

Chris Roffey

Cambridge IGCSE® and O Level

# Computer Science

## Programming Book

for Python

**Completely Cambridge**  
Cambridge resources  
for  
Cambridge qualifications





Chris Roffey

Cambridge IGCSE<sup>®</sup> and O Level

# Computer Science

Programming Book

For Python



CAMBRIDGE  
UNIVERSITY PRESS

**CAMBRIDGE**  
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi – 110002, India

79 Anson Road, #06 -04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

Information on this title: [www.cambridge.org](http://www.cambridge.org)

© Cambridge University Press 2017

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2017

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Printed in Spain by GraphyCems

*A catalogue record for this publication is available from the British Library*

ISBN 978-1-316-61782-3 Paperback

Additional resources for this publication at [www.cambridge.org](http://www.cambridge.org)

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party internet websites referred to in this publication, and does not guarantee that any content on such websites is, or will remain, accurate or appropriate. Information regarding prices, travel timetables, and other factual information given in this work is correct at the time of first printing but Cambridge University Press does not guarantee the accuracy of such information thereafter.

IGCSE is the registered trademark of Cambridge International Examinations

All examination-style questions, sample mark schemes, solutions and/or comments that appear in this book were written by the author. In examination, the way marks would be awarded to answers like these may be different.

.....  
NOTICE TO TEACHERS IN THE UK

It is illegal to reproduce any part of this work in material form (including photocopying and electronic storage) except under the following circumstances:

- (i) where you are abiding by a licence granted to your school or institution by the Copyright Licensing Agency;
- (ii) where no such licence exists, or where you wish to exceed the terms of a licence, and you have gained the written permission of Cambridge University Press;
- (iii) where you are allowed to reproduce without permission under the provisions of Chapter 3 of the Copyright, Designs and Patents Act 1988, which covers, for example, the reproduction of short passages within certain types of educational anthology and reproduction for the purposes of setting examination questions.

# Contents

Introduction	iv
How to use this book: a guided tour	vi
Acknowledgements	viii
1 Python 3	1
2 Sequence	8
3 Variables and Arithmetic Operators	14
4 Subroutines	23
5 GUI Applications (Optional)	30
6 Selection	37
7 Iteration	53
8 Designing Algorithms	72
9 Checking Inputs	81
10 Testing	92
11 Arrays	105
12 Pre-release Task Preparation	119
13 Examination Practice	128
14 Solutions	133
Appendix – Tkinter Reference	193



# Introduction

When Richard Morgan wrote the Visual Basic edition of this book he had two aims in mind. The first was to provide a programming book that specifically covered the material relevant to the Cambridge IGCSE® and O Level Computer Science syllabuses (0478/2210). The second, and perhaps more important, aim was to provide the student with a start to the exciting and rewarding process of being able to create their own computer programs. These are admirable aims that I hope have not been lost in this derivative translation into Python 3.

There are a few subtle changes to the flow diagrams and the pseudocode in this Python edition but fundamentally, wherever possible, the algorithms used are the same as those in the Visual Basic book. This has the exciting outcome that students can be taught the same material in a Cambridge IGCSE and O Level Computer Science class with a mixture of the two books. They can work on solutions in groups and then go and write the code for working implementations of their algorithms in either language.

Python and Visual Basic have different strengths and weaknesses and so they lend themselves to slightly different approaches. For this reason, the chapters have been slightly reordered in this book. The original Chapter 1 has been split: only text-based programming is introduced in Chapter 1 while how to produce GUIs has been moved to the optional Chapter 5. There is also an additional chapter on preparing for the pre-release task. All other chapter titles remain the same so easy comparison should be possible.

## Language

The syntax and structures used to implement programming techniques will vary across different languages. This book is entirely based around Python 3, one of the three recommended languages for the Cambridge International AS and A Level syllabus. Python has, at its core, the principle that code should be easy to read. This means that in many ways it is very close to pseudocode.

The pseudocode structure used in the Cambridge IGCSE and O Level Computer Science examination papers uses a language neutral style. Although students are expected to be familiar with this and be able to read and follow the logic easily, they are not expected to produce their own pseudocode in exactly this style. Pseudocode is meant to be a way of expressing clearly the logic of a program, free from the worries of syntax.

Python also has a recommended style guide that can be found at <https://www.python.org/dev/peps/pep-0008/>.

Here, for example, it is recommended that Python programmers name functions and variables with descriptive all lower case characters separated by underscores, for example `my_variable`. This style is never used in Cambridge IGCSE and O Level Computer Science pseudocode; however, students should not be marked down for doing so in their own pseudocode. As it could be very confusing to keep swapping naming conventions, this book assumes that students are going to stick, wherever possible, to the correct Python style but be flexible enough thinkers to be able to read other pseudocode styles. It is recommended that when preparing for exams, students ensure they are aware of the exam board variable naming style. Chapter 13 in this book provides some examination style questions.

## Examination focused

The Cambridge IGCSE and O Level Computer Science course will test computational thinking independent of any specific programming language. It will do this through the use of program design tools such as structure diagrams and flowcharts. It will also make use of pseudocode, a structured method for describing the logic of computer programs.

It is crucial that the student becomes familiar with these techniques. Throughout this book all the programming techniques are demonstrated in the non-language-specific format required, with the exception of variable and function naming. This will help to prepare the student to answer the types of question they will meet in their studies.

To support learning, many of the chapters include examination-style tasks. Chapter 14 has examples of appropriate code solutions that show how to turn logical ideas into actual programs. There is also a series of examination-style questions in Chapter 13, which has a sample mark scheme giving possible solutions and showing where the marks might be awarded.

## Developing programming skills

One of the advantages of Python is that it provides a language that encourages the student to program solutions making use of the basic programming constructs: sequence, selection and iteration. Although the language does have access to many powerful pre-written code libraries, they are not generally used in this book.

Computational thinking is the ability to break down a problem into its constituent parts and to provide a logical and efficient coded solution. Experience shows that knowing how to think computationally relies much more on an understanding of the underlying programming concepts than on the ability to learn a few shortcut library routines.

This book is aimed at teaching those underlying skills which can be applied to the languages of the future. It is without doubt that programming languages will develop over the coming years but the ability to think computationally will remain a constant.

# How to use this book: a guided tour

72

## Chapter 8: Designing Algorithms

### Learning objectives

By the end of this chapter you will understand:

- that systems are made up of subsystems, which may in turn be made up of further subsystems
- how to apply top-down design and structure diagrams to simplify a complex system
- how to combine the constructs of sequence, selection and iteration to design complex systems
- how to produce effective and efficient solutions to complex tasks.

**Learning objectives** – are included at the beginning of each chapter and present the learning aims for the unit.

This is very similar to the Cambridge IGCSE and O Level Computer Science pseudocode format:

```

WHILE counter > 0 DO
  //code to be iterated
  counter = counter + 1
ENDWHILE
    
```

**Key terms** – provide clear definitions for the most important terms within each unit.

### KEY TERMS

**WHILE loop:** A type of iteration that will repeat a sequence of code while a set of criteria continue to be met. If the criteria are not met at the outset the loop will not run and the code within it will not be executed.

### SYLLABUS CHECK

**Pseudocode:** understand and use pseudocode, using WHILE ... DO ... ENDMETHOD loop structures.

**Syllabus checks** – link programming concepts to points on the Cambridge IGCSE syllabus.

Each individual element of the loop performs an important role in achieving the iteration, as shown in Table 7.04.

Table 7.04

Element	Description
while	The start of the loop
counter > 0	The condition that controls the loop. Each time the iteration is run, the condition is evaluated and if it remains True, the iteration will run. Once the condition is False, execution of the code is directed to the line following the loop. In counter-controlled WHILE loops, it is important that code is included within the loop to increment or decrement the counter. In a FOR loop, the counter is automatically incremented. The same facility does not apply to WHILE loops and, as a result, the programmer must include appropriate code.
end of indented code	The end of the current iteration. Execution of the program returns to while so the condition can be re-evaluated and further iterations actioned. Do not forget to add ENDMETHOD when writing pseudocode.



### TIP

Remember that WHILE loops iterate while the condition evaluates to True. It is possible to create an infinite loop rather easily:

```

>>> while True:
    print('Hello', end='')
    
```

It is therefore important to know how to break out of infinite loops. To do so, hold down the **CTRL** key on your keyboard and press **⏏**. Try the code above yourself in an interactive session. The optional parameter `end=''` provided in the `print()` function suppresses the default line return.

**Tip boxes** – offer quick suggestions to remind students about important learning points.



**Task 2 – Discussion Question**

- a What is the aim of this flowchart?
- b What kind of loop is being suggested here?

**Extension tasks** – build on task exercises to help the student further develop their knowledge and understanding.

**11.07 Array Tasks****Task 6**

- a Draw a flowchart and create a pseudocode algorithm that iterates through an array of Integers and outputs the average. Declare and initialise the array with the following set of Integers: 12, 14, 10, 6, 7, 11 and 3.
- b Test that your algorithm works by programming and running the code in Python.

**Task 7**

An algorithm will take an Integer value,  $n$ . It will call a subroutine to place into an array 12 incremental multiples of  $n$  (the first array index will hold  $1 \times n$  and the last index position  $12 \times n$ ). An additional subroutine will allow the user to output all the multiples in order.

- a Draw a flowchart and create pseudocode for this algorithm.
- b Test that your algorithm works by programming and running the code in Python.

**Task 8**

The data in Table 11.06 is to be organised in arrays so that the user can search via User ID and the system will display all the data related to that User ID.

Table 11.06

User ID	Age	Gender
112	45	Male
217	16	Female
126	27	Female

- a Draw a flowchart and create a pseudocode algorithm that accepts a User ID and displays the related data.
- b Test that your algorithm works by programming and running the code in Python.

**Tasks** – contain exercises for the student to test their knowledge of the topic.

**Summary**

- An array is a variable that can hold a set of data items, of the same data type, under a single identifier.
- When an array is declared, its size is defined. In Python indexes start from zero.
- Each element or data item in an array can be referenced by its index.
- The index can be used to read or write values in an array.
- A FOR loop can be used to iterate through the index locations in an array. The loop counter is used to identify successive index numbers.
- Holding records which consist of more than one data item can be achieved by the use of multiple arrays. Data for each record is held at the same index position in the different arrays.
- When using Python to implement algorithms involving arrays, a list is used as a substitute for an array.

**Summary checklists** – are included at the end of each chapter to review what the student has learned.

# Acknowledgements

*Thanks to the following for permission to reproduce images:*

Cover image: Soulart/Shutterstock; Chapter opener 1 isak55/Shutterstock; Chapter opener 2 aimy27feb/Shutterstock; Chapter opener 3 Image Source/Getty Images; Chapter opener 4 Devrimb/iStock/Getty Images; Chapter opener 5 Andrew Brookes/Getty Images; Chapter opener 6 Magictorch/Ikon Images/Getty Images; Chapter opener 7 alexaldo/iStock/Getty Images; Chapter opener 8 Ioana Davies (Drutu)/Shutterstock; Chapter openers 9, 10 Kutay Tanir/Photodisc/Getty Images; Chapter opener 11 ILeyden/Shutterstock; Chapter opener 12 Kamil Krawczyk/E+/Getty Images; Chapter opener 13 Aeriform/Getty Images



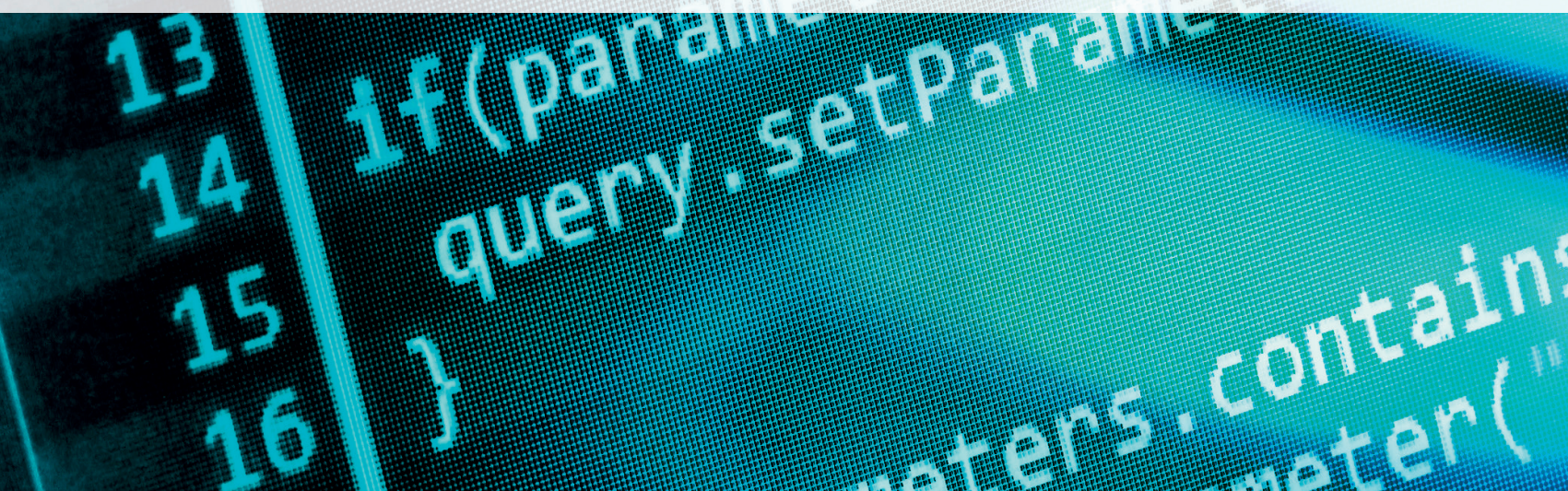


# Chapter 1: Python 3

## Learning objectives

*By the end of this chapter you will understand how to:*

- obtain a simple IDE to support your programming
- use both interactive mode and script mode in Python
- program and save a text-based application in script mode.





## 1.01 Getting Python 3 and IDLE For Your Computer

Python 3 is the latest version of the Python programming language. It is a loosely typed script language. Loosely typed means that it is usually not necessary to declare variable types; the interpreter looks after this. Script languages do not have a compiler. This means that, in general, Python programs cannot run as quickly as compiled languages; however, this brings numerous advantages, such as fast and agile development.

Python is a powerful, modern programming language used by many famous organisations such as YouTube and National Aeronautics and Space Administration (NASA) and it is one of the three programming languages that can be used to develop Google Apps.

There are installers for Windows and Apple computers available at <https://www.python.org/downloads/>. You should choose the latest stable version of Python 3 (Python 3.5.0 at time of writing). If you have a Raspberry Pi, then two versions of Python are already installed.

On the Raspberry Pi you can start programming in **interactive mode** straight away by selecting *Python 3* from *Programming* in the main *Menu* in the task bar (Figure 1.01).



Figure 1.01 Starting Python 3 on a Raspberry Pi

This opens IDLE which is the **IDE** (Integrated Development Environment) that comes packaged with Python (Figure 1.02).

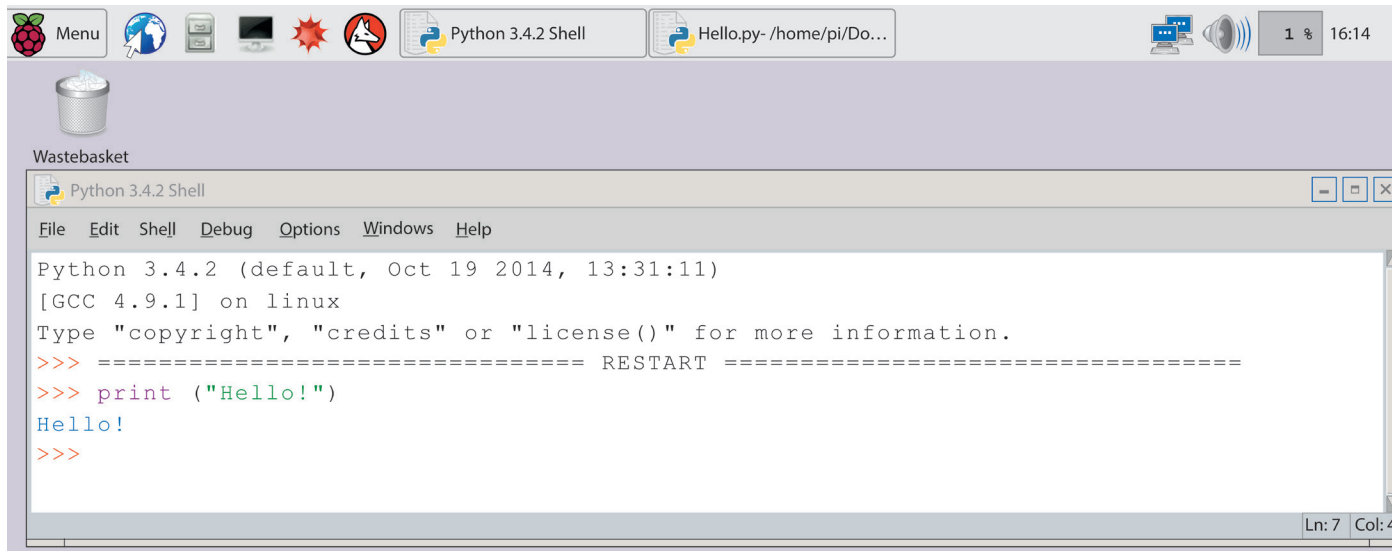


Figure 1.02 IDLE's Python Shell – working in interactive mode on a Raspberry Pi



#### KEY TERMS

**Interactive mode:** When writing and running code in the Python Shell window, interactive mode allows us to try out snippets of code without saving.

**IDE:** An Interactive Development Environment is a special text editor with useful built-in tools for programmers.

After installing Python 3 on Apple computers, IDLE can be found in the main *Python 3* folder in your *Applications* folder.

On Windows computers, once installed, IDLE can be opened by looking for the *Python 3.5* folder found in *All Programs* when opening the *Start* menu. From the *Python 3.5* folder choose *IDLE*.

In all cases this opens a window containing the Python Shell. This can run simple programs at the `>>>` prompt. Executing small programs in the Shell window is referred to as working in interactive mode. It provides a very useful environment for running short code experiments when developing larger programs in **script mode**. Throughout this book you will be prompted to try out code snippets and run short experiments so that you get used to new functions and syntax in interactive sessions. These sessions can be accessed extremely quickly by opening IDLE and typing directly into the Shell window.



#### KEY TERMS

**Script mode:** When writing code in a new window in IDLE that will be saved in a file as a Python script.

To create a script that can contain more complex programs and, more significantly, can be saved and reused, you should obtain a new window by selecting *New File* from the *File* menu. This opens a blank script window into which you can type and save your code (always with

the extension `.py`). IDLE provides help with code colouring and auto-indenting in whichever mode you are working.

In script mode, the Shell window takes on a new role as a console. Text output from your programs appears in this window (see Figure 1.03). It is also where users provide input, and error messages appear. The console is still available as a Shell window to use for quick experiments while developing your scripts.

To run your scripts, you should save your file to a sensibly named folder in your *Documents* folder and then select *Run Module* from the *Run* menu, or press F5 on your keyboard.

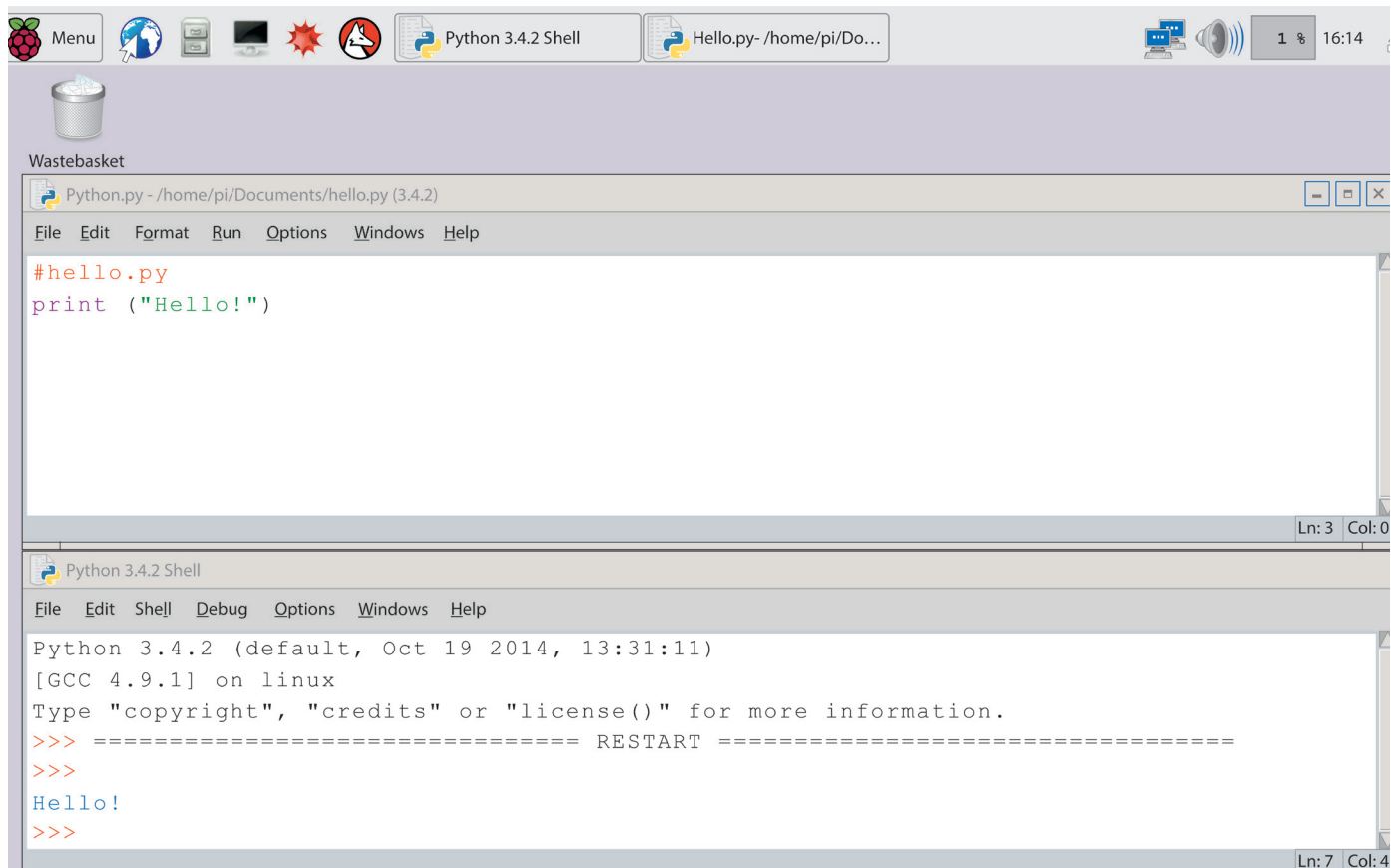


Figure 1.03 IDLE's Python Shell and a script window open on a Raspberry Pi

## 1.02 Other Integrated Development Environments

IDLE is perfectly adequate for performing all the tasks required in this book. However, if you have been programming with IDLE for a number of years, you might like to try one of the other many IDEs available.

The one that is used for the remainder of the screenshots in this chapter, and occasionally later in the book, is Wing IDE 101 (Figure 1.04). This is a free version of a commercial IDE that provides a carefully selected set of facilities that are useful for students. It can be downloaded from <http://wingware.com/downloads/wingide-101> where brief introductory videos and installation instructions are available. Please be aware that the Raspberry Pi is currently not powerful enough to run this or most other commercial IDEs satisfactorily. Wing IDE 101 is available for Windows, Apple computers, Ubuntu and other versions of Linux.



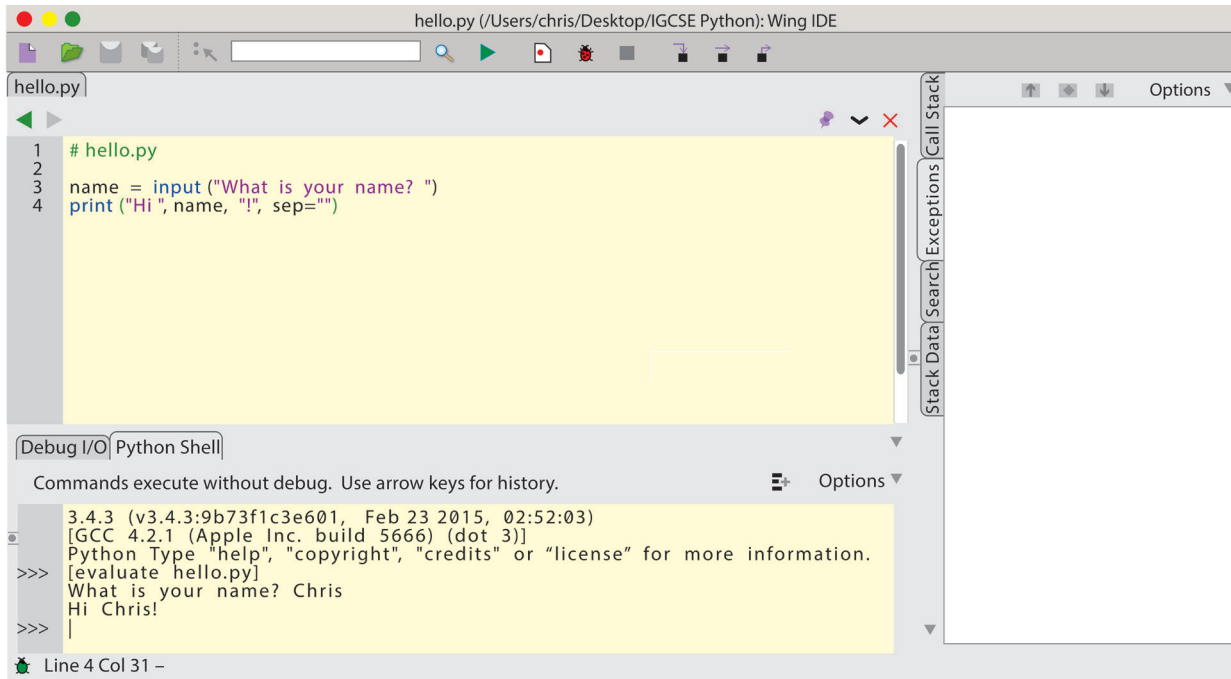


Figure 1.04 Wing IDE 101 Integrated Development Environment.

The large panel in the middle of the application is where you write your scripts. Interactive sessions can be run in the Shell tab below this window.

There are two ways to run a program in Wing IDE. Clicking the run button (▶) will access the Python Shell as shown in Figure 1.04. An alternative – and recommended – way of running your scripts is to click on the bug (🐞) to the right of the run button (Figure 1.05). This opens the Debug I/O panel and now provides error messages in the Exceptions tab on the right.

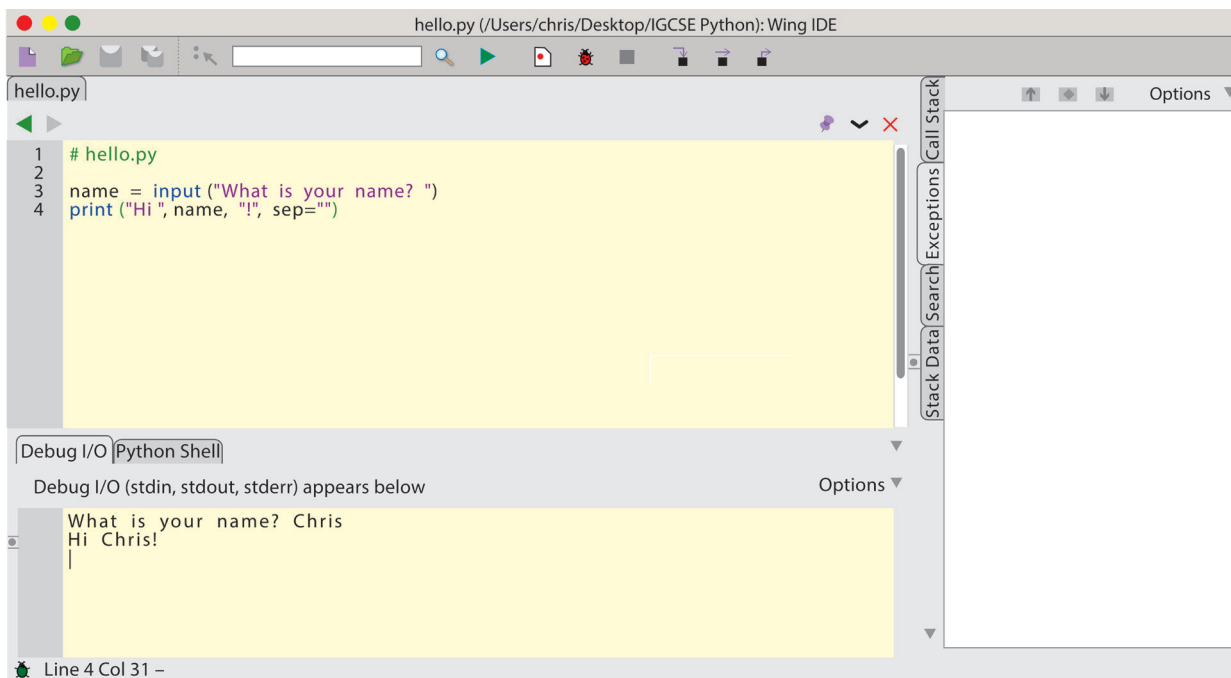


Figure 1.05 Wing IDE 101 showing input and output after pressing the bug button

## 1.03 Make Your First Program Using Interactive Mode

In IDLE's interactive mode window or in the Python Shell tab in Wing IDE, type out the following line of code at the `>>>` prompt and then press return.

### INTERACTIVE SESSION

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

You have now run your first interactive mode program. Your code told the computer to print the text 'Hello world!' to the screen. It executed your code when you pressed the return key on your keyboard. You can also use interactive mode as a simple calculator. Try entering this sum and press return:

### INTERACTIVE SESSION

```
>>> 3*4
```



### TIP

Interactive sessions are used to illustrate simple concepts or to show the correct use of some new syntax. It is a good idea to start your own interactive session and try the code yourself. You may well want to experiment further to deepen your understanding.

## 1.04 Make Your First Program Using Script Mode

Working in IDLE select *New File* from the *File* menu to open a new window into which you can type your code and then save it as a script. In Wing IDE, simply type into the main script panel. Whichever IDE you are using, copy the following code and then save your file as `hello.py` to a new folder called *Python Code* in your *Documents* folder.

```
# hello.py
print('Hello world!')
```

If using IDLE, run the script by selecting *Run Module* from the *Run* menu or by pressing F5. In Wing IDE click the bug button.

Any code preceded by a hash symbol (#) is called a comment. This is ignored by the computer when executing the script and is purely for the programmer. It can be useful to include the file name in its own comment at the top of a script.

## 1.05 Graphical user interface Applications

Although not required by the syllabus, your Python scripts are not limited to text-based applications. By importing the **tkinter** module, it is easy to produce visually rich **graphical user interfaces (GUIs)** and attach your algorithms to buttons in windows.

Producing GUI based applications is outside the syllabus.



## KEY TERMS

**tkinter:** An example of a GUI toolkit which is provided as part of the standard library when you install Python.

**graphical user interface (GUI):** Graphical user interfaces contain items like buttons, text entry boxes and radio buttons.

Chapter 5, GUI applications, is an optional chapter included in this book. In it, you will learn how to build your own GUIs and how to repurpose your algorithm solutions to work with them. From Chapter 5 onwards, there will be some tasks provided that include making GUIs. Although these are not required by the Cambridge IGCSE and O Level Computer Science syllabus, repurposing your solutions to work with GUIs will make you a more flexible programmer and allow you to produce more professional looking applications.

## 1.06 Additional Support

The intention of this book is to introduce programming concepts that make use of the non-language-specific formats included in the syllabus. Python 3 is used to provide the opportunity for you to use a real programming language to develop your understanding of these concepts. The official documentation for the Python programming language can be accessed at <https://docs.python.org/3/>.

A simple syntax reference guide that can be printed out and fits in your pocket is available from the Coding Club website at <http://codingclub.co.uk/codecards/>.

This textbook also has its own companion website at [\[companion website URL\]](#).

## Summary

- Python 3 is a loosely typed programming language that is designed to encourage easily read code.
- Python 3 comes with a simple Integrated Development Environment called IDLE.
- There are many other IDEs available, such as Wing IDE 101, which is specifically designed for students.
- There are three main styles of programming in Python 3:
  - interactive mode: quick tests and trials that can be programmed in the Python Shell
  - text-based: in script mode, text-based scripts can be saved so that your applications can be reused
  - GUI applications: full, visually rich applications that can be produced in script mode.

# Chapter 2:

## Sequence

### Learning objectives

*By the end of this chapter you will:*

- know the difference between the three programming constructs: sequence, selection and iteration
- understand the role of flowcharts and pseudocode when designing programs
- understand the main symbols used in flowcharts
- understand the preferred format of pseudocode when using sequence solutions.

## 2.01 Logical Design Considerations

When designing programs, it is crucial to consider the order in which the task needs to be completed. All tasks will follow some logical order. When working on a solution to a problem, you should first apply the **top-down design** technique to break down the big problem into smaller ones. In terms of computational thinking, this is referred to as **decomposition**.



### KEY TERMS

**Top-down design:** Design process where a complex task is broken down into smaller tasks.

**Decomposition:** The process of breaking down a large task into smaller tasks.

For example, to calculate the time it would take to complete a journey, you need to know the distance to be travelled and the intended speed. The first step would be to calculate the distance to be travelled. Without this **data** the rest of the task could not be completed.

The **sequence** in which instructions are programmed can be crucial. Consider the following algorithm:

Distance = Speed \* Time  
 Speed = 12 kilometres per hour  
 Time = 15 minutes



### KEY TERMS

**Data:** Raw facts and figures.

**Sequence:** Code is executed in the order it is written.

A human would recognise that the values for speed and time have been given after the calculation. A coded program would simply try to complete the task in the order given and crash. This is because at the time of the calculation no values have been provided for speed or time. In fact, the variables Speed and Time will not even be recognised by the program at this first step.

A human would probably also recognise that the speed is quoted 'per hour' while the time is given in minutes. They would be able to correctly calculate the distance as 3 kilometres ( $12 * 15/60$ ). Even if the values had been provided before the calculation, the program would calculate distance incorrectly as 180 kilometres by simply multiplying the given values ( $12 * 15$ ).

## 2.02 Programming Constructs

Python and other procedural languages make use of three basic programming constructs: sequence, **selection** and **iteration**. Combining these constructs provides a programmer with the tools required to solve logical problems. Selection and iteration offer a number of alternative approaches and are covered in detail in Chapters 6 and 7.



### KEY TERMS

**Selection:** Code branches and follows a different sequence based on decisions made by the program.

**Iteration:** Code repeats a certain sequence of code a number of times depending on certain conditions.



## Sequence

The order in which a process is completed is often crucial. Take the mathematical expression  $A + B \times C + D$ . The rules of precedence state that the multiply operation must be completed first. If a programmer wishes that the operations  $A + B$  and  $C + D$  be completed before multiplying, then it would be necessary to either complete the two additions separately first or write the expression in the form  $(A + B) \times (C + D)$ .

In programming, the sequence is indicated by the order in which the code is written, usually top to bottom. The program will execute the first line of code before moving to the second and subsequent lines.

Sequence is the subject of this chapter so this will be discussed in more detail later.

## Selection

Often your programs will perform different processes dependent on user input. Consider a system designed to provide access to the school network based on when a user inputs a username and password. The system would need to follow a different path if the user inputs an incorrect password or username. In this circumstance, the user might simply be prompted to re-input their details. See Chapter 6 for more details.

## Iteration

It is common for a program to perform identical processes on different data items. Consider a program that takes a series of coordinates and produces a line graph. The code that provides the instructions that plot each new coordinate will be repeated for each of the coordinates given. To repeat instructions, we put them in a loop, which is referred to as iteration. See Chapter 7 for more details.

## 2.03 Design Tools

When you design programs, it is normal to plan the logic of the program before you start to code the solution. This is an important step in the design of effective systems because a flaw in the logic will often result in programs that run but produce unexpected outputs.

The first step in the design process is to break down the problem into smaller problems. This is called top-down design. It makes it easier to plan and write code for these smaller problems. A **structure diagram** is used to help organise the top-down design. Chapter 8 provides more detail about top-down design and structure diagrams.

The next stage is to design an algorithm for the individual problems. Two approaches that can be used at this stage to help generate logically accurate systems are **flowcharts** and **pseudocode**.



### KEY TERMS

**Structure diagrams:** A diagrammatical method of expressing a system as a series of subsystems.

**Flowchart:** A graphical representation of the logic of a system.

**Pseudocode:** A language-independent system for defining the logic of a system without the need for strict syntax.



To succeed in your course, you will be expected to have a working understanding of flowcharts and pseudocode. You will need to be able to use them to explain the logic of your solutions to given tasks. Both methods are used throughout this book.








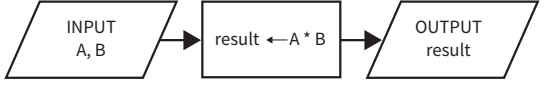
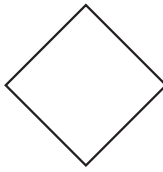
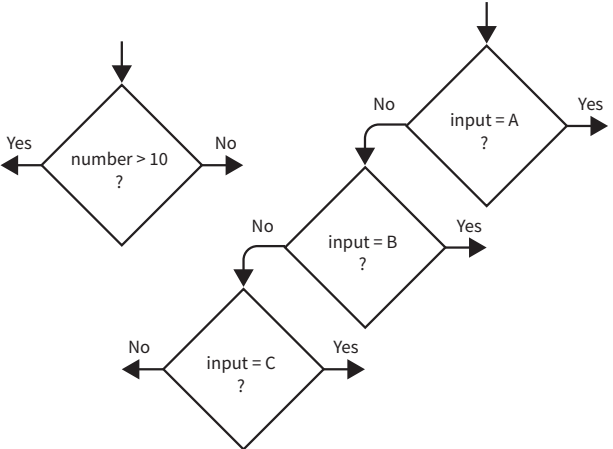
### SYLLABUS CHECK

**Problem solving and design:** use flowcharts and pseudocode.

## 2.04 Flowcharts

Flowcharts are graphical representations of the logic of the intended system. They make use of symbols to represent operations or processes and are joined by lines indicating the sequence of operations. Table 2.01 details the symbols used.

Table 2.01

Symbol	Notes	Example
Terminator 	The START or END of a system.	
Input or output 	Use when INPUT is required from the user or OUTPUT is being sent to the user.	
Process 	A process within the system. Beware of making the process too generic. For example, a process entitled 'Calculate Average' would be too generic. It needs to indicate the values used to calculate the average.	
Data flow line 	Joins two operations. The arrowhead indicates the direction of the flow. Iteration (looping) can be indicated by arrows returning to an earlier process in the flowchart.	
Decision 	A point in the sequence where alternative paths can be taken The condition is written within the symbol. Where multiple alternatives exist, this is indicated by chained decision symbols. Each 'No' condition directs to another decision in the process.	

## 2.05 Pseudocode

Pseudocode is a method of describing the logic and sequence of a system. It uses keywords and constructs similar to those used in programming languages but without the strict use of syntax required by formal languages. It allows the logic of the system to be defined in a language-independent format. This can then be coded using any programming language. Hence, the flow diagrams and pseudocode in this book are almost entirely the same as those used in the Visual Basic sister book of the series.

Pseudocode follows a number of underlying principles:

- Use capital letters for keywords close to those used in programming languages.
- Use lower case letters for natural language descriptions.
- Use indentation to show the start and end of blocks of code statements, primarily when using selection and iteration.

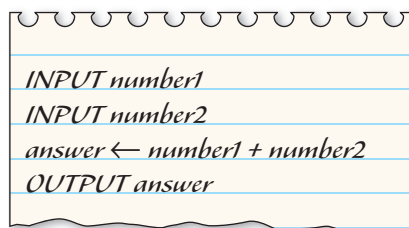
One of the advantages of learning to program using Python is that the actual coding language is structured in a similar way to natural language and therefore closely resembles pseudocode. Python IDEs such as IDLE or Wing IDE also automatically indent instructions where appropriate.

### SYLLABUS CHECK

**Pseudocode:** understand and use pseudocode for assignment, using  $\leftarrow$ .

## 2.06 Pseudocode Example

The following pseudocode is for an algorithm that accepts the input of two numbers. These values are added together and the result is stored in a memory area called **answer**. The value in **answer** is then displayed to the user. (In Chapter 3, we will learn that this memory area is known as a variable.)



```

INPUT number1
INPUT number2
answer ← number1 + number2
OUTPUT answer

```

Note the use of  $\leftarrow$  to show the passing of values. This is pseudocode's assignment operator. In pseudocode the equals symbol (=) is used to compare values. It is important to note that in Python the equals symbol is used for assignment and two equals symbols (==) are used to compare values.

### TASK

#### Task 1

Construct a flowchart to represent this pseudocode example.

## 2.07 Effective use of Flowcharts and Pseudocode

Due to their universal nature, flowcharts and pseudocode are used extensively in the Cambridge IGCSE and O Level Computer Science syllabus.

The aim of this book is to help you to learn how to code effective systems in Python. The following chapters make use of flowcharts and pseudocode to define the logic of systems before moving on to specific Python solutions.

**TIP**

After completing a flowchart or pseudocode, it is a good idea to try and follow it through a step at a time in the same way a computer would in order to identify if you have any missing steps.

Learning how to explain the logic of programs by using these design techniques is important not only in your preparation for examination but also for your preparation in using the languages of the future. Language syntax is likely to change but the need for effective computational thinking will remain.

### Summary

- Programmers make use of three constructs when writing code:
  - sequence: the logical order in which code is executed
  - selection: branching of code onto different paths based on certain conditions
  - iteration: repetition of sections of code.
- Before coding a program, it is crucial to design an appropriate algorithm.
- Flowcharts are graphical representations of the logic of a system. They make use of symbols to represent operations or processes, and lines indicate the sequence of operations.
- Pseudocode describes the logic of a system in a similar way to a programming language but without such strict syntax requirements.



# Chapter 3:

## Variables and Arithmetic Operators

### Learning objectives

*By the end of this chapter you will understand how to:*

- declare and use variables and constants
- use the data types Integer, Real, Char, String and Boolean
- use basic mathematical operators to process input values
- design and represent simple programs using flowcharts and pseudocode.



### 3.01 Variables and Constants

Programs are normally designed to accept and input data, and process that data to produce the required output. Data used in programs can vary depending on the aim of the program; a calculator will process numerical data while a program designed to check email addresses will process textual data. When writing programs, you will use variables or constants to refer to these data values. A **variable** identifies data that can be changed during the execution of a program while a **constant** is used for data values that remain fixed. In many computer languages, the **data type** must be provided when a variable or constant is declared. The data type is used by the computer to allocate a suitable location in memory. These languages, such as Java, are said to be strongly typed.



#### KEY TERMS

**Variable:** The identifier (name) given to a memory location used to store data; the value can be changed during program execution.

**Constant:** A named memory location that contains data that can be read but not changed by the program. (In Python, the data can be changed. However, by capitalising your variable name, you are indicating to readers of your code the intention that the value of the data should not be.)

**Data type:** The format of the data in the field.

Python is an example of a loosely typed programming language. In Python, all variables are in actual fact objects. The computer decides on a variable's data type from the context you provide. Compare these two variable declarations, first in Visual Basic and then in Python:

In Visual Basic:

```
Dim Score As Integer = 0
```

In Python:

```
score = 0
```

### 3.02 Types of Data

If Python decides which data types are required for the programmer, how can we know what data type has been allocated? This is achieved by using the built in `type()` function. Study this interactive session in the Python Shell to see how to use this function:

#### INTERACTIVE SESSION

```
>>> my_integer = 3
>>> type(my_integer)
<class 'int'>
>>> my_string = 'hello'
>>> type(my_string)
<class 'str'>
```

The basic data types you need to know are identified in Table 3.01.

Table 3.01

Data type	Description and use	Python type (variable) query returns:
Integer	Whole numbers, either positive or negative  Used with quantities such as the number of students at a school – you cannot have half a student.	'int'
Real	Positive or negative fractional values  Used with numerical values that require decimal parts, such as currency.  Real is the data type used by many programming languages and is also referred to in the Cambridge IGCSE and O Level Computer Science syllabus.	'float'  Python does not use the term Real. The equivalent data type in Python is called 'floating point'.
Char	A single character or symbol (for example A, z, \$, 6)  A Char variable that holds a digit, cannot be used in calculations.	'str'  Python treats characters as small Strings.  Note:  <pre>&gt;&gt;&gt; my_var = '3' &gt;&gt;&gt; type(my_var) &lt;class 'str'&gt; &gt;&gt;&gt; my_var = 3 &gt;&gt;&gt; type(my_var) &lt;class 'int'&gt;</pre>
String	More than one character (a String of characters)  Used to hold words, names or sentences.	'str'
Boolean	One of two values, either True or False  Used to indicate the result of a condition. For example, in a computer game, a Boolean variable might be used to store whether a player has chosen to have the sound effects on.	'bool'  e.g.  <pre>&gt;&gt;&gt; sfx = False &gt;&gt;&gt; type(sfx) &lt;class 'bool'&gt;</pre>

**SYLLABUS CHECK**

**Programming concepts:** understand and use Integer, Real, Char, String and Boolean.

### 3.03 Pseudo Numbers

Telephone numbers and ISBNs both consist of digits but are not truly numbers. They are only a collection of digits used to uniquely identify an item; sometimes they contain spaces or start with a zero. They are not intended to be used in calculations. These are known as pseudo numbers and it is normal to store them in a String variable. If you store a mobile phone number as an Integer, any leading zeroes will be removed and spaces and symbols are not permitted.

## 3.04 Naming Conventions in Python

There are a variety of naming conventions in Python. Here are a few of them:

### Variable Names

Use all lower case, starting with a letter and joining words with underscores. It is considered good practice to use descriptive names as this aids readability and reduces the need for so much commenting.

For example:

```
score_total = 56    ✓
Total = 56         ✗
t = 56             ✗
```

#### FURTHER INFORMATION

There are 31 reserved words that cannot be used as your own variable names:

```
and as assert break class continue def del elif else except
finally for from global if import in is lambda nonlocal not or
pass print raise return try while with yield.
```

### Constants

Use all upper case characters to indicate constants.

For example:

```
PI = 3.1415
```

It is considered good practice to give global variables an initial value when **declaring variables**: this is known as **initialising variables**. See the next section for more about global and local variables.



#### KEY TERMS

**Declaring variables:** When a variable is given a name and assigned no value. It is important to declare or initialise global variables.

**Initialising variables:** When a variable is given a start value.

## 3.05 Variable Scope

When declaring a variable, the placement of the declaration in the code will determine which elements of the program are able to make use of the variable.

**Global variables** are those that can be accessed from any routine within the program. To give a variable global status, it must be declared outside any specific subroutine. It is good practice to make all the global variable declarations at the start of your code directly below any import statements.

To access global variables in functions, they can be called as normal; however, if the function is going to change the value stored in the global variable it must be re-declared using the `global` keyword (see example on page 18).

Variable Scope is  
outside the syllabus



**Local variables** can only be accessed in the code element in which they are declared. They are used when the use of the variable will be limited to a single routine such as a function. Using local variables reduces the possibility of accidentally changing variable values in other parts of your program.



## KEY TERMS

**Global variables:** Variables that can be accessed from any routine within the program.

**Local variables:** Variables that can only be accessed in the code element in which they are declared.

In the following Python code example, there is a global variable (`player_score`) and one local variable (`result`). As the value of the global variable might be changed by the `update_player_score()` function, `player_score` needs to be re-declared at the start of the function with the `global` keyword:

```
player_score = 0

def update_player_score():
    global player_score
    result = 5
    if player_score < result:
        player_score = player_score+1
```

## 3.06 Arithmetic Operators

There are a number of operations that can be performed on numerical data. Combining these operations and appropriate variables allows you to create programs that are capable of performing numerical computational tasks.

The basic operators used in Python 3 are shown in Table 3.02

Table 3.02

Operation	Example of use	Description
Addition	<code>result = number1 + number2</code>	Adds the values held in the variables <code>number1</code> and <code>number2</code> and stores the result in the variable <code>result</code> .
Subtraction	<code>result = number1 - number2</code>	Subtracts the value held in <code>number2</code> from the value in <code>number1</code> and stores the result in the variable <code>result</code> .
Multiplication	<code>result = number1 * number2</code>	Multiplies the values held in the variables <code>number1</code> and <code>number2</code> and stores the result in the variable <code>result</code> .
Division	<code>result = number1 / number2</code>	Divides the value held in the variable <code>number1</code> by the value held in <code>number2</code> and stores the result in the variable <code>result</code> .
Integer division	<code>result = number1 // number2</code>	Finds the number of times <code>number2</code> can go into <code>number1</code> completely, discards the remainder, and stores the result in the variable <code>result</code> .



## TIP

If you find yourself having to write some Python 2 programs, it is important to be aware that the syntax for division and Integer division is the other way around.

Now is a good time to open up a Python Shell and have an interactive session to try out some of these operators yourself. To get you started, try completing these two interactive sessions by pressing return after the final line in each case. Don't forget to find out what value is stored in `c` as well.

**INTERACTIVE SESSION**

```
>>> a = 7
>>> b = 3
>>> c = a/b
>>> type(c)
```

**TASK****Task 1**

How can you find out what value is stored in `c`?

**INTERACTIVE SESSION**

```
>>> a = 7
>>> b = 3
>>> c = a//b
>>> type(c)
```

## 3.07 Programming Tasks

**DEMO TASK****Multiply Machine**

Produce a system called Multiply Machine that takes two numbers inputted by the user. It then multiplies them together and outputs the result.

**TIP**

Whenever you are provided with a demo task, it is a good idea to open a new file in script mode and copy in the code provided. Think about what each line of code is doing as you type. Then save the script and try it out.

First you need to design the algorithm. Figure 3.01 shows flowchart and pseudocode solutions for the task.

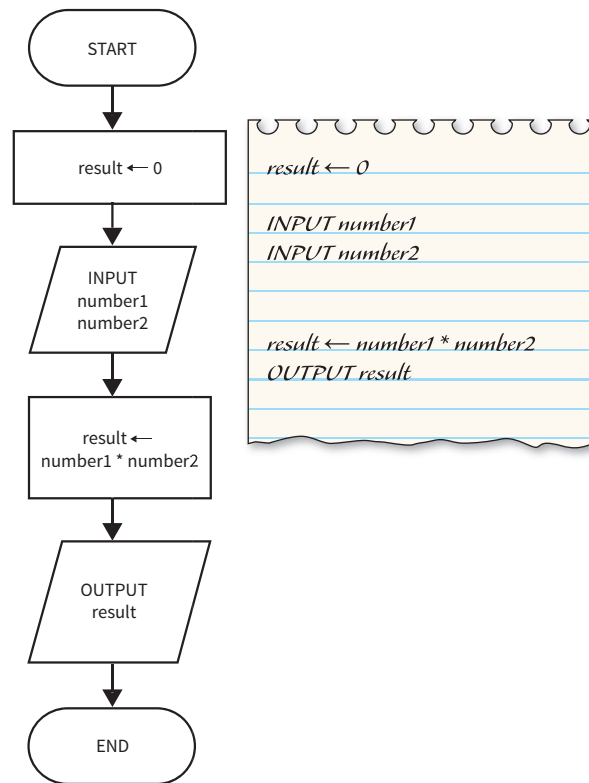


Figure 3.01 Flowchart and pseudocode for multiplication algorithm.

In Python, assignment is indicated by the use of the = symbol. In pseudocode the ← is used.

#### SYLLABUS CHECK

**Pseudocode:** understand and use pseudocode commands INPUT and OUTPUT.



#### TIP

You will want to use Python's `input()` function to send a message to the user and collect their keyboard input. Remember that `input()` only returns String data types, so if you need to do calculations on numbers supplied by your user, you will have to **cast** the String into an Integer by using the `int()` function.

For example:

```
age = int(input('How old are you?'))
```

Here is a Python implementation of the solution from Figure 3.01:

```
# multiply_demo.py

# Request and store user input
number1 = int(input('Please insert first number: '))
number2 = int(input('Please insert second number: '))

result = number1 * number2

# Display the value held in the variable result
print('The answer is ', result)

# End nicely by waiting for the user to press the return key.
input('/n/nPress RETURN to finish.')
```



#### KEY TERMS

**Casting:** The process of changing the data type of a given variable into another data type. For example, a variable that holds the string value '2' could be cast into an integer variable storing the value 2.

#### TASKS

### Task 2 – Addition Machine

Amend the Multiply Machine replacing multiplication with addition.

Will all the numerical values remain Integer data types throughout the life of the application?

### Task 3 – Volume of Water in Aquarium

Design a program where the inputs will be the height, width and depth of an aquarium. The output should be the number of litres of water that the aquarium will hold ( $1\text{ l} = 1000\text{ cm}^3$ ).

### Task 4 – Area and Circumference of a Circle

A system takes the radius of a circle as its input and calculates the area of the circle and its circumference.

- 1 Draw a flowchart and create a pseudocode algorithm that will output the area of the circle and the circumference based on the input radius.
- 2 Test that your algorithm works by programming and running the code in Python.

## 3.08 Development Challenges

Challenge yourself, or your fellow students, to complete a programming task. The following are some examples of the type of task you might like to consider. The last two are complex mathematical challenges.

For each challenge, you should draw a flowchart and create a pseudocode algorithm before programming and running the code in Python.

## Task 5

Program a system that takes as inputs:

- The length of the base of a triangle.
- The perpendicular height of the triangle.

The system should output the area of the triangle.

## Task 6

Program a system that takes as inputs:

- The average speed of a car over the length of a journey.
- The distance that the car has to travel.

The system should output, in minutes, the length of time the journey will take.

## Task 7

Program a system that takes the three inputs required to calculate the area of a trapezoid and outputs the area.

## Task 8

Program a system that takes the length of one side of a regular octagon and outputs the resultant area of the octagon.

**Hint:** To take a square root of a number in Python use this code: `number**0.5`

## Summary

- Programs use variables and constants to hold values.
- Variables and constants have identifiers (names) which are used to refer to them in the program.
- Variables are able to have the value they contain changed during the execution of a program. The values within constants remain the same while the program is running.
- In Python, variable names should be descriptive and consist of lower case words joined by underscores.
- In Python, constant names should contain all capital letters. In Cambridge IGCSE and O Level Computer Science pseudocode, they should be preceded with the `CONSTANT` keyword.
- It is important to know what data types your variables are using. This can be checked by using the `type()` function in Python.
- The `input()` function returns values from the user as String data types. If number inputs are required, the values returned must be cast into Integers or Floats using the `int()` or `float()` functions.
- Mathematical operators can be used with values held in numeric variables.
- Local variables are those that are declared inside a subroutine (see Chapter 4). They cannot be accessed by the rest of the program.
- Global variables are accessed by all parts of a program and are often initialised near the top of a script.
- When designing algorithms, it is crucial to consider the logical sequence of execution. It is important to declare and initialise global variables as well as obtaining user input before completing any processing that requires them.



# Chapter 4:

## Subroutines

### Learning objectives

*By the end of this chapter you will understand:*

- how subroutines are used in programming
- how values are passed to and received from subroutines
- how to design, program and use a function
- how to design, program and use a procedure.



## 4.01 Subroutines

A subroutine is a sequence of program code that performs a specific task but does not represent the entire system.

All subroutines in Python require a name and the keyword `def` which is short for define.

When a subroutine is activated (this is referred to as 'called'), the calling program is halted and control is transferred to the subroutine. After the subroutine has completed execution, control is passed back to the calling program. This modularised approach to programming brings with it advantages over a simple sequenced program.

Consider a GUI program that maintains its running status while waiting for various subroutines to be called by activation of event triggers. The subroutines execute their code and pass control back to the main program.

This allows the programmer to generate the complete program from a series of individual subroutines. Some code is executed when the script is loaded, other elements when certain buttons are clicked and possibly further elements of code are activated when text is changed in a text box. Imagine the complexity of the program code if only a single sequence of code was available to the programmer.

### SYLLABUS CHECK

**Programming concepts:** use predefined procedures or functions.

### Advantages of Using Subroutines

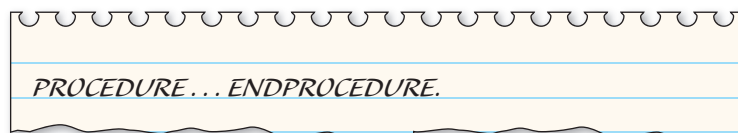
The ability to call subroutines from the main code offers a number of advantages:

- The subroutine can be called when needed: A single block of code can be used many times in the entire program, avoiding the need for repeating identical code sequences throughout. This improves the modularity of the code, makes it easier to understand and helps in the identification of errors.
- There is only one section of code to debug: If an error is located in a subroutine, only the individual subroutine needs to be debugged. Had the code been repeated throughout the main program, each occurrence would need to be altered.
- There is only one section of code to update: Improvements and extensions of the code are available everywhere the subroutine is called.

### Types of Subroutine

Two main types of subroutine exist:

- **Procedures** are small sections of code that can be reused. They do not return a value. In pseudocode, a procedure is named and takes the form:



They are called by using the CALL statement.



- **Functions** are similar to procedures. The difference is that functions have one or more values passed to them and one or more values are returned to the main program after they have completed running. In pseudocode, a function takes the form:

```
FUNCTION(values to be passed in) ... ENDFUNCTION.
```

The CALL statement is used to execute the function but the values required must be passed to the function at the same time:

```
CALL my_function(values required by the function)
```



#### KEY TERMS

**Procedure:** A small section of code that can run repeatedly from different parts of the program.

**Function:** A procedure that returns a value.

## 4.02 Programming a Function

The syntax for defining a function in Python is shown here:

```
def circle(r):
    # code to draw a circle goes here
```

To draw a circle of radius ten in the main part of the program we would write:

```
circle(10)
```

Notice how the radius has been passed to the function at call time and there is no need to use a CALL keyword as is used in Cambridge IGCSE and O Level Computer Science pseudocode; the function name suffices in Python.

### Passing Parameters to a Procedure

The passing of parameters can be very useful. For example, a procedure to check network logon details could take the parameters **username** and **password**. Having checked the data against the logon database, it could return **True** or **False** to indicate if the details match records, or **'update password'** if the password has expired. The procedure could be called repeatedly and passed different parameters every time a user attempts to log on.

Programming a Function is outside the syllabus

### Multiples

A function is required that will be passed an Integer and output the first five multiples of that value.

The pseudocode for this function and its call from the main program are as shown here:

```

FUNCTION multiples(number)
  FOR i = 1 TO 5
    OUTPUT number * i
  NEXT
ENDFUNCTION

CALL multiples(10)

```

This uses a **FOR loop**. FOR loops are introduced more fully in Chapter 7.



#### KEY TERMS

**FOR loop:** A type of iteration that will repeat a section of code a known number of times.

#### TASK

### Task 1

Create a pseudocode algorithm for an amended version of the Multiples procedure that accepts two parameters: a number to use as the multiplier and another to indicate the maximum number of multiplications required.

26

## Functions That Return Values to the Calling Routine

Often programmers write functions that are required to produce answers for repetitive tasks and then return those values to the main program. For example, it might be necessary for a program to calculate the circumference of several circles from their radii.

#### DEMO TASK

### Circumference

A function is required that will be passed the radius of a circle and return the circumference.

The pseudocode for this function and its call from the main program are as shown below.

```

FUNCTION circumference(r)
  c →  $2 * 3.142 * r$ 
  RETURN c
ENDFUNCTION

INPUT radius
circumf → CALL circumference(radius)
OUTPUT circumf

```

Here is a Python implementation:

```

def circumference(r):
    c = 2 * 3.142 * r
    return c

```

```
radius = int(input('What is the radius of your circle? '))
```

```
circumf = circumference(radius)
print('The circumference of your circle is', circumf)
```

Note how the function is not activated by use of the keyword CALL in Python. The name of the function is used as a variable in an assignment statement. Each time the name is used, the function is executed and the return value placed in the variable or output indicated.

## TASK

### Task 2

Create a pseudocode algorithm for an amended version of this function that, when passed the radius, returns the area of a circle.

Test that your algorithm works by programming and running the code in Python.

## Returning Two Values from a Function

It is easy to return two values in pseudocode:

```

RETURN value1, value2

```

In Python, this is accomplished in the same way. Look at this interactive session to see how this works:

**INTERACTIVE SESSION**

```
>>> def my_function():
    return 1,2

>>> a,b = my_function()
>>> print(a)
1
>>> print(b)
2
>>>
```

**TASK****Task 3**

Create a pseudocode and flowchart algorithm for an amended version of the circumference function. When passed the radius, this function returns the area and the circumference of a circle.

Test that your algorithm works by programming and running the code in Python.

**4.03 Programming a Procedure**

The Python code for a procedure is similar to that used for a function. In this case empty brackets are used to show that no parameters are required by the subroutine. See how this works in the interactive session shown below:

**INTERACTIVE SESSION**

```
>>> def greeting():
    print('Hello', 'Hello', 'Hello')

>>> greeting()
Hello Hello Hello

>>>
```

Notice how the `greeting()` function contains the built-in function, `print()`.

**TASK****Task 4**

- a** Create a pseudocode algorithm for a procedure called `dead_end()` that prints out 'I am sorry, you can go no further this way!' This might then be called in a maze game whenever a player reaches the end of a passage.
- b** Test that your algorithm works by programming the procedure in Python and providing a call to the procedure.

Programming a Procedure is outside the syllabus



## Summary

- Subroutines provide an independent section of code that can be called when needed from another routine while the program is running. In this way subroutines can be used to perform common tasks within a program.
- As an independent section of code, a subroutine is easier to debug, maintain or update than repetitive code within the main program.
- Subroutines are called from another routine. Once they have completed execution they pass control back to the calling routine.
- Subroutines can be passed values known as parameters.
- A procedure is used to separate out repetitive code from the main program.
- A function is a type of subroutine which can receive multiple parameters and return values.

# Chapter 5:

## GUI Applications (Optional)

### Learning objectives

*By the end of this chapter you will understand:*

- how to produce and save GUI applications
- how to program windowed applications using the built-in tkinter GUI module
- how to add widgets to a GUI application
- how to lay out widgets in an application window
- how to trigger function calls with buttons.

## 5.01 Introduction

In Chapter 1, you were introduced to interactive mode and script-based programming. This optional chapter shows you how to create programs that appear in windows and have features such as buttons and text boxes. The Cambridge IGCSE and O Level Computer Science syllabus does not require that you produce applications with GUIs; however, you may well want to produce more visually interesting and professional looking solutions to problems. Doing so will also make you a more flexible programmer as you reformat your scripts into GUI applications.

From now on, normal script-based solutions are going to be referred to as text-based solutions and programs that appear in windows are going to be called GUI solutions. After this chapter, you will often be asked to produce two solutions, first a text-based one and then a GUI solution. Producing the GUI programs can be considered optional extensions.

Producing GUI based applications is outside the syllabus.

## 5.02 Make Your First Application in a Window with a Button

By importing the `tkinter` module, it is easy to produce visually rich GUIs and attach your algorithms to buttons in windows.

Tkinter is an example of a GUI toolkit and is provided as part of the standard library when you install Python. Therefore, you already have access to the objects and methods required to make GUI applications, and you just need to do these extra tasks:

- 1 Import the `tkinter` module.
- 2 Create the main tkinter window.
- 3 Add one or more tkinter **widgets** to your application.
- 4 Enter the main event loop, which listens to and acts upon events triggered by the user.



### KEY TERMS

**widget:** Interface items such as buttons and text boxes that can be used to build GUIs.

A button can call for a particular action to happen by referring to a function by name after `command=` in the button definition code. To create the application shown in Figure 5.01 copy the code into a new script and save it as `hello-gui.py` into your Python Code folder:

```

# hello-gui.py

# Import everything required from the tkinter module
from tkinter import *

# Function called by clicking my_button:
def change_text():
    my_label.config(text='Hello World')

# Create the main tkinter window
window = Tk()
window.title('My Application')

# Add an empty tkinter label widget and place it in a grid layout
my_label = Label(window, width=25, height=1, text='')
my_label.grid(row=0, column=0)

# Add a tkinter button widget, place it in the grid layout
# and attach the change_text() function
my_button = Button(window, text='Say Hi', width=10, command=change_text)
my_button.grid(row=1, column=0)

# Enter the main event loop
window.mainloop()

```

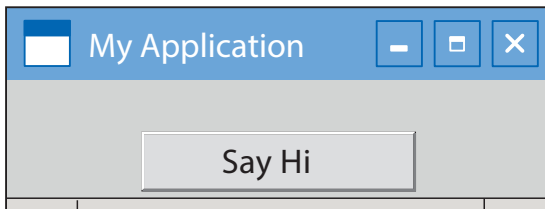


Figure 5.01 A GUI application with a button

After running the code, press the 'Say Hi' button to see how this small application works. Notice how the button is linked to the `change_text()` function by `command=` in the button definition.

#### FURTHER INFORMATION

The tkinter module provides classes, objects and methods that you can access and use in your own applications. Tkinter is written using object-oriented programming (OOP), which is beyond the scope of the syllabus. In OOP programs, object methods are accessed using the dot operator. This can be seen above in the `change_text()` function where the `config()` method is applied to the label widget.

If you want to learn more about OOP you might like to work through *Python: Building Big Apps*, a level 3 book in the Coding Club series, or perhaps try *Introduction to Programming with Greenfoot* by Michael Kölling, which teaches Java programming in a very interactive, game-based way. It is worth noting that while the syllabus focuses on solving problems through a top-down design process, discussed in detail in Chapter 8, OOP is a good example of how to solve problems through bottom-up design.



When laying out GUI applications, you can use the `grid()` method, which organises as many cells as you require in your window using a coordinate system. Note how the numbers start from zero in the top left corner of the window:

row=0, column=0	row=0, column=1	row=0, column=2
row=1, column=0	row=1, column=1	row=1, column=2
row=2, column=0	row=2, column=1	row=2, column=2

It is possible to further arrange tkinter widgets by grouping them in frames.

## 5.03 Other Tkinter Widgets You Can Use in Your Applications

Below are a few other useful widget examples you might want to include in your applications. These code snippets should all be added after `window = Tk()` and above `window.mainloop()` as indicated by the comment in the following recipe for an empty tkinter window:

```
from tkinter import *

window = Tk()
window.title('My Application')
```

```
# widget code goes here
```

```
window.mainloop()
```

A text entry box with a label:

```
Label(window, text='Name:').grid(row=0, column=0)
my_text_box = Entry(window, width=15)
my_text_box.grid(row=0, column=1)
```

Two frames:

```
frame1 = Frame(window,height=20,width=100,bg='green')
frame1.grid(row=0, column=0)
frame2 = Frame(window,height=20,width=100,bg='red')
frame2.grid(row=1, column=1)
```

A drop-down menu:

```
options = (1,2,3)
my_variable_object = IntVar() # access the value with .get()
my_variable_object.set('choose:')
my_dropdown = OptionMenu(window, my_variable_object, *options)
my_dropdown.grid()
```



### TIP

When programming graphical implementations of tasks set in future chapters, remember to consult the Appendix where you will find a full set of recipes for the widgets you will be asked to use.



## TASK

## Task 1 – Tkinter Widgets

Open a new script and add the code from the empty window recipe on page 33. Save this script and then add the code for the example widgets, one at a time, to see how they appear. Do not worry about your scripts doing anything at this stage.

## DEMO TASK

## Gender GUI Application

Create a radio button application that gives the user a choice of two radio buttons to indicate their gender. Your application should show how to align tkinter widgets to the left (West) side of a `grid()` cell. It should also demonstrate how to access the value selected in the radio buttons using a tkinter `StringVar()` object and display the choice made (Figure 5.02).

**TIP**

When building GUI applications, it is good practice to separate the logic from the design. To do this, compartmentalise your algorithm solutions into functions at the top of your script and then build your GUI code at the bottom of your script.

Here is the Python code that demonstrates how to produce a GUI for this very simple one function program.

34

```
# gender-gui.py

from tkinter import *

# Functions go here:
def change_text():
    my_label.config(text=gender.get())

# GUI code goes here:
# Create the main tkinter window
window = Tk()
window.title('My Application')

# Add an empty tkinter label widget and place it in a grid layout
my_label = Label(window, width=25, height=1, text='')
my_label.grid(row=0, column=0)

# Add a tkinter button widget, place it in the grid layout
# and attach the change_text() function
my_button = Button(window, text='Submit', width=10, command=change_text)
my_button.grid(row=1, column=0)

# Create a tkinter string variable object for the radio buttons
gender = StringVar()
```