



Python Quick Start

A beginner's introduction
to Python programming

Martin McBride

Python Quick Start

A beginner's introduction to Python programming

Martin McBride

This book is for sale at <http://leanpub.com/pythonquickstart>

This version was published on 2021-01-31



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2020 - 2021 Martin McBride

Contents

1. Foreword	1
1.1 Who is this book for?	1
1.2 About the author	1
1.3 Keep in touch	1
2. Introduction	2
2.1 What is Python	2
2.2 History of Python	3
2.3 Zen of Python	3
3. Getting started with Python	1
3.1 Installing Python	1
3.1.1 Windows	1
3.1.2 Linux	1
3.2 Running IDLE	2
3.3 Simple maths in Python	3
3.4 Variables	3
3.4.1 Calculations with variables	4
3.4.2 Changing the value of a variable	4
3.4.3 Variable names	5
3.5 Strings	5
3.5.1 Strings in variables	5
3.6 Console shortcuts	6
4. Hello world	7
4.1 Writing a program	7
4.1.1 Running the IDLE console window	7
4.1.2 Opening an edit window	8
4.1.3 Typing in your code	8
4.1.4 Type, don't copy and paste	9
4.1.5 Running your code	9
4.1.6 Seeing the result	9
4.2 In case of errors	10
4.3 Print statements - outputting text	11

CONTENTS

4.4	Sequencing	11
4.5	Input statements - getting user input	12
4.5.1	Inputting a number	12
4.6	Selection	13
4.6.1	Indentation	14
4.7	Iteration	14
4.8	Comments	15
5.	Variables and values	17
5.1	Creating and using variables	17
5.2	Variables can store any type of data	18
5.3	Naming variables	18
5.3.1	Legal variable names	19
5.3.2	Reserved words	19
5.3.3	Choosing variable names	20
5.3.4	Naming conventions	20
5.3.5	Unicode characters	21
5.4	Variables point to values	22
5.5	None	22
5.6	Multiple assignments	22
6.	Numbers and maths	24
6.1	Types of number	24
6.1.1	Integers	24
6.1.2	Floating point numbers	25
6.1.3	Finding the type	25
6.2	Basic maths operations	26
6.2.1	Addition	26
6.2.2	Int and float results	27
6.2.3	Subtraction	28
6.2.4	Multiplication	28
6.2.5	Division	28
6.2.6	Power	29
6.2.7	Floor divide	29
6.2.8	Modulo	30
6.2.9	Unary operators	30
6.2.10	Augmented assignment	31
6.3	Precedence and brackets	32
6.3.1	Statements and expressions	33
6.4	Built-in maths functions	34
6.4.1	abs()	34
6.4.2	min() and max()	34
6.4.3	int()	34

CONTENTS

6.4.4	float()	35
6.5	math module	36
6.6	Limitations of floats	36
6.7	More advanced topics	36
6.7.1	Scientific notation for floats	36
6.7.2	Complex numbers	37
7.	Strings	39
7.1	Creating strings	39
7.2	String type	39
7.3	Operations on strings	39
7.3.1	Methods	39
7.4	Converting between strings and other data types	39
7.5	How to use special characters in a string	39
7.5.1	Quote characters	40
7.5.2	Unicode characters	40
7.5.3	Newline characters	40
7.5.4	The escape character \	40
8.	Loops	41
8.1	For loops	41
8.1.1	The loop variable	41
8.2	The range function	41
8.2.1	Changing the start value	41
8.2.2	Using a step value	41
8.3	For loop example - printing a times table	42
8.4	While loops	42
8.5	While loop example - getting user input	42
8.6	Nested loops	42
9.	Input and output	43
9.1	Input prompts	43
9.2	Inputting numbers	43
9.3	Print	43
9.4	Print separators	43
10.	If statements	44
10.1	if statements	44
10.1.1	Example if statement	44
10.2	Comparison operators	44
10.3	Else statement	44
10.4	Elif statement	44
10.5	Compound conditions	44
10.6	Comparison operator chaining	45

CONTENTS

10.7	Precedence	45
11.	Lists	46
11.1	What is a list?	46
11.1.1	Lists vs arrays	46
11.2	Creating lists	46
11.2.1	Empty lists	46
11.2.2	The list function	46
11.3	The list type	46
11.4	Accessing list elements	47
11.4.1	Reading an element	47
11.4.2	Changing an element	47
11.5	Operations on lists	47
11.6	Looping over a list	47
11.6.1	Doing calculations in a loop	47
11.7	List functions	47
11.7.1	Finding the length of the list	47
11.7.2	Finding the min a max values	48
11.7.3	Adding up the values in a list	48
11.8	List methods	48
11.8.1	Adding elements	48
11.8.2	Searching for elements in a list	48
11.8.3	Removing elements	48
11.8.4	Other methods	48
11.9	Two-dimensional lists	48
11.9.1	Ragged arrays	49
11.10	Aliasing	49
11.10.1	Reassigning variables	49
11.10.2	Does aliasing affect numbers?	49
11.11	Examples	49
11.11.1	Finding every element of a given value	49
11.11.2	Creating a 2-dimensional list of zeros	49
12.	Using functions	50
12.1	Why use functions?	50
12.2	Built-in functions	50
12.2.1	repr	50
12.2.2	chr and ord	50
12.2.3	help and dir	50
12.3	Importing modules	50
12.3.1	Simple import statement	51
12.3.2	Importing individual functions	51
12.3.3	Aliasing a module	51

CONTENTS

12.3.4	Importing all functions (don't do this)	51
12.4	How to call functions	51
12.4.1	Optional parameters	51
12.4.2	Variable number of arguments	51
12.4.3	Named optional parameters	51
12.5	Defining your own functions	52
12.5.1	Printing @s	52
12.5.2	Printing more @s	52
12.5.3	Returning a value	52
13.	More loops	53
13.1	The break operator	53
13.1.1	Break in a for loop	53
13.1.2	Break in a while loop	53
13.2	The continue operator	53
13.3	Using else with a loop	53
13.4	Nested loop statements	53
13.4.1	Printing times tables	54
13.5	Modifying a for loop	54
13.6	Looping in reverse order	54
13.6.1	Using range() to count backwards - ugly	54
13.6.2	reversed()	54
13.7	sorted()	54
13.8	Looping over multiple items	54
13.8.1	zip	54
13.8.2	How zip() works	55
13.9	More about zip()	55
13.10	Accessing the loop count using enumerate	55
13.11	Looping over selected items	55
13.11.1	Using filter	55
13.11.2	The advantage of using filter	55
14.	Programming logic	56
14.1	Boolean types	56
14.2	Comparison operators and their opposites	56
14.3	Boolean operations and De Morgan's Laws	56
14.4	Ternary operators	56
14.5	Comparing containers	56
14.5.1	Strings	56
14.5.2	Lists	57
14.5.3	Other containers	57
14.6	Membership testing	57
14.7	Identity testing	57

CONTENTS

14.8	Truthy values	57
14.8.1	Numbers	57
14.8.2	Strings	57
14.8.3	Lists	57
14.8.4	Other collections	58
14.8.5	None	58
14.8.6	Other objects	58
14.9	Short circuit evaluation	58
14.9.1	Short-circuiting with the or operator	58
14.9.2	Short-circuiting with the and operator	58
14.9.3	Short-circuiting to avoid errors	58
14.9.4	The value of an or/and expression	58
14.9.5	Getting a true or false value	59
15.	Slices	60
15.1	Slicing a list	60
15.2	Using negative indices	60
15.3	Delete or replace a slice of a list	60
15.4	Slicing tuples and strings	60
15.5	Loop over a slice	60
15.6	Using steps	60
16.	More strings	61
16.1	Raw strings	61
16.2	Multi-line strings	61
16.3	Looping over a string	61
16.4	Joining strings	61
16.5	Splitting a string	61
16.6	Formatting strings	61
16.7	The in operator	62
16.8	Other string methods	62
16.8.1	Changing case	62
16.8.2	Dealing with whitespace	62
16.8.3	Padding	62
16.8.4	Search	62
16.8.5	Splitting strings	62
16.8.6	Categorising strings	62
17.	Tuples	63
17.1	What is a tuple?	63
17.2	Creating a tuple	63
17.3	Accessing elements	63
17.4	Packing and unpacking tuples	63
17.4.1	Packing a tuple	63

CONTENTS

17.4.2	Unpacking a tuple	64
17.4.3	Real-life examples	64
17.5	Tuple vs list operations	64
17.5.1	Tuple slices	64
17.5.2	Adding elements to a tuple	64
17.5.3	Finding elements	64
17.5.4	Removing elements	64
17.5.5	Looping	65
17.5.6	In case of emergency	65
17.6	Pros and cons of immutability	65
18.	Exceptions	66
18.1	Program errors	66
18.2	What are exceptions	66
18.3	Exception types	66
18.3.1	ImportError	66
18.3.2	IndexError	66
18.3.3	TypeError	66
18.3.4	ValueError	67
18.3.5	ZeroDivisionError	67
18.4	Catching exceptions	67
18.4.1	Only catch exceptions you can handle	67
18.4.2	Accessing the exception message	67
18.5	Using else with exceptions	67
18.6	Using finally with exceptions	67
18.7	Throwing exceptions	67
19.	Working with files	68
19.1	Using files	68
19.1.1	Opening the file	68
19.1.2	Reading and writing data	68
19.1.3	Closing the file	68
19.2	Reading data	68
19.2.1	Code for reading a file	68
19.2.2	Reading lines from a file	69
19.2.3	Looping over the lines in a file	69
19.3	Writing data	69
19.3.1	Code to write a file	69
19.3.2	The write function	69
19.4	Using with statements	69
19.5	CSV data	69
19.5.1	Reading CSV data	69
19.5.2	Trying the code	70

CONTENTS

19.5.3	Formatting the output	70
19.5.4	Writing CSV data	70
19.5.5	Understanding the code	70
19.5.6	Trying the code	70
20.	More books from this author	71
20.1	Numpy Recipes	71
20.2	Computer Graphics in Python with Pycairo	72
20.3	Functional Programming in Python	72

1. Foreword

This book provides an introduction to the basics of programming in Python. After reading this book you should be able to create and understand Python scripts and simple programs.

1.1 Who is this book for?

This book is aimed at anyone wishing to learn Python, whether you are a complete beginner or have some basic experience and are looking to learn more. It requires no previous programming experience, as all the necessary programming concepts are explained as we go along. However, if you have previously programmed with other languages, this book will help you to transfer your existing skills to Python.

1.2 About the author

Martin McBride is a software developer, specialising in computer graphics, sound, and mathematical programming. He has been writing code since the 1980s in a wide variety of languages from assembler through to C++, Java and Python. He writes for PythonInformer.com and is the author of several books on Python. He is interested in generative art and works on the generativepy open source project.

1.3 Keep in touch

If you have any comments or questions you can get in touch by any of the following methods:

- Joining the Python Informer forum at <http://pythoninformer.boards.net/>¹.
- Signing up for the Python Informer newsletter at pythoninformer.com
- Following @pythoninformer on Twitter.
- Contacting me directly by email (info@axlesoft.com).

¹<http://pythoninformer.boards.net/>

2. Introduction

This book is intended for anyone who wants to learn to program in Python. It doesn't assume any previous knowledge of Python, or even of programming. The aim is to give you a good grounding in the basics of general programming in Python.

If you have programmed before in any other language, you can use this book to learn about the unique features that make Python special. If you have already dabbled with a bit of Python programming, this book provides a framework to consolidate and extend your knowledge.

Although Python is one of the easier programming languages for new programmers, it is extremely powerful and there is always more to learn. No one book can cover every aspect of Python, but this book will teach you all you need to know to tackle most problems in Python and to be able to read Python code.

This book uses Python 3.6 or later.

2.1 What is Python

Here are some of the key features of Python:

- **High level, general-purpose language** - whatever you want to code, from a simple script to back up your files, a website, a game, or an office suite, Python is a good choice.
- **Simple** - Python syntax is much simpler than many languages, although it is every bit as powerful.
- **Easy to learn** - you can start learning Python by typing code into the console, or by making a simple program of a few lines of code, and progress from there.
- **Free and open source** - while ever people still want to use Python, someone somewhere will be able to fix bugs, port to new platforms and add new features.
- **Portable** - Python is written in C, without doubt the most widely supported language in the world. If a platform has a C compiler (and most do) it can probably support Python.
- **Extensive libraries** - there are Python libraries for pretty much anything you might want your program to do.
- **Supports 3 styles of programming:**
 - **Procedural** - procedural programming uses expressions, if statements, loops and functions to create simple scripts and programs.
 - **Object Oriented** - Python allows you to use and create classes and objects. This is an advanced topic that is not covered in this book.

- **Functional** - functional programming means different things to different people, but Python supports it at various levels. This is an advanced topic that is not covered in this book.
- **Embeddable** - many programs these days are scriptable, allowing you to create scripts or macros to automate repetitive tasks. Increasingly, Python is being used as the scripting language, which is great. If you want to do something clever with Gimp or Inkscape, you can just create a Python script to do it.
- **Extensible** - you can extend Python quite easily. For example, it is generally quite easy to create a Python wrapper for a library written in C, to allow it to be used from Python.

2.2 History of Python

Python was developed by Guido van Rossum 1990. It was originally based on a language called ABC. It was influenced by C++, Haskell, Java, Lisp and Perl, amongst others. Of course, it drew its influence from those languages in 1990, which is a long time ago in computing terms. The languages Python is based itself on have all moved on in the intervening decades, and Python itself has been extended and modified over time.

Major releases of Python are quite infrequent. Version 2 was released in 2000, Version 3 was released in 2008, and at the time of writing Python 4 is expected to be released at some time before 2025.

Currently, Python 3 is the only officially supported version.

Python has nothing to do with snakes! Van Rossum is a fan of the British surreal comedy troupe *Monty Python*, and the language is named in their honour. The name of the built-in Integrated Development Environment (IDE) is IDLE, a wordplay on the name of Monty Python member Eric Idle.

2.3 Zen of Python

Tim Peters's Zen of Python sets out what Python is all about. Some of these things may mean something to you, some may not.

- Beautiful is better than ugly.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.

- In the face of ambiguity, refuse the temptation to guess.
- There should be one - and preferably only one - obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea - let's do more of those!

3. Getting started with Python

In this chapter, we will start by learning how to install Python on a Windows or Linux computer.

We will then use Python's built in console application, called IDLE, to interact with Python directly. This is a good way to get to know some of the basic things Python can do, before we move on to do some real programming. We will learn about:

- Installing Python.
- Doing simple maths in Python.
- Using variables.
- Creating text data (called strings in Python).

3.1 Installing Python

Python can be installed on most types of computer. You should install Python 3.6 or a later version of Python 3.

The official Python website is python.org, and that is where you can find installers for all supported versions.

If your operating system already has Python installed, it is worth checking the version, in case it is older than 3.6, in which case you should reinstall a later version.

3.1.1 Windows

For Windows, you can download a Windows installer file from python.org. You can run this installer to install Python, in the same way that you would install any other Windows application.

3.1.2 Linux

For Linux Ubuntu, you can install Python from the command line, using

```
sudo apt-get update
sudo apt-get install python3.6
```

You can use a later version if you wish.

For Debian based Linux, use:

```
sudo apt update
sudo apt install python3.6
```

For other flavours of Linux, you might need to use a different package manager.

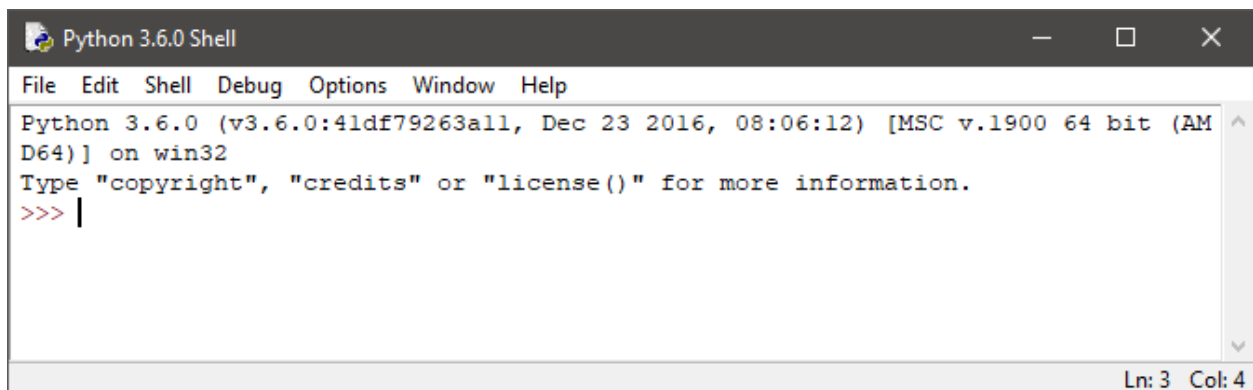
3.2 Running IDLE

Python comes with a simple, built-in user interface called IDLE.

IDLE contains two different types of window:

- **Console window.** This type of window allows you to interact directly with Python. You can type in commands, and Python will immediately run the command and print the result. We will be using this window in this chapter to perform simple mathematical operations (using Python as a calculator) and a few other tasks.
- **Editor window.** This type of window allows you to type in Python code. The editor understands Python syntax, so it will help you by automatically colouring your code as you type it in, to make it easier to read. After typing a program into this window, you can run the program and the result will appear in the console window.

Search for IDLE program in the program finder (eg Start Menu in Windows, Dash in Ubuntu). Make sure that you have chosen V3.6. Here is the window that should appear. It is an IDLE console window:



The version of Python is displayed in the title bar, and an initial message is displayed on the screen. You can type instructions into this window. We will show things you type in like this:

```
>>> 3 + 2
```

This means that you should type `3 + 2` into the window, and hit return. Don't type in the `>>>`, that is just the prompt Python displays! We will show Python's response like this, without the prompt:

5

When you type in $3 + 2$, Python's response is to calculate the result (which is 5 of course), and display it back to you in the console window.

Here is another example, showing the complete transaction:

```
>>> 10 * 3
30
```

Here, you have typed in $10 * 3$ (10 multiplied by 3) followed by a return, and Python has replied with 30.

3.3 Simple maths in Python

As we have seen, when you type in some maths in the IDLE console window, Python evaluates it and displays the answer. Here are some more examples. In each case, type in the request (the letters after the `>>>`) and you should see Python print the response (shown in italics). Try these:

```
>>> 1 + 2 + 4
7
>>> 1 + 2 - 4
-1
>>> 1 - 2 + 4
3
```

These are some simple additions and subtractions. Nothing too complicated here.

You can also multiply and divide. Try these:

```
>>> 3 * 5
15
>>> 3 / 2
1.5
```

Python uses `*` for multiply and `/` for divide, similar to most other programming languages (and spreadsheets).

Python has a full set of maths operators (and also maths functions such as sine and square root) that we will look at in a later chapter.

3.4 Variables

A variable is a way to store a value to use later in your program. Try this:

```
>>> a = 3
```

This creates a variable called `a` and gives it a value 3. Python doesn't print a response to this, it just stores the value.

We call this type of operation an *assignment*. We say that we are *assigning* the value 3 to the variable `a`.

Now try this:

```
>>> b = 6
```

This creates another variable called `b`, and assigns it a value 6.

You can find the value of a variable by typing its name:

```
>>> a
3
>>> b
6
```

3.4.1 Calculations with variables

You can use a variable in a calculation, instead of a value. The current value of the variable will be used.

In this example, we add `a` to 2. Python gets the value of `a` and adds 2 to it - since `a` has previously been set to 3, the result is 5:

```
>>> a + 2
5
```

In the next example, we multiply `a` and `b`. Again, Python fetches the values of `a` and `b`. Since these variables have previously been set to 3 and 6, the result is 18.

```
>>> a * b
18
```

3.4.2 Changing the value of a variable

We can change the value of a variable like this, which sets `a` to 8:

```
>>> a = 8
```

This is similar to the code above, where we assigned 2 to `a`. But this time, the variable `a` already exists, so Python doesn't need to create the variable, it just uses the existing variable and replaces its value with a new value 8.

We can combine a calculation with an assignment. This line calculates `a + 3`, which will be 11 of course, because `a` has the value 8. It assigns the value 11 to `b`:

```
>>> b = a + 3
```

If we repeat the original calculations, we will get a different result this time, because the variables have different values, Try this:

```
>>> a + 2
10
>>> a * b
88
```

3.4.3 Variable names

Python has rules about variable names, as we will see in the later chapter on *Variables*.

For now, we will stick to letters like `a`, `b`, `c`, or short words like `name` or `date`, all in lower case. There are a few names you can't use because they have special meanings in Python: `if` and `for` aren't allowed, for example. We will look at this in more detail later on.

3.5 Strings

Python doesn't just work with numbers. It can also handle text data. A piece of text is called a *string* in coding jargon. A string is created using quote characters:

```
>>> 'Hello'
'Hello'
```

When you enter the value `'Hello'`, Python responds by repeating the value you typed in.

3.5.1 Strings in variables

You can store a string value in a variable:

```
>>> a = 'Hello'
```

You can retrieve the value as we saw before:

```
>>> a
'Hello'
```

3.6 Console shortcuts

The variable `_` (just an underscore character) has a special meaning in the console. It represents the result of the most recent operation. For example:

```
>>> 10 + 2
12
>>> _ + 3
15
```

In the first line, `10 + 2` gives the result 12.

In the second line, `_` represents the value of the previous result, which is 12 in this case. So `_ + 3` gives the result 15.

You can also repeat a previous line without retyping it. To do this:

- Move the text cursor to the line you want to replicate (using the up arrows, or clicking with the mouse).
- Hit Enter.

The line will be duplicated as if you had retyped it. You can then edit it if you wish, then hit Enter again to execute the line.

4. Hello world

The traditional first program you write when learning any programming language is a ‘Hello, world!’ program, that simply prints a message on the screen.

In this chapter, we do a bit more than that. We will give a quick tour of several important aspects of programming:

- Output - allowing your program to communicate with a user (using *print* statements).
- Sequencing - getting your program to do several things, one after another.
- Input - allowing your program to accept input from the user (using *input* statements).
- Selection - letting your program decide whether to do a particular thing or not (aka *if statements*).
- Iteration - getting your program to do the same things, over and over (aka *loops*).

This might seem like a lot to cover in one chapter, but these are the main aspects you need to make a simple program that does useful things. Nailing these concepts now will mean that we can use more realistic and interesting examples as we explore more features of Python.

We will only look at the simplest, most basic cases. All these topics will be revisited in later chapters.

4.1 Writing a program

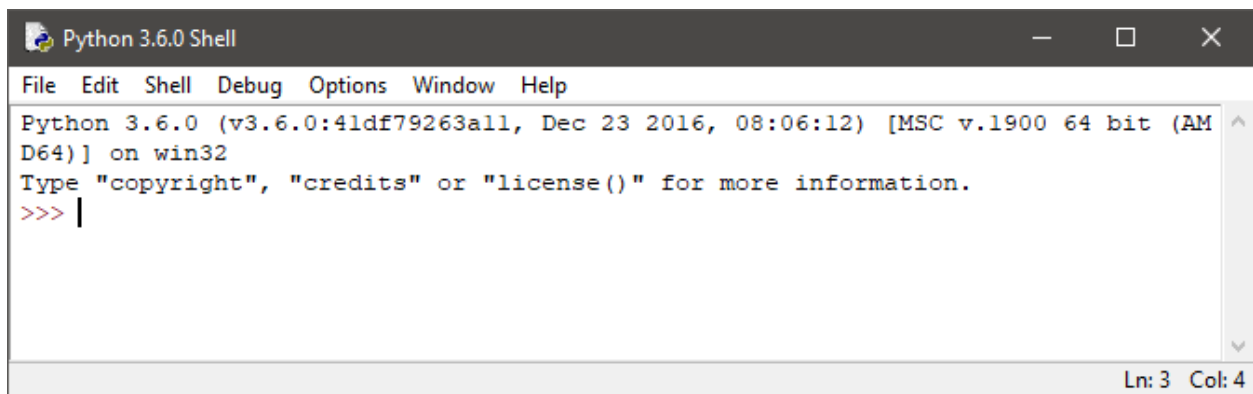
So far we have learnt to interact with Python using the IDLE **console window**. We have used that to perform simple calculations using Python. That is a useful start, but it isn’t really programming.

To write a program, we need to make use of a second type of window in IDLE, the **edit window**.

4.1.1 Running the IDLE console window

We have already seen how to run the IDLE **console window**, in the previous chapter *Getting started with Python*.

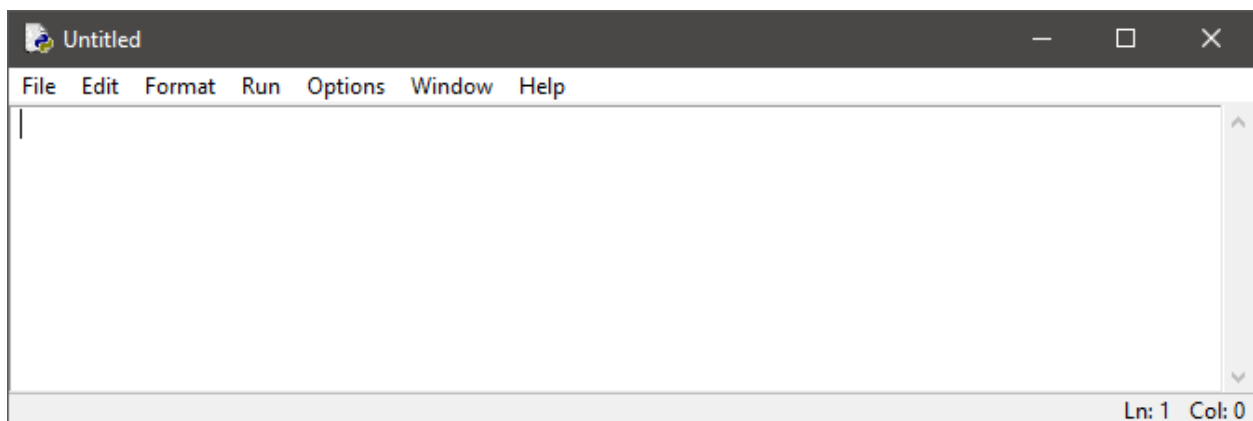
Open up the IDLE **console window** as before:



4.1.2 Opening an edit window

IDLE has a second type of window called an **edit window**. This allows you to type code in, then save it and run it.

From the IDLE **console window** you should choose the **File | New File** menu item. This will open the new editing window:



You will normally keep both the **console window** and **edit window** open, and switch between them as needed:

- The **edit window** is where you type in the code you want to run.
- The **console window** is where you look to see any output or error messages.

4.1.3 Typing in your code

We are now going to write a *Hello, world!* program - a simple program that outputs the message *'Hello, world!'*.

Here is the program. Type it in, exactly as shown, in the **edit window**.

```
print('Hello, world!')
```

When you type the code into the editor, you will notice that some parts of the text are in different colours. This is called *syntax highlighting*, and it is something the editor window does automatically to help make your code easier to read. The editor understands the Python language and can show different parts (such as numbers and strings) in different colours to make them stand out.

It is important to type in the code exactly as shown, taking account of spelling and capital letters (typing `Print` instead of `print` won't work, because of the capital `P`). Also, don't add any spaces at the start of the line, or try to split the code over two lines. This will all become quite natural after you have practised a little.

This program only has one line of code. It uses a *print statement* to display some text in the **console window**.

The text is stored as a string value. We met strings in the previous lesson, a string is just some text enclosed in single quote characters. The single quote is sometimes also called an apostrophe. Here is an example of a text string:

```
'text'
```

4.1.4 Type, don't copy and paste

Most of the examples in this book are quite short. It is better to actually type the code into your editor rather than copying and pasting it. Typing the code will help cement the syntax rules so that they will be natural and automatic to you when you start to write your own code.

4.1.5 Running your code

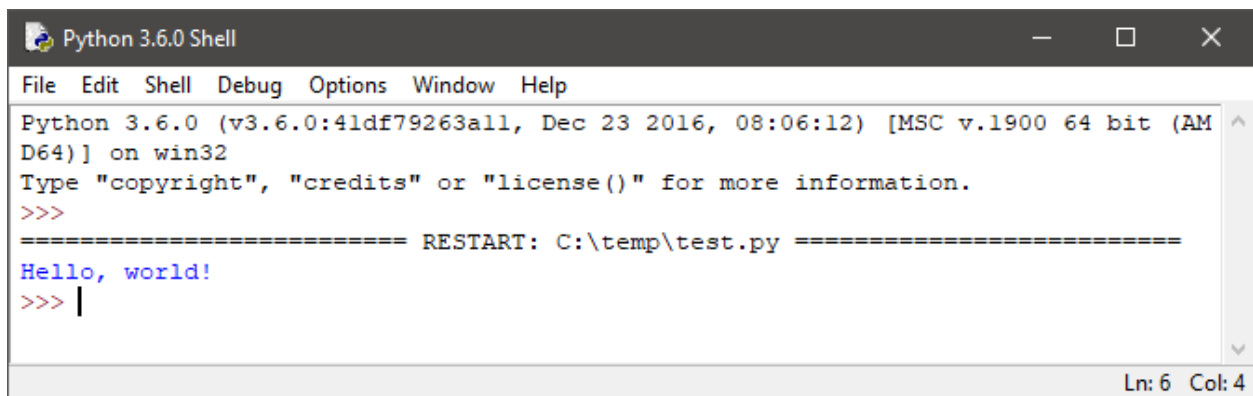
You can now run your program.

1. Select the **Run | Run Module** menu item (or just use the **F5** shortcut key).
2. Before the program runs, you will be asked to save your code to file. Save it wherever you like - for example, you could save it as *hello.py* in the *c:\temp* folder.
3. Then your code should now run. You can switch back to the console window to see the output.

If you get an error message, see the hints later on.

4.1.6 Seeing the result

You should now look the **console window**, where Python displays its results:

A screenshot of a Python 3.6.0 Shell window. The window has a title bar with the text 'Python 3.6.0 Shell' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following output: 'Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32', 'Type "copyright", "credits" or "license()" for more information.', and a prompt '>>>'. Below the prompt is a line of text '===== RESTART: C:\temp\test.py =====' followed by the output 'Hello, world!' in blue text. The prompt '>>>' is followed by a cursor. At the bottom right of the window, the status bar shows 'Ln: 6 Col: 4'.

Ignore the RESTART message, Python prints that each time it runs a new program. It just tells you that Python has thrown away anything you did previously (such as importing modules or defining variables) and is running your program with a clean slate

Our output text is displayed in blue underneath – it is the text from our print statement:

Hello, world!

4.2 In case of errors

Like most programming languages, Python expects a program to obey exact rules (called the Python syntax). If not, it will give an error. If your program gave an error instead of the expected result, here are some things to check:

- You must use Python version 3.6 or later. Earlier versions might not run this code.
- The code must be exactly as shown – any differences or missing characters can cause an error.
- `print` must be at the start of the line (there shouldn't be any spaces, tabs or anything else before the word `print`).
- The code should all be on one line, not split over two lines.
- `print` must be in lower case. For example, `Print` isn't valid and will give an error.

For certain errors, Python will pop up an error window to tell you of the error. It will also mark the line where the error occurred, in the edit window.

For other errors, Python will print out an error message, in the console window. This can be useful in helping you find the error (although sometimes it can be a little cryptic). It will tell you which line the error is on, so you can look back at the edit window to try to find it.

4.3 Print statements - outputting text

Our code above uses a `print` statement. This is simply a line that uses the `print` function.

A function is a piece of code that you can *call* to do a specific thing. The `print` function displays text on the console window.

A function name (`print` in this case) is always followed by a pair of parentheses `()`.

```
print('Hello, world!')
```

In this case, the string value `'Hello, world!'` is included within the parentheses of the `print` function. We call this value the *argument* of the function call (an argument is sometimes called a parameter - it means exactly the same thing).

In this case, `print` has one argument, the string that will be printed.

Functions can have more than one argument, for example:

```
print('Hello, world!', 'with 2 arguments')
```

This time we are passing two separate strings into `print`. The two arguments are separated by a comma. When `print` is passed more than one argument, it will print all on them, on one line, separated by a space. The output in this case would be:

Hello, world! with 2 arguments

You can also call `print` with no arguments. In that case, it prints an empty line:

```
print('Hello, world!', 'with 2 arguments')
```

`print` is called a *built-in* function, which means you don't need to import any module to use it, it is always available.

4.4 Sequencing

Now we are going to write a program that does several things, one after another. This is called *sequencing*, and it is one of the most basic things programs do.

We will print three lines of text. This is a very simple program, it just contains a list of instructions. Python will execute the instructions, one at a time, in the order they are written.

```
print('This is my first python program')
print('that has more than one line!')
print('Two plus two is', 2 + 2)
```

Then we print some messages. The third message has two arguments. The second argument is a mathematical expression $2 + 2$. Python calculates this and passes the value 4 into the `print` function.

Run your program. It should display something like this in the console window:

```
This is my first python program
that has more than one line!
Two plus two is 4
```

4.5 Input statements - getting user input

An input statement prints a line of text in the console, and allows the user to type in a text response.

Here is a simple program using input:

```
name = input('What is your name?')
print('Hello', name)
```

The first line calls the `input` function, which displays the question 'What is your name?' and waits for the user to type an answer.

If you type your name in and hit Enter, the `input` function returns a string containing your name (or whatever else you typed in). The string is stored in the variable `name`.

The second line then runs, printing 'Hello' followed by your name.

4.5.1 Inputting a number

Here is a neat trick that you can use to input a number. Don't worry too much about how it works for the moment, we will cover that later in the book.

```
age = int(input('How old are you?'))
print('You are', age, 'years old')
```

This time the question asks for your age. However, the `input` call is wrapped in a call to the `int` function. This converts the input string to a number. So if you type in the string 25 the variable `age` will be set to the *number* 25.

Make sure you type in a number. If you type in 'xyz' you will get a program error (we will see how to deal with that later in the book).

This code asks two questions, and gives one combined reply:

```
name = input('What is your name?')
age = int(input('How old are you?'))
print('Hello', name, 'you are', age, 'years old')
```

4.6 Selection

An `if` statement allows us to decide whether particular lines run, based on some *condition*. Here is an example:

```
number = int(input('Input a number less than 10'))

if number < 10:
    print(number, 'is less than 10')
    print('Well done!')

print('Goodbye')
```

What does this code do?

1. It asks the user to input a number that is less than 10, and stores the result in the variable `number`.
2. Uses an `if` statement to check if `number` really is less than 10.
3. *If the condition is true* that `number < 10`:

* Prints a message confirming that the number is less than 10.

* Prints a slightly sarcastic message of congratulation.

4. *Always* prints goodbye.

So as you can see, the `if` statement includes a *test*, in this `number < 10`. If the test is true, it will run the *body* of the `if` statement. The body consists of the two indented print statements. Those statements will only run if the test is true.

The final line of the program is not indented, so it is not part of the `if` statement body. It will always run, whether the test was true or not.

So if you type in 3, the program will output:

```
3 is less than 10
Well done!
Goodbye
```

But if you type in 11, the program will output:

Goodbye

Here is a diagram that shows the parts of the program:

```
number = int(input('Input a number less than 10'))  
  
if number < 10: if statement  
    print(number, 'is less than 10')  
    print('Well done!') if body  
  
print('Goodbye')
```

The body of the `if` statement is sometimes called the *if block* or the *if clause*. It means the same thing.

4.6.1 Indentation

As we saw, the `if` statement body is indented.

We have indented the code by adding four spaces at the start of each line.

You can use any number of spaces, it doesn't have to be four. Two or three are also quite common (one space doesn't really make the indentation that noticeable, and to add more than four spaces is just a bit excessive). The important things are:

- You must use the same number of spaces for each line in the body.
- The next line after the end of the body must return to the original indentation (no spaces).

4.7 Iteration

Iteration means automatically repeating the same actions multiple times. This is called looping.

Python has two different types of loop - `for` loops and `while` loops. We will just look at the simplest type of `for` loop here, but there will be much more about loops later in the book.

Here is a simple `for` loop:

```
print('Start')

for i in range(5):
    print('Next number:')
    print(i)

print('End')
```

The loop is controlled by the line

```
for i in range(5):
```

The range function creates a *sequence* of values. The parameter, 5 in this case, controls the endpoint of the sequence. The sequence runs from 0 *up to but not including* 5. This means that the sequence has five values: 0, 1, 2, 3, and 4.

The loop runs five times, which means the body of the loop runs five times.

The variable `i` is called the loop variable. Each time through the loop it takes the next value in the loop sequence. So the first time through the loop, `i` is set to 0. The second time it is set to 1, and so on.

Here is a diagram that shows the parts of the loop:

```
print('Start')

for i in range(5):
    print('Next number:')
    print(i)

print('End')
```

Diagram labels:

- `for i in range(5):` is labeled **loop statement** (highlighted in blue).
- `print('Next number:')` and `print(i)` are labeled **loop body** (highlighted in pink).

4.8 Comments

Comments allow you to add explanations or descriptions to your code. Python itself completely ignores comments, but they can be very useful to other programmers who are trying to understand

your code. They are often even useful to the person who wrote the code in the first place, if they need to make changes some months later.

Adding comments to Python is very simple. If you add a hash character #, Python will ignore the hash character and everything that comes after it in that line.

This can be used in two ways. Block comments are placed on their own line, for longer descriptions. They will often be several lines long. Inline comments are added at the end of a line of code, to give a quick description of just that line. Here are some examples:

```
# This is a block comment.  
# It is two lines long.
```

```
x = 3    # This is an inline comment.
```

When Python runs this file, all it will execute is:

```
x = 3
```

5. Variables and values

In Python, *variables* provide a way for your program to store values, so you can use them later.

In this chapter and most of the later chapters, we will be using small Python programs as illustrations. You can (and should) try these programs out by running them in the IDLE editor as described in the previous chapter. Most of the examples are quite short and there is considerable benefit in actually typing the code in (rather than copying and pasting) as that will ensure that you don't skim over any important lines of code.

Type the code into an editor window and observe the results in the console window, in a similar way to the previous chapter.

5.1 Creating and using variables

This code creates several variables:

```
x = 3
name = 'John'
weight = 5.3
```

The first thing to know is that, in Python, the = sign doesn't mean quite the same thing as it does in maths. It actually means *assign*.

In maths, $x = 3$ states a *fact*, that x is equal to 3. In Python, $x = 3$ is an *instruction*. It tells Python to create a variable called x , and set its value to 3.

So, in the code above we do this:

- Create a variable called x and assign it an integer value 3.
- Create a variable called `my_name` and assign it a text string value John.
- Create a variable called `weight` and assign it a floating-point value 5.3.

We can create variables with pretty much any name we like (there are rules, as we will see later). When you are writing code you should choose names that describe what the variable is being used for, it will make your code much easier to understand.

When you use a variable in your code, Python fetches its value and uses it:

```
print(name)          # Prints 'John'

total = x + weight   # total set to 8.3
```

You can also set a variable to a new value:

```
name = 'Eric'
weight = 7.2
x = 'abc'
```

When you assign a value to a variable that already exists, Python doesn't create a new variable. Instead, it assigns a new value to the existing variable. So after running the code above, `name`, `weight` and `x` all have new values.

But be aware that when you do this, the old value is gone forever.

5.2 Variables can store any type of data

One interesting thing about the previous code is that the variable `x` initially held the value 3 (an integer), but we then assigned a new string value 'abc'. That is, data of a different type.

That is perfectly OK in Python. Any variable can hold any type of data. It can be assigned a number at one time, then be assigned a string later on.

But just because you can doesn't mean you should. As you will see from the examples in this book, each variable is usually used for a specific purpose, because that makes your code a lot easier to understand.

For example, the variable `name` we declared earlier should be used to hold a name, which means it will almost certainly contain string data. If your code also needs to process someone's age, you shouldn't reuse the `name` variable to hold the age. That would be confusing on two levels, firstly there would be an integer in a variable you might expect to hold a string, and secondly, why would someone's age be stored in a variable called `name`?

5.3 Naming variables

Python has particular rules that all variable names must follow. Names that break these rules are not valid in Python, so your program will give an error if you try to use them.

Beyond that, there are also some common sense rules about how you should choose names variables. If you break these rules, your code will still work, but it might not be as easy to understand.

5.3.1 Legal variable names

Python variables can consist of one or more characters from the set of:

- Alphabetic characters (a to z and A to Z).
- Digits 0 to 9 (but the first character cannot be a digit).
- The underscore character `_`.

In addition, Python contains several reserved words - words that form part of the language - that you cannot use as variable names. Reserved words include `if`, `for`, `while`, and several others, see below.

Valid variable names include:

- `x`
- `maximum_value`
- `level3`

Invalid names include:

- `1st_value` - can't start with a number.
- `maximum-value` - the `-` character isn't allowed.
- `return` - this is a reserved word.
- `level.3` - the `.` character isn't allowed in a variable name.

5.3.2 Reserved words

Reserved words are used by the Python language, so you cannot use them as variable names.

The following is a list of reserved words in Python 3.6:

`'False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def',
'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import',
'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',
'while', 'with', 'yield'`

The following program will print a list of all the reserved words in the version of Python you are using:

```
import keyword  
print(keyword.kwlist)
```

Python also has many built-in functions (for example, `print`). You *can* use these names as variables, but it isn't generally a good idea, because that will mean that you can't use that built-in function any more. For example:

```
print = 3
x = print*2
print(x)
```

This code assigns a value 3 to the variable `print`. It then assigns `print*2` to the variable `x`. That all works fine, `x` is equal to 6 as you would expect.

The problem is that if we try to print `x` it won't work. `print` no longer represents the built-in `print` function, `print` is not set to 3. So when we try to print `x`, what we are telling Python to do is `3(x)`, which makes no sense.

There are quite a lot of built-in functions. We will meet many of them in this book, but you can also refer to the documentation on the python.org website.

5.3.3 Choosing variable names

In general, it is best to use variable names that describe the value that the variable will hold, so for example `first_name`, `maximum_value`, `current_time` and so on. This makes your code more readable.

For similar reasons, you shouldn't normally use single-character variable names such as `a`, `b` etc, or pointlessly abbreviated names such as `fst_nm`. It just makes your code more difficult to understand. However, there is no absolute rule against using single-letter variable names if they are the clearest names. For instance, if you are dealing with `(x, y)` coordinates, it is fine to call them `x` and `y`, what else would you call them? Also in some of the examples in this book we will use single letter variables because some are so simple and generic there isn't really any obvious descriptive name.

Having said that, a longer name isn't always a better name. Avoid pointless extras such as `the_`, `height` or `my_password`, or including the type of data such as `name_string`. And generally avoid extremely long variable names, because they also make code difficult to read. There are no fixed rules, but if a variable name is longer than about 20 characters it is worth considering whether you could choose a shorter name that is just as descriptive.

5.3.4 Naming conventions

Python has a couple of *naming conventions*. These aren't absolute rules, but they are things that a lot of Python programmers do.

If you have a variable name that contains two or more words, you can use either of the following ways to make the word distinct:

- `first_name` uses an underscore between the different words. This format is sometimes called *snake case*.
- `firstName` uses a capital letter at the start of each word (except the first). This format is sometimes called *camel case*.

The official Python style guide recommends snake case, but there are plenty of developers who use camel case instead. You should avoid mixing then, as that just creates confusion.

Python often treats names that *start* with an underscore as “private”. If you are sharing your code with other developers, the initial underscore is a sign that they shouldn’t do anything with that variable because your code is using it internally for something important. Due to this convention, it is best not to start your normal variable names with an underscore because that could confuse other people reading your code.

The variable `_`, that is a variable whose name is just the single character underscore, is usually reserved to indicate a variable that isn’t used anywhere. We will see examples later, but don’t name an ordinary variable as `_` because again it will confuse anyone else who tries to understand your code.

5.3.5 Unicode characters

Python allows you to use Unicode characters in variable names. For example, these are valid Python variable names:

- π (the Greek letter Pi).
- prénom (Contains an accented character).

You can use any character that is considered to be a letter or number in the Unicode definitions. Pure symbols are not allowed, so you can’t use smiley faces in your variable names!

There are a few potential pitfalls with using these characters, especially if you are sharing your code with other people:

- Other programmers might not know how to enter non-ASCII characters - if their keyboard doesn’t support a character directly they will have to enter it as a numeric code, or cut and paste it from elsewhere.
- Some computers might not be able to display certain characters. Most modern computers will support Greek characters and accented characters, but for more obscure characters a special font might be required.
- Some editors might not handle non-ASCII characters properly. For example, if you double click the string prénom in IDLE, it treats pr and nom as if they were separate words.
- Other programmers might not even know the name of the letter. While most people know recognise π , not everyone recognises the Greek letter ξ , so they would not even know what to look up.

5.4 Variables point to values

We said at the start of this chapter that variables are used to store values.

While that is usually a good way to imagine it, it isn't entirely true. In fact, values are stored in memory, and a variable simply points to that value in memory. We say that the variable *references* the value.

This doesn't usually matter, but sometimes you have a situation where two variables reference the *same* object in memory. This is called *aliasing*, and we will discuss it more in the chapter on *Lists*.

5.5 None

The only way to create a variable in Python is via an assignment statement. But what happens if we want to create a value but we don't yet have a value to set it to?

One solution is to use a dummy value, such as `0` or `''` (an empty string). But if we find a variable with value `0` how do we know if it has been deliberately set to `0`, if it just has a value `0` because it hasn't been set at all?

Python provides a value `None` which is specifically intended to indicate "no value". You can use it like this:

```
x = None
```

Of course, `x` does have a value, it has the value `None`, but that indicates that we should treat it as if it doesn't have any specific value.

5.6 Multiple assignments

You can perform more than one assignment in a single statement, for example:

```
a, b = 1, 2
print(a, b)      # Prints 1 2
```

Here the values `1` and `2` are assigned to the variables `a` and `b` respectively. This can be used to initialise several variables on one line, although it is usually better to avoid that. Assigning each value individually is generally easier to read.

Where this does come in quite useful is if you want to swap two values. You might naively try to swap values like this (assuming `a` is `1` and `b` is `2`):

```
a = b
b = a
print(a, b)      # Prints 2, 2
```

This doesn't work properly, because the line `a = b` sets `a` to 2 but loses the original value of `a`. Then the line `b = a` just copies 2 back to `b`. It doesn't swap them, they both end up set to 2.

But you can do this (again assuming `a` is 1 and `b` is 2):

```
a, b = b, a
print(a, b)      # Prints 2 1
```

This works because Python first *packs* the values of `b` and `a` into a structure called a *tuple*. The pair `(2, 1)` is assigned in one go. The tuple is then *unpacked* into the variables `a` and `b`, so `a` gets set to 2 and `b` gets set to 1. We will cover this in the chapter on *Tuples*. For now, just treat this code as a *recipe* for swapping two values.

6. Numbers and maths

In this chapter, we will look at how Python stores numbers and performs mathematical calculations. We will cover:

- The two different types of number Python uses, and their characteristics.
- Basic mathematical operators (addition subtraction, multiplication etc).
- Some standard mathematical functions in (such as min and max, sine, square root and so on).
- A brief look at a few more advanced topics.

6.1 Types of number

Python uses two different types of number:

- Integers - these are whole numbers such as 10, 0, -3 etc.
- Floating-point numbers - these are numbers with a fractional part, such as 3.14 or -1.5.

Python also supports a third type, complex numbers, which are far less common, but we will discuss them briefly at the end of this chapter.

6.1.1 Integers

Integers, often called *ints* in programming, are whole numbers. They include:

- The counting numbers 0, 1, 2, 3 ...
- Negative whole numbers -1, -2, -3 ...

You should always use ints whenever you know for certain that a number has to be an integer, for example:

- For counting. For example, the text string 'abc' has 3 characters. The number of characters has to be an integer, you can't have a string that contains three and a half characters.
- For indexing. For example, in the text string 'abc', the letter 'a' is at position 0 (Python counts positions starting from 0). The letter 'b' is at position 1. The letter 'c' is at position 2. We can't ask which letter is at position one and a quarter, that makes no sense.

In these cases, and many others, the value definitely must be a whole number, so you should store it as an int value. This is very important because Python stores ints as *exact values*, so there can never be any confusion or error (this is different for floats, as we will see).

You create an int value by writing the number with no decimal point, for example:

```
i = 5
```

This creates an int with value 5 and assigns it to the variable `i`.

Certain Python functions automatically return ints. For example:

```
size = len('abc')
```

Here we use the Python function `len` that returns the length of the string `'abc'`. This function will return an int value of 3, which gets stored the variable `size`.

Python allows you to create very large integers. It doesn't have a specific limit, but it can handle integers with thousands of digits, with no problem.

6.1.2 Floating point numbers

Floating-point numbers are often called *floats* in programming, or (less commonly) *reals*.

A float is a number with a fraction part, including:

- Positive decimals like `1.234`.
- Negative decimals like `-5.932`.
- Decimal whole numbers like `2.0`, `0.0`, `-4.0`.

Floats are typically used to stored measured quantities like length, weight, temperature.

Floats are stored in a different way to ints. Even for whole numbers, `2.0` is stored differently to `2`. You don't normally need to worry about this, because if you do a calculation the uses both int and float values, Python will automatically convert the values so the calculation works correctly.

But it is useful to bear in mind that floats only store numbers up to a maximum of about 16 significant figures (in a similar way to a pocket calculator). This means that some numbers cannot be represented exactly (but they are very close). For example `1.0` divided by `3.0` will give a value of `0.3333333333333333`.

Floats have a maximum range of about 10 to the power 308 (that is, a one followed by 308 zeros), so they are plenty large enough to cope with most numbers that you will ever need in a typical computer program.

6.1.3 Finding the type

You can find the type of a number (or any other data), using the `type` function, like this:

```
a = 10
t = type(a)
print(t)

b = 1.2
print(type(b))
```

The first part creates a variable `a` with value `10`, then gets its type, which it stores in variable `t`. It then prints `t`:

```
<class 'int'>
```

The important part here is `'int'`, that shows the data is of type `int`. The second part creates a variable `b` with value `1.2`. This time the code finds the type of `b` and prints it, without storing the type in a variable. It prints:

```
<class 'float'>
```

6.2 Basic maths operations

Let's look at some basic maths operations. Here is a diagram that summarises the operations:

$x + y$	Add	$x ** y$	Power
$x - y$	Subtract	$x // y$	Floor divide
$x * y$	Multiply	$x \% y$	Modulo
x / y	Divide		

6.2.1 Addition

This code shows a few examples of addition:


```
a = 10
b = 4

c = a + b          # c is set to 14
d = a + a + 7.5    # d is set to 27.5
print(b + 9.0)     # prints 13.0
```

After declaring `a` and `b`, the next line adds `a` (which has the value 10) and `b` (which has the value 4), giving a result of 14, which is assigned to the variable `c`.

The line after adds `a`, `a`, and 7.5 to give the value 27.5, which is assigned to the variable `d`.

The final line adds `b` to 9.0. The result isn't assigned to a variable. Instead, we print it.

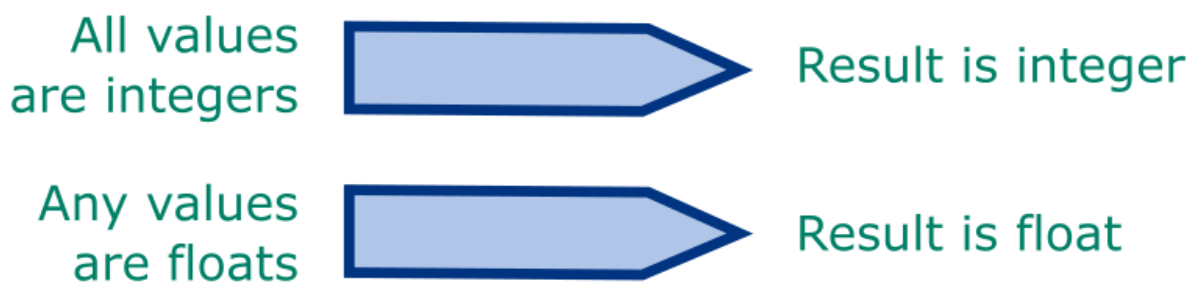
6.2.2 Int and float results

As you can see, the result is sometimes an int value, sometimes a float value. Here is the basic rule that covers *most* cases:

- If *any* of the values involved in the calculation are floats, the result will be a float.
- But if *all* of the values are ints, the result will be an int.

Notice that `b + 9.0` gives a float result 13.0. Even though the result is a whole number, it is still created as a float because one of the inputs, 9.0, is a float.

There are some exceptions to this rule that we will meet later - the rules are different for division, power and some maths functions.



Exceptions:

- divide result is always float
- power result is float if $y < 0$

6.2.3 Subtraction

This code shows some examples of subtraction:

```
a = 10
b = 4

c = a - b          # c is set to 6
d = a + a - 7.5    # d is set to 12.5
print(b - 9.0)     # prints -5.0
```

This code is very similar to the previous addition code but uses subtraction instead.

6.2.4 Multiplication

Multiplication uses the symbol `*` (asterisk). Here are some multiplication examples:

```
a = 10
b = 4

c = a * b          # c is set to 40
d = b * b * 0.1    # d is set to 1.6
print(a * 2.0)     # prints 20.0
```

Once again, if all the numbers in the expression are integers, the result will be an integer, but if any of the numbers are floats the result will be a float.

6.2.5 Division

Division uses the symbol `/` (forward slash). Its behaviour is different to the previous operators, because division *always produces a float result*:

```
a = 10
b = 4

c = a / b          # c is set to 2.5
print(a / 2)       # prints 5.0
```

Even if both values are ints, the result is always a float. Even if the numbers divide (for example `10 / 2`) the result is still a float.

If you have learnt other languages before, this might surprise you. Several common languages will do an integer division if both numbers are integers. For example:

```
## This is how C++ and Java work, but NOT Python
10/5    # Gives 2 (5 divides into 10 2 times)
10/4    # Gives 2 (4 divides into 10 2 times, with 2 left over)
```

In Python, if you want this behaviour you need to use *floor divide* (see later).

6.2.6 Power

The power operator uses the symbol `**` (two asterisk characters). It raises a number to a power.

```
a = 2
b = 3

c = a ** b      # c is set to 8
print(a ** 4)   # prints 16

d = 2 ** -1     # d is set to 0.5
e = 9 ** 0.5    # e is set to 3.0
```

If you are not familiar with power is maths, `x ** y` means `x` multiplied by itself `y` times. For example:

- `x ** 2` means `x * x` (sometimes called *x squared*).
- `x ** 3` means `x * x * x` (sometimes called *x cubed*).
- `x ** 4` means `x * x * x * x`.

You can use negative values as powers too. `x` to the power `-1` is equivalent to $1/x$. So `2 ** -1` is equivalent to $1/2$, which gives a result of `0.5`.

Finally, you can use fractional values as powers. For example `x ** 0.5` calculates the square root of `x`. So `9 ** 0.5` is `3.0` because 3 is the square root of 9.

Powers follow the normal rules that the result is an int if both values are ints, with one exception. If the second parameter is negative, the result will always be a float (because as we saw above, a negative power implies division).

Of course, if one or both of the values is a float, the result will also be a float.

6.2.7 Floor divide

Floor divide is a type of integer division, similar to the way it is usually taught in the early school years as *share by* and *remainder*. It uses the symbol `//` (two forward slash characters).

This expression:

$x // y$

calculates a whole number that represents how many times you can subtract y from x without leaving a negative value. So for example, $6 // 3$ gives 2 because 3 divides into 6 exactly 2 times.

As another example, $9 // 2$ gives 4. 2 doesn't divide into 9 exactly, but it divides 4 times with a remainder of 1.

The more formal definition of $x // y$ is:

- Divide x by y .
- Round the result *down* to a whole number.

If both numbers are ints, the result will be an int.

If one or both numbers are floats, the result will still be a whole number, but a float whole number. For example $6.0 // 3.0$ is 2.0 .

You need to be careful with negative numbers. For example $-9 // 2$ gives -5 (not -4 as you might expect). This is because -9 divided by 2 is -4.5, but when we round that down to a whole number, it is rounded to -5.

6.2.8 Modulo

Modulo calculates the remainder when you divide two numbers. It uses the symbol $\%$.

So for example $9 \% 2$ gives 1 because when you divide 9 by 2, the remainder is 1.

Modulo is usually used with integers. Here are some useful things to know about $n \% m$, where n and m are integers:

- The result will always be an integer in the range 0 to $m - 1$.
- The result will be 0 if and only if n is a multiple of m .
- When m is 2, the modulo will be 0 if n is even, and 1 if n is odd.

As usual, if both values are ints the result will be an int, but if one or both are floats the result will be a float.

You can use modulo with floats. The general definition of modulo, that works with all n and m (even if they aren't whole numbers) is:

$$n - (n // m) * m$$

6.2.9 Unary operators

The $-$ symbol is also used for *unary negative*. For example:

```
a = 5
b = -2
print(-a)    # Prints -5
print(-b)    # Prints 2
```

Unary negative has the same effect as multiplying the number by -1. So these two lines have the same effect:

```
print(-a)
print(a * -1)
```

A unary operator only has one argument. It negates the value that follows it, so for example if a has the value 5, then -a has the value -5.

Python also provides a unary positive operator, +. The unary positive operator has the same effect as multiplying the number by 1, this is it leaves the original number unchanged. It isn't used often because it doesn't really do anything:

```
a = 5
b = -2
print(+a)    # Prints 5
print(+b)    # Prints -2
```

6.2.10 Augmented assignment

Python has a set of special operators called *augmented assignment operators*. Well, the assignment operator is just a fancy name for =:

```
x = 3
```

The = *assigns* the value 3 to x, so we call it the assignment operator. Here is an augmented assignment operator:

```
x += 3
```

This operator doesn't assign 3 to x. What it does instead is take the current value of x, add 3 to that, then assign the new value back to x. So in this case:

```
n = 2
n += 5
```

The final value of `n` 7 (it is initially assigned a value 2, then 5 is added to its value).

There are other similar operators:

- `n -= 5` subtracts 5 from `n` and stores the result in `n`.
- `n *= 5` multiplies `n` by 5 and stores the result in `n`.
- `n /= 5` divides `n` by 5 and stores the result in `n`.

Similarly there are `//=` and `%=`

A statement such as `a *= 4` is similar to `a = a*4`, but it isn't exactly the same. That is because the right hand side is calculated in full before the assignment happens. Try this case:

```
b *= 5 - 2
```

This is **not** equivalent to:

```
b = b * 5 - 2
```

The `5 - 2` is calculated before the multiplication, so the equivalent statement would be:

```
b = b * (5 - 2)
```

6.3 Precedence and brackets

If an expression has more than one arithmetic operator, it is important to know the order they will be applied. For example:

```
5 + 3 * 2
```

In this case, Python evaluates multiplication first, the addition, so this equation is evaluated as `3 * 2` gives 6, then add 5 gives the result 11. (If Python evaluated addition first, the answer would be `5 + 3` gives 8, multiplied by 2 gives the result 16.)

The order of evaluation of the different operators is called the *precedence*. Operators have a defined order of precedence, as follows:

- `+`, `-` - Addition and subtraction.
- `*`, `/`, `//`, `%` - Multiplication, division, floor division, modulo.
- `+x`, `-x` - Positive, negative.
- `**` - Power.

Operators that are further down the list have higher precedence, and so are evaluated first (for example `*` is evaluated before `+`, as in the previous example). Where operators have equal precedence, they are evaluated left to right. Here are some more examples:

$$7 - 3 + 2$$

Subtraction and addition have the same precedence, so they are evaluated left to right. So $7 - 3$ is evaluated first, giving 4, then 2 is added to give a result of 6.

$$3 / 2 + 1$$

Division has higher precedence than addition, so $3 / 2$ is evaluated first (giving 1.5), then 1 is added, so the final result is 2.5.

We can use brackets to change the precedence. For example:

$$7 - (3 + 2)$$

This time the $3 + 2$ is evaluated first, so the expression becomes $7 - 5$ which gives 2.

Similarly:

$$3 / (2 + 1)$$

With the brackets, the $2 + 1$ is evaluated first, so the expression becomes $3 / 3$ which gives 1.0 (as a float, of course, because it is division).

6.3.1 Statements and expressions

You may have noticed the terms *statement* and *expression*. They are similar but slightly different things.

A statement is a piece of code that *does something*. For example this code calculates a value and assigns it to a variable:

```
c = a + b
```

This code is another statement, it calculates a value and prints it:

```
print(b + 9.0)
```

However, this code is an expression. It calculates a value but doesn't do anything with the result:

```
a + b
```

6.4 Built-in maths functions

Python includes several *built-in* functions for dealing with numbers. Built-in functions are functions that you can just call without the need to import any modules.

6.4.1 abs()

The `abs` function finds the absolute value of a number. The absolute value is the magnitude of the number, ignoring the sign. So for example, `abs(3)` is 3, and `abs(-3)` is also 3.

Here is an example:

```
n = -10
print(abs(n))    # prints 10
```

`abs` works with ints or floats. The return value is of the same type as the input value.

6.4.2 min() and max()

The `min` function accepts two numbers, and return the smallest. Here is an example:

```
a = 3
b = 2
print(min(a, b))    # prints 2
```

`max` works in the same way, but it returns the largest of the two values.

`min` and `max` also work with ints or floats.

6.4.3 int()

The `int` function converts a value to an integer. It is mainly used with number or string values.

If we call the `int` function with a `float` value, the value will be converted to an `int`. The number will be truncated, rather than rounded - basically, anything after the decimal point will be ignored:

```
int(6.7)          # Truncated to 6
int(-6.7)         # Truncated to -6
```

We can also call the `int` function with an `int` value, as you might expect it will just return exactly what was passed in.

If you pass a string value into the `int` function it will attempt to convert the value:


```
int('3')      # Return integer 3
int('-3')     # Return integer -3
int('+3')     # Return integer 3
int(' 3 ')   # Return integer 3 (int will automatically remove spaces)
```

You will get an program error (of a type called `ValueError`) if the string does not exactly convert to an integer:

```
int('3.0')    # Error, this is a float not an integer
int('--3')    # Error, -- isn't a valid sign
int('3x')     # Error, x isn't valid part of an integer
int('2 + 2')  # Error, int can't evaluate expressions, it can only accept a number
```

6.4.4 float()

The `float` function converts a value to a floating-point number. It is mainly used with number or string values.

If we call the `float` function with an `int` value, the value will be converted to a `float`. The number will be truncated, rather than rounded - basically, anything after the decimal point will be ignored:

```
float(5)       # Gives 5.0
float(-5)      # Gives -5.0
```

We can also call the `float` function with a `float` value, similar to the `int` case it will just return exactly what was passed in.

If you pass a string value into the `float` function it will attempt to convert the value:

```
float('2.1')   # Return float 2.1
float('-2.1')   # Return float -2.1
float('+2.1')   # Return float 2.1
float(' 2.1 ') # Return float 2.1 (int will automatically remove spaces)
```

You will get an program error (of a type called `ValueError`) if the string does not exactly convert to either a float or int value:

```
float('--2.1') # Error, -- isn't a valid sign
float('2x.1')  # Error, x isn't valid part of an integer
float('2.1 + 2.1') # Error, int can't evaluate expressions, it can only accept a number
```

6.5 math module

The `math` module provides standard mathematical functions, including logarithms and trig functions (`sin`, `cos` and `tan`). To use these functions you will need to import `math`.

Here are a couple of examples:

```
import math

print(math.pi)          # pi = 3.141592653589793
print(math.sqrt(4))      # square root of 4 = 2
print(math.sqrt(2))      # square root of 2 = 1.4142135623730951
```

6.6 Limitations of floats

A float gives a very precise representation of a real value - as we saw earlier, it is stored with about 16 digits of precision. But it is not absolutely perfect. For example, look at the following code:

```
a = .8 + .1 + .1
b = .7 + .1 + .1 + .1
```

Now you would expect both `a` and `b` to be equal to `1.0`. But if you try this you will find that `a` is `1.0`, while `b` is `0.9999999999999999`. This is a tiny error, and it won't make any detectable difference to most calculations.

But if you were to *compare* `a` and `b`, they are different. They should be exactly the same, but due to the tiny error, Python will tell you they are not the same. If your program relies on comparing the two values and expects them to be the same, it might cause the program to fail.

You should always try to avoid comparing two floats in this way because it won't always work. Always try to use ints if you need to compare exact values.

6.7 More advanced topics

6.7.1 Scientific notation for floats

For very large or very small floating-point numbers, Python uses *scientific notation* to display the number. This same notation is used by scientific calculators, or even spreadsheets, to represent very large or very small numbers.

For example the number:

```
12000000000000000.0
```

would be displayed as:

```
1.2e+16
```

This means that the number is equal to 1.2, multiplied by 10 to the power 16. It saves us writing so many zeros, especially if we have an even bigger number like the googol we met earlier. Instead of a hundred zeros, we could write this in floating point as:

```
1e+100
```

We can do the same thing for small numbers. In fact, Python used scientific notation for anything smaller than 0.0001. For example:

```
0.000012
```

Would appear as:

```
1.2e-05
```

This means that the number is equal to 1.2 divided by 10 to the power 5.

If you haven't used scientific notation before, don't worry about it too much, but you might sometimes see Python spit out numbers that look like these, especially if you have a mistake in one of your calculations. So it is worth at least recognising the notation, even if you don't intend to use it.

6.7.2 Complex numbers

Python has built-in support for complex numbers. If you aren't familiar with complex numbers, it is a whole branch of mathematics that we aren't going to attempt to cover in a few paragraphs of a book on coding. But if you are already conversant with complex numbers, this is a short description of how they are handled in Python.

In mathematics, a complex number consists of two value, the real value and the imaginary value. In Python, these are represented by two floats.

There are two ways to create a complex number in Python:

```
c = 1.0 + 3.5j  
d = complex(1.0, 3.5)
```

These both create the same value, a complex number with a real part of `1.0` and an imaginary part of `3.5`. Notice that Python uses `j` to denote imaginary numbers. In mathematics, we normally use `i`, but electrical engineering uses `j`. Like it or not, Python uses `j` and that isn't likely to change.

Note also that it is `3.5j` not `3.5*j`.

When we print a complex number, Python also uses `j`:

```
print(c)    # (1 + 3.5j)
```

The normal mathematical operators `+`, `-`, `*`, `/` and `**` work with complex numbers (or any combination of complex and real numbers). The floor divide and mod operators (`//` and `%`) don't work because you can't find the floor of a complex number.

The `math` module doesn't generally work with complex numbers, but there is a `cmath` module that provides many of the same functions for complex numbers.

Complex numbers are quite a specialised area, this is just a brief overview, there is plenty more information on the python.org website.

7. Strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.1 Creating strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.2 String type

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.3 Operations on strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.3.1 Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.4 Converting between strings and other data types

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.5 How to use special characters in a string

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.5.1 Quote characters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.5.2 Unicode characters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.5.3 Newline characters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

7.5.4 The escape character \

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8. Loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.1 For loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.1.1 The loop variable

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.2 The range function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.2.1 Changing the start value

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.2.2 Using a step value

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.3 For loop example - printing a times table

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.4 While loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.5 While loop example - getting user input

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

8.6 Nested loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

9. Input and output

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

9.1 Input prompts

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

9.2 Inputting numbers

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

9.3 Print

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

9.4 Print separators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10. If statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.1 if statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.1.1 Example if statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.2 Comparison operators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.3 Else statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.4 Elif statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.5 Compound conditions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.6 Comparison operator chaining

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

10.7 Precedence

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11. Lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.1 What is a list?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.1.1 Lists vs arrays

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.2 Creating lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.2.1 Empty lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.2.2 The list function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.3 The list type

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.4 Accessing list elements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.4.1 Reading an element

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.4.2 Changing an element

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.5 Operations on lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.6 Looping over a list

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.6.1 Doing calculations in a loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.7 List functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.7.1 Finding the length of the list

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.7.2 Finding the `min` a `max` values

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.7.3 Adding up the values in a list

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.8 List methods

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.8.1 Adding elements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.8.2 Searching for elements in a list

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.8.3 Removing elements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.8.4 Other methods

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.9 Two-dimensional lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.9.1 Ragged arrays

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.10 Aliasing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.10.1 Reassigning variables

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.10.2 Does aliasing affect numbers?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.11 Examples

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.11.1 Finding every element of a given value

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

11.11.2 Creating a 2-dimensional list of zeros

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12. Using functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.1 Why use functions?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.2 Built-in functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.2.1 repr

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.2.2 chr and ord

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.2.3 help and dir

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.3 Importing modules

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.3.1 Simple import statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.3.2 Importing individual functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.3.3 Aliasing a module

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.3.4 Importing all functions (don't do this)

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.4 How to call functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.4.1 Optional parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.4.2 Variable number of arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.4.3 Named optional parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.5 Defining your own functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.5.1 Printing @s

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.5.2 Printing more @s

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

12.5.3 Returning a value

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13. More loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.1 The break operator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.1.1 Break in a for loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.1.2 Break in a while loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.2 The continue operator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.3 Using else with a loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.4 Nested loop statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.4.1 Printing times tables

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.5 Modifying a for loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.6 Looping in reverse order

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.6.1 Using range() to count backwards - ugly

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.6.2 reversed()

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.7 sorted()

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.8 Looping over multiple items

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.8.1 zip

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.8.2 How zip() works

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.9 More about zip()

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.10 Accessing the loop count using enumerate

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.11 Looping over selected items

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.11.1 Using filter

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

13.11.2 The advantage of using filter

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14. Programming logic

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.1 Boolean types

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.2 Comparison operators and their opposites

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.3 Boolean operations and De Morgan's Laws

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.4 Ternary operators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.5 Comparing containers

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.5.1 Strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.5.2 Lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.5.3 Other containers

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.6 Membership testing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.7 Identity testing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8 Truthy values

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8.1 Numbers

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8.2 Strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8.3 Lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8.4 Other collections

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8.5 None

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.8.6 Other objects

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.9 Short circuit evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.9.1 Short-circuiting with the or operator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.9.2 Short-circuiting with the and operator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.9.3 Short-circuiting to avoid errors

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.9.4 The value of an or/and expression

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

14.9.5 Getting a true or false value

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15. Slices

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15.1 Slicing a list

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15.2 Using negative indices

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15.3 Delete or replace a slice of a list

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15.4 Slicing tuples and strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15.5 Loop over a slice

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

15.6 Using steps

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16. More strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.1 Raw strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.2 Multi-line strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.3 Looping over a string

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.4 Joining strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.5 Splitting a string

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.6 Formatting strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.7 The in operator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8 Other string methods

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8.1 Changing case

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8.2 Dealing with whitespace

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8.3 Padding

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8.4 Search

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8.5 Splitting strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

16.8.6 Categorising strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17. Tuples

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.1 What is a tuple?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.2 Creating a tuple

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.3 Accessing elements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.4 Packing and unpacking tuples

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.4.1 Packing a tuple

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.4.2 Unpacking a tuple

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.4.3 Real-life examples

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5 Tuple vs list operations

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5.1 Tuple slices

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5.2 Adding elements to a tuple

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5.3 Finding elements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5.4 Removing elements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5.5 Looping

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.5.6 In case of emergency

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

17.6 Pros and cons of immutability

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18. Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.1 Program errors

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.2 What are exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.3 Exception types

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.3.1 ImportError

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.3.2 IndexError

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.3.3 TypeError

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.3.4 ValueError

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.3.5 ZeroDivisionError

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.4 Catching exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.4.1 Only catch exceptions you can handle

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.4.2 Accessing the exception message

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.5 Using else with exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.6 Using finally with exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

18.7 Throwing exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19. Working with files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.1 Using files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.1.1 Opening the file

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.1.2 Reading and writing data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.1.3 Closing the file

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.2 Reading data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.2.1 Code for reading a file

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.2.2 Reading lines from a file

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.2.3 Looping over the lines in a file

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.3 Writing data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.3.1 Code to write a file

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.3.2 The write function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.4 Using with statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.5 CSV data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.5.1 Reading CSV data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.5.2 Trying the code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.5.3 Formatting the output

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.5.4 Writing CSV data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

19.5.5 Understanding the code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

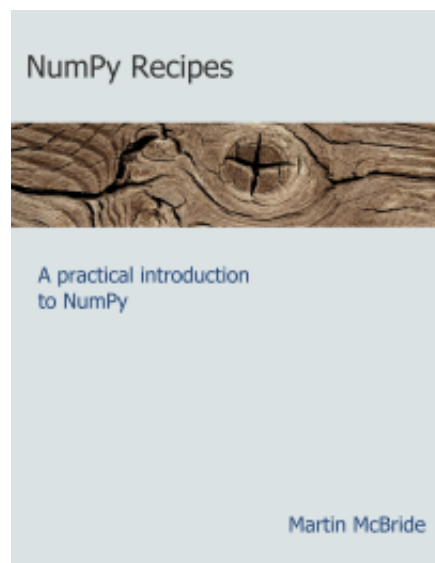
19.5.6 Trying the code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/pythonquickstart>.

20. More books from this author

I have several other Python books available. See <https://pythoninformer.com/books/> for more details.

20.1 Numpy Recipes



NumPy Recipes takes practical approach to the basics of NumPy

This book is primarily aimed at developers who have at least a small amount of Python experience, who wish to use the NumPy library for data analysis, machine learning, image or sound processing, or any other mathematical or scientific application. It only requires a basic understanding of Python programming.

Detailed examples show how to create arrays to optimise storage different types of information, and how to use universal functions, vectorisation, broadcasting and slicing to process data efficiently. Also contains an introduction to file i/o and data visualisation with Matplotlib.

20.2 Computer Graphics in Python with Pycairo



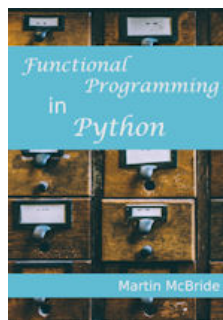
The Pycairo library is a Python graphics library. This book covers the library in detail, with lots of practical code examples.

PyCairo is an efficient, fully-featured, high-quality graphics library, with similar drawing capabilities to other vector libraries and languages such as SVG, PDF, HTML canvas and Java graphics.

Typical use cases include: standalone Python scripts to create an image, chart, or diagram; server-side image creation for the web (for example a graph of share prices that updates hourly); desktop applications, particularly those that involve interactive images or diagrams.

The power of Pycairo, with the expressiveness of Python, is also a great combination for making procedural images such as mathematical illustrations and generative art. It is also quite simple to generate image sequences that can be converted to video or animated gifs.

20.3 Functional Programming in Python



Python's best-kept secret is its built-in support for functional programming. Even better, it allows functional programming to be blended seamlessly with procedural and object-oriented coding styles. This book explains what functional programming is, how Python supports it, and how you can use it to write clean, efficient and reliable code.

The book covers the basics of functional programming including function objects, immutability, recursion, iterables, comprehensions and generators. It also covers more advanced topics such as closures, memoization, partial functions, currying, functors and monads. No prior knowledge of functional programming is required, just a working knowledge of Python.