

Python Challenge

There's no substitute for practice when it comes to learning a new skill!

Python syntax is simple to learn, but becoming an expert in writing programs to solve different kinds of problems takes a bit longer. That's why this book has a short **explanation** of each new statement or technique, followed by one or more **examples** and then loads of practice **challenges**.



Helpful programming tips



Coded solutions with answers given in the back of the book



Starred challenges – Partially written programs for you to complete

Some of the challenges will take you only a minute or two, using the Python Interactive window to try out new statements and get immediate results. As you get further into the book, you will be challenged to write programs to perform different kinds of tasks - for example to find the results of a calculation, write a program for a simplified cash machine, sort a list of items into alphabetical order, or to record data in a text file to be read, formatted, and printed.

The programming solutions to some challenges have been helpfully simplified for an inexperienced programmer to modify rather than to write from scratch. This builds your confidence in problem-solving. That's why **35 challenges consist of partially written programs for you to complete**.

You'll find all the solutions to the Python programs in a free downloadable pack from www.clearrevise.com or www.pgonline.co.uk.



Python Challenge

Learn to program **fast** in:

155 Challenges
53 Examples
85 Pages

Python Challenge

Learn to program **fast** in

155 Challenges
53 Examples
85 Pages

Published by
PG Online Limited
The Old Coach House
35 Main Road
Tolpuddle
Dorset
DT2 7EW
United Kingdom

sales@pgonline.co.uk
www.pgonline.co.uk
2021



PG ONLINE

ACKNOWLEDGEMENTS

Every effort has been made to trace and acknowledge ownership of copyright. The publishers will be happy to make any future amendments with copyright owners that it has not been possible to contact. The publisher would like to thank the following companies and individuals who granted permission for the use of their images in this textbook.

All sections

Design and artwork: Jessica Webb / PG Online Ltd

Graphics / images: © Shutterstock

First edition 2021. 10 9 8 7 6 5 4 3 2 1

A catalogue entry for this book is available from the British Library

ISBN: 978-1-910523-35-3

Contributor: PM Heathcote

Copyright © PG Online 2021

All rights reserved

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the prior written permission of the copyright owner.

Printed on FSC® certified paper by Bell and Bain Ltd, Glasgow, UK.



CONTENTS AND CHECKLIST

			<input checked="" type="checkbox"/>
Introduction	iv	<input type="checkbox"/>
Chapter 1	Getting started.....	1	<input type="checkbox"/>
Chapter 2	Data types and operators.....	11	<input type="checkbox"/>
Chapter 3	Selection	19	<input type="checkbox"/>
Chapter 4	Iteration	27	<input type="checkbox"/>
Chapter 5	Arrays and lists.....	35	<input type="checkbox"/>
Chapter 6	Strings	43	<input type="checkbox"/>
Chapter 7	Subroutines	53	<input type="checkbox"/>
Chapter 8	Turtle graphics.....	61	<input type="checkbox"/>
Chapter 9	Searching and sorting.....	67	<input type="checkbox"/>
Chapter 10	Reading and writing text files.....	77	<input type="checkbox"/>
Syntax guide	90	
Index	92	

start here
Just follow the line



INTRODUCTION

Learn (through explanations and examples)

- Each new concept, Python statement or programming technique is carefully explained and one or more examples given. All the example programs are given in the **Examples** folder of the free **Python programs** pack, and you can copy, run and edit them if you wish.

Practice (with more than 150 challenges)

As each new statement or programming technique is introduced, several practice challenges are given for you to use newly learned skills in writing, editing and running programs.

The challenges are of three different types:

- Short challenges that ask you to write a Python statement or answer a question. You will find the answers to these challenges in the back of the book. 
- Challenges that ask you to write and test a program. For these, you need to create your own folder to write, save, edit and run your programs. You could call the folder, for example, **My Python solutions**. You will find suggested solutions to all these challenges in a folder called **Challenge solutions**. This is a subfolder of the **Python programs** folder. 
- Challenges that consist of a partially written program for you to complete. These are all identified with a star and are held in a subfolder of the **Python programs** folder called **Starred challenges (incomplete)**. You should download this entire folder to your own computer. When you are ready to do one of these programming challenges, copy the incomplete program to your own folder and rename it, e.g. **Chapter 1 Challenge 8 (your initials).py**. Then complete this program and test it. 

Completed solutions to these partially written programs are also held in in the **Challenge solutions** folder. If you get stuck, you can look at the suggested solution, and compare it with yours. There are often several ways of writing correct Python statements to solve the problem, so don't worry if your solution is not identical to the one given. Just make sure yours works correctly!

Review (by selecting topics, regular and starred challenges)

As you work through the book, and the challenges become longer, you will be constantly revisiting statements and techniques you have already covered, using them in different ways. And, if you have already done a Python course and just need to revise what you have already learned, this book is ideal.

Before you start

Download the free Python programs pack from www.clearrevise.com or www.pgonline.co.uk.

There is a quick reference guide to all of the basic Python syntax at the back of this book. Download and run `syntaxsummary.py` from the **Python programs** folder.

GETTING STARTED

Python is one of the most popular text-based programming languages, widely used in schools and colleges, and also in web development, game development, artificial intelligence, business, computer-aided design and many more applications.

The Python language

Python has two modes of entering and running programs. In **interactive mode**, you can type instructions and Python will respond immediately. This is very useful for trying out statements that you are not sure about, and is a good place to start.

Download Python from www.python.org/downloads

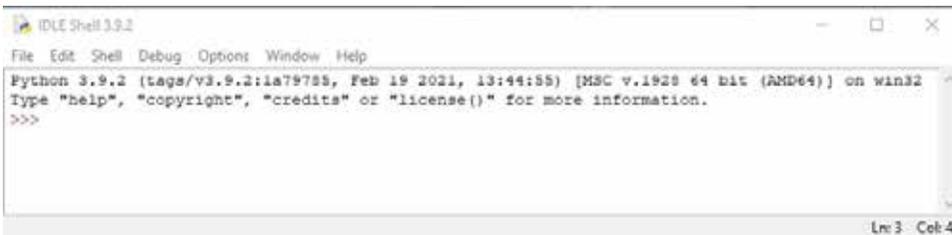


You cannot save a program that you have written in interactive mode. This has to be done in **script mode**, used in the other chapters of this book.

An interactive session

To begin an interactive session, open **Python 3**. On a PC, you can use the **Search** box in the bottom left corner of your screen, or the **Start** menu. Then select the version installed on your computer (e.g. Python 3.9). Load Python's **integrated development environment**, named **IDLE**.

You will see a screen similar to the one below:



In this window, which is called the **Python Shell** or **Interactive window**, you can type commands at the `>>>` prompt.

Example 1

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

The Python interpreter responds to your command and you will see `Hello world` appear on the next line, followed by a new line with the prompt `>>>`.

```
Hello world
>>>
```

Output statement (print)

Use the Python Shell (Interactive mode) for Challenges 1–5.

You can use single or double quotes in the `print` statement. If you need speech marks or an apostrophe inside the text to be printed, use single quotes around the whole text to be printed, and double quotes inside it, or vice versa:

Example 2

```
>>> print('The guide told us "Stay very still! " ')
The guide told us "Stay very still! "
```

Example 3

```
>>> print("A gorilla's diet is mainly vegetarian.\
They feed mainly on stems, bamboo shoots and fruits.")
>>>
```

When you press **Enter**, the computer will print as much as it can on the first line, depending on the size of your window, and continue on the next line:

```
>>> print("A gorilla's diet is mainly vegetarian.\
They feed mainly on stems, bamboo shoots and fruits.")
A gorilla's diet is mainly vegetarian.They feed mainly on st
ems, bamboo shoots and fruits.
>>>
```

Tip

If you have a long line of code, you can split it over two lines by typing a backslash (`\`), and continuing the statement on the next line.

Challenge 1

Write a Python statement to display the line:

```
Gorillas are the largest living primates.
```

Challenge 2

Write a Python `print` statement split over two lines to print on one line. Use the tip above to help you:

```
The scientific name for the Western Lowland
gorilla is "Gorilla gorilla gorilla"
```

Assignment statements

An assignment statement always contains an = sign, but it is not an equation. It means "Take whatever is on the right-hand side of the = sign and put it into the variable named on the left-hand side of the = sign."

Simple assignment statements look like this:

```
age = 17
distanceTravelled = 35.6
street = "Granville Street"
over18 = False
winner = True
count = count + 1
```

The last statement adds 1 to whatever was in the variable `count` and puts the result back in `count`.

Challenge 22

Enter the following statements in the Python Shell (Interactive window).

```
cabFare = 4.50
flight = 91.25
train = 11.80
total = cabFare + flight + train
print("Total travel expenses =",total)
```

This should print:

```
Total travel expenses = 107.55
```



Converting from strings to numbers

In Python, all data items input by the user are treated as **string** variables. Numbers input by the user have to be converted to either integers or floating point numbers before they can be used in calculations.

Function	Description	Example	Returns
<code>float(x)</code>	Converts a string to a floating point value	<code>float("4.65")</code>	4.65
<code>int(x)</code>	Converts a string to an integer	<code>int("3")</code>	3
<code>str(x)</code>	Converts a number x to a string value	<code>str(5.0)</code>	'5.0'

To convert a string to an integer, use the `int()` function.

To convert a string to a floating point number, use the `float()` function.

Inputting numerical data

Example 3



```
#Program name: Ch 2 Example 3 Add and multiply two integers
x = int(input("Enter the first integer: "))
y = int(input("Enter the second integer: "))
sumXY = x + y
product = x * y
print("Sum =", sumXY)
print("Product =", product)
```

Example 4



Below is a Python program to allow the user to enter costs of travel, using the same values as in Challenge 23. The statements appear as follows:

```
#Program name: Ch 2 Example 4 Cost of travel
cabFare = input("Enter cost of the cab fare: ")
flight = input("Enter cost of your flight: ")
train = input("Enter cost of any train fare: ")
total = cabFare + flight + train
print("Total travel expenses: ",total)
```

Run and test the program using 4.50, 91.25 and 11.80 as input values. The output appears as:

```
Total travel expenses: 4.5091.2511.80
```

What has happened? All the data has been entered as strings and concatenated (joined together) in the `print` statement. All the costs need to be converted to floating point values.

Challenge 50

Use the following test data to test your program. (Challenge 49.)

Test	Model	dB level	Expected result	Actual result
1	A	93	"Failed minimum limit"	
2	B	110	"Failed. Requires adjustment or reclassification as Model A"	
3	C	110	"Failed. Requires adjustment or reclassification"	
4	C	126	"Passed"	
5				
6				
7				

Challenge 51



Make up three more tests to test different aspects of the program. Your tests should include valid and invalid data, boundary data and normal data.



Challenge 65

Write a program to enter the names and times in seconds of runners participating in 100m race heats. The program should prompt for a name, then prompt for their time, e.g.

Enter time in seconds for King, R:

The end of data is signalled by entering **xxx** for the runner's name.

Print the average time, in seconds, of all the runners.

Challenge 67

Load **Ch 4 Challenge 67 Double a number repeatedly incomplete** from the **Starred challenges (incomplete)** folder.

The program asks the user to enter a number between 1 and 10. If the number is not in this range, keep printing a warning message and asking the user to re-enter the number. Then keep doubling this number until the result is 100,000 or more. Print the final result and the number of times the number was doubled.

Challenge 66

Load **Ch 4 Challenge 66 Tiles required to cover given area incomplete** from the **Starred challenges (incomplete)** folder.

The program asks the user to enter the size in m^2 of an area that needs tiling. The user then selects the size of tiles they want from a range of sizes given in cm. The program tells them how many tiles they will need. (Ten percent is added to the calculated number to allow for tiles that need to be cut, and this number is rounded down to the nearest whole number.)

Copy and complete the program and save it in your own folder.

Test your completed program by entering 20 cm for tile length, 12 cm for tile width and $2.4 m^2$ for area to be covered. The integer number of tiles required, including the 10% extra, is 110.

Challenge 68

Amend the program **Ch 4 Challenge 68 Double a number repeatedly completed**

so that the user can enter the final target. Include validation to ensure the target entered by the user is between 20 and 100,000. Also include a **print** statement so that the number, and number of times doubled, is printed each time it is doubled.



ARRAYS AND LISTS

5

An **array** is a data structure that stores values of the same data type, such as integers, floating point numbers, characters or strings.

An array is a common data structure in languages such as Visual Basic and Java, but a **list** is a much more common data structure in Python. In Python, a list is used instead of an array.

A Python **list** can store values of different data types, for example:

```
list1 = ["Alan", "Johnson", 27, "M", "Blonde"]
```

Working with lists

You can print the contents of a list with a single **print** statement.

Try this in the Python Shell:

```
>>> list1 = ["Alan", "Johnson", 27, "M", "Blonde"]
>>> print(list1)
['Alan', 'Johnson', 27, 'M', 'Blonde']
```

You can initialise a list like this:

```
planet = ["Jupiter", "Saturn", "Uranus", "Neptune", "Earth", "Venus"]
```

Items in a list may be referred to using an **index**. The first item has an index of 0.

```
print(planet[0]) #will print Jupiter
```

Example 1



Print all the items in the **planet** list, with each item on a separate line.

```
#Program name: Ch 5 Ex 1 List of planets
planet = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn"]
n = len(planet) #sets n equal to number of items in the list
for index in range(n):
    print(planet[index])
```

Challenge

71



Write a single statement to print the list of planets.

List methods and functions

(See Page 45 for comparison of methods and functions.)

Method / Function	Description	Example
<code>list.append(item)</code>	Adds a new item to the end of the list	<code>planet.append("Uranus")</code>
<code>del <list>[<index>]</code>	Removes the item at index from the list	<code>del planet[3]</code>
<code>list.insert(index,item)</code>	Inserts an item just before an existing one at index	<code>planet.insert(3,"Pluto")</code>
<code><item> = list()</code> <code><item> = []</code>	Two methods of creating an empty list structure	<code>planet = list()</code> <code>planet = []</code>

Challenge 72

Write a program to initialise the list of planets:

```
planet = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn"]
```

Use the `list.append()` method to append Uranus and Neptune to the end of list.

Print the list in the format:

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

Challenge 73

Write a program to initialise the list:

```
["Mercury", "Venus", "Mars", "Jupiter", "Saturn"]
```

Use a list method to insert "Earth" between "Venus" and "Mars".

Print the amended list.

Challenge 74

Write a program to initialise and print the list of planets:

```
planet = ["Mercury", "Venus", "Earth", "Mars",  
"Jupiter", "Saturn"]
```

Then print the list with each planet on a separate line starting with Saturn, and ending with Mercury.

Example 7



The program below generates 20 random floating point numbers between 0 and 1. Using these numbers, random numbers between 0 and 200, and between 50 and 250 are generated. The three sets of numbers are printed in neat columns.

```
#Program name: Ch 6 Example 7 Formatting printed columns
import random

for index in range (20):
    num1 = random.random()           #returns a random number between 0 and 1
    num2 = num1 * 200                 #returns a random number between 0 and 200
    num3 = num1 * 200 + 50           #returns a random number between 50 and 250

    #print num1 to 4 decimal places,
    #num2 to 2 decimal places, num3 to 1 decimal place

    #each number occupies 20 spaces and is right aligned
    print("{:>20.4f}{:>20.2f}{:>20.1f}".format(num1, num2, num3))
```

Challenge 102



Write a program that allows a user to enter two numbers **area1** and **area2**, e.g. 2.347 and 1, representing average daily rainfall in two different areas. Print a heading, and print the numbers with two decimal places on one line, each in a 10-character space.

```
Enter average daily rainfall for area 1: 2.347
Enter average daily rainfall for area 2: 1
Area1      Area2
 2.35      1.00
```

Challenge 103



Load the program **Ch 6 Challenge 103 Strawberries incomplete** from the **Starred challenges (incomplete)** folder. Save it in your own folder, and complete the missing lines.

Example 2



```
#Program name: Ch 9 Example 2 Bubble sort (fish)

fish = ["parrotfish", "grouper", "boxfish", "damselfish", \
        "snapper", "ray"]

#get number of items in the list
numItems = len(fish)
passNumber = numItems - 1
swapMade = True
while passNumber > 0 and swapMade:
    swapMade = False
    for j in range(passNumber):
        if fish[j] > fish[j + 1]:
            temp = fish[j]
            fish[j] = fish[j + 1]
            fish[j + 1] = temp
            swapMade = True
    passNumber = passNumber - 1
print("\nSorted list:\n",fish)
```

Challenge 133



How many passes will be required to sort the following list?

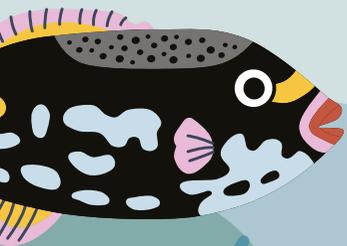
parrotfish, grouper, boxfish,
damselfish, snapper, ray

Challenge 134



Load program **Ch 9 Challenge 134 Bubble sort incomplete** from the **Starred challenges (incomplete)** folder. Amend it so that it prints the state of the list and the value of the Boolean variable **swapMade** after each pass.

Explain why in this algorithm, the last pass does not always change the list.



READING AND WRITING TEXT FILES

10

The programs that you have written so far sometimes use data entered by the user. When the program ends, the data is lost.

Data often needs to be stored on a permanent storage device such as hard disk or SSD, so that it can be read and processed as often as required.

Most real-life applications will store data in a database, from which information can be easily retrieved using queries written in a language called SQL (Structured Query Language).

In this chapter you will practise reading from and writing to **text files**. A text file can be created in Python, or even in Notepad. Each line in the file is a separate **record**, usually consisting of several **fields** of different data types. There are several different ways, in Python, to read records sequentially from a text file.

Reading a text file using `for <line> in <fileid>`

Example 1



A text file has been created to hold the names, genders and ages of seven students going on a outdoor survival weekend.

The program below opens the text file, reads all the records, prints them and closes the file.

```
#Program name: Ch 10 Example 1 print text file
#reads records from a file named students1.txt and prints each record
studentFile = open("students1.txt","r")
for studentRec in studentFile:
    print(studentRec)
studentFile.close()
```

When this program is executed, the output is double-spaced:

```
Diaz,Rosa,F,15
Hamilton,Jerome,M,16
Head,Jennifer,F,16
(etc.)
```

```
students1.txt - Notepad
File Edit Format View Help
Diaz,Rosa,F,15
Hamilton,Jerome,M,16
Head,Jennifer,F,16
Robinson,Keith,M,15
Zelter,Cordelia,F,16
Kelly,Jason,M,16
Browne,Gary,M,15
```

ANSWERS

Note: All programmed solutions are in subfolders of the Python programs folder.

Chapter 1

- Challenge 2: `print(The scientific name for the Western Lowland \ gorilla is "Gorilla gorilla gorilla")`
- Challenge 3: `print("Don't worry.\nBe happy")`
- Challenge 5: Missing quote at end of line. The 'EOL while scanning string literal message' means the debugger reached the End Of Line while scanning the text string and couldn't find an end quote.
`firstName = input("What is your name?)`
- Challenge 7: `animal1`, `animal2` or `animal3` (or alternative variable names that you used)
- Challenge 9: All valid except (c) `1stClass` and (f) `A-1`
- Challenge 10: `10 5`
- Challenge 11: `Fred Tom Fred`
- Challenge 13: (a) Max lives could be a constant in a computer game. (b) Lives remaining will vary during a computer game. (c) Population can vary over time. (d) The unit of gravity on earth is a constant.
- Challenge 14:

```
>>> firstName = input("Enter firstname: ")
Enter firstname: Jo
>>> surname = input("Enter surname: ")
Enter surname: Brown
>>> print(surname,firstName)
Brown Jo
>>>
```
- Challenge 16: (a) Missing quote at the end of a string. (**EOL** stands for End Of Line.)
(b) Incorrectly spelled or undefined variable name.

Chapter 2

- Challenge 18: `print("13685 divided by 27 =",13687//27, " remainder ", 13687%27)`
Result: 13685 divided by 27 = 506 remainder 23
- Challenge 19: `mondayMiles = 20.0`
`tuesdayMiles = 23.75`
`print("Total miles travelled on Monday and Tuesday =",\ mondayMiles + tuesdayMiles)`
- Challenge 20: `print((7*3) + (16/5))`
- Challenge 21: `print("2 to the power 10 =",2**10)`
- Challenge 24: (a) integer (b) string (c) float
- Challenge 27: Fahrenheit and Celsius temperatures are the same at `-40`
- Challenge 28:

```
>>> num1 = 24
>>> num2 = 3
>>> print("The sum of " + str(num1) + " and " + str(num2) + " is " + str(num1 + num2))
The sum of 24 and 3 is 27
```

Challenge 30:

```
x = input("Enter value for x : ")
y = input("Enter value for y : ")
temp = x
x = y
y = temp
print("Swapped values: x =",x, " y =", y)
```

Challenge 33: 20

Chapter 3

Challenge 34:

```
>>> 10 * 7 == 70
True
```

Challenge 35:

```
>>> "John" != "JOHN"
True
```

Challenge 36:

```
>>> (6 <= 2*4) and (28 <= 7*4)
True
```

Challenge 37:

```
>>> (x < 0) and (y < 0)
```

Challenge 38:

```
>>> (x == 0) or (b == 0)
```

Challenge 39:

```
>>> a = 24/8
>>> b = 24 + 8
>>> print(a==b)
False
```

Challenge 40:

```
>>> print(237//17 == 237/17)
False
```

Challenge 41:

```
>>> print(10 == 5*2 and 45 != 9*5)
False
```

Challenge 42:

```
>>> print(6*3 >= 9*2 or 30//7 == 2)
True
```

Challenge 51: For example, use test data such as:
 A, x (Invalid data, expected result – runtime error)
 B, 109 (Boundary data, expected result "Failed. Requires adjustment." message)
 D, 120 (invalid model data, valid decibel data. Expected result error message "Model must be A, B or C. No result given.")

Chapter 4

Challenge 56: These are some sample tests that you could use:

Test	a	b	Expected result	Actual result
1	3	26	13, 26	
2	3	27	13, 26	
3	13	39	13, 39	
4	12	45	13, 39	
5	-30	0	-29, -13, 0	
6	-40	-13	-39, -26, -13	
7	27	13	nothing printed	

Challenge 64:

```
Enter -1 when no more data
Enter number of lengths on day 1: ( etc)
```

Chapter 5

Challenge 71: `print(planet)`

Challenge 76: `totals = [0] * 100`

Challenge 82: `booking[1][2] = 96`

Chapter 6

Challenge 87: `20`

Challenge 88: `7` The "escape sequence" `\n` is counted as one character..

Challenge 89: `name = input("Enter full name")`
`print(len(name))`
`print(name[0])`
`print(name[len(name) - 1])`

Challenge 96: `response.upper() == "Y"`

Challenge 98: It converts the characters entered by the user to uppercase. This means that the user can enter "aa", "AA", "Aa", "ba", etc. for the flight number to be accepted.

Challenge 100: `myStringUppercase = mystring.upper()` is incorrect. It should be:
`myStringUppercase = myString.upper()` (Note the 'S' in `myString`.)

Challenge 101: Test data1: `powerGenerated = abc` (non-numeric value)
Test data 2: `powerGenerated = 75, daysLowWind = 7` (division by zero)

Chapter 7

Challenge 104: `price2 = round(price1, 2)` assigns 5.33 to `price2`

Challenge 105: `print(ord("A"))`
`print(chr(68))` prints D

Challenge 106: `for n in range(0,100,3):`
`print(n)`

Challenge 107: `import math`
`area = 2 * math.pi*5.8`
`print(round(area, 2))`
(prints 36.44)

Challenge 108: `import math`
`print(math.floor(38.0))`
`print(math.floor(53.9))`

Challenge 116: Python will report a runtime error when it reaches the line
`secs = int(time[4:6])`

Challenge 117: (a) Global variables: `title` (defined as `global` in the function) `heading` (declared in the main program)
(b) Local variables `face`, `n`, `dieroll`, `dieface`
(c) A runtime error will occur as the local variable `face` is only recognised within the function.

Challenge 118: `seconds = random.random()* 20`

Chapter 8

Challenge 129: `drawPolygon(100, 3)`

Chapter 9

- Challenge 133: Two passes, and an extra pass to verify that no swaps were made in the previous pass
- Challenge 134: The Boolean variable `swapMade` is set to `False` if no swaps are made during a pass. The `while` loop then terminates.
- Challenge 135: 5 passes though the list of 6 items
- Challenge 136:

```
myList = [3,7,1,9,4,6]
myListLength = len(myList)
temp = myList[2]
```
- Challenge 138:

```
print(searchItem, "was item", index + 1, "in the list")
```
- Challenge 139: (a) 500 (b) 1000
- Challenge 140: Ken Oliver Peter
- Challenge 141: (a) `midpoint = 5` (b) `alist[midpoint] = "Jas"`
- Challenge 142: `print` statement executed four times within loop.
Final `print` statement:
`Item is not in the list`
- Challenge 145: Ch 9 Challenge 145 binary search with errors (in Starred Challenges (incomplete) folder).

Syntax errors:

```
print("loop executed)  missing quote mark
else                    missing colon
```

Logic errors:

```
midpoint = (firstIndex + lastIndex) / 2  should be int((first + last) / 2)
lastIndex = len(aList)                    should be last = len(aList) - 1
```

Runtime errors:

```
print("List of names to be searched:", aList[12])
list index out of range
print("Found at position" + index + "in...")
should be str(index) or use comma instead of +
```

- Challenge 146: The last test results in a runtime error

```
if aList[midpoint] == searchItem:
    IndexError: list index out of range
```


This is because `lastindex` should be initialised as `last = len(aList) - 1`
- Challenge 147: Actual outcome should be as expected for all tests.
You could add tests to test the lower boundary data (1), test outside the top boundary (101), and test an invalid input (not `<`, `=`, or `>`).

Chapter 10

- Challenge 150:

```
Output will be:
Rosa Diaz is 15
years old
Jerome Hamilton is 16
years old
etc.
```

PYTHON SYNTAX GUIDE



Download [syntaxsummary.py](#) from the Python programs folder available from www.ClearRevise.com for a Python version of this quick reference guide.

Input statement

```
name = input("Enter name: ")
visitors = int(input("How many visitors? "))
cost = float(input("Enter cost per person: "))
totalCost = visitors * cost
```

Output statement

```
print("Total cost:",totalCost)

or use concatenation to join two strings:
print("TotalCost " + str(totalCost))

"end" parameter prevents automatic newline
print("Total cost ",end="")
print(totalCost)

newline character "\n" causes a skip to a new line
```

Arithmetic operators

		Result
addition	a = 15 + 2	a = 17
subtraction	b = 15 - 2	b = 13
multiplication	c = 15 * 2	c = 30
division	d = 15 / 2	d = 7.5
exponentiation	e = 5 ** 2	e = 25
integer division (div)	f = 15 // 2	f = 7
remainder (mod)	g = 15 % 2	g = 1

Logical and Boolean operators

		Examples (a = 3, b = 5)	
less than	<	a < b	True
greater than	>	a > b	False
less than or equal	<=	a <= b	True
greater than or equal	>=	a >= b	False
equal	==	a == b	False
not equal	!=	a != b	True
logical and	and	a == b and b == 5	False
logical or	or	a == b or b == 5	True
logical not	not	not(a == b)	True

Definite iteration

```
print numbers 0 to 5:
for n in range(6):
    print(n)

print numbers 5 to 8:
for n in range(5,9):
    print(n)

print every third number between 1 and 10:
for n in range(1,11,3):
    print(n)

print every third number counting down from
99 to 90, counting down
for n in range(99,89,-3):
    print(n)
```

Indefinite iteration

```
daysRemaining = 6
while daysRemaining > 0:
    daysRemaining = daysRemaining - 1
    print("Days to Christmas",\
          daysRemaining)
print("Happy Christmas!")
```

Selection

```
if statement
if (age < 18):
    print("Under age")

if...else statement
if (score >= 50):
    print("Pass")
else:
    print("Fail")

if...elif...else statement
if (score >= 80):
    print("Well done")
    print("Distinction")
elif (score >= 70):
    print("Merit")
elif (score >= 50):
    print("Pass")
else:
    print("Fail")
```

Strings

```
myString = "This is a string"
len(myString) evaluates to 16 (characters in the string)
```

Iterating through a string

```
for nchar in myString:
    print(nchar)
```

Two useful string methods

```
myStringUpperCase = myString.upper()
myStringLowerCase = myString.lower()
```

Slicing strings

e.g. to isolate characters 5 to 8 of myString:

```
chars5to8 = myString[5:8]
```

Turtle methods

Import the turtle module before using any turtle statement:

```
import turtle

turtle.setpos(-300,0) positions the turtle at (-300,0)
turtle.home() positions turtle at (0,0)

turtle.forward(100)
turtle.right(90) turns through 90 degrees clockwise
turtle.left(30) turns through 30 degrees anti-clockwise

turtle.penup() lifts the pen
turtle.pendown() puts the pen down
turtle.pensize(10) sets the pensize
turtle.pencolor("red") sets the pen colour
```

Lists

```
myList = ["Bob", "Mandy", "Fred", "Jo", "Keira"]
print(myList[0]) prints Bob
print(myList) prints the whole list
print(len(myList)) prints 5 (the number of items in
the list)
aList = [] * 100 initialises an empty list of 100 items
```

Two-dimensional lists

A quiz has 4 rounds and 3 teams
Results are entered in a 2-D list.
Row indices go from 0 to 2 (teams)
Column indices go from 0 to 3 (rounds)

```
quizResults = [[ 5, 7, 9, 4], (row 0)
               [12,14,10,18], (row 1)
               [27,21,23,20]] (row 2)

print(quizResults[1][3]) prints 18, the result in second
row and fourth column
```

Subroutines

Function (returns a value)

```
def myFunction(a,b):
    x = a + b
    return(x)
```

To call the function:

```
sum = myFunction(3,4)
```

Procedure (does not return a value)

```
def myproc():
```

or pass parameters to the procedure:

```
def greet(greeting,name):
    print(greeting, name)
```

To call the procedure:

```
greet("Hello", "Jane")
```

Random module

```
import random imports the module
random.randint(a,b) returns a random
integer X so that a <= X <= b
random.random() returns value from 0.0-1.0
```

Text files

Opening a text file

A text file can be opened for reading, writing or appending data

```
gameFile = open("gamescore.
txt", "r") opens the file for reading
```

Set the final parameter to "w" to write a new file or overwrite an existing one

Set the final parameter to "a" to append data or create a new file if no file can be found

Closing a text file

```
gameFile.close()
```

Reading a text file

```
gameFile.read() reads the entire file
gameRec = gameFile.readline()
reads one line of a text file. This returns
an empty string "" on reaching the end
of file
```

Writing or appending a line to a text file

```
gameFile.write(field1 + field2 +
...
+ "\n" )
```

INDEX

0-9

2D lists 39

A

arithmetic operators 11

array 35

assignment statement 14

B

backslash (\) 2

binary search 71

Boolean

 data type 11, 19

 expression 19

 operator 19

bubble sort 67

C

camel caps 7

cancel a running program 25, 33

casting 16, 31

chr() function 53

colours, syntax 5

comments 5, 6

comparison operators 19

concatenation 10, 17, 31, 90

constants 8

convert between data types 15

D

data types 11

div operator (div) 11

E

Editor window 4

elif 23, 90

errors 74

 logic 74

 runtime 50, 74

 syntax 74

F

field 79

file (see *text file*)

for loop 27

format() method 51, 85

formatting output 51, 85

functions 36, 53, 56

G

global variables 59

I

identifier 7

IDLE 1, 4, 5

if .. else 22

indentation 22

indexing strings 43

initialise 36

input statement 9

integer division (div) 11

Interactive mode 1, 4

iteration 27

 definite 27

 indefinite 30

K

keywords 5

L

library modules 54

list 35

 methods and functions 36

local variables 59

logical operators 19

M

math module 54

methods 36

mod operator (mod) 11

N

nested for loop 29

nested if statements 24

newline

 character 3, 43, 78

 prevent 28

O

ord() function 53

output statement (print) 2

P

PI 13

prevent automatic newline 28, 90

print statement 2, 13

procedures 53

programming constructs 19

Python Shell 1

R

random module 60

range() function 27, 53

readline() 80

readlines() 81

relational operators 19

remainder (mod) 11

repetition 27

reserved words 5

round() function 31, 53

runtime error 50, 74

S

script mode 1

searching algorithms 70

selection 22

sequence 19

Shell (Interactive window) 1

slicing 46

sorting algorithms 67

split(<char>) method 79

splitting output over two lines 3

starting a new program 4

stop a running program 25, 33

string

 indexing 43

 iterating through 43

 methods 44

 slicing 46

 str() function 15

 variable 9

strip(<char>) method 79

syntax errors 6, 74

syntax highlighting 5

T

terminate a program (Ctrl-C) 25

testing 25, 75

test plan 76

text file 77

 append 83

 close 78

 open mode 83

 read 77, 78, 80, 81

 write 83

time module 55

turtle module 61

two-dimensional lists 39

V

validation 48

variable 7

 local and global 59

 names 6, 7

W

while loop 30