np.zeros([nr],dtype=np.int32)
csum = np.zeros([nc],dtype=np.int32)
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
rsum=np.sum(x,axis=0)
csum=np.sum(x,axis=1)
print("\n 3 * 3 matrix \n\n",x)
print("\n Row sum      = ",rsum)
print(" Column sum    = ",csum)
print(x.sum())
print(x.min())
print(x.max())
print(x.mean())
print(x.std())
```

Here is the output.

```
$ 3 * 3 matrix

[[1 2 3]
 [4 5 6]
 [7 8 9]]

Row sum      = [12 15 18]
Column sum   = [ 6 15 24]
45
1
9
5.0
2.58198889747
```

6.5.2 Example 11 - Shape manipulation

Python has a reshape intrinsic. Here is an example.

```
import numpy as np
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
y = x.reshape(1,9)
z = x.reshape(9,1)
print(x)
print(x.shape)
```

```
print(x.size)
print(y)
print(y.shape)
print(y.size)
print(z)
print(z.shape)
print(z.size)
```

Here is the output.

```
$ [[1 2 3]
   [4 5 6]
   [7 8 9]]
(3, 3)
9
[[1 2 3 4 5 6 7 8 9]]
(1, 9)
9
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
(9, 1)
9
```

6.5.3 Example 12 - Copies or views

Look at the following example.

```
import numpy as np
x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
y = x
print(x)
print(y)
y[2,2]=99
print(x)
print(y)
```

Here is the output.

```
$ python3 numpy_02.py
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 1  2  3]
 [ 4  5  6]]
```

```
[ 7  8 99]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8 99]]
```

There is a copy method if we need a copy.

6.6 Numpy documentation

There are 2 pdfs that are available.

Numpy User Guide, 107. Free download.

Numpy Reference Manual, 1528. Free download.

6.7 Problems

1. Rewrite example 1 to use the sum method.
2. Rewrite example 6 to produce the similar output to the C++ version.
3. Here is a table of exam results.

Name	Physics	Maths	Biology	History	English	French
Fowler L.	50	47	28	89	30	46
Barron L.W	37	67	34	65	68	98
Warren J.	25	45	26	48	10	36
Mallory D.	89	56	33	45	30	65
Codd S.	68	78	38	76	98	65

Write a program that initialises a 5 * 6 array with the data as shown above. Then scale the biology data by 2.5. Then generate the people and subject sums. When you have done this calculate the people and subject averages.

Print out the original matrix with the row and column sums added.

4. Choose another site from the Met Office list. Use the sum intrinsic. Calculate the sum and average.

7 Text in Python: Strings

7.1 Introduction

The following information is taken from

<https://docs.python.org/3.5/library/stdtypes.html#str>

Text in Python is provided by the text sequence type - str, Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

Single quotes: 'allows embedded "double" quotes'

Double quotes: "allows embedded 'single' quotes".

Triple quoted: """Three single quotes", """"Three double quotes""""

Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.

String literals that are part of a single expression and have only whitespace between them will be implicitly converted to a single string literal. That is, ("spam " "eggs") == "spam eggs".

Strings may also be created from other objects using the str constructor.

Since there is no separate “character” type, indexing a string produces strings of length 1. That is, for a non-empty string s, s[0] == s[0:1].

There is also no mutable string type, but str.join() or io.StringIO can be used to efficiently construct strings from multiple fragments.

7.2 String Methods

Strings implement all of the common sequence operations, along with the additional methods described below.

Strings also support two styles of string formatting, one providing a large degree of flexibility and customization (see str.format(), Format String Syntax and String Formatting) and the other based on C printf style formatting that handles a narrower range of types and is slightly harder to use correctly, but is often faster for the cases it can handle (printf-style String Formatting).

The Text Processing Services section of the standard library covers a number of other modules that provide various text related utilities (including regular expression support in the re module).

str.capitalize()

Return a copy of the string with its first character capitalized and the rest lowercased.

str.casefold()

Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.

str.center(width[, fillchar])

Return centered in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).

str.count(sub[, start[, end]])

Return the number of non-overlapping occurrences of substring `sub` in the range `[start, end]`. Optional arguments `start` and `end` are interpreted as in slice notation.

`str.encode(encoding="utf-8", errors="strict")`

Return an encoded version of the string as a bytes object. Default encoding is 'utf-8'. `errors` may be given to set a different error handling scheme. The default for `errors` is 'strict', meaning that encoding errors raise a `UnicodeError`. Other possible values are 'ignore', 'replace', 'xmlcharrefreplace', 'backslashreplace' and any other name registered via `codecs.register_error()`, see section Error Handlers. For a list of possible encodings, see section Standard Encodings.

`str.endswith(suffix[, start[, end]])`

Return `True` if the string ends with the specified suffix, otherwise return `False`. `suffix` can also be a tuple of suffixes to look for. With optional `start`, test beginning at that position. With optional `end`, stop comparing at that position.

`str.expandtabs(tabsize=8)`

Return a copy of the string where all tab characters are replaced by one or more spaces, depending on the current column and the given tab size. Tab positions occur every `tabsize` characters (default is 8, giving tab positions at columns 0, 8, 16 and so on). To expand the string, the current column is set to zero and the string is examined character by character. If the character is a tab (`\t`), one or more space characters are inserted in the result until the current column is equal to the next tab position. (The tab character itself is not copied.) If the character is a newline (`\n`) or return (`\r`), it is copied and the current column is reset to zero. Any other character is copied unchanged and the current column is incremented by one regardless of how the character is represented when printed.

`str.find(sub[, start[, end]])`

Return the lowest index in the string where substring `sub` is found, such that `sub` is contained in the slice `s[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation. Return `-1` if `sub` is not found.

`str.format(*args, **kwargs)`

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces `{}`. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

`str.format_map(mapping)`

Similar to `str.format(**mapping)`, except that `mapping` is used directly and not copied to a dict. This is useful if for example `mapping` is a dict subclass:

`str.index(sub[, start[, end]])`

Like `find()`, but raise `ValueError` when the substring is not found.

`str.isalnum()`

Return `true` if all characters in the string are alphanumeric and there is at least one character, `false` otherwise. A character `c` is alphanumeric if one of the following returns `True`: `c.isalpha()`, `c.isdecimal()`, `c.isdigit()`, or `c.isnumeric()`.

`str.isalpha()`

Return true if all characters in the string are alphabetic and there is at least one character, false otherwise. Alphabetic characters are those characters defined in the Unicode character database as “Letter”, i.e., those with general category property being one of “Lm”, “Lt”, “Lu”, “Ll”, or “Lo”. Note that this is different from the “Alphabetic” property defined in the Unicode Standard.

`str.isdecimal()`

Return true if all characters in the string are decimal characters and there is at least one character, false otherwise. Decimal characters are those from general category “Nd”. This category includes digit characters, and all characters that can be used to form decimal-radix numbers, e.g. U+0660, ARABIC-INDIC DIGIT ZERO.

`str.isdigit()`

Return true if all characters in the string are digits and there is at least one character, false otherwise. Digits include decimal characters and digits that need special handling, such as the compatibility superscript digits. Formally, a digit is a character that has the property value `Numeric_Type=Digit` or `Numeric_Type=Decimal`.

`str.isidentifier()`

Return true if the string is a valid identifier according to the language definition, section Identifiers and keywords.

`str.islower()`

Return true if all cased characters [4] in the string are lowercase and there is at least one cased character, false otherwise.

`str.isnumeric()`

Return true if all characters in the string are numeric characters, and there is at least one character, false otherwise. Numeric characters include digit characters, and all characters that have the Unicode numeric value property, e.g. U+2155, VULGAR FRACTION ONE FIFTH. Formally, numeric characters are those with the property value `Numeric_Type=Digit`, `Numeric_Type=Decimal` or `Numeric_Type=Numeric`.

`str.isprintable()`

Return true if all characters in the string are printable or the string is empty, false otherwise. Nonprintable characters are those characters defined in the Unicode character database as “Other” or “Separator”, excepting the ASCII space (0x20) which is considered printable. (Note that printable characters in this context are those which should not be escaped when `repr()` is invoked on a string. It has no bearing on the handling of strings written to `sys.stdout` or `sys.stderr`.)

`str.isspace()`

Return true if there are only whitespace characters in the string and there is at least one character, false otherwise. Whitespace characters are those characters defined in the Unicode character database as “Other” or “Separator” and those with bidirectional property being one of “WS”, “B”, or “S”.

`str.istitle()`

Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

`str.isupper()`

Return true if all cased characters [4] in the string are uppercase and there is at least one cased character, false otherwise.

`str.join(iterable)`

Return a string which is the concatenation of the strings in the iterable iterable. A `TypeError` will be raised if there are any non-string values in iterable, including bytes objects. The separator between elements is the string providing this method.

`str.ljust(width[, fillchar])`

Return the string left justified in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to `len(s)`.

`str.lower()`

Return a copy of the string with all the cased characters [4] converted to lowercase.

`str.lstrip([chars])`

Return a copy of the string with leading characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace. The chars argument is not a prefix; rather, all combinations of its values are stripped:

`static str.maketrans(x[, y[, z]])`

This static method returns a translation table usable for `str.translate()`.

`str.partition(sep)`

Split the string at the first occurrence of sep, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

`str.replace(old, new[, count])`

Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

`str.rfind(sub[, start[, end]])`

Return the highest index in the string where substring sub is found, such that sub is contained within `s[start:end]`. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.

`str.rindex(sub[, start[, end]])`

Like `rfind()` but raises `ValueError` when the substring sub is not found.

`str.rjust(width[, fillchar])`

Return the string right justified in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to `len(s)`.

`str.rpartition(sep)`

Split the string at the last occurrence of sep, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

`str.split(sep=None, maxsplit=-1)`

Return a list of the words in the string, using `sep` as the delimiter string. If `maxsplit` is given, at most `maxsplit` splits are done, the rightmost ones. If `sep` is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

`str.rstrip([chars])`

Return a copy of the string with trailing characters removed. The `chars` argument is a string specifying the set of characters to be removed. If omitted or `None`, the `chars` argument defaults to removing whitespace. The `chars` argument is not a suffix; rather, all combinations of its values are stripped:

`str.split(sep=None, maxsplit=-1)`

Return a list of the words in the string, using `sep` as the delimiter string. If `maxsplit` is given, at most `maxsplit` splits are done (thus, the list will have at most `maxsplit+1` elements). If `maxsplit` is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

`str.splitlines([keepends])`

Return a list of the lines in the string, breaking at line boundaries. Line breaks are not included in the resulting list unless `keepends` is given and true.

`str.startswith(prefix[, start[, end]])`

Return `True` if string starts with the prefix, otherwise return `False`. `prefix` can also be a tuple of prefixes to look for. With optional `start`, test string beginning at that position. With optional `end`, stop comparing string at that position.

`str.strip([chars])`

Return a copy of the string with the leading and trailing characters removed. The `chars` argument is a string specifying the set of characters to be removed. If omitted or `None`, the `chars` argument defaults to removing whitespace. The `chars` argument is not a prefix or suffix; rather, all combinations of its values are stripped:

`str.swapcase()`

Return a copy of the string with uppercase characters converted to lowercase and vice versa. Note that it is not necessarily true that `s.swapcase().swapcase() == s`.

`str.title()`

Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

`str.translate(table)`

Return a copy of the string in which each character has been mapped through the given translation table. The table must be an object that implements indexing via `__getitem__()`, typically a mapping or sequence. When indexed by a Unicode ordinal (an integer), the table object can do any of the following: return a Unicode ordinal or a string, to map the character to one or more other characters; return `None`, to delete the character from the return string; or raise a `LookupError` exception, to map the character to itself.

`str.upper()`

Return a copy of the string with all the cased characters [4] converted to uppercase. Note that `str.upper().isupper()` might be `False` if `s` contains uncased characters or if

the Unicode category of the resulting character(s) is not “Lu” (Letter, uppercase), but e.g. “Lt” (Letter, titlecase).

`str.zfill(width)`

Return a copy of the string left filled with ASCII '0' digits to make a string of length width. A leading sign prefix ('+/-') is handled by inserting the padding after the sign character rather than before. The original string is returned if width is less than or equal to `len(s)`.

As can be seen there are a lot of string methods. We will look at a small number of examples using some of the above.

7.3 String example 1 - initialisation, len and find methods

The first example illustrates the following string concepts

- string initialisation
- the use of the `len()` internal method
- a string as an array
- the find method
- sections or slices of a string

Here is the example.

```
name = "Ian Chivers"
l=len(name)
for i in range(0,l):
    print(name[i],end="")
print()
blank=name.find(" ")
print(name[0:blank])
print(name[blank+1:l])
```

Here is the output.

```
$ python3 c0701.py
Ian Chivers
Ian
Chivers
```

7.4 String example 2 - concatenation and split method

This example shows

- string concatenation
- the use of the `split()` method

Here is the program.

```
line1 = "The important thing about a language, is not so"
line2 = "much what features the language posses, but"
line3 = "the features it does posses, are sufficient, to"
line4 = "support the desired programming styles, in the"
line5 = "desired application areas."
total = line1 + " " + line2 + " " + line3 + " " + line4 +
" " + line5
```

```
print(total)
words=total.split()
for word in words:
    print(word)
```

Here is the output.

```
$ python3 c0702.py
```

```
The important thing about a language, is not so much what
features the language posses, but the features it does pos-
ses, are sufficient, to support the desired programming
styles, in the desired application areas.
```

```
The
important
thing
about
a
language,
is
not
so
much
what
features
the
language
posses,
but
the
features
it
does
posses,
are
sufficient,
to
support
the
desired
programming
styles,
in
the
desired
application
areas.
```

The next example use regular expressions to break the text into phrases broken at punctuation.

7.5 String example 3 - split variant

Here is the source.

```
import re

def main():

    line1 = "The important thing about a language, is not so"
    line2 = "much what features the language possess, but"
    line3 = "the features it does possess, are sufficient, to"
    line4 = "support the desired programming styles, in the"
    line5 = "desired application areas."
    total_string = line1 + " " + line2 + " " + line3 + " " +
line4 + " " + line5
    print(total_string)
    phrases=re.split('[.,]',total_string)
    for phrase in phrases:
        print(phrase)

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
The important thing about a language, is not so much what
features the language possess, but the features it does pos-
sess, are sufficient, to support the desired programming
styles, in the desired application areas.
```

```
The important thing about a language
is not so much what features the language possess
but the features it does possess
are sufficient
to support the desired programming styles
in the desired application areas
```

The key difference is

```
phrases=re.split('[.,]',total_string)
```

where we now have a regular expression[.,] as the basis of the characters to split on.

7.6 String example 4 - reading from an external file

In this example we read data from an external file. We will use a file from <http://www.metoffice.gov.uk/public/weather/climate-historic/#?tab=climateHistoric>

In the example below I will use

```
http://www.metoffice.gov.uk/pub/data/weather/uk/cli-
mate/stationdata/cwmystwythdata.txt
```

In the problems you can replace this file wiith one of your own choice.

Here is the program.

```
data_file="cwmystwythdata.txt"
f=open(data_file)
```



```

print(" Here are the header lines\n\n")
for i in range(0,7):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)
print("\n\n ** 1959 and 1960 have missing rainfall values
\n\n")
for i in range(0,12):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)
for i in range(0,12):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)
print("\n\n ** 1961, 1962 and 1963 are incomplete ** \n\n")
for i in range(0,9):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)
for i in range(0,8):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)
for i in range(0,10):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)
print("\n\n ** 1964 is the first complete year of rainfall
values ** \n\n")
for i in range(0,12):
    line=f.readline()
    line=line.rstrip('\n')
    print(line)

```

Here is the output from running the program.

Here are the header lines

```

Cwmystwyth
Location: 2773E 2749N, 301 metres amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked
by ---.
Sunshine data taken from an automatic Kipp & Zonen sensor
marked with a #, otherwise sunshine data taken from a Camp-
bell Stokes recorder.
    yyyy   mm   tmax   tmin   af   rain   sun
           degC degC   days   mm   hours

```

** 1959 and 1960 have missing rainfall values

1959	1	4.5	-1.9	20	---	57.2
1959	2	7.3	0.9	15	---	87.2
1959	3	8.4	3.1	3	---	81.6
1959	4	10.8	3.7	1	---	107.4
1959	5	15.8	5.8	1	---	213.5
1959	6	16.9	8.2	0	---	209.4
1959	7	18.5	9.5	0	---	167.8
1959	8	19.0	10.5	0	---	164.8
1959	9	18.3	5.9	0	---	196.5
1959	10	14.8	7.9	1	---	101.1
1959	11	8.8	3.9	3	---	38.9
1959	12	7.2	2.5	3	---	19.2
1960	1	6.3	0.6	15	---	30.7
1960	2	5.3	-0.3	17	---	50.2
1960	3	8.2	2.4	4	---	73.9
1960	4	11.2	2.6	7	---	146.8
1960	5	15.4	6.5	2	---	153.9
1960	6	18.5	8.2	0	---	225.6
1960	7	16.0	9.3	0	---	111.3
1960	8	16.5	9.4	0	---	119.2
1960	9	15.0	7.9	0	---	120.3
1960	10	12.0	5.3	5	---	---
1960	11	8.8	2.9	5	---	37.3
1960	12	5.9	0.4	13	---	33.9

** 1961, 1962 and 1963 are incomplete **

1961	1	5.4	0.2	11	144.8	31.0
1961	2	8.7	2.9	2	112.5	45.2
1961	3	10.2	2.1	10	77.2	102.6
1961	4	11.9	5.0	1	130.7	83.9
1961	5	---	---	---	66.3	173.7
1961	6	---	7.4	---	66.1	190.6
1961	7	16.7	8.2	0	141.1	149.2
1961	8	16.8	10.1	0	149.5	106.6
1961	9	17.4	9.3	0	134.8	79.7
1962	5		4.2	3	117.8	102.2
1962	6		6.8	1	72.8	163.9
1962	7	16.8	9.1	0	56.7	---
1962	8	15.6	9.3	0	236.2	---
1962	9	14.6	7.8	1	218.0	---
1962	10	---	---	---	69.7	---
1962	11	7.6	1.8	9	85.2	---
1962	12	5.3	-1.0	18	204.4	---

1963	3	---	---	---	106.2	---
1963	4	10.6	3.0	1	159.7	---
1963	5	12.1	4.9	1	126.9	117.4
1963	6	---	---	---	121.6	---
1963	7	---	---	---	62.9	---
1963	8	---	---	---	154.3	78.8
1963	9	---	---	---	165.0	---
1963	10	---	---	---	139.0	---
1963	11	9.8	4.3	1	234.4	55.6
1963	12	4.4	-0.9	18	19.7	59.9

** 1964 is the first complete year of rainfall values **

1964	1	5.6	0.6	15	83.1	30.6
1964	2	5.7	0.7	14	38.5	47.5
1964	3	6.5	0.7	15	67.3	88.7
1964	4	10.3	3.6	5	76.4	102.4
1964	5	15.2	7.5	1	90.4	142.8
1964	6	15.2	8.2	0	83.5	104.5
1964	7	16.6	10.1	0	177.0	95.4
1964	8	17.2	9.4	0	180.5	155.7
1964	9	16.4	8.2	0	66.0	140.3
1964	10	11.2	3.6	0	171.9	92.1
1964	11	9.5	4.3	6	174.5	43.6
1964	12	5.9	-0.2	14	334.8	44.0

7.7 String example 5 - reading data from a file and calculating sum and average rainfall values

In the previous example the first year with 12 months of rainfall data was 1964. We will read this data in and extract the rainfall values for the year and calculate the sum and average of the rainfall in inches.

Here is the program.

```
import numpy as np
data_file="cwmystwythdata.txt"
nmonths=12
cmsum=0.0
imperial_sum      = 0.0
imperial_average = 0.0
x = np.empty([nmonths] , dtype=np.float64)
f=open(data_file)
print(" ** Skipping header lines ** \n")
for i in range(0,7):
    line=f.readline()
print(" ** Skipping 1959 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1960 ** \n")
```

```

for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1961 ** \n")
for i in range(0,9):
    line=f.readline()
print(" ** Skipping 1962 ** \n")
for i in range(0,8):
    line=f.readline()
print(" ** Skipping 1963 ** \n")
for i in range(0,10):
    line=f.readline()
print(" ** Reading 1964 ** \n")
for i in range(0,12):
    line=f.readline()
    x[i]=(float)(line[36:42])
    print(" %6.1f " % x[i])
print(" ** mms ** ")
print(" %6.1f " % x.sum())
print(" %6.1f " % (x.sum()/nmonths))
imperial_sum=x.sum()/25.4
imperial_average=imperial_sum/nmonths
print(" ** inches ** ")
print(" %6.1f " % imperial_sum)
print(" %6.1f " % imperial_average)

```

Here is the output.

```

$ python3 c0804.py
** Skipping header lines **

** Skipping 1959 **

** Skipping 1960 **

** Skipping 1961 **

** Skipping 1962 **

** Skipping 1963 **

** Reading 1964 **

83.1
38.5
67.3
76.4
90.4
83.5
177.0
180.5
66.0

```

```

171.9
174.5
334.8
** mms **
1543.9
128.7
** inches **
60.8
5.1

```

We have also formatted the numeric output.

7.8 String example 6 - simple variant of the previous example using the .format option

This is a simple variant of the last example using .format.

```

import numpy as np
data_file="cwmystwythdata.txt"
nmonths=12
cmsum=0.0
imperial_sum      = 0.0
imperial_average = 0.0
x = np.empty([nmonths] , dtype=np.float64)
f=open(data_file)
print(" ** Skipping header lines ** \n")
for i in range(0,7):
    line=f.readline()
print(" ** Skipping 1959 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1960 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1961 ** \n")
for i in range(0,9):
    line=f.readline()
print(" ** Skipping 1962 ** \n")
for i in range(0,8):
    line=f.readline()
print(" ** Skipping 1963 ** \n")
for i in range(0,10):
    line=f.readline()
print(" ** Reading 1964 ** \n")
for i in range(0,12):
    line=f.readline()
    x[i]=(float)(line[36:42])
    print(" %6.1f ".format(x[i]))
print(" ** mms ** ")
print(" %6.1f ".format(x.sum()))
print(" %6.1f ".format((x.sum()/nmonths)))
imperial_sum=x.sum()/25.4

```

```
imperial_average=imperial_sum/nmonths
print(" ** inches ** ")
print(" %6.1f ".format(imperial_sum))
print(" %6.1f ".format(imperial_average))
```

The output is as in the previous example.

7.9 Character data in Python

The next set of examples look at character data in Python. Some of the background material in this section is taken from the following Wikipedia entry.

https://en.wikipedia.org/wiki/Character_encoding

The term character encoding means the use of a coding mechanism to represent a set of characters. Some of the early character encodings are

Morse code - a character encoding scheme used in telecommunication that encodes text characters as standardized sequences of two different signal durations called dots and dashes. Morse code is named for Samuel F. B. Morse, an inventor of the telegraph.

Baudot code, a five-bit encoding, was created by Émile Baudot in 1870, patented in 1874, modified by Donald Murray in 1901, and standardized by CCITT as International Telegraph Alphabet No. 2 (ITA2) in 1930

ASCII - abbreviated from American Standard Code for Information Interchange, is a character encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications equipment, and other devices. Most modern character-encoding schemes are based on ASCII, although they support many additional characters.

Most programming languages use the ASCII character set as the basis for the characters used in writing programs.

7.10 String example 7 - the ASCII character set

The following program

```
def main():

    for i in range(32,64):

        c1 = chr(i)
        c2 = chr(i+32)
        c3 = chr(i+64)

        print( "      {0:3d} {1}      {2:3d} {3}      {4:3d} {5}
".format( (i) , c1 , (i+32) , c2 , (i+64) , c3 ) )

if ( __name__ == "__main__" ):
    main()
```

prints out the ASCII character set. Here is the output from running the program.

```
32      @      64 @      96 `
33 !      65 A      97 a
34 "      66 B      98 b
```

35 #	67 C	99 c
36 \$	68 D	100 d
37 %	69 E	101 e
38 &	70 F	102 f
39 '	71 G	103 g
40 (72 H	104 h
41)	73 I	105 i
42 *	74 J	106 j
43 +	75 K	107 k
44 ,	76 L	108 l
45 -	77 M	109 m
46 .	78 N	110 n
47 /	79 O	111 o
48 0	80 P	112 p
49 1	81 Q	113 q
50 2	82 R	114 r
51 3	83 S	115 s
52 4	84 T	116 t
53 5	85 U	117 u
54 6	86 V	118 v
55 7	87 W	119 w
56 8	88 X	120 x
57 9	89 Y	121 y
58 :	90 Z	122 z
59 ;	91 [123 {
60 <	92 \	124
61 =	93]	125 }
62 >	94 ^	126 ~
63 ?	95 _	127

There are a number of limitations with the ASCII character set, including

- no support for languages other than English
- no access to symbols widely used in science and engineering
- no support for ligatures
- no support for mathematical equations

Unicode and the Unicode standard is an attempt to address some of these issues.

7.11 Unicode

Here is the home address of the Unicode Consortium.

<https://unicode.org/>

In the ASCII character set we see the use of numbers to provide access to characters. Computers store letters and other characters by assigning a number for each one.

Before Unicode was invented, there were hundreds of different systems, called character encodings, for assigning these numbers. These early character encodings were limited and could not contain enough characters to cover all the world's languages. Even for a single language like English no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

Early character encodings also conflicted with one another. That is, two encodings could use the same number for two different characters, or use different numbers for the same character. Any given computer (especially servers) would need to support many different encodings. However, when data is passed through different computers or between different encodings, that data runs the risk of corruption.

Unicode solves many of these problems. Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. The standard is maintained by the Unicode Consortium, and as of June 2018 the most recent version, Unicode 11.0, contains a repertoire of 137,439 characters covering 146 modern and historic scripts, as well as multiple symbol sets and emoji. The character repertoire of the Unicode Standard is synchronized with ISO/IEC 10646, and both are code-for-code identical.

The Unicode Standard consists of a set of code charts for visual reference, an encoding method and set of standard character encodings, a set of reference data files, and a number of related items, such as character properties, rules for normalization, decomposition, collation, rendering, and bidirectional display order (for the correct display of text containing both right-to-left scripts, such as Arabic and Hebrew, and left-to-right scripts).

Unicode's success at unifying character sets has led to its widespread and predominant use in the internationalization and localization of computer software. The standard has been implemented in many recent technologies, including modern operating systems, XML, Java (and other programming languages), and the .NET Framework.

Unicode can be implemented by different character encodings. The Unicode standard defines UTF-8, UTF-16, and UTF-32, and several other encodings are in use. The most commonly used encodings are UTF-8, UTF-16, and UCS-2, a precursor of UTF-16.

UTF-8, the dominant encoding on the World Wide Web (used in over 92% of websites), uses one byte for the first 128 code points, and up to 4 bytes for other characters. The first 128 Unicode code points are the ASCII characters, which means that any ASCII text is also a UTF-8 text.

UCS-2 uses two bytes (16 bits) for each character but can only encode the first 65,536 code points, the so-called Basic Multilingual Plane (BMP). With 1,114,112 code points on 17 planes being possible, and with over 137,000 code points defined so far, UCS-2 is only able to represent less than half of all encoded Unicode characters. Therefore, UCS-2 is obsolete, though still widely used in software. UTF-16 extends UCS-2, by using the same 16-bit encoding as UCS-2 for the Basic Multilingual Plane, and a 4-byte encoding for the other planes. As long as it contains no code points in the reserved range U+D800–U+DFFF, a UCS-2 text is a valid UTF-16 text.

UTF-32 (also referred to as UCS-4) uses four bytes for each character. Like UCS-2, the number of bytes per character is fixed, facilitating character indexing; but unlike UCS-2, UTF-32 is able to encode all Unicode code points. However, because each character uses four bytes, UTF-32 takes significantly more space than other encodings, and is not widely used.

7.12 String example 8 - Unicode characters

The following program

```
import array

def main():
```



```

# The first Unicode standard supported 16 bit characters
# 65535

n = (2**16)-1
print(" Size = ",n)

# Create a character array of size n and
# fill the character array with X

character_array = ['X'] * n

start = 0
end = int(n/64)

# 1023 lines

print(" start = ",start)
print(" end    = ",end)

text_buffer = array.array( 'u' , character_array )

# The
#
# chr()
#
# function returns the character that represents the speci-
# fied unicode.

for i in range(31,n):
    text_buffer[i] = chr(i)

for l in range(0,32):
    print(" Line = {0:4d} ".format(l),end=" ")
    for c in range(0,64):
        print( text_buffer[start] , end="" )
        start = start + 1
    print()

if ( __name__ == "__main__" ):
    main()

```

looks at creating an array of Unicode characters and printing some of them out. Try running it from an IDE and from the command line. What do you notice about the output?

7.13 Example 9 - another unicode example

Here is the source code.

```

import array

def main():

```

```

# A more recent Unicode standard supports
# 1,114,112 code points

n = 1114112
print(" Size = ",n)

# Create a character array of size n and
# fill the character array with X

character_array = ['X'] * n

start = 0
end = int(n/64)

# 1023 lines

print(" start = ",start)
print(" end    = ",end)

text_buffer = array.array( 'u' , character_array )

# The
#
# chr()
#
# function returns the character that represents the speci-
# fied unicode.

for i in range(31,n):
    text_buffer[i] = chr(i)

for l in range(0,32):
    print(" Line = {0:4d} ".format(l),end=" ")
    for c in range(0,64):
        print( text_buffer[start] , end="" )
        start = start + 1
    print()

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

Size = 1114112
start = 0
end = 17408
Traceback (most recent call last):
  File "c0709.py", line 43, in <module>
    main()
  File "c0709.py", line 33, in main
    text_buffer[i] = chr(i)

```

`TypeError: array item must be unicode character`

We will look at resolving the error with this program in the practicals.

7.14 Problems

1. Compile and run the examples in this chapter.,
2. Replace the Cwmystwyth file with one of your choice from the Met Office site. Make the necessary changes to print the first 6 years data.
3. Using one of the earlier programs write a program that produces the following output.

```
!
"#
$%&
'()*
+,-./
012345
6789:;<
=>?@ABCD
EFGHIJKLM
NOPQRSTUVWXYZ
XYZ[\]^_`ab
cdefghijklmn
opqrstuvwxyz{
|}~
```

One way to print multiple characters on a line is to use the print method and `end=""`.

The above are the printing characters from the ASCII character set.

4. Modify the above program to produce the following output.

```
!
"#$
%&'()
*+,-./0
123456789
:;<=>?@ABCD
EFGHIJKLMNOPQ
RSTUVWXYZ[\]^_`
abcdefghijklmnopq
rstuvwxyz{|}~
```

Again we assume the ASCII character set

5. Using the split examples as a starting point write a program that generates the following output.

Output:

```
Words = 34
1: a [1]
2: is so it to in [5]
3: The not the but the are the the [8]
4: much what does [3]
5: issue about areas [3]
6: styles [1]
```

```
7: possess support desired desired [4]
8: language features language features [4]
9: important possesses [2]
10: sufficient [1]
11: programming application [2]
```

7. Visit the UNICODE site.

<http://www.unicode.org/standard/WhatIsUnicode.htm>

Chose a code chart.

<http://www.unicode.org/charts/>

This is one of the code tables.

<http://www.unicode.org/charts/PDF/U13A0.pdf>

Choose one or more characters. Can you get your characters to display in a Python program?

Summarising: as a slow-witted human being I have a very small head and I had better learn to live with it and to respect my limitations and give them full credit, rather than try to ignore them, for the latter vain effort will be punished by failure.

Edsger W. Dijkstra, Structured Programming.

8 Control Structures - compound statements

There are a reasonable range of control structures in Python. The following information is taken from section 8 of the reference manual.

8.1 Compound statements

Compound statements contain (groups of) other statements; they affect or control the execution of those other statements in some way. In general, compound statements span multiple lines, although in simple incarnations a whole compound statement may be contained in one line.

The `if`, `while` and `for` statements implement traditional control flow constructs. `try` specifies exception handlers and/or cleanup code for a group of statements, while the `with` statement allows the execution of initialization and finalization code around a block of code. Function and class definitions are also syntactically compound statements.

A compound statement consists of one or more ‘clauses.’ A clause consists of a header and a ‘suite.’ The clause headers of a particular compound statement are all at the same indentation level. Each clause header begins with a uniquely identifying keyword and ends with a colon. A suite is a group of statements controlled by a clause. A suite can be one or more semicolon-separated simple statements on the same line as the header, following the header’s colon, or it can be one or more indented statements on subsequent lines. Only the latter form of a suite can contain nested compound statements; the following is illegal, mostly because it wouldn’t be clear to which `if` clause a following `else` clause would belong:

```
if test1: if test2: print(x)
```

Also note that the semicolon binds tighter than the colon in this context, so that in the following example, either all or none of the `print()` calls are executed:

```
if x < y < z: print(x); print(y); print(z)
```

The compound statements are

- `if_stmt`
- `while_stmt`
- `for_stmt`
- `try_stmt`
- `with_stmt`
- `funcdef`
- `classdef`
- `async_with_stmt`
- `async_for_stmt`

```

async_funcdef
suite      ::= stmt_list NEWLINE | NEWLINE INDENT statement+ DEDENT
statement  ::= stmt_list NEWLINE | compound_stmt
stmt_list ::= simple_stmt (";" simple_stmt)* [";"]

```

Note that statements always end in a NEWLINE possibly followed by a DEDENT. Also note that optional continuation clauses always begin with a keyword that cannot start a statement, thus there are no ambiguities (the ‘dangling else’ problem is solved in Python by requiring nested if statements to be indented).

The formatting of the grammar rules in the following sections places each clause on a separate line for clarity.

8.2 The if statement

The if statement is used for conditional execution:

```

if_stmt ::= "if" expression ":" suite
         ( "elif" expression ":" suite ) *
         ["else" ":" suite]

```

It selects exactly one of the suites by evaluating the expressions one by one until one is found to be true (see section Boolean operations for the definition of true and false); then that suite is executed (and no other part of the if statement is executed or evaluated). If all expressions are false, the suite of the else clause, if present, is executed.

8.3 The while statement

The while statement is used for repeated execution as long as an expression is true:

```

while_stmt ::= "while" expression ":" suite
            ["else" ":" suite]

```

This repeatedly tests the expression and, if it is true, executes the first suite; if the expression is false (which may be the first time it is tested) the suite of the else clause, if present, is executed and the loop terminates.

A break statement executed in the first suite terminates the loop without executing the else clause’s suite. A continue statement executed in the first suite skips the rest of the suite and goes back to testing the expression.

8.4 The for statement

The for statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object:

```

for_stmt ::= "for" target_list "in" expression_list ":"
           suite
           ["else" ":" suite]

```

The expression list is evaluated once; it should yield an iterable object. An iterator is created for the result of the expression_list. The suite is then executed once for each item provided by the iterator, in the order returned by the iterator. Each item in turn is assigned to the target list using the standard rules for assignments (see Assignment statements), and then the suite is executed. When the items are exhausted (which is immediately when the sequence is empty or an iterator raises a StopIteration exception), the suite in the else clause, if present, is executed, and the loop terminates.

A break statement executed in the first suite terminates the loop without executing the else clause's suite. A continue statement executed in the first suite skips the rest of the suite and continues with the next item, or with the else clause if there is no next item.

The for-loop makes assignments to the variables(s) in the target list. This overwrites all previous assignments to those variables including those made in the suite of the for-loop:

```
for i in range(10):
    print(i)
    i = 5
# this will not affect the for-loop
# because i will be overwritten with the next
# index in the range
```

Names in the target list are not deleted when the loop is finished, but if the sequence is empty, they will not have been assigned to at all by the loop. Hint: the built-in function range() returns an iterator of integers suitable to emulate the effect of Pascal's for i := a to b do; e.g., list(range(3)) returns the list [0, 1, 2].

Note There is a subtlety when the sequence is being modified by the loop (this can only occur for mutable sequences, i.e. lists). An internal counter is used to keep track of which item is used next, and this is incremented on each iteration. When this counter has reached the length of the sequence the loop terminates. This means that if the suite deletes the current (or a previous) item from the sequence, the next item will be skipped (since it gets the index of the current item which has already been treated). Likewise, if the suite inserts an item in the sequence before the current item, the current item will be treated again the next time through the loop. This can lead to nasty bugs that can be avoided by making a temporary copy using a slice of the whole sequence, e.g.,

```
for x in a[:]:
    if x < 0: a.remove(x)
```

8.5 The try statement

The try statement specifies exception handlers and/or cleanup code for a group of statements:

```
try_stmt ::= try1_stmt | try2_stmt
try1_stmt ::= "try" ":" suite
              ("except" [expression ["as" identifier]]
               ":" suite)+
              ["else" ":" suite]
              ["finally" ":" suite]
try2_stmt ::= "try" ":" suite
              "finally" ":" suite
```

The except clause(s) specify one or more exception handlers. When no exception occurs in the try clause, no exception handler is executed. When an exception occurs in the try suite, a search for an exception handler is started. This search inspects the except clauses in turn until one is found that matches the exception. An expression-less except clause, if present, must be last; it matches any exception. For an except clause with an expression, that expression is evaluated, and the clause matches the exception if the resulting object is "compatible" with the exception. An object is compatible with an exception if it is the class or a base class of the exception object or a tuple containing an item compatible with the exception.

If no `except` clause matches the exception, the search for an exception handler continues in the surrounding code and on the invocation stack. [1]

If the evaluation of an expression in the header of an `except` clause raises an exception, the original search for a handler is cancelled and a search starts for the new exception in the surrounding code and on the call stack (it is treated as if the entire `try` statement raised the exception).

When a matching `except` clause is found, the exception is assigned to the target specified after the `as` keyword in that `except` clause, if present, and the `except` clause's suite is executed. All `except` clauses must have an executable block. When the end of this block is reached, execution continues normally after the entire `try` statement. (This means that if two nested handlers exist for the same exception, and the exception occurs in the `try` clause of the inner handler, the outer handler will not handle the exception.)

When an exception has been assigned using `as` target, it is cleared at the end of the `except` clause. This is as if

```
except E as N:
    foo
```

was translated to

```
except E as N:
    try:
        foo
    finally:
        del N
```

This means the exception must be assigned to a different name to be able to refer to it after the `except` clause. Exceptions are cleared because with the traceback attached to them, they form a reference cycle with the stack frame, keeping all locals in that frame alive until the next garbage collection occurs.

Before an `except` clause's suite is executed, details about the exception are stored in the `sys` module and can be accessed via `sys.exc_info()`. `sys.exc_info()` returns a 3-tuple consisting of the exception class, the exception instance and a traceback object (see section The standard type hierarchy) identifying the point in the program where the exception occurred. `sys.exc_info()` values are restored to their previous values (before the call) when returning from a function that handled an exception.

The optional `else` clause is executed if and when control flows off the end of the `try` clause. [2] Exceptions in the `else` clause are not handled by the preceding `except` clauses.

If `finally` is present, it specifies a 'cleanup' handler. The `try` clause is executed, including any `except` and `else` clauses. If an exception occurs in any of the clauses and is not handled, the exception is temporarily saved. The `finally` clause is executed. If there is a saved exception it is re-raised at the end of the `finally` clause. If the `finally` clause raises another exception, the saved exception is set as the context of the new exception. If the `finally` clause executes a `return` or `break` statement, the saved exception is discarded:

```
>>>
>>> def f():
...     try:
...         1/0
...     finally:
...         return 42
...
>>> f()
```


42

The exception information is not available to the program during execution of the finally clause.

When a return, break or continue statement is executed in the try suite of a try...finally statement, the finally clause is also executed ‘on the way out.’ A continue statement is illegal in the finally clause. (The reason is a problem with the current implementation — this restriction may be lifted in the future).

The return value of a function is determined by the last return statement executed. Since the finally clause always executes, a return statement executed in the finally clause will always be the last one executed:

```
>>>
>>> def foo():
...     try:
...         return 'try'
...     finally:
...         return 'finally'
...
>>> foo()
'finally'
```

Additional information on exceptions can be found in section Exceptions, and information on using the raise statement to generate exceptions may be found in section The raise statement.

8.6 The with statement

The with statement is used to wrap the execution of a block with methods defined by a context manager (see section With Statement Context Managers). This allows common try...except...finally usage patterns to be encapsulated for convenient reuse.

```
with_stmt ::= "with" with_item ("," with_item)* ":" suite
with_item ::= expression ["as" target]
```

The execution of the with statement with one “item” proceeds as follows:

1. The context expression (the expression given in the with_item) is evaluated to obtain a context manager.
2. The context manager’s `__exit__()` is loaded for later use.
3. The context manager’s `__enter__()` method is invoked.
4. If a target was included in the with statement, the return value from `__enter__()` is assigned to it.

Note The with statement guarantees that if the `__enter__()` method returns without an error, then `__exit__()` will always be called. Thus, if an error occurs during the assignment to the target list, it will be treated the same as an error occurring within the suite would be. See step 6 below.

5. The suite is executed.
6. The context manager’s `__exit__()` method is invoked. If an exception caused the suite to be exited, its type, value, and traceback are passed as arguments to `__exit__()`. Otherwise, three None arguments are supplied.

If the suite was exited due to an exception, and the return value from the `__exit__()` method was false, the exception is reraised. If the return value

was true, the exception is suppressed, and execution continues with the statement following the with statement.

If the suite was exited for any reason other than an exception, the return value from `__exit__()` is ignored, and execution proceeds at the normal location for the kind of exit that was taken.

With more than one item, the context managers are processed as if multiple with statements were nested:

```
with A() as a, B() as b:
    suite
```

is equivalent to

```
with A() as a:
    with B() as b:
        suite
```

We will have a look at some examples next.

8.7 The pass statement

The pass statement does nothing. It can be used when a statement is required syntactically but the program requires no action. For example:

```
while True:
    pass
```

This is commonly used for creating minimal classes:

```
class MyEmptyClass:
    pass
```

Another place pass can be used is as a place-holder for a function or conditional body when you are working on new code, allowing you to keep thinking at a more abstract level. The pass is silently ignored:

```
def initlog(*args):
    pass # Remember to implement this!
```

8.8 Example 1 - the if statement

Here is the first example. We have an if then else in this program.

```
import numpy as np
n=5
height=np.array([1.85,1.80,1.85,1.70,1.75])
weight=np.array([85.0,76.0,85.0,90.0,69.0])
bmi    =np.empty([n],dtype=np.float64)
for i in range(0,n):
    bmi[i]=weight[i]/(height[i]**2)
    print( "{0:5.2f}".format(bmi[i]),end=" ")
    if ( (bmi[i]>20.00) and (bmi[i]<25.00) ):
        print(" Normal")
    elif ( (bmi[i]>25.00) and (bmi[i]<30.00) ):
        print(" Overweight")
    elif ( (bmi[i]>30.00) and (bmi[i]<40.00) ):
        print(" Obese")
```

Here is the output.

```
$ python3 c0801.py
24.84 Normal
23.46 Normal
24.84 Normal
31.14 Obese
22.53 Normal
```

The data is taken from a first year class of Mechanical Engineering students.

8.9 Example 2 - the while statement

Here is a while example. This program evaluates $\exp(1.0)$ using a summation.

```
import math
tol=1.0e-16
etox=1.0
term=1.0
nterm=0
x=1.0
while ( term > tol ):
    nterm+=1
    term = (x/nterm) * term
    etox+=term
print("Calculated      = {0:20.15f} ".format(etox))
print("math.exp        = {0:20.15f} ".format(math.exp(x)))
print("N iterations    = {0:6d} ".format(nterm))
```

Here is the program output.

```
$ python3 c0802.py
Calculated      =      2.718281828459046
math.exp        =      2.718281828459046
N iterations    =          19
```

The series converges relatively quickly.

8.10 Example 3 - the for loop with arrays

You have already seen examples of for loops and arrays. Here is a variation of an earlier one. Here is the program.

```
import numpy as np
n=12
month=0
rainfall = np.array([ 3.1 , 2.0 , 2.4 , 2.1 , 2.2 , 2.2 ,
1.8 , 2.2 , 2.7 , 2.9 , 3.1 , 3.1 ] )
for month in range (0,n):
    print(" {0:2d} {1:4.1f} ".format(month,rainfall[month]))
```

Here is the output.

```
$ python3 c0803.py
0  3.1
1  2.0
2  2.4
```

```
3 2.1
4 2.2
5 2.2
6 1.8
7 2.2
8 2.7
9 2.9
10 3.1
11 3.1
```

8.11 Example 4 - the for loop with lists and enumerate

This example looks at the type of for loop you can use with lists. Here is the program.

```
months=["January","February","March","April","May",
"June","July","August","September","October","November",
"December"]

print(type(months))
for m in months:
    print(m)
for index,value in enumerate(months):
    print(index,value)
```

Here is the output.

```
$ python3 c0804.py
<class 'list'>
January
February
March
April
May
June
July
August
September
October
November
December
0 January
1 February
2 March
3 April
4 May
5 June
6 July
7 August
8 September
9 October
10 November
11 December
```

Note that months is not an array, but a list.

8.12 Example 5 - the for in statement

Here is a variation on the previous example in how to initialise a collection of strings, in this case day names.

```
day_names = [s for s in [ 'Sunday', 'Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday', 'Saturday' ] ]
print(type(day_names))
print(day_names)
```

Here is the output.

```
<class 'list'>
['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday']
```

Note that day_names is not an array, but a list.

8.13 Example 6 - try and except

Here is the example.

```
try:
    testfile = open("silly")
    testfile.write(" Silly billy")
except IOError:
    print("File error")
else:
    print("It worked")
```

When the file does not exist we get.

```
$ python3 c0805.py
File error
```

8.14 Additional material

We will have a look at additional examples throughout other sections of the notes. We will look at

- with
- funcdef
- class def
- async with
- async for
- async funcdef

where appropriate.

8.15 Problems

1. Write a program to print out the 12 times table. Output should be roughly of the form

```
1    *    12    =    12
2    *    12    =    24
```

2. Write a program that produces a conversion table from litres to pints and vice versa. One litre is approximately $1 \frac{3}{4}$ pints. The output should comprise three columns. The middle

column should be an integer and the columns to the left and right should be the corresponding pints and litre values. This enables the middle column to be scanned quickly and the corresponding equivalent in litres or pints read easily.

3. Rewrite the program for the period of a pendulum. The new program should print out the length of the pendulum and period for lengths of the pendulum from 0 to 100 cm in steps of 0.5 cm.

The physical world has many examples where processes require some threshold to be overcome before they begin operation: critical mass in nuclear reactions, a given slope to be exceeded before friction is overcome, and so on. Unfortunately, most of these sorts of calculations become rather complex and not really appropriate here. The following problem tries to restrict the range of calculation, whilst illustrating the possibilities of decision making.

4. If a cubic equation is expressed as

$$z^3 + a_2 z^2 + a_1 z + a_0 = 0$$

and we let

$$q = a_1/3 - (a_2^2 * a_2)/9$$

and

$$r = (a_1 a_2 - 3 a_0)/6 - (a_2 a_2 a_2)/27$$

we can determine the nature of the roots as follows:

$$q^3 + r^2 > 0; \text{ one real root and a pair of complex;}$$

$$q^3 + r^2 = 0; \text{ all roots real, and at least two equal;}$$

$$q^3 + r^2 < 0; \text{ all roots real;}$$

Incorporate this into a suitable program, to determine the nature of the roots of a cubic from suitable input.

5. The form of breaking waves on beaches is a continuum, but for convenience we commonly recognise three major types: surging, plunging and spilling. These may be classified empirically by reference to the wave period, T (seconds), the breaker wave height, H_b (metres), and the beach slope, m. These three variables are combined into a single parameter, B, where

$$B = H_b / (g m T^2)$$

g is the gravitational constant (981 cm sec⁻²). If B is less than .003, the breakers are surging; if B is greater than 0.068, they are spilling, and between these values, plunging breakers are observed.

(i) On the east coast of New Zealand, the normal pattern of waves is swell waves, with wave heights of 1 to 2 metres, and wave periods of 10 to 15 seconds. During storms, the wave period is generally shorter, say 6 to 8 seconds, and the wave heights higher, 3 to 5 metres. The beach slope may be taken as about 0.1. What changes occur in breaker characteristics as a storm builds up?

(ii) Similarly, many beaches have a concave profile. The lower beach generally has a very low slope, say less than 1 degree (m=0.018), but towards the high tide mark, the slope in-

creases dramatically, to say 10 degrees or more ($m=0.18$). What changes in wave type will be observed as the tide comes in?

6. Personal taxation is usually structured in the following way:–

no taxation on the first m_0 units of income;

taxation at $t_1\%$ on the next m_1 units;

taxation at $t_2\%$ on the next m_2 units;

taxation at $t_3\%$ on anything above.

For some reason, this is termed progressive taxation. Write a generalised program to determine net income after tax deductions. Write out the gross income, the deductions and the net income. You will have to make some realistic estimates of the tax thresholds m_i and the taxation levels t_i . You could use this sort of model to find out how sensitive revenue from taxation was in relation to cosmetic changes in thresholds and tax rates.

8. The specific heat capacity of water is $2009 \text{ J kg}^{-1} \text{ K}^{-1}$; the specific latent heat of fusion (ice/water) is 335 kJ kg^{-1} , and the specific latent heat of vaporization (water/steam) is 2500 kJ kg^{-1} . Assume that the specific heat capacity of ice and steam are identical to that of water. Write a program which will read in two temperatures, and will calculate the energy required to raise (or lower) ice, water or steam at the first temperature, to ice, water or steam at the second. Take the freezing point of water as 273 K , and its boiling point as 373 K . For those happier with Celsius, 0° C is 273 K , while 100° C is 373 K . One calorie is 4.1868 J , and for the truly atavistic, 1 BTU is 1055 J (approximately).

9. Get height and weight measurements for yourself and the other people in the class. Calculate BMI values.

8.16 Bibliography

Dahl O. J., Dijkstra E. W., Hoare C. A. R., Structured Programming, Academic Press, 1972.

This is the original text, and a must. The quote at the start of the chapter by Dijkstra summarises beautifully our limitations when programming and the discipline we must have to successfully master programming.

Knuth D. E., Structured Programming with GOTO Statements, in Current Trends in Programming Methodology, Volume 1, Prentice Hall.

The chapter by Knuth provides a very succinct coverage of the arguments for the adoption of structured programming, and dispells many of the myths concerning the use of the GOTO statement. Highly recommended.

I can call spirits from the vasty deep.
 Why so can I, or so can any man; but will they come
 when you do call for them?

William Shakespeare, King Henry IV, part 1

9 Functions

This is the major step forward in the ability to construct larger programs. We have to have a way of breaking problems down into smaller sub-problems. The terminology varies with programming languages:

- in Fortran we have functions and subroutines;
- in the Algol family of languages we have functions and procedures;
- in the C family we have functions;
- in object oriented programming we have methods;

The basic idea is the same. We start by looking at a small number of user written functions. We will then look at some of the intrinsics.

9.1 Example 1 - a bigger function

```
def bigger(a,b):
    if (a>b):
        return a
    else:
        return b

x=10
y=20
print(" Biggest of {:3d} and {:3d} is {:3d} ".format(x,y,big-
ger(x,y)))
x1=1.0
x2=2.0
print(" Biggest of {:4.2f} and {:4.2f} is {:4.2f} ".for-
mat(x1,x2,bigger(x1,x2)))
```

The function is given below

```
def bigger(a,b):
    if (a>b):
        return a
    else:
        return b
```

and we use the keyword `def` to indicate the start of the function definition. The function is called `bigger` and takes two arguments, `a` and `b`. We test to see if `a` is greater than `b` and if it is we return the value of `a`, else we return the value of `b`.

We call the function twice in the program, once with integer arguments, and once with real arguments. The function is what is called generic or polymorphic in that it can work with arguments of a variety of types.

Here is the output.

```
$ python3 c0901.py
Biggest of 10 and 20 is 20
Biggest of 1.00 and 2.00 is 2.00
```

The function works with both integer and real arguments.

9.2 Example 2 - a swap function

A classic function to implement to test out parameter passing in any language is a swap function.

Here is a traditional implementation style in Python.

```
def swap(a,b):
    t=a
    a=b
    b=t

x=10
y=20
print(" Before {:3d} and {:3d} ".format(x,y))
swap(x,y)
print(" After  {:3d} and {:3d} ".format(x,y))
x1=1.0
x2=2.0
print(" Before {:4.1f} and {:4.1f} ".format(x1,x2))
swap(x1,x2)
print(" After  {:4.1f} and {:4.1f} ".format(x1,x2))
```

Here is the output.

```
$ python3 c0902.py
Before 10 and 20
After 10 and 20
Before 1.0 and 2.0
After 1.0 and 2.0
```

The values for x and y and for x1 and x2 are the same as before the call, i.e. the swap of the values has not taken place. Note again that the function can take arguments of both integer and real type.

9.3 Example 3 - another swap

Here is a working Python implementation.

```
def swap(a,b):
    return b,a

x=10
y=20
print(" Before {:3d} and {:3d} ".format(x,y))
x,y=swap(x,y)
print(" After  {:3d} and {:3d} ".format(x,y))
x1=1.0
x2=2.0
```

```
print(" Before {:4.1f} and {:4.1f} ".format(x1,x2))
x1,x2=swap(x1,x2)
print(" After  {:4.1f} and {:4.1f} ".format(x1,x2))
```

Here is the output.

```
$ python3 c0903.py
Before 10 and 20
After 20 and 10
Before 1.0 and 2.0
After 2.0 and 1.0
```

The swap has now taken place.

9.4 Example 4 - yet another swap

Here is an even shorter implementation.

```
x=10
y=20
print(" Before {:3d} and {:3d} ".format(x,y))
x,y=y,x
print(" After  {:3d} and {:3d} ".format(x,y))
x1=1.0
x2=2.0
print(" Before {:4.1f} and {:4.1f} ".format(x1,x2))
x1,x2=x2,x1
print(" After  {:4.1f} and {:4.1f} ".format(x1,x2))
```

Note - no function!

9.5 Example 5 - recursive functions

Here is the program.

```
def factorial(i):
    if (i==0):
        return 1
    else:
        return i*factorial(i-1)

i=5
print(" Factorial of {:5d} is {:10d} ".format(i,factorial(i)))
```

Here is the output.

```
$ python3 c0905.py
Factorial of      5 is          120
```

The next example is a simple variation where we read the integer number in.

9.6 Example 6 - simple factorial variant, reading the value in

Here is the source code.

```
def factorial(i):
    if (i==0):
        return 1
```

```

else:
    return i*factorial(i-1)

i = int(input(" Type in the number you want to find the fac-
torial of - an integer  "))
print(" Factorial of {:5d} is {:10d} ".format(i,facto-
rial(i)))

```

Here is the output for 50.

```
Type in the number you want to find the factorial of - an integer 50
```

```
Factorial of 50 is
```

```
30414093201713378043612608166064768844377641568960512000000000
000
```

How does this compare with other programming languages you know?

9.7 Intrinsic maths functions

The following information is taken from the library documentation.

9.7.1 math — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.7.1.1 Number-theoretic and representation functions

- | | |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>math.ceil(x)</code> | Return the ceiling of <code>x</code> , the smallest integer greater than or equal to <code>x</code> . If <code>x</code> is not a float, delegates to <code>x.__ceil__()</code> , which should return an Integral value. |
| <code>math.copysign(x, y)</code> | Return a float with the magnitude (absolute value) of <code>x</code> but the sign of <code>y</code> . On platforms that support signed zeros, <code>copysign(1.0, -0.0)</code> returns <code>-1.0</code> . |
| <code>math.fabs(x)</code> | Return the absolute value of <code>x</code> . |
| <code>math.factorial(x)</code> | Return <code>x</code> factorial. Raises <code>ValueError</code> if <code>x</code> is not integral or is negative. |
| <code>math.floor(x)</code> | Return the floor of <code>x</code> , the largest integer less than or equal to <code>x</code> . If <code>x</code> is not a float, delegates to <code>x.__floor__()</code> , which should return an Integral value. |
| <code>math.fmod(x, y)</code> | Return <code>fmod(x, y)</code> , as defined by the platform C library. Note that the Python expression <code>x % y</code> may not return the same result. The intent of the C standard is that <code>fmod(x, y)</code> be exactly (mathematically; to in- |

finite precision) equal to $x - n*y$ for some integer n such that the result has the same sign as x and magnitude less than $\text{abs}(y)$. Python's $x \% y$ returns a result with the sign of y instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be represented exactly as a float, and rounds to the surprising `1e100`. For this reason, function `fmod()` is generally preferred when working with floats, while Python's $x \% y$ is preferred when working with integers.

`math.frexp(x)` Return the mantissa and exponent of x as the pair (m, e) . m is a float and e is an integer such that $x == m * 2**e$ exactly. If x is zero, returns $(0.0, 0)$, otherwise $0.5 \leq \text{abs}(m) < 1$. This is used to “pick apart” the internal representation of a float in a portable way.

`math.fsum(iterable)` Return an accurate floating point sum of values in the iterable. Avoids loss of precision by tracking multiple intermediate partial sums:

```
>>>>> sum([.1, .1, .1, .1, .1, .1, .1, .1, .1])
0.9999999999999999
>>> fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1])
1.0
```

The algorithm's accuracy depends on IEEE-754 arithmetic guarantees and the typical case where the rounding mode is half-even. On some non-Windows builds, the underlying C library uses extended precision addition and may occasionally double-round an intermediate sum causing it to be off in its least significant bit.

For further discussion and two alternative approaches, see the ASPN cookbook recipes for accurate floating point summation.

`math.gcd(a, b)` Return the greatest common divisor of the integers a and b . If either a or b is nonzero, then the value of `gcd(a, b)` is the largest positive integer that divides both a and b . `gcd(0, 0)` returns 0.

`math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)`

Return True if the values a and b are close to each other and False otherwise.

Whether or not two values are considered close is determined according to given absolute and relative tolerances.

`rel_tol` is the relative tolerance – it is the maximum allowed difference between a and b , relative to the larger absolute value of a or b . For example, to set a tolerance of 5%, pass `rel_tol=0.05`. The default tolerance is `1e-09`, which assures that the two values are the same within about 9 decimal digits. `rel_tol` must be greater than zero.

`abs_tol` is the minimum absolute tolerance – useful for comparisons near zero. `abs_tol` must be at least zero.

If no errors occur, the result will be: $\text{abs}(a-b) \leq \max(\text{rel_tol} * \max(\text{abs}(a), \text{abs}(b)), \text{abs_tol})$.

The IEEE 754 special values of NaN, inf, and -inf will be handled according to IEEE rules. Specifically, NaN is not considered close to any other value, including NaN. inf and -inf are only considered close to themselves.

`math.isfinite(x)`

	Return True if x is neither an infinity nor a NaN, and False otherwise. (Note that 0.0 is considered finite.)
<code>math.isinf(x)</code>	Return True if x is a positive or negative infinity, and False otherwise.
<code>math.isnan(x)</code>	Return True if x is a NaN (not a number), and False otherwise.
<code>math.ldexp(x, i)</code>	Return $x * (2^{**i})$. This is essentially the inverse of function <code>frexp()</code> .
<code>math.modf(x)</code>	Return the fractional and integer parts of x . Both results carry the sign of x and are floats.
<code>math.trunc(x)</code>	Return the Real value x truncated to an Integral (usually an integer). Delegates to <code>x.__trunc__()</code> .

Note that `frexp()` and `modf()` have a different call/return pattern than their C equivalents: they take a single argument and return a pair of values, rather than returning their second return value through an ‘output parameter’ (there is no such thing in Python).

For the `ceil()`, `floor()`, and `modf()` functions, note that all floating-point numbers of sufficiently large magnitude are exact integers. Python floats typically carry no more than 53 bits of precision (the same as the platform C double type), in which case any float x with `abs(x) >= 2**52` necessarily has no fractional bits.

9.7.1.2 Power and logarithmic functions

<code>math.exp(x)</code>	Return e^{**x} .
<code>math.expm1(x)</code>	Return $e^{**x} - 1$. For small floats x , the subtraction in <code>exp(x) - 1</code> can result in a significant loss of precision; the <code>expm1()</code> function provides a way to compute this quantity to full precision:

```
>>>>> from math import exp, expm1
>>> exp(1e-5) - 1 # gives result accurate to 11 places
1.0000050000069649e-05
>>> expm1(1e-5) # result accurate to full precision
1.0000050000166668e-05
```

`math.log(x[, base])` With one argument, return the natural logarithm of x (to base e).

With two arguments, return the logarithm of x to the given base, calculated as `log(x)/log(base)`.

<code>math.log1p(x)</code>	Return the natural logarithm of $1+x$ (base e). The result is calculated in a way which is accurate for x near zero.
<code>math.log2(x)</code>	Return the base-2 logarithm of x . This is usually more accurate than <code>log(x, 2)</code> .
<code>math.log10(x)</code>	Return the base-10 logarithm of x . This is usually more accurate than <code>log(x, 10)</code> .
<code>math.pow(x, y)</code>	Return x raised to the power y . Exceptional cases follow Annex ‘F’ of the C99 standard as far as possible. In particular, <code>pow(1.0, x)</code> and <code>pow(x, 0.0)</code> always return 1.0, even when x is a zero or a NaN. If both x and y are finite, x is negative, and y is not an integer then <code>pow(x, y)</code> is undefined, and raises <code>ValueError</code> .

Unlike the built-in `**` operator, `math.pow()` converts both its arguments to type float. Use `**` or the built-in `pow()` function for computing exact integer powers.

<code>math.sqrt(x)</code>	Return the square root of x .
---------------------------	---------------------------------

9.7.1.3 Trigonometric functions

<code>math.acos(x)</code>	Return the arc cosine of x , in radians.
<code>math.asin(x)</code>	Return the arc sine of x , in radians.
<code>math.atan(x)</code>	Return the arc tangent of x , in radians.
<code>math.atan2(y, x)</code>	Return <code>atan(y / x)</code> , in radians. The result is between $-\pi$ and π . The vector in the plane from the origin to point (x, y) makes this angle with the positive X axis. The point of <code>atan2()</code> is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, <code>atan(1)</code> and <code>atan2(1, 1)</code> are both $\pi/4$, but <code>atan2(-1, -1)</code> is $-3\pi/4$.
<code>math.cos(x)</code>	Return the cosine of x radians.
<code>math.hypot(x, y)</code>	Return the Euclidean norm, $\sqrt{x^2 + y^2}$. This is the length of the vector from the origin to point (x, y) .
<code>math.sin(x)</code>	Return the sine of x radians.
<code>math.tan(x)</code>	Return the tangent of x radians.

9.7.1.4 Angular conversion

<code>math.degrees(x)</code>	Convert angle x from radians to degrees.
<code>math.radians(x)</code>	Convert angle x from degrees to radians.

9.7.1.5 Hyperbolic functions

Hyperbolic functions are analogs of trigonometric functions that are based on hyperbolas instead of circles.

<code>math.acosh(x)</code>	Return the inverse hyperbolic cosine of x .
<code>math.asinh(x)</code>	Return the inverse hyperbolic sine of x .
<code>math.atanh(x)</code>	Return the inverse hyperbolic tangent of x .
<code>math.cosh(x)</code>	Return the hyperbolic cosine of x .
<code>math.sinh(x)</code>	Return the hyperbolic sine of x .
<code>math.tanh(x)</code>	Return the hyperbolic tangent of x .

9.7.1.6 Special functions

<code>math.erf(x)</code>	Return the error function at x .
--------------------------	------------------------------------

The `erf()` function can be used to compute traditional statistical functions such as the cumulative standard normal distribution:

```
def phi(x):
    'Cumulative distribution function for the standard normal distribution'
    return (1.0 + erf(x / sqrt(2.0))) / 2.0
```

<code>math.erfc(x)</code>	Return the complementary error function at x . The complementary error function is defined as $1.0 - \text{erf}(x)$. It is used for large values of x where a subtraction from one would cause a loss of significance.
<code>math.gamma(x)</code>	Return the Gamma function at x .
<code>math.lgamma(x)</code>	Return the natural logarithm of the absolute value of the Gamma function at x .

9.7.1.7 Constants

<code>math.pi</code>	The mathematical constant $\pi = 3.141592\dots$, to available precision.
----------------------	---------------------------------------------------------------------------

<code>math.e</code>	The mathematical constant $e = 2.718281\dots$, to available precision.
<code>math.inf</code>	A floating-point positive infinity. (For negative infinity, use <code>-math.inf</code> .) Equivalent to the output of <code>float('inf')</code> .
<code>math.nan</code>	A floating-point “not a number” (NaN) value. Equivalent to the output of <code>float('nan')</code> .

CPython implementation detail: The `math` module consists mostly of thin wrappers around the platform C math library functions. Behavior in exceptional cases follows Annex F of the C99 standard where appropriate. The current implementation will raise `ValueError` for invalid operations like `sqrt(-1.0)` or `log(0.0)` (where C99 Annex F recommends signalling invalid operation or divide-by-zero), and `OverflowError` for results that overflow (for example, `exp(1000.0)`). A NaN will not be returned from any of the functions above unless one or more of the input arguments was a NaN; in that case, most functions will return a NaN, but (again following C99 Annex F) there are some exceptions to this rule, for example `pow(float('nan'), 0.0)` or `hypot(float('nan'), float('inf'))`.

Note that Python makes no effort to distinguish signalling NaNs from quiet NaNs, and behavior for signalling NaNs remains unspecified. Typical behavior is to treat all NaNs as though they were quiet.

9.8 Example 7 - testing out the maths functions

The following program tests out the maths functions.

```
import math
i=1
a=10
b=20
p=1.000000000001
q=1.000000000002
x=1.0
y=2.0
print("math.ceil(x) = " , math.ceil(x))
print("math.copysign(x, y) = " , math.copysign(x, y))
print("math.fabs(x) = " , math.fabs(x))
print("math.factorial(x) = " , math.factorial(x))
print("math.floor(x) = " , math.floor(x))
print("math.fmod(x, y) = " , math.fmod(x, y))
print("math.frexp(x) = " , math.frexp(x))
print("math.gcd(a, b) = " , math.gcd(a, b))
print("math.isclose(p, q) = " , math.isclose(p, q))
print("math.isfinite(x) = " , math.isfinite(x))
print("math.isinf(x) = " , math.isinf(x))
print("math.isnan(x) = " , math.isnan(x))
print("math.ldexp(x, i) = " , math.ldexp(x, i))
print("math.modf(x) = " , math.modf(x))
# print("math.remainder(x, y) = " , math.remainder(x, y))
print("math.trunc(x) = " , math.trunc(x))
print("math.exp(x) = " , math.exp(x))
print("math.expml(x) = " , math.expml(x))
print("math.log(x) = " , math.log(x))
print("math.log1p(x) = " , math.log1p(x))
```

```

print("math.log2(x) = " , math.log2(x))
print("math.log10(x) = " , math.log10(x))
print("math.pow(x, y) = " , math.pow(x, y))
print("math.sqrt(x) = " , math.sqrt(x))
print("math.acos(x) = " , math.acos(x))
print("math.asin(x) = " , math.asin(x))
print("math.atan(x) = " , math.atan(x))
print("math.atan2(y, x) = " , math.atan2(y, x))
print("math.cos(x) = " , math.cos(x))
print("math.hypot(x, y) = " , math.hypot(x, y))
print("math.sin(x) = " , math.sin(x))
print("math.tan(x) = " , math.tan(x))
print("math.degrees(x) = " , math.degrees(x))
print("math.radians(x) = " , math.radians(x))
print("math.acosh(x) = " , math.acosh(x))
print("math.asinh(x) = " , math.asinh(x))
# print("math.atanh(x)) = " , math.atanh(x))
print("math.cosh(x) = " , math.cosh(x))
print("math.sinh(x) = " , math.sinh(x))
print("math.tanh(x) = " , math.tanh(x))
print("math.erf(x) = " , math.erf(x))
print("math.erfc(x) = " , math.erfc(x))
print("math.gamma(x) = " , math.gamma(x))
print("math.lgamma(x) = " , math.lgamma(x))
print("math.pi = " , math.pi)
print("math.e = " , math.e)
print("math.tau = " , math.tau)
print("math.inf = " , math.inf)
print("math.nan = " , math.nan)
print(math.e)

```

Here is the output.

```

math.ceil(x) = 1
math.copysign(x, y) = 1.0
math.fabs(x) = 1.0
math.factorial(x) = 1
math.floor(x) = 1
math.fmod(x, y) = 1.0
math.frexp(x) = (0.5, 1)
math.gcd(a, b) = 10
math.isclose(p,q) = True
math.isfinite(x) = True
math.isinf(x) = False
math.isnan(x) = False
math.ldexp(x, i) = 2.0
math.modf(x) = (0.0, 1.0)
math.trunc(x) = 1
math.exp(x) = 2.718281828459045
math.expm1(x) = 1.718281828459045
math.log(x) = 0.0

```



```

math.log1p(x) = 0.6931471805599453
math.log2(x) = 0.0
math.log10(x) = 0.0
math.pow(x, y) = 1.0
math.sqrt(x) = 1.0
math.acos(x) = 0.0
math.asin(x) = 1.5707963267948966
math.atan(x) = 0.7853981633974483
math.atan2(y, x) = 1.1071487177940904
math.cos(x) = 0.5403023058681397
math.hypot(x, y) = 2.23606797749979
math.sin(x) = 0.8414709848078965
math.tan(x) = 1.5574077246549023
math.degrees(x) = 57.29577951308232
math.radians(x) = 0.017453292519943295
math.acosh(x) = 0.0
math.asinh(x) = 0.8813735870195429
math.cosh(x) = 1.5430806348152437
math.sinh(x) = 1.1752011936438014
math.tanh(x) = 0.7615941559557649
math.erf(x) = 0.842700792949715
math.erfc(x) = 0.157299207050285
math.gamma(x) = 1.0
math.lgamma(x) = 0.0
math.pi = 3.141592653589793
math.e = 2.718281828459045
math.tau = 6.283185307179586
math.inf = inf
math.nan = nan

```

This is consistent with the use of the C run time maths library and gives double as the underlying type. What about the two commented out lines?

9.9 Example 8 - math module sin function

The example uses the plain Python array type to hold a set of angles in degrees. We use a couple of the build in math modules functions to print out the sines of these angles.

```

import math
import array
angles=array.array('I', [-1, 0, 1, 29, 30, 31, 44, 45, 46, 59, 60, 61, 89, 90, 91])
print(type(angles))
l=len(angles)
for i in range(0,l):
    print(" {:4d}  {:20.16f} ".format(angles[i],
math.sin(math.radians(angles[i]))))

```

Here is the output.

```

<class 'array.array'>
-1    -0.0174524064372835
 0     0.0000000000000000

```

```

1      0.0174524064372835
29     0.4848096202463371
30     0.4999999999999999
31     0.5150380749100542
44     0.6946583704589973
45     0.7071067811865475
46     0.7193398003386512
59     0.8571673007021123
60     0.8660254037844386
61     0.8746197071393957
89     0.9998476951563913
90     1.0000000000000000
91     0.9998476951563913

```

9.10 Example 9 - math module using numpy arrays

This is a variant of the previous using numpy arrays instead.

```

import math
import numpy
angles=numpy.array(
[-1,0,1,29,30,31,44,45,46,59,60,61,89,90,91])
print(type(angles))
l=len(angles)
for i in range(0,l):
    print(" {:4d}  {:20.16f} ".format(angles[i],
math.sin(math.radians(angles[i]))))

```

Here is the output.

```

<class 'numpy.ndarray'>
-1     -0.0174524064372835
 0      0.0000000000000000
 1      0.0174524064372835
29      0.4848096202463371
30      0.4999999999999999
31      0.5150380749100542
44      0.6946583704589973
45      0.7071067811865475
46      0.7193398003386512
59      0.8571673007021123
60      0.8660254037844386
61      0.8746197071393957
89      0.9998476951563913
90      1.0000000000000000
91      0.9998476951563913

```

The output is the same as in the previous example.

9.11 Example 10 - math module using a pi shortcut

In this example we use introduce the idea of a short cut for math.pi. We also introduce the idea of a main program method.

```

import math

PI = math.pi

def area_of_circle(radius):
    area=PI*radius**2
    return(area)

def main():

    def read_radius():
        r = float(input(" Type in the radius ? "))
        return(r)

    r=read_radius()

    print( " Area of circle is {0:10.4f} ".format(
area_of_circle(r) ) )

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

Type in the radius ? 10
Area of circle is      314.1593

```

Note the following

The statement `PI = math.pi` provides with a shorthand way of referencing the mathematical constant `pi` in our Python program;

We have a `main()` function which has an internal function `read_radius()`

The local variable `r` in `read_radius()` is internal to the function;

The variable `r` in the `main()` function is local to the `main()` function and independent of the variable `r` in the `read_radius()` function;

In the examples we have seen so far we have implicitly used the `main()` function in Python. In this example we make explicit the fact that we have a `main()` function;

The following code

```

if ( __name__ == "__main__" ):
    main()

```

calls the `main()` function, to start the execution. The `if` test returns true if the module is executed by the interpreter (which it is in this case). The `if` test returns false when the module is imported into another module.

9.12 Fibonacci implementations

Here we look at three implementations of the Fibonacci series.

9.13 Example 11 - Using generators

```
# imperative
# generators

def fibonacci(n, first=0, second=1):
    for i in range(n):
        yield first # Return current iteration
        first, second = second, first + second

print([x for x in fibonacci(10)])
```

9.14 Example 12 - Iterative

```
# iterative

def fibonacci(n):
    first, second = 0, 1
    for i in range(n):
        print(first) # Print current iteration
        first, second = second, first + second #Calculate
next values

fibonacci(10)
```

9.15 Example 13 - Recursive

```
# recursive

def fibonacci(n, first=0, second=1):
    if n == 1:
        return [first]
    else:
        return [first] + fibonacci(n - 1, second, first +
second)

print(fibonacci(10))
```

Run these three examples.

9.16 Functional programming in Python

There is an appendix on functional programming for people without any background in this area. Functional programming is one of several programming paradigms supported by Python. Simplistically functional programming decomposes a problem into a set of functions. Functions only take inputs and produce outputs, and don't have any internal state that affects the output produced for a given input. Some well known functional languages include the ML family (Standard ML, OCaml, and other variants) and Haskell. Here are some sources.

<https://docs.python.org/2/howto/functional.html>
<http://www.ibm.com/developerworks/library/l-prog/index.html>

Python has a number of tools that are useful in functional programming:

```

map(function,list)
filter(function,list)
reduce(function,list)
lambda
list comprehension

```

A for loop can be replaced with a map function.

List comprehensions enable us to build lists in a simple fashion.

Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called "lambda". This is not exactly the same as lambda in functional programming languages, but it is a concept that's well integrated into Python and is often used in conjunction with typical functional concepts like filter(), map() and reduce().

9.17 Example 14 - generating prime numbers

Here is a list comprehension example that generates prime numbers.

```

noprimes = [j for i in range(2, 10) for j in range(i*2, 100,
i)]
primes = [x for x in range(2, 100) if x not in noprimes]
print(primes)

```

Here is the output.

```

$ python3 list_comprehension_01.py
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

```

9.18 Example 15 - list and lambda usage

Here is a combined list comprehension and lambda example.

```

words = 'The quick brown fox jumps over the lazy
dog'.split()
print("\n words are",end=" ")
print(words)
print("\n\n List comprehension solution\n\n")
# list comprehension solution
stuff = [[w.upper(), w.lower(), len(w)] for w in words]
for i in stuff:
    print(i)
print("\n\n Lambda solution")
# lambda solution
print("\n\n")
stuff = map(lambda w: [w.upper(), w.lower(), len(w)], words)
for i in stuff:
    print(i)

```

Here is the output.

```

$ python3 list_comprehension_02.py

```

```
words are ['The', 'quick', 'brown', 'fox', 'jumps', 'over',
'the', 'lazy', 'dog']
```

List comprehension solution

```
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

Lambda solution

```
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

Here are some more examples.

9.19 Example 16 - functional example

Here is the first example.

```
x = [1,2,3]
print(" \n for i in x          ",end=" ")
for i in x:
    print(i,end=" ")
print(" \n for i in iter(x)    ",end=" ")
for i in iter(x):
    print(i,end=" ")
print(" \n list(x_iterator))   ",end=" ")
x_iterator = iter(x)
print(list(x_iterator))
print(" list(x)                ",end=" ")
print(list(x))
square      = lambda y: y*y
cube        = lambda y: y*y*y
```

```

reciprocal = lambda y: 1/y
z=map(square,x)
print("\n Square          ",end=" ")
print(list(z))
z=map(cube,x)
print(" Cube          ",end=" ")
print(list(z))
z=map(reciprocal,x)
print(" Reciprocal      ",end=" ")
print(list(z))

```

Here is the output.

```
python3 fun_01.py
```

```

for i in x          1 2 3
for i in iter(x)   1 2 3
list(x_iterator)  [1, 2, 3]
list(x)           [1, 2, 3]

Square            [1, 4, 9]
Cube              [1, 8, 27]
Reciprocal       [1.0, 0.5, 0.3333333333333333]

```

We have three functions, square, cube and reciprocal.

We then use the map statement to invoke the functions on some data.

9.20 Example 17 - functional example

Here is the second example.

```

days = ("Monday","Tuesday","Wednesday","Thursday","Friday",
"Saturday","Sunday")
for day in days:
    print(day)
print(days)
uppercase = lambda x: x.upper()
DAYS=map(uppercase,days)
print(list(DAYS))

```

Here is the output.

```

$ python3 fun_02.py
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday')
['MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY', 'FRIDAY',
'SATURDAY', 'SUNDAY']

```

9.21 Example 18 - functional example variant using the array module

Here is the third example. It is a simple variant of 1, using the Python intrinsic array module.

```
import array
x = array.array('I', [1,2,3])
print(" \n for i in x          ",end=" ")
for i in x:
    print(i,end=" ")
print(" \n for i in iter(x)    ",end=" ")
for i in iter(x):
    print(i,end=" ")
print(" \n list(x_iterator))  ",end=" ")
x_iterator = iter(x)
print(list(x_iterator))
print(" list(x)              ",end=" ")
print(list(x))
square      = lambda y: y*y
cube        = lambda y: y*y*y
reciprocal  = lambda y: 1/y
z=map(square,x)
print("\n Square            ",end=" ")
print(list(z))
z=map(cube,x)
print(" Cube                ",end=" ")
print(list(z))
z=map(reciprocal,x)
print(" Reciprocal         ",end=" ")
print(list(z))
```

The output is the same as the first example.

9.22 Example 19 - functional variant using the numpy module

Here is the fourth example. This is a variant of the last using the Numpy module.

```
import numpy
x = numpy.array([1,2,3])
print(" \n for i in x          ",end=" ")
for i in x:
    print(i,end=" ")
print(" \n for i in iter(x)    ",end=" ")
for i in iter(x):
    print(i,end=" ")
print(" \n list(x_iterator))  ",end=" ")
x_iterator = iter(x)
print(list(x_iterator))
print(" list(x)              ",end=" ")
print(list(x))
square      = lambda y: y*y
```



```

cube          = lambda y: y*y*y
reciprocal    = lambda y: 1/y
z=map(square,x)
print("\n Square          ",end=" ")
print(list(z))
z=map(cube,x)
print(" Cube              ",end=" ")
print(list(z))
z=map(reciprocal,x)
print(" Reciprocal        ",end=" ")
print(list(z))

```

The output is the same as the first and third examples.

9.23 Problems

1. Write a program that calculates the sine, cosine and tangent of angles between -1 and 91 degrees, at one degree intervals.
2. Write a program that reads in the lengths a and b of two sides of a right angled triangle. Calculate the hypotenuse c. Use the sqrt function.
3. Write a program that will read in the lengths a and b of two sides of a triangle and the angle in between them (in degrees). Calculate the third size c using the cosine rule.

$$c^2 = a^2 + b^2 - 2ab\cos$$

4. Stirling's approximation for large n is given by

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Write a function in Python for this function and also write a program to test out the function for a value of 50. Do the results agree with the earlier example?

Russell's theory of types leads to certain complexities in the foundations of mathematics,... Its interesting features for our purposes are that types are used to prevent certain erroneous expressions from being used in logical and mathematical formulae; and that a check against violation of type constraints can be made purely by scanning the text, without any knowledge of the value which a particular symbol might happen to have.

C. A. R. Hoare, Structured Programming.

It is said that Lisp programmers know that memory management is so important that it cannot be left to the users and C programmers know that memory management is so important that it cannot be left to the system.

anon

10 Object oriented programming and classes in Python

User defined data types are an essential part of general purpose programming languages. Early languages provided this functionality via concrete data types, later languages by abstract data types.

Python provides the ability to program user defined types using classes. In this chapter we will look at some simple examples highlighting the Python syntax.

10.1 Example 1 - base shape class

Here is the source code.

```
#
# c1001.py
#
class shape:

    def __init__(self,x,y):
        self.x=x
        self.y=y

    def get_x(self):
        return (self.x)

    def get_y(self):
        return (self.y)

    def set_x(self,x):
        self.x=x

    def set_y(self,y):
        self.y=y

    def draw(self):
        print(" x = {0:4d}".format(self.x))
        print(" y = {0:4d}".format(self.y))
```

```
def main():

    s=shape(10,20)
    s.draw()
    s.set_x(100)
    s.set_y(200)
    s.draw()
    print(s.get_x())
    print(s.get_y())

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
x = 10
y = 20
x = 100
y = 200
100
200
```

Note that in this example we have the source code all in one file. We can use shape as a constructor name.

10.2 Example 2 - variation using modules

In this example we have a simple variant of the above using modules.

Here is the source code for the shape module.

```
#
# shape.py
#
class shape:

    def __init__(self,x,y):
        self.x=x
        self.y=y

    def get_x(self):
        return (self.x)

    def get_y(self):
        return (self.y)

    def set_x(self,x):
        self.x=x

    def set_y(self,y):
        self.y=y

    def draw(self):
```

```

    print(" x = {0:4d}".format(self.x))
    print(" y = {0:4d}".format(self.y))

```

The file is called `shape.py`.

Here is the source code for the test program.

```

#
# c1002.py
#
import shape

def main():

    s=shape.shape(10,20)
    s.draw()
    s.set_x(100)
    s.set_y(200)
    s.draw()
    print(s.get_x())
    print(s.get_y())

if ( __name__ == "__main__" ):
    main()

```

Here is the output from running the test program.

```

x =    10
y =    20
x =   100
y =   200
100
200

```

Note the syntax to invoke the constructor. We can't use `shape` on its own. Compare this to the first example.

10.3 Example 3 - a circle derived class

Here is the source code for the circle class.

```

#
# circle.py
#
import shape

class circle(shape.shape):

    def __init__(self,x,y,r):
        self.x=x
        self.y=y
        self.r=r

    def get_radius(self):
        return(self.radius)

```

```
def set_radius(self,r):
    self.radius=r

def draw(self):
    print(" x = {0:4d}".format(self.x))
    print(" y = {0:4d}".format(self.y))
    print(" r = {0:4d}".format(self.r))
```

Here is the test program.

```
#
# c1003.py
#

import circle

def main():

    c=circle.circle(10,20,30)
    c.draw()
    c.set_x(100)
    c.set_y(200)
    c.set_radius(300)
    c.draw()
    print(c.get_x())
    print(c.get_y())
    print(c.get_radius())

if ( __name__ == "__main__" ):
    main()
```

10.4 Example 4 - test program for the shape and circle classes

Here is the complete source code.

```
#
# c1004.py
#
import shape
import circle

def main():

    s=shape.shape(10,20)
    s.draw()
    s.set_x(100)
    s.set_y(200)
    s.draw()
    print(s.get_x())
    print(s.get_y())
```

```

c=circle.circle(100,200,300)
c.draw()
c.set_x(111)
c.set_y(222)
c.set_radius(333)
c.draw()
print(c.get_x())
print(c.get_y())
print(c.get_radius())

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

x = 10
y = 20
x = 100
y = 200
100
200
x = 100
y = 200
r = 300
x = 111
y = 222
r = 300
111
222
333

```

10.5 Example 5 - polymorphism and dynamic binding

This example illustrates polymorphism and dynamic binding in Python. Here is the code.

```

#
# c1005.py
#
import shape
import circle

def main():

    s=shape.shape(10,20)
    c=circle.circle(100,200,300)

    shape_array = []

    shape_array.append(s)
    shape_array.append(c)

    for i in range(0,2):

```

```
        shape_array[i].draw()

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
x = 10
y = 20
x = 100
y = 200
r = 300
```

10.6 Example 6 - data structuring using the Met Office data

This example looks at creating an example in Python that works with the Met Office historic data.

Here is the program source.

```
#
# weather_01
#

class weather:

# CONSTRUCTOR

    def __init__(self, year, month, tmax, tmin, af_days, rain, sun) :
        self.year=year
        self.month=month
        self.tmax = tmax
        self.tmin = tmin
        self.af_days = af_days
        self.rain = rain
        self.sun = sun

# GETTERS

    def get_year(self):
        return (self.year)

    def get_month(self):
        return (self.month)

    def get_tmax(self):
        return (self.tmax)

    def get_tmin(self):
        return (self.tmin)

    def get_af_days(self):
```

```

    return (self.af_days)

def get_rain(self):
    return (self.rain)

def get_sun(self):
    return (self.sun)

# SETTERS

def set_year(self, year):
    self.year=year

def set_month(self, month):
    self.month=month

def set_tmax(self, tmax):
    self.tmax=tmax

def set_tmin(self, tmin):
    self.tmin=tmin

def set_af_days(self, af_days):
    self.af_days=af_days

def set_rain(self, rain):
    self.rain=rain

def set_sun(self, sun):
    self.sun=sun

# DISPLAY

def display(self):
    print(" {0:4d}".format(self.year),end=" ")
    print(" {0:4d}".format(self.month),end=" ")
    print(" {0:4.1f}".format(self.tmax),end=" ")
    print(" {0:4.1f}".format(self.tmin),end=" ")
    print(" {0:3d}".format(self.af_days),end=" ")
    print(" {0:5.1f}".format(self.rain),end=" ")
    print(" {0:5.1f}".format(self.sun))

def display_heading(self):
    print(" Year Month      tmax      tmin      af      rain
sun")

    print("          degC      degC      days      mm
hours")

```



```
#
# End of class
#

def main():

    s=weather(2018,1,20.0,10.0,5,100.0,5.0)
    print("Using object methods\n")
    s.display_heading()
    s.display()
    print("\nUsing getters\n")
    print(" {0:4d}".format( s.get_year() ),end=" ")
    print(" {0:4d}".format( s.get_month() ),end=" ")
    print(" {0:4.1f}".format( s.get_tmax() ),end=" ")
    print(" {0:4.1f}".format( s.get_tmin() ),end=" ")
    print(" {0:3d}".format( s.get_af_days() ),end=" ")
    print(" {0:5.1f}".format( s.get_rain() ),end=" ")
    print(" {0:5.1f}".format( s.get_sun() ) )

if ( __name__ == "__main__" ):
    main()
```

Note that this version says nothing about the type of the data components in the Met Office data records. We will look at an improved way of working with this data using the strong data typing facilities provided by the numpy class in the next chapter.

10.7 Problems

1. Compile and run the examples.
2. Add a rectangle class to the third and fourth examples.

Common sense is the best distributed commodity in the world, for every man is convinced that he is well supplied with it.

Descartes.

11 IO

In this chapter we will look at the the facilities in Python for input and output. The following is taken from the on line documentation

The `io` module provides Python's main facilities for dealing with various types of I/O. There are three main types of I/O: text I/O, binary I/O and raw I/O. These are generic categories, and various backing stores can be used for each of them. A concrete object belonging to any of these categories is called a file object. Other common terms are stream and file-like object.

Independently of its category, each concrete stream object will also have various capabilities: it can be read-only, write-only, or read-write. It can also allow arbitrary random access (seeking forwards or backwards to any location), or only sequential access (for example in the case of a socket or pipe).

All streams are careful about the type of data you give to them. For example giving a `str` object to the `write()` method of a binary stream will raise a `TypeError`. So will giving a `bytes` object to the `write()` method of a text stream.

We start by providing a small number of examples.

11.1 Example 1 - reading from a file using substrings

This example is taken from the chapter on strings. The key is the open statement, where we link an internal variable with an external file. We then use the `readline` method.

```
import numpy as np
data_file="cwmystwythdata.txt"
nmonths=12
cmsum=0.0
imperial_sum      = 0.0
imperial_average = 0.0
x = np.empty([nmonths] , dtype=np.float64)
f=open(data_file)
print(" ** Skipping header lines ** \n")
for i in range(0,7):
    line=f.readline()
print(" ** Skipping 1959 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1960 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1961 ** \n")
for i in range(0,9):
    line=f.readline()
print(" ** Skipping 1962 ** \n")
```

```

for i in range(0,8):
    line=f.readline()
print(" ** Skipping 1963 ** \n")
for i in range(0,10):
    line=f.readline()
print(" ** Reading 1964 ** \n")
for i in range(0,12):
    line=f.readline()
    x[i]=(float)(line[36:42])
    print(" {0:6.1f} ".format(x[i]))
print(" ** mms ** ")
print(" {0:6.1f} ".format(x.sum()))
print(" {0:6.1f} ".format((x.sum()/nmonths)))
imperial_sum=x.sum()/25.4
imperial_average=imperial_sum/nmonths
print(" ** inches ** ")
print(" {0:6.1f} ".format(imperial_sum))
print(" {0:6.1f} ".format(imperial_average))

```

Here is the output.

```

$ ** Skipping header lines **

** Skipping 1959 **

** Skipping 1960 **

** Skipping 1961 **

** Skipping 1962 **

** Skipping 1963 **

** Reading 1964 **

    83.1
    38.5
    67.3
    76.4
    90.4
    83.5
    177.0
    180.5
    66.0
    171.9
    174.5
    334.8
** mms **
1543.9
    128.7
** inches **

```

```
60.8
5.1
```

So we are using the `file.readline` method.

11.2 Example 2 - reading the same file using the `split()` method

Here is the source file.

```
import numpy as np
data_file="cwmystwythdata.txt"
nmonths=12
cmsum=0.0
imperial_sum      = 0.0
imperial_average = 0.0
x = np.empty([nmonths] , dtype=np.float64)
f=open(data_file)
print(" ** Skipping header lines ** \n")
for i in range(0,7):
    line=f.readline()
print(" ** Skipping 1959 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1960 ** \n")
for i in range(0,12):
    line=f.readline()
print(" ** Skipping 1961 ** \n")
for i in range(0,9):
    line=f.readline()
print(" ** Skipping 1962 ** \n")
for i in range(0,8):
    line=f.readline()
print(" ** Skipping 1963 ** \n")
for i in range(0,10):
    line=f.readline()
print(" ** Reading 1964 ** \n")
for i in range(0,12):
    line=f.readline()
    r=0
    columns=line.split()
    for data in columns:
        if (r==5):
            x[i]=(float)(data)
            print(" {0:6.1f} ".format(x[i]))
        r=r+1
print(" ** mms ** ")
print(" {0:6.1f} ".format(x.sum()))
print(" {0:6.1f} ".format((x.sum()/nmonths)))
imperial_sum=x.sum()/25.4
imperial_average=imperial_sum/nmonths
print(" ** inches ** ")
print(" {0:6.1f} ".format(imperial_sum))
```

```
print(" {0:6.1f} ".format(imperial_average))
```

The output is as in the previous example.

11.3 Example 3 - internet file read

This example looks at reading a file that exists on the UK Met Office Historic Data site. Visit

<http://www.metoffice.gov.uk/public/weather/climate-historic/#?tab=climateHistoric>

for more information.

You may need to change the web address if the Met Office move things around. Here is a valid address as of January 2017.

<http://www.metoffice.gov.uk/pub/data/weather/uk/climate/stationdata/cwmystwythdata.txt>

We will read one data file. Being Welsh I have chosen the Cwmystwyth site. Here is the program.

```
import time
import requests

def main():
    start_time=time.time()
    print(" ** Start time                **",end=" ")
    print(start_time)
    cwmystwyth_data=requests.get("http://www.metoffice.gov.uk/climate/uk/stationdata/cwmystwythdata.txt").text
    print(cwmystwyth_data)
    t1=time.time()
    file_read=t1-start_time
    print(" ** Internet file read took    **",end=" ")
    print(" {0:12.6f}".format(file_read))

if ( __name__ == "__main__" ):
    main()
```

Here is an extract of the output.

2009	12	5.2	-0.2	13	167.7	36.7
2010	1	3.4	-2.3	22	127.9	32.9
2010	2	4.8*	-1.6*	19*	70.4*	72.2*
2010	3	8.7	0.8	16	102.0	119.3
2010	4	13.0	3.8	4	56.8	194.3
2010	5	14.2	4.7	4	71.5	207.3
2010	6	18.8	8.0	0	80.5	220.0
2010	7	17.3	11.9	0	209.3	80.3
2010	8	16.6	9.3	0	88.8	130.5
2010	9	16.3	8.7	1	181.2	135.5*
2010	10	12.5*	5.2*	5*	108.0	117.2*
2010	11	7.1*	0.5*	11*	154.9*	73.3*
2010	12	3.1	-3.7	23	82.6	52.4*
2011	1	5.8	-0.3	16	191.4	44.7
2011	2	8.3	3.1	5	165.8	43.5
2011	3	10.3	1.4	12	35.5	145.0

Site closed

```
** Internet file read took **          0.179377
```

11.4 Example 4 - variation on the internet file read where we save the file

Here is the source.

```
import time
import requests

def main():
    start_time=time.time()
    print(" ** Start time          **",end=" ")
    print(start_time)
    datafile="cwmystwyth.txt"
    f=open(datafile,"w")
    cwmystwyth_data=re-
quests.get("http://www.metoffice.gov.uk/cli-
mate/uk/stationdata/cwmystwythdata.txt").text
    f.write(cwmystwyth_data)
    t1=time.time()
    file_read=t1-start_time
    print(" ** Internet file read took **",end=" ")
    print(" {0:12.6f}".format(file_read))

if ( __name__ == "__main__" ):
```

```
main()
```

Timing is similar to the previous.

11.5 Example 5 - reading all of the station data files with timing

Here is the source.

```
import time
import requests

def main():

    start_time=time.time()
    print(" ** Start time                **",end=" ")
    print(start_time)

    n_stations = 37

    base_address =
"http://www.metoffice.gov.uk/pub/data/weather/uk/cli-
mate/stationdata/"

    station_names = ["aberporthdata.txt"
"armaghdata.txt"
"bradforddata.txt"
"cambornedata.txt"
"cambridgedata.txt"
"chivenordata.txt"
"cwmystwytthdata.txt"
"durhamdata.txt"
"eastbournedata.txt"
"heathrowdata.txt"
"hurndata.txt"
"leucharsdata.txt"
"lowestoftdata.txt"
"nairndata.txt"
"newtonriggdata.txt"
"paisleydata.txt"
"ringwaydata.txt"
"shawburydata.txt"
"sheffielddata.txt"
"stornowaydata.txt"
"suttonboningtondata.txt"
"valleydata.txt"
"waddingtondata.txt"
"wickairportdata.txt"
"yeoviltontdata.txt"]

    for i in range(0,n_stations):

        print(            station_names[i]            )
```

```

complete_address = base_address + station_names[i]
f=open(station_names[i],"w")
station_data = requests.get(url=complete_address).text
f.write(station_data)
f.close()

t1=time.time()
file_read=t1-start_time
print(" ** Internet file read took  **",end=" ")
print(" {0:12.6f}".format(file_read))

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

** Start time                ** 1549558759.968851
aberporthdata.txt
armaghdata.txt
ballypatrickdata.txt
bradforddata.txt
braemardata.txt
cambornedata.txt
cambridgedata.txt
cardiffdata.txt
chivenordata.txt
cwmystwythdata.txt
dunstaffnagedata.txt
durhamdata.txt
eastbournedata.txt
eskdalemuirdata.txt
heathrowdata.txt
hurndata.txt
lerwickdata.txt
leucharsdata.txt
lowestoftdata.txt
manstondata.txt
nairndata.txt
newtonriggdata.txt
oxforddata.txt
paisleydata.txt
ringwaydata.txt
rossonwyedata.txt
shawburydata.txt
sheffielddata.txt
southamptondata.txt
stornowaydata.txt
suttonboningtondata.txt
tireedata.txt
valleydata.txt
waddingtondata.txt

```



```

whitbydata.txt
wickairportdata.txt
yeoviltondata.txt
** Internet file read took **          6.891494

```

We now have all of the files saved locally.

On Unix and Linux there are a number of commans that are very useful when working with text files, and they include

- wc - a utility that prints line, word and byute counts for files;
- diff -a utility that can be used to compare text files a line at a time;
- vi and vim - a powerful editor with extensive pattern matching, and powerful command set;
- sed - a stream editor for manipulating text files;
- unix2dos - a utility to convert Unix text files to dos format;
- dos2unix - a utility to convert Windows text files to Unix format;

All of them have been used in the production of these notes and I have been using vi since I worked at Imperial College in the 1970s. It is possible to install a Unix shell on a Windows machine to gain the above functionality.

The following command

```
wc *data.txt
```

produces the following output after the running this example program.

```

  945      6652      50557 aberporthdata.txt
2001     14046     106386 armaghdata.txt
  699      4933      37070 ballypatrickdata.txt
1341      9424      71475 bradforddata.txt
  730      5153      39054 braemardata.txt
  493      3487      26511 cambornedata.txt
  729      5141      38965 cambridgedata.txt
  505      3574      27090 cardiffdata.txt
  765      5391      40405 chivenordata.txt
  626      4396      32556 cwmystwythdata.txt
  580      4097      30776 dunstaffnagedata.txt
1677     11776      89188 durhamdata.txt
  729      5140      38959 eastbournedata.txt
1305      9171      69361 eskdalemuirdata.txt
  861      6065      45954 heathrowdata.txt
  753      5308      40339 hurndata.txt
1066      7499      56889 lerwickdata.txt
  753      5308      40328 leucharsdata.txt
1258      8855      66929 lowestoftdata.txt
  939      6610      50040 manstondata.txt
1066      7508      56844 nairndata.txt
  729      5141      38950 newtonriggdata.txt
2001     14044     106385 oxforddata.txt
  729      5140      38987 paisleydata.txt
  714      5013      37164 ringwaydata.txt
1066      7499      56953 rososnyedata.txt

```

```

    885      6232      47371 shawburydata.txt
  1641     11524     87305 sheffielddata.txt
  1752     12284     92910 southamptondata.txt
  1755     12322     93469 stornowaydata.txt
    729      5141     38956 suttonboningtondata.txt
  1101      7744     58814 tireedata.txt
  1066      7499     56959 valleydata.txt
    873      6147     46663 waddingtondata.txt
    698      4768     36275 whitbydata.txt
  1269      8920     66963 wickairportdata.txt
    661      4664     35479 yeoviltondata.txt
37490  263616 1995279 total

```

and we will use the line count data in several problems in the rest of the notes and examples. The line count data is the first column of output. This program was run at 18:50 on the 1st May 2019. Running it today will produce updated information.

11.6 Example 6 - Writing to a set of files names generated within Python

This program creates 10 files, and generates part of the file name from the for loop index.

```

base_file_name="test_file_"
for i in range (1,11):
    file_name=base_file_name + str(i)
    f=open(file_name,'w')
    f.write(file_name)
    f.close()

```

Run the program and do a ls or dir after running the program.

11.7 Example 7 - Copying a file and replacing missing values

This example can be used to replace the Met Office flag for missing data --- with -99 in the new version.

Here is the source.

```

import numpy as np
input_data_file ="cwmystwythdata.txt"
output_data_file="cwmystwythdata_after.txt"
nlines=626
f1=open(input_data_file)
f2=open(output_data_file,'w')
for i in range(0,nlines+1):
    line=f1.readline()
    after=line.replace("---","-99")
    f2.write(after)
f1.close()
f2.close()

```

This program is easy to modify to replace the * character in the Met Office files.

11.8 Example 8 - creating an SQL file

Here is the source.

```

import os
input_data_file ="cwmystwythdata.txt"
output_data_file="cwmystwythdata_sql.txt"
#
# header lines
#
nh=7
#
# total lines
#
nt=626
f1=open(input_data_file)
f2=open(output_data_file,'w')
#
# skip header lines
#
for i in range(0,nh):
    line=f1.readline()
#
# process data lines
#
for i in range(0,nt-nh-1):
    line=f1.readline()
    pass1=line.replace("--- ", "null")
    pass2=pass1.replace("---", "null")
    pass3=pass2.replace("*", " ")
    columns=pass3.split()
    pass4="("
    for column in columns:
        pass4 = pass4 + column + ","
    l=len(pass4)
    pass5=pass4[0:l-1] + ")"
    pass6 = pass5 + os.linesep
    f2.write(pass6)
f1.close()
f2.close()

```

11.9 Example 9 - Creating a csv file

Here is the source. It is a minor variation on the previous.

```

import os
input_data_file ="cwmystwythdata.txt"
output_data_file="cwmystwythdata.csv"
#
# header lines
#
nh=7
#
# total lines
#

```

```

nt=626
f1=open(input_data_file)
f2=open(output_data_file,'w')
#
# skip header lines
#
for i in range(0,nh):
    line=f1.readline()
#
# process data lines
#
for i in range(0,nt-nh-1):
    line=f1.readline()
    pass1=line.replace("*"," ")
    columns=pass1.split()
    pass2=""
    for column in columns:
        pass2 = pass2 + column + ","
    l=len(pass2)
    pass3=pass2[0:l-1]
    pass4 = pass3 + os.linesep
    f2.write(pass4)
f1.close()
f2.close()

```

11.10 Example 10 - CSV files and the csv module

CSV stands for comma separated values. It is a commonly used file interchange format. The csv module was introduced in Python 2.3 Here is a simple example program.

```

import csv
file_name="lines_per_station.csv"
f=open(file_name)
reader = csv.reader(f)
for row in reader:
    print(row)

```

Here is the actual csv file.

```

aberporth,877,7,870
armagh,1933,7,1926
ballypatrick,631,7,624
bradford,1273,7,1266
braemar,661,8,653
camborne,425,7,418
cambridge,661,7,654
cardiff,437,7,430
chivenor,697,7,690
cwmystwyth,627,7,620
dunstaffnage,512,7,505
durham,1609,7,1602
eastbourne,661,7,654

```

eskdalemuir,1237,7,1230
heathrow,793,7,786
hurn,685,7,678
lerwick,998,7,991
leuchars,685,7,678
lowestoft,1189,8,1181
manston,871,7,864
nairn,997,8,989
newtonrigg,661,7,654
oxford,1933,7,1926
paisley,661,7,654
ringway,715,7,708
rossonwye,998,7,991
shawbury,817,7,810
sheffield,1573,7,1566
southampton,1752,8,1744
stornoway,1687,7,1680
suttonbonington,661,7,654
tiree,1033,7,1026
valley,998,7,991
waddington,805,7,798
whitby,629,8,621
wickairport,1201,7,1194
yeovilton,593,7,586

Here is the output.

```
['aberporth', '877', '7', '870']  
['armagh', '1933', '7', '1926']  
['ballypatrick', '631', '7', '624']  
['bradford', '1273', '7', '1266']  
['braemar', '661', '8', '653']  
['camborne', '425', '7', '418']  
['cambridge', '661', '7', '654']  
['cardiff', '437', '7', '430']  
['chivenor', '697', '7', '690']  
['cwmystwyth', '627', '7', '620']  
['dunstaffnage', '512', '7', '505']  
['durham', '1609', '7', '1602']  
['eastbourne', '661', '7', '654']  
['eskdalemuir', '1237', '7', '1230']  
['heathrow', '793', '7', '786']  
['hurn', '685', '7', '678']  
['lerwick', '998', '7', '991']  
['leuchars', '685', '7', '678']  
['lowestoft', '1189', '8', '1181']  
['manston', '871', '7', '864']  
['nairn', '997', '8', '989']  
['newtonrigg', '661', '7', '654']  
['oxford', '1933', '7', '1926']  
['paisley', '661', '7', '654']
```

```

['ringway', '715', '7', '708']
['rossonweye', '998', '7', '991']
['shawbury', '817', '7', '810']
['sheffield', '1573', '7', '1566']
['southampton', '1752', '8', '1744']
['stornoway', '1687', '7', '1680']
['suttonbonington', '661', '7', '654']
['tiree', '1033', '7', '1026']
['valley', '998', '7', '991']
['waddington', '805', '7', '798']
['whitby', '629', '8', '621']
['wickairport', '1201', '7', '1194']
['yeovilton', '593', '7', '586']

```

As the file is read each row of the input data is parsed and converted into a list of strings.

11.11 Example 11 - CSV usage and data extraction

This example is a variation of the previous. In this one we extract the data from each row of the CSV file and assign it to an array of the appropriate type.

Here is the source.

```

import csv
import numpy as np
file_name="lines_per_station.csv"
# n is always less than 128
n = 128
# station name | total lines | header lines | data lines
f=open(file_name)
met_office_data = csv.reader(f)
station_names = ["" for x in range(n)]
total_lines = np.zeros([n] , dtype=np.int32)
header_lines = np.zeros([n] , dtype=np.int32)
data_lines = np.zeros([n] , dtype=np.int32)
r = 0
for row in met_office_data:
    station_names[r] = row[0]
    total_lines[r] = int(row[1])
    header_lines[r] = int(row[2])
    data_lines[r] = int(row[3])
    r=r+1
for i in range(r):
    print(station_names[i],end=" : ")
    print(total_lines[i],end=" : ")
    print(header_lines[i],end=" : ")
    print(data_lines[i])

```

Here is the output.

```

aberporth : 877 : 7 : 870
armagh : 1933 : 7 : 1926
ballypatrick : 631 : 7 : 624
bradford : 1273 : 7 : 1266

```

```

braemar : 661 : 8 : 653
camborne : 425 : 7 : 418
cambridge : 661 : 7 : 654
cardiff : 437 : 7 : 430
chivenor : 697 : 7 : 690
cwmystwyth : 627 : 7 : 620
dunstaffnage : 512 : 7 : 505
durham : 1609 : 7 : 1602
eastbourne : 661 : 7 : 654
eskdalemuir : 1237 : 7 : 1230
heathrow : 793 : 7 : 786
hurn : 685 : 7 : 678
lerwick : 998 : 7 : 991
leuchars : 685 : 7 : 678
lowestoft : 1189 : 8 : 1181
manston : 871 : 7 : 864
nairn : 997 : 8 : 989
newtonrigg : 661 : 7 : 654
oxford : 1933 : 7 : 1926
paisley : 661 : 7 : 654
ringway : 715 : 7 : 708
rossonwye : 998 : 7 : 991
shawbury : 817 : 7 : 810
sheffield : 1573 : 7 : 1566
southampton : 1752 : 8 : 1744
stornoway : 1687 : 7 : 1680
suttonbonington : 661 : 7 : 654
tiree : 1033 : 7 : 1026
valley : 998 : 7 : 991
waddington : 805 : 7 : 798
whitby : 629 : 8 : 621
wickairport : 1201 : 7 : 1194
yeovilton : 593 : 7 : 586

```

11.12 Example 12 - reading a met office file using the csv module

Here is the program.

```

import csv
import numpy as np
file_name="cwmystwyth.csv"
n = 1024
# year , month , tmax , tmin , af_days , rain , sun
f=open(file_name)
met_office_data = csv.reader(f)
year_array = np.zeros([n] , dtype=np.int32)
month_array = np.zeros([n] , dtype=np.int32)
tmax_array = np.zeros([n] , dtype=np.float64)
tmin_array = np.zeros([n] , dtype=np.float64)
af_days_array = np.zeros([n] , dtype=np.int32)
rain_array = np.zeros([n] , dtype=np.float64)

```

```

sun_array = np.zeros([n] , dtype=np.float64)
r = 0
#
for row in met_office_data:
    print(row)
    year_array[r] = int(row[0])
    month_array[r] = int(row[1])
    tmax_array[r] = float(row[2])
    tmin_array[r] = float(row[3])
    af_days_array[r] = int(row[4])
    rain_array[r] = float(row[5])
    sun_array[r] = float(row[6])
    r=r+1
#
print(" Actual lines = ",r)
rain_sum = sum(rain_array[0:r])
rain_average = rain_sum/r
#
print (" Average = {0:7.2f} mm ".format(rain_average))
rain_average=rain_average/25.4
print ("                {0:5.2f} ins".format(rain_average))

```

Here is a sample of the output.

```

[' 2010 ', ' 12 ', ' 3.1 ', ' -3.7 ', ' 23 ', ' 82.6
', ' 52.4']
[' 2011 ', ' 1 ', ' 5.8 ', ' -0.3 ', ' 16 ', ' 191.4
', ' 44.7']
[' 2011 ', ' 2 ', ' 8.3 ', ' 3.1 ', ' 5 ', ' 165.8
', ' 43.5']
[' 2011 ', ' 3 ', ' 10.3 ', ' 1.4 ', ' 12 ', ' 35.5
', ' 145.0']
Actual lines = 618
Average = 138.18 mm
          5.44 ins

```

11.13 Example 13 - reading data using the genfromtxt method

In the OO chapter we had a simple data structuring example for working with the Met Office station data. This example had the disadvantage in that it used the default dynamic typing mechanism used in Python, which leaves the determination of type until run time, which can cause programs to fail through incorrect parameter passing. Here is an example using the Numpy genfromtxt method that is strongly typed and provides a better environment for actually doing arithmetic on the data.

Here is the source.

```

import numpy as np
data_file_name="cwmystwythdata.txt"
matrix = np.genfromtxt( data_file_name, \
                        skip_header=7 , \
                        skip_footer=1 , \
                        usecols=(0,1,2,3,4,5,6), \

```



```

        autostrip=True , \
dtype=(int,int,float,float,int,float,float), \
        missing_values={"---"},\
    )
print(" Type = ",type(matrix))
print(" Size = ",matrix.size)
print(matrix)

```

Here is an extract of the output.

```

Type = <class 'numpy.ndarray'>
Size = 618
[(1959, 1, 4.5, -1.9, 20, nan, 57.2)
 (1959, 2, 7.3, 0.9, 15, nan, 87.2)
 (1959, 3, 8.4, 3.1, 3, nan, 81.6)
 (1959, 4, 10.8, 3.7, 1, nan, 107.4)
 (1959, 5, 15.8, 5.8, 1, nan, 213.5)
 (1959, 6, 16.9, 8.2, 0, nan, 209.4)
 (1959, 7, 18.5, 9.5, 0, nan, 167.8)
 (1959, 8, 19. , 10.5, 0, nan, 164.8)
 (1959, 9, 18.3, 5.9, 0, nan, 196.5)
 (1959, 10, 14.8, 7.9, 1, nan, 101.1)
 (1959, 11, 8.8, 3.9, 3, nan, 38.9)
 (1959, 12, 7.2, 2.5, 3, nan, 19.2)
...
...
...
(2010, 1, 3.4, -2.3, 22, 127.9, 32.9)
 (2010, 2, 4.8, -1.6, 19, 70.4, 72.2)
 (2010, 3, 8.7, 0.8, 16, 102. , 119.3)
 (2010, 4, 13. , 3.8, 4, 56.8, 194.3)
 (2010, 5, 14.2, 4.7, 4, 71.5, 207.3)
 (2010, 6, 18.8, 8. , 0, 80.5, 220. )
 (2010, 7, 17.3, 11.9, 0, 209.3, 80.3)
 (2010, 8, 16.6, 9.3, 0, 88.8, 130.5)
 (2010, 9, 16.3, 8.7, 1, 181.2, 135.5)
 (2010, 10, 12.5, 5.2, 5, 108. , 117.2)
 (2010, 11, 7.1, 0.5, 11, 154.9, 73.3)
 (2010, 12, 3.1, -3.7, 23, 82.6, 52.4)
 (2011, 1, 5.8, -0.3, 16, 191.4, 44.7)
 (2011, 2, 8.3, 3.1, 5, 165.8, 43.5)
 (2011, 3, 10.3, 1.4, 12, 35.5, 145. )]

```

We can now easily do arithmetic on Met Office data. The example that does some calculations is in the SQL chapter where we compare the SQL Met Office program with one based on the above example.

Here are some notes about this example

You must not use the file open method before calling the `genfromtxt` method. Doing this will generate run time errors;

The Met Office data file needs pre-processing to remove the * characters that indicate estimated data. The * character at the end of line can be removed completely, and the * character in the middle of the lines can be replaced with a space character. We used vi to do this.

On a Windows platform you will probably need to use the unix2dos command to resolve issues with carriage return and line feed characters.

We recommend installing vim on Windows as a replacement for vi. We also recommend installing cywin under Windows to provide unix functionality.

11.14 Example 14 - Writing a CSV file

Here is the source

```
import csv
import numpy as np
file_name      = "lines_per_station.csv"
output_file    = "output_file.csv"
# n is always less than 128
n = 128
# station name | total lines | header lines | data lines
f=open(file_name)
met_office_data = csv.reader(f)
station_names   = ["" for x in range(n)]
total_lines     = np.zeros([n] , dtype=np.int32)
header_lines    = np.zeros([n] , dtype=np.int32)
data_lines      = np.zeros([n] , dtype=np.int32)
r = 0
for row in met_office_data:
    station_names[r] = row[0]
    total_lines[r]   = int(row[1])
    header_lines[r]  = int(row[2])
    data_lines[r]    = int(row[3])
    r=r+1
#for i in range(r):
#    print(station_names[i],end=" : ")
#    print(total_lines[i],end=" : ")
#    print(header_lines[i],end=" : ")
#    print(data_lines[i])
ofile=open(output_file,'w')
output_data = csv.writer(ofile,delimiter = ';')
for i in range(r):
    output_data.writerow((station_names[i],total_lines[i],header_lines[i],data_lines[i]))
```

Here is the file created.

```
aberporth;877;7;870
armagh;1933;7;1926
ballypatrick;631;7;624
bradford;1273;7;1266
braemar;661;8;653
camborne;425;7;418
```

```

cambridge;661;7;654
cardiff;437;7;430
chivenor;697;7;690
cwmystwyth;627;7;620
dunstaffnage;512;7;505
durham;1609;7;1602
eastbourne;661;7;654
eskdalemuir;1237;7;1230
heathrow;793;7;786
hurn;685;7;678
lerwick;998;7;991
leuchars;685;7;678
lowestoft;1189;8;1181
manston;871;7;864
nairn;997;8;989
newtonrigg;661;7;654
oxford;1933;7;1926
paisley;661;7;654
ringway;715;7;708
rossonwye;998;7;991
shawbury;817;7;810
sheffield;1573;7;1566
southampton;1752;8;1744
stornoway;1687;7;1680
suttonbonington;661;7;654
tiree;1033;7;1026
valley;998;7;991
waddington;805;7;798
whitby;629;8;621
wickairport;1201;7;1194
yeovilton;593;7;586

```

We used semicolon as the delimiter on output.

11.15 Example 15 - write large array as text file, element by element, with timing

Here is the program.

```

import time
start_time=time.time()
print(" ** Start time
**",end=" ")
print(start_time)
import numpy as np
data_file="large_data_file.txt"
f=open(data_file,"w")
n=10000000
x = np.empty([n],dtype=np.int32)
t1=time.time()
initialisation_time=t1-start_time

```

```

print(" ** Variable creation                               **",end="
")
print(" {0:12.6f}".format(initialisation_time))
for i in range (0,n):
    x[i]=i
t2=time.time()
array_time=t2-t1
print(" ** Array initialisation                           **",end="
")
print(" {0:12.6f}".format(array_time))
for i in range (0,n):
    f.write("%12d \n" % x[i])
t3=time.time()
write_time=t3-t2
print(" ** Text array write, element by element **",end=" ")
print(" {0:12.6f}".format(write_time))

```

Here is the output.

```

$ ** Start time                                     **
                                                1447502147.7951984
** Variable creation                               **          0.163511
** Array initialisation                           **          1.940731
** Text array write, element by element **         12.593289

```

The variable creation and array initialisation take very little time. The formatted output of the array dominates the program execution time.

11.16 Example 16 - write large array as binary file , element by element, with timing

Here is the program.

```

import time
start_time=time.time()
print(" ** Start time
**",end=" ")
print(start_time)
import numpy as np
data_file="large_data_file.dat"
f=open(data_file,"wb")
n=10000000
x = np.empty([n],dtype=np.int32)
t1=time.time()
initialisation_time=t1-start_time
print(" ** Variable creation
**",end=" ")
print(" {0:12.6f}".format(initialisation_time))
for i in range (0,n):
    x[i]=i
t2=time.time()
array_time=t2-t1
print(" ** Array initialisation
**",end=" ")

```

```

print(" {0:12.6f}".format(array_time))
for i in range (0,n):
    f.write(x[i])
t3=time.time()
write_time=t3-t2
print(" ** Binary array write, element by element **",end="
")
print(" {0:12.6f}".format(write_time))

```

Here is the output.

```

$ ** Start time                **
1447502130.3637505
** Variable creation           **      0.153189
** Array initialisation       **      1.781546
** Binary array write, element by element **      11.331043

```

Results are similar to the previous example.

11.17 Example 17 - write large array as binary file , whole array, with timing

Here is the program.

```

import time
start_time=time.time()
print(" ** Start time                **",end=" ")
print(start_time)
import numpy as np
data_file="large_data_file.dat"
f=open(data_file,"wb")
n=10000000
x = np.empty([n],dtype=np.int32)
t1=time.time()
initialisation_time=t1-start_time
print(" ** Variable creation           **",end=" ")
print(" {0:12.6f}".format(initialisation_time))
for i in range (0,n):
    x[i]=i
t2=time.time()
array_time=t2-t1
print(" ** Array initialisation       **",end=" ")
print(" {0:12.6f}".format(array_time))
f.write(x)
t3=time.time()
write_time=t3-t2
print(" ** Binary array write, whole array **",end=" ")
print(" {0:12.6f}".format(write_time))

```

Here is the output.

```

$ python3 c1204.py
** Start time                ** 1447502254.5353513
** Variable creation         **      0.152651
** Array initialisation      **      1.783490

```

```
** Binary array write, whole array **          0.363772
```

We now have a considerable improvement in the write timing.

11.18 Example 18 - listing subdirectories

We will be using `pathlib`, which was new in version 3.4.

This module offers classes representing filesystem paths with semantics appropriate for different operating systems. Path classes are divided between pure paths, which provide purely computational operations without I/O, and concrete paths, which inherit from pure paths but also provide I/O operations.

Here is a short program that prints all sub directories. We use two syntax variants.

```
#Listing subdirectories:
from pathlib import Path
p = Path('.')
for x in p.iterdir():
    if x.is_dir():
        print(x)
print(" Now print posix details")
print([x for x in p.iterdir() if x.is_dir()])
```

Here is the output.

```
dbms
effbot
grayson
tkinter
__pycache__
Now print posix details
[PosixPath('dbms'), PosixPath('effbot'), PosixPath('grayson'),
PosixPath('tkinter'), PosixPath('__pycache__')]
```

11.19 Example 19 - listing all Python files

Here is the source.

```
#Listing Python source files in this directory tree:
from pathlib import Path
p = Path('.')
for filename in p.glob('**/*.py'):
    print(filename)
```

Here is a segment of output.

```
c01201.py
c0201.py
c0202.py
c0203.py
c0301.py
c0302.py
..
..
tkinter/tt077.py
tkinter/tt078.py
```

```
tkinter/tt079.py
tkinter/tt080.py
tkinter/tt090.py
tkinter/tt095.py
tkinter/tt100.py
tkinter/z.py
```

Try this on your system.

11.20 Background i/o technical information

A small amount of background material is covered in the sections below.

11.21 Text I/O

Text I/O expects and produces str objects. This means that whenever the backing store is natively made of bytes (such as in the case of a file), encoding and decoding of data is made transparently as well as optional translation of platform-specific newline characters.

The easiest way to create a text stream is with `open()`, optionally specifying an encoding:

```
f = open("myfile.txt", "r", encoding="utf-8")
```

In-memory text streams are also available as `StringIO` objects:

```
f = io.StringIO("some initial text data")
```

The text stream API is described in detail in the documentation of `TextIOBase`.

11.22 Binary I/O

Binary I/O (also called buffered I/O) expects and produces bytes objects. No encoding, decoding, or newline translation is performed. This category of streams can be used for all kinds of non-text data, and also when manual control over the handling of text data is desired.

The easiest way to create a binary stream is with `open()` with 'b' in the mode string:

```
f = open("myfile.jpg", "rb")
```

In-memory binary streams are also available as `BytesIO` objects:

```
f = io.BytesIO(b"some initial binary data: \x00\x01")
```

The binary stream API is described in detail in the docs of `BufferedIOBase`.

Other library modules may provide additional ways to create text or binary streams. See `socket.socket.makefile()` for example.

11.23 Raw I/O

Raw I/O (also called unbuffered I/O) is generally used as a low-level building-block for binary and text streams; it is rarely useful to directly manipulate a raw stream from user code. Nevertheless, you can create a raw stream by opening a file in binary mode with buffering

```
f = open("myfile.jpg", "rb", buffering=0)
```

The raw stream API is described in detail in the docs of `RawIOBase`.

11.24 Performance

This section discusses the performance of the provided concrete I/O implementations.

11.24.1 Binary I/O

By reading and writing only large chunks of data even when the user asks for a single byte, buffered I/O hides any inefficiency in calling and executing the operating system's

unbuffered I/O routines. The gain depends on the OS and the kind of I/O which is performed. For example, on some modern OSes such as Linux, unbuffered disk I/O can be as fast as buffered I/O. The bottom line, however, is that buffered I/O offers predictable performance regardless of the platform and the backing device. Therefore, it is almost always preferable to use buffered I/O rather than unbuffered I/O for binary data.

11.24.2 Text I/O

Text I/O over a binary storage (such as a file) is significantly slower than binary I/O over the same storage, because it requires conversions between unicode and binary data using a character codec. This can become noticeable handling huge amounts of text data like large log files. Also, `TextIOWrapper.tell()` and `TextIOWrapper.seek()` are both quite slow due to the reconstruction algorithm used.

`StringIO`, however, is a native in-memory unicode container and will exhibit similar speed to `BytesIO`.

11.24.3 Multi-threading

`FileIO` objects are thread-safe to the extent that the operating system calls (such as `read(2)` under Unix) they wrap are thread-safe too.

Binary buffered objects (instances of `BufferedReader`, `BufferedWriter`, `BufferedRandom` and `BufferedRWPair`) protect their internal structures using a lock; it is therefore safe to call them from multiple threads at once.

`TextIOWrapper` objects are not thread-safe.

11.24.4 Reentrancy

Binary buffered objects (instances of `BufferedReader`, `BufferedWriter`, `BufferedRandom` and `BufferedRWPair`) are not reentrant. While reentrant calls will not happen in normal situations, they can arise from doing I/O in a signal handler. If a thread tries to re-enter a buffered object which it is already accessing, a `RuntimeError` is raised. Note this doesn't prohibit a different thread from entering the buffered object.

The above implicitly extends to text files, since the `open()` function will wrap a buffered object inside a `TextIOWrapper`. This includes standard streams and therefore affects the built-in function `print()` as well.

11.25 Problems

1. Try running the examples in this chapter.

“Errors using inadequate data are much less than those using no data at all.”

Charles Babbage

Aims

The aims of this chapter are to provide an introduction to algorithms and their behaviour. In Computer Science this is normally done using the so called big O notation.

12 An Introduction to Algorithms and the Big O notation

A method for dealing with approximations was introduced by Bachman in 1892 in his work *Analytische Zahlen Theorie*. This is the big O notation.

The big O notation is used to classify algorithms by how they perform depending on the size of the input data set they are working on. This typically means looking at both their space and time behaviour.

A more detailed and mathematical coverage can be found in Knuth's *Fundamental Algorithms*. Chapter one of this book looks at the basic concepts and mathematical preliminaries required for analysing algorithms, and is around 120 pages. Well worth a read.

12.1 Basic background

The table below summarises some of the details regarding commonly occurring types of problem.

Notation	Name
$O(1)$	constant
$O(n)$	linear
$O(\log n)$	logarithmic
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, quasilinear
$O(\log \log n)$	double logarithmic
$O(n \log^* n)$	n log-star n
$O(n^2)$	quadratic
$O(n^c) \ 0 < c < 1$	fractional power
$O(n^c) \ c > 1$	polynomial or algebraic
$O(c^n) \ c > 1$	exponential
$O(n!)$	factorial

12.1.1 Brief explanation

$O(1)$	Determining if a number is even or odd; using a constant-size lookup table
$O(\log \log n)$	Finding an item using interpolation search in a sorted array of uniformly distributed values.
$O(\log n)$	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap.
$O(n^c) 0 < c < 1$	Searching in a kd-tree
$O(n)$	Finding an item in an unsorted list or a malformed tree (worst case) or in an unsorted array;
$O(n \log^* n)$	Performing triangulation of a simple polygon using Seidel's algorithm.
$O(n \log n)$	Performing a Fast Fourier transform; heapsort, quicksort (best and average case), or merge sort.
$O(n^2)$	Multiplying two n-digit numbers by a simple algorithm; bubble sort (worst case or naive implementation), Shell sort, quicksort (worst case), selection sort or insertion sort.
$O(n^c) c > 1$	Tree-adjointing grammar parsing; maximum matching for bipartite graphs.
$O(c^n) c > 1$	Finding the (exact) solution to the travelling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search.
$O(n!)$	Solving the travelling salesman problem via brute-force search; generating all unrestricted permutations of a poset; finding the determinant with expansion by minors.

The following table illustrates the behaviour of 4 of the above for increasing n.

n	$O(1)$	$O(n)$	$O(n*n)$	$O(\log n)$	$O(n \log n)$
1	1	1	1.00E+00	0	0.00E+00
10	1	10	1.00E+02	2.3	2.30E+01
100	1	100	1.00E+04	4.61	4.61E+02
1,000	1	1,000	1.00E+06	6.91	6.91E+03
10,000	1	10,000	1.00E+08	9.21	9.21E+04
100,000	1	100,000	1.00E+10	11.51	1.15E+06
1,000,000	1	1,000,000	1.00E+12	13.82	1.38E+07
10,000,000	1	10,000,000	1.00E+14	16.12	1.61E+08
100,000,000	1	100,000,000	1.00E+16	18.42	1.84E+09
1,000,000,000	1	1,000,000,000	1.00E+18	20.72	2.07E+10

12.2 Quicksort and insertion sort comparison

Algorithm	Data Structure	Time Complexity			Worst Case
		Best	Average	Worst	Worst
Quicksort	Array	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(n)$
Insertion Sort	Array	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$

12.3 Basic array and linked list performance

The following table summarises this information.

Data Structure	Time Complexity								Space Complexity
	Average				Worst				
	Index	Search	Insert	Delete	Index	Search	Insert	Delete	
Basic Array	$O(1)$	$O(n)$	-	-	$O(1)$	$O(n)$	-	-	$O(n)$
Dy-namic Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Singly Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$

12.4 Bibliography

The earliest books that we have used in this area are those by Donald Knuth, and details are given below in chronological order.

Volume 1, Fundamental Algorithms, first edition, 1968, xxi+634pp, ISBN 0-201-03801-3.

Volume 2, Seminumerical Algorithms, first edition, 1969, xi+624pp, ISBN 0-201-03802-1.

Volume 3, Sorting and Searching, first edition, 1973, xi+723pp+centerfold, ISBN 0-201-03803-X

Volume 1, second edition, 1973, xxi+634pp, ISBN 0-201-03809-9.

Volume 2, second edition, 1981, xiii+ 688pp, ISBN 0-201-03822-6.

Knuth uses the Mix assembly language (an artificial language) and this limits the accessibility of the books.

However within the Computer Science community they are generally regarded as the first and most comprehensive treatment of its subject. itemize

For something more accessible, Sedgewick has written several programming language versions of a book on algorithms. He was a student of Knuth's. The earliest used Pascal, and later editions have used C, C++ and Modula 2 and Modula 3.

Sedgewick, Robert (1992). Algorithms in C++, Addison-Wesley.
ISBN 0-201-51059-6.

Sedgewick, Robert (1993). Algorithms in Modula 3, Addison-Wesley.
ISBN 0-201-53351-0. itemize

12.5 Problems

There are no problems in this chapter.

The good teacher is a guide who helps others dispense with his services.
R. S. Peters, Ethics and Education.

13 Sequence types, Iterators and Lists

The information in this chapter is taken from section 4 of
<https://docs.python.org/3/library/index.html>
which is on the Python built in data types, as listed below

- Truth Value Testing
- Boolean Operations — and, or, not
- Comparisons
- Numeric Types — int, float, complex
- Iterator Types
- Sequence Types — list, tuple, range
- Text Sequence Type — str
- Binary Sequence Types — bytes, bytearray, memoryview
- Set Types — set, frozenset
- Mapping Types — dict
- Context Manager Types
- Other Built-in Types
- Special Attributes

and we will concentrate on

- Iterator Types
- Sequence Types — list, tuple, range
- Set Types — set, frozenset
- Mapping Types — dict

13.1 Iterator types

Python supports a concept of iteration over containers. This is implemented using two distinct methods; these are used to allow user-defined classes to support iteration. Sequences, described below in more detail, always support the iteration methods.

Python's generators provide a convenient way to implement the iterator protocol. If a container object's `__iter__()` method is implemented as a generator, it will automatically return an iterator object (technically, a generator object) supplying the `__iter__()` and `__next__()` methods. More information about generators can be found in the documentation for the `yield` expression.

13.2 Example 1 - Simple iterator usage

Here is a simple iterator example.

```
l=[1,2,3,4]
for i in l:
```

```
print(i)
for line in open("c1301.txt"):
    print(line)
```

Here is the output

```
$
1
2
3
4
This is a file

with some text in it

over three lines
```

In this example we iterate over a list and the lines in a file.

13.3 Sequence types

There are three basic sequence types: lists, tuples, and range objects.

13.3.1 Common Sequence Operations

The operations in the following table are supported by most sequence types, both mutable and immutable. The `collections.abc.Sequence` ABC is provided to make it easier to correctly implement these operations on custom sequence types.

This table lists the sequence operations sorted in ascending priority. In the table, `s` and `t` are sequences of the same type, `n`, `i`, `j` and `k` are integers and `x` is an arbitrary object that meets any type and value restrictions imposed by `s`.

The `in` and `not in` operations have the same priorities as the comparison operations. The `+` (concatenation) and `*` (repetition) operations have the same priority as the corresponding numeric operations.

Operation	Result (Notes)
<code>x in s</code>	True if an item of <code>s</code> is equal to <code>x</code> , else False
<code>x not in s</code>	False if an item of <code>s</code> is equal to <code>x</code> , else True
<code>s + t</code>	the concatenation of <code>s</code> and <code>t</code>
<code>s * n</code> or <code>n * s</code>	equivalent to adding <code>s</code> to itself <code>n</code> times
<code>s[i]</code>	<code>i</code> th item of <code>s</code> , origin 0
<code>s[i:j]</code>	slice of <code>s</code> from <code>i</code> to <code>j</code>
<code>s[i:j:k]</code>	slice of <code>s</code> from <code>i</code> to <code>j</code> with step <code>k</code>
<code>len(s)</code>	length of <code>s</code>
<code>min(s)</code>	smallest item of <code>s</code>

<code>max(s)</code>	largest item of <code>s</code>
<code>s.index(x[, i[, j]])</code>	index of the first occurrence of <code>x</code> in <code>s</code> (at or after index <code>i</code> and before index <code>j</code>)
<code>s.count(x)</code>	total number of occurrences of <code>x</code> in <code>s</code>

See the full reference for the meaning of the notes.

13.3.2 Immutable Sequence Types

The only operation that immutable sequence types generally implement that is not also implemented by mutable sequence types is support for the `hash()` built-in.

This support allows immutable sequences, such as tuple instances, to be used as dict keys and stored in set and frozenset instances.

Attempting to hash an immutable sequence that contains unhashable values will result in `TypeError`.

13.3.3 Mutable Sequence Types

The operations in the following table are defined on mutable sequence types. The `collections.abc.MutableSequence` ABC is provided to make it easier to correctly implement these operations on custom sequence types.

In the table `s` is an instance of a mutable sequence type, `t` is any iterable object and `x` is an arbitrary object that meets any type and value restrictions imposed by `s` (for example, `bytearray` only accepts integers that meet the value restriction $0 \leq x \leq 255$).

Operation	Result (Notes)
<code>s[i] = x</code>	item <code>i</code> of <code>s</code> is replaced by <code>x</code>
<code>s[i:j] = t</code>	slice of <code>s</code> from <code>i</code> to <code>j</code> is replaced by the contents of the iterable <code>t</code>
<code>del s[i:j]</code>	same as <code>s[i:j] = []</code>
<code>s[i:j:k] = t</code>	the elements of <code>s[i:j:k]</code> are replaced by those of <code>t</code>
<code>del s[i:j:k]</code>	removes the elements of <code>s[i:j:k]</code> from the list
<code>s.append(x)</code>	appends <code>x</code> to the end of the sequence (same as <code>s[len(s):len(s)] = [x]</code>)
<code>s.clear()</code>	removes all items from <code>s</code> (same as <code>del s[:]</code>)
<code>s.copy()</code>	creates a shallow copy of <code>s</code> (same as <code>s[:]</code>)
<code>s.extend(t)</code> or <code>s += t</code>	extends <code>s</code> with the contents of <code>t</code> (for the most part the same as <code>s[len(s):len(s)] = t</code>)
<code>s *= n</code>	updates <code>s</code> with its contents repeated <code>n</code> times
<code>s.insert(i, x)</code>	inserts <code>x</code> into <code>s</code> at the index given by <code>i</code> (same as <code>s[i:i] = [x]</code>)
<code>s.pop([i])</code>	retrieves the item at <code>i</code> and also removes it from <code>s</code>
<code>s.remove(x)</code>	remove the first item from <code>s</code> where <code>s[i] == x</code>

`s.reverse()` reverses the items of `s` in place

See the full text for the meaning of the notes.

13.4 Lists

Lists are mutable sequences, typically used to store collections of homogeneous items (where the precise degree of similarity will vary by application).

```
class list([iterable])
```

Lists may be constructed in several ways:

Using a pair of square brackets to denote the empty list: `[]`

Using square brackets, separating items with commas: `[a]`, `[a, b, c]`

Using a list comprehension: `[x for x in iterable]`

Using the type constructor: `list()` or `list(iterable)`

The constructor builds a list whose items are the same and in the same order as `iterable`'s items. `iterable` may be either a sequence, a container that supports iteration, or an iterator object. If `iterable` is already a list, a copy is made and returned, similar to `iterable[:]`. For example, `list('abc')` returns `['a', 'b', 'c']` and `list((1, 2, 3))` returns `[1, 2, 3]`. If no argument is given, the constructor creates a new empty list, `[]`.

Many other operations also produce lists, including the `sorted()` built-in.

Lists implement all of the common and mutable sequence operations. Lists also provide the following additional method:

```
sort(*, key=None, reverse=None)
```

This method sorts the list in place, using only `<` comparisons between items. Exceptions are not suppressed - if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

`sort()` accepts two arguments that can only be passed by keyword (keyword-only arguments):

`key` specifies a function of one argument that is used to extract a comparison key from each list element (for example, `key=str.lower`). The key corresponding to each item in the list is calculated once and then used for the entire sorting process. The default value of `None` means that list items are sorted directly without calculating a separate key value.

The `functools.cmp_to_key()` utility is available to convert a 2.x style `cmp` function to a key function.

`reverse` is a boolean value. If set to `True`, then the list elements are sorted as if each comparison were reversed.

This method modifies the sequence in place for economy of space when sorting a large sequence. To remind users that it operates by side effect, it does not return the sorted sequence (use `sorted()` to explicitly request a new sorted list instance).

The `sort()` method is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

CPython implementation detail: While a list is being sorted, the effect of attempting to mutate, or even inspect, the list is undefined. The C implementation of Python makes the list appear empty for the duration, and raises `ValueError` if it can detect that the list has been mutated during a sort.

13.5 Example 2 - list type initialisation and simple for in statement

Here is the program.

```
months = ["January", "February", "March", "April", "May",
"June", "July", "August", "September", "October",
"November", "December"]
days = ["Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"]
for m in months:
    print(m, end=" ")
print()
for d in days:
    print(d, end=" ")
print()
```

Here is the output

```
January February March April May June July August September
October November December
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
```

13.6 Example 3 - list type and various sequence methods

Here is the program.

```
list_1 = range(10)
print(" List 1 is", end=": ")
for i in list_1:
    print(i, end=" ")
print()
n1=len(list_1)
print(" Length of list is ", n1)
list_2 = range(5)
n2=len(list_2)
list_3 = []
list_3[ 0 : (n1-1) ] = list_1
list_3[ n1 : (n1+n2-1)] = list_2
print(" List 2 is", end=": ")
for i in list_2:
    print(i, end=" ")
print()
print(" Length of list is ", len(list_2))
print(" List 3 is", end=": ")
for i in list_3:
    print(i, end=" ")
print()
print(" Length of list is ", len(list_3))
list_3.sort()
print(" Sorted list is: ", end=" ")
for i in list_3:
    print(i, end=" ")
print()
list_3.reverse()
```

```

print(" Reverse list is: ",end=" ")
for i in list_3:
    print(i,end=" ")
print()
list_3.insert(5, 99)
list_3.insert(6,999)
print(" List after insert: ",end=" ")
for i in list_3:
    print(i,end=" ")
print()

```

Here is the output.

```

List 1 is: 0 1 2 3 4 5 6 7 8 9
Length of list is 10
List 2 is: 0 1 2 3 4
Length of list is 5
List 3 is: 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
Length of list is 15
Sorted list is: 0 0 1 1 2 2 3 3 4 4 5 6 7 8 9
Reverse list is: 9 8 7 6 5 4 4 3 3 2 2 1 1 0 0
List after insert: 9 8 7 6 5 99 999 4 4 3 3 2 2 1 1 0 0

```

This example shows the use of

- list construction using ranges
- list construction using an empty list
- iterating over a list
- the len method
- list assignment
- list sort method
- list reverse method
- list insert method

The documentation has details of the complete list of methods available.

13.7 Example 4 - list assignment versus copy() method

Consider the following example.

```

list_1 = [1,2,3,4,5]
print(" List 1 is",end=": ")
for i in list_1:
    print(i,end=" ")
print()
n1=len(list_1)
print(" Length of list is ",n1)
list_2 = list_1
print(" List 2 is",end=": ")
for i in list_2:
    print(i,end=" ")
print()

```

```

list_2[3]=99
print(" List 1 is",end=": ")
for i in list_1:
    print(i,end=" ")
print()
list_3 = list_1.copy()
print(" List 3 is",end=": ")
for i in list_3:
    print(i,end=" ")
print()
list_3[1]=999
print(" List 3 is",end=": ")
for i in list_3:
    print(i,end=" ")
print()
print(" List 1 is",end=": ")
for i in list_1:
    print(i,end=" ")
print()

```

Here is the output.

```

List 1 is: 1 2 3 4 5
Length of list is 5
List 2 is: 1 2 3 4 5
List 1 is: 1 2 3 99 5
List 3 is: 1 2 3 99 5
List 3 is: 1 999 3 99 5
List 1 is: 1 2 3 99 5

```

Assignment does a shallow copy.

13.8 List comprehensions

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

13.9 Example 5 - simple list comprehension

Here is the source code.

```

def main():

    squares = []
    for x in range(10):
        squares.append(x**2)
    print(squares)

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

13.10 Example 6 - more list comprehensions

Here is the source.

```
def main():

    vec = [-4, -2, 0, 2, 4]
    # create a new list with the values doubled
    print([x*2 for x in vec])

    # filter the list to exclude negative numbers
    print([x for x in vec if x >= 0])

    # apply a function to all the elements
    print([abs(x) for x in vec])

    # call a method on each element
    freshfruit = [' banana', ' loganberry ', 'passion fruit
']
    print([weapon.strip() for weapon in freshfruit])

    # create a list of 2-tuples like (number, square)
    print([(x, x**2) for x in range(6)])

    # flatten a list using a listcomp with two 'for'
    vec = [[1,2,3], [4,5,6], [7,8,9]]
    print([num for elem in vec for num in elem])

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
[-8, -4, 0, 4, 8]
[0, 2, 4]
[4, 2, 0, 2, 4]
['banana', 'loganberry', 'passion fruit']
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

13.11 Example 7 - more list comprehensions

Here is the source

```
from math import pi

def main():

    print([str(round(pi, i)) for i in range(1, 6)])

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

13.12 Example 8 - even more list comprehensions

Here is the source

```
def main():

    matrix = [
        [1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
        ]

    print([[row[i] for row in matrix] for i in range(4)])

    # which is equivalent to

    transposed = []
    for i in range(4):
        transposed.append([row[i] for row in matrix])
    print(transposed)

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

13.13 Tuples

Tuples are immutable sequences, typically used to store collections of heterogeneous data (such as the 2-tuples produced by the `enumerate()` built-in). Tuples are also used for cases where an immutable sequence of homogeneous data is needed (such as allowing storage in a set or dict instance).

```
class tuple([iterable])
```

Tuples may be constructed in a number of ways:

- Using a pair of parentheses to denote the empty tuple: `()`

- Using a trailing comma for a singleton tuple: `a,` or `(a,)`

- Separating items with commas: `a, b, c` or `(a, b, c)`

- Using the `tuple()` built-in: `tuple()` or `tuple(iterable)`

The constructor builds a tuple whose items are the same and in the same order as `iterable`'s items. `iterable` may be either a sequence, a container that supports iteration, or an iterator object. If `iterable` is already a tuple, it is returned unchanged. For example, `tuple('abc')` returns `('a', 'b', 'c')` and `tuple([1, 2, 3])` returns `(1, 2, 3)`. If no argument is given, the constructor creates a new empty tuple, `()`.

Note that it is actually the comma which makes a tuple, not the parentheses. The parentheses are optional, except in the empty tuple case, or when they are needed to avoid syntactic

ambiguity. For example, `f(a, b, c)` is a function call with three arguments, while `f((a, b, c))` is a function call with a 3-tuple as the sole argument.

Tuples implement all of the common sequence operations.

For heterogeneous collections of data where access by name is clearer than access by index, `collections.namedtuple()` may be a more appropriate choice than a simple tuple object.

13.14 Example 9 - simple tuple usage

Here is the source code.

```
def main():

    tuple_array = []

    t1 = ("ian", "02077333896")
    t2 = ("joan", "02078482671")
    t3 = 'ian', 'david', 'chivers'

    tuple_array.append(t1)
    tuple_array.append(t2)
    tuple_array.append(t3)

    (key, value) = tuple_array[0]

    for i in range(len(tuple_array)):
        print(tuple_array[i])

    print(key)
    print(value)

    for s in t3:
        print(s)

    for i in range(len(t3)):
        print(t3[i])

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
('ian', '02077333896')
('joan', '02078482671')
('ian', 'david', 'chivers')
ian
02077333896
ian
david
chivers
ian
david
chivers
```

13.15 Ranges

The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

```
class range(stop)
class range(start, stop[, step])
```

The arguments to the range constructor must be integers (either built-in int or any object that implements the `__index__` special method). If the step argument is omitted, it defaults to 1. If the start argument is omitted, it defaults to 0. If step is zero, ValueError is raised.

For a positive step, the contents of a range r are determined by the formula $r[i] = \text{start} + \text{step} * i$ where $i \geq 0$ and $r[i] < \text{stop}$.

For a negative step, the contents of the range are still determined by the formula $r[i] = \text{start} + \text{step} * i$, but the constraints are $i \geq 0$ and $r[i] > \text{stop}$.

A range object will be empty if $r[0]$ does not meet the value constraint. Ranges do support negative indices, but these are interpreted as indexing from the end of the sequence determined by the positive indices.

Ranges containing absolute values larger than `sys.maxsize` are permitted but some features (such as `len()`) may raise OverflowError.

13.16 Example 10 - simple range usage

Here is the source code.

```
def main():

    print(list(range(10)))
    print(list(range(1, 11)))
    print(list(range(0, 30, 5)))
    print(list(range(0, 10, 3)))
    print(list(range(0, -10, -1)))
    print(list(range(0)))
    print(list(range(1, 0)))

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 5, 10, 15, 20, 25]
[0, 3, 6, 9]
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
[]
[]
```

13.17 Problems

1. Run the examples in this chapter.

The good teacher is a guide who helps others dispense with his services.

R. S. Peters, *Ethics and Education*.

14 Set types

14.1 Set Types

A set object is an unordered collection of distinct hashable objects. Common uses include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference. (For other containers see the built-in dict, list, and tuple classes, and the collections module.)

Like other collections, sets support `x in set`, `len(set)`, and `for x in set`. Being an unordered collection, sets do not record element position or order of insertion. Accordingly, sets do not support indexing, slicing, or other sequence-like behavior.

There are currently two built-in set types, `set` and `frozenset`. The `set` type is mutable — the contents can be changed using methods like `add()` and `remove()`. Since it is mutable, it has no hash value and cannot be used as either a dictionary key or as an element of another set. The `frozenset` type is immutable and hashable — its contents cannot be altered after it is created; it can therefore be used as a dictionary key or as an element of another set.

Non-empty sets (not `frozensets`) can be created by placing a comma-separated list of elements within braces, for example: `{'jack', 'sjoerd'}`, in addition to the set constructor.

The constructors for both classes work the same:

```
class set([iterable])
class frozenset([iterable])
```

Return a new set or `frozenset` object whose elements are taken from `iterable`. The elements of a set must be hashable. To represent sets of sets, the inner sets must be `frozenset` objects. If `iterable` is not specified, a new empty set is returned.

Instances of `set` and `frozenset` provide the following operations:

<code>len(s)</code>	Return the cardinality of set <code>s</code> .
<code>x in s</code>	Test <code>x</code> for membership in <code>s</code> .
<code>x not in s</code>	Test <code>x</code> for non-membership in <code>s</code> .
<code>isdisjoint(other)</code>	Return <code>True</code> if the set has no elements in common with <code>other</code> . Sets are disjoint if and only if their intersection is the empty set.
<code>issubset(other)</code> <code>set <= other</code>	Test whether every element in the set is in <code>other</code> . Test whether the set is a proper subset of <code>other</code> , that is, <code>set <= other</code> and <code>set != other</code> .
<code>issuperset(other)</code> <code>set >= other</code>	Test whether every element in <code>other</code> is in the set. Test whether the set is a proper superset of <code>other</code> , that is, <code>set >= other</code> and <code>set != other</code> .
<code>union(other, ...)</code> <code>set other ...</code>	Return a new set with elements from the set and all others.
<code>intersection(other, ...)</code> <code>set & other & ...</code>	Return a new set with elements common to the set and all others.

<code>difference(other, ...)</code> <code>set - other - ...</code>	Return a new set with elements in the set that are not in the others.
<code>symmetric_difference(other)</code> <code>set ^ other</code>	Return a new set with elements in either the set or other but not both.
<code>copy()</code>	Return a new set with a shallow copy of s.

Note, the non-operator versions of `union()`, `intersection()`, `difference()`, and `symmetric_difference()`, `issubset()`, and `issuperset()` methods will accept any iterable as an argument. In contrast, their operator based counterparts require their arguments to be sets. This precludes error-prone constructions like `set('abc') & 'cbs'` in favor of the more readable `set('abc').intersection('cbs')`.

Both `set` and `frozenset` support set to set comparisons. Two sets are equal if and only if every element of each set is contained in the other (each is a subset of the other). A set is less than another set if and only if the first set is a proper subset of the second set (is a subset, but is not equal). A set is greater than another set if and only if the first set is a proper superset of the second set (is a superset, but is not equal).

Instances of `set` are compared to instances of `frozenset` based on their members. For example, `set('abc') == frozenset('abc')` returns `True` and so does `set('abc')` in `set([frozenset('abc')])`. The subset and equality comparisons do not generalize to a total ordering function. For example, any two nonempty disjoint sets are not equal and are not subsets of each other, so all of the following return `False`: `a < b`, `a == b`, or `a > b`.

Since sets only define partial ordering (subset relationships), the output of the `list.sort()` method is undefined for lists of sets.

Set elements, like dictionary keys, must be hashable.

Binary operations that mix set instances with `frozenset` return the type of the first operand. For example: `frozenset('ab') | set('bc')` returns an instance of `frozenset`.

The following table lists operations available for `set` that do not apply to immutable instances of `frozenset`:

<code>update(other, ...)</code> <code>set = other ...</code>	Update the set, adding elements from all others.
<code>intersection_update(other, ...)</code> <code>set &= other & ...</code>	Update the set, keeping only elements found in it and all others.
<code>difference_update(other, ...)</code> <code>set -= other ...</code>	Update the set, removing elements found in others.
<code>symmetric_difference_update(other)</code> <code>set ^= other</code>	Update the set, keeping only elements found in either set, but not in both.
<code>add(elem)</code>	Add element <code>elem</code> to the set.
<code>remove(elem)</code>	Remove element <code>elem</code> from the set. Raises <code>KeyError</code> if <code>elem</code> is not contained in the set.
<code>discard(elem)</code>	Remove element <code>elem</code> from the set if it is present.
<code>pop()</code>	Remove and return an arbitrary element from the set. Raises <code>KeyError</code> if the set is empty.
<code>clear()</code>	Remove all elements from the set.

Note, the non-operator versions of the `update()`, `intersection_update()`, `difference_update()`, and `symmetric_difference_update()` methods will accept any iterable as an argument.

Note, the `elem` argument to the `__contains__()`, `remove()`, and `discard()` methods may be a set. To support searching for an equivalent frozenset, the `elem` set is temporarily mutated during the search and then restored. During the search, the `elem` set should not be read or mutated since it does not have a meaningful value.

14.2 Example 1 - simple set usage

Here is the source

```
def main():

    basket = {'apple', 'orange', 'apple', 'pear', 'orange',
             'banana'}
    print(basket)                # show that dupli-
    cates have been removed

    print('orange' in basket)    # fast member-
    ship testing

    print('crabgrass' in basket)

    # Demonstrate set operations on unique letters from two
    words

    a = set('abracadabra')
    b = set('alacazam')

    print(a)                    # unique
    letters in a

    print(a - b)                # letters
    in a but not in b

    print(a | b)                # letters
    in either a or b

    print(a & b)                # letters
    in both a and b

    print(a ^ b)                # letters
    in a or b but not both

    a = {x for x in 'abracadabra' if x not in 'abc'}
    print(a)

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
{'apple', 'orange', 'pear', 'banana'}
True
```

```
False
{'c', 'd', 'b', 'a', 'r'}
{'d', 'b', 'r'}
{'m', 'd', 'r', 'a', 'l', 'b', 'z', 'c'}
{'a', 'c'}
{'m', 'd', 'r', 'l', 'b', 'z'}
{'d', 'r'}
```

14.3 Example 2 - simple dictionary

In this example we look at creating a dictionary of words for use as a spelling checker. Here is the source code.

```
import time

def main():
    start_time=time.time()
    print(" ** Start time                **",end=" ")
    print(start_time)
    data_file="words"
    nwords=173528
    dictionary = set()
    f=open(data_file)
    for i in range(0,nwords):
        line=f.readline()
        word=line.rstrip('\n')
        dictionary.add(word)
    t1=time.time()
    initialisation_time=t1-start_time
    print(" ** Dictionary read took **",end=" ")
    print(" {0:12.6f}".format(initialisation_time))
    print(" Dictionary length = ",end=" ")
    print(len(dictionary))
    my_word = input(" Type in a word ?")
    print(" Looking for ",end=" ")
    print(my_word)
    t1=time.time()
    if (my_word in dictionary):
        print(" ",end=" ")
        print(my_word,end=" ")
        print(" in dictionary")
    t2=time.time()
    find_time=t2-t1
    print(" **Set word look up took **",end=" ")
    print(" {0:12.6f}".format(find_time))

if ( __name__ == "__main__" ):
    main()
```

Here is the output from a sample run.

```
c:\document\python\examples>python set_02.py
** Start time                ** 1453652432.7970338
** Dictionary read took    **      0.187796
Dictionary length = 173528
Type in a word ?banana
Looking for banana
  banana in dictionary
**Set word look up took    **      0.000000
```

14.4 Problems

1. Run these examples.

What timing did you get for the dictionary example?

The good teacher is a guide who helps others dispense with his services.

R. S. Peters, Ethics and Education.

15 Mapping types

The information in.

15.1 Mapping types

A mapping object maps hashable values to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the dictionary. (For other containers see the built-in list, set, and tuple classes, and the collections module.)

A dictionary's keys are almost arbitrary values. Values that are not hashable, that is, values containing lists, dictionaries or other mutable types (that are compared by value rather than by object identity) may not be used as keys. Numeric types used for keys obey the normal rules for numeric comparison: if two numbers compare equal (such as 1 and 1.0) then they can be used interchangeably to index the same dictionary entry. (Note however, that since computers store floating-point numbers as approximations it is usually unwise to use them as dictionary keys.)

Dictionaries can be created by placing a comma-separated list of key: value pairs within braces, for example: {'jack': 4098, 'sjoerd': 4127} or {4098: 'jack', 4127: 'sjoerd'}, or by the dict constructor.

15.2 Example 1 - simple dict usage

Here is the source code.

```
def main():

    a = dict(one=1, two=2, three=3)
    b = {'one': 1, 'two': 2, 'three': 3}
    c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
    d = dict([('two', 2), ('one', 1), ('three', 3)])
    e = dict({'three': 3, 'one': 1, 'two': 2})
    print(a == b == c == d == e)

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

True

15.3 Example 2 - dict view usage

Here is the source code

```
def main():

    dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam':
500}
    keys = dishes.keys()
    values = dishes.values()
```

```
# iteration
n = 0
for val in values:
    n += val
print(n)

# keys and values are iterated over in the same order
list(keys)
list(values)

# view objects are dynamic and reflect dict changes
del dishes['eggs']
del dishes['sausage']
list(keys)

# set operations
print(keys & {'eggs', 'bacon', 'salad'})
print(keys ^ {'sausage', 'juice'})

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
504
{'bacon'}
{'bacon', 'spam', 'juice', 'sausage'}
```

15.4 Problems

1. Run the examples.

16 Operator overloading

16.1 Introduction

Operator overloading is already implemented for a variety of the built-in classes or types in Python. The following table has a list of Python's operator overloading methods.

Method Definition	Operator	Description
<code>__add__(self,y)</code>	<code>x + y</code>	The addition of two objects. The type of x determines which add operator is called.
<code>__contains__(self,y)</code>	<code>y in x</code>	When x is a collection you can test to see if y is in it.
<code>__eq__(self,y)</code>	<code>x == y</code>	Returns True or False depending on the values of x and y.
<code>__ge__(self,y)</code>	<code>x >= y</code>	Returns True or False depending on the values of x and y.
<code>__getitem__(self,y)</code>	<code>x[y]</code>	Returns the item at the yth position in x.
<code>__gt__(self,y)</code>	<code>x > y</code>	Returns True or False depending on the values of x and y.
<code>__hash__(self)</code>	<code>hash(x)</code>	Returns an integral value for x.
<code>__int__(self)</code>	<code>int(x)</code>	Returns an integer representation of x.
<code>__iter__(self)</code>	<code>for v in x</code>	Returns an iterator object for the sequence x.
<code>__le__(self,y)</code>	<code>x <= y</code>	Returns True or False depending on the values of x and y.
<code>__len__(self)</code>	<code>len(x)</code>	Returns the size of x where x has some length attribute.
<code>__lt__(self,y)</code>	<code>x < y</code>	Returns True or False depending on the values of x and y.
<code>__mod__(self,y)</code>	<code>x % y</code>	Returns the value of x modulo y. This is the remainder of x/y.
<code>__mul__(self,y)</code>	<code>x * y</code>	Returns the product of x and y.
<code>__ne__(self,y)</code>	<code>x != y</code>	Returns True or False depending on the values of x and y.
<code>__neg__(self)</code>	<code>-x</code>	Returns the unary negation of x.
<code>__repr__(self)</code>	<code>repr(x)</code>	Returns a string version of x suitable to be evaluated by the eval function.
<code>__setitem__(self,i,y)</code>	<code>x[i] = y</code>	Sets the item at the ith position in x to y.
<code>__str__(self)</code>	<code>str(x)</code>	Return a string representation of x suitable for user-level interaction.
<code>__sub__(self,y)</code>	<code>x - y</code>	The difference of two objects.

Python Operator Magic Methods

16.2 Example 1 - simple operator overloading

Here is a simple example illustrating operator overloading in Python.

```
class position:
    def __init__(self, x, y):
        self.x=x
```

```
self.y=y

def X(self):
    return(self.x)

def Y(self):
    return(self.y)

def __add__(self,position_2):
    return position( self.x + position_2.x , self.y + posi-
tion_2.y )

def main():
    p1=position(10,20)
    p2=position(100,200)
    p3=position(1000,2000)
    print(p1.X(),p1.Y())
    print(p2.X(),p2.Y())
    print(p3.X(),p3.Y())
    p3=p1+p2
    print(p3.X(),p3.Y())

if __name__ == "__main__":
    main()
```

Here is the output.

```
10 20
100 200
1000 2000
110 220
```

16.3 Problems

1. Add a subtract method to the above example.

‘When I use a word,’ Humpty Dumpty said, in a rather scornful tone, ‘it means just what I choose it to mean - neither more nor less’

‘The question is,’ said Alice, ‘whether you can make words mean so many different things.’

Lewis Carroll, *Through the Looking Glass and What Alice found there*.

17 Decimals, fractions, random numbers

17.1 Introduction

Python provides a number of additional mathematic modules. In this chapter we will have a look at

decimal

fraction

random

The information is taken from:

<https://docs.python.org/3/library/decimal.html>

<https://docs.python.org/3/library/decimal.html#module-decimal>

Python's built in floating point operations are based on C's double, which in turn is based on the IEEE 64 bit floating point representation in most cases. This is suitable for a wide range of science and engineering calculations.

Where more accuracy is required Python provides the decimal module.

17.2 The Decimal module

The decimal module provides support for fast correctly-rounded decimal floating point arithmetic. It offers several advantages over the float datatype:

Decimal “is based on a floating-point model which was designed with people in mind, and necessarily has a paramount guiding principle – computers must provide an arithmetic that works in the same way as the arithmetic that people learn at school.” – excerpt from the decimal arithmetic specification.

Decimal numbers can be represented exactly. In contrast, numbers like 1.1 and 2.2 do not have exact representations in binary floating point. End users typically would not expect $1.1 + 2.2$ to display as 3.3000000000000003 as it does with binary floating point.

The exactness carries over into arithmetic. In decimal floating point, $0.1 + 0.1 + 0.1 - 0.3$ is exactly equal to zero. In binary floating point, the result is 5.5511151231257827e-017. While near to zero, the differences prevent reliable equality testing and differences can accumulate. For this reason, decimal is preferred in accounting applications which have strict equality invariants.

The decimal module incorporates a notion of significant places so that $1.30 + 1.20$ is 2.50. The trailing zero is kept to indicate significance. This is the customary presentation for monetary applications. For multiplication, the “school-

book” approach uses all the figures in the multiplicands. For instance, $1.3 * 1.2$ gives 1.56 while $1.30 * 1.20$ gives 1.5600.

Unlike hardware based binary floating point, the decimal module has a user alterable precision (defaulting to 28 places) which can be as large as needed for a given problem:

We have a look at a small number of decimal examples.

17.3 Example 1 - using `getcontext()`

This example just prints out the default context for decimal arithmetic.

Here is the source.

```
from decimal import *
print(getcontext())
```

Here is the output.

```
Context (
prec=28,
rounding=ROUND_HALF_EVEN,
Emin=-999999,
Emax=999999,
capitals=1,
clamp=0,
flags=[],
traps=[InvalidOperation, DivisionByZero, Overflow])
```

The output has been split over several lines to make it easier to read.

17.4 Function availability

<code>abs(x)</code>	Returns the absolute value of x.
<code>add(x, y)</code>	Return the sum of x and y.
<code>canonical(x)</code>	Returns the same Decimal object x.
<code>compare(x, y)</code>	Compares x and y numerically.
<code>compare_signal(x, y)</code>	Compares the values of the two operands numerically.
<code>compare_total(x, y)</code>	Compares two operands using their abstract representation.
<code>compare_total_mag(x, y)</code>	Compares two operands using their abstract representation, ignoring sign.
<code>copy_abs(x)</code>	Returns a copy of x with the sign set to 0.
<code>copy_negate(x)</code>	Returns a copy of x with the sign inverted.
<code>copy_sign(x, y)</code>	Copies the sign from y to x.
<code>divide(x, y)</code>	Return x divided by y.
<code>divide_int(x, y)</code>	Return x divided by y, truncated to an integer.
<code>divmod(x, y)</code>	Divides two numbers and returns the integer part of the result.
<code>exp(x)</code>	Returns $e ** x$.
<code>fma(x, y, z)</code>	Returns x multiplied by y, plus z.
<code>is_canonical(x)</code>	Returns True if x is canonical; otherwise returns False.
<code>is_finite(x)</code>	Returns True if x is finite; otherwise returns False.
<code>is_infinite(x)</code>	Returns True if x is infinite; otherwise returns False.

<code>is_nan(x)</code>	Returns True if x is a qNaN or sNaN; otherwise returns False.
<code>is_normal(x)</code>	Returns True if x is a normal number; otherwise returns False.
<code>is_qnan(x)</code>	Returns True if x is a quiet NaN; otherwise returns False.
<code>is_signed(x)</code>	Returns True if x is negative; otherwise returns False.
<code>is_snan(x)</code>	Returns True if x is a signaling NaN; otherwise returns False.
<code>is_subnormal(x)</code>	Returns True if x is subnormal; otherwise returns False.
<code>is_zero(x)</code>	Returns True if x is a zero; otherwise returns False.
<code>ln(x)</code>	Returns the natural (base e) logarithm of x.
<code>log10(x)</code>	Returns the base 10 logarithm of x.
<code>logb(x)</code>	Returns the exponent of the magnitude of the operand's MSD.
<code>logical_and(x, y)</code>	Applies the logical operation and between each operand's digits.
<code>logical_invert(x)</code>	Invert all the digits in x.
<code>logical_or(x, y)</code>	Applies the logical operation or between each operand's digits.
<code>logical_xor(x, y)</code>	Applies the logical operation xor between each operand's digits.
<code>max(x, y)</code>	Compares two values numerically and returns the maximum.
<code>max_mag(x, y)</code>	Compares the values numerically with their sign ignored.
<code>min(x, y)</code>	Compares two values numerically and returns the minimum.
<code>min_mag(x, y)</code>	Compares the values numerically with their sign ignored.
<code>minus(x)</code>	Minus corresponds to the unary prefix minus operator in Python.
<code>multiply(x, y)</code>	Return the product of x and y.
<code>next_minus(x)</code>	Returns the largest representable number smaller than x.
<code>next_plus(x)</code>	Returns the smallest representable number larger than x.
<code>next_toward(x, y)</code>	Returns the number closest to x, in direction towards y.
<code>normalize(x)</code>	Reduces x to its simplest form.
<code>number_class(x)</code>	Returns an indication of the class of x.
<code>plus(x)</code>	Plus corresponds to the unary prefix plus operator in Python. This operation applies the context precision and rounding, so it is not an identity operation.
<code>power(x, y, modulo=None)</code>	Return x to the power of y, reduced modulo modulo if given. With two arguments, compute $x^{**}y$. If x is negative then y must be integral. The result will be inexact unless y is integral and the result is finite and can be expressed exactly in 'precision' digits. The rounding mode of the context is used. Results are always correctly-rounded in the Python version. With three arguments, compute $(x^{**}y) \% \text{modulo}$. For the three argument form, the following restrictions on the arguments hold:
	all three arguments must be integral
	y must be nonnegative
	at least one of x or y must be nonzero

modulo must be nonzero and have at most 'precision' digits

The value resulting from `Context.power(x, y, modulo)` is equal to the value that would be obtained by computing $(x**y) \% \text{modulo}$ with unbounded precision, but is computed more efficiently. The exponent of the result is zero, regardless of the exponents of `x`, `y` and `modulo`. The result is always exact.

<code>quantize(x, y)</code>	Returns a value equal to <code>x</code> (rounded), having the exponent of <code>y</code> .
<code>radix()</code>	Just returns 10, as this is Decimal, :)
<code>remainder(x, y)</code>	Returns the remainder from integer division. The sign of the result, if non-zero, is the same as that of the original dividend.
<code>remainder_near(x, y)</code>	Returns $x - y * n$, where <code>n</code> is the integer nearest the exact value of x / y (if the result is 0 then its sign will be the sign of <code>x</code>).
<code>rotate(x, y)</code>	Returns a rotated copy of <code>x</code> , <code>y</code> times.
<code>same_quantum(x, y)</code>	Returns True if the two operands have the same exponent.
<code>scaleb(x, y)</code>	Returns the first operand after adding the second value its exp.
<code>shift(x, y)</code>	Returns a shifted copy of <code>x</code> , <code>y</code> times.
<code>sqrt(x)</code>	Square root of a non-negative number to context precision.
<code>subtract(x, y)</code>	Return the difference between <code>x</code> and <code>y</code> .
<code>to_eng_string(x)</code>	Converts a number to a string, using scientific notation.
<code>to_integral_exact(x)</code>	Rounds to an integer.
<code>to_sci_string(x)</code>	Converts a number to a string using scientific notation

17.5 Example 2 - values for the maths constants e and pi

Here is the source.

```
from decimal import *
import math

def pi():
    """Compute Pi to the current precision.

    >>> print(pi())
    3.141592653589793238462643383

    """
    getcontext().prec += 2 # extra digits for intermediate
    steps
    three = Decimal(3) # substitute "three=3.0" for
    regular floats
    lasts, t, s, n, na, d, da = 0, three, 3, 1, 0, 0, 24
    while s != lasts:
        lasts = s
        n, na = n+na, na+8
        d, da = d+da, da+32
        t = (t * n) / d
        s += t
```

```

    getcontext().prec -= 2
    return +s                # unary plus applies the new
precision

def main():

    print(pi())
    print(Decimal(math.pi))

print('3.14159265358979323846264338327950288419716939937510')
    print(Decimal(1).exp())
    print(Decimal(math.e))

print('2.71828182845904523536028747135266249775724709369995')

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

3.141592653589793238462643383
3.141592653589793115997963468544185161590576171875
3.14159265358979323846264338327950288419716939937510
2.718281828459045235360287471
2.718281828459045090795598298427648842334747314453125
2.71828182845904523536028747135266249775724709369995

```

Care must be taken when using extended precision.

17.6 Example 3 - summation using float and decimal

Here is the source.

```

from decimal import *

def main():

    print( sum( [Decimal('0.1')]*10 ) )
    print( Decimal('1.0') )
    print( sum( [Decimal('0.1')]*10 ) == Decimal('1.0') )

    print(sum([0.1]*10))
    print(1.0)
    print( sum([0.1]*10) == 1.0 )

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

1.0
1.0
True
0.9999999999999999

```

1.0
False

17.7 The Fraction module

The information in this section is taken from
<http://docs.python.org/3/library/fractions.html>

The fractions module provides support for rational number arithmetic. A Fraction instance can be constructed from a pair of integers, from another rational number, or from a string.

```
class fractions.Fraction(numerator=0, denominator=1)
class fractions.Fraction(other_fraction)
class fractions.Fraction(float)
class fractions.Fraction(decimal)
class fractions.Fraction(string)
```

The first version requires that numerator and denominator are instances of numbers.Rational and returns a new Fraction instance with value numerator/denominator. If denominator is 0, it raises a ZeroDivisionError. The second version requires that other_fraction is an instance of numbers.Rational and returns a Fraction instance with the same value. The next two versions accept either a float or a decimal.Decimal instance, and return a Fraction instance with exactly the same value. Note that due to the usual issues with binary floating-point (see Floating Point Arithmetic: Issues and Limitations), the argument to Fraction(1.1) is not exactly equal to 11/10, and so Fraction(1.1) does not return Fraction(11, 10) as one might expect. (But see the documentation for the limit_denominator() method below.) The last version of the constructor expects a string or unicode instance. The usual form for this instance is:

```
[sign] numerator ['/' denominator]
```

where the optional sign may be either '+' or '-' and numerator and denominator (if present) are strings of decimal digits. In addition, any string that represents a finite value and is accepted by the float constructor is also accepted by the Fraction constructor. In either form the input string may also have leading and/or trailing whitespace.

17.8 Example 4 - simple fraction usage

Here is the source code.

```
from fractions import Fraction
from decimal import Decimal

def main():

    f1 = Fraction(16, -10)
    f2 = Fraction(123)
    f3 = Fraction()
    f4 = Fraction('3/7')
    f5 = Fraction(' -3/7 ')
    f6 = Fraction('1.414213 \t\n')
    f7 = Fraction('-.125')
    f8 = Fraction('7e-6')
    f9 = Fraction(2.25)
    f10 = Fraction(1.1)
```

```

f11 = Fraction(Decimal('1.1'))

print(" f1 = " , f1)
print(" f2 = " , f2)
print(" f3 = " , f3)
print(" f4 = " , f4)
print(" f5 = " , f5)
print(" f6 = " , f6)
print(" f7= " , f7)
print(" f8 = " , f8)
print(" f9 = " , f9)
print(" f10 = " , f10)
print(" f11 = " , f11)

if ( __name__ == "__main__" ):
    main()

```

Here is the output.

```

f1 = -8/5
f2 = 123
f3 = 0
f4 = 3/7
f5 = -3/7
f6 = 1414213/1000000
f7= -1/8
f8 = 7/1000000
f9 = 9/4
f10 = 2476979795053773/2251799813685248
f11 = 11/10

```

17.9 The Random module

The information in this section is taken from.

<https://docs.python.org/3/library/random.html>

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the semi-open range `[0.0, 1.0)`. Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of $2^{19937}-1$. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the `random.Random` class. You can instantiate your own instances of `Random` to get generators that don't share state.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the `random()`, `seed()`, `getstate()`, and `setstate()` methods. Optionally, a new generator can supply a `getrandbits()` method — this allows `randrange()` to produce selections over an arbitrarily large range.

The `random` module also provides the `SystemRandom` class which uses the system function `os.urandom()` to generate random numbers from sources provided by the operating system.

17.10 Example 5 - simple random usage

Here is the program.

```
import random

def main():

    r1 = random.random()
    # Random float x, 0.0 <= x < 1.0
    r2 = random.uniform(1, 10)
    # Random float x, 1.0 <= x < 10.0
    r3 = random.randrange(10)
    # Integer from 0 to 9
    r4 = random.randrange(0, 101, 2)
    # Even integer from 0 to 100
    r5 = random.choice('abcdefghij')
    # Single random element
    r6 = [1, 2, 3, 4, 5, 6, 7]
    r7 = random.sample([1, 2, 3, 4, 5], 3)
    # Three samples without replacement

    print(" r1 = " , r1)
    print(" r2 = " , r2)
    print(" r3 = " , r3)
    print(" r4 = " , r4)
    print(" r5 = " , r5)
    print(" r6 = " , r6)
    random.shuffle(r6)
    print(" r6 = " , r6)
    print(" r7 = " , r7)

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
r1 = 0.9522312278705292
r2 = 1.8207726773335011
r3 = 3
r4 = 30
r5 = h
```



```
r6 = [1, 2, 3, 4, 5, 6, 7]
r6 = [4, 5, 3, 2, 6, 1, 7]
r7 = [3, 5, 1]
```

17.11 Problems

1. Run the examples in this chapter.

18 Databases and sqlite

18.1 Introduction to database management systems

There are three main types of data

- tabular data

- text based data

- spatial data

and database management systems have been developed to handle each of them. The main database management systems (dbms) to handle tabular data are relational systems. There is a separate chapter on SQL and the Relational Model.

18.2 SQL based systems and Python

The following information is taken from

<https://wiki.python.org/moin/SQL%20Server>

18.2.1 Microsoft SQL Server

URL

<http://www.microsoft.com/sql/default.msp>

licence commercial/proprietary software, although a free (gratis) edition "SQL Server 2008 R2 Express" is available for platforms Windows 2000 and later

Pros

SQL Server is a robust and fully-featured database, and it performs very well. Moreover, I have not had any problems using this database with Python.

The SQL Server Express versions are free to download, use and can even be redistributed with products.

Cons

Windows only.

SQL Server comes in various flavours. The latest free version has a 10GB database size limit. It comes with the GUI tools and Reporting Services. The standard and other versions include many extra features.

18.2.2 DB API 2.0 Drivers

adodbapi

URL

<http://adodbapi.sourceforge.net/>

SourceForge

<http://sourceforge.net/projects/adodbapi>

licence LGPL platforms Windows only

pymssql

URL

<http://pymssql.org>

licence LGPL platforms Windows and Unix

mssql

URL

<http://www.object-craft.com.au/projects/mssql/>

licenceBSD platformsWindows

mxODBC

URL

<http://www.egenix.com/>

LicenseeGenix.com Commercial License PlatformsWindows, Unix, Mac OS X, FreeBSD, Solaris, AIX, other platforms on request Python versions2.4 - 2.7

mxODBC requires an ODBC driver to talk to SQL Server. On Windows, you can use the MS SQL Server Native Client ODBC driver for Windows, on the other platforms, there are several commercial ODBC high quality drivers available, an open-source <http://www.freetds.org/> FreeTDS ODBC driver for Unix platforms and the free MS SQL Server Native Client ODBC driver for Linux x64.

mxODBC comes with full support for stored procedures, multiple result sets, Unicode, a common interface on all platforms and many other useful features.

pyodbc

URL

<http://code.google.com/p/pyodbc>

LicenseMIT PlatformsWindows, Linux, MacOS X, FreeBSD, Solaris, Any (source provided) Python versions2.4 - 3.2

Actively maintained Open Source project.

Precompiled binaries are available for Windows. RedHat Enterprise Linux, Centos, and Fedora have precompiled RPMs available in their Extras repositories.

Supports ANSI and Unicode data and SQL statements and includes an extensive set of unit tests for SQL Server. pyODBC require ODBC driver to work correctly with SQL Server. You may download latest SQL Server ODBC driver and use it freely. Or you may choose Microsoft ODBC driver for that needs which is posted above in mxODBC driver description.

pypyodbc (Pure Python)

URL

<http://code.google.com/p/pypyodbc>

LicenseMIT PlatformsWindows, Linux Python versions2.4 - 3.3

A Hello World script of pypyodbc database programing

Connect SQL Server in 3 steps with pypyodbc on Linux

Run SQLAlchemy on PyPy with pypyodbc driver PyPyODBC is a pure Python script, it runs on CPython / IronPython / PyPy , Version 2.4 / 2.5 / 2.6 / 2.7 , Win / Linux , 32 / 64 bit.

Almost totally same usage as pyodbc (can be seen as a re-implementation of pyodbc in pure Python).

Simple - the whole module is implemented in a single python script with less than 3000 lines.

Built-in Access MDB file creation and compression functions on Windows.

ODBC

It is possible to connect to an SQL Server database using ODBC, either the mxODBC driver or the one included with Win32all. However, this is not recommended - adodbapi is a better solution, in part because it supports unicode.

Comment: This is actually not true at all: ODBC is the native API used for SQL Server and does support Unicode all the way. In fact, ODBC is the preferred way of accessing SQL Server if you care for performance. Microsoft has just released the SQL Server Native Client which is an extended ODBC driver for SQL Server. ADO is just a layer on top of the ODBC interface and a lot slower as a result. See e.g. MS TechNet for a comparison of ODBC, OLE DB and ADO, or this cookbook entry.

Comment: Note about the comment above -- just because it should be pointed out, mxODBC is not a free product from what I can see, and the 'cookbook entry' from 2005 referenced above indicates that it is.

We will look at a simple offering using SQLite/

18.3 SQLite

Here is their site.

<http://sqlite.org/about.html>

The following information is taken from that site.

SQLite is an in-process library that implements a self-contained, server less, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced below 300KiB. SQLite can also be made to run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving

hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists and minute-by-minute chronologies of bug reports and code changes.

The SQLite code base is supported by an international team of developers who work on SQLite full-time. The developers continue to expand the capabilities of SQLite and enhance its reliability and performance while maintaining backwards compatibility with the published interface spec, SQL syntax, and database file format. The source code is absolutely free to anybody who wants it, but professional support is also available.

We the developers hope that you find SQLite useful and we charge you to use it well: to make good and beautiful products that are fast, reliable, and simple to use. Seek forgiveness for yourself as you forgive others. And just as you have received SQLite for free, so also freely give, paying the debt forward.

We will use SQLite in this chapter.

Some sample SQLite web sites

<http://pythoncentral.io/introduction-to-sqlite-in-python/>
<http://www.blog.pythonlibrary.org/2012/07/18/python-a-simple-step-by-step-sqlite-tutorial/>

18.4 On line documentation at W3 Schools

Visit

<https://www.w3schools.com/sql/default.asp>

for a very good free reference.

18.5 SQL examples

In this example we will create a simple database taken from a UNEP database used in the production of the following publication.

United Nations Environment Programme, Environmental Data Report,
1989-1990, Basil Blackwell Ltd., ISBN 0-631-16987-3

Chapter 9 of the report is on natural disasters, and in this example we will look at some of the earthquake and tsunami tables. We will look at the following steps

database creation

table creation

loading tables

querying tables

I had two secondments to UNEP where I worked with a wide range of environmental data sets.

18.5.1 Example 1 - Database creation

Here is the short command file to create our database. Running these examples will create a file called chapter09, which is our database.

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('chapter09')
db.close()
```

18.5.2 Example 2 - Table creation

We will create three tables. These are

regions - a table with country and region classifications

tsunami - a table with tsunami data

earthqk - a table with earthquake data

The regions table contains the following columns

```
Regions WITH +
region +
countryn +
regions +
Country +
Countryf +
WHOreg +
WHOregs +
ECEmembr +
WRlreg +
WRlregs +
FAOclass +
FAOreg +
FAOsortn
```

The above information was taken from the relational dbms that we used at UNEP.

The tsunami table contains the following columns

```
runup WITH +
region +
regions +
Country +
zregionn +
latitude +
longitud +
comments +
year +
Month +
Day +
```

height +
 ahh +
 amm +
 ass +
 tmm +
 tss +
 countrys +
 aregion

The earthqk table contains the following columns

earthqk WITH +
 countrys +
 Earthqkn +
 year +
 Yearn +
 Month +
 Day +
 Time +
 latitude +
 longitud +
 depth +
 magnitud +
 unknown1 +
 unknown2 +
 deaths +
 strength +
 refs +
 sregion

Here are details of the data types for each of the columns in the complete database.

1974	TEXT	1
1975	TEXT	1
1976	TEXT	1
1977	TEXT	1
1978	TEXT	1
1979	TEXT	1
1980	TEXT	1
1981	TEXT	1
1982	TEXT	1
1983	TEXT	1
1984	TEXT	1
1985	TEXT	1
ahh	INTEGER	
amm	INTEGER	

area	TEXT	60
aregion	TEXT	40
ass	INTEGER	
cause	TEXT	3
cnotes	TEXT	20
comment	TEXT	50
comments	TEXT	40
Country	TEXT	25
Countryf	TEXT	100
countryn	INTEGER	KEY
damage	TEXT	1
Day	INTEGER	
deathn	TEXT	1
deaths	INTEGER	
depth	REAL	
Describe	TEXT	1500
Earthqkn	INTEGER	
ECEmembr	TEXT	1
f_depth	TEXT	2
FAOclass	TEXT	27
FAOreg	TEXT	42
FAOsortn	INTEGER	
height	TEXT	1
latitude	REAL	
location	TEXT	30
longitud	REAL	
magnitud	REAL	
Month	INTEGER	
note	TEXT	1
notes	TEXT	1400
r_up	REAL	
ref	TEXT	30
refer	TEXT	1500
refhead	TEXT	1500
refs	TEXT	24
reg_no	INTEGER	
region	TEXT	13
regions	INTEGER	
site	TEXT	25
source	TEXT	28
sregion	TEXT	60
strength	TEXT	13
Tablenam	TEXT	8
Time	INTEGER	
tint	REAL	
tmag	REAL	
tmm	INTEGER	
Today	DATE	
total	INTEGER	

tregion	TEXT	50
tss	INTEGER	
unk	TEXT	1
unknown1	INTEGER	
unknown2	TEXT	1
validity	INTEGER	
WHOreg	TEXT	25
WHOregs	INTEGER	
WRIreg	TEXT	23
WRIregs	INTEGER	
year	INTEGER	
Yearn	TEXT	1
zregionn	INTEGER	

SQLite supports text, real and integer types.

Here is the command file used to create the tables.

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('chapter09')
# Get a cursor object
cursor = db.cursor()
cursor.execute('''
create table regions
(
region text ,
countryn integer ,
regions integer ,
country text ,
countryf text ,
whoreg text ,
WHOregs integer ,
ECEmembr text ,
WRIreg text ,
WRIregs integer ,
FAOclass text ,
FAOreg text ,
FAOsortn integer
)
''')
db.commit()
print(" Regions table created")
cursor.execute('''
create table tsunami
(
region text ,
regions integer ,
country text ,
zregionn integer ,
latitude real ,
```

```

longitud real ,
comments text ,
year      integer ,
Month     year ,
Day       integer ,
height    text ,
ahh       integer ,
amm       integer ,
ass       integer ,
tmm       integer ,
tss       integer ,
countryn  integer ,
aregion   text
)
'''
db.commit()
print(" Tsunami table created")
cursor.execute('''
create table earthqk
(
countryn integer ,
Earthqkn integer ,
year      integer ,
Yearn     text ,
Month     integer ,
Day       integer ,
Time      integer ,
latitude  real ,
longitud  real ,
depth     real ,
magnitud  real ,
unknown1  integer ,
unknown2  text ,
deaths    integer ,
strength  text ,
refs      text ,
sregion   text
)
''')
db.commit()
print(" Earthquake table created")
db.close()

```

18.5.3 Example 3 - loading the earthqk table

Here are extracts of the command files used to load the tables.

```

import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('chapter09')
# Get a cursor object
cursor = db.cursor()

```

```

cursor.execute('''INSERT INTO earthqk VALUES
(199,1,-2000,"-",null,null,null,38.,58.2,null,null,null,"S",
null,null,"44","Western Turkenia")''')
cursor.execute('''INSERT INTO earthqk VALUES
(101,2,-1566,"-",null,null,null,null,null,null,null,10,"-",
null,null,"611","Jericho")''')
cursor.execute('''INSERT INTO earthqk VALUES
(95,3,-600,"-",null,null,null,35.,45.,null,null,null,"-",null,
null,"57","Sinkarah: Temple of Taras")''')
cursor.execute('''INSERT INTO earthqk VALUES
(77,4,-225,"-",null,null,null,36.,28.5,null,null,null,"-",
null,"Severe","147","Rhodes")''')
cursor.execute('''INSERT INTO earthqk VALUES
(39,5,-186,"-",2,22,null,33.4,104.8,null,null,null,"-",760,
null,"48",null)''')
cursor.execute('''INSERT INTO earthqk VALUES
(39,6,-70,"-",6,1,null,36.3,118.,null,null,9,"-",6000,null,
"48",null)''')
...
...
print(" Earthquake table loaded")
db.commit()
db.close()

```

18.5.4 Example 4 - loading the regions table

Here is the second.

```

import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('chapter09')
# Get a cursor object
cursor = db.cursor()
cursor.execute('''INSERT INTO regions VALUES
("Africa",3,1,"Algeria",
"People's Democratic Republic of Algeria","Africa",1,
null,"Africa",1,"Developing market economies","Africa",5)''')
cursor.execute('''INSERT INTO regions VALUES
("Africa",6,1,"Angola","People's Republic of Angola",
"Africa",1,null,"Africa", 1,"Developing market economies",
"Africa",5)''')
cursor.execute('''INSERT INTO regions VALUES
("Africa",17,1,"Benin","People's Republic of Benin",
"Africa",1,null,"Africa", 1,"Developing market economies",
"Africa",5)''')
...
...
print(" Regions table loaded")
db.commit()
print(" Database commit")
db.close()

```

```
print(" Database close")
```

18.5.5 Example 5 - loading the tsunami table

Here is the third,

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('chapter09')
# Get a cursor object
cursor = db.cursor()
cursor.execute('''INSERT INTO tsunami VALUES
("Asia",4,"China",4,23.13,113.33,null,1765,5,null,
" ",null,null,null,null,null,39,null)''')
cursor.execute('''INSERT INTO tsunami VALUES
("Asia",4,"China",4,22.3,114.18,null,1960,5,22,
" ",23,22,20,27,9,39,null)''')
cursor.execute('''INSERT INTO tsunami VALUES
("Asia",4,"China; Taiwan",4,25.15,121.75,null,1917,5,6,
" ",null,null,null,4,42,40,"Taiwan; Keelung")''')
...
...
print(" Tsunami table loaded")
db.commit()
print(" Database commit")
db.close()
print(" Database close")
```

18.5.6 Example 6 - Querying the tables

Here is the query command file.

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('chapter09')
# Get a cursor object
cursor = db.cursor()
print(" Regions query")
for row in cursor.execute("SELECT * FROM regions"):
    print(row)
db.close()
```

18.6 Using SQLite from the command line

You can also use SQLite from the command line.

18.7 Creating a database of the Met Office data

In this section we create a database based on the data in the Met Office station data.

18.7.1 Example 7 - creating the database

Here is the source file.

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
db.close()
```

18.7.2 Example 8 - creating a table for one of the sites

Here is the source file

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
# Get a cursor object
cursor = db.cursor()
cursor.execute('''
create table cwmystwyth
(
s_year    integer ,
s_month   integer ,
t_max     real    ,
t_min     real    ,
af_days   integer ,
rain      real    ,
sun       real
)
''')
db.commit()
print(" Cwmystwyth table created")
db.close()
```

18.7.3 Example 9 - loading data into the table

Here is a sample of the source file.

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
# Get a cursor object
cursor = db.cursor()
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,1,4.5,-1.9,20,null,57.2)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,2,7.3,0.9,15,null,87.2)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,3,8.4,3.1,3,null,81.6)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,4,10.8,3.7,1,null,107.4)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,5,15.8,5.8,1,null,213.5)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,6,16.9,8.2,0,null,209.4)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,7,18.5,9.5,0,null,167.8)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,8,19.0,10.5,0,null,164.8)''')
cursor.execute('''INSERT INTO cwmystwyth VALUES(1959,9,18.3,5.9,0,null,196.5)''')
```

```

cursor.execute(''INSERT INTO cwmystwyth VAL-
UES(1959,10,14.8,7.9,1,null,101.1)''')
cursor.execute(''INSERT INTO cwmystwyth VAL-
UES(1959,11,8.8,3.9,3,null,38.9)''')
cursor.execute(''INSERT INTO cwmystwyth VAL-
UES(1959,12,7.2,2.5,3,null,19.2)''')
cursor.execute(''INSERT INTO cwmystwyth VAL-
UES(1960,1,6.3,0.6,15,null,30.7)''')
cursor.execute(''INSERT INTO cwmystwyth VAL-
UES(1960,2,5.3,-0.3,17,null,50.2)''')

```

This is just a subset.

18.7.4 Example 10 - simple table query

Here is the source file.

```

import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
# Get a cursor object
cursor = db.cursor()
print(" Cwmystwyth query")
for row in cursor.execute("SELECT * FROM cwmystwyth order by
rain"):
    print(row)
db.close()

```

Here is some sample output.

```

Cwmystwyth query
(1959, 1, 4.5, -1.9, 20, None, 57.2)
(1959, 2, 7.3, 0.9, 15, None, 87.2)
(1959, 3, 8.4, 3.1, 3, None, 81.6)
(1959, 4, 10.8, 3.7, 1, None, 107.4)
...
...
(1967, 10, 11.5, 6.5, 0, 400.1, 43.6)
(2002, 2, 8.6, 2.9, 5, 401.3, 38.2)
(1965, 12, 7.0, 1.6, 8, 417.3, 31.4)
(1966, 12, 7.5, 2.7, 3, 419.8, 17.9)
(2000, 11, 8.2, 3.5, 1, 424.4, 15.2)
(2009, 11, 9.4, 4.9, 0, 425.4, 34.0)

```

The row is the record returned by the SQL select statement.

18.7.5 Example 11 - computing averages

Here is the source file.

```

import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
# Get a cursor object
cursor = db.cursor()

```

```

months=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep",
"Oct","Nov","Dec"]
print(" Cwmystwyth monthly averages")
print("          mm          ins")
for month in range(1,13):
    for row in cursor.execute("SELECT AVG(rain) FROM cwmystwyth
where s_month == ?",(month,)):
        line=row[0]
        average=float(line)
        average_ins=average/25.4
        print(" {0:}   {1:7.2f}   {2:7.2f}").for-
mat(months[month-1],average,average_ins))
db.close()

```

Here is the output.

```

Cwmystwyth monthly averages
          mm          ins
Jan      186.55         7.34
Feb      134.09         5.28
Mar      139.91         5.51
Apr      109.43         4.31
May      104.87         4.13
Jun      107.35         4.23
Jul      124.83         4.91
Aug      137.44         5.41
Sep      150.97         5.94
Oct      189.16         7.45
Nov      205.41         8.09
Dec      210.75         8.30

```

Here we loop over the months as well.

18.7.6 Example 12 - Finding the wettest month and displaying the year, month and rainfall

Here is the source.

```

import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
# Get a cursor object
cursor = db.cursor()
print(" Cwmystwyth maximum monthly rainfall")
print("          mm          ins")
for row in cursor.execute("SELECT s_year , s_month ,
MAX(rain) FROM cwmystwyth"):
    y=row[0]
    m=row[1]
    rmm = row[2]
    rins= rmm/25.4
    print(" {0:}   {1:}   {2:7.2f}   {3:7.2f}").for-
mat(y,m,rmm,rins)

```

db.close()

Here is the output.

```
Cwmystwyth maximum monthly rainfall
                mm      ins
2009  11  425.40    16.75
```

18.7.7 Example 13 - Finding the wettest months and displaying the year, month and rainfall

Here is the source.

```
import sqlite3
db = sqlite3.connect(':memory:')
db = sqlite3.connect('met_office')
# Get a cursor object
cursor = db.cursor()
print(" Cwmystwyth maximum monthly rainfall values")
print("                mm      ins")
for row in cursor.execute("SELECT s_year , s_month ,
MAX(rain) FROM cwmystwyth GROUP BY s_month"):
    y=row[0]
    m=row[1]
    rmm = row[2]
    rins= rmm/25.4
    print(" {0:4}  {1:2}  {2:7.2f}  {3:7.2f}").for-
mat(y,m,rmm,rins)
db.close()
```

Here is the output.

```
Cwmystwyth maximum monthly rainfall values
                mm      ins
2008   1  348.90    13.74
2002   2  401.30    15.80
1981   3  350.00    13.78
1970   4  222.70     8.77
1979   5  247.00     9.72
1985   6  259.00    10.20
2007   7  260.80    10.27
1992   8  257.40    10.13
2004   9  283.50    11.16
1967  10  400.10    15.75
2009  11  425.40    16.75
1966  12  419.80    16.53
```

We could make the y and m variables into numpy arrays of course.

18.8 Example 14 - doing monthly average calculations using the genfromtxt example in the IO chapter

Here is an example based on the genfromtxt example in chapter 11. Here is the new source.


```

import numpy as np
import math
data_file_name="cwmystwythdata.txt"
month_names = ["January" , "Februry" , "March" , "April" ,
"May" , "June" , "July" , "August" , "September" , "October"
, "November" , "December"]
matrix = np.genfromtxt( data_file_name, \
                        skip_header=7 , \
                        skip_footer=1 , \
                        usecols=(0,1,2,3,4,5,6), \
                        autostrip=True , \

dtype=(int,int,float,float,int,float,float), \
      missing_values={"---"},\
      )

n=matrix.size
print(n)
print(type(matrix))
n_months=12
month=0
monthly_sums          = np.zeros([n_months],dtype=np.float64)
monthly_averages      = np.zeros([n_months],dtype=np.float64)
monthly_averages_ins  = np.zeros([n_months],dtype=np.float64)
monthly_counts        = np.zeros([n_months],dtype=np.int32  )
monthly_nans          = np.zeros([n_months],dtype=np.int32  )
for i in range(0,n):
    row=matrix[i]
    month      = row[1] - 1
    rainfall   = row[5]
    if ( math.isnan(rainfall) ):
        monthly_nans[month]      = monthly_nans[month]      + 1
    else:
        monthly_counts[month] = monthly_counts[month] + 1
        monthly_sums[month]=monthly_sums[month] + rainfall
print(" Month          mm          in      Valid      Miss-
ing")
for i in range(0,n_months):
    monthly_averages[i]          = monthly_sums[i] /
monthly_counts[i]
    monthly_averages_ins[i] = monthly_averages[i] / 25.4
    print("  {:12s}      {:6.2f}      {:6.2f}      {:4d}
{:4d}").format(month_names[i],monthly_averages[i],monthly_aver-
ages_ins[i],monthly_counts[i],monthly_nans[i]))

```

Here is the output.

618

```
<class 'numpy.ndarray'>
```

Month	mm	in	Valid	Missing
January	186.55	7.34	49	2
February	134.09	5.28	49	2

March	139.91	5.51	50	2
April	109.43	4.31	49	2
May	104.87	4.13	50	2
June	107.35	4.23	50	2
July	124.83	4.91	50	2
August	137.44	5.41	50	2
September	150.97	5.94	50	2
October	189.16	7.45	49	2
November	205.41	8.09	48	3
December	210.75	8.30	46	5

Compare the output to that from example 11. Do the results agree?

18.9 Problems

1. Run the examples in this chapter.

2. If you visit

<http://www.un.org/en/members>

you will see a list of UN member states. Here is a list based on that information.

Albania	14/12/1955
Algeria	08/10/1962
Andorra	28/07/1993
Angola	01/12/1976
Antigua and Barbuda	11/11/1981
Argentina	24/10/1945
Armenia	02/03/1992
Australia	01/11/1945
Austria	14/12/1955
Azerbaijan	02/03/1992
Bahamas	18/09/1973
Bahrain	21/09/1971
Bangladesh	17/09/1974
Barbados	09/12/1966
Belarus	24/10/1945
Belgium	27/12/1945
Belize	25/09/1981

Benin	20/09/1960
Bhutan	21/09/1971
Bolivia (Plurinational State of)	14/11/1945
Bosnia and Herzegovina	22/05/1992
Botswana	17/10/1966
Brazil	24/10/1945
Brunei Darussalam	21/09/1984
Bulgaria	14/12/1955
Burkina Faso	20/09/1960
Burundi	18/09/1962
Cabo Verde	16/09/1975
Cambodia	14/12/1955
Cameroon	20/09/1960
Canada	09/11/1945
Central African Republic	20/09/1960
Chad	20/09/1960
Chile	24/10/1945
China	24/10/1945
Colombia	05/11/1945
Comoros	12/11/1975
Congo	20/09/1960
Costa Rica	02/11/1945
Côte D'Ivoire	20/09/1960
Croatia	22/05/1992
Cuba	24/10/1945
Cyprus	20/09/1960
Czech Republic	19/01/1993

Democratic People's Republic of Korea	17/09/1991
Democratic Republic of the Congo	20/09/1960
Denmark	24/10/1945
Djibouti	20/09/1977
Dominica	18/12/1978
Dominican Republic	24/10/1945
Ecuador	21/12/1945
Egypt	24/10/1945
El Salvador	24/10/1945
Equatorial Guinea	12/11/1968
Eritrea	28/05/1993
Estonia	17/09/1991
Ethiopia	13/11/1945
Fiji	13/10/1970
Finland	14/12/1955
France	24/10/1945
Gabon	20/09/1960
Gambia	21/09/1965
Georgia	31/07/1992
Germany	18/09/1973
Ghana	08/03/1957
Greece	25/10/1945
Grenada	17/09/1974
Guatemala	21/11/1945
Guinea	12/12/1958
Guinea Bissau	17/09/1974
Guyana	20/09/1966

Haiti	24/10/1945
Honduras	17/12/1945
Hungary	14/12/1955
Iceland	19/11/1946
India	30/10/1945
Indonesia	28/09/1950
Iran (Islamic Republic of)	24/10/1945
Iraq	21/12/1945
Ireland	14/12/1955
Israel	11/05/1949
Italy	14/12/1955
Jamaica	18/09/1962
Japan	18/12/1956
Jordan	14/12/1955
Kazakhstan	02/03/1992
Kenya	16/12/1963
Kiribati	14/09/1999
Kuwait	14/05/1963
Kyrgyzstan	02/03/1992
Lao People's Democratic Republic	14/12/1955
Latvia	17/09/1991
Lebanon	24/10/1945
Lesotho	17/10/1966
Liberia	02/11/1945
Libya	14/12/1955
Liechtenstein	18/09/1990
Lithuania	17/09/1991

Luxembourg	24/10/1945
Madagascar	20/09/1960
Malawi	01/12/1964
Malaysia	17/09/1957
Maldives	21/09/1965
Mali	28/09/1960
Malta	01/12/1964
Marshall Islands	17/09/1991
Mauritania	27/10/1961
Mauritius	24/04/1968
Mexico	07/11/1945
Micronesia (Federated States of)	17/09/1991
Monaco	28/05/1993
Mongolia	27/10/1961
Montenegro	28/06/2006
Morocco	12/11/1956
Mozambique	16/09/1975
Myanmar	19/04/1948
Namibia	23/04/1990
Nauru	14/09/1999
Nepal	14/12/1955
Netherlands	10/12/1945
New Zealand	24/10/1945
Nicaragua	24/10/1945
Niger	20/09/1960
Nigeria	07/10/1960
Norway	27/11/1945

Oman	07/10/1971
Pakistan	30/09/1947
Palau	15/12/1994
Panama	13/11/1945
Papua New Guinea	10/10/1975
Paraguay	24/10/1945
Peru	31/10/1945
Philippines	24/10/1945
Poland	24/10/1945
Portugal	14/12/1955
Qatar	21/09/1971
Republic of Korea	17/09/1991
Republic of Moldova	02/03/1992
Romania	14/12/1955
Russian Federation	24/10/1945
Rwanda	18/09/1962
Saint Kitts and Nevis	23/09/1983
Saint Lucia	18/09/1979
Saint Vincent and the Grenadines	16/09/1980
Samoa	15/12/1976
San Marino	02/03/1992
Sao Tome and Principe	16/09/1975
Saudi Arabia	24/10/1945
Senegal	28/09/1960
Serbia	01/11/2000
Seychelles	21/09/1976
Sierra Leone	27/09/1961

Singapore	21/09/1965
Slovakia	19/01/1993
Slovenia	22/05/1992
Solomon Islands	19/09/1978
Somalia	20/09/1960
South Africa	07/11/1945
South Sudan	14/07/2011
Spain	14/12/1955
Sri Lanka	14/12/1955
Sudan	12/11/1956
Suriname	04/12/1975
Swaziland	24/09/1968
Sweden	19/11/1946
Switzerland	10/09/2002
Syrian Arab Republic	24/10/1945
Tajikistan	02/03/1992
Thailand	16/12/1946
The former Yugoslav Republic of Macedonia	08/04/1993
Timor-Leste	27/09/2002
Togo	20/09/1960
Tonga	14/09/1999
Trinidad and Tobago	18/09/1962
Tunisia	12/11/1956
Turkey	24/10/1945
Turkmenistan	02/03/1992
Tuvalu	05/09/2000
Uganda	25/10/1962

Ukraine	24/10/1945
United Arab Emirates	09/12/1971
United Kingdom of Great Britain and Northern Ireland	24/10/1945
United Republic of Tanzania	14/12/1961
United States of America	24/10/1945
Uruguay	18/12/1945
Uzbekistan	02/03/1992
Vanuatu	15/09/1981
Venezuela (Bolivarian Republic of)	15/11/1945
Viet Nam	20/09/1977
Yemen	30/09/1947
Zambia	01/12/1964
Zimbabwe	25/08/1980
Afghanistan	19/11/1946

Compare this data with that in the regions table. What updates are you going to make to the regions table in the light of the changes?

3. Here are the column names from the Earthquake table.

```

earthqk WITH +
countryn +
Earthqkn +
year      +
Yearn    +
Month    +
Day      +
Time     +
latitude +
longitud +
depth    +
magnitud +
unknown1 +
unknown2 +
deaths   +
strength +
refs     +
sregion

```

Do a select on the table using the following columns

latitude
longitud
magnitud
deaths

Order by magnitude and number of deaths.

Write the data to a file. We will use the data in the chapter on matplotlib, and plot a map of earthquakes.

19 Regular expressions and pattern matching

The Unix operating systems was one of the first computer systems to make pattern matching tools available using regular expressions. The following components of the Unix and Linux operating system support pattern matching

ed	the standard command driven editor
vi	the Unix editor. It is sometimes the only editor available on a Unix or Linux system.
grep	general regular expression parser
egrep	extended version of the above
fgrep	file based version of the above
sed	stream version of vi. Works really well on large files.

They enable the solution of quite a range of problems that would be difficult or even impossible in practice if they didn't exist.

I've used them several times in anger

the production of a typeset version of the college telephone directory whilst at Imperial College. The departments were asked to provide files which contained details of the people in their departments. These files were processed in two ways. The first involved the production of the departmental entries in the telephone directory and the second involved a sorted list of all people and personnel at Imperial. The process is going to be repetitive and so manually producing the telephone directory every 2 or 3 months would be very time consuming and error prone.

the production of the United Nations Environment Program Data reports. I had two secondments to UNEP to work on the production of the second and third editions of these reports. Some of the tables are complex and span over 20 pages. They involve large amounts of complex numeric data. Again the manual typesetting of these reports is tedious and error prone.

Database administrator for an EEC study into environmental kidney damage due to exposure to heavy metals and organic solvents. It involved the construction of a database of all of the test results (over 100 tests) from 8 laboratories across Europe involving Belgium (Antwerp and Louvain), Germany (Hannover and Berlin), Spain, Italy, Poland and the UK. Once the database had been constructed statistical analysis of the data started. Several hundred models were run, and some of the command files for these runs ran into several thousand lines of SAS, the statistical package we used to analyse the data.

I use the tools on a day to day basis, mainly the vi editor, which is one of the first things I install on any Windows based PC I work with. Learning how to use regular expressions and pattern matching will quickly repay the time and effort involved.

A theoretical coverage can be found in the Aho, Hopcroft and Ullman book. Details are given in the bibliography at the end of the chapter.

A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs.

I've taken the information below from standard Unix and Linux sources.

19.1 Metacharacters

Symbol	Action
.	any character
*	zero or more
^	start of line
\$	end of line
\	escape
[]	match one from a set
\()	named regular expression
\{ \}	range of instances
\< \>	match words beginning or end
+	one or more preceding
?	zero or more preceding
	separate choices to match
()	group expressions to match

Substitutions are regular expression language elements that are supported in replacement patterns.

19.2 Example 1 - UK post codes

The following information is taken from Wikipedia. The format of UK post codes is as follows, where A signifies a letter and 9 a digit:

Format	Example	Coverage
A9 9AA	M1 1AA	B, E, G, L, M, N, S, W postcode areas
A99 9AA	B33 8TH	
AA9 9AA	CR2 6XH	All postcode areas except B, E, G, L, M, N, S, W, WC
AA99 9AA	DN55 1PT	
A9A 9AA	W1A 1HQ	E1W, N1C, N1P, W1 postcode districts (high-density areas where more codes were needed)
AA9A 9AA	EC1A 1BB	WC postcode area; EC1-EC4, NW1W, SE1P, SW1 postcode districts (high-density areas where more codes were needed)

We will use the above information to write a program that will test for the validity of a UK post code using the regular expression capability of Python.

The first thing we need to do is construct the patterns we are interested in matching. The following lines do this.

```
" ([a-zA-Z][0-9][\\s][0-9][a-zA-Z][a-zA-Z]) "
```

```
" | ([a-zA-Z][0-9][0-9][\\s][0-9][a-zA-Z][a-zA-Z]) "
```

```
" | ([a-zA-Z][a-zA-Z][0-9][\\s][0-9][a-zA-Z][a-zA-Z]) "
```

```
" | ([a-zA-Z][a-zA-Z][0-9][0-9][\\s][0-9][a-zA-Z][a-zA-Z]) "
```

```
" | ([a-zA-Z][0-9][a-zA-Z][\\s][0-9][a-zA-Z][a-zA-Z]) "
```

```
" | ([a-zA-Z][a-zA-Z][0-9][a-zA-Z][\\s][0-9][a-zA-Z][a-zA-Z]) "
```

Each line matches one of the valid UK postcode patterns.

Here is the source code.

```
import re
```

```
def main():
    s1 = "([a-zA-Z][0-9][\\s][0-9][a-zA-Z][a-zA-Z])"
    s2 = "|([a-zA-Z][0-9][0-9][\\s][0-9][a-zA-Z][a-zA-Z])"
    s3 = "|([a-zA-Z][a-zA-Z][0-9][\\s][0-9][a-zA-Z][a-zA-Z])"
    s4 =
"|[a-zA-Z][a-zA-Z][0-9][0-9][\\s][0-9][a-zA-Z][a-zA-Z]"
    s5 = "|([a-zA-Z][0-9][a-zA-Z][\\s][0-9][a-zA-Z][a-zA-Z])"
    s6 =
"|[a-zA-Z][a-zA-Z][0-9][a-zA-Z][\\s][0-9][a-zA-Z][a-zA-Z]"

    postcode = s1+s2+s3+s4+s5+s6;

    p = re.compile(postcode)
    print(p)
    print(p.match(""))
    print(p.match('sw2 5jb'))
    print(p.match('np12 0pe'))
    print(p.match('sw2 5jb np12 0pe'))
    print(p.findall('sw2 5jb np12 0pe'))
    print(p.findall('sw2 5jb np12 0pe xx xx'))

if ( __name__ == "__main__" ):
    main()
```

Here is the output.

```
re.compile
('([a-zA-Z][0-9][\\s][0-9][a-zA-Z][a-zA-Z])|([a-zA-Z][0-9][0-9][\\s][0-9][a-zA-Z][a-zA-Z])|([a-zA-Z][a-zA-Z][0-9][\\s][0-9][a-zA-Z][a-zA-Z])|([a-zA-Z][a-zA-Z][0-9][0-9][\\s][0-9][a-zA-Z][a-zA-Z])|([a-z)
None
<_sre.SRE_Match object; span=(0, 7), match='sw2 5jb'>
<_sre.SRE_Match object; span=(0, 8), match='np12 0pe'>
<_sre.SRE_Match object; span=(0, 7), match='sw2 5jb'>
[('', '', 'sw2 5jb', '', '', ''), ('', '', '', 'np12 0pe',
'', '')]
[('', '', 'sw2 5jb', '', '', ''), ('', '', '', 'np12 0pe',
'', '')]
```

Test the program out with your home postcode.

19.3 Problems

1. Try the example out.

19.4 Bibliography

Aho A.V., Hopcroft J.E., Ullman J.D., The Design and Analysis of Computer Algorithms, Addison Wesley.

United Nations Environment Programme, Environmental Data Report, 1989-1990, Blackwell, ISBN 0-631-16987-3.

United Nations Environment Programme, Environmental Data Report, 1991-1992, Blackwell, ISBN 0-631-18083-4.

Don't interrupt me while I'm interrupting.

Winston Churchill.

20 Built in exceptions

The information in this chapter is taken from the documentation from the Python Standard Library.

In Python, all exceptions must be instances of a class that derives from `BaseException`. In a `try` statement with an `except` clause that mentions a particular class, that clause also handles any exception classes derived from that class (but not exception classes from which it is derived). Two exception classes that are not related via subclassing are never equivalent, even if they have the same name.

The built-in exceptions listed below can be generated by the interpreter or built-in functions. Except where mentioned, they have an “associated value” indicating the detailed cause of the error. This may be a string or a tuple of several items of information (e.g., an error code and a string explaining the code). The associated value is usually passed as arguments to the exception class's constructor.

User code can raise built-in exceptions. This can be used to test an exception handler or to report an error condition “just like” the situation in which the interpreter raises the same exception; but beware that there is nothing to prevent user code from raising an inappropriate error.

The built-in exception classes can be subclassed to define new exceptions; programmers are encouraged to derive new exceptions from the `Exception` class or one of its subclasses, and not from `BaseException`. More information on defining exceptions is available in the Python Tutorial under User-defined Exceptions.

When raising (or re-raising) an exception in an `except` or `finally` clause `__context__` is automatically set to the last exception caught; if the new exception is not handled the traceback that is eventually displayed will include the originating exception(s) and the final exception.

When raising a new exception (rather than using a bare `raise` to re-raise the exception currently being handled), the implicit exception context can be supplemented with an explicit cause by using `from with raise`:

```
raise new_exc from original_exc
```

The expression following `from` must be an exception or `None`. It will be set as `__cause__` on the raised exception. Setting `__cause__` also implicitly sets the `__suppress_context__` attribute to `True`, so that using `raise new_exc from None` effectively replaces the old exception with the new one for display purposes (e.g. converting `KeyError` to `AttributeError`, while leaving the old exception available in `__context__` for introspection when debugging).

The default traceback display code shows these chained exceptions in addition to the traceback for the exception itself. An explicitly chained exception in `__cause__` is always shown when present. An implicitly chained exception in `__context__` is shown only if `__cause__` is `None` and `__suppress_context__` is `false`.

In either case, the exception itself is always shown after any chained exceptions so that the final line of the traceback always shows the last exception that was raised.

20.1 Exception hierarchy

The class hierarchy for built-in exceptions is:

```

BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
    |   +-- FloatingPointError
    |   +-- OverflowError
    |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
    +-- LookupError
    |   +-- IndexError
    |   +-- KeyError
    +-- MemoryError
    +-- NameError
    |   +-- UnboundLocalError
    +-- OSError
    |   +-- BlockingIOError
    |   +-- ChildProcessError
    |   +-- ConnectionError
    |   |   +-- BrokenPipeError
    |   |   +-- ConnectionAbortedError
    |   |   +-- ConnectionRefusedError
    |   |   +-- ConnectionResetError
    |   +-- FileExistsError
    |   +-- FileNotFoundError
    |   +-- InterruptedError
    |   +-- IsADirectoryError
    |   +-- NotADirectoryError
    |   +-- PermissionError
    |   +-- ProcessLookupError
    |   +-- TimeoutError
    +-- ReferenceError
    +-- RuntimeError
    |   +-- NotImplementedError
    |   +-- RecursionError
    +-- SyntaxError
    |   +-- IndentationError
    |   +-- TabError
    +-- SystemError
    +-- TypeError
    +-- ValueError
    |   +-- UnicodeError
    |   +-- UnicodeDecodeError

```



```
|          +-- UnicodeEncodeError
|          +-- UnicodeTranslateError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning
    +-- ResourceWarning
```

20.2 Problems

There are no problems in this chapter.

21 Concurrent execution - threading

The material in this chapter is taken from the Standard Python Library reference material. The modules described in this chapter provide support for concurrent execution of code. The appropriate choice of tool will depend on the task to be executed (CPU bound vs IO bound) and preferred style of development (event driven cooperative multitasking vs pre-emptive multitasking).

21.1 Thread based parallelism - the threading package

In this section we have two short examples. The first is a serial solution to a problem, and the second is a multi threaded solution. The example is taken from

<http://www.wellho.net/solutions/python-python-threads-a-first-example.html>

In the example below I use ip addresses based on my set up at home. My network has a base address of 192.168.0, and I have machines in the range 0 to 30. Both examples ran on an openSuSe 13.1 system. This example will not run on a Windows system.

21.2 Example 1 - Serial solution

Here is the source

```
import os
import re
import time
import sys

def main():

    lifeline = re.compile(r"(\d) received")
    report = ("No response", "Partial Response", "Alive")

    print(time.ctime())

    for host in range(0,30):
        ip = "192.168.0."+str(host)
        pingaling = os.popen("ping -q -c2 "+ip,"r")
        print("Testing ",ip)
        sys.stdout.flush()
        while 1:
            line = pingaling.readline()
            if not line: break
            igot = re.findall(lifeline,line)
            if igot:
                print(report[int(igot[0])])

    print(time.ctime())

if ( __name__ == "__main__" ):
    main()
```

Here is the output

```
python3 c2101.py
Fri Feb 15 15:21:00 2019
Testing
192.168.0.0
Do you want to ping broadcast? Then -b. If not, check your
local firewall rules.
Testing 192.168.0.1
Alive
Testing 192.168.0.2
No response
Testing 192.168.0.3
No response
Testing 192.168.0.4
No response
Testing 192.168.0.5
No response
Testing 192.168.0.6
No response
Testing 192.168.0.7
No response
Testing 192.168.0.8
No response
Testing 192.168.0.9
No response
Testing 192.168.0.10
Alive
Testing 192.168.0.11
Alive
Testing 192.168.0.12
No response
Testing 192.168.0.13
No response
Testing 192.168.0.14
No response
Testing 192.168.0.15
Alive
Testing 192.168.0.16
No response
Testing 192.168.0.17
No response
Testing 192.168.0.18
No response
Testing 192.168.0.19
Alive
Testing 192.168.0.20
No response
Testing 192.168.0.21
No response
Testing 192.168.0.22
Alive
```

```

Testing 192.168.0.23
No response
Testing 192.168.0.24
No response
Testing 192.168.0.25
No response
Testing 192.168.0.26
No response
Testing 192.168.0.27
No response
Testing 192.168.0.28
No response
Testing 192.168.0.29
No response
Fri Feb 15 15:22:33 2019

```

We have a total time of one minute 32 seconds.

21.3 Example 2 - Multi-threaded solution

Here is the source.

```

import os
import re
import time
import sys
from threading import Thread

class testit(Thread):
    def __init__(self,ip):
        Thread.__init__(self)
        self.ip = ip
        self.status = -1
    def run(self):
        pingaling = os.popen("ping -q -c2 "+self.ip,"r")
        while 1:
            line = pingaling.readline()
            if not line: break
            igot = re.findall(testit.lifeline,line)
            if igot:
                self.status = int(igot[0])

testit.lifeline = re.compile(r"(\d) received")
report = ("No response","Partial Response","Alive")

print(time.ctime())

pinglist = []

for host in range(0,30):
    ip = "192.168.0."+str(host)
    current = testit(ip)

```

```

pinglist.append(current)
current.start()

for pingle in pinglist:
    pingle.join()
    print("Status from ",pingle.ip,"is",report[pingle.status])

print(time.ctime())

```

Here is the output from the same system as the serial solution.

```

python3 c2102.py
Fri Feb 15 15:24:15 2019
Do you want to ping broadcast? Then -b. If not, check your
local firewall rules.
Status from 192.168.0.0 is Alive
Status from 192.168.0.1 is Alive
Status from 192.168.0.2 is No response
Status from 192.168.0.3 is No response
Status from 192.168.0.4 is No response
Status from 192.168.0.5 is No response
Status from 192.168.0.6 is No response
Status from 192.168.0.7 is No response
Status from 192.168.0.8 is No response
Status from 192.168.0.9 is No response
Status from 192.168.0.10 is Alive
Status from 192.168.0.11 is Alive
Status from 192.168.0.12 is No response
Status from 192.168.0.13 is No response
Status from 192.168.0.14 is No response
Status from 192.168.0.15 is Alive
Status from 192.168.0.16 is No response
Status from 192.168.0.17 is No response
Status from 192.168.0.18 is No response
Status from 192.168.0.19 is Alive
Status from 192.168.0.20 is No response
Status from 192.168.0.21 is No response
Status from 192.168.0.22 is Alive
Status from 192.168.0.23 is No response
Status from 192.168.0.24 is No response
Status from 192.168.0.25 is No response
Status from 192.168.0.26 is No response
Status from 192.168.0.27 is No response
Status from 192.168.0.28 is No response
Status from 192.168.0.29 is No response
Fri Feb 15 15:24:27 2019

```

The elapsed time is now 12 seconds. Quite an improvement over the serial version.

21.4 Problems

1. Run the examples in this chapter. What timing figures did you get?

22 Concurrent execution - multi processing

22.1 Introduction

The material in this chapter is taken from the Standard Python Library reference material.

<https://docs.python.org/3/library/multiprocessing.html>

The modules described in this chapter provide support for concurrent execution of code. The appropriate choice of tool will depend on the task to be executed (CPU bound vs IO bound) and preferred style of development (event driven cooperative multitasking vs pre-emptive multitasking).

22.2 Process based parallelism - the multiprocessing package

multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.

The multiprocessing module also introduces APIs which do not have analogs in the threading module. A prime example of this is the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism). The following example demonstrates the common practice of defining such functions in a module so that child processes can successfully import that module. This basic example of data parallelism using Pool,

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    p = Pool(5)
    print(p.map(f, [1, 2, 3]))
```

will print to standard output

```
[1, 4, 9]
```

22.3 Contexts and start methods¶

Depending on the platform, multiprocessing supports three ways to start a process. These start methods are

spawn

The parent process starts a fresh python interpreter process. The child process will only inherit those resources necessary to run the process objects run() method. In particular, unnecessary file descriptors and handles from the parent process will not be inherited. Starting a process using this method is rather slow compared to using fork or forkserver.

Available on Unix and Windows. The default on Windows.

fork

The parent process uses `os.fork()` to fork the Python interpreter. The child process, when it begins, is effectively identical to the parent process. All resources of the parent are inherited by the child process. Note that safely forking a multithreaded process is problematic.

Available on Unix only. The default on Unix.

forkserver

When the program starts and selects the `forkserver` start method, a server process is started. From then on, whenever a new process is needed, the parent process connects to the server and requests that it fork a new process. The fork server process is single threaded so it is safe for it to use `os.fork()`. No unnecessary resources are inherited.

Available on Unix platforms which support passing file descriptors over Unix pipes.

Changed in version 3.4: `spawn` added on all unix platforms, and `forkserver` added for some unix platforms. Child processes no longer inherit all of the parents inheritable handles on Windows.

22.4 Example 1 - Simple multi-processing on a 6 core system

In this program we are going to look at breaking a computational problem down and partition it over several processes. We will also look at timing of both the parallel solution and the serial solution. Note that this example does not run correctly on the anaconda installation on Windows as of September 2018. You will need to run these examples on the cygwin version. Here is the source for a six core system - an AMD Phenom II X6. Example 2 is for an 8 core I7 system.

```
# This example is designed for a 6 core system

from multiprocessing import Pool

import time
import numpy as np
import math

scale_factor = 10000000
n             = 6 * scale_factor
nn           = 6

x = np.empty([n], dtype=np.float64)

start = np.array([0,
                  1 * scale_factor,
                  2 * scale_factor,
```



```

        3 * scale_factor,
        4 * scale_factor,
        5 * scale_factor])

end    = np.array([    scale_factor,
                    2 * scale_factor,
                    3 * scale_factor,
                    4 * scale_factor,
                    5 * scale_factor,
                    6 * scale_factor])

def partial_sum(i):
    return( sum( x[ start[i] : end[i] ] ) )

if __name__ == '__main__':
    print(" Program starts")
    print(" ",time.ctime(),end=" ")
    t1=time.time()
    print(t1)
    print(" n = ",n)

    parallel_sum = 0.0
    serial_sum    = 0.0
    psum = np.empty([nn],dtype=np.float64)
    for i in range(0,nn):
        psum[i]=0.0

    x = np.empty([n],dtype=np.float64)
    for i in range(0,n):
        x[i]=1.0

    print(" x array initialised",end=" ")
    t2=time.time()
    print(" {0:2.12f} ".format(t2-t1))
    t1=t2
    print(" *** Pool called ")

    pool = Pool(processes=nn)

    print(" Pool creation took", end=" ")
    t2=time.time()
    t3=t2-t1
    print(" {0:2.12f} ".format(t3))
    t1=t2

    result = pool.map( partial_sum , ( (i) for i in
range(nn)) )

    pool.close

```

```

pool.join

print(" *** Pool closed",end=" ")
t2=time.time()
t4=t2-t1
print(" {0:2.12f} ".format(t2-t1))
t1=t2

for i in range(nn):
    psum[i]=result[i]

parallel_sum = sum(psum)
print(" Parallel sum = ",parallel_sum)
t2=time.time()
t5=t2-t1+t3+t4
print(" Parallel time
{0:2.12f}".format(t5))
t1=t2

serial_sum=sum(x)
print(" Serial sum = ",serial_sum)
t2=time.time()

print(" Serial time
{0:2.12f}".format(t2-t1))
if ( math.fabs( parallel_sum - serial_sum ) > (1.0e-16) ):
    print(" ***** Error *****")
    print(" Parallel and serial sums do not match")
    print(" Are you using native Python on Windows?")
print(" Program ends")

```

22.5 Example 2 - Simple variant for an 8 core system

Here is the source code for an 8 core I7 system. Timing details for both versions are given later.

```

# This example is designed for an 8 core system
from multiprocessing import Pool
import time
import numpy as np
import math
scale_factor = 10000000
n          = 8 * scale_factor
nn         = 8
x = np.empty([n],dtype=np.float64)
start = np.array([0,
                  1 * scale_factor,
                  2 * scale_factor,
                  3 * scale_factor,
                  4 * scale_factor,
                  5 * scale_factor,

```

```

        6 * scale_factor,
        7 * scale_factor])
end      = np.array([
        scale_factor,
        2 * scale_factor,
        3 * scale_factor,
        4 * scale_factor,
        5 * scale_factor,
        6 * scale_factor,
        7 * scale_factor,
        8 * scale_factor])

def partial_sum(i):
    return( sum( x[ start[i] : end[i] ] ) ) )
if __name__ == '__main__':
    print(" Program starts")
    print(" ",time.ctime(),end=" ")
    t1=time.time()
    print(t1)
    print(" n = ",n)
    parallel_sum = 0.0
    serial_sum    = 0.0
    psum = np.empty([nn],dtype=np.float64)
    for i in range(0,nn):
        psum[i]=0.0
    x = np.empty([n],dtype=np.float64)
    for i in range(0,n):
        x[i]=1.0
    print(" x array initialised",end=" ")
    t2=time.time()
    print(" {0:2.12f} ".format(t2-t1))
    t1=t2
    print(" *** Pool called ")
    pool = Pool(processes=nn)
    print(" Pool creation took", end=" ")
    t2=time.time()
    t3=t2-t1
    print(" {0:2.12f} ".format(t3))
    t1=t2
    result = pool.map( partial_sum , ( (i) for i in
range(nn)) )
    pool.close
    pool.join
    print(" *** Pool closed",end=" ")
    t2=time.time()
    t4=t2-t1
    print(" {0:2.12f} ".format(t4))
    t1=t2
    for i in range(nn):
        psum[i]=result[i]
    parallel_sum = sum(psum)
    print(" Parallel sum = ",parallel_sum)

```

```

t2=time.time()
t5=t2-t1+t3+t4
print(" Parallel time
{0:2.12f}".format(t5))
t1=t2
serial_sum=sum(x)
print(" Serial sum    = ",serial_sum)
t2=time.time()
print(" Serial time
{0:2.12f}".format(t2-t1))
if ( math.fabs( parallel_sum - serial_sum ) > (1.0e-16) ):
    print(" ***** Error *****")
    print(" Parallel and serial sums do not match")
    print(" Are you using native Python on Windows?")
print(" Program ends")

```

22.6 Differences between the two version

Here is the diff output showing the differences between the two versions.

```

1c1
< # This example is designed for a 6 core system
---
> # This example is designed for an 8 core system
10,11c10,11
< n          = 6 * scale_factor
< nn         = 6
---
> n          = 8 * scale_factor
> nn         = 8
20c20,22
<
          5 * scale_factor])
---
>
          5 * scale_factor,
>
          6 * scale_factor,
>
          7 * scale_factor])
27c29,31
<
          6 * scale_factor])
---
>
          6 * scale_factor,
>
          7 * scale_factor,
>
          8 * scale_factor])

```

22.7 Sample runs

Here is a run on the AMD 6 core system for a native Windows of Python.

```

Program starts
  Fri May 10 14:37:46 2019 1557495466.1634073
n = 60000000
x array initialised          13.227007865906
*** Pool called
Pool creation took          0.275989532471

```

```
*** Pool closed 2.543994665146
Parallel sum = 0.0
Parallel time 2.821985721588
Serial sum = 60000000.0
Serial time 9.751012563705
***** Error *****
Parallel and serial sums do not match
Are you using native Python on Windows?
Program ends
```

As can be seen the parallel and serial sums differ.

Here is a sample run for the Cygwin Python implementation of Windows.

```
Program starts
Fri May 10 14:39:25 2019 1557495565.3735597
n = 60000000
x array initialised 11.209353685379
*** Pool called
Pool creation took 2.172768115997
*** Pool closed 2.029023408890
Parallel sum = 60000000.0
Parallel time 4.201887845993
Serial sum = 60000000.0
Serial time 9.101023674011
Program ends
```

The parallel and serial sums are the same.

22.8 Summary timing table

Here are the results of running the programs on three systems under both Windows and Linux.

AMD	Physical cores	6	
	Windows		openSuSe
	cygwin	python	
Initialisation	13.545349359512	12.626096487045	11.328769207001
Pool creation	2.256503820419	0.609584331512	0.015357732773
Pool closed	1.991185426712	2.359382629395	1.463187217712
Parallel time	1.991320371628	2.359382629395	1.463279008865
Serial	9.890810012817	8.423060894012	7.641946315765
Pool + parallel	6.239009618759	5.328349590302	2.941823959350
Total time	29.675168991088	26.377506971359	21.912539482116
Intel	Physical cores	4	
	Hyperthreading	* 2	
	Windows		openSuSe
	cygwin	anaconda	
Initialisation	13.213007688522	15.205041885376	18.766371488571
Pool creation	2.221862316132	0.296875238419	0.116709947586
Pool closed	3.448457002640	3.296873569489	2.308131217957
Parallel time	3.448578357697	3.296873569489	0.000143527985
Serial	14.807518482208	8.238389492035	9.565270185471
Pool + parallel	9.118897676469	6.890622377397	2.424984693528
Total time	37.139423847199	30.334053754808	30.756626367570
Intel	Physical cores	4	
	Hyperthreading	* 2	
	Windows		openSuSe
	cygwin	anaconda	
Initialisation		14.877320051193	15.358544111252
Pool creation		0.484356403350	0.160874843597
Pool closed		4.468796491623	2.801591157913
Parallel time		4.468796491623	2.801677703857
Serial		9.499331951141	9.257079601288
Pool + parallel		9.421949386596	5.764143705367
Total time		33.798601388930	30.379767417907

22.9 Problems

1. Run the example(s) in this chapter. What sums and timing figures did you get?

23 Modules

23.1 Introduction

The following is a complete list of the Python modules taken from the 3.5.1 documentation.

—	
<code>__future__</code>	Future statement definitions
<code>__main__</code>	The environment where the top-level script is run.
<code>_dummy_thread</code>	Drop-in replacement for the <code>_thread</code> module.
<code>_thread</code>	Low-level threading API.
a	
<code>abc</code>	Abstract base classes according to PEP 3119.
<code>aifc</code>	Read and write audio files in AIFF or AIFC format.
<code>argparse</code>	Command-line option and argument parsing library.
<code>array</code>	Space efficient arrays of uniformly typed numeric values.
<code>ast</code>	Abstract Syntax Tree classes and manipulation.
<code>asynchat</code>	Support for asynchronous command/response protocols.
<code>asyncio</code>	Asynchronous I/O, event loop, coroutines and tasks.
<code>asyncore</code>	A base class for developing asynchronous socket handling services.
<code>atexit</code>	Register and execute cleanup functions.
<code>audioop</code>	Manipulate raw audio data.
b	
<code>base64</code>	RFC 3548: Base16, Base32, Base64 Data Encodings; Base85 and Ascii85
<code>bdb</code>	Debugger framework.
<code>binascii</code>	Tools for converting between binary and various ASCII-encoded binary representations.
<code>binhex</code>	Encode and decode files in binhex4 format.
<code>bisect</code>	Array bisection algorithms for binary searching.
<code>builtins</code>	The module that provides the built-in namespace.
<code>bz2</code>	Interfaces for bzip2 compression and decompression.
c	
<code>calendar</code>	Functions for working with calendars, including some emulation of the Unix <code>cal</code> program.
<code>cgi</code>	Helpers for running Python scripts via the Common Gateway Interface.
<code>cgitb</code>	Configurable traceback handler for CGI scripts.
<code>chunk</code>	Module to read IFF chunks.
<code>cmath</code>	Mathematical functions for complex numbers.
<code>cmd</code>	Build line-oriented command interpreters.
<code>code</code>	Facilities to implement read-eval-print loops.

codecs	Encode and decode data and streams.
codeop	Compile (possibly incomplete) Python code.
collections	Container datatypes
colorsys	Conversion functions between RGB and other color systems.
compileall	Tools for byte-compiling all Python source files in a directory tree.
concurrent	
configparser	Configuration file parser.
contextlib	Utilities for with-statement contexts.
copy	Shallow and deep copy operations.
copyreg	Register pickle support functions.
cProfile	
crypt (Unix)	The crypt() function used to check Unix passwords.
csv	Write and read tabular data to and from delimited files.
ctypes	A foreign function library for Python.
curses (Unix)	An interface to the curses library, providing portable terminal handling.
d	
datetime	Basic date and time types.
dbm	Interfaces to various Unix "database" formats.
decimal	Implementation of the General Decimal Arithmetic Specification.
difflib	Helpers for computing differences between objects.
dis	Disassembler for Python bytecode.
distutils	Support for building and installing Python modules into an existing Python installation.
doctest	Test pieces of code within docstrings.
dummy_threading	Drop-in replacement for the threading module.
e	
email	Package supporting the parsing, manipulating, and generating email messages, including MIME documents.
encodings	
ensurepip	Bootstrapping the "pip" installer into an existing Python installation or virtual environment.
enum	Implementation of an enumeration class.
errno	Standard errno system symbols.
f	
faulthandler	Dump the Python traceback.
fcntl (Unix)	The fcntl() and ioctl() system calls.
filecmp	Compare files efficiently.
fileinput	Loop over standard input or a list of files.
fnmatch	Unix shell style filename pattern matching.
formatter	Deprecated: Generic output formatter and device interface.

fpectl (Unix)	Provide control for floating point exception handling.
fractions	Rational numbers.
ftplib	FTP protocol client (requires sockets).
functools	Higher-order functions and operations on callable objects.
g	
gc	Interface to the cycle-detecting garbage collector.
getopt	Portable parser for command line options; support both short and long option names.
getpass	Portable reading of passwords and retrieval of the userid.
gettext	Multilingual internationalization services.
glob	Unix shell style pathname pattern expansion.
grp (Unix)	The group database (getgrnam() and friends).
gzip	Interfaces for gzip compression and decompression using file objects.
h	
hashlib	Secure hash and message digest algorithms.
heapq	Heap queue algorithm (a.k.a. priority queue).
hmac	Keyed-Hashing for Message Authentication (HMAC) implementation
html	Helpers for manipulating HTML.
http	HTTP status codes and messages
i	
imaplib	IMAP4 protocol client (requires sockets).
imghdr	Determine the type of image contained in a file or byte stream.
imp	Deprecated: Access the implementation of the import statement.
importlib	The implementation of the import machinery.
inspect	Extract information and source code from live objects.
io	Core tools for working with streams.
ipaddress	IPv4/IPv6 manipulation library.
itertools	Functions creating iterators for efficient looping.
j	
json	Encode and decode the JSON format.
k	
keyword	Test whether a string is a keyword in Python.
l	
lib2to3	the 2to3 library
linecache	This module provides random access to individual lines from text files.
locale	Internationalization services.
logging	Flexible event logging system for applications.
lzma	A Python wrapper for the liblzma compression library.
m	

macpath	Mac OS 9 path manipulation functions.
mailbox	Manipulate mailboxes in various formats
mailcap	Mailcap file handling.
marshal	Convert Python objects to streams of bytes and back (with different constraints).
math	Mathematical functions (sin() etc.).
mimetypes	Mapping of filename extensions to MIME types.
mmap	Interface to memory-mapped files for Unix and Windows.
modulefinder	Find modules used by a script.
msilib (Windows)	Creation of Microsoft Installer files, and CAB files.
msvcrt (Windows)	Miscellaneous useful routines from the MS VC++ runtime.
multiprocessing	Process-based parallelism.
n	
netrc	Loading of .netrc files.
nis (Unix)	Interface to Sun's NIS (Yellow Pages) library.
nntplib	NNTP protocol client (requires sockets).
numbers	Numeric abstract base classes (Complex, Real, Integral, etc.).
o	
operator	Functions corresponding to the standard operators.
optparse	Deprecated: Command-line option parsing library.
os	Miscellaneous operating system interfaces.
ossaudiodev (Linux, FreeBSD)	Access to OSS-compatible audio devices.
p	
parser	Access parse trees for Python source code.
pathlib	Object-oriented filesystem paths
pdb	The Python debugger for interactive interpreters.
pickle	Convert Python objects to streams of bytes and back.
pickletools	Contains extensive comments about the pickle protocols and pickle-machine opcodes, as well as some useful functions.
pipes (Unix)	A Python interface to Unix shell pipelines.
pkgutil	Utilities for the import system.
platform	Retrieves as much platform identifying data as possible.
plistlib	Generate and parse Mac OS X plist files.
poplib	POP3 protocol client (requires sockets).
posix (Unix)	The most common POSIX system calls (normally used via module os).
pprint	Data pretty printer.
profile	Python source profiler.
pstats	Statistics object for use with the profiler.
pty (Linux)	Pseudo-Terminal Handling for Linux.

pwd (Unix)	The password database (getpwnam() and friends).
py_compile	Generate byte-code files from Python source files.
pyclbr	Supports information extraction for a Python class browser.
pydoc	Documentation generator and online help system.
q	
queue	A synchronized queue class.
quopri	Encode and decode files using the MIME quoted-printable encoding.
r	
random	Generate pseudo-random numbers with various common distributions.
re	Regular expression operations.
readline (Unix)	GNU readline support for Python.
reprlib	Alternate repr() implementation with size limits.
resource (Unix)	An interface to provide resource usage information on the current process.
rlcompleter	Python identifier completion, suitable for the GNU readline library.
runpy	Locate and run Python modules without importing them first.
s	
sched	General purpose event scheduler.
select	Wait for I/O completion on multiple streams.
selectors	High-level I/O multiplexing.
shelve	Python object persistence.
shlex	Simple lexical analysis for Unix shell-like languages.
shutil	High-level file operations, including copying.
signal	Set handlers for asynchronous events.
site	Module responsible for site-specific configuration.
smtpd	A SMTP server implementation in Python.
smtplib	SMTP protocol client (requires sockets).
sndhdr	Determine type of a sound file.
socket	Low-level networking interface.
socketserver	A framework for network servers.
spwd (Unix)	The shadow password database (getspnam() and friends).
sqlite3	A DB-API 2.0 implementation using SQLite 3.x.
ssl	TLS/SSL wrapper for socket objects
stat	Utilities for interpreting the results of os.stat(), os.lstat() and os.fstat().
statistics	mathematical statistics functions
string	Common string operations.
stringprep	String preparation, as per RFC 3453
struct	Interpret bytes as packed binary data.
subprocess	Subprocess management.

sunau	Provide an interface to the Sun AU sound format.
symbol	Constants representing internal nodes of the parse tree.
symtable	Interface to the compiler's internal symbol tables.
sys	Access system-specific parameters and functions.
sysconfig	Python's configuration information
syslog (Unix)	An interface to the Unix syslog library routines.
t	
tabnanny	Tool for detecting white space related problems in Python source files in a directory tree.
tarfile	Read and write tar-format archive files.
telnetlib	Telnet client class.
tempfile	Generate temporary files and directories.
termios (Unix)	POSIX style tty control.
test	Regression tests package containing the testing suite for Python.
textwrap	Text wrapping and filling
threading	Thread-based parallelism.
time	Time access and conversions.
timeit	Measure the execution time of small code snippets.
tkinter	Interface to Tcl/Tk for graphical user interfaces
token	Constants representing terminal nodes of the parse tree.
tokenize	Lexical scanner for Python source code.
trace	Trace or track Python statement execution.
traceback	Print or retrieve a stack traceback.
tracemalloc	Trace memory allocations.
tty (Unix)	Utility functions that perform common terminal control operations.
turtle	An educational framework for simple graphics applications
turtledemo	A viewer for example turtle scripts
types	Names for built-in types.
typing	Support for type hints (see PEP 484).
u	
unicodedata	Access the Unicode Database.
unittest	Unit testing framework for Python.
urllib	
uu	Encode and decode files in uuencode format.
uuid	UUID objects (universally unique identifiers) according to RFC 4122
v	
venv	Creation of virtual environments.
w	
warnings	Issue warning messages and control their disposition.
wave	Provide an interface to the WAV sound format.

weakref	Support for weak references and weak dictionaries.
webbrowser	Easy-to-use controller for Web browsers.
wireg (Windows)	Routines and objects for manipulating the Windows registry.
winsound (Windows)	Access to the sound-playing machinery for Windows.
wsgiref	WSGI Utilities and Reference Implementation.
x	
xdrlib	Encoders and decoders for the External Data Representation (XDR).
xml	Package containing XML processing modules
xmlrpc	
z	
zipapp	Manage executable python zip archives
zipfile	Read and write ZIP-format archive files.
zipimport	support for importing Python modules from ZIP archives.
zlib	Low-level interface to compression and decompression routines compatible with gzip.

The following information is taken from the Python tutorial.

23.2 Introduction to modules

If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

A module is a file containing Python definitions and statements. The file name is the module name with the suffix `.py` appended. Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.

23.3 Example 1 - simple module usage

For instance, use your favourite text editor to create a file called `fibonacci.py` in the current directory with the following contents:

```
# Fibonacci numbers module
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()
def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
```

```

while b < n:
    result.append(b)
    a, b = b, a+b
return result

```

Now enter the Python interpreter and import this module with the following command:

```
>>> import fibo
```

This does not enter the names of the functions defined in `fibo` directly in the current symbol table; it only enters the module name `fibo` there. Using the module name you can access the functions:

```

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]python3

>>> fibo.__name__
'fibo'

```

If you intend to use a function often you can assign it to a local name:

```

>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377

```

23.4 More on Modules

A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the first time the module name is encountered in an import statement. [1] (They are also run if the file is executed as a script.)

Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module. Thus, the author of a module can use global variables in the module without worrying about accidental clashes with a user's global variables. On the other hand, if you know what you are doing you can touch a module's global variables with the same notation used to refer to its functions, `modname.itemname`.

Modules can import other modules. It is customary but not required to place all import statements at the beginning of a module (or script, for that matter). The imported module names are placed in the importing module's global symbol table.

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table. For example:

```

>>> from fibo import fib, fib2
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377

```

This does not introduce the module name from which the imports are taken in the local symbol table (so in the example, `fibo` is not defined).

There is even a variant to import all names that a module defines:

```

>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377

```

This imports all names except those beginning with an underscore (`_`). In most cases Python programmers do not use this facility since it introduces an unknown set of names into the interpreter, possibly hiding some things you have already defined.

Note that in general the practice of importing `*` from a module or package is frowned upon, since it often causes poorly readable code. However, it is okay to use it to save typing in interactive sessions.

23.4.1 Note:

For efficiency reasons, each module is only imported once per interpreter session. Therefore, if you change your modules, you must restart the interpreter – or, if it's just one module you want to test interactively, use `imp.reload()`, e.g. `import imp; imp.reload(modulename)`.

23.5 Executing modules as scripts

When you run a Python module with

```
python fibo.py <arguments>
```

the code in the module will be executed, just as if you imported it, but with the `__name__` set to `"__main__"`. That means that by adding this code at the end of your module:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

you can make the file usable as a script as well as an importable module, because the code that parses the command line only runs if the module is executed as the “main” file:

```
$ python fibo.py 50
1 1 2 3 5 8 13 21 34
```

If the module is imported, the code is not run:

```
>>> import fibo
>>>
```

This is often used either to provide a convenient user interface to a module, or for testing purposes (running the module as a script executes a test suite).

23.6 The Module Search Path

When a module named `spam` is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named `spam.py` in a list of directories given by the variable `sys.path`. `sys.path` is initialized from these locations:

- The directory containing the input script (or the current directory when no file is specified).
- `PYTHONPATH` (a list of directory names, with the same syntax as the shell variable `PATH`).
- The installation-dependent default.

23.6.1 Note:

On file systems which support symlinks, the directory containing the input script is calculated after the symlink is followed. In other words the directory containing the symlink is not added to the module search path.

After initialization, Python programs can modify `sys.path`. The directory containing the script being run is placed at the beginning of the search path, ahead of the standard library path. This means that scripts in that directory will be loaded instead of modules of the same

name in the library directory. This is an error unless the replacement is intended. See section Standard Modules for more information.

23.7 “Compiled” Python files

To speed up loading modules, Python caches the compiled version of each module in the `__pycache__` directory under the name `module.version.pyc`, where the version encodes the format of the compiled file; it generally contains the Python version number. For example, in CPython release 3.3 the compiled version of `spam.py` would be cached as `__pycache__/spam.cpython-33.pyc`. This naming convention allows compiled modules from different releases and different versions of Python to coexist.

Python checks the modification date of the source against the compiled version to see if it's out of date and needs to be recompiled. This is a completely automatic process. Also, the compiled modules are platform-independent, so the same library can be shared among systems with different architectures.

Python does not check the cache in two circumstances. First, it always recompiles and does not store the result for the module that's loaded directly from the command line. Second, it does not check the cache if there is no source module. To support a non-source (compiled only) distribution, the compiled module must be in the source directory, and there must not be a source module.

Some tips for experts:

- You can use the `-O` or `-OO` switches on the Python command to reduce the size of a compiled module. The `-O` switch removes assert statements, the `-OO` switch removes both assert statements and `__doc__` strings. Since some programs may rely on having these available, you should only use this option if you know what you're doing. “Optimized” modules have an `opt-` tag and are usually smaller. Future releases may change the effects of optimization.
- A program doesn't run any faster when it is read from a `.pyc` file than when it is read from a `.py` file; the only thing that's faster about `.pyc` files is the speed with which they are loaded.
- The module `compileall` can create `.pyc` files for all modules in a directory.
- There is more detail on this process, including a flow chart of the decisions, in PEP 3147.

23.8 Standard Modules

Python comes with a library of standard modules, described in a separate document, the Python Library Reference (“Library Reference” hereafter). Some modules are built into the interpreter; these provide access to operations that are not part of the core of the language but are nevertheless built in, either for efficiency or to provide access to operating system primitives such as system calls. The set of such modules is a configuration option which also depends on the underlying platform. For example, the `winreg` module is only provided on Windows systems. One particular module deserves some attention: `sys`, which is built into every Python interpreter. The variables `sys.ps1` and `sys.ps2` define the strings used as primary and secondary prompts:

```
>>> import sys
>>> sys.ps1
'>>> '
>>> sys.ps2
```

```
'... '
>>> sys.ps1 = 'C> '
C> print('Yuck!')
Yuck!
C>
```

These two variables are only defined if the interpreter is in interactive mode.

The variable `sys.path` is a list of strings that determines the interpreter's search path for modules. It is initialized to a default path taken from the environment variable `PYTHONPATH`, or from a built-in default if `PYTHONPATH` is not set. You can modify it using standard list operations:

```
>>> import sys
>>> sys.path.append('/ufs/guido/lib/python')
```

23.9 The `dir()` Function

The built-in function `dir()` is used to find out which names a module defines. It returns a sorted list of strings:

```
>>> import fibo, sys
>>> dir(fibo)
['_name__', 'fib', 'fib2']
>>> dir(sys)
['_displayhook__', '__doc__', '__excepthook__', '__loader__',
'_name__',
'__package__', '__stderr__', '__stdin__', '__stdout__',
'_clear_type_cache', '_current_frames', '_debugmallocstats',
'_getframe',
'_home', '_mercurial', '_xoptions', 'abiflags', 'api_ver-
sion', 'argv',
'base_exec_prefix', 'base_prefix', 'builtin_module_names',
'byteorder',
'call_tracing', 'callstats', 'copyright', 'displayhook',
'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_pre-
fix',
'executable', 'exit', 'flags', 'float_info',
'float_repr_style',
'getcheckinterval', 'getdefaultencoding', 'getdlopenflags',
'getfilesystemencoding', 'getobjects', 'getprofile',
'getrecursionlimit',
'getrefcount', 'getsizeof', 'getswitchinterval',
'gettotalrefcount',
'gettrace', 'hash_info', 'hexversion', 'implementation',
'int_info',
'intern', 'maxsize', 'maxunicode', 'meta_path', 'modules',
'path',
'path_hooks', 'path_importer_cache', 'platform', 'prefix',
'ps1',
'setcheckinterval', 'setdlopenflags', 'setprofile',
'setrecursionlimit',
'setswitchinterval', 'settrace', 'stderr', 'stdin',
'stdout',
```

```
'thread_info', 'version', 'version_info', 'warnoptions']
```

Without arguments, `dir()` lists the names you have defined currently:

```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo
>>> fib = fibo.fib
>>> dir()
['_builtins__', '__name__', 'a', 'fib', 'fibo', 'sys']
```

Note that it lists all types of names: variables, modules, functions, etc.

`dir()` does not list the names of built-in functions and variables. If you want a list of those, they are defined in the standard module `builtins`:

```
>>> import builtins
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError',
'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError',
'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError',
'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError',
'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception',
'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError',
'InterruptedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryError',
'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError',
'ProcessLookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError',
'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning',
'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True',
'TypeError',
'UnboundLocalError', 'UnicodeDecodeError',
'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning',
'UserWarning',
'ValueError', 'Warning', 'ZeroDivisionError', '_',
'__build_class__',
'__debug__', '__doc__', '__import__', '__name__', '__pack-
age__', 'abs',
```

```

'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes',
'callable',
'chr', 'classmethod', 'compile', 'complex', 'copyright',
'credits',
'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval',
'exec', 'exit',
'filter', 'float', 'format', 'frozenset', 'getattr',
'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
'issubclass',
'iter', 'len', 'license', 'list', 'locals', 'map', 'max',
'memoryview',
'min', 'next', 'object', 'oct', 'open', 'ord', 'pow',
'print', 'property',
'quit', 'range', 'repr', 'reversed', 'round', 'set',
'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple',
'type', 'vars',
'zip']

```

23.10 Packages

Packages are a way of structuring Python’s module namespace by using “dotted module names”. For example, the module name A.B designates a submodule named B in a package named A. Just like the use of modules saves the authors of different modules from having to worry about each other’s global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or the Python Imaging Library from having to worry about each other’s module names.

Suppose you want to design a collection of modules (a “package”) for the uniform handling of sound files and sound data. There are many different sound file formats (usually recognized by their extension, for example: .wav, .aiff, .au), so you may need to create and maintain a growing collection of modules for the conversion between the various file formats. There are also many different operations you might want to perform on sound data (such as mixing, adding echo, applying an equalizer function, creating an artificial stereo effect), so in addition you will be writing a never-ending stream of modules to perform these operations. Here’s a possible structure for your package (expressed in terms of a hierarchical filesystem):

```

sound/                                Top-level package
    __init__.py                        Initialize the sound
package
    formats/                           Subpackage for file for-
mat conversions
    __init__.py
    wavread.py
    wavwrite.py
    aiffread.py
    aiffwrite.py
    auread.py
    auwrite.py
    ...

```

```

effects/                               Subpackage for sound ef-
fects
    __init__.py
    echo.py
    surround.py
    reverse.py
    ...
filters/                               Subpackage for filters
    __init__.py
    equalizer.py
    vocoder.py
    karaoke.py
    ...

```

When importing the package, Python searches through the directories on `sys.path` looking for the package subdirectory.

The `__init__.py` files are required to make Python treat the directories as containing packages; this is done to prevent directories with a common name, such as `string`, from unintentionally hiding valid modules that occur later on the module search path. In the simplest case, `__init__.py` can just be an empty file, but it can also execute initialization code for the package or set the `__all__` variable, described later.

Users of the package can import individual modules from the package, for example:

```
import sound.effects.echo
```

This loads the submodule `sound.effects.echo`. It must be referenced with its full name.

```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

An alternative way of importing the submodule is:

```
from sound.effects import echo
```

This also loads the submodule `echo`, and makes it available without its package prefix, so it can be used as follows:

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

Yet another variation is to import the desired function or variable directly:

```
from sound.effects.echo import echofilter
```

Again, this loads the submodule `echo`, but this makes its function `echofilter()` directly available:

```
echofilter(input, output, delay=0.7, atten=4)
```

Note that when using `from package import item`, the item can be either a submodule (or subpackage) of the package, or some other name defined in the package, like a function, class or variable. The import statement first tests whether the item is defined in the package; if not, it assumes it is a module and attempts to load it. If it fails to find it, an `ImportError` exception is raised.

Contrarily, when using syntax like `import item.subitem.subsubitem`, each item except for the last must be a package; the last item can be a module or a package but can't be a class or function or variable defined in the previous item.

23.11 Importing * From a Package

Now what happens when the user writes `from sound.effects import *`? Ideally, one would hope that this somehow goes out to the filesystem, finds which submodules are present in

the package, and imports them all. This could take a long time and importing sub-modules might have unwanted side-effects that should only happen when the sub-module is explicitly imported.

The only solution is for the package author to provide an explicit index of the package. The import statement uses the following convention: if a package's `__init__.py` code defines a list named `__all__`, it is taken to be the list of module names that should be imported when from `package import *` is encountered. It is up to the package author to keep this list up-to-date when a new version of the package is released. Package authors may also decide not to support it, if they don't see a use for importing `*` from their package. For example, the file `sound/effects/__init__.py` could contain the following code:

```
__all__ = ["echo", "surround", "reverse"]
```

This would mean that `from sound.effects import *` would import the three named submodules of the sound package.

If `__all__` is not defined, the statement `from sound.effects import *` does not import all submodules from the package `sound.effects` into the current namespace; it only ensures that the package `sound.effects` has been imported (possibly running any initialization code in `__init__.py`) and then imports whatever names are defined in the package. This includes any names defined (and submodules explicitly loaded) by `__init__.py`. It also includes any submodules of the package that were explicitly loaded by previous import statements.

Consider this code:

```
import sound.effects.echo
import sound.effects.surround
from sound.effects import *
```

In this example, the `echo` and `surround` modules are imported in the current namespace because they are defined in the `sound.effects` package when the `from...import` statement is executed. (This also works when `__all__` is defined.)

Although certain modules are designed to export only names that follow certain patterns when you use `import *`, it is still considered bad practise in production code.

Remember, there is nothing wrong with using `from Package import specific_submodule!` In fact, this is the recommended notation unless the importing module needs to use submodules with the same name from different packages.

23.12 Intra-package References

When packages are structured into subpackages (as with the sound package in the example), you can use absolute imports to refer to submodules of siblings packages. For example, if the module `sound.filters.vocoder` needs to use the `echo` module in the `sound.effects` package, it can use `from sound.effects import echo`.

You can also write relative imports, with the `from module import name` form of import statement. These imports use leading dots to indicate the current and parent packages involved in the relative import. From the `surround` module for example, you might use:

```
from . import echo
from .. import formats
from ..filters import equalizer
```

Note that relative imports are based on the name of the current module. Since the name of the main module is always `"__main__"`, modules intended for use as the main module of a Python application must always use absolute imports.

23.13 Packages in Multiple Directories

Packages support one more special attribute, `__path__`. This is initialized to be a list containing the name of the directory holding the package's `__init__.py` before the code in that file is executed. This variable can be modified; doing so affects future searches for modules and subpackages contained in the package.

While this feature is not often needed, it can be used to extend the set of modules found in a package.

Footnotes

[1] In fact function definitions are also ‘statements’ that are ‘executed’; the execution of a module-level function definition enters the function name in the module’s global symbol table.

23.14 Summary

This chapter briefly introduces some of the concepts involved in using modules in Python.

23.15 Problems

1. Compile and run the examples in this chapter.

24 SciPy and Pandas

24.1 Introduction

The SciPy site is

<http://www.scipy.org/>

and the following information is taken from that site.

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

NumPy	Base N-dimensional array package
SciPy library	Fundamental library for scientific computing
Matplotlib	Comprehensive 2D Plotting
IPython	Enhanced Interactive Console
Sympy	Symbolic mathematics
pandas	Data structures & analysis

We covered Numpy in an earlier chapter and cover matplotlib in a later chapter. In this chapter we will have a look at a small number of Pandas examples, processing the Met Office station data.

Here is some additional information taken from the scipy site.

24.2 Documentation

Documentation is also available.

<http://docs.scipy.org/doc/scipy/reference/>

24.3 Tutorials

A tutorial is available covering the following subjects.

- Introduction
- Basic functions
- Special functions (`scipy.special`)
- Integration (`scipy.integrate`)
- Optimization (`scipy.optimize`)
- Interpolation (`scipy.interpolate`)
- Fourier Transforms (`scipy.fftpack`)
- Signal Processing (`scipy.signal`)
- Linear Algebra (`scipy.linalg`)
- Sparse Eigenvalue Problems with ARPACK
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial data structures and algorithms (`scipy.spatial`)
- Statistics (`scipy.stats`)
- Multidimensional image processing (`scipy.ndimage`)
- File IO (`scipy.io`)
- Weave (`scipy.weave`)

24.4 Reference material

Reference material is also available. Here are some of the areas.

- Clustering package (`scipy.cluster`)
- Constants (`scipy.constants`)
- Discrete Fourier transforms (`scipy.fftpack`)
- Integration and ODEs (`scipy.integrate`)
- Interpolation (`scipy.interpolate`)
- Input and output (`scipy.io`)
- Linear algebra (`scipy.linalg`)
- Miscellaneous routines (`scipy.misc`)
- Multi-dimensional image processing (`scipy.ndimage`)
- Orthogonal distance regression (`scipy.odr`)
- Optimization and root finding (`scipy.optimize`)
- Signal processing (`scipy.signal`)
- Sparse matrices (`scipy.sparse`)
- Sparse linear algebra (`scipy.sparse.linalg`)
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial algorithms and data structures (`scipy.spatial`)
- Special functions (`scipy.special`)
- Statistical functions (`scipy.stats`)
- Statistical functions for masked arrays (`scipy.stats.mstats`)
- C/C++ integration (`scipy.weave`)

24.5 Pandas

In this section we look at Pandas. Here is the Wikipedia entry.

[https://en.wikipedia.org/wiki/Pandas_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))

Here are some extracts from that site.

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

Library features

- DataFrame object for data manipulation with integrated indexing.

- Tools for reading and writing data between in-memory data structures and different file formats.

- Data alignment and integrated handling of missing data.

- Reshaping and pivoting of data sets.

Label-based slicing, fancy indexing, and subsetting of large data sets.

Data structure column insertion and deletion.

Group by engine allowing split-apply-combine operations on data sets.

Data set merging and joining.

Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.

Time series-functionality: Date range generation[4] and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

Provides data filtration.

The library is highly optimized for performance, with critical code paths written in Cython or C.

Here is the main Pandas site.

<https://pandas.pydata.org>

Here are some extracts from that site.

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NumFOCUS sponsored project. This will help ensure the success of development of pandas as a world-class open-source project, and makes it possible to donate to the project.

What problem does pandas solve?

Python has long been great for data munging and preparation, but less so for data analysis and modeling. pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

Combined with the excellent IPython toolkit and other libraries, the environment for doing data analysis in Python excels in performance, productivity, and the ability to collaborate.

pandas does not implement significant modeling functionality outside of linear and panel regression; for this, look to statsmodels and scikit-learn. More work is still needed to make Python a first class statistical modeling environment, but we are well on our way toward that goal

In this chapter we will look at processing the Met Office data using Pandas. We will be using the Cwmystwyth data throughout the examples.

24.5.1 Example 1 - Basic Pandas syntax

The first example just looks at creating a Pandas DataFrame from the Cwmystwyth data, and showing some simple examples of Pandas functionality. Here is the source.

```
import numpy as np
import pandas as pd
import math
data_file_name="cwmystwythdata.txt"
month_names = ["January","Febru-
ary","March","April","May","June","July","August","Septem-
ber","October","November","December"]
matrix = np.genfromtxt( data_file_name, \
                        skip_header=7 , \
                        skip_footer=1 , \
                        usecols=(0,1,2,3,4,5,6), \
                        autostrip=True , \

dtype=(int,int,float,float,int,float,float), \
      missing_values={"---"}\
)

n=matrix.size
print(n)
print(type(matrix))
dataframe_1 = pd.DataFrame(matrix)
print(dataframe_1.columns)
print(dataframe_1.index)
print(dataframe_1['f1'])
print(dataframe_1.ix[[0,1]])
print(dataframe_1.sort_index(axis=1))
print(dataframe_1.sort_index(by='f5'))
```

We create the data frame using the matrix created by `genfromtxt`, which we used in an earlier example.

Here is some sample output. We have deleted repetitive sections of the output.

```
618
<class 'numpy.ndarray'>
Index(['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6'], dtype='ob-
ject')
RangeIndex(start=0, stop=618, step=1)
0          1
1          2
2          3
      ..
608        6
609        7
610        8
611        9
612       10
613       11
```

```

614      12
615       1
616       2
617       3
Name: f1, Length: 618, dtype: int32
      f0  f1  f2  f3  f4  f5  f6
0  1959   1  4.5 -1.9  20 NaN  57.2
1  1959   2  7.3  0.9  15 NaN  87.2
      f0  f1  f2  f3  f4  f5  f6
0  1959   1  4.5 -1.9  20  NaN  57.2
1  1959   2  7.3  0.9  15  NaN  87.2
2  1959   3  8.4  3.1   3  NaN  81.6
3  1959   4 10.8  3.7   1  NaN 107.4
..     ... ..     ... ..     ...
613 2010  11  7.1  0.5  11 154.9  73.3
614 2010  12  3.1 -3.7  23  82.6  52.4
615 2011   1  5.8 -0.3  16 191.4  44.7
616 2011   2  8.3  3.1   5 165.8  43.5
617 2011   3 10.3  1.4  12  35.5 145.0

```

```

[618 rows x 7 columns]
      f0  f1  f2  f3  f4  f5  f6
447 1997   1  5.1 -0.7  19  8.2  NaN
202 1976   8 21.5  9.7   0  9.3 260.2
316 1986   2 -0.1 -4.9  26  9.6 116.5
570 2007   4 15.8  4.4   3 18.8 232.5
294 1984   4 11.9  2.4  11 19.1 201.7
..     ... ..     ... ..     ...
493 2000  11  8.2  3.5   1 424.4  15.2
601 2009  11  9.4  4.9   0 425.4  34.0
...
18  1960   7 16.0  9.3   0  NaN 111.3
19  1960   8 16.5  9.4   0  NaN 119.2
20  1960   9 15.0  7.9   0  NaN 120.3
21  1960  10 12.0  5.3   5  NaN  NaN
22  1960  11  8.8  2.9   5  NaN  37.3
23  1960  12  5.9  0.4  13  NaN  33.9
314 1985  12  NaN  NaN  -1  NaN  NaN
410 1993  12  7.1  1.9   8  NaN  12.4
445 1996  11  NaN  NaN  -1  NaN  NaN
446 1996  12  NaN  NaN  -1  NaN  NaN

```

```
[618 rows x 7 columns]
```

24.5.2 Example 2 - Calculating overall averages

In this example we calculate the overall monthly rainfall average for the site.

```

import numpy as np
import pandas as pd

```

```

import math
data_file_name="cwmystwythdata.txt"
month_names = ["January","Febru-
ary","March","April","May","June","July","August","Septem-
ber","October","November","December"]
matrix = np.genfromtxt( data_file_name, \
                        skip_header=7 , \
                        skip_footer=1 , \
                        usecols=(0,1,2,3,4,5,6), \
                        autostrip=True , \

dtype=(int,int,float,float,int,float,float), \
      missing_values={"---"}\
)

n=matrix.size
print(n)
print(type(matrix))
dataframe_1 = pd.DataFrame(matrix)
print(dataframe_1.columns)
print(dataframe_1.index)
print(" Average monthly rainfall = {:.6.2f} mm".for-
mat(dataframe_1['f5'].mean()))

```

Here is the output.

```

618
<class 'numpy.ndarray'>
Index(['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6'], dtype='ob-
ject')
RangeIndex(start=0, stop=618, step=1)
Average monthly rainfall = 149.43 mm

```

24.5.3 Example 3 - Calculating minimum and maximum values

This one calculates the minimum and maximum values.

```

import numpy as np
import pandas as pd
import math
data_file_name="cwmystwythdata.txt"
month_names = ["January","Febru-
ary","March","April","May","June","July","August","Septem-
ber","October","November","December"]
matrix = np.genfromtxt( data_file_name, \
                        skip_header=7 , \
                        skip_footer=1 , \
                        usecols=(0,1,2,3,4,5,6), \
                        autostrip=True , \

dtype=(int,int,float,float,int,float,float), \
      missing_values={"---"}\
)

```

```

n=matrix.size
print(n)
print(type(matrix))
dataframe_1 = pd.DataFrame(matrix)
print(dataframe_1.columns)
print(dataframe_1.index)
print(" Minimum monthly rainfall = {:.2f} mm".format(dataframe_1['f5'].min()))
print(" Maximum monthly rainfall = {:.2f} mm".format(dataframe_1['f5'].max()))

```

Here is the output.

```

618
<class 'numpy.ndarray'>
Index(['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6'], dtype='object')
RangeIndex(start=0, stop=618, step=1)
  Minimum monthly rainfall =    8.20 mm
  Maximum monthly rainfall = 425.40 mm

```

24.5.4 Example 4 - Using the groupby method

This example uses the groupby method.

Here is the source.

```

import numpy as np
import pandas as pd
import math
data_file_name="cwmystwythdata.txt"
month_names = ["January","February",
               "March","April","May","June","July","August","September",
               "October","November","December"]
matrix = np.genfromtxt( data_file_name, \
                        skip_header=7 , \
                        skip_footer=1 , \
                        usecols=(0,1,2,3,4,5,6), \
                        autostrip=True , \
                        dtype=(int,int,float,float,int,float,float), \
                        missing_values={"---"}\
                        )
n=matrix.size
print(n)
print(type(matrix))
dataframe_1 = pd.DataFrame(matrix)
print(dataframe_1.groupby('f1').mean())
dataframe_2 = dataframe_1[['f1','f5']]
print(dataframe_2)
print(" Average monthly rainfall in mm")
print( dataframe_2.groupby('f1').mean() )

```

Here is the program output.

```

618
<class 'numpy.ndarray'>
          f0          f2          f3          f4
f5          f6
f1
1  1985.882353  5.902128  0.852000  11.647059  186.553061
32.502041
2  1985.882353  6.129167  0.586000  11.862745  134.089796
56.218000
3  1985.442308  8.000000  1.640000  9.211538
139.906000  83.558000
4  1984.941176  10.528000  2.850980  6.274510  109.428571
129.322449
5  1984.500000  13.920833  5.521569  1.942308  104.872000
161.125490
6  1984.500000  16.195652  8.130000  0.153846  107.348000
148.166667
7  1984.500000  17.655102  10.084000  -0.038462  124.834000
142.354167
8  1984.500000  17.806250  10.096000  -0.038462  137.436000
138.422449
9  1984.500000  15.675510  8.146000  0.269231  150.972000
109.264583
10 1984.960784  12.477551  6.191837  1.843137  189.161224
83.142553
11 1984.960784  8.730612  3.178000  6.117647  205.414583
46.360870
12 1984.960784  6.637500  1.338776  10.215686  210.752174
30.087234
      f1      f5
0      1      NaN
1      2      NaN
2      3      NaN
3      4      NaN
4      5      NaN
5      6      NaN
6      7      NaN
7      8      NaN
8      9      NaN
9     10      NaN
10     11      NaN
11     12      NaN
12      1      NaN
13      2      NaN
14      3      NaN
15      4      NaN
16      5      NaN
17      6      NaN

```

```
18      7      NaN
19      8      NaN
20      9      NaN
21     10      NaN
22     11      NaN
23     12      NaN
24      1    144.8
25      2    112.5
26      3     77.2
27      4    130.7
28      5     66.3
29      6     66.1
..     ..     ...
588    10    317.6
589    11    237.3
590    12    140.1
591     1    167.1
592     2     35.1
593     3    110.8
594     4     78.6
595     5    126.1
596     6     99.9
597     7    219.9
598     8    140.7
599     9     86.3
600    10    126.6
601    11    425.4
602    12    167.7
603     1    127.9
604     2     70.4
605     3    102.0
606     4     56.8
607     5     71.5
608     6     80.5
609     7    209.3
610     8     88.8
611     9    181.2
612    10    108.0
613    11    154.9
614    12     82.6
615     1    191.4
616     2    165.8
617     3     35.5
```

```
[618 rows x 2 columns]
Average monthly rainfall in mm
      f5
```

```
f1
1    186.553061
```



```
2    134.089796
3    139.906000
4    109.428571
5    104.872000
6    107.348000
7    124.834000
8    137.436000
9    150.972000
10   189.161224
11   205.414583
12   210.752174
```

The first groupy output has calculated avearges for the years!

24.6 Summary

We have just shown part of what is possible with Pandas. Our interest was in in showing some of the ways you could process the Met Office data. A lot of help is available on line. Happy searching.

24.7 Problems

Run the examples in this chapter.

Modify the examples to work with a Met Office station of your choice.

25 Windows programming in Python

25.1 Introduction to Windows programming

The following information is taken from the Python FAQ. These are some of the platform-independent GUI toolkits for Python

- Tkinter
- wxWidgets
- Qt
- Gtk+
- FLTK
- FOX
- OpenGL

In these notes we will be using Tkinter.

25.2 Tkinter

Standard builds of Python include an object-oriented interface to the Tcl/Tk widget set, called Tkinter. This is available on both the Linux and Windows systems I use. They are dual boot Windows 10 and openSuSe 13.1.

For more info about Tk, including pointers to the source, see the Tcl/Tk home page at <http://www.tcl.tk>.

Tcl/Tk is fully portable to the Mac OS X, Windows, and Unix platforms.

The web site below

<https://wiki.python.org/moin/TkInter>

is where the following information was taken.

Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk.

Tkinter is not the only Gui Programming toolkit for Python. It is however the most commonly used one. Cameron Laird calls the yearly decision to keep TkInter "one of the minor traditions of the Python world."

There is a Tkinter wiki:

<http://tkinter.unpythonic.net/wiki/>

which is a good read ;-)

We will start by having a look at a few simple programs.

25.3 Example 1 - simple test program included with Tkinter distribution

Here is the possibly the simplest example. This just tests out the installation.

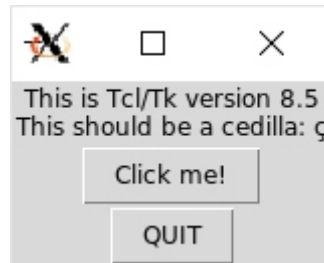
```
python3
>>import tkinter
>>tkinter._test()
```

This pops up a small window with the following text

```
This is Tcl/Tk version 8.5
This should be a cedilla
```

```
[Click me]
[QUIT]
```

This is the simplest example you can think of, only two lines of code! You can try this out from the command line.



The output above is taken from a Windows 10 system. In this example we use

```
import tkinter
```

which is one of the ways that Python has for importing a module. In most graphics examples on the web you will see a variety of import statements. Here is a brief explanation of the various forms

`import foobah` - import the module foobah and create a reference to that module in the current namespace. This enables you to use `foobah.name` to refer to things defined in the foobah module;

`from foobah import *` - imports the module foobah, and creates references in the current namespace to all public objects defined by that module (that is, everything that doesn't have a name starting with “_”). After you've run this statement, you can simply use a plain name to refer to things defined in module foobah. foobah itself is not defined, so `foobah.name` doesn't work;

`from foobah import a,b,c` - import the module foobah and create references in the current namespace to the specific objects, i.e. you can use a, b and c in your program;

`foobah = __import__('foobah')` works like `import foobah` with the difference that first you pass the module name as a string, and second explicitly assign it to a variable in your current namespace;

The Python recommendation is to use `import`. However they make an exception for Tkinter, where the recommendation is to use `from ... import`. Tkinter is designed to add only the widget classes and related constants to your current namespace. Using the `import Tkinter` makes your program slightly harder to read. Let us have a look now at a simple variant.

25.4 Example 2 - Hello world version 1

This is the simple hello world example. The hello world example goes back to C and Kernighan and Ritchie.

```
from tkinter import *
```

```
root = Tk()
```

```
w = Label(root, text="Hello world")
w.pack()
```

```
root.mainloop()
```

The output is similar to the test example. We have a Window with the typical gui layout. Here is a screen shot of this program.



The output is from a Windows 10 system. Let us look at the code in more depth.

You start by importing the Tkinter module. It contains all classes, functions and other things needed to work with the Tk toolkit. The following is a Python 3 version.

```
from tkinter import *
```

To initialize Tkinter, we have to create a Tk root widget. This is a window with a small number of gui components, provided by your window manager. Here is the code.

```
root = Tk()
```

We are going to use a Label widget to display hello world.

```
w = Label(root, text="Hello world")
w.pack()
```

Label widgets can display text, graphics or icons. In this example we use the text option.

The pack method tells the widget it to size itself to fit the given text, and make itself visible. The window appears when we've entered the Tkinter event loop:

```
root.mainloop()
```

The program will stay in the event loop until we close the window. The event loop doesn't only handle events from the user (such as mouse clicks and key presses) or the windowing system (such as redraw events and window configuration messages), it also handle operations queued by Tkinter itself. Among these operations are geometry management (queued by the pack method) and display updates. This also means that the application window will not appear before you enter the main loop.

There are two variants. They use

```
import tkinter
```

```
import tkinter as tk
```

respectively.

25.5 Example 3 - Hello world variant 1

Here is the first variant.

```
import tkinter
```

```
root = tkinter.Tk()
```

```
w = tkinter.Label(root, text="Hello world")
w.pack()
```

```
root.mainloop()
```

In this example we prefix the components with tkinter.

25.6 Example 4 - Hello world variant 2

Here is the second variant.

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Hello world")
w.pack()

root.mainloop()
```

Using the `as` option reduces the amount of typing.

25.7 Example 5 - Hello world version 2

Here is the source code for this example. It was taken from <http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/minimal-app.html>

It only contains a [QUIT] button.

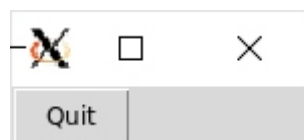
```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.grid()
        self.createWidgets()

    def createWidgets(self):
        self.quitButton = tk.Button(self, text='Quit',
                                     command=self.quit)
        self.quitButton.grid()

app = Application()
app.master.title('Sample application')
app.mainloop()
```

Run the program and look at the output. You will see something similar to the screen shot below.



Let us look at the code in a bit more depth.

```
import tkinter as tk

Inherit from the Frame class.
```

```
tk.Frame.__init__(self, master)
```

Call the constructor.

```
self.grid()
```

The grid method displays a widget on your application screen. In fact it registers the widget with the grid geometry manager.

```
def createWidgets(self):
    self.quitButton = tk.Button(self, text='Quit',
                                command=self.quit)
```

Creates a [Quit] button.

```
self.quitButton.grid()
```

Places the button on the frame.

```
app = Application()
```

Instantiate the Application class.

```
app.master.title('Sample application')
```

Set the title to 'Sample Application'

```
app.mainloop()
```

Starts the program.

25.8 Example 6 - Hello world version 3

Here is the source code for the third hello world program.

```
from tkinter import *
```

```
class App:
```

```
    def __init__(self, master):
```

```
        frame = Frame(master)
        frame.pack()
```

```
        self.button = Button(frame, text="QUIT",
fg="red", command=frame.quit)
        self.button.pack(side=LEFT)
```

```
        self.hi_there = Button(frame, text="Press me",
command=self.say_hi)
        self.hi_there.pack(side=LEFT)
```

```
    def say_hi(self):
        print("Hello world")
```

```
root = Tk()
```

```
app = App(root)
```

```
root.mainloop()
root.destroy() # optional; see description below
```

This application is written as a class. The constructor (the `__init__` method) is called with a parent widget (the master), to which it adds a number of child widgets. The constructor starts by creating a `Frame` widget. A frame is a simple container, and is in this case only used to hold the other two widgets.

```
class App:
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
```

The frame instance is stored in a local variable called `frame`. After creating the widget, we immediately call the `pack` method to make the frame visible. We then create two `Button` widgets, as children to the frame.

```
self.button = Button(frame, text="QUIT", fg="red",
command=frame.quit)
self.button.pack(side=LEFT)
```

```
self.hi_there = Button(frame, text="Press me",
command=self.say_hi)
self.hi_there.pack(side=LEFT)
```

This time, we pass a number of options to the constructor, as keyword arguments. The first button is labelled “QUIT”, and is made red (`fg` is short for foreground). The second is labelled “Hello”. Both buttons also take a `command` option. This option specifies a function, or (as in this case) a bound method, which will be called when the button is clicked.

The button instances are stored in instance attributes. They are both packed, but this time with the `side=LEFT` argument. This means that they will be placed as far left as possible in the frame; the first button is placed at the frame’s left edge, and the second is placed just to the right of the first one (at the left edge of the remaining space in the frame, that is). By default, widgets are packed relative to their parent (which is `master` for the frame widget, and the frame itself for the buttons). If the `side` is not given, it defaults to `TOP`.

The “hello” button callback is given next. It simply prints a message to the console every time the button is pressed:

```
def say_hi(self):
    print "Hello world"
```

In computer programming, a callback is a piece of executable code that is passed as an argument to other code, which is expected to call back (execute) the argument at some convenient time. Programming languages support callbacks in different ways, often implementing them with subroutines, lambda expressions, blocks, or function pointers. Python allows a function object to be passed. Events and event handlers, as used in .NET languages, provide generalized syntax for callbacks.

Finally, we provide some script level code that creates a `Tk` root widget, and one instance of the `App` class using the root widget as its parent:

```
root = Tk()

app = App(root)

root.mainloop()
root.destroy()
```

The mainloop call enters the Tk event loop, in which the application will stay until the quit method is called (just click the QUIT button), or the window is closed.

25.9 The remaining examples

The rest of the examples are based on information from the following site.

<http://effbot.org/tkinterbook/tkinter-index.htm#introduction>

We will be looking at examples using the following widgets:

Button

A button can contain text or images. A function or callback can be associated with the button. When the button is pressed the function is invoked.

Entry

The entry widget is used to enter text strings. This widget is restricted to one line. The text widget works with multiple lines. A get method is used to retrieve the text typed in.

Label

Displays text or an image on the screen. The text may span multiple lines.

Message

Display a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.

Text

Formatted text display. Allows you to display and edit text with various styles and attributes. Also supports embedded images and windows.

A callback is executable code. It is passed as an argument to other code. Callbacks are commonly used in windowing systems, where there is a requirement to respond to events, e.g. mouse clicks or button presses.

We will look at simple examples of each of them.

25.10 Example 7 - simple button example

Here is the source code.

```
# converted to python3

# Tkinter -> tkinter
# print    -> print()

from tkinter import *

master = Tk()

def bl():
    print("Button 1 pressed")
```



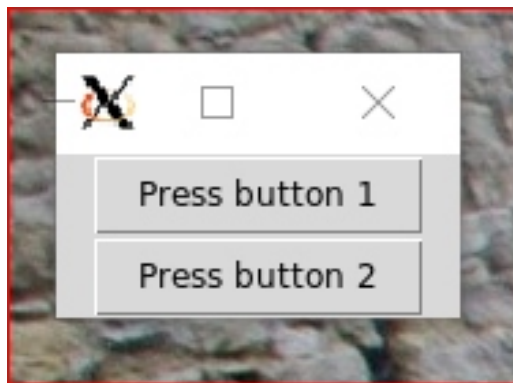
```
def b2():
    print("Button 2 pressed")

b1 = Button(master, text="Press button 1", command=b1)
b2 = Button(master, text="Press button 2", command=b2)

b1.pack()
b2.pack()

mainloop()
```

Here is the output from a Windows 10 system.



Pressing the buttons causes print statements in the command window.

First we define two methods to carry out the action we want when the buttons are pressed. In Windows or GUI programming we link the press of a button to a handler or method. The name for this method is a callback in general computing terminology.

The next thing we do is create two buttons, b1 and b2, using the Button constructor.

We next apply the pack() method to make the buttons visible.

Finally we start the program.

Here is sample output from the command window.

```
$ python3 button_01.py
Button 1 pressed
Button 1 pressed
Button 2 pressed
Button 2 pressed
```

See

<http://effbot.org/tkinterbook/button.htm>

for more detailed coverage of buttons.

25.11 Example 8 - Button and message example

Here is the source code.

```
# converted to python3

# Tkinter -> tkinter
# print    -> print()
```

```
from tkinter import *

master = Tk()

def b1():
    m1 = Message(master, text="Button 1 pressed")
    m1.pack()

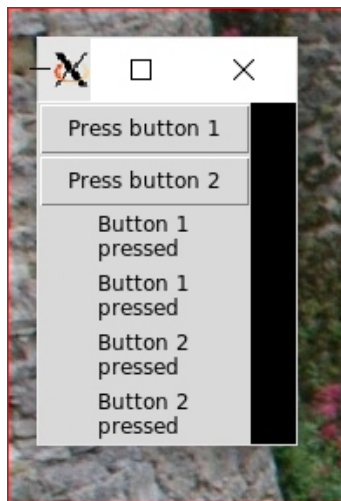
def b2():
    m2 = Message(master, text="Button 2 pressed")
    m2.pack()

b1 = Button(master, text="Press button 1", command=b1)
b2 = Button(master, text="Press button 2", command=b2)

b1.pack()
b2.pack()

mainloop()
```

This is a simple variant of the first button example, where we have replaced the `print()` statements with display messages.



The sample output is from a Windows 10 system.

See

<http://effbot.org/tkinterbook/message.htm>
for more detailed coverage of messages.

25.12 Example 9 - Button, message and entry example

In this example we get user input and then press one of two buttons to do a calculation and display the results. Here is the source.

```
# converted to python3

# Tkinter -> tkinter
# print    -> print()
```

```
from tkinter import *

master = Tk()

e = Entry(master,width=25)
e.pack()

e.focus_set()

def square():
    i=int(e.get())
    j=(i*i)
    k=IntVar()
    k.set(j)
    m1 = Message(master,text=" Number squared is
",textvariable=k)
    m1.pack()

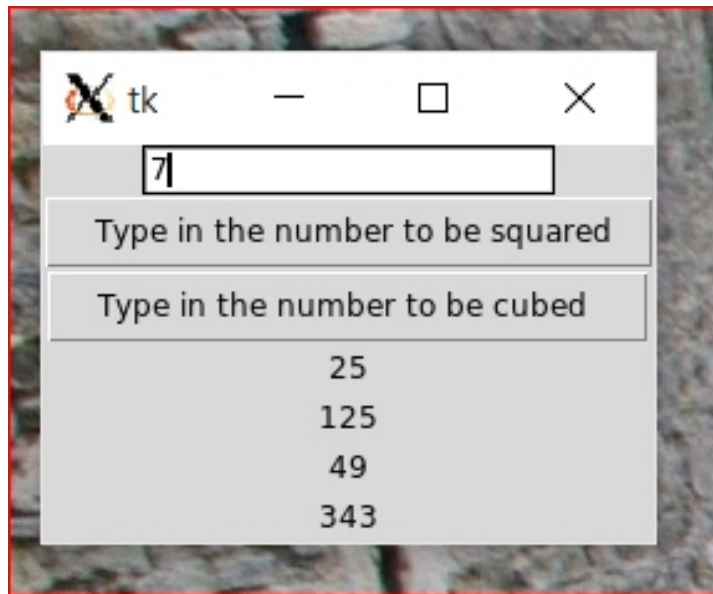
def cube():
    i=int(e.get())
    j = (i * i * i)
    k=IntVar()
    k.set(j)
    m2 = Message(master,text=" Number cubed is ",
textvariable=k)
    m2.pack()

square = Button(master, text=" Type in the number to be
squared", command=square)
cube    = Button(master, text=" Type in the number to be
cubed  ", command=cube)

square.pack()
cube.pack()

mainloop()
```

Here is a screenshot of this example.



This is from a Windows 10 system.

See

<http://effbot.org/tkinterbook/entry.htm>

for more detailed coverage of the entry widget.

25.13 Example 10 - Button, entry and text widget example

In this example we replace the message widgets with a text widget. Here is the source code.

```
# converted to python3

# Tkinter -> tkinter
# print    -> print()

from tkinter import *

master = Tk()

# text entry

e = Entry(master,width=25)
e.pack()
e.focus_set()

# Text widget

T = Text(master,height=10,width=30)
T.pack()

def square():
    i=int(e.get())
    j=(i*i)
    j_string = str(j)
```

```

T.insert(END," Number squared is ")
T.insert(END,j_string)
T.insert(END,"\n")
T.pack()

def cube():
    i=int(e.get())
    j = (i * i * i)
    j_string = str(j)
    T.insert(END," Number cubed is ")
    T.insert(END,j_string)
    T.insert(END,"\n")
    T.pack()

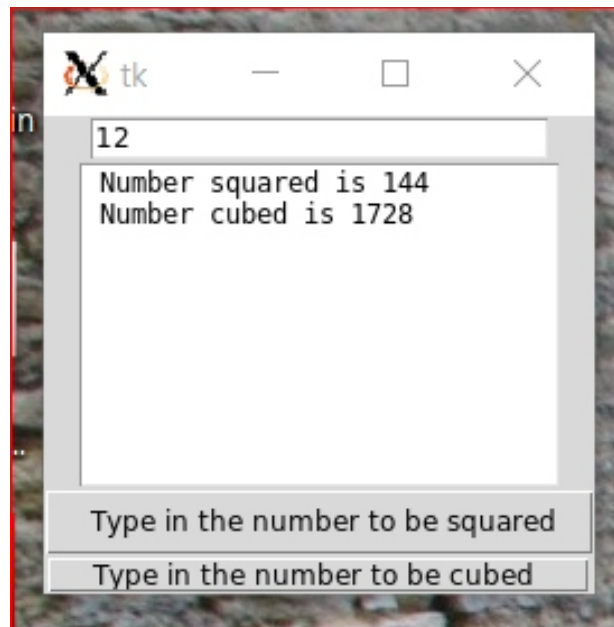
square = Button(master, text=" Type in the number to be
squared", command=square)
cube    = Button(master, text=" Type in the number to be
cubed  ", command=cube)

square.pack()
cube.pack()

mainloop()

```

Here is sample output.



See

<http://effbot.org/tkinterbook/text.htm>

for more detailed coverage of the text widget.

25.14 Tkinter on line examples and resources

I have used and reworked examples from a variety of sources including

John Shipmans site. Visit

<http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html>

this is the home page. There is also a 168 page Tkinter 8.5 reference manual there. We downloaded and used the pdf version.

the Thinking in Tkinter site.

<http://thinkingtkinter.sourceforge.net/>

This site has some simple examples.

The Introduction to Tkinter site.

<http://effbot.org/tkinterbook/>

The explanation on this site is very good.

25.15 Other options

There are other options in this area and we provide some additional information below.

25.15.1 QT Creator

The following is taken from the wikipedia entry.

Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which is part of the SDK for the Qt GUI application development framework.

It includes a visual debugger and an integrated GUI layout and forms designer.

The editor's features include syntax highlighting and autocompletion.

Qt Creator uses the C++ compiler from the GNU Compiler Collection on Linux and FreeBSD.

On Windows it can use MinGW or MSVC with the default install and can also use Microsoft Console Debugger when compiled from source code.

Clang is also supported.

The following is taken from the Qt Designer Manual

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. You can compose and customize your windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test them using different styles and resolutions.

Widgets and forms created with Qt Designer integrate seamlessly with programmed code, using Qt's signals and slots mechanism, so that you can easily assign behavior to graphical elements. All properties set in Qt Designer can be changed dynamically within the code. Furthermore, features like widget promotion and custom plugins allow you to use your own components with Qt Designer.

Note: You have the option of using Qt Quick for user interface design rather than widgets. It is a much easier way to write many kinds of applications. It enables a completely customizable appearance, touch-reactive elements, and smooth animated transitions, backed up by the power of OpenGL graphics acceleration.

If you are new to Qt Designer, you can take a look at the Getting To Know Qt Designer document. For a quick tutorial on how to use Qt Designer, refer to A Quick Start to Qt Designer.

Details of pricing can be found at

<https://www1.qt.io/buy-product/>

25.16 Problems

1. Compile and run the examples in this chapter.
2. Rewrite some of the earlier examples to have a gui interface.

26 Graphics plotting in Python using matplotlib

26.1 Graphics plotting with matplotlib

In this chapter we will look at a small number of examples of plotting graphs using matplotlib. Their site is given below.

<http://matplotlib.org/>

Here is a bit of blurb from the above site.

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB or Mathematica), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, , etc, with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Examples are available.

<http://matplotlib.org/examples/index.html>

and with most plotting libraries choosing an example that does part of what you want is normally an effective way to start.

Here is a list of their example headings.

- animation scatter plots
- api
- axes_grid
- color
- event_handling
- images_contours_and_fields
- lines_bars_and_markers
- misc
- mplot3d
- pie_and_polar_charts
- pylab_
- scales
- shapes_and_collections ce
- showcase

specialty_plots
statistics
style_sheets
subplots_axes_and_figures
tests b
text_labels_and_annotations
ticks_and_spines
units
user_interfaces
widgets

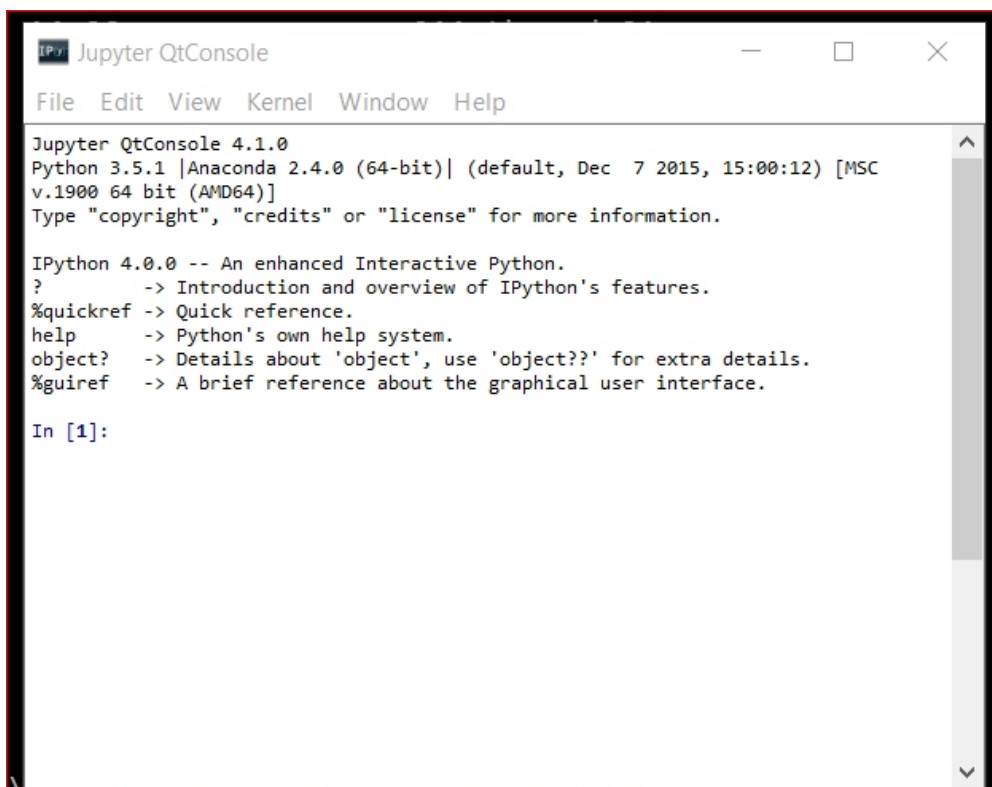
On a Windows system with the most recent version of cygwin the following worked

```
cygwin install
conda install matplotlib
jupyter qtconsole
    from pylab import *
    plot([1,2,3,4])
    show()
```

and this produced a sample plot.

26.2 The jupyter qtconsole on Windows

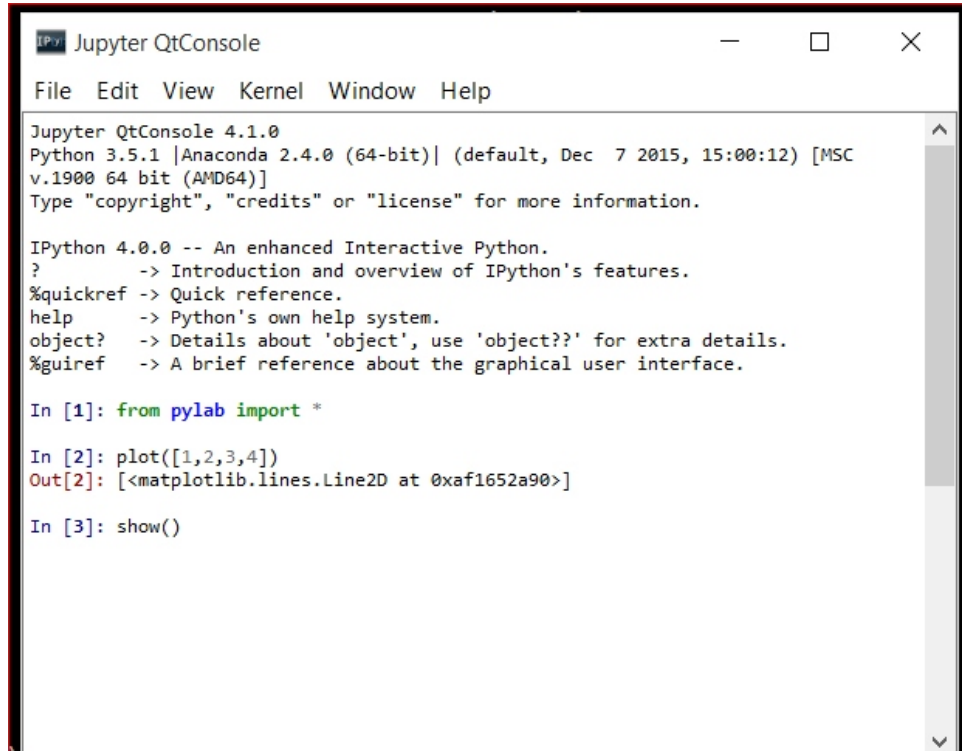
Here is a screenshot of the jupyter qtconsole on a Windows 10 system running a cygwin install from January 2016.



The next screenshot shows typing in the following commands

```
from pylab import *  
plot([1,2,3,4])  
show()
```

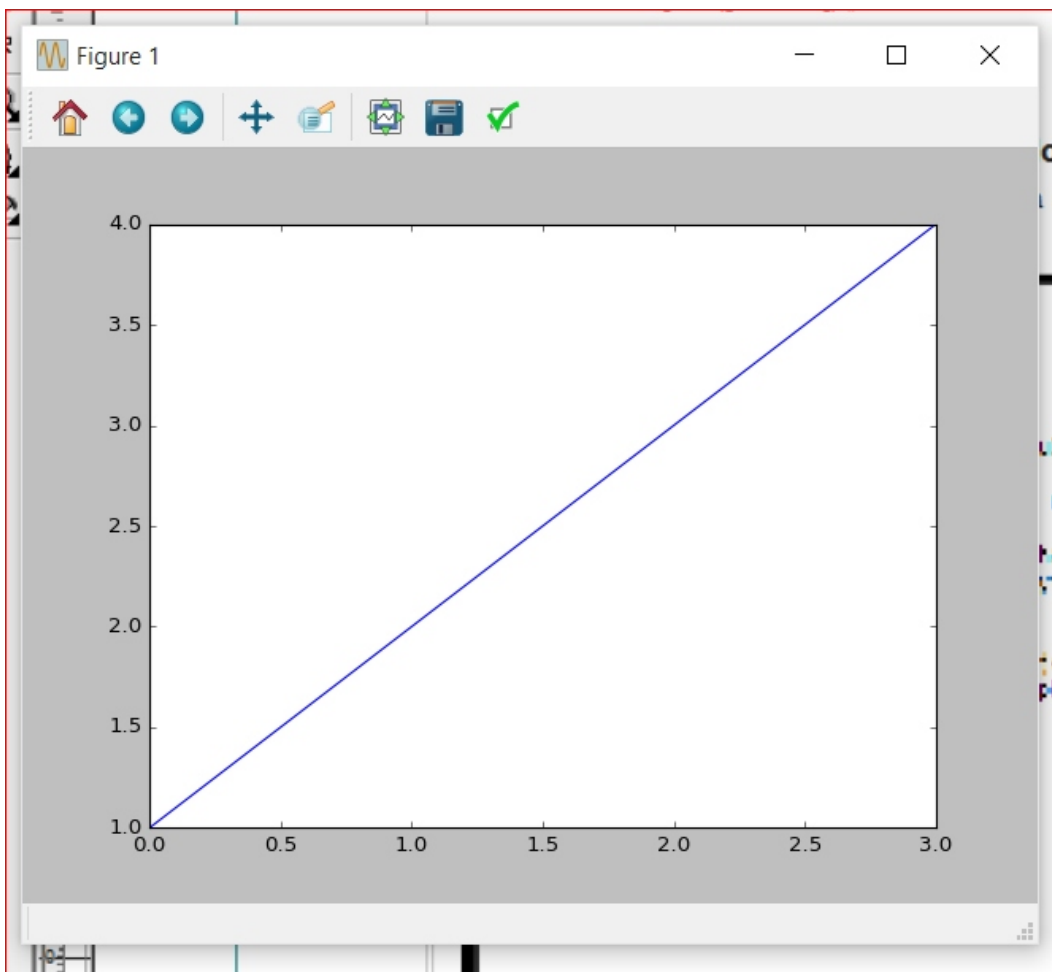
Here is the screen shot.



The screenshot shows a Jupyter QtConsole window with the following content:

```
Jupyter QtConsole 4.1.0  
Python 3.5.1 |Anaconda 2.4.0 (64-bit)| (default, Dec 7 2015, 15:00:12) [MSC  
v.1900 64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more information.  
  
IPython 4.0.0 -- An enhanced Interactive Python.  
?          -> Introduction and overview of IPython's features.  
%quickref -> Quick reference.  
help      -> Python's own help system.  
object?   -> Details about 'object', use 'object??' for extra details.  
%gui      -> A brief reference about the graphical user interface.  
  
In [1]: from pylab import *  
  
In [2]: plot([1,2,3,4])  
Out[2]: [<matplotlib.lines.Line2D at 0xaf1652a90>]  
  
In [3]: show()
```

Here is the plot.

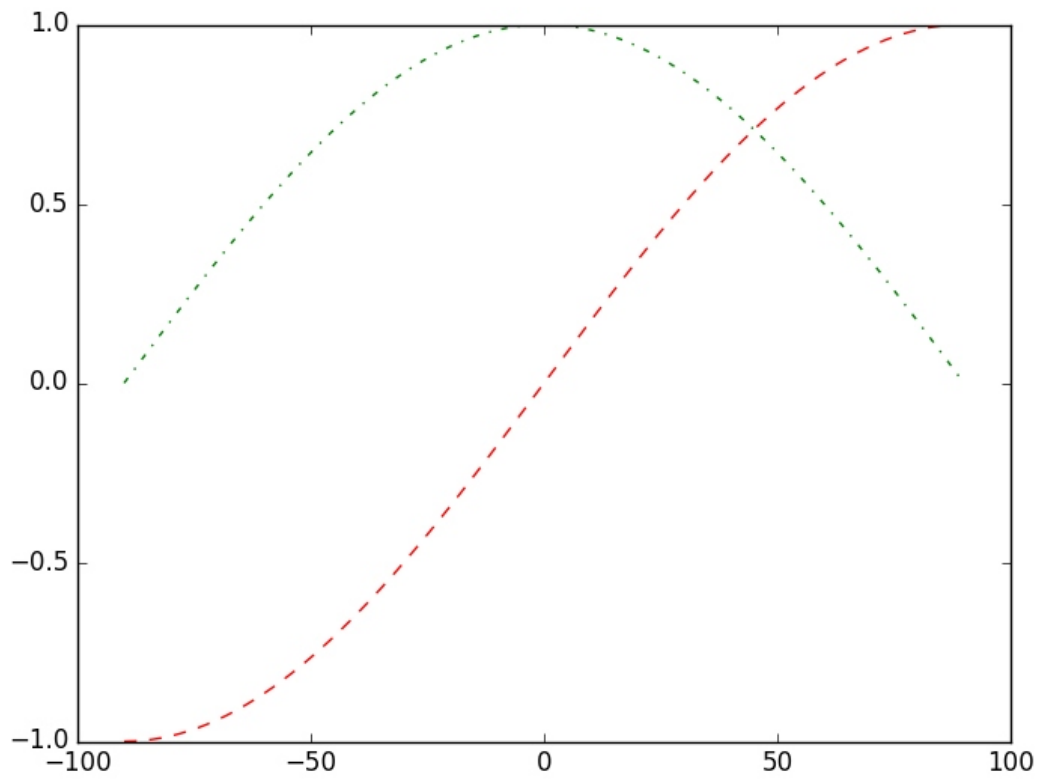


26.3 Example 1 - Simple trigonometric plot

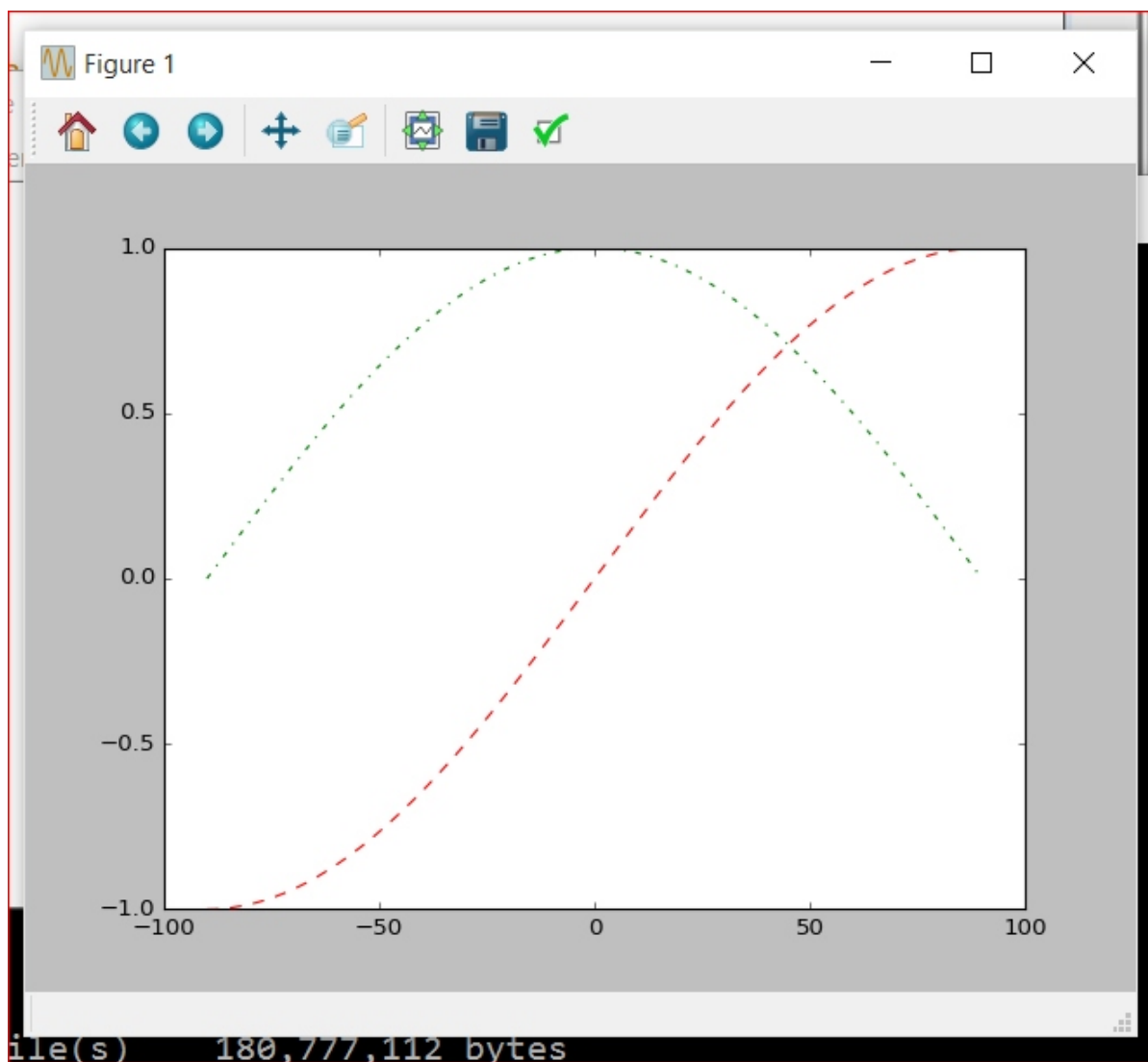
Here is the source code.

```
from pylab import *
import numpy as np
import math
size=181
x=0.0
xaxis = np.empty([size],dtype=int32)
y1     = np.empty([size],dtype=float64)
y2     = np.empty([size],dtype=float64)
for i in range(size):
    xaxis[i]=(i-90)
    x=(i-90)*math.pi/180.0
    y1[i] = math.sin(x)
    y2[i] = math.cos(x)
plot(xaxis,y1,'r--',xaxis,y2,'g-.')
show()
```

Here is the saved plot file.



Here is the screen shot of the window it appears in.



Here is a brief coverage of the matplotlib api.

<code>acorr(x, *[, data])</code>	Plot the autocorrelation of x.
<code>angle_spectrum(x[, Fs, Fc, window, pad_to, ...])</code>	Plot the angle spectrum.
<code>annotate(text, xy, *args, **kwargs)</code>	Annotate the point xy with text s.
<code>arrow(x, y, dx, dy, **kwargs)</code>	Add an arrow to the axes.
<code>autoscale([enable, axis, tight])</code>	Autoscale the axis view to the data (toggle).
<code>autumn()</code>	Set the colormap to autumn". "
<code>axes([arg])</code>	Add an axes to the current figure and make it the current axes.
<code>axhline([y, xmin, xmax])</code>	Add a horizontal line across the axis.

<code>axhspan(ymin, ymax[, xmin, xmax])</code>	Add a horizontal span (rectangle) across the axis.
<code>axis(*v, **kwargs)</code>	Convenience method to get or set some axis properties.
<code>axvline([x, ymin, ymax])</code>	Add a vertical line across the axes.
<code>axvspan(xmin, xmax[, ymin, ymax])</code>	Add a vertical span (rectangle) across the axes.
<code>bar(x, height[, width, bottom, align, data])</code>	Make a bar plot.
<code>barbs(*args[, data])</code>	Plot a 2-D field of barbs.
<code>barh(y, width[, height, left, align])</code>	Make a horizontal bar plot.
<code>bone()</code>	Set the colormap to bone". "
<code>box([on])</code>	Turn the axes box on or off on the current axes.
<code>boxplot(x[, notch, sym, vert, whis, ...])</code>	Make a box and whisker plot.
<code>broken_barh(xranges, yrange, *[, data])</code>	Plot a horizontal sequence of rectangles.
<code>cla()</code>	Clear the current axes.
<code>clabel(CS, *args, **kwargs)</code>	Label a contour plot.
<code>clf()</code>	Clear the current figure.
<code>clim([vmin, vmax])</code>	Set the color limits of the current image.
<code>close([fig])</code>	Close a figure window.
<code>cohere(x, y[, NFFT, Fs, Fc, detrend, ...])</code>	Plot the coherence between x and y.
<code>colorbar([mappable, cax, ax])</code>	Add a colorbar to a plot.
<code>connect(s, func)</code>	Connect event with string s to func.
<code>contour(*args[, data])</code>	Plot contours.
<code>contourf(*args[, data])</code>	Plot contours.
<code>cool()</code>	Set the colormap to cool". "
<code>copper()</code>	Set the colormap to copper". "
<code>csd(x, y[, NFFT, Fs, Fc, detrend, window, ...])</code>	Plot the cross-spectral density.

<code>delaxes([ax])</code>	Remove the Axes <code>ax</code> (defaulting to the current axes) from its figure.
<code>disconnect(cid)</code>	Disconnect callback id <code>cid</code>
<code>draw()</code>	Redraw the current figure.
<code>errorbar(x, y[, yerr, xerr, fmt, ecolor, ...])</code>	Plot <code>y</code> versus <code>x</code> as lines and/or markers with attached errorbars.
<code>eventplot(positions[, orientation, ...])</code>	Plot identical parallel lines at the given positions.
<code>figimage(*args, **kwargs)</code>	Add a non-resampled image to the figure.
<code>figlegend(*args, **kwargs)</code>	Place a legend in the figure.
<code>fignum_exists(num)</code>	Return whether the figure with the given id exists.
<code>figtext(x, y, s, *args, **kwargs)</code>	Add text to figure.
<code>figure([num, figsize, dpi, facecolor, ...])</code>	Create a new figure.
<code>fill(*args[, data])</code>	Plot filled polygons.
<code>fill_between(x, y1[, y2, where, ...])</code>	Fill the area between two horizontal curves.
<code>fill_betweenx(y, x1[, x2, where, step, ...])</code>	Fill the area between two vertical curves.
<code>findobj([o, match, include_self])</code>	Find artist objects.
<code>flag()</code>	Set the colormap to <code>flag</code> ". "
<code>gca(**kwargs)</code>	Get the current Axes instance on the current figure matching the given keyword args, or create one.
<code>gcf()</code>	Get a reference to the current figure.
<code>gci()</code>	Get the current colorable artist.
<code>get_current_fig_manager()</code>	Return the figure manager of the active figure.
<code>get_figlabels()</code>	Return a list of existing figure labels.
<code>get_fignums()</code>	Return a list of existing figure numbers.
<code>get_plot_commands()</code>	Get a sorted list of all of the plotting commands.
<code>ginput(*args, **kwargs)</code>	Blocking call to interact with a figure.
<code>gray()</code>	Set the colormap to <code>gray</code> ". "
<code>grid([b, which, axis])</code>	Configure the grid lines.

<code>hexbin(x, y[, C, gridsize, bins, xscale, ...])</code>	Make a hexagonal binning plot.
<code>hist(x[, bins, range, density, weights, ...])</code>	Plot a histogram.
<code>hist2d(x, y[, bins, range, normed, weights, ...])</code>	Make a 2D histogram plot.
<code>hlines(y, xmin, xmax[, colors, linestyles, ...])</code>	Plot horizontal lines at each y from xmin to xmax.
<code>hot()</code>	Set the colormap to hot". "
<code>hsv()</code>	Set the colormap to hsv". "
<code>imread(fname[, format])</code>	Read an image from a file into an array.
<code>imsave(fname, arr, **kwargs)</code>	Save an array as in image file.
<code>imshow(X[, cmap, norm, aspect, ...])</code>	Display an image, i.e.
<code>inferno()</code>	Set the colormap to inferno". "
<code>install_repl_displayhook()</code>	Install a repl display hook so that any stale figure are automatically redrawn when control is returned to the repl.
<code>ioff()</code>	Turn the interactive mode off.
<code>ion()</code>	Turn the interactive mode on.
<code>isinteractive()</code>	Return the status of interactive mode.
<code>jet()</code>	Set the colormap to jet". "
<code>legend(*args, **kwargs)</code>	Place a legend on the axes.
<code>locator_params([axis, tight])</code>	Control behavior of tick locators.
<code>loglog(*args, **kwargs)</code>	Make a plot with log scaling on both the x and y axis.
<code>magma()</code>	Set the colormap to magma". "
<code>magnitude_spectrum(x[, Fs, Fc, window, ...])</code>	Plot the magnitude spectrum.
<code>margins(*margins[, x, y, tight])</code>	Set or retrieve autoscaling margins.
<code>matshow(A[, fignum])</code>	Display an array as a matrix in a new figure window.
<code>minorticks_off()</code>	Remove minor ticks from the axes.
<code>minorticks_on()</code>	Display minor ticks on the axes.

<code>nipy_spectral()</code>	Set the colormap to <code>nipy_spectral</code> ". "
<code>pause(interval)</code>	Pause for interval seconds.
<code>pcolor(*args[, alpha, norm, cmap, vmin, ...])</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>pcolormesh(*args[, alpha, norm, cmap, vmin, ...])</code>	Create a pseudocolor plot with a non-regular rectangular grid.
<code>phase_spectrum(x[, Fs, Fc, window, pad_to, ...])</code>	Plot the phase spectrum.
<code>pie(x[, explode, labels, colors, autopct, ...])</code>	Plot a pie chart.
<code>pink()</code>	Set the colormap to <code>pink</code> ". "
<code>plasma()</code>	Set the colormap to <code>plasma</code> ". "
<code>plot(*args[, scalex, scaley, data])</code>	Plot y versus x as lines and/or markers.
<code>plot_date(x, y[, fmt, tz, xdate, ydate, data])</code>	Plot data that contains dates.
<code>plotfile(fname[, cols, plotfuncs, comments, ...])</code>	Plot the data in a file.
<code>polar(*args, **kwargs)</code>	Make a polar plot.
<code>prism()</code>	Set the colormap to <code>prism</code> ". "
<code>psd(x[, NFFT, Fs, Fc, detrend, window, ...])</code>	Plot the power spectral density.
<code>quiver(*args[, data])</code>	Plot a 2-D field of arrows.
<code>quiverkey(Q, X, Y, U, label, **kw)</code>	Add a key to a quiver plot.
<code>rc(group, **kwargs)</code>	Set the current rc params.
<code>rc_context([rc, fname])</code>	Return a context manager for managing rc settings.
<code>rcdefaults()</code>	Restore the rc params from Matplotlib's internal default style.
<code>rgrids(*args, **kwargs)</code>	Get or set the radial gridlines on the current polar plot.
<code>savefig(*args, **kwargs)</code>	Save the current figure.
<code>sca(ax)</code>	Set the current Axes instance to ax.
<code>scatter(x, y[, s, c, marker, cmap, norm, ...])</code>	A scatter plot of y vs x with varying marker size and/or color.

<code>sci(im)</code>	Set the current image.
<code>semilogx(*args, **kwargs)</code>	Make a plot with log scaling on the x axis.
<code>semilogy(*args, **kwargs)</code>	Make a plot with log scaling on the y axis.
<code>set_cmap(cmap)</code>	Set the default colormap.
<code>setp(obj, *args, **kwargs)</code>	Set a property on an artist object.
<code>show(*args, **kw)</code>	Display a figure.
<code>specgram(x[, NFFT, Fs, Fc, detrend, window, ...])</code>	Plot a spectrogram.
<code>spring()</code>	Set the colormap to spring". "
<code>spy(Z[, precision, marker, markersize, ...])</code>	Plot the sparsity pattern of a 2D array.
<code>stackplot(x, *args[, data])</code>	Draw a stacked area plot.
<code>stem(*args[, linefmt, markerfmt, basefmt, ...])</code>	Create a stem plot.
<code>step(x, y, *args[, where, data])</code>	Make a step plot.
<code>streamplot(x, y, u, v[, density, linewidth, ...])</code>	Draw streamlines of a vector flow.
<code>subplot(*args, **kwargs)</code>	Add a subplot to the current figure.
<code>subplot2grid(shape, loc[, rowspan, colspan, fig])</code>	Create an axis at specific location inside a regular grid.
<code>subplot_tool([targetfig])</code>	Launch a subplot tool window for a figure.
<code>subplots([nrows, ncols, sharex, sharey, ...])</code>	Create a figure and a set of subplots.
<code>subplots_adjust([left, bottom, right, top, ...])</code>	Tune the subplot layout.
<code>summer()</code>	Set the colormap to summer". "
<code>suptitle(t, **kwargs)</code>	Add a centered title to the figure.
<code>switch_backend(newbackend)</code>	Close all open figures and set the Matplotlib backend.
<code>table(**kwargs)</code>	Add a table to the current axes.
<code>text(x, y, s[, fontdict, withdash])</code>	Add text to the axes.
<code>thetagrids(*args, **kwargs)</code>	Get or set the theta gridlines on the current polar plot.

<code>tick_params([axis])</code>	Change the appearance of ticks, tick labels, and gridlines.
<code>ticklabel_format(*[, axis, style, ...])</code>	Change the ScalarFormatter used by default for linear axes.
<code>tight_layout([pad, h_pad, w_pad, rect])</code>	Automatically adjust subplot parameters to give specified padding.
<code>title(label[, fontdict, loc, pad])</code>	Set a title for the axes.
<code>tricontour(*args, **kwargs)</code>	Draw contours on an unstructured triangular grid.
<code>tricontourf(*args, **kwargs)</code>	Draw contours on an unstructured triangular grid.
<code>tripcolor(*args, **kwargs)</code>	Create a pseudocolor plot of an unstructured triangular grid.
<code>triplot(*args, **kwargs)</code>	Draw a unstructured triangular grid as lines and/or markers.
<code>twinx([ax])</code>	Make a second axes that shares the x-axis.
<code>twiny([ax])</code>	Make a second axes that shares the y-axis.
<code>uninstall_repl_displayhook()</code>	Uninstall the matplotlib display hook.
<code>violinplot(dataset[, positions, vert, ...])</code>	Make a violin plot.
<code>viridis()</code>	Set the colormap to viridis". "
<code>vlines(x, ymin, ymax[, colors, linestyle, ...])</code>	Plot vertical lines.
<code>waitforbuttonpress(*args, **kwargs)</code>	Blocking call to interact with the figure.
<code>winter()</code>	Set the colormap to winter". "
<code>xcorr(x, y[, normed, detrend, usevlines, ...])</code>	Plot the cross correlation between x and y.
<code>xkcd([scale, length, randomness])</code>	Turn on xkcd sketch-style drawing mode. This will only have effect on things drawn after this function is called..
<code>xlabel(xlabel[, fontdict, labelpad])</code>	Set the label for the x-axis.
<code>xlim(*args, **kwargs)</code>	Get or set the x limits of the current axes.
<code>xscale(value, **kwargs)</code>	Set the x-axis scale.

<code>xticks([ticks, labels])</code>	Get or set the current tick locations and labels of the x-axis.
<code>ylabel(ylabel[, fontdict, labelpad])</code>	Set the label for the y-axis.
<code>ylim(*args, **kwargs)</code>	Get or set the y-limits of the current axes.
<code>yscale(value, **kwargs)</code>	Set the y-axis scale.
<code>yticks([ticks, labels])</code>	Get or set the current tick locations and labels of the y-axis.

We will only use a very small subset of these functions in the examples.

26.4 Example 2 - Enhanced trigonometric plot

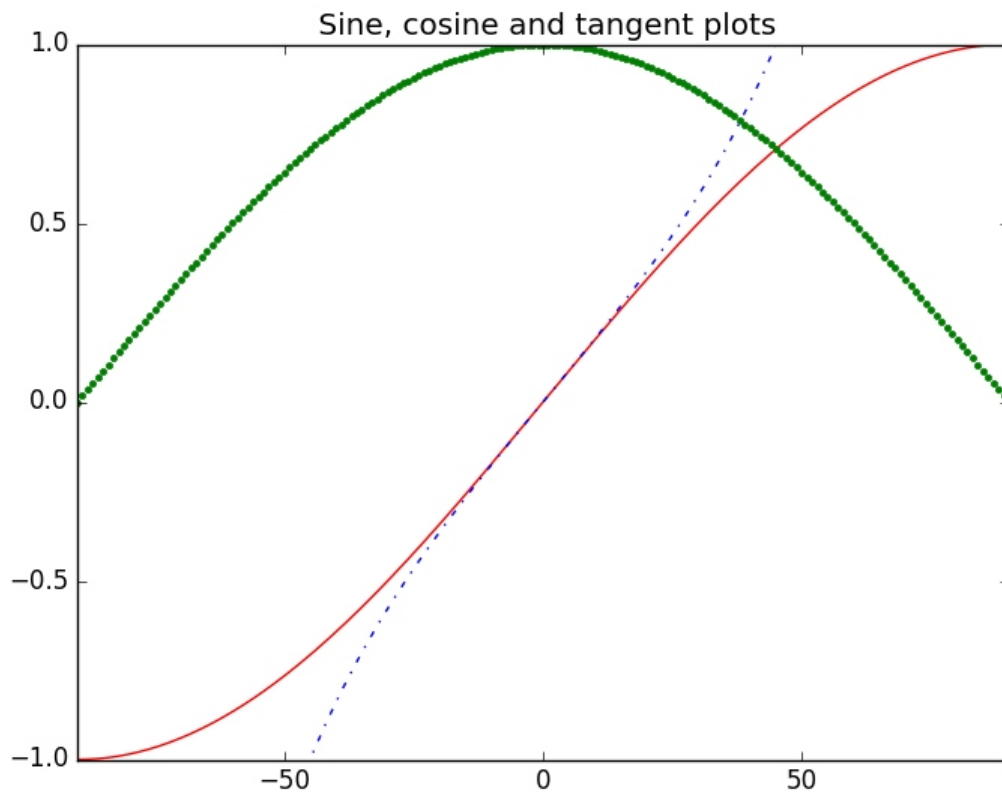
Here is the source code.

```

from pylab import *
import numpy as np
import math
size=181
x=0.0
xaxis = np.empty([size], dtype=int32)
y1      = np.empty([size], dtype=float64)
y2      = np.empty([size], dtype=float64)
y3      = np.empty([size], dtype=float64)
for i in range(size):
    xaxis[i]=(i-90)
    x=(i-90)*math.pi/180.0
    y1[i] = math.sin(x)
    y2[i] = math.cos(x)
    y3[i] = math.tan(x)
axis([-90, 90, -1, 1])
title(' Sine, cosine and tangent plots')
plot(xaxis, y1, 'r-')
plot(xaxis, y2, 'g.')
plot(xaxis, y3, 'b-.')
show()

```

We have added a tangent plot, title and relabelled the axis. Here is the output.



We use this example in the next couple of examples.

26.5 Example 3 - adding a legend, matplotlib defaults

Here is the program.

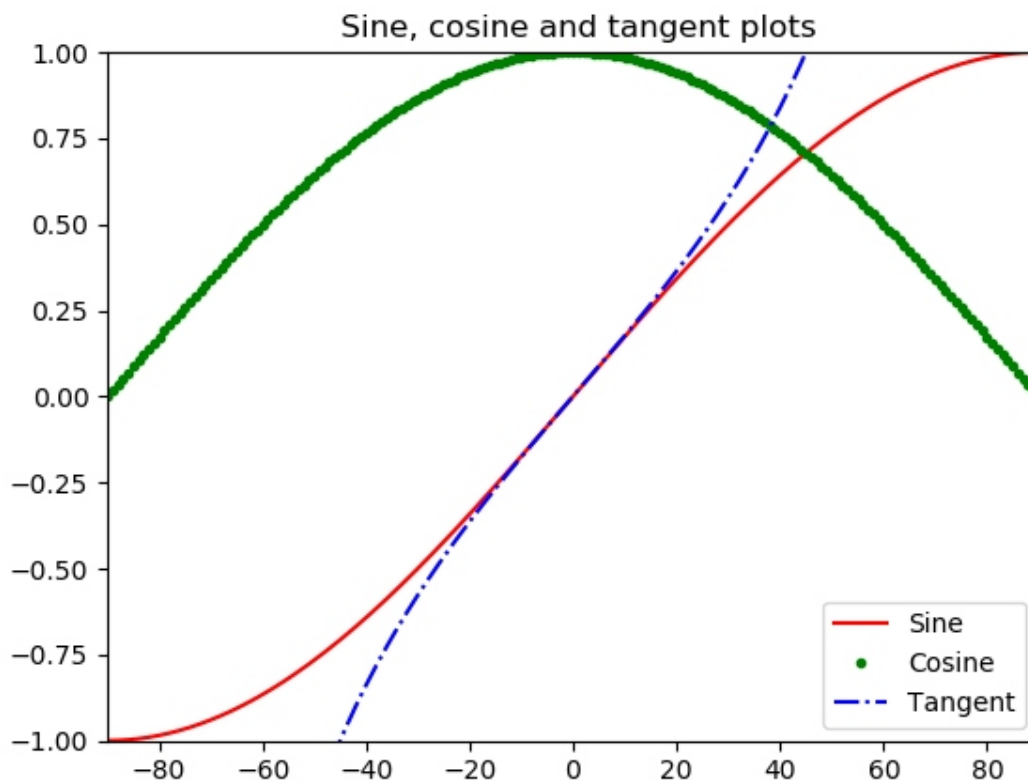
```
from pylab import *
import numpy as np
import math
size=181
x=0.0
xaxis = np.empty([size],dtype=int32)
y1     = np.empty([size],dtype=float64)
y2     = np.empty([size],dtype=float64)
y3     = np.empty([size],dtype=float64)
for i in range(size):
    xaxis[i]=(i-90)
    x=(i-90)*math.pi/180.0
    y1[i] = math.sin(x)
    y2[i] = math.cos(x)
    y3[i] = math.tan(x)
axis([-90,90,-1,1])
title(' Sine, cosine and tangent plots')
plot(xaxis,y1,'r-')
```

```

plot(xaxis,y2,'g.')
plot(xaxis,y3,'b-.')
legend(['Sine','Cosine','Tangent'])
show()

```

Here is the plot.



Python finds a space to put the plot. The positioning can be controlled using the matplotlib api. Here is a link to some of the documentation.

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html

Happy reading.

26.6 Example 4 - adding a legend with manual positioning

Here is the program.

```

from pylab import *
import numpy as np
import math
size=181
x=0.0
xaxis = np.empty([size],dtype=int32)
y1     = np.empty([size],dtype=float64)
y2     = np.empty([size],dtype=float64)
y3     = np.empty([size],dtype=float64)
for i in range(size):

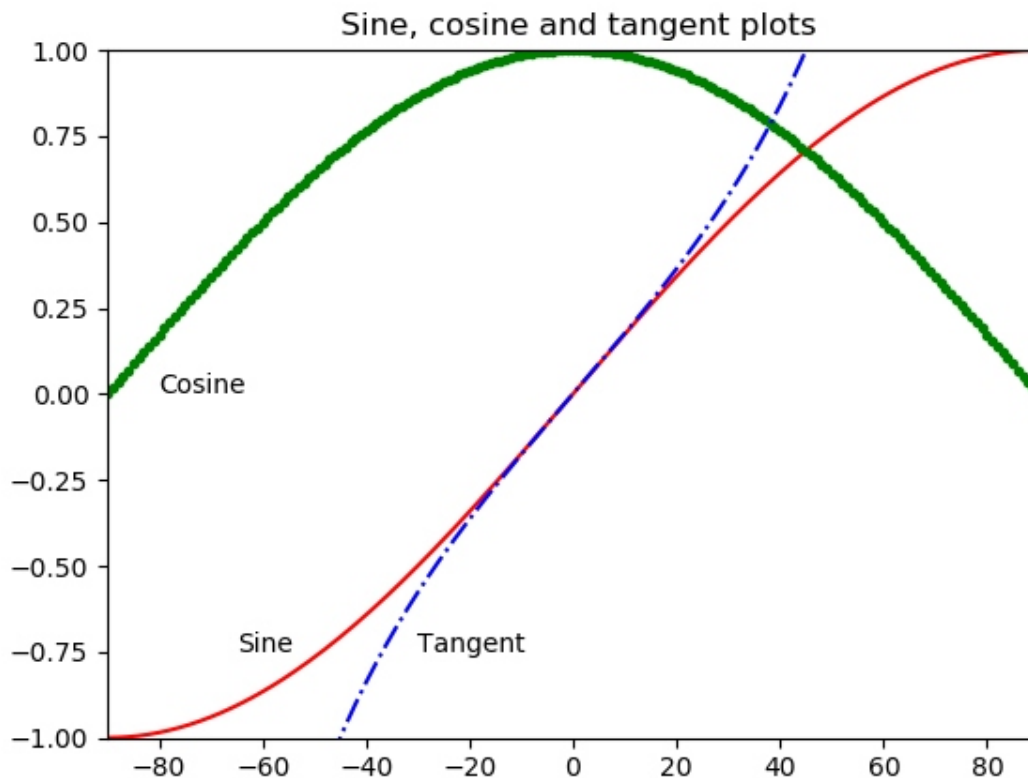
```

```

xaxis[i]=(i-90)
x=(i-90)*math.pi/180.0
y1[i] = math.sin(x)
y2[i] = math.cos(x)
y3[i] = math.tan(x)
axis([-90,90,-1,1])
title(' Sine, cosine and tangent plots')
plot(xaxis,y1,'r-')
plot(xaxis,y2,'g.')
plot(xaxis,y3,'b-.')
text(-65,-0.75,'Sine')
text(-80, 0.00,'Cosine')
text(-30,-0.75,'Tangent')
show()

```

Here is the plot.



The legends have been placed using the text component of the matplotlib api. The coordinates have been chosen with a bit of trial and error.

Longer text descriptions could be achieved by annotating the plots and making references to the actual text of the document.

26.7 Example 5 - Bar charts

Here are some bar chart examples. The data is taken from the following site:

<http://www.metoffice.gov.uk/public/weather/climate-historic/#?tab=climateHistoric>

A variety of programs are used to manipulate this data.

Programs to download the files from the Met Office site and save the files locally are written in C# and Java.

A sed script is used to convert the missing values (---) to a flag value (-999)

A Fortran program is used to produce summary calculations on the data

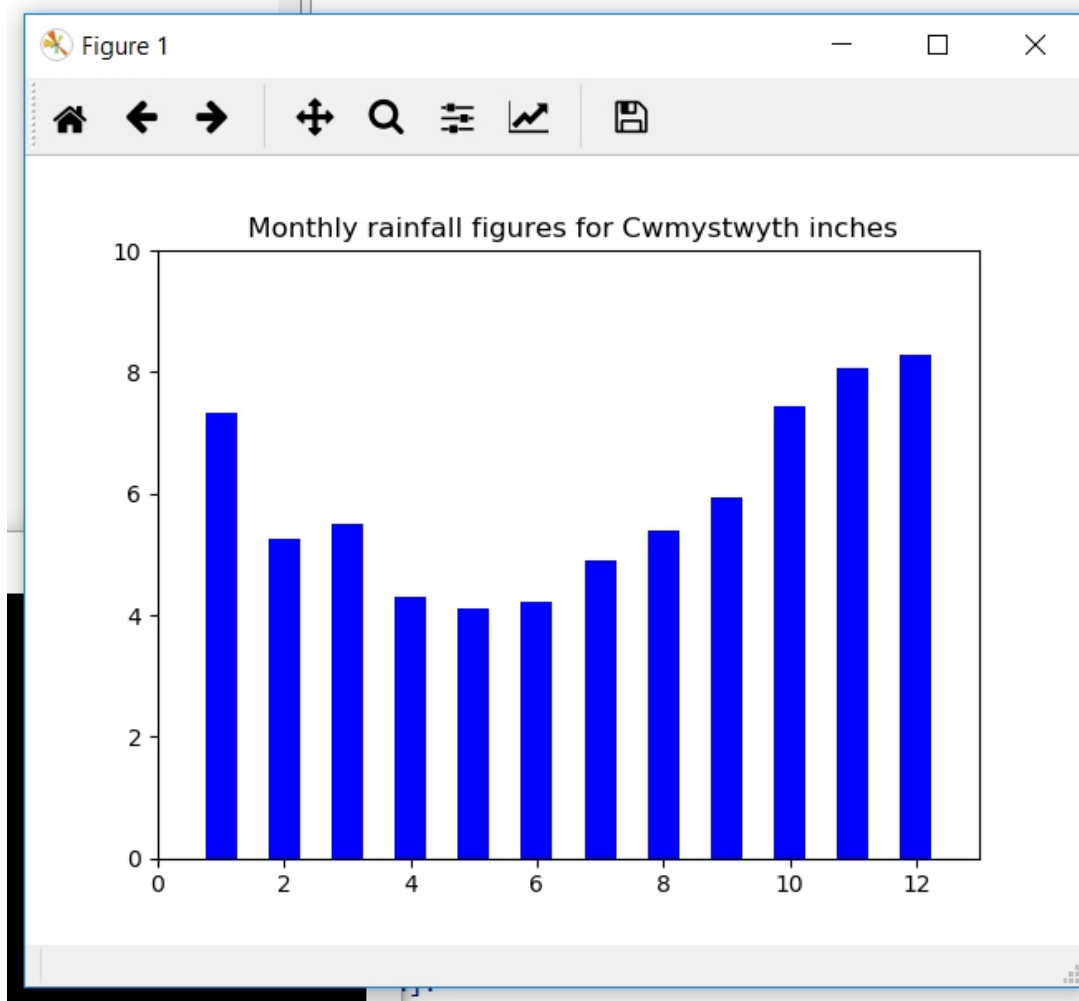
Python programs are used to do plots on the data

Use the best tool for the job!

Here is the source code for the first program.

```
from pylab import *
import numpy as np
nmonths=12
offset=0.5
xaxis = np.empty([nmonths], dtype=int32)
y1      = np.empty([nmonths], dtype=float64)
y2      = np.empty([nmonths], dtype=float64)
y1=[7.33,5.26,5.49,4.29,4.11,4.21,4.90,5.39,5.93,7.43,8.07,8.28]
for i in range(nmonths):
    xaxis[i]=i+1
    y2[i]=y1[i]*25.4
axis([1,12,0.0,10.0])
title(' Monthly rainfall figures for Cwmystwyth inches')
bar(xaxis,y1,offset,color='b')
#axis([0,13,0.0,200.0])
#title(' Monthly rainfall figures for Cwmystwyth mm')
#bar(xaxis+offset,y2,offset,color='r')
show()
```


Here is the plot.



26.8 Example 6 - bar chart with standard deviations

Here is the source code for the second program.

```

from pylab import *
import numpy as np
nmonths=12
offset1 = 0.4
offset2 = 0.5
offset3 = 0.6
xaxis = np.empty([nmonths],dtype=int32)
y1     = np.empty([nmonths],dtype=float64)
y2     = np.empty([nmonths],dtype=float64)
y1     = [ 7.33 ,  5.26 ,  5.49 ,  4.29 ,  4.11 ,  4.21
,  4.90 ,  5.39 ,  5.93 ,  7.43 ,  8.07 ,  8.28]
y2     = [186.10 ,133.67 ,139.50 ,109.02 ,104.46 ,106.92
,124.40 ,136.28 , 150.54 ,188.76 ,204.96 ,210.28]
std_dev = [ 78.01 , 77.36 , 70.51 , 51.33 , 49.12 , 52.13 ,
59.28 , 55.72 , 55.72 , 61.84 , 85.96 , 90.57]
for i in range(nmonths):
    xaxis[i]=i+1
axis([1,13,0.0,325.0])

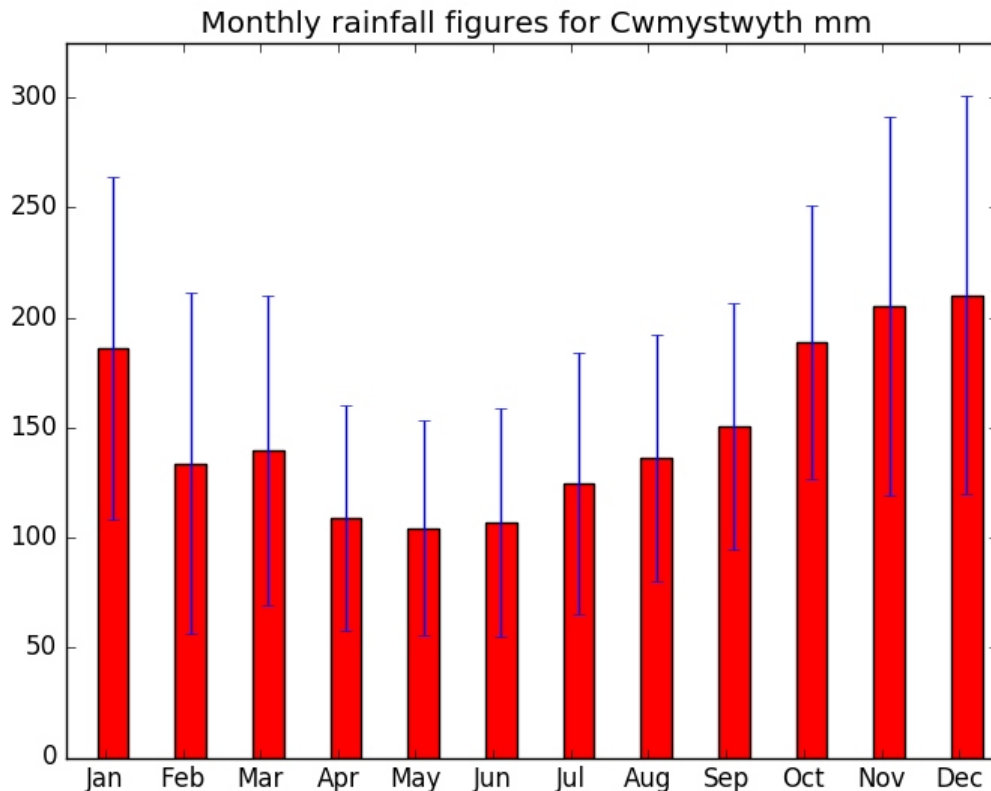
```

```

title(' Monthly rainfall figures for Cwmystwyth mm')
bar(xaxis+offset1,y2,offset1,color='r',yerr=std_dev)
xticks(xaxis+offset2,['Jan','Feb','Mar','Apr','May','Jun','Jul',
', 'Aug','Sep','Oct','Nov','Dec'])
show()

```

Here is the plot.



There is obviously quite a wide variation in monthly rainfall over this 50 year period.

26.9 Example 7 - bar chart with 4 frequencies

Here is the source code.

```

from pylab import *
import numpy as np
n_intervals= 10
offset      = 10

#xaxis = np.empty([n_intervals],dtype=int32)
#xaxis = [10,20,30,40,50,60,70,80,90,100]
#axis([0,110,0,50])

```

```
labels =
('2010','2011','2012','2013','2014','2015','2016','2017','2019
','2019')
y_pos = np.arange(len(labels))

y1 = np.empty([n_intervals],dtype=int32)
y1 = [ 184, 186, 180, 179, 180, 176, 176, 154, 160, 147]

y2 = np.empty([n_intervals],dtype=int32)
y2 = [ 0, 164, 169, 162, 166, 166, 176, 141, 145, 140]

y3 = np.empty([n_intervals],dtype=int32)
y3 = [ 0, 20, 17, 18, 13, 4, 0, 36, 11,
20]

y4 = np.empty([n_intervals],dtype=int32)
y4 = [ 0, 22, 11, 17, 14, 10, 9, 13, 18,
8]

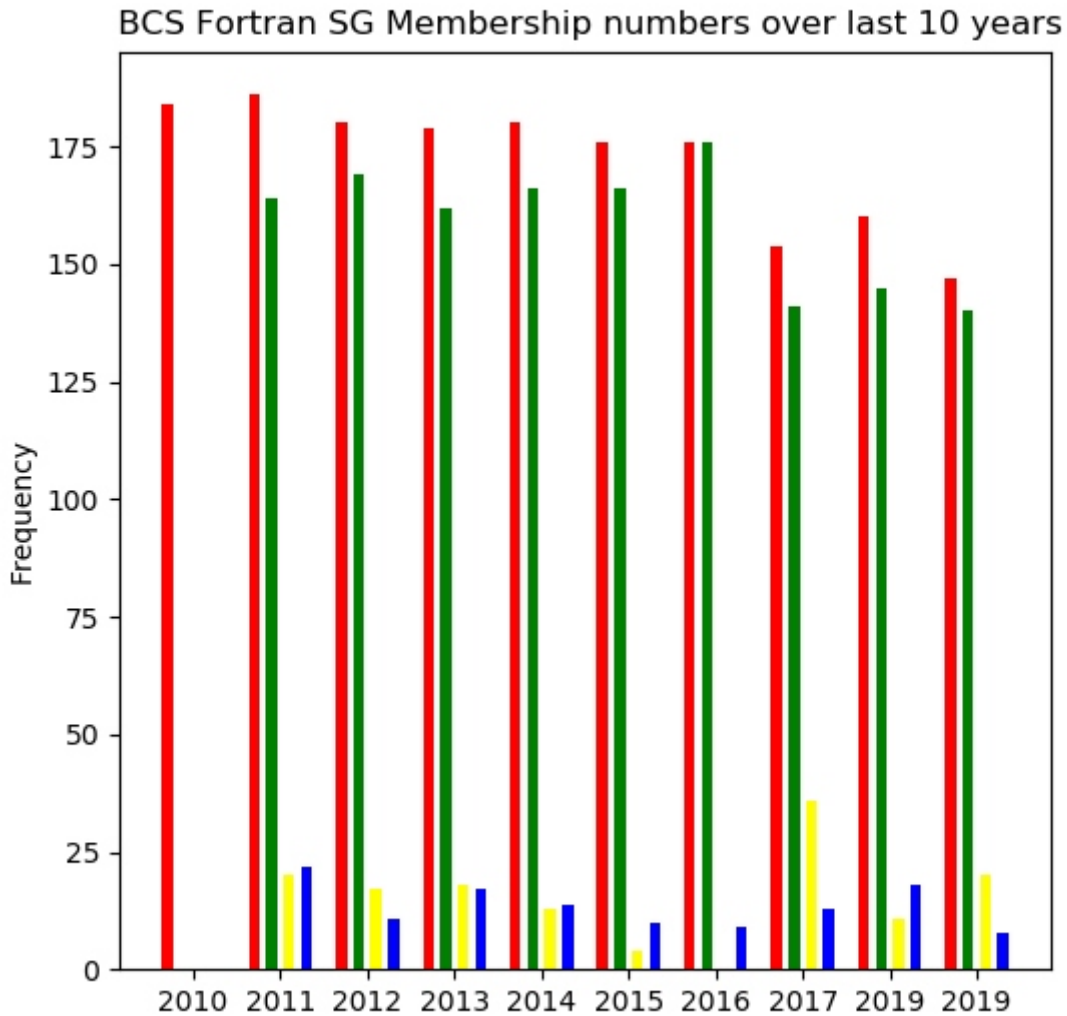
plt.figure(figsize=(6,6))

bar(y_pos-0.30,y1,align='center',color='red',width=0.125)
bar(y_pos-0.10,y2,align='center',color='green',width=0.125)
bar(y_pos+0.10,y3,align='center',color='yellow',width=0.125)
bar(y_pos+0.30,y4,align='center',color='blue',width=0.125)

plt.xticks(y_pos, labels)
plt.ylabel('Frequency')
title(' BCS Fortran SG Membership numbers over last 10
years')

show()
```

Here is the plot.



The meaning of the colours are given below:

red - total member numbers in August

green - common numbers between last year and this year

yellow - the number who left

blue - the number who joined

The captions would have difficult to add to the plot.

26.10 Example 8 - bar chart with 10 frequencies

Here is the source code.

```
from pylab import *
import numpy as np
n_intervals= 8
offset      = 10

#xaxis = np.empty([n_intervals],dtype=int32)
#xaxis = [10,20,30,40,50,60,70,80,90,100]
#axis([0,110,0,50])

labels =
('20-29','30-39','40-49','50-59','60-69','70-79','80-89','90-9
9')
y_pos  = np.arange(len(labels))

# 2010

y0     = np.empty([n_intervals],dtype=int32)
y0     = [ 7, 23, 42, 36, 46, 15, 0, 0]

# 2011

y1     = np.empty([n_intervals],dtype=int32)
y1     = [ 7, 18, 46, 55, 40, 18, 1, 0]

# 2012

y2     = np.empty([n_intervals],dtype=int32)
y2     = [ 8, 15, 42, 49, 43, 21, 1, 0]

# 2013

y3     = np.empty([n_intervals],dtype=int32)
y3     = [ 4, 17, 38, 51, 45, 20, 2, 1]

# 2014

y4     = np.empty([n_intervals],dtype=int32)
y4     = [ 3, 16, 32, 50, 51, 23, 2, 1]

# 2015

y5     = np.empty([n_intervals],dtype=int32)
y5     = [ 4, 11, 32, 50, 50, 25, 2, 1]
```

```

# 2016

y6      = np.empty([n_intervals],dtype=int32)
y6      = [ 3, 9,31,49,53,26, 3, 1]
# 2017

y7      = np.empty([n_intervals],dtype=int32)
y7      = [ 2,10,25,37,48,25, 6, 1]

# 2018

y8      = np.empty([n_intervals],dtype=int32)
y8      = [ 4,10,27,37,44,28, 8, 1]

# 2019

y9      = np.empty([n_intervals],dtype=int32)
y9      = [ 1, 6, 22, 38, 43, 27, 9, 0]

plt.figure(figsize=(6,6))

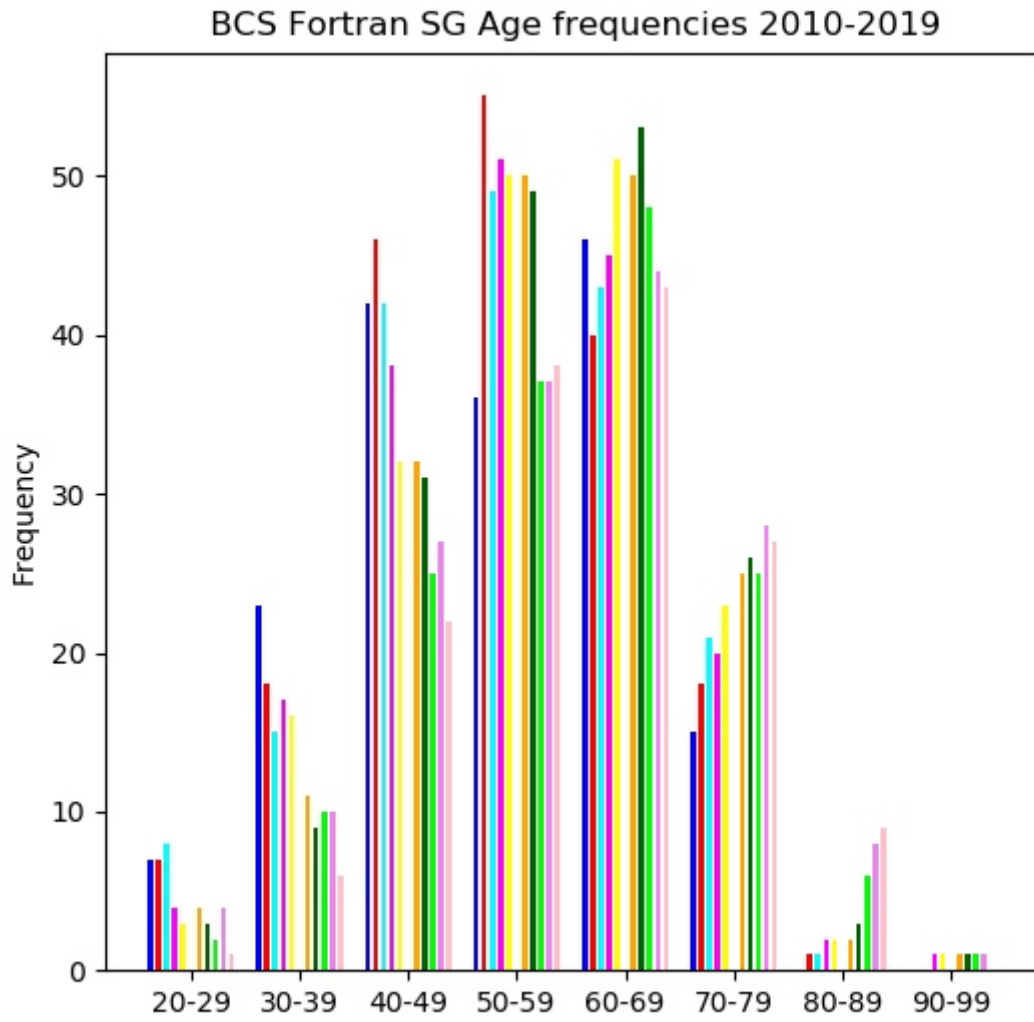
bar(y_pos-0.375,y0,align='center',color='blue',width=0.05)
bar(y_pos-0.300,y1,align='center',color='red',width=0.05)
bar(y_pos-0.225,y2,align='center',color='cyan',width=0.05)
bar(y_pos-0.150,y3,align='center',color='magenta',width=0.05)
bar(y_pos-0.075,y4,align='center',color='yellow',width=0.05)
bar(y_pos+0.075,y5,align='center',color='orange',width=0.05)
bar(y_pos+0.150,y6,align='cen-
ter',color='darkgreen',width=0.05)
bar(y_pos+0.225,y7,align='center',color='lime',width=0.05)
bar(y_pos+0.300,y8,align='center',color='violet',width=0.05)
bar(y_pos+0.375,y9,align='center',color='pink',width=0.05)

plt.xticks(y_pos, labels)
plt.ylabel('Frequency')
title(' BCS Fortran SG Age frequencies 2010-2019')

show()

```

Here is the plot.

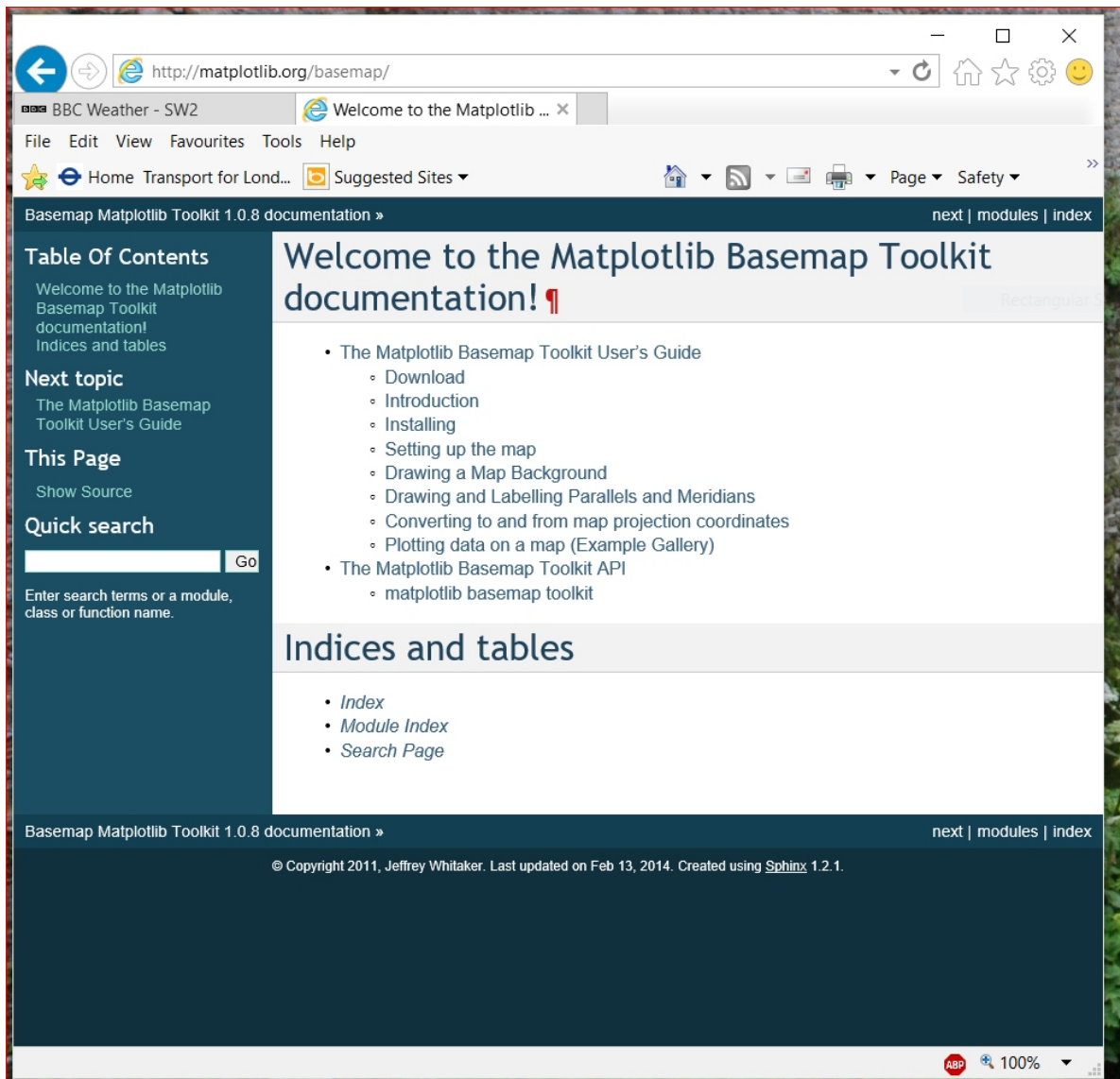


26.11 Example 9 - Mapping with Python 2.x and basemap

Visit

<http://matplotlib.org/basemap/>

for some basic information. You may have some problems running this example, due to import issues, and dependencies between versions of Python and the various imported modules.



We are going to create a map of tsunami events. I did the original plot whilst on a United Nations Environment Programme secondment. Section 9 of their Environmental Reports cover natural disasters, and these include

- Earthquakes
- Volcanoes
- Tsunamis
- Floods
- Landslide
- Natural Dams

Droughts

Wildfires

The bibliography has details of the publications I worked on. The map plots have been done on both Windows and openSuSe.

On a Windows platform you need to download and install the Python 2.7 version as basemap works with this version, but not with the Python 3.5 version.

On the openSuSE system I did the following as root

downloaded the Python 3.5 version anaconda version;

installed anaconda doing bash download.sh file;

chose /opt/anaconda3 as the install directory;

you must add the above directory to your path to be able to run the software;

On both platforms you will need to run

```
conda install basemap
```

from a console to install the additional mapping software. You can then run jupyter qtconsole and then running the python source file will generate the plots.

Here is the source file.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
```

```
# tsunami data file is called
```

```
# tsunami.txt
```

```
# There are 3033 entries
```

```
# region size
```

```
# 0          378
```

```
# 1          206
```

```
# 2           41
```

```
# 3           54
```

```
# 4           60
```

```
# 5         1540
```

```
# 6           80
```

```
# 7          144
```

```
# 8          245
```

```
# 9          285
```

```
#
```

```
# 1x,f7.2,2x,f7.2
```

```
#
```

```
# i need 9 * 2 arrays
```

```
#
```

```
tsunami_file = "tsunami.txt"
```

```
reg0 = 378
```

```
reg1 = 206
```

```
reg2 = 41
```

```
reg3 = 54
```

```

reg4 = 60
reg5 = 1540
reg6 = 80
reg7 = 144
reg8 = 245
reg9 = 285
lat0 = np.empty([reg0] , dtype=np.float64)
lon0 = np.empty([reg0] , dtype=np.float64)
lat1 = np.empty([reg1] , dtype=np.float64)
lon1 = np.empty([reg1] , dtype=np.float64)
lat2 = np.empty([reg2] , dtype=np.float64)
lon2 = np.empty([reg2] , dtype=np.float64)
lat3 = np.empty([reg3] , dtype=np.float64)
lon3 = np.empty([reg3] , dtype=np.float64)
lat4 = np.empty([reg4] , dtype=np.float64)
lon4 = np.empty([reg4] , dtype=np.float64)
lat5 = np.empty([reg5] , dtype=np.float64)
lon5 = np.empty([reg5] , dtype=np.float64)
lat6 = np.empty([reg6] , dtype=np.float64)
lon6 = np.empty([reg6] , dtype=np.float64)
lat7 = np.empty([reg7] , dtype=np.float64)
lon7 = np.empty([reg7] , dtype=np.float64)
lat8 = np.empty([reg8] , dtype=np.float64)
lon8 = np.empty([reg8] , dtype=np.float64)
lat9 = np.empty([reg9] , dtype=np.float64)
lon9 = np.empty([reg9] , dtype=np.float64)

```

```
f=open(tsunami_file)
```

```

for i in range(0,reg0):
    line=f.readline()
    lat0[i]=(float)(line[1:7])
    lon0[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat0[i])
# print " {0:7.2f} ".format(lon0[i])

```

```

for i in range(0,reg1):
    line=f.readline()
    lat1[i]=(float)(line[1:7])
    lon1[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat1[i])

# print " {0:7.2f} ".format(lon1[i])

```

```

for i in range(0,reg2):
    line=f.readline()
    lat2[i]=(float)(line[1:7])
    lon2[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat2[i])

```

```
# print " {0:7.2f} ".format(lon2[i])

for i in range(0,reg3):
    line=f.readline()
    lat3[i]=(float)(line[1:7])
    lon3[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat3[i])
# print " {0:7.2f} ".format(lon3[i])

for i in range(0,reg4):
    line=f.readline()
    lat4[i]=(float)(line[1:7])
    lon4[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat4[i])
# print " {0:7.2f} ".format(lon4[i])

for i in range(0,reg5):
    line=f.readline()
    lat5[i]=(float)(line[1:7])
    lon5[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat5[i])
# print " {0:7.2f} ".format(lon5[i])

for i in range(0,reg6):
    line=f.readline()
    lat6[i]=(float)(line[1:7])
    lon6[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat6[i])
# print " {0:7.2f} ".format(lon6[i])

for i in range(0,reg7):
    line=f.readline()
    lat7[i]=(float)(line[1:7])
    lon7[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat7[i])
# print " {0:7.2f} ".format(lon7[i])

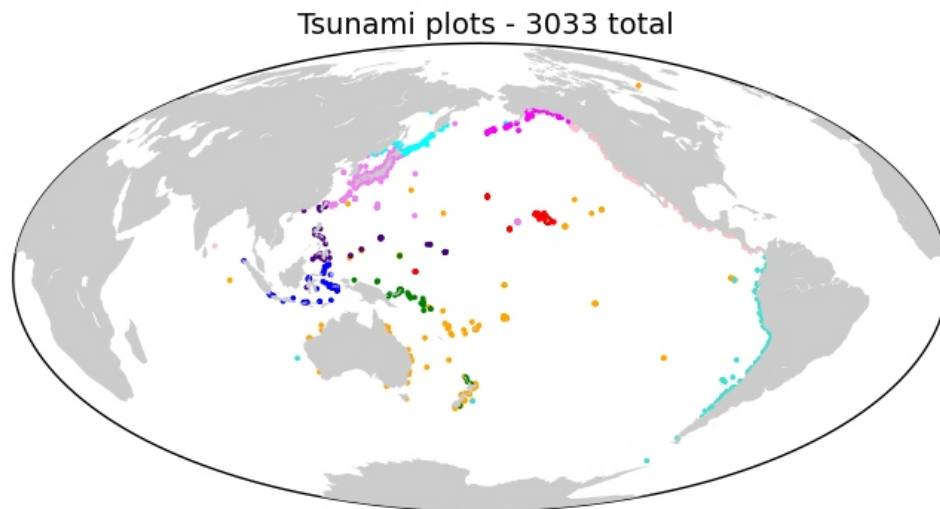
for i in range(0,reg8):
    line=f.readline()
    lat8[i]=(float)(line[1:7])
    lon8[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat8[i])
# print " {0:7.2f} ".format(lon8[i])

for i in range(0,reg9):
    line=f.readline()
    lat9[i]=(float)(line[1:7])
    lon9[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat9[i])
```

```
# print " {0:7.2f} ".format(lon9[i])

# draw map with markers for float locations
m = Basemap(projection='hammer',lon_0=180)
m.drawmapboundary()
m.fillcontinents()
x, y = m(lon0,lat0)
m.scatter(x,y,3,marker='o',color='red')
x, y = m(lon1,lat1)
m.scatter(x,y,3,marker='o',color='orange')
x, y = m(lon2,lat2)
m.scatter(x,y,3,marker='o',color='green')
x, y = m(lon3,lat3)
m.scatter(x,y,3,marker='o',color='blue')
x, y = m(lon4,lat4)
m.scatter(x,y,3,marker='o',color='indigo')
x, y = m(lon5,lat5)
m.scatter(x,y,3,marker='o',color='violet')
x, y = m(lon6,lat6)
m.scatter(x,y,3,marker='o',color='cyan')
x, y = m(lon7,lat7)
m.scatter(x,y,3,marker='o',color='magenta')
x, y = m(lon8,lat8)
m.scatter(x,y,3,marker='o',color='pink')
x, y = m(lon9,lat9)
m.scatter(x,y,3,marker='o',color='turquoise')
plt.title(' Tsunami plots - 3033 total',fontsize=14)
plt.show()
```

Here is the plot.



The plot file is in png format. You can save in a variety of formats.

26.12 Mapping with Python 3 and Cartopy

Cartopy is a modern mapping package that works with Python 3.x Here is their home page.

<http://scitools.org.uk/cartopy/index.htm>

Here is some information taken from their pages.

Introduction

Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.

Cartopy makes use of the powerful PROJ.4, numpy and shapely libraries and includes a programmatic interface built on top of Matplotlib for the creation of publication quality maps.

Key features of cartopy are its object oriented projection definitions, and its ability to transform points, lines, vectors, polygons and images between those projections.

You will find cartopy especially useful for large area / small scale data, where Cartesian assumptions of spherical data traditionally break down. If you've ever experienced a singularity at the pole or a cut-off at the dateline, it is likely you will appreciate cartopy's unique features!

Getting started - The installation guide provides information on getting up and running. Cartopy's documentation is arranged in user guide form, with reference documentation available inline.

Coordinate reference systems in Cartopy

Cartopy projection list

Using cartopy with matplotlib

The cartopy Feature interface

Understanding the transform and projection keywords

Using the cartopy shapereader

Cartopy developer interfaces

The outline link found above the cartopy logo on all pages can be used to quickly find the reference documentation for known classes or functions.

For those updating from an older version of cartopy, the what's new page outlines recent changes, new features, and future development plans.

Getting involved

Cartopy was originally developed at the UK Met Office to allow scientists to visualise their data on maps quickly, easily and most importantly, accurately. Cartopy has been made freely available under the terms of the GNU Lesser General Public License. It is suitable to be used in a variety of scientific fields and has an active development community.

26.12.1 Example 10 - tsunami plot using cartopy

Here is a rewrite of the tsunami example to use cartopy, rather than basemap. You need to run the following

```
conda install -c conda-forge cartopy
```

after the Anaconda install. You need to run this as administrator.

Here is the output on one system from running this command.

```
conda install -c conda-forge cartopy
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==
current version: 4.4.10
latest version: 4.6.2
```

Please update conda by running

```
$ conda update -n base conda
```

```
## Package Plan ##
```

```
environment location: C:\ProgramData\Anaconda3
```

```
added / updated specs:
```

```
- cartopy
```

```
The following packages will be downloaded:
```

package		build
-----		-----
pykdtree-1.3.1		py36h452e1ab_1002
54 KB conda-forge		
pyproj-1.9.6		py36h1fcc0e4_1000
237 KB conda-forge		
numpy-1.14.2		py36h5c71026_0
3.7 MB		
blas-1.0		mkl
6 KB		
matplotlib-2.2.2		py36_1
6.5 MB conda-forge		
shapely-1.6.4		py36hc90234e_1000
379 KB conda-forge		
owslib-0.17.1		py_0
118 KB conda-forge		
openssl-1.0.2p		hfa6e2cd_1002
5.4 MB conda-forge		
kiwisolver-1.0.1		py36he980bc4_1002
60 KB conda-forge		
ca-certificates-2018.11.29		ha4d7672_0
179 KB conda-forge		
proj4-5.2.0		hfa6e2cd_1001
3.4 MB conda-forge		
cartopy-0.17.0		py36h2ddefc_1001
2.1 MB conda-forge		
certifi-2018.11.29		py36_1000
145 KB conda-forge		

```
-----  
Total:
```

```
22.2 MB
```

```
The following NEW packages will be INSTALLED:
```

```
blas: 1.0-mkl
```

```
kiwisolver:          1.0.1-py36he980bc4_1002 conda-forge
pykdtree:           1.3.1-py36h452e1ab_1002 conda-forge
```

The following packages will be UPDATED:

```
ca-certificates: 2017.08.26-h94faf87_0
--> 2018.11.29-ha4d7672_0 conda-forge
cartopy:         0.16.0-py36_0 conda-forge
--> 0.17.0-py36h2ddefc_1001 conda-forge
certifi:        2018.1.18-py36_0 conda-forge
--> 2018.11.29-py36_1000 conda-forge
matplotlib:     2.1.2-py36h016c42a_0
--> 2.2.2-py36_1 conda-forge
numpy:          1.14.0-py36h4a99626_1
--> 1.14.2-py36h5c71026_0
openssl:        1.0.2n-h74b6da3_0
--> 1.0.2p-hfa6e2cd_1002 conda-forge
owslib:         0.16.0-py_0 conda-forge
--> 0.17.1-py_0 conda-forge
proj4:          4.9.3-vc14_5 conda-forge
--> 5.2.0-hfa6e2cd_1001 conda-forge
pyproj:         1.9.5.1-py36_0 conda-forge
--> 1.9.6-py36h1fcc0e4_1000 conda-forge
shapely:        1.6.4-py36_0 conda-forge
--> 1.6.4-py36hc90234e_1000 conda-forge
```

Proceed ([y]/n)? y

Downloading and Extracting Packages

pykdtree 1.3.1:

```
#####
##### | 100%
```

pyproj 1.9.6:

```
#####
##### | 100%
```

numpy 1.14.2:

```
#####
##### | 100%
```

blas 1.0:

```
#####
##### | 100%
```

matplotlib 2.2.2:

```
#####
##### | 100%
```

shapely 1.6.4:

```
#####
##### | 100%
```



```

owslib 0.17.1:
#####
##### | 100%
openssl 1.0.2p:
#####
##### | 100%
kiwisolver 1.0.1:
#####
##### | 100%
ca-certificates 2018.11.29:
#####
##### | 100%
proj4 5.2.0:
#####
##### | 100%
cartopy 0.17.0:
#####
##### | 100%
certifi 2018.11.29:
#####
##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

```

Here is the source code for this version.

```

import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

# tsunami data file is called
# tsunami.txt
# There are 3033 entries
# region size
# 0          378
# 1          206
# 2           41
# 3           54
# 4           60
# 5         1540
# 6           80
# 7          144
# 8          245
# 9          285
#
# 1x, f7.2, 2x, f7.2
#
# i need 9 * 2 arrays
#

```

```
tsunami_file = "tsunami.txt"

reg0 = 378
reg1 = 206
reg2 = 41
reg3 = 54
reg4 = 60
reg5 = 1540
reg6 = 80
reg7 = 144
reg8 = 245
reg9 = 285

lat0 = np.empty([reg0] , dtype=np.float64)
lon0 = np.empty([reg0] , dtype=np.float64)
lat1 = np.empty([reg1] , dtype=np.float64)
lon1 = np.empty([reg1] , dtype=np.float64)
lat2 = np.empty([reg2] , dtype=np.float64)
lon2 = np.empty([reg2] , dtype=np.float64)
lat3 = np.empty([reg3] , dtype=np.float64)
lon3 = np.empty([reg3] , dtype=np.float64)
lat4 = np.empty([reg4] , dtype=np.float64)
lon4 = np.empty([reg4] , dtype=np.float64)
lat5 = np.empty([reg5] , dtype=np.float64)
lon5 = np.empty([reg5] , dtype=np.float64)
lat6 = np.empty([reg6] , dtype=np.float64)
lon6 = np.empty([reg6] , dtype=np.float64)
lat7 = np.empty([reg7] , dtype=np.float64)
lon7 = np.empty([reg7] , dtype=np.float64)
lat8 = np.empty([reg8] , dtype=np.float64)
lon8 = np.empty([reg8] , dtype=np.float64)
lat9 = np.empty([reg9] , dtype=np.float64)
lon9 = np.empty([reg9] , dtype=np.float64)

f=open(tsunami_file)

for i in range(0,reg0):
    line=f.readline()
    lat0[i]=(float)(line[1:7])
    lon0[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat0[i])
# print " {0:7.2f} ".format(lon0[i])

for i in range(0,reg1):
    line=f.readline()
    lat1[i]=(float)(line[1:7])
    lon1[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat1[i])
```

```
# print " {0:7.2f} ".format(lon1[i])

for i in range(0,reg2):
    line=f.readline()
    lat2[i]=(float)(line[1:7])
    lon2[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat2[i])
# print " {0:7.2f} ".format(lon2[i])

for i in range(0,reg3):
    line=f.readline()
    lat3[i]=(float)(line[1:7])
    lon3[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat3[i])
# print " {0:7.2f} ".format(lon3[i])

for i in range(0,reg4):
    line=f.readline()
    lat4[i]=(float)(line[1:7])
    lon4[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat4[i])
# print " {0:7.2f} ".format(lon4[i])

for i in range(0,reg5):
    line=f.readline()
    lat5[i]=(float)(line[1:7])
    lon5[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat5[i])
# print " {0:7.2f} ".format(lon5[i])

for i in range(0,reg6):
    line=f.readline()
    lat6[i]=(float)(line[1:7])
    lon6[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat6[i])
# print " {0:7.2f} ".format(lon6[i])

for i in range(0,reg7):
    line=f.readline()
    lat7[i]=(float)(line[1:7])
    lon7[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat7[i])
# print " {0:7.2f} ".format(lon7[i])

for i in range(0,reg8):
    line=f.readline()
    lat8[i]=(float)(line[1:7])
    lon8[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat8[i])
```

```
# print " {0:7.2f} ".format(lon8[i])

for i in range(0,reg9):
    line=f.readline()
    lat9[i]=(float)(line[1:7])
    lon9[i]=(float)(line[10:16])
# print " {0:7.2f} ".format(lat9[i])
# print " {0:7.2f} ".format(lon9[i])

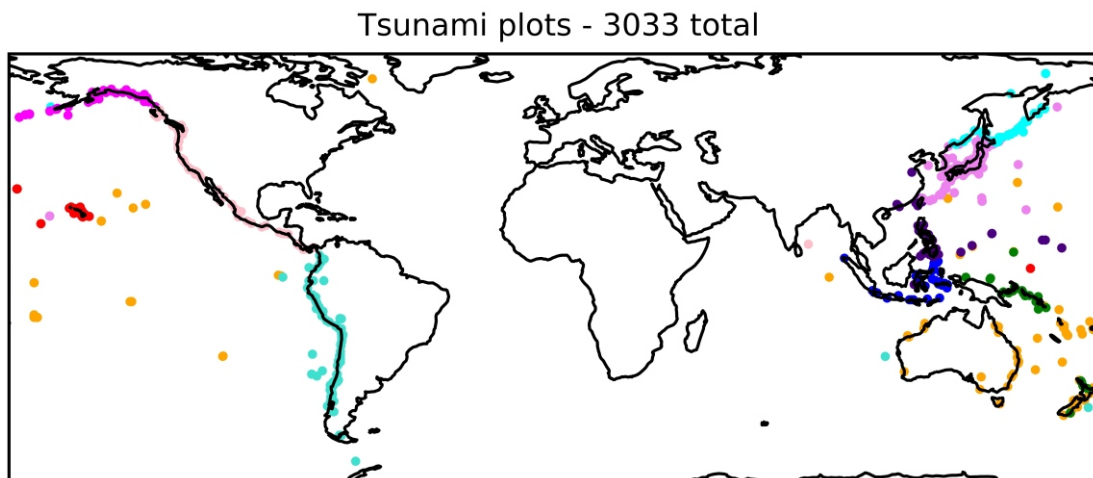
plt.figure(figsize=(20, 24))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()

x, y = lon0,lat0

plt.scatter(x,y,3,marker='o',color='red')
x, y = lon1,lat1
plt.scatter(x,y,3,marker='o',color='orange')
x, y = lon2,lat2
plt.scatter(x,y,3,marker='o',color='green')
x, y = lon3,lat3
plt.scatter(x,y,3,marker='o',color='blue')
x, y = lon4,lat4
plt.scatter(x,y,3,marker='o',color='indigo')
x, y = lon5,lat5
plt.scatter(x,y,3,marker='o',color='violet')
x, y = lon6,lat6
plt.scatter(x,y,3,marker='o',color='cyan')
x, y = lon7,lat7
plt.scatter(x,y,3,marker='o',color='magenta')
x, y = lon8,lat8
plt.scatter(x,y,3,marker='o',color='pink')
x, y = lon9,lat9
plt.scatter(x,y,3,marker='o',color='turquoise')

plt.title(' Tsunami plots - 3033 total',fontsize=14)
plt.show()
```

Here is a sample plot.



The map plot is similar to the basemap example.

26.12.2 Example 11 - shifting the center of the map

This is a simple variant of the previous with the centre of the map now based on the international dateline. Here is the source.

```
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

# tsunami data file is called
# tsunami.txt
# There are 3033 entries
# region size
# 0          378
# 1          206
# 2           41
# 3           54
# 4           60
# 5         1540
# 6           80
# 7          144
# 8          245
# 9          285
#
# 1x, f7.2, 2x, f7.2
#
# i need 9 * 2 arrays
#
```

```
tsunami_file = "tsunami.txt"

reg0 = 378
reg1 = 206
reg2 = 41
reg3 = 54
reg4 = 60
reg5 = 1540
reg6 = 80
reg7 = 144
reg8 = 245
reg9 = 285

lat0 = np.empty([reg0] , dtype=np.float64)
lon0 = np.empty([reg0] , dtype=np.float64)
lat1 = np.empty([reg1] , dtype=np.float64)
lon1 = np.empty([reg1] , dtype=np.float64)
lat2 = np.empty([reg2] , dtype=np.float64)
lon2 = np.empty([reg2] , dtype=np.float64)
lat3 = np.empty([reg3] , dtype=np.float64)
lon3 = np.empty([reg3] , dtype=np.float64)
lat4 = np.empty([reg4] , dtype=np.float64)
lon4 = np.empty([reg4] , dtype=np.float64)
lat5 = np.empty([reg5] , dtype=np.float64)
lon5 = np.empty([reg5] , dtype=np.float64)
lat6 = np.empty([reg6] , dtype=np.float64)
lon6 = np.empty([reg6] , dtype=np.float64)
lat7 = np.empty([reg7] , dtype=np.float64)
lon7 = np.empty([reg7] , dtype=np.float64)
lat8 = np.empty([reg8] , dtype=np.float64)
lon8 = np.empty([reg8] , dtype=np.float64)
lat9 = np.empty([reg9] , dtype=np.float64)
lon9 = np.empty([reg9] , dtype=np.float64)

f=open(tsunami_file)

for i in range(0,reg0):
    line=f.readline()
    lat0[i]=(float)(line[1:7])
    lon0[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat0[i])
# print " {0:7.2f} ".format(lon0[i])

for i in range(0,reg1):
    line=f.readline()
    lat1[i]=(float)(line[1:7])
    lon1[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat1[i])

# print " {0:7.2f} ".format(lon1[i])
```

```
for i in range(0,reg2):
    line=f.readline()
    lat2[i]=(float)(line[1:7])
    lon2[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat2[i])
# print " {0:7.2f} ".format(lon2[i])

for i in range(0,reg3):
    line=f.readline()
    lat3[i]=(float)(line[1:7])
    lon3[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat3[i])
# print " {0:7.2f} ".format(lon3[i])

for i in range(0,reg4):
    line=f.readline()
    lat4[i]=(float)(line[1:7])
    lon4[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat4[i])
# print " {0:7.2f} ".format(lon4[i])

for i in range(0,reg5):
    line=f.readline()
    lat5[i]=(float)(line[1:7])
    lon5[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat5[i])
# print " {0:7.2f} ".format(lon5[i])

for i in range(0,reg6):
    line=f.readline()
    lat6[i]=(float)(line[1:7])
    lon6[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat6[i])
# print " {0:7.2f} ".format(lon6[i])

for i in range(0,reg7):
    line=f.readline()
    lat7[i]=(float)(line[1:7])
    lon7[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat7[i])
# print " {0:7.2f} ".format(lon7[i])

for i in range(0,reg8):
    line=f.readline()
    lat8[i]=(float)(line[1:7])
    lon8[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat8[i])
# print " {0:7.2f} ".format(lon8[i])
```

```

for i in range(0,reg9):
    line=f.readline()
    lat9[i]=(float)(line[1:7])
    lon9[i]=(float)(line[10:16])+180.00
# print " {0:7.2f} ".format(lat9[i])
# print " {0:7.2f} ".format(lon9[i])

# plt.figure(figsize=(6, 3))
# ax = plt.axes(projection=ccrs.PlateCarree(
#     central_longitude=180))
# ax.coastlines(resolution='110m')
# ax.gridlines()

plt.figure(figsize=(20, 24))
ax = plt.axes(projection=ccrs.PlateCarree(
    central_longitude=180))
# ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()

x, y = lon0,lat0
plt.scatter(x,y,3,marker='o',color='red')

x, y = lon1,lat1
plt.scatter(x,y,3,marker='o',color='orange')

x, y = lon2,lat2
plt.scatter(x,y,3,marker='o',color='green')

x, y = lon3,lat3
plt.scatter(x,y,3,marker='o',color='blue')

x, y = lon4,lat4
plt.scatter(x,y,3,marker='o',color='indigo')

x, y = lon5,lat5
plt.scatter(x,y,3,marker='o',color='violet')

x, y = lon6,lat6
plt.scatter(x,y,3,marker='o',color='cyan')

x, y = lon7,lat7
plt.scatter(x,y,3,marker='o',color='magenta')

x, y = lon8,lat8
plt.scatter(x,y,3,marker='o',color='pink')

x, y = lon9,lat9

```

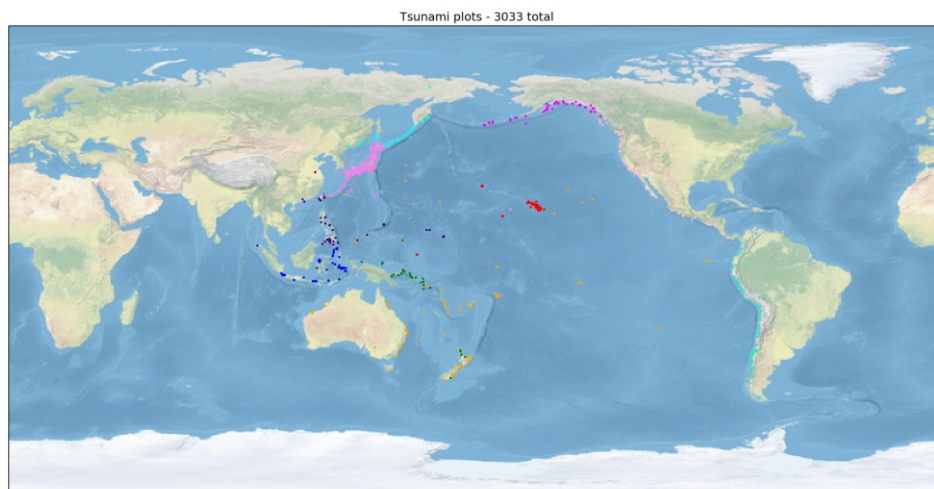


```
plt.scatter(x, y, 3, marker='o', color='turquoise')

plt.title(' Tsunami plots - 3033 total', fontsize=14)
plt.show()
```

Note that we have to shift the coordinates.

Here is the plot.



The plots were done with different versions of matplotlib and cartopy.

26.12.3 Example 12 - mapping using UK postcodes

In this example we look at plotting a membership map of the BCS Fortran Specialist Group. I am membership secretary.

The following site

<https://gridreferencefinder.com/postcodeBatchConverter/>
provides the ability to convert UK postcodes into UK mapping formats.

Here are 5 sample postcodes.

SM5 4JT
BN16 2QT
G77 5DR
SO50 4LQ
PA6 7NY

Here is the converted data.

Postcode	Description	Grid Reference	X (easting)	Y (northing)	Latitude	Longitude
SM5 4JT	SM5 4JT	TQ 27078 62759	527078	162759	51.34971	-0.17660918

BN16 2QT	BN16 2QT	TQ 04481 01558	504481	101558	50.804169	-0.51863235
G77 5DR	G77 5DR	NS 55211 56378	255211	656378	55.778979	-4.3101071
SO50 4LQ	SO50 4LQ	SU 45516 22106	445516	122106	50.996555	-1.3527812
PA6 7NY	PA6 7NY	NS 42045 66100	242045	666100	55.862124	-4.5254592

We convert the postcodes, and then edit the output to generate data files that we can read into the Python program to actually do the mapping.

Here is a sample of the data input file for the Python program.

```
51.34971 , -0.17661
50.80417 , -0.51863
55.77898 , -4.31011
50.99669 , -1.35290
55.86234 , -4.52512
```

Here is the program source.

```
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

# ax.gridlines()

# BCS Fortran member dat
# Two sets of data
#
# I need 2 * 2 arrays
#

data_file = "bcs_2018.txt"

reg0 = 56
reg1 = 87

lat0 = np.empty([reg0] , dtype=np.float64)
lon0 = np.empty([reg0] , dtype=np.float64)

lat1 = np.empty([reg1] , dtype=np.float64)
lon1 = np.empty([reg1] , dtype=np.float64)

f=open(data_file)

for i in range(0,reg0):
    line=f.readline()
    lat0[i]=(float)(line[0:7])
    lon0[i]=(float)(line[11:18])
# print(" {0:7.2f} ".format(lat0[i]))
# print(" {0:7.2f} ".format(lon0[i]))

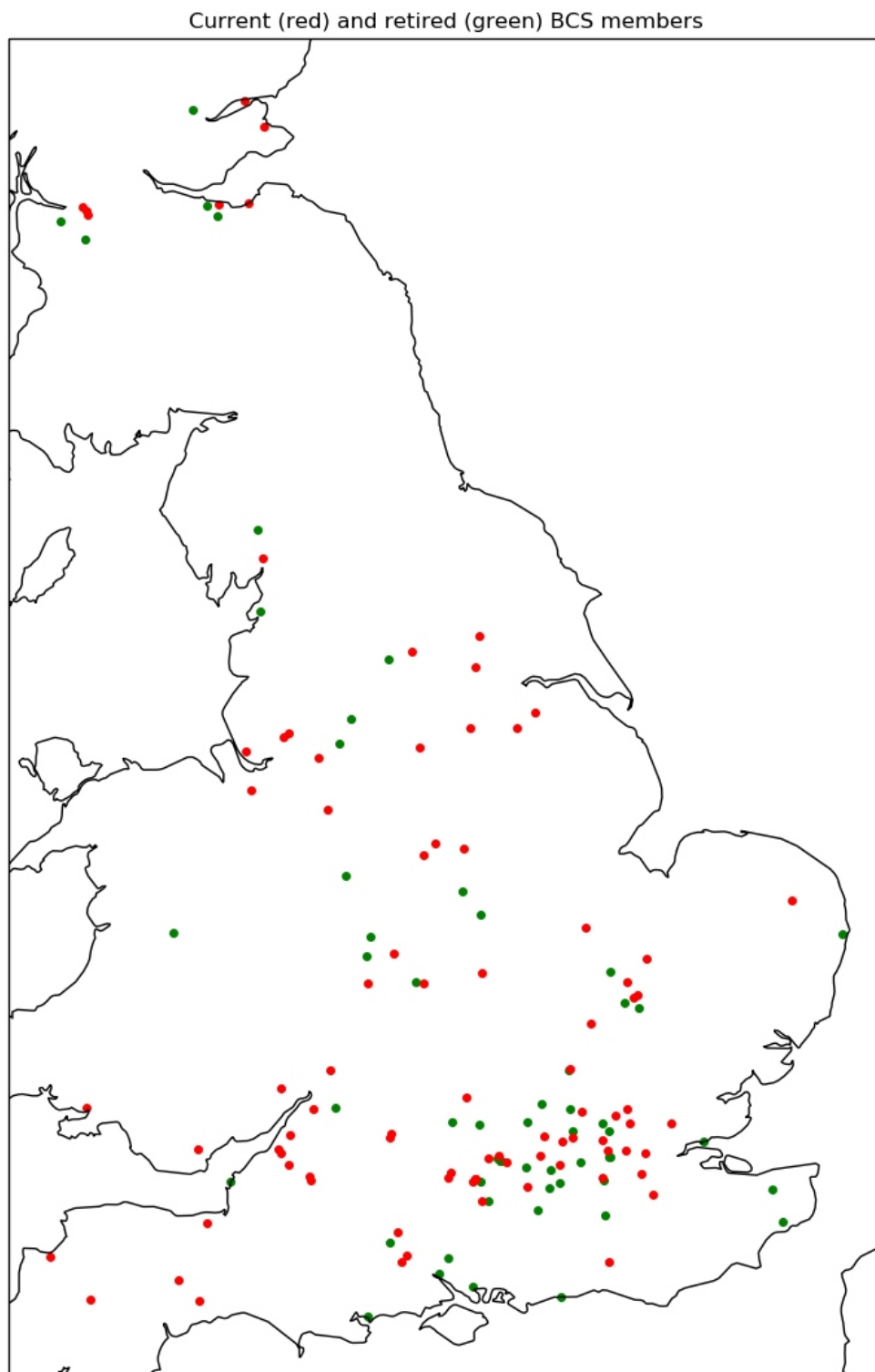
for i in range(0,reg1):
    line=f.readline()
    lat1[i]=(float)(line[0:7])
    lon1[i]=(float)(line[11:18])
```

```
# print " {0:7.2f} ".format(lat1[i])
# print " {0:7.2f} ".format(lon1[i])

x, y = lon0, lat0

plt.figure(figsize=(30,50))
ax = plt.axes(projection=ccrs.OSGB())
#ax.stock_img()
ax.set_title("Current (red) and retired (green) BCS members")
ax.coastlines(resolution='10m')
ax.scatter(x,y,16,marker='o',color='green',transform=ccrs.PlateCarree())
x, y = lon1, lat1
ax.scatter(x,y,16,marker='o',color='red' ,transform=ccrs.PlateCarree())
plt.show()
```

Here is the plot.



Note that the UK outline is just a skeleton.

26.13 Bibliography

26.13.1 Python

The on line documentation is good, especially the examples. Here are some other books and sources.

Python Data Analytics, Fabio Nelli, Apress. This has a chapter on matplotlib. The book is available as an ebook through Springer. As a Springer author I got a discount :-).

26.13.2 Cartopy

Cartopy, Met Office, Cartopy: A cartographic python library with a Matplotlib interface, 2010-2015, Exeter, Devon.

<http://scitools.org.uk/cartopy>.

26.13.3 Map data

Visit

<https://scitools.org.uk/cartopy/docs/latest/citation.html#data-copyright-table>

for complete information.

OpenStreetMap: Copyright OpenStreetMap

Natural Earth raster and vector map data: Made with Natural Earth.

26.13.4 UNEP

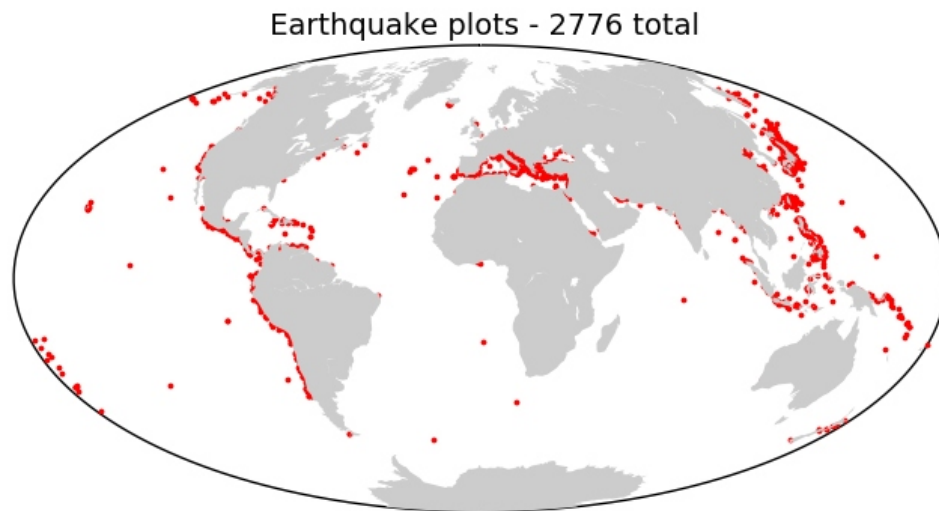
UNEP, Environmental Data Report, Second Edition, ISBN 0-631-16987-3, 1989.

UNEP, Environmental Data Report, Third Edition, ISBN 0-631-18083-4, 1991.

26.14 Problems

1. Run the examples in this chapter.

2. Using the earthquake data from an earlier chapter plot a map of earthquakes. In the first instance just do a plot of all of the earthquakes. Here is an example plot.



When you have successfully done this break down the plots by magnitude, using a different colour for each magnitude range.

If you feeling particularly adventurous do a plot by number of deaths.

27 Python performance versus other programming languages

27.1 Introduction

In this chapter we look at comparing the timing of Python against some other languages, including Fortran, C++ and Java.

27.2 Example 1 - Python solution

Here is the source.

```
import numpy as np
import time

def main():

    print(" Program starts          ",end=" ")
    print(time.ctime(),end=" ")
    t1=time.time()
    print(t1)
    n = 100000000

    x = np.empty([n],dtype=np.float64)
    t2=time.time()
    print(" Create empty array    ",end=" ")
    print(time.ctime(),end=" ")
    print(" {0:3.6f} ".format(t2-t1))
    t1=t2
    for i in range(0,n):
        x[i]=1.0

    t2=time.time()
    print(" Initalise the array ",end=" ")
    print(time.ctime(),end=" ")
    print(" {0:3.6f} ".format(t2-t1))
    t1=t2
    array_sum = sum(x)

    t2=time.time()
    print(" Sum          the array ",end=" ")
    print(time.ctime(),end=" ")
    print(" {0:3.6f} ".format(t2-t1))
    t1=t2
    print(" Sum = ",end=" ")
    print(array_sum)

    print(" Program ends          ",end=" ")
    print(time.ctime(),end=" ")
    t2=time.time()
    print(t2)
```

```
if ( __name__ == "__main__" ):
    main()
```

Timing details are at the end of the chapter.

27.3 Example 2 - Fortran solution

Here is the source.

```
module timing_module

    implicit none
    integer, dimension (8), private :: dt
    real, private :: h, m, s, ms, tt
    real, private :: last_tt

contains

    subroutine start_timing()
        implicit none

        call date_and_time(values=dt)
        print 100, dt(1:3), dt(5:8)
        h = real(dt(5))
        m = real(dt(6))
        s = real(dt(7))
        ms = real(dt(8))
        last_tt = 60*(60*h+m) + s + ms/1000.0
100 format (1x, i4, '/', i2, '/', i2, 1x, i2, ':', i2, ':',
i2, 1x, i3)
    end subroutine start_timing

    subroutine end_timing()
        implicit none

        call date_and_time(values=dt)
        print 100, dt(1:3), dt(5:8)
100 format (1x, i4, '/', i2, '/', i2, 1x, i2, ':', i2, ':',
i2, 1x, i3)
    end subroutine end_timing

    real function time_difference()
        implicit none

        tt = 0.0
        call date_and_time(values=dt)
        h = real(dt(5))
        m = real(dt(6))
        s = real(dt(7))
        ms = real(dt(8))
        tt = 60*(60*h+m) + s + ms/1000.0
```



```

        time_difference = tt - last_tt
        last_tt = tt
    end function time_difference

end module timing_module

module precision_module
    implicit none
    integer, parameter :: sp = selected_real_kind(6, 37)
    integer, parameter :: dp = selected_real_kind(15, 307)
    integer, parameter :: qp = selected_real_kind(30, 291)
end module precision_module

program array_sum
    use timing_module
    use precision_module, wp => dp
    implicit none
    integer , parameter :: n=100000000
    real (wp) ,dimension(n) :: x
    real (wp) :: x_sum
    call start_timing()
    x=1.0_wp
    print *, " array initialisation ",time_difference()
    x_sum=sum(x)
    print *, " array summation          ",time_difference()
    print *,x_sum
    call end_timing()
end program array_sum

```

27.4 Example 3 - C++ solution

Here is a C++ solution.

```

#include <iostream>
#include <cassert>
#include <chrono>
#include <string>
#include <cstdlib>

using namespace std;

class timer
{
public:

    timer() : start_timing(hi_res_clock::now()) {}
    void reset()
    {
        start_timing = hi_res_clock::now();
    }
}

```

```

double elapsed() const
{
    return(std::chrono::duration_cast<second_>
        (hi_res_clock::now() - start_timing).count());
}

private:

typedef std::chrono::high_resolution_clock hi_res_clock;
typedef std::chrono::duration<double, std::ratio<1> >
    second_;
std::chrono::time_point<hi_res_clock> start_timing;
};

void print_time(const string & heading , const double & t)
{
    cout << heading << " : " ;
    cout.setf(ios::right);
    cout.setf(ios::showpoint);
    cout.setf(ios::fixed);
    cout.width(10);
    cout.precision(6);
    cout << t << endl;
}

int main()
{

    string heading;
    double t;
    timer timer_01;
    timer timer_02;
    heading = "\n Program starts    ";
    t=timer_01.elapsed();
    print_time( heading , t);
    cout << endl;

    constexpr int n=100000000;
    int i ;
    double x_sum=0.0;
    double * x;

    heading = " Array allocation ";

    x = new double[n];

    for (i=0;i < n ; ++i)
    {
        x[i]=1.0;

```

```

}

t=timer_01.elapsed();
print_time(heading , t);
timer_01.reset();
heading = " Array summation ";

for (i=0;i < n ; ++i)
{
    x_sum += x[i];
}

t=timer_01.elapsed();
print_time(heading , t);
timer_01.reset();

cout << " \n Sum = " ;
cout.width(12);
cout.precision(2);
cout.setf(ios::right);
cout.setf(ios::showpoint);
cout.setf(ios::fixed);

cout << x_sum << endl;

t=timer_02.elapsed();
heading = "\n Total time ";
print_time(heading , t);

return(0);
}

```

27.5 Example 4 - Java solution

Here is the source and timing information.

```

import java.time.LocalDateTime;

class ch2704
{
    public static void main(String[] args)
    {
        System.out.print(" Program starts ");
        LocalDateTime t = LocalDateTime.now();
        System.out.println(t.toString());

        final int n=100000000;
        double x_sum=0.0;
    }
}

```

```

double[] x=new double[n];

System.out.print(" Allocation      ");
t = LocalDateTime.now();
System.out.println(t.toString());

int i ;
for (i=0;i < n ; ++i)
{
    x[i]=1.0;
}

System.out.print(" Assignment      ");
t = LocalDateTime.now();
System.out.println(t.toString());

for (i=0;i < n ; ++i)
{
    x_sum += x[i];
}

System.out.print(" Summation      ");
t = LocalDateTime.now();
System.out.println(t.toString());
System.out.println(x_sum);

}

}

```

27.6 Summary

Here is a summary of the timing. All runs were on the same system.

	Python	Fortran		C++		Java
		Nag	Intel	Microsoft	g++	
	3.6.5	6.2	19	17.x	7.3.0	1.8.0_131
Initialisation	12.314372	0.433094	0.589469	0.394777	0.420932	0.609000
Summation	14.283259	0.125000	0.132813	0.148464	0.158285	0.164000
Total	26.597631	0.558094	0.722282	0.543241	0.579217	0.773000
Percentage		2.10%	2.72%	2.04%	2.18%	2.91%

The percentage figures refer to the run time as a percentage of the Python run time. Python is not the quickest!

27.7 Problems

1. Run the examples in this chapter. What timing figures did you get?

28 Calling the Nag library from Python

28.1 Introduction

In this chapter we look at calling the Nag library from Python.

Here is the Nag base url.

```
https://www.nag.com/numeric/py/
nagdoc_latest/readme.html#installing-using-pip
```

Here is a summary of the installation steps on one of my systems.

```
python -m pip install --upgrade pip
python -m pip install msgpack
python -m pip install --extra-index-url
https://www.nag.com/downloads/py/naginterfaces_mkl
naginterfaces
```

Nag suggest using

```
python -m pydoc naginterfaces
```

to test the implementation. This produced the following output.

Help on package naginterfaces:

NAME

naginterfaces

DESCRIPTION

Package Summary

Python interfaces for the NAG Library Engine, which is the software implementation of NAG's collection of several hundred mathematical and statistical routines serving a diverse range of application areas.

Subpackage Summary

Interfaces to the NAG Library are provided in the :mod:`~naginterfaces.library` subpackage.

The ``base.utils`` submodule contains a number of core utilities for working with the supplied Library interfaces. See the documentation for the :mod:`~naginterfaces.base` subpackage for more information.

Some utilities for interacting with the NAG Kusari licence-management system are in a :mod:`~naginterfaces.kusari` submodule.

Submodules must be imported separately:

```
>>> from naginterfaces.library import opt, roots
```

```
-- More --
```

The next step is to get a licence. Running

```
python -m naginterfaces.kusari
```

brings up a gui form, which generates the information that Nag require to provide a key.

Batch files can be found on the Rhymney Consulting site that help with the install.

28.2 Example 1 - testing the Nag library calls

Here is a test program from the Nag documentation.

```
from naginterfaces.library.opt import bounds_bobyqa_func
rosen = lambda x: ( sum(100.0*(x[1:]-x[:-1])**2.0)**2.0 +
(1.0-x[:-1])**2.0 )
import numpy as np; x = np.array([1.2, 1.0, 1.2, 1.0])
n = len(x)
bl, bu = ([0.0]*n, [2.0]*n)
npt = 2*n + 1
rhobeg, rhoend = (1e-1, 1e-6)
maxcal = 500
x_min, f, nf = bounds_bobyqa_func
(rosen, npt, x, bl, bu, rhobeg, rhoend, maxcal,)
print('Function value at lowest point found is
{:.5f}.'.format(f))
print('The objective function was called
{:d} times.'.format(nf))
print('The corresponding x is
(' + ', '.join(['{:.4f}'] * n).format( *x_min ) + ').')
```

Here is the output.

```
(base) C:\document\python\examples>
python -m naginterfaces.kusari
```

```
(base) C:\document\python\examples>
(base) C:\document\python\examples>python nag_01.py
Function value at lowest point found is 0.00000.
The objective function was called 143 times.
The corresponding x is (1.0000, 1.0000, 1.0000, 1.0000).
```

which illustrates that the installation has worked.

A downloadable archive of the Library Manual to accompany this release can be found at

https://www.nag.com/numeric/fl/nagdoc_26.2/nagdoc_26.2.zip

We will be using the documentation in the course.

28.3 Example 2 - testing the Python random number generators

In this example we test the precision to which the random number generators work. Here is the source.

```
import random
import numpy as np
import time

def main():

    print(" Program starts",end=" ")
    print(time.ctime())

    print(type(random.random()))
    print(type(np.random.random()))

if ( __name__ == "__main__" ):
    main()
```

Here is the output from a Windows system.

```
Program starts Tue Oct 2 11:11:49 2018
<class 'float'>
<class 'float'>
```

Float is normally IEEE double.

28.4 Example 3 - Python native timing

In this example we look at timing random number generation and sorting with Python. Here is the source.

```
import random
import numpy as np
import time

def main():

    print(" Program starts",end=" ")
    print(time.ctime(),end=" ")
    t1=time.time()
    print(t1)

# set n to suit your system

n = 100000000
x = np.empty([n],dtype=np.float64)

print(type(x))
print(type(x[1]))
print( type( random.random() ) )

t2=time.time()
```



```

print(" Create empty array                ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

for i in range (0,n):
    x[i] = random.random()                # Random
float x, 0.0 <= x < 1.0

t2=time.time()
print(" Set array to random values        ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

for i in range(0,5):
    print(" {0:20.16f} ".format(x[i]))

temp = np.empty([n],dtype=np.float64)

t2=time.time()
print(" Create temporary array            ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

temp = np.sort(x)

t2=time.time()
print(" Numpy sort method                  ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

for i in range((n-5),n):
    print(" {0:20.16f} ".format(temp[i]))

if ( __name__ == "__main__" ):
    main()

```

Here is some sample output.

```

Program starts                Sat Oct  6 11:41:38 2018
1538822498.7798429
<class 'numpy.ndarray'>
<class 'numpy.float64'>
<class 'float'>
Create empty array            Sat Oct  6 11:41:38 2018
0.001497

```

```

Set array to random values      Sat Oct  6 11:42:02 2018
23.880204
    0.6783614879021483
    0.2778128395928219
    0.4947912341740682
    0.7813040366914789
    0.2953368493077114
Create temporary array          Sat Oct  6 11:42:02 2018
0.002001
Numpy sort method              Sat Oct  6 11:42:15 2018
12.380830
    0.9999999307082369
    0.9999999564000758
    0.9999999747624113
    0.9999999766293831
    0.999999998143773

```

In the next example we will look at calling the Nag library routines for generating the random numbers and sorting.

28.5 Example 4 - Nag timing

In this example we look at using routines from the Nag library. Here is the program source.

```

import random
import numpy as np
import time
from naginterfaces.library.sort import realvec_sort
from naginterfaces.library.rand import init_nonrepeat
from naginterfaces.library.rand import dist_uniform01

def main():

    print(" Program starts                ",end=" ")
    print(time.ctime(),end=" ")
    t1=time.time()
    print(t1)

# set n to suit your system

n = 100000000
x = np.empty([n],dtype=np.float64)

t2=time.time()
print(" Create empty array                ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

genid      = 1                # Nag basic genera-
tor
statecomm = init_nonrepeat(genid)

```

```

x = dist_uniform01(n, statecomm)

print( type( x      ) )
print( type( x[1] ) )

for i in range(0,5):
    print(" {0:20.16f} ".format(x[i]))

t2=time.time()
print(" Set array to random values      ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

temp = np.empty([n],dtype=np.float64)

t2=time.time()
print(" Create temporary array          ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

temp = np.sort(x)

t2=time.time()
print(" Numpy sort method                ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

for i in range((n-5),n):
    print(" {0:20.16f} ".format(temp[i]))

order = 'A'
temp = realvec_sort(x,1,order)

t2=time.time()
print(" Nag      sort method              ",end=" ")
print(time.ctime(),end=" ")
print(" {0:3.6f} ".format(t2-t1))
t1=t2

for i in range((n-5),n):
    print(" {0:20.16f} ".format(temp[i]))

if ( __name__ == "__main__" ):
    main()

```

Here is a sample run.

```

Program starts                               Sat Oct  6 11:44:29 2018
1538822669.644663
Create empty array                           Sat Oct  6 11:44:29 2018
0.002000
<class 'numpy.ndarray'>
<class 'numpy.float64'>
  0.5042501560547314
  0.9779543443423764
  0.1287328480622457
  0.9849552600053949
  0.2424051132344787
Set array to random values                   Sat Oct  6 11:44:31 2018
1.474478
Create temporary array                       Sat Oct  6 11:44:31 2018
0.002003
Numpy sort method                           Sat Oct  6 11:44:43 2018
12.350846
  0.9999999580795395
  0.9999999603659318
  0.9999999628400260
  0.9999999707114384
  0.9999999957634748
Nag sort method                             Sat Oct  6 11:44:56 2018
12.998827
  0.9999999580795395
  0.9999999603659318
  0.9999999628400260
  0.9999999707114384
  0.9999999957634748

```

Note the improvement with the Nag random number timing.

So this example shows how easy it is to call the Nag library from Python.

28.6 Problems

1. Run the examples in this chapter. What timing figures did you get?

29 Functional programming background

29.1 Introduction

In this chapter we provide a background to functional programming. The following information is taken from the Wikipedia entry.

https://en.wikipedia.org/wiki/Functional_programming

29.2 Background

In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time. Eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional programming has its roots in lambda calculus, a formal system developed in the 1930s to investigate computability, the Entscheidungsproblem, function definition, function application, and recursion. Many functional programming languages can be viewed as elaborations on the lambda calculus. Another well-known declarative programming paradigm, logic programming, is based on relations.[1]

In contrast, imperative programming changes state with commands in the source language, the most simple example being assignment. Imperative programming does have functions—not in the mathematical sense—but in the sense of subroutines. They can have side effects that may change the value of program state. Functions without return values therefore make sense. Because of this, they lack referential transparency, i.e. the same language expression can result in different values at different times depending on the state of the executing program.[1]

Functional programming languages, especially purely functional ones such as Hope and Rex, have largely been emphasized in academia rather than in commercial software development. However, prominent programming languages which support functional programming such as Common Lisp, Scheme, [2] [3] [4] [5] Clojure, [6] [7] Wolfram Language [8] (also known as Mathematica), Racket, [9] Erlang, [10] [11] [12] OCaml, [13] [14] Haskell, [15] [16] and F#[17] [18] have been used in industrial and commercial applications by a wide variety of organizations. Functional programming is also supported in some domain-specific programming languages like R (statistics), [19] J, K and Q from Kx Systems (financial analysis), XQuery/XSLT (XML), [20] [21] and Opal.[22] Widespread domain-specific declarative languages like SQL and Lex/Yacc use some elements of functional programming, especially in eschewing mutable values.[23]

Programming in a functional style can also be accomplished in languages that are not specifically designed for functional programming. For example, the imperative Perl programming language has been the subject of a book describing how to apply functional programming concepts.[24] This is also true of the PHP programming language.[25] C# 3.0 and Java 8 added constructs to facilitate the functional style. The Julia language also offers

functional programming abilities. An interesting case is that of Scala[26] – it is frequently written in a functional style, but the presence of side effects and mutable state place it in a grey area between imperative and functional languages.

29.3 History

Lambda calculus provides a theoretical framework for describing functions and their evaluation. Although it is a mathematical abstraction rather than a programming language, it forms the basis of almost all functional programming languages today. An equivalent theoretical formulation, combinatory logic, is commonly perceived as more abstract than lambda calculus and preceded it in invention. Combinatory logic and lambda calculus were both originally developed to achieve a clearer approach to the foundations of mathematics.[27]

An early functional-flavored language was Lisp, developed by John McCarthy while at Massachusetts Institute of Technology (MIT) for the IBM 700/7000 series scientific computers in the late 1950s.[28] Lisp introduced many features now found in functional languages, though Lisp is technically a multi-paradigm language. Scheme and Dylan were later attempts to simplify and improve Lisp.

Information Processing Language (IPL) is sometimes cited as the first computer-based functional programming language.[29] It is an assembly-style language for manipulating lists of symbols. It does have a notion of "generator", which amounts to a function accepting a function as an argument, and, since it is an assembly-level language, code can be used as data, so IPL can be regarded as having higher-order functions. However, it relies heavily on mutating list structure and similar imperative features.

Kenneth E. Iverson developed APL in the early 1960s, described in his 1962 book *A Programming Language* (ISBN 9780471430148). APL was the primary influence on John Backus's FP. In the early 1990s, Iverson and Roger Hui created J. In the mid-1990s, Arthur Whitney, who had previously worked with Iverson, created K, which is used commercially in financial industries along with its descendant Q.

John Backus presented FP in his 1977 Turing Award lecture "Can Programming Be Liberated From the von Neumann Style? A Functional Style and its Algebra of Programs".[30] He defines functional programs as being built up in a hierarchical way by means of "combining forms" that allow an "algebra of programs"; in modern language, this means that functional programs follow the principle of compositionality. Backus's paper popularized research into functional programming, though it emphasized function-level programming rather than the lambda-calculus style which has come to be associated with functional programming.

In the 1970s, ML was created by Robin Milner at the University of Edinburgh, and David Turner initially developed the language SASL at the University of St. Andrews and later the language Miranda at the University of Kent. Also in Edinburgh in the 1970s, Burstall and Darlington developed the functional language NPL.[31] NPL was based on Kleene Recursion Equations and was first introduced in their work on program transformation.[32] Burstall, MacQueen and Sannella then incorporated the polymorphic type checking from ML to produce the language Hope.[33] ML eventually developed into several dialects, the most common of which are now OCaml and Standard ML. Meanwhile, the development of Scheme (a partly functional dialect of Lisp), as described in the influential *Lambda Papers* and the 1985 textbook *Structure and Interpretation of Computer Programs*, brought awareness of the power of functional programming to the wider programming-languages community.

In the 1980s, Per Martin-Löf developed intuitionistic type theory (also called constructive type theory), which associated functional programs with constructive proofs of arbitrarily complex mathematical propositions expressed as dependent types. This led to powerful new approaches to interactive theorem proving and has influenced the development of many subsequent functional programming languages.

The Haskell language began with a consensus in 1987 to form an open standard for functional programming research; implementation releases have been ongoing since 1990.

29.4 Concepts

A number of concepts and paradigms are specific to functional programming, and generally foreign to imperative programming (including object-oriented programming). However, programming languages are often hybrids of several programming paradigms, so programmers using "mostly imperative" languages may have utilized some of these concepts.[34]

29.4.1 First-class and higher-order functions

Higher-order functions are functions that can either take other functions as arguments or return them as results. In calculus, an example of a higher-order function is the differential operator d/dx , which returns the derivative of a function f .

Higher-order functions are closely related to first-class functions in that higher-order functions and first-class functions both allow functions as arguments and results of other functions. The distinction between the two is subtle: "higher-order" describes a mathematical concept of functions that operate on other functions, while "first-class" is a computer science term that describes programming language entities that have no restriction on their use (thus first-class functions can appear anywhere in the program that other first-class entities like numbers can, including as arguments to other functions and as their return values).

Higher-order functions enable partial application or currying, a technique in which a function is applied to its arguments one at a time, with each application returning a new function that accepts the next argument. This allows one to succinctly express, for example, the successor function as the addition operator partially applied to the natural number one.

29.4.2 Pure functions

Purely functional functions (or expressions) have no side effects (memory or I/O). This means that pure functions have several useful properties, many of which can be used to optimize the code:

If the result of a pure expression is not used, it can be removed without affecting other expressions.

If a pure function is called with arguments that cause no side-effects, the result is constant with respect to that argument list (sometimes called referential transparency), i.e. if the pure function is again called with the same arguments, the same result will be returned (this can enable caching optimizations such as memoization).

If there is no data dependency between two pure expressions, then their order can be reversed, or they can be performed in parallel and they cannot interfere with one another (in other terms, the evaluation of any pure expression is thread-safe).

If the entire language does not allow side-effects, then any evaluation strategy can be used; this gives the compiler freedom to reorder or combine the evaluation of expressions in a program (for example, using deforestation).

While most compilers for imperative programming languages detect pure functions and perform common-subexpression elimination for pure function calls, they cannot always do this for pre-compiled libraries, which generally do not expose this information, thus preventing

optimizations that involve those external functions. Some compilers, such as gcc, add extra keywords for a programmer to explicitly mark external functions as pure, to enable such optimizations. Fortran 95 also allows functions to be designated "pure".

29.4.3 Recursion

Iteration (looping) in functional languages is usually accomplished via recursion. Recursive functions invoke themselves, allowing an operation to be performed over and over until the base case is reached. Though some recursion requires maintaining a stack, tail recursion can be recognized and optimized by a compiler into the same code used to implement iteration in imperative languages. The Scheme language standard requires implementations to recognize and optimize tail recursion. Tail recursion optimization can be implemented by transforming the program into continuation passing style during compiling, among other approaches.

Common patterns of recursion can be factored out using higher order functions, with catamorphisms and anamorphisms (or "folds" and "unfolds") being the most obvious examples. Such higher order functions play a role analogous to built-in control structures such as loops in imperative languages.

Most general purpose functional programming languages allow unrestricted recursion and are Turing complete, which makes the halting problem undecidable, can cause unsoundness of equational reasoning, and generally requires the introduction of inconsistency into the logic expressed by the language's type system. Some special purpose languages such as Coq allow only well-founded recursion and are strongly normalizing (nonterminating computations can be expressed only with infinite streams of values called codata). As a consequence, these languages fail to be Turing complete and expressing certain functions in them is impossible, but they can still express a wide class of interesting computations while avoiding the problems introduced by unrestricted recursion. Functional programming limited to well-founded recursion with a few other constraints is called total functional programming.[35]

29.4.4 Strict versus non-strict evaluation

Functional languages can be categorized by whether they use strict (eager) or non-strict (lazy) evaluation, concepts that refer to how function arguments are processed when an expression is being evaluated. The technical difference is in the denotational semantics of expressions containing failing or divergent computations. Under strict evaluation, the evaluation of any term containing a failing subterm will itself fail. For example, the expression:

```
print length([2+1, 3*2, 1/0, 5-4])
```

will fail under strict evaluation because of the division by zero in the third element of the list. Under lazy evaluation, the length function will return the value 4 (i.e., the number of items in the list), since evaluating it will not attempt to evaluate the terms making up the list. In brief, strict evaluation always fully evaluates function arguments before invoking the function. Lazy evaluation does not evaluate function arguments unless their values are required to evaluate the function call itself.

The usual implementation strategy for lazy evaluation in functional languages is graph reduction.[36] Lazy evaluation is used by default in several pure functional languages, including Miranda, Clean, and Haskell.

Hughes 1984 argues for lazy evaluation as a mechanism for improving program modularity through separation of concerns, by easing independent implementation of producers and consumers of data streams.[37] Launchbury 1993 describes some difficulties that lazy evaluation introduces, particularly in analyzing a program's storage requirements, and proposes

an operational semantics to aid in such analysis.[38] Harper 2009 proposes including both strict and lazy evaluation in the same language, using the language's type system to distinguish them.[39]

29.4.5 Type systems

Especially since the development of Hindley–Milner type inference in the 1970s, functional programming languages have tended to use typed lambda calculus, as opposed to the untyped lambda calculus used in Lisp and its variants (such as Scheme). The use of algebraic datatypes and pattern matching makes manipulation of complex data structures convenient and expressive; the presence of strong compile-time type checking makes programs more reliable, while type inference frees the programmer from the need to manually declare types to the compiler.

Some research-oriented functional languages such as Coq, Agda, Cayenne, and Epigram are based on intuitionistic type theory, which allows types to depend on terms. Such types are called dependent types. These type systems do not have decidable type inference and are difficult to understand and program with[citation needed]. But dependent types can express arbitrary propositions in predicate logic. Through the Curry–Howard isomorphism, then, well-typed programs in these languages become a means of writing formal mathematical proofs from which a compiler can generate certified code. While these languages are mainly of interest in academic research (including in formalized mathematics), they have begun to be used in engineering as well. Compcert is a compiler for a subset of the C programming language that is written in Coq and formally verified.[40]

A limited form of dependent types called generalized algebraic data types (GADT's) can be implemented in a way that provides some of the benefits of dependently typed programming while avoiding most of its inconvenience.[41] GADT's are available in the Glasgow Haskell Compiler, in OCaml (since version 4.00) and in Scala (as "case classes"), and have been proposed as additions to other languages including Java and C#[42]

29.4.6 Referential Transparency

Functional programs do not have assignment statements, that is, the value of a variable in a functional program never changes once defined. This eliminates any chances of side effects because any variable can be replaced with its actual value at any point of execution. So, functional programs are referentially transparent. [43]

Consider C assignment statement $x = x * 10$, this changes the value assigned to the variable x . Let us say that the initial value of x was 10, then two consecutive evaluations of the variable x will yield 10 and 100 respectively. Clearly, replacing $x = x * 10$ with either 10 or 100 gives a program with different meaning, and so the expression is not referentially transparent. In fact, assignment statements are never referentially transparent.

Now, consider another function such as `int plusone(int x) {return x+1;}` is transparent, as it will not implicitly change the input x and thus has no such side effects. Functional programs exclusively use this type of function and are therefore referentially transparent.

29.4.7 Functional programming in non-functional languages

It is possible to use a functional style of programming in languages that are not traditionally considered functional languages.[44] For example, both D and Fortran 95 explicitly support pure functions.[45]

JavaScript, Lua[46] and Python had first class functions from their inception.[47] Amrit Prem added support to Python for "lambda", "map", "reduce", and "filter" in 1994, as well as closures in Python 2.2, [48] though Python 3 relegated "reduce" to the functools standard

library module.[49] First-class functions have been introduced into other mainstream languages such as PHP 5.3, Visual Basic 9, C# 3.0, and C++11.[citation needed]

In Java, anonymous classes can sometimes be used to simulate closures; [50] however, anonymous classes are not always proper replacements to closures because they have more limited capabilities. [51] Java 8 supports lambda expressions as a replacement for some anonymous classes. [52] However, the presence of checked exceptions in Java can make functional programming inconvenient, because it can be necessary to catch checked exceptions and then rethrow them—a problem that does not occur in other JVM languages that do not have checked exceptions, such as Scala.[citation needed]

In C#, anonymous classes are not necessary, because closures and lambdas are fully supported. Libraries and language extensions for immutable data structures are being developed to aid programming in the functional style in C#.

Many object-oriented design patterns are expressible in functional programming terms: for example, the strategy pattern simply dictates use of a higher-order function, and the visitor pattern roughly corresponds to a catamorphism, or fold.

Similarly, the idea of immutable data from functional programming is often included in imperative programming languages, [53] for example the tuple in Python, which is an immutable array.

29.5 Comparison to imperative programming

Functional programming is very different from imperative programming. The most significant differences stem from the fact that functional programming avoids side effects, which are used in imperative programming to implement state and I/O. Pure functional programming completely prevents side-effects and provides referential transparency, which makes it easier to verify, optimize, and parallelize programs, and easier to write automated tools to perform those tasks.[citation needed]

Higher-order functions are rarely used in older imperative programming. A traditional imperative program might use a loop to traverse and modify a list. A functional program, on the other hand, would probably use a higher-order “map” function that takes a function and a list, generating and returning a new list by applying the function to each list item.

29.5.1 Simulating state

There are tasks (for example, maintaining a bank account balance) that often seem most naturally implemented with state. Pure functional programming performs these tasks, and I/O tasks such as accepting user input and printing to the screen, in a different way.

The pure functional programming language Haskell implements them using monads, derived from category theory. Monads offer a way to abstract certain types of computational patterns, including (but not limited to) modeling of computations with mutable state (and other side effects such as I/O) in an imperative manner without losing purity. While existing monads may be easy to apply in a program, given appropriate templates and examples, many students find them difficult to understand conceptually, e.g., when asked to define new monads (which is sometimes needed for certain types of libraries).[54]

Another way in which functional languages can simulate state is by passing around a data structure that represents the current state as a parameter to function calls. On each function call, a copy of this data structure is created with whatever differences are the result of the function. This is referred to as 'state-passing style'.

Impure functional languages usually include a more direct method of managing mutable state. Clojure, for example, uses managed references that can be updated by applying pure

functions to the current state. This kind of approach enables mutability while still promoting the use of pure functions as the preferred way to express computations.

Alternative methods such as Hoare logic and uniqueness have been developed to track side effects in programs. Some modern research languages use effect systems to make the presence of side effects explicit.

29.5.2 Efficiency issues

Functional programming languages are typically less efficient in their use of CPU and memory than imperative languages such as C and Pascal.[55] This is related to the fact that some mutable data structures like arrays have a very straightforward implementation using present hardware (which is a highly evolved Turing machine). Flat arrays may be accessed very efficiently with deeply pipelined CPUs, prefetched efficiently through caches (with no complex pointer-chasing), or handled with SIMD instructions. It is also not easy to create their equally efficient general-purpose immutable counterparts. For purely functional languages, the worst-case slowdown is logarithmic in the number of memory cells used, because mutable memory can be represented by a purely functional data structure with logarithmic access time (such as a balanced tree).[56] However, such slowdowns are not universal. For programs that perform intensive numerical computations, functional languages such as OCaml and Clean are only slightly slower than C.[57] For programs that handle large matrices and multidimensional databases, array functional languages (such as J and K) were designed with speed optimizations.

Immutability of data can in many cases lead to execution efficiency by allowing the compiler to make assumptions that are unsafe in an imperative language, thus increasing opportunities for inline expansion.[58]

Lazy evaluation may also speed up the program, even asymptotically, whereas it may slow it down at most by a constant factor (however, it may introduce memory leaks if used improperly). Launchbury 1993[38] discusses theoretical issues related to memory leaks from lazy evaluation, and O'Sullivan et al. 2008[59] give some practical advice for analyzing and fixing them. However, the most general implementations of lazy evaluation making extensive use of dereferenced code and data perform poorly on modern processors with deep pipelines and multi-level caches (where a cache miss may cost hundreds of cycles)[citation needed].

29.5.3 Coding styles

Imperative programs tend to emphasize the series of steps taken by a program in carrying out an action, while functional programs tend to emphasize the composition and arrangement of functions, often without specifying explicit steps. A simple example illustrates this with two solutions to the same programming goal (calculating Fibonacci numbers). The imperative example is in Python. Working versions of these Python programs can be found in the functions chapter.

29.5.3.1 Version 1 – With Generators

```
# Fibonacci numbers, imperative style
# https://docs.python.org/2.7/tutorial/modules.html
def fibonacci(n, first=0, second=1):
    for i in range(n):
        yield first # Return current iteration
        first, second = second, first + second
```

```
print [x for x in fibonacci(10)]
```

29.5.3.2 Version 2 – Iterative

```
def fibonacci(n):
    first, second = 0, 1
    for i in range(n):
        print first # Print current iteration
        first, second = second, first + second #Calculate next values
```

```
fibonacci(10)
```

29.5.3.3 Version 3 – Recursive

```
def fibonacci(n, first=0, second=1):
    if n == 1:
        return [first]
    else:
        return [first] + fibonacci(n - 1, second, first + second)
```

```
print fibonacci(10)
```

29.5.3.4 Haskell

A functional version (in Haskell) has a different feel to it[vague]:

```
-- Fibonacci numbers, functional style
-- describe an infinite list based on the recurrence relation for Fibonacci numbers
fibRecurrence first second = first : fibRecurrence second (first + second)
-- describe fibonacci list as fibRecurrence with initial values 0 and 1
fibonacci = fibRecurrence 0 1
-- describe action to print the 10th element of the fibonacci list
main = print (fibonacci !! 10)
```

Or, more concisely:

```
fibonacci2 = 0:1:zipWith (+) fibonacci2 (tail fibonacci2)
```

The imperative style describes the intermediate steps involved in calculating fibonacci(N), and places those steps inside a loop statement. In contrast, the functional implementation shown here states the mathematical recurrence relation that defines the entire Fibonacci sequence, then selects an element from the sequence (see also recursion). This example relies on Haskell's lazy evaluation to create an "infinite" list of which only as much as needed (the first 10 elements in this case) will actually be computed. That computation happens when the runtime system carries out the action described by "main".

29.5.3.5 Erlang

The same program in Erlang provides a simple example of how functional languages in general do not require their syntax to contain an "if" statement.

```
-module(fibonacci).
-export([start/1]).
```

```
%% Fibonacci numbers in Erlang
```

```
start(N) -> do_fib(0, 1, N).
do_fib(_, B, 1) -> B;
do_fib(A, B, N) -> do_fib(B, A + B, N - 1).
```

This program is contained within a module called "fibonacci" and declares that the start/1 function will be visible from outside the scope of this module.

The function start/1 accepts a single parameter (as denoted by the "/1" syntax) and then calls an internal function called do_fib/3.

In direct contrast to the imperative coding style, Erlang does not need an "if" statement because the Erlang runtime will examine the parameters being passed to a function, and call the first function having a signature that matches the current pattern of parameters. (Erlang syntax does provide an "if" statement, but it is considered syntactic sugar and, compared to its usage in imperative languages, plays only a minor role in application logic design).

In this case, it is unnecessary to test for a parameter value within the body of the function because such a test is implicitly performed by providing a set of function signatures that describe the different patterns of values that could be received by a function.

In the case above, the first version of do_fib/3 will only be called when the third parameter has the precise value of 1. In all other cases, the second version of do_fib/3 will be called.

This example demonstrates that functional programming languages often implement conditional logic implicitly by matching parameter patterns rather than explicitly by means of an "if" statement.

29.5.3.6 Elixir

Elixir is a functional, concurrent, general-purpose programming language that runs on the Erlang virtual machine (BEAM).

The Fibonacci function can be written in Elixir as follows:

```
defmodule Fibonacci do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n), do: fib(n-1) + fib(n-2)
end
```

29.5.3.7 Lisp

The Fibonacci function can be written in Common Lisp as follows:

```
(defun fib (n &optional (a 0) (b 1))
  (if (= n 0)
      a
      (fib (- n 1) b (+ a b))))
```

The program can then be called as

```
(fib 10)
```

29.5.3.8 D

D has support for functional programming[clarification needed] [citation needed]:

```
import std.stdio;
import std.range;
```

```
void main()
```

```

{
  /* 'f' is a range representing the first 10 Fibonacci numbers */
  auto f = recurrence!((seq, i) => seq[0] + seq[1])(0, 1)
    .take(10);
  writeln(f);
}

```

29.5.3.9 R

R (programming language) is an environment for statistical computing and graphics. It is also a functional programming language.

The Fibonacci function can be written in R as a recursive function as follows:

```

fib <- function(n) {
  if (n <= 2) 1
  else fib(n - 1) + fib(n - 2)
}

```

Or it can be written as a singly recursive function:

```

fib <- function(n,a=1,b=1) {
  if (n == 1) a
  else fib(n-1,b,a+b)
}

```

Or it can be written as an iterative function:

```

fib <- function(n) {
  if (n == 1) 1
  else if (n == 2) 1
  else {
    fib<-c(1,1)
    for (i in 3:n) fib<-c(0,fib[1])+fib[2]
    fib[2]
  }
}

```

The function can then be called as

```
fib(10)
```

29.6 Use in industry

Functional programming has long been popular in academia, but with few industrial applications.[60]:page 11 However, recently several prominent functional programming languages have been used in commercial or industrial systems. For example, the Erlang programming language, which was developed by the Swedish company Ericsson in the late 1980s, was originally used to implement fault-tolerant telecommunications systems.[11] It has since become popular for building a range of applications at companies such as T-Mobile, Nortel, Facebook, Électricité de France and WhatsApp. [10] [12] [61] [62] [63] The Scheme dialect of Lisp was used as the basis for several applications on early Apple Macintosh computers, [2] [3] and has more recently been applied to problems such as training simulation software[4] and telescope control.[5] OCaml, which was introduced in the

mid-1990s, has seen commercial use in areas such as financial analysis, [13] driver verification, industrial robot programming, and static analysis of embedded software.[14] Haskell, although initially intended as a research language, [16] has also been applied by a range of companies, in areas such as aerospace systems, hardware design, and web programming.[15] [16]

Other functional programming languages that have seen use in industry include Scala, [64] F#, [17] [18] (both being functional-OO hybrids with support for both purely functional and imperative programming) Wolfram Language, [8] Lisp, [65] Standard ML, [66] [67] and Clojure.[68]

29.7 In education

Functional programming is being used as a method to teach problem solving, algebra and geometric concepts.[69] It has also been used as a tool to teach classical mechanics in Structure and Interpretation of Classical Mechanics.

Aims

This chapter provides a short background to SQL.

30 SQL background

This chapter provides a brief background to SQL. Most of the information is taken from the Wikipedia entry. The Wikipedia entry is a good place to start. I have been using relational database management software since working at Imperial College in the 1970s and 1980s. One of the first relational systems was RIM (Relational Information Management), originally a NASA developed software package as part of the Space Shuttle project. We used the CDC implementation at Imperial. We also used MicroRIM which was the first pc based relational dbms. This is now available as Rbase. See the following site for more information.

<http://www.rbase.com/>

I have also used

IBM's DB2

Oracle

MySQL

Most programmers will end up working with SQL at some point.

30.1 SQL background

The Wikipedia home page is:

<https://en.wikipedia.org/wiki/SQL>

Paradigm	Multi-paradigm
Designed by	Donald D. Chamberlin Raymond F. Boyce
Developer	ISO/IEC
First appeared	1974
Stable release	SQL:2011 / 2011
Typing discipline	Static, strong
OS	Cross-platform
File formats	File format details
Filename extension	.sql
Internet media type	application/sql[1][2]
Developed by	ISO/IEC
Initial release	1986
Latest release	SQL:2011 (2011; 4 years ago)
Type of format	Database Standard ISO/IEC 9075
Open format?	Yes
Major implementations	Many
Dialects	SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011
Influenced by	Datalog

Influenced CQL, LINQ, SOQL, Windows PowerShell,[3] JPQL, jOOQ

SQL Structured Query Language is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and a data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

SQL was one of the first commercial languages for Edgar F. Codd's relational model, as described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks." [10] Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language. [11][12]

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. [13] Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, though, most SQL code is not completely portable among different database systems without adjustments. We have coverage below of the following:

History

Design

Syntax: Language elements

Operators

Queries: Subqueries

Inline View

Null or three-valued logic (3VL)

Data manipulation

Transaction controls

Data definition

Data types

Data control

Procedural extensions

Interoperability and standardization

Alternatives

Distributed SQL processing

See also

Notes

References

External links

30.1.1 History

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. [14] This version, initially called SEQUEL (Structured English QUery Lan-

guage), was designed to manipulate and retrieve data stored in IBM's original quasi-relational database management system, System R, which a group at IBM San Jose Research Laboratory had developed during the 1970s.[14] The acronym SEQUEL was later changed to SQL because "SEQUEL" was a trademark of the UK-based Hawker Siddeley aircraft company.[15]

In the late 1970s, Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Codd, Chamberlin, and Boyce, and developed their own SQL-based RDBMS with aspirations of selling it to the U.S. Navy, Central Intelligence Agency, and other U.S. government agencies. In June 1979, Relational Software, Inc. introduced the first commercially available implementation of SQL, Oracle V2 (Version2) for VAX computers.

After testing SQL at customer test sites to determine the usefulness and practicality of the system, IBM began developing commercial products based on their System R prototype including System/38, SQL/DS, and DB2, which were commercially available in 1979, 1981, and 1983, respectively.[16]

30.1.2 SQL online documentation

Visit

<https://www.w3schools.com/sql/default.asp>

for a free reference.

30.1.3 Design

SQL deviates in several ways from its theoretical foundation, the relational model and its tuple calculus. In that model, a table is a set of tuples, while in SQL, tables and query results are lists of rows: the same row may occur multiple times, and the order of rows can be employed in queries (e.g. in the LIMIT clause).

Critics argue that SQL should be replaced with a language that strictly returns to the original foundation: for example, see The Third Manifesto.

30.1.4 Syntax

It has been suggested that this section be split into a new article titled SQL syntax. (Discuss) (June 2015)

30.1.5 Language elements

A chart showing several of the SQL language elements that compose a single statement

The SQL language is subdivided into several language elements, including:

- Clauses, which are constituent components of statements and queries. (In some cases, these are optional.)[17]

- Expressions, which can produce either scalar values, or tables consisting of columns and rows of data

- Predicates, which specify conditions that can be evaluated to SQL three-valued logic (3VL) (true/false/unknown) or Boolean truth values and are used to limit the effects of statements and queries, or to change program flow.

- Queries, which retrieve the data based on specific criteria. This is an important element of SQL.

- Statements, which may have a persistent effect on schemata and data, or may control transactions, program flow, connections, sessions, or diagnostics. SQL

statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.

Insignificant whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

30.1.6 Operators

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<> 'Sales'	Not equal to	(many DBMSs accept != in addition to <>) Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05
BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'
IN	Equal to one of multiple possible values	DeptCode IN (101, 103, 209)
IS or IS NOT	Compare to null (missing data)	Address IS NOT NULL
IS NOT DISTINCT FROM	Is equal to value or both are nulls (missing data)	Debt IS NOT DISTINCT FROM - Receivables
AS	Used to change a field name when viewing results	SELECT employee AS 'department1'

Other operators have at times been suggested and/or implemented, such as the skyline operator (for finding only those records that are not 'worse' than any others).

SQL has the case/when/then/else/end expression, which was introduced in SQL-92. In its most general form, which is called a "searched case" in the SQL standard, it works like else if in other programming languages:

```
CASE WHEN n > 0
      THEN 'positive'
      WHEN n < 0
      THEN 'negative'
      ELSE 'zero'
END
```

SQL tests WHEN conditions in the order they appear in the source. If the source does not specify an ELSE expression, SQL defaults to ELSE NULL. An abbreviated syntax—called "simple case" in the SQL standard—mirrors switch statements:

```
CASE n WHEN 1
      THEN 'one'
      WHEN 2
      THEN 'two'
      ELSE 'I cannot count that high'
END
```

This syntax uses implicit equality comparisons, with the usual caveats for comparing with NULL.

For the Oracle-SQL dialect, the latter can be shortened to an equivalent DECODE construct:

```
SELECT DECODE(n, 1, 'one',
              2, 'two',
              'i cannot count that high')
FROM   some_table;
```

The last value is the default; if none is specified, it also defaults to NULL. However, unlike the standard's "simple case", Oracle's DECODE considers two NULLs equal with each other.[18]

30.1.7 Queries

The most common operation in SQL, the query, makes use of the declarative SELECT statement. SELECT retrieves data from one or more tables, or expressions. Standard SELECT statements have no persistent effects on the database. Some non-standard implementations of SELECT can have persistent effects, such as the SELECT INTO syntax provided in some databases.[19]

Queries allow the user to describe desired data, leaving the database management system (DBMS) to carry out planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

A query includes a list of columns to include in the final result, normally immediately following the SELECT keyword. An asterisk ("*") can be used to specify that the query should return all columns of the queried tables. SELECT is the most complex statement in SQL, with optional keywords and clauses that include:

The FROM clause, which indicates the table(s) to retrieve data from. The FROM clause can include optional JOIN subclauses to specify the rules for joining tables.

The WHERE clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set where the comparison predicate does not evaluate to True.

The GROUP BY clause projects rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.

The HAVING clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.

The ORDER BY clause identifies which column[s] to use to sort the resulting data, and in which direction to sort them (ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

The DISTINCT keyword[20] eliminates duplicate data.[21]

The following example of a SELECT query returns a list of expensive books. The query retrieves all rows from the Book table in which the price column contains a value greater than 100.00. The result is sorted in ascending order by title. The asterisk (*) in the select list indicates that all columns of the Book table should be included in the result set.

```
SELECT *
FROM Book
WHERE price > 100.00
ORDER BY title;
```

The example below demonstrates a query of multiple tables, grouping, and aggregation, by returning a list of books and the number of authors associated with each book.

```
SELECT Book.title AS Title,
       count(*) AS Authors
FROM Book
JOIN Book_author
ON Book.isbn = Book_author.isbn
GROUP BY Book.title;
```

Example output might resemble the following:

Title	Authors
-----	-----
SQL Examples and Guide	4
The Joy of SQL	1
An Introduction to SQL	2
Pitfalls of SQL	1

Under the precondition that isbn is the only common column name of the two tables and that a column named title only exists in the Books table, one could re-write the query above in the following form:

```
SELECT title,
       count(*) AS Authors
FROM Book
NATURAL JOIN Book_author
GROUP BY title;
```

However, many[quantify] vendors either do not support this approach, or require certain column-naming conventions for natural joins to work effectively.

SQL includes operators and functions for calculating values on stored values. SQL allows the use of expressions in the select list to project data, as in the following example, which returns a list of books that cost more than 100.00 with an additional sales_tax column containing a sales tax figure calculated at 6% of the price.

```
SELECT isbn,
       title,
       price,
       price * 0.06 AS sales_tax
FROM Book
WHERE price > 100.00
ORDER BY title;
```

30.1.8 Subqueries

Queries can be nested so that the results of one query can be used in another query via a relational operator or aggregation function. A nested query is also known as a subquery. While joins and other table operations provide computationally superior (i.e. faster) alternatives in many cases, the use of subqueries introduces a hierarchy in execution that can be useful or necessary. In the following example, the aggregation function AVG receives as input the result of a subquery:

```
SELECT isbn,
       title,
       price
FROM   Book
WHERE  price < (SELECT AVG(price) FROM Book)
ORDER BY title;
```

A subquery can use values from the outer query, in which case it is known as a correlated subquery.

Since 1999 the SQL standard allows named subqueries called common table expressions (named and designed after the IBM DB2 version 2 implementation; Oracle calls these subquery factoring). CTEs can also be recursive by referring to themselves; the resulting mechanism allows tree or graph traversals (when represented as relations), and more generally fixpoint computations.

30.1.9 Inline View

An Inline view is the use of referencing an SQL subquery in a FROM clause. Essentially, the inline view is a subquery that can be selected from or joined to. Inline View functionality allows the user to reference the subquery as a table. The inline view also is referred to as a derived table or a subselect. Inline view functionality was introduced in Oracle 9i.[22]

In the following example, the SQL statement involves a join from the initial Books table to the Inline view "Sales". This inline view captures associated book sales information using the ISBN to join to the Books table. As a result, the inline view provides the result set with additional columns (the number of items sold and the company that sold the books):

```
Select b.isbn, b.title, b.price, sales.items_sold, sales.com-
pany_nm
from Book b,
(select SUM(Items_Sold) Items_Sold, Company_Nm, ISBN
from Book_Sales
group by Company_Nm, ISBN) sales
WHERE sales.isbn = b.isbn
```

30.1.10 Null or three-valued logic (3VL)

Main article: Null (SQL)

The concept of Null was introduced[by whom?] into SQL to handle missing information in the relational model. The word NULL is a reserved keyword in SQL, used to identify the Null special marker. Comparisons with Null, for instance equality (=) in WHERE clauses, results in an Unknown truth value. In SELECT statements SQL returns only results for which the WHERE clause returns a value of True; i.e., it excludes results with values of False and also excludes those whose value is Unknown.

Along with True and False, the Unknown resulting from direct comparisons with Null thus brings a fragment of three-valued logic to SQL. The truth tables SQL uses for AND, OR, and NOT correspond to a common fragment of the Kleene and Lukasiewicz three-valued

logic (which differ in their definition of implication, however SQL defines no such operation).[23]

p AND q		p		
		True	False	Unknown
q	True	True	False	Unknown
	False	False	False	False
	Unknown	Unknown	False	Unknown
p OR q		p		
		True	False	Unknown
q	True	True	True	True
	False	True	False	Unknown
	Unknown	True	Unknown	Unknown
p = q		p		
		True	False	Unknown
q	True	True	False	Unknown
	False	False	True	Unknown
	Unknown	Unknown	Unknown	Unknown
q		NOT q		
	True	False		
	False	True		
	Unknown	Unknown		

There are however disputes about the semantic interpretation of Nulls in SQL because of its treatment outside direct comparisons. As seen in the table above, direct equality comparisons between two NULLs in SQL (e.g. NULL = NULL) return a truth value of Unknown. This is in line with the interpretation that Null does not have a value (and is not a member of any data domain) but is rather a placeholder or "mark" for missing information. However, the principle that two Nulls aren't equal to each other is effectively violated in the SQL specification for the UNION and INTERSECT operators, which do identify nulls with each other.[24] Consequently, these set operations in SQL may produce results not representing sure information, unlike operations involving explicit comparisons with NULL (e.g. those in a WHERE clause discussed above). In Codd's 1979 proposal (which was basically adopted by SQL92) this semantic inconsistency is rationalized by arguing that removal of duplicates in set operations happens "at a lower level of detail than equality testing in the

evaluation of retrieval operations".[23] However, computer-science professor Ron van der Meyden concluded that "The inconsistencies in the SQL standard mean that it is not possible to ascribe any intuitive logical semantics to the treatment of nulls in SQL."[24]

Additionally, because SQL operators return Unknown when comparing anything with Null directly, SQL provides two Null-specific comparison predicates: IS NULL and IS NOT NULL test whether data is or is not Null.[25] SQL does not explicitly support universal quantification, and must work it out as a negated existential quantification.[26][27][28] There is also the "<row value expression> IS DISTINCT FROM <row value expression>" infix comparison operator, which returns TRUE unless both operands are equal or both are NULL. Likewise, IS NOT DISTINCT FROM is defined as "NOT (<row value expression> IS DISTINCT FROM <row value expression>)". SQL:1999 also introduced BOOLEAN type variables, which according to the standard can also hold Unknown values. In practice, a number of systems (e.g. PostgreSQL) implement the BOOLEAN Unknown as a BOOLEAN NULL.

30.1.11 Data manipulation

The Data Manipulation Language (DML) is the subset of SQL used to add, update and delete data:

INSERT adds rows (formally tuples) to an existing table, e.g.:

```
INSERT INTO example
  (field1, field2, field3)
VALUES
  ('test', 'N', NULL);
```

UPDATE modifies a set of existing table rows, e.g.:

```
UPDATE example
  SET field1 = 'updated value'
  WHERE field2 = 'N';
```

DELETE removes existing rows from a table, e.g.:

```
DELETE FROM example
  WHERE field2 = 'N';
```

MERGE is used to combine the data of multiple tables. It combines the INSERT and UPDATE elements. It is defined in the SQL:2003 standard; prior to that, some databases provided similar functionality via different syntax, sometimes called "upsert".

```
MERGE INTO table_name USING table_reference ON (condition)
  WHEN MATCHED THEN
    UPDATE SET column1 = value1 [, column2 = value2 ...]
  WHEN NOT MATCHED THEN
    INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...])
```

30.1.12 Transaction controls

Transactions, if available, wrap DML operations:

START TRANSACTION (or BEGIN WORK, or BEGIN TRANSACTION, depending on SQL dialect) marks the start of a database transaction, which either completes entirely or not at all.

SAVE TRANSACTION (or SAVEPOINT) saves the state of the database at the current point in transaction

```
CREATE TABLE tbl_1(id int);
INSERT INTO tbl_1(id) VALUES(1);
```



```

INSERT INTO tbl_1(id) VALUES(2);
COMMIT;
UPDATE tbl_1 SET id=200 WHERE id=1;
SAVEPOINT id_1upd;
UPDATE tbl_1 SET id=1000 WHERE id=2;
ROLLBACK to id_1upd;
SELECT id from tbl_1;

```

COMMIT makes all data changes in a transaction permanent.

ROLLBACK discards all data changes since the last COMMIT or ROLLBACK, leaving the data as it was prior to those changes. Once the COMMIT statement completes, the transaction's changes cannot be rolled back.

COMMIT and ROLLBACK terminate the current transaction and release data locks. In the absence of a START TRANSACTION or similar statement, the semantics of SQL are implementation-dependent. The following example shows a classic transfer of funds transaction, where money is removed from one account and added to another. If either the removal or the addition fails, the entire transaction is rolled back.

```

START TRANSACTION;
UPDATE Account SET amount=amount-200 WHERE account_number=1234;
UPDATE Account SET amount=amount+200 WHERE account_number=2345;
IF ERRORS=0 COMMIT;
IF ERRORS<>0 ROLLBACK;

```

30.1.13 Data definition

The Data Definition Language (DDL) manages table and index structure. The most basic items of DDL are the CREATE, ALTER, RENAME, DROP and TRUNCATE statements:

CREATE creates an object (a table, for example) in the database, e.g.:

```

CREATE TABLE example(
  column1 INTEGER,
  column2 VARCHAR(50),
  column3 DATE NOT NULL,
  PRIMARY KEY (column1, column2)
);

```

ALTER modifies the structure of an existing object in various ways, for example, adding a column to an existing table or a constraint, e.g.:

```
ALTER TABLE example ADD column4 NUMBER(3) NOT NULL;
```

TRUNCATE deletes all data from a table in a very fast way, deleting the data inside the table and not the table itself. It usually implies a subsequent COMMIT operation, i.e., it cannot be rolled back (data is not written to the logs for rollback later, unlike DELETE).

```
TRUNCATE TABLE example;
```

DROP deletes an object in the database, usually irretrievably, i.e., it cannot be rolled back, e.g.:

```
DROP TABLE example;
```

30.1.14 Data types

Each column in an SQL table declares the type(s) that column may contain. ANSI SQL includes the following data types.[29]

30.1.14.1 Character strings

CHARACTER(n) or CHAR(n):

fixed-width n-character string, padded with spaces as needed

CHARACTER VARYING(n) or VARCHAR(n):

variable-width string with a maximum size of n characters

NATIONAL CHARACTER(n) or NCHAR(n):

fixed width string supporting an international character set

NATIONAL CHARACTER VARYING(n) or NVARCHAR(n):

variable-width NCHAR string

30.1.14.2 Bit strings

BIT(n): an array of n bits

BIT VARYING(n): an array of up to n bits

30.1.14.3 Numbers

INTEGER, SMALLINT and BIGINT

FLOAT, REAL and DOUBLE PRECISION

NUMERIC(precision, scale) or DECIMAL(precision, scale)

For example, the number 123.45 has a precision of 5 and a scale of 2. The precision is a positive integer that determines the number of significant digits in a particular radix (binary or decimal). The scale is a non-negative integer. A scale of 0 indicates that the number is an integer. For a decimal number with scale S, the exact numeric value is the integer value of the significant digits divided by 10^S.

SQL provides a function to round numerics or dates, called TRUNC (in Informix, DB2, PostgreSQL, Oracle and MySQL) or ROUND (in Informix, SQLite, Sybase, Oracle, PostgreSQL and Microsoft SQL Server)[30]

30.1.14.4 Temporal (date/time)

DATE: for date values (e.g. 2011-05-03)

TIME: for time values (e.g. 15:51:36). The granularity of the time value is usually a tick (100 nanoseconds).

TIME WITH TIME ZONE or TIMETZ: the same as TIME, but including details about the time zone in question.

TIMESTAMP: This is a DATE and a TIME put together in one variable (e.g. 2011-05-03 15:51:36).

TIMESTAMP WITH TIME ZONE or TIMESTAMPTZ: the same as TIMESTAMP, but including details about the time zone in question.

SQL provides several functions for generating a date / time variable out of a date / time string (TO_DATE, TO_TIME, TO_TIMESTAMP), as well as for extracting the respective members (seconds, for instance) of such variables. The current system date / time of the database server can be called by using functions like NOW. The IBM Informix implementation provides the EXTEND and the FRACTION functions to increase the accuracy of time, for systems requiring sub-second precision.[31]

30.1.15 Data control

The Data Control Language (DCL) authorizes users to access and manipulate data. Its two main statements are:

GRANT authorizes one or more users to perform an operation or a set of operations on an object.

REVOKE eliminates a grant, which may be the default grant.

Example:

```
GRANT SELECT, UPDATE
  ON example
  TO some_user, another_user;
REVOKE SELECT, UPDATE
  ON example
  FROM some_user, another_user;
```

30.1.16 Procedural extensions

SQL is designed for a specific purpose: to query data contained in a relational database. SQL is a set-based, declarative programming language, not an imperative programming language like C or BASIC. However, extensions to Standard SQL add procedural programming language functionality, such as control-of-flow constructs. These include:

Source	Common name	Full name
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase / Firebird	PSQL	Procedural SQL
IBM DB2	SQL PL	SQL Procedural Language (implements SQL/PSM)
IBM Informix	SPL	Stored Procedural Language
IBM Netezza	NZPLSQL [3]	(based on Postgres PL/pgSQL)
Microsoft / Sybase	T-SQL	Transact-SQL
Mimer SQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MySQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MonetDB	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
NuoDB	SSP	Starkey Stored Procedures
Oracle	PL/SQL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL (based on Oracle PL/SQL)

PostgreSQL	PL/PSM	Procedural Language/ Persistent Stored Modules (implements SQL/PSM)
Sybase	Watcom-SQL	SQL Anywhere Watcom-SQL Dialect
Teradata	SPL	Stored Procedural Language
SAP	SAP HANA	SQL Script

In addition to the standard SQL/PSM extensions and proprietary SQL extensions, procedural and object-oriented programmability is available on many SQL platforms via DBMS integration with other languages. The SQL standard defines SQL/JRT extensions (SQL Routines and Types for the Java Programming Language) to support Java code in SQL databases. SQL Server 2005 uses the SQLCLR (SQL Server Common Language Runtime) to host managed .NET assemblies in the database, while prior versions of SQL Server were restricted to unmanaged extended stored procedures primarily written in C. PostgreSQL lets users write functions in a wide variety of languages — including Perl, Python, Tcl, and C.[32]

30.1.17 Interoperability and standardization

SQL implementations are incompatible between vendors and do not necessarily completely follow standards. In particular date and time syntax, string concatenation, NULLs, and comparison case sensitivity vary from vendor to vendor. A particular exception is PostgreSQL, which strives for standards compliance.[33]

Popular implementations of SQL commonly omit support for basic features of Standard SQL, such as the DATE or TIME data types. The most obvious such examples, and incidentally the most popular commercial and proprietary SQL DBMSs, are Oracle (whose DATE behaves as DATETIME,[34][35] and lacks a TIME type)[36] and MS SQL Server (before the 2008 version). As a result, SQL code can rarely be ported between database systems without modifications.

There are several reasons for this lack of portability between database systems:

- The complexity and size of the SQL standard means that most implementors do not support the entire standard.

- The standard does not specify database behavior in several important areas (e.g. indexes, file storage...), leaving implementations to decide how to behave.

- The SQL standard precisely specifies the syntax that a conforming database system must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to ambiguity.

- Many database vendors have large existing customer bases; where the newer version of the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.

- There is little commercial incentive for vendors to make it easier for users to change database suppliers (see vendor lock-in).

- Users evaluating database software tend to place other factors such as performance higher in their priorities than standards conformance.

SQL was adopted as a standard by the American National Standards Institute (ANSI) in 1986 as SQL-86[37] and the International Organization for Standardization (ISO) in 1987. Nowadays the standard is subject to continuous improvement by the Joint Technical Com-

mittee ISO/IEC JTC 1, Information technology, Subcommittee SC 32, Data management and interchange, which affiliate to ISO as well as IEC. It is commonly denoted by the pattern: ISO/IEC 9075-n:yyyy Part n: title, or, as a shortcut, ISO/IEC 9075.

ISO/IEC 9075 is complemented by ISO/IEC 13249: SQL Multimedia and Application Packages (SQL/MM), which defines SQL based interfaces and packages to widely spread applications like video, audio and spatial data.

Until 1996, the National Institute of Standards and Technology (NIST) data management standards program certified SQL DBMS compliance with the SQL standard. Vendors now self-certify the compliance of their products.[38]

The original standard declared that the official pronunciation for "SQL" was an initialism ("es queue el").[11] Regardless, many English-speaking database professionals (including Donald Chamberlin himself[39]) use the acronym-like pronunciation of "sequel"),[40] mirroring the language's pre-release development name of "SEQUEL".[14][15]

The SQL standard has gone through a number of revisions:

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), Entry Level SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT).
2003	SQL:2003	SQL 2003	Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	SQL 2006	ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[41]
2008	SQL:2008	SQL 2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers Adds the TRUNCATE statement.[42]
2011	SQL:2011		

Interested parties may purchase SQL standards documents from ISO,[43] IEC or ANSI. A draft of SQL:2008 is freely available as a zip archive.[44]

The SQL standard is divided into nine parts.

ISO/IEC 9075-1:2011 Part 1: Framework (SQL/Framework). It provides logical concepts.

ISO/IEC 9075-2:2011 Part 2: Foundation (SQL/Foundation). It contains the most central elements of the language and consists of both mandatory and optional features.

ISO/IEC 9075-3:2008 Part 3: Call-Level Interface (SQL/CLI). It defines interfacing components (structures, procedures, variable bindings) that can be used to execute SQL statements from applications written in Ada, C respectively C++, COBOL, Fortran, MUMPS, Pascal or PL/I. (For Java see part 10.) SQL/CLI is defined in such a way that SQL statements and SQL/CLI procedure calls are treated as separate from the calling application's source code. Open Database Connectivity is a well-known superset of SQL/CLI. This part of the standard consists solely of mandatory features.

ISO/IEC 9075-4:2011 Part 4: Persistent Stored Modules (SQL/PSM) It standardizes procedural extensions for SQL, including flow of control, condition handling, statement condition signals and resignals, cursors and local variables, and assignment of expressions to variables and parameters. In addition, SQL/PSM formalizes declaration and maintenance of persistent database language routines (e.g., "stored procedures"). This part of the standard consists solely of optional features.

ISO/IEC 9075-9:2008 Part 9: Management of External Data (SQL/MED). It provides extensions to SQL that define foreign-data wrappers and datalink types to allow SQL to manage external data. External data is data that is accessible to, but not managed by, an SQL-based DBMS. This part of the standard consists solely of optional features.

ISO/IEC 9075-10:2008 Part 10: Object Language Bindings (SQL/OLB). It defines the syntax and semantics of SQLJ, which is SQL embedded in Java (see also part 3). The standard also describes mechanisms to ensure binary portability of SQLJ applications, and specifies various Java packages and their contained classes. This part of the standard consists solely of optional features, as opposed to SQL/OLB JDBC, which is not part of the SQL standard, which defines an API.[citation needed]

ISO/IEC 9075-11:2011 Part 11: Information and Definition Schemas (SQL/Schemata). It defines the Information Schema and Definition Schema, providing a common set of tools to make SQL databases and objects self-describing. These tools include the SQL object identifier, structure and integrity constraints, security and authorization specifications, features and packages of ISO/IEC 9075, support of features provided by SQL-based DBMS implementations, SQL-based DBMS implementation information and sizing items, and the values supported by the DBMS implementations.[45] This part of the standard contains both mandatory and optional features.

ISO/IEC 9075-13:2008 Part 13: SQL Routines and Types Using the Java Programming Language (SQL/JRT). It specifies the ability to invoke static Java methods as routines from within SQL applications ('Java-in-the-database'). It also calls for the ability to use Java classes as SQL structured user-defined types. This part of the standard consists solely of optional features.

ISO/IEC 9075-14:2011 Part 14: XML-Related Specifications (SQL/XML). It specifies SQL-based extensions for using XML in conjunction with SQL. The XML data type is introduced, as well as several routines, functions, and XML-to-SQL data type mappings to support manipulation and storage of XML in an SQL database.[41] This part of the standard consists solely of optional features.[citation needed]

ISO/IEC 9075 is complemented by ISO/IEC 13249 SQL Multimedia and Application Packages. This closely related but separate standard is developed by the same committee. It defines interfaces and packages based on SQL. The aim is a unified access to typical database applications like text, pictures, data mining or spatial data.

ISO/IEC 13249-1:2007 Part 1: Framework

ISO/IEC 13249-2:2003 Part 2: Full-Text

ISO/IEC 13249-3:2011 Part 3: Spatial

ISO/IEC 13249-5:2003 Part 5: Still image

ISO/IEC 13249-6:2006 Part 6: Data mining

ISO/IEC 13249-8:xxxx Part 8: Metadata registries (MDR) (work in progress)

30.1.18 Alternatives

A distinction should be made between alternatives to SQL as a language, and alternatives to the relational model itself. Below are proposed relational alternatives to the SQL language. See navigational database and NoSQL for alternatives to the relational model.

.QL: object-oriented Datalog

4D Query Language (4D QL)

BQL: a superset that compiles down to SQL

Datalog: critics suggest that Datalog has two advantages over SQL: it has cleaner semantics, which facilitates program understanding and maintenance, and it is more expressive, in particular for recursive queries.[46]

HTSQL: URL based query method

IBM Business System 12 (IBM BS12): one of the first fully relational database management systems, introduced in 1982

ISBL

jOOQ: SQL implemented in Java as an internal domain-specific language

Java Persistence Query Language (JPQL): The query language used by the Java Persistence API and Hibernate persistence library

LINQ: Runs SQL statements written like language constructs to query collections directly from inside .Net code.

Object Query Language

OttoQL

QBE (Query By Example) created by Moshè Zloof, IBM 1977

Quel introduced in 1974 by the U.C. Berkeley Ingres project.

Tutorial D

XQuery

30.1.19 Distributed SQL processing

Distributed Relational Database Architecture (DRDA) was designed by a work group within IBM in the period 1988 to 1994. DRDA enables network connected relational databases to cooperate to fulfill SQL requests.[47][48]

An interactive user or program can issue SQL statements to a local RDB and receive tables of data and status indicators in reply from remote RDBs. SQL statements can also be compiled and stored in remote RDBs as packages and then invoked by package name. This is important for the efficient operation of application programs that issue complex, high-frequency queries. It is especially important when the tables to be accessed are located in remote systems.

The messages, protocols, and structural components of DRDA are defined by the Distributed Data Management Architecture.

30.1.20 See also

The following are hot links from the Wikipedia pages.

- Comparison of object-relational database management systems

- Comparison of relational database management systems

- D (data language specification)

- D4 (programming language)

- Hierarchical model

- List of relational database management systems

- MUMPS

- NoSQL

- Transact-SQL

- Online analytical processing (OLAP)

- Online transaction processing (OLTP)

- Data warehouse

- relational data stream management system

- Star schema

- Snowflake schema

- DB2 SQL return codes

30.1.21 Notes

1. "Media Type registration for application/sql". Internet Assigned Numbers Authority. 10 April 2013. Retrieved 10 April 2013.
2. "The application/sql Media Type, RFC 6922". Internet Engineering Task Force. April 2013. p. 3. Retrieved 10 April 2013.
3. Paul, Ryan. "A guided tour of the Microsoft Command Shell". Ars Technica. Retrieved 10 April 2011.
4. Beaulieu, Alan (April 2009). Mary E Treseler, ed. Learning SQL (2nd ed.). Sebastapol, CA, USA: O'Reilly. ISBN 978-0-596-52083-0.
5. "SQL, n.". Oxford English Dictionary. Oxford University Press. Retrieved 2014-11-27.
6. Encyclopedia Britannica. "SQL". Retrieved 2013-04-02.

7. Oxford Dictionaries. "SQL".
8. IBM. "SQL Guide".
9. Microsoft. "Structured Query Language (SQL)".
10. Codd, Edgar F (June 1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM (Association for Computing Machinery)* 13 (6): 377–87. doi:10.1145/362384.362685. Retrieved 2007-06-09.
11. Jump up to: a b Chapple, Mike. "SQL Fundamentals". *Databases. About.com*. Retrieved 2009-01-28.
12. "Structured Query Language (SQL)". *International Business Machines*. October 27, 2006. Retrieved 2007-06-10.
13. "ISO/IEC 9075-1:2008: Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)".
14. Jump up to: a b c Chamberlin, Donald D; Boyce, Raymond F (1974). "SEQUEL: A Structured English Query Language" (PDF). *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control (Association for Computing Machinery)*: 249–64. Retrieved 2007-06-09.
15. Jump up to: a b Oppel, Andy (February 27, 2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media. pp. 90–1. ISBN 0-07-146960-5.
16. "History of IBM, 1978". *IBM Archives*. IBM. Retrieved 2007-06-09.
17. ANSI/ISO/IEC International Standard (IS). *Database Language SQL—Part 2: Foundation (SQL/Foundation)*. 1999.
18. "DECODE". *Docs.oracle.com*. Retrieved 2013-06-14.
19. "Transact-SQL Reference". *SQL Server Language Reference. SQL Server 2005 Books Online*. Microsoft. 2007-09-15. Retrieved 2007-06-17.
20. SAS 9.4 SQL Procedure User's Guide. SAS Institute. 2013. p. 248. ISBN 9781612905686. Retrieved 2015-10-21. "Although the UNIQUE argument is identical to DISTINCT, it is not an ANSI standard."
21. Leon, Alexis; Leon, Mathews (1999). "Eliminating duplicates - SELECT using DISTINCT". *SQL: A Complete Reference*. New Delhi: Tata McGraw-Hill Education (published 2008). p. 143. ISBN 9780074637081. Retrieved 2015-10-21. "[...] the keyword DISTINCT [...] eliminates the duplicates from the result set."
22. "Derived Tables". ORACLE.
23. Jump up to: a b Hans-Joachim, K. (2003). "Null Values in Relational Databases and Sure Information Answers". *Semantics in Databases. Second International Workshop Dagstuhl Castle, Germany, January 7–12, 2001. Revised Papers. Lecture Notes in Computer Science* 2582. pp. 119–138. doi:10.1007/3-540-36596-6_7. ISBN 978-3-540-00957-3.
24. Jump up to: a b Ron van der Meyden, "Logical approaches to incomplete information: a survey" in Chomicki, Jan; Saake, Gunter (Eds.) *Logics for Databases and Information Systems*, Kluwer Academic Publishers ISBN 978-0-7923-8129-7, p. 344
25. ISO/IEC. ISO/IEC 9075-2:2003, "SQL/Foundation". ISO/IEC.
26. M. Negri, G. Pelagatti, L. Sbattella (1989) *GUIDE Semantics and problems of universal quantification in SQL*
27. Fratarcangeli, Claudio (1991). *Technique for universal quantification in SQL*. ACM.org.

28. Kawash, Jalal (2004) Complex quantification in Structured Query Language (SQL): a tutorial using relational calculus - Journal of Computers in Mathematics and Science Teaching ISSN 0731-9258 Volume 23, Issue 2, 2004 AACE Norfolk, Virginia. Thefreelibrary.com
29. "Information Technology: Database Language SQL". CMU. (proposed revised text of DIS 9075).
30. Arie Jones, Ryan K. Stephens, Ronald R. Plew, Alex Kriegel, Robert F. Garrett (2005), SQL Functions Programmer's Reference. Wiley, 127 pages.
31. [1]
32. PostgreSQL contributors (2011). "PostgreSQL server programming". PostgreSQL 9.1 official documentation. postgresql.org. Retrieved 2012-03-09.
33. PostgreSQL contributors (2012). "About PostgreSQL". PostgreSQL 9.1 official website. PostgreSQL Global Development Group. Retrieved March 9, 2012. "PostgreSQL prides itself in standards compliance. Its SQL implementation strongly conforms to the ANSI-SQL:2008 standard"
34. Lorentz, Diana; Roeser, Mary Beth; Abraham, Sundeep; Amor, Angela; Arora, Geeta; Arora, Vikas; Ashdown, Lance; Baer, Hermann; Bellamkonda, Shrikanth (October 2010) [1996]. "Basic Elements of Oracle SQL: Data Types". Oracle Database SQL Language Reference 11g Release 2 (11.2). Oracle Database Documentation Library. Redwood City, CA: Oracle USA, Inc. Retrieved December 29, 2010. "For each DATE value, Oracle stores the following information: century, year, month, date, hour, minute, and second"
35. Lorentz, Diana; Roeser, Mary Beth; Abraham, Sundeep; Amor, Angela; Arora, Geeta; Arora, Vikas; Ashdown, Lance; Baer, Hermann; Bellamkonda, Shrikanth (October 2010) [1996]. "Basic Elements of Oracle SQL: Data Types". Oracle Database SQL Language Reference 11g Release 2 (11.2). Oracle Database Documentation Library. Redwood City, CA: Oracle USA, Inc. Retrieved December 29, 2010. "The datetime data types are DATE..."
36. Lorentz, Diana; Roeser, Mary Beth; Abraham, Sundeep; Amor, Angela; Arora, Geeta; Arora, Vikas; Ashdown, Lance; Baer, Hermann; Bellamkonda, Shrikanth (October 2010) [1996]. "Basic Elements of Oracle SQL: Data Types". Oracle Database SQL Language Reference 11g Release 2 (11.2). Oracle Database Documentation Library. Redwood City, CA: Oracle USA, Inc. Retrieved December 29, 2010. "Do not define columns with the following SQL/DS and DB2 data types, because they have no corresponding Oracle data type:... TIME"
37. "Finding Aid". X3H2 Records, 1978–95. American National Standards Institute.
38. Doll, Shelley (June 19, 2002). "Is SQL a Standard Anymore?". TechRepublic's Builder.com. TechRepublic. Archived from the original on 2013-01-02. Retrieved 2010-01-07.
39. Gillespie, Patrick. "Pronouncing SQL: S-Q-L or Sequel?". Pronouncing SQL: S-Q-L or Sequel?. Retrieved 12 February 2012.
40. Melton, Jim; Alan R Simon (1993). "1.2. What is SQL?". Understanding the New SQL: A Complete Guide. Morgan Kaufmann. p. 536. ISBN 1-55860-245-3. "SQL (correctly pronounced "ess cue ell," instead of the somewhat common "sequel")..."
41. Jump up to: a b Wagner, Michael (2010). SQL/XML:2006 - Evaluierung der Standardkonformität ausgewählter Datenbanksysteme. Diplomica Verlag. p. 100. ISBN 3-8366-9609-6.
42. "SQL:2008 now an approved ISO international standard". Sybase. July 2008.

43. "ISO/IEC 9075-2:2011: Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation)".
44. "SQL:2008 draft" (Zip). Whitemarsh Information Systems Corporation.
45. "ISO/IEC 9075-11:2008: Information and Definition Schemas (SQL/Schemata)". 2008. p. 1.
46. [2]
47. Reinsch, R. (1988). "Distributed database for SAA". IBM Systems Journal 27 (3): 362–389. doi:10.1147/sj.273.0362.
48. Distributed Relational Database Architecture Reference. IBM Corp. SC26-4651-0. 1990.

References

Codd, Edgar F (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–87. doi:10.1145/362384.362685.

Discussion on alleged SQL flaws (C2 wiki)

C. J. Date with Hugh Darwen: A Guide to the SQL standard : a users guide to the standard database language SQL, 4th ed., Addison Wesley, USA 1997, ISBN 978-0-201-96426-4

30.2 My Bibliography

These are some of the SQL books I've used, YMMV.

Cannan St., Otten G., SQL - The Standard Handbook, McGraw Hill.

This book covers ISO 9075: 1992(E).

Date C.J., Darwen H., A Guide to the SQL Standard, Third Edition, Addison Wesley.

Covers the 1992 standard.

31 Example summary

31.1 Introduction

In this chapter we provide summary details about the examples. The following table has the details.

Chapter number	Example number	Description	Page number
2	Example 1	Hello World	67
2	Example 2	Simple text I/o using Python style strings	67
2	Example 3	Simple numeric i/o	68
4	Example 1	assignment and division	77
4	Example 2	division with integers	78
4	Example 3	time taken to reach the earth from the Sun.	78
4	Example 4	converting from Fahrenheit to centigrade.	79
4	Example 5	converting from Centigrade to Fahrenheit.	79
4	Example 6	numbers getting too large - overflow	79
4	Example 7	numbers getting too small - underflow	79
4	Example 8	subtraction of two similar values	80
4	Example 9	summation	80
5	Example 1	array and conventional for loop syntax	87
5	Example 2	using the len function to determine the size of array	88
5	Example 3	reading in the array size	88
6	Example 1	simple rainfall example	96
6	Example 2	variant of one using len intrinsic function	96
6	Example 3	setting the size at run time	96
6	Example 4	two d array using numpy.zeros method	97
6	Example 5	two d array using numpy.array() method	97
6	Example 6	two d array and the numpy.sum() method	97
6	Example 7	simple one d slicing	100
6	Example 8	two d slicing	100
6	Example 9	arithmetic and slicing	101

6	Example 10	Aggregate usage	102
6	Example 11	Shape manipulation	102
6	Example 12	Copies or views	103
7	Example 1	initialisation, len and find methods	110
7	Example 2	concatenation and split method	110
7	Example 3	split variant	112
7	Example 4	reading from an external file	112
7	Example 5	reading data from a file and calculating sum and average rainfall values	115
7	Example 6	simple variant of the previous example using the .format option	117
7	Example 7	the ASCII character set	118
7	Example 8	Unicode characters	120
7	Example 9	another unicode example	121
8	Example 1	the if statement	130
8	Example 2	the while statement	131
8	Example 3	the for loop with arrays	131
8	Example 4	the for loop with lists and enumerate	132
8	Example 5	the for in statement	133
8	Example 6	try and except	133
9	Example 1	a bigger function	136
9	Example 2	a swap function	137
9	Example 3	another swap	137
9	Example 4	yet another swap	138
9	Example 5	recursive functions	138
9	Example 6	simple factorial variant, reading the value in	138
9	Example 7	testing out the maths functions	143
9	Example 8	math module sin function	145
9	Example 9	math module using numpy arrays	146
9	Example 10	math module using a pi shortcut	146

9	Example 11	Using generators	148
9	Example 12	Iterative	148
9	Example 13	Recursive	148
9	Example 14	generating prime numbers	149
9	Example 15	list and lambda usage	149
9	Example 16	functional example	150
9	Example 17	functional example	151
9	Example 18	functional example variant using the array module	152
9	Example 19	functional variant using the numpy module	152
10	Example 1	base shape class	154
10	Example 2	variation using modules	155
10	Example 3	a circle derived class	156
10	Example 4	test program for the shape and circle classes	157
10	Example 5	polymorphism and dynamic binding	158
10	Example 6	data structuring using the Met Office data	159
11	Example 1	reading from a file using substrings	162
11	Example 2	reading the same file using the split() method	164
11	Example 3	internet file read	165
11	Example 4	variation on the internet file read where we save the file	166
11	Example 5	reading all of the station data files with timing	167
11	Example 6	Writing to a set of files names generated within Python	170
11	Example 7	Copying a file and replacing missing values	170
11	Example 8	creating an SQL file	170
11	Example 9	Creating a csv file	171
11	Example 10	CSV files and the csv module	172
11	Example 11	CSV usage and data extraction	174
11	Example 12	reading a met office file using the csv module	175
11	Example 13	reading data using the genfromtxt method	176
11	Example 14	Writing a CSV file	178

11	Example 15	write large array as text file, element by element, with timing	179
11	Example 16	write large array as binary file , element by element, with timing	180
11	Example 17	write large array as binary file , whole array, with timing	181
11	Example 18	listing subdirectories	182
11	Example 19	listing all Python files	182
13	Example 1	Simple iterator usage	189
13	Example 2	list type initialisation and simple for in statement	193
13	Example 3	list type and various sequence methods	193
13	Example 4	list assignment versus copy() method	194
13	Example 5	simple list comprehension	195
13	Example 6	more list comprehensions	196
13	Example 7	more list comprehensions	196
13	Example 8	even more list comprehensions	197
13	Example 9	simple tuple usage	198
13	Example 10	simple range usage	199
14	Example 1	simple set usage	202
14	Example 2	simple dictionary	203
15	Example 1	simple dict usage	205
15	Example 2	dict view usage	205
16	Example 1	simple operator overloading	207
17	Example 1	using getcontext()	210
17	Example 2	values for the maths constants e and pi	212
17	Example 3	summation using float and decimal	213
17	Example 4	simple fraction usage	214
17	Example 5	simple random usage	216
18	Example 1	Database creation	222
18	Example 2	Table creation	222

18	Example 3	loading the earthqk table	226
18	Example 4	loading the regions table	227
18	Example 5	loading the tsunami table	228
18	Example 6	Querying the tables	228
18	Example 7	creating the database	228
18	Example 8	creating a table for one of the sites	229
18	Example 9	loading data into the table	229
18	Example 10	simple table query	230
18	Example 11	computing averages	230
18	Example 12	Finding the wettest month and displaying the year, month and rainfall	231
18	Example 13	Finding the wettest months and displaying the year, month and rainfall	232
18	Example 14	doing monthly average calculations using the genfromtxt example in the IO chapter	232
19	Example 1	UK post codes	244
21	Example 1	Serial solution	250
21	Example 2	Multi-threaded solution	252
22	Example 1	Simple multi-processing on a 6 core system	256
22	Example 2	Simple variant for an 8 core system	258
23	Example 1	simple module usage	270
24	Example 1	Basic Pandas syntax	283
24	Example 2	Calculating overall averages	284
24	Example 3	Calculating minimum and maximum values	285
24	Example 4	Using the groupby method	286
25	Example 1	simple test program included with Tkinter distribution	290
25	Example 2	Hello world version 1	291
25	Example 3	Hello world variant 1	292
25	Example 4	Hello world variant 2	293
25	Example 5	Hello world version 2	293

25	Example 6	Hello world version 3	294
25	Example 7	simple button example	296
25	Example 8	Button and message example	297
25	Example 9	Button, message and entry example	298
25	Example 10	Button, entry and text widget example	300
26	Example 1	Simple trigonometric plot	307
26	Example 2	Enhanced trigonometric plot	316
26	Example 3	adding a legend, matplotlib defaults	317
26	Example 4	adding a legend with manual positioning	318
26	Example 5	Bar charts	319
26	Example 6	bar chart with standard deviations	321
26	Example 7	bar chart with 4 frequencies	322
26	Example 8	bar chart with 10 frequencies	325
26	Example 9	Mapping with Python 2.x and basemap	328
26	Example 10	tsunami plot using cartopy	334
26	Example 11	shifting the center of the map	341
26	Example 12	mapping using UK postcodes	345
27	Example 1	Python solution	351
27	Example 2	Fortran solution	352
27	Example 3	C++ solution	353
27	Example 4	Java solution	355
28	Example 1	testing the Nag library calls	359
28	Example 2	testing the Python random number generators	360
28	Example 3	Python native timing	360
28	Example 4	Nag timing	362

Note that some of the examples are not Python programs.

31.2 Chapter notes

Chapter 6 has two variants for c0606.py. Here is the diff output.

```
diff c0606.py c0606_1.py
6c6,8
```

```
< x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
---
> x = np.array([[1,2,3] ,
>               [4,5,6] ,
>               [7,8,9]])
```

i.e. there is a difference in the initialisation of the array.

```
diff c0606.py c0606_2.py
6c6,8
< x = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
---
> x = np.array([[1,2,3] , \
>               [4,5,6] , \
>               [7,8,9]])
```

where we use explicit continuation markers.

Chapter 7 has an extra example based on a later Unicode standard.

Chapter 27 has Fortran, C++ and Java source files.

The Nag examples in chapter 28 need a Nag library licence.