Workbook: Programming in Python (Version 3)

Brandon Bennett
B.Bennett@leeds.ac.uk

Sam Wilson
scsswi@leeds.ac.uk

Elaine Duffin
sc10ed@leeds.ac.uk

Sarah Cook
sc10sc@leeds.ac.uk

Semester 1
2014-2015

Preface

This workbook has been designed to accompany a series of lecture to teach programming in Python (version 3). The workbook contains a set of exercises that roughly follow the content covered in the lectures. Most questions are compulsory, but some optional questions are also given. You are encouraged to attempt as many of these as you can.

It is expected that everyone will complete the compulsory of questions within a reasonable time frame. The last set will be more demanding and although we encourage everyone to attempt them they may take more time. You will be expected to demonstrate that you have completed the exercises from the previous week in the lab classes.

The symbols in the margin next to each question will indicate how challenging the question is and whether it is compulsory or optional. The meaning of the symbols is as follows.

These questions should not take too long to complete (Compulsory).

These questions will take some time to complete (Compulsory).

These questions will require some additional reading to answer (Optional).

These questions require a level of maths past that of GCSE (Optional).

These questions look into advanced programming concepts or peculiarities of how Python works and are only intended for particularly curious students (Optional).

It is important that you remember two things when learning to program. You have to be inquisitive and explore your understanding of what you have just learned. (You will not break the computer by changing lines of code.) Secondly, learning to program will not happen overnight; the only way to learn to program and to think like a programmer is to read and write code. You must practice programming by finding as many examples as you can and 'playing' around with them. Just because you have completed the set of exercises for a week does not mean your learning must stop there. We encourage you to read the books on the reading list and attempt questions given in them.

Many of the exercises in this workbook are quite typical of those found in introduction to programming courses; and as a result you will easily find sample solutions by using your favourite search engine. However you will not learn to program by simply copying and pasting from websites. By all means use the internet to supplement your learning, but when it comes to answering the exercises, you should attempt them yourself before resorting to a search engine.

The suggested reading to accompany this module is:

- Practical Programming — An introduction to Computer Science using Python 3
- How to think like a Programmer — Problem solving for the bewildered
- How to think like a Programmer — Solutions for the bewildered

Lectures and Lab classes

Type	Time/Location
Lectures	
Lab classes	

You will have been assigned to one of two lab sessions that will be held each week. You may also attend the other Lab class if you require further help with any of the exercises or coursework. But, in the unlikely event that there are no spare machines, priority will be given to those students attending their scheduled lab class.

Coursework

There will be coursework for this part of the module, the following information is regarding hand in dates and the percentage towards the overall module.

Assessment	Hand out date	Hand in date	% of module grade
Lab Exercises (this workbook)	Weeks 2–7	Week 7	5%
Coursework 1	Week 5	Week 7	10%

Week 1

Getting started with Python (V 3)

AIMS:

- to be familiar with executing Python instructions
- to be familiar with using *Idle* (Integrated DeveLopment Environment)
- to author your first Python programs.

The exercises this week give an introduction to writing and executing Python 3 instructions and creating simple programs using the *Idle* (Integrated DeveLopment Environment) development tool. To complete this workbook on a non School of Computing machine you will need to ensure you have Python 3.x installed. This can be downloaded from:

https://www.python.org/download/

Preliminaries

First we will need to start the Python programming environment. We will be using an *integrated development environment* (IDE) called *Idle*. Snippets of code in this workbook contain a faint '␣' character to denote the space character in the source code, every time you see a '␣' type a single space character. To start *Idle* using python 3, open a terminal by clicking **Main Menu ->System Tools ->Terminal** and type in the following command and press enter;

```
1 idle3 &
```

After executing this command a window will appear, this is *Idle*'s Python interactive shell window. The Python shell is interactive meaning you can type Python code into it and the Python interpreter will execute the instructions. Typically there is a result of executing an instruction, the result will be displayed in the interactive Python shell window. The last line of the interactive Python shell will be '>>>', this is the Python prompt inviting you to enter some Python code.

Let's give it a go, enter the following lines into the Python shell (you should press enter at the end of each line and the Python interpreter will display the result, you will need to press enter twice for the last instruction);

```
1  print("Hello World!!!")
2  1 + 1
3  2 * 23
4  7 / 4
5  7 % 4
6  2 ** 2
7  name = "Snake master"
8  print("hello" , name)
9  y = 53
10 z = y
11 print("The value of z is",z)
12 for x in range(10):
13     print("Python is the best")
```

examples/snippet1–1.py

If you encounter an error while typing the above snippet, check carefully what you have typed including punctionation and whitespace.

1. What value does variable z have on line 11 have?

2. What is the output from line 5 of snippet1-1?

3. Try entering some more lines with the same form as line 5 but with different numbers. What does the % operator do?
   Use the space below to write a short explanation of the % operator:

4. What is the output from line 6 of snippet1-1?

5. Try entering line 6 using different values, What does the ** operator do?
   Give a brief explanation of this operator:

6. What code should you enter to make the Python shell output 'Hello, <*your name*>'?

7. Assuming that you know your age in years and also the number of months remaining until your next birthday, how might you use the interactive Python shell to calculate your age in months? Write the code that you would use. Note down any problems or complications that might affect this computation.

Now that we are familiar with the interactive Python shell we will progress to writing a program into a file. From the previous examples hopefully you can see that the interactive Python shell is like a very sophisticated calculator.

Creating a program in a file

The interactive Python shell is useful for experimenting with short snippets of code but for long programs it is beneficial to be able to save the code into a file so that it can be run again later. This allows us to change, reuse and develop programs. To create and run a program in *Idle*, carry out the following steps:

— Open an editing window in *Idle* by clicking on the 'File' menu at the top of the window and select 'New Window' option. A new window should appear which is mostly blank. This is where you will enter your program.

— Enter a few lines of Python code (maybe reusing some of the code from snippet1-1) into the window.

— Save your program by selecting save under the 'File' menu. Ensure that you give it a suitable name.

— It is now possible to run the program. From the 'Run' menu select the 'Run Module' option. The output from the program should appear in the python shell. It is possible that you will see an error message, if this is the case edit the program. (If you cannot get rid of the error try using the exact instructions from snippet1-1)

Idle uses most of the common shortcuts — for example *Ctrl+s* to save. The *F5* key will run the file you currently have open. Using shortcuts will drastically reduce the time you spend writing programs so it is worth learning them. Throughout this workbook there are many tasks to complete and most of the tasks require you to write programs. It is essential that you give your programs sensible filenames so that you can come back to them later. It may be worth writing the name of the file in the margin next to the tasks or exercise so you can come back to it later.

8. Using the code from snippet1-1 modify the program to print your name out 13 times .

9. Using the code from snippet1-1 write a program that prints your name followed by your age in months.

10. Write a program that prints out your name in acsii art.

```
print(" CCC  OOO  M   M PPPP   11  5555 5555  11  ")
print("C    O   O MM MM P   P 111  5    5    111  ")
print("C    O   O M M M PPPP   11  555  555   11  ")
print("C    O   O M   M P      11     5    5  11  ")
print(" CCC  OOO  M   M P      1111 555  555 1111 ")
```

examples/snippet1–ascii.py

A simple program involving user input

A program can seem a little useless if it does not interact with the user, in the next section we will cover how to get the program to prompt the user for input. Create a new file in *Idle*, the same way as you did in the previous task and type the following code. Once you have typed in the code press *F5* and see what the application does.

```
1  ## Program for computing and displaying squares of numbers
2  print("*********************************************")
3  print("Welcome to the Amazing Square Computing Program")
4  print("Enter any whole number: ", end = "" )
5
6  # Reads in a string of characters
7  numstring = input()
8
9  #Converts the string to a whole number
10 number = int(numstring)
11
12 # Prints a string to the shell
13 print("The square of", number, "is", number * number, "!!!")
```

examples/snippet1–2.py

Let's have a look at the program to see what it does. There are a couple of interesting things which are noteworthy. The symbol # denotes a comment. The remainder of the line after the symbol is a comment. Observe how the code has comments throughout it to help you understand what is happening. Comments are ignored by the Python interpreter, so you can type anything in a comment. When you are completing the exercises in this workbook, you should provide comments in all your code. This will help you to revise later, and will also help the module staff understand what your programs are doing.

On line 7 we see the instruction:

```
numstring = input()
```

This instruction prompts the user to enter a value, when the user presses enter the value is passed to the program and the program continues to execute the instruction. The variable called `numstring` will contain the value the user entered. The `input` command gets a string from the user. In order to be able to do calculations we must convert the string into a number. This conversion is done on line 10.

Now that we understand what snippet1-2 is doing let's make a few changes to alter what it does.

11. Modify the code from snippet1-2 so that it gets two numbers from the user and prints out the result of multiplying them together (Hint: you will need two variables to store each of the users inputs).

12. What instruction would be used if we wanted to calculate the result of the first number divided by the second?

13. What instruction would we use if we wanted to print out the result of raising the first number to the power of the second?

14. What happens if you run snippet1-2 and enter an input which is not a whole number? (What input did you enter? What happened?)

Before we move on to learning Python there are a couple of things that you should remember and which will save you time. Python is a case-sensitive programming language, that is you have to be careful with variable and function names. The following example demonstrates this:

```
1 number = 3
2 Number = 4
3 print(number)
4 print(Number)
```

This shows that the variables `number` and `Number` are different.

15. What is the output from the above code?

The next thing to remember is that Python is whitespace sensitive, that is spaces and tabs have a meaning. We will learn more about this in later weeks. If you get get errors from your Python code regarding indentation error it is likely that you have too many spaces (or not enough). You may indent your code using tabs or spaces, but remember: do not mix them together. Stick to either tabs or spaces, in this workbook we will use spaces.

Data types and operators

AIMS:

- to be aware of the different data types in Python
- to be able to select an appropriate data type for a value
- to be able to use operators to manipulate data types
- to be aware of the Python documentation

In this set of exercises we will look at the different data types Python has built in and we shall learn about the operators for manipulating them. Python's built in data types can be separated into six categories;

- Numbers
- Strings
- Lists
- Tuples
- Dictionaries
- Boolean

We shall cover all of the types here and provide examples of what kind of values they can store. To complete this set of exercises you will need to open idle, most of the exercises will require the use of the interactive Python shell. All variables in Python have a type associated with them, unlike some languages Python is dynamically typed so you don't have to explicitly define the type when you create a variable. The syntax for creating a variable is;

```
<<variable_name>> = <<value>>
```

where '<<variable_name>>' is any sequence of letters, numbers or underscore characters, but cannot start with numerical character. Similar notation is used in this workbook, wherever you encounter <<something>> it can be replaced for a user defined sequence of characters. Python has a set of *keywords* that cannot be used as variable or function names. These 'reserved' words are listed in Table 1.1 below.

It will be useful to give variables sensible names to make the code easier to read. A variable name should not only make sense in the context of the program but should also be meaningful to anyone else reading the code.

The '=' symbol is called the assignment operator and should be interpreted as "set x to be equal to 5" (for the above example). A variable cannot be used in an instruction until it has been defined and the variable must also be in *scope*. The section of a program within which a variable

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

Table 1.1: Reserved words in Python 3

can be accessed is called the variable's scope. We will cover more about scope when we get to loops and functions.

16. Which of the following are valid names for variables in Python? (In each case state if it is a valid name or why it is not a valid name).

    (a) `temp`

    (b) `Fred Flintstone`

    (c) `123xy`

    (d) `x_coordinate`

    (e) `Break`

    (f) `newVariable`

    (g) `ALLCAPS`

    (h) `except`

    (i) `_`

Python associates a type to every variable, to find out what type Python has associated with a variable named x you can type:

```
1 x = 5
2 type(x)
```

17. What is the type of the variable x in the above example?

Numbers

Python has several data types for storing number values. Much like in mathematics where we have integer, rational number, irrational number, real number and complex number (to name a few). Python has different data types for storing different types of numbers (visit wikipedia if you are unsure about these types of numbers, or better still ask your neighbour). Python 3 supports the following number data types;

int — int can hold integer valued number of unspecified length e.g. -17, -1, 0, 2, 5, 10, 42, 15647989
(See http://en.wikipedia.org/wiki/Integer)

float — float can hold floating point number allowing the approximate representation of real numbers
(See http://en.wikipedia.org/wiki/Real_number)

complex — complex can hold a complex number
(See http://en.wikipedia.org/wiki/Complex_number).

To create a variable of type 'int', use the assignment operator and set the variable to any integer value. For example;

```
1 x = 5
```

A floating point number can be considered as being a representation of a real number, that is any decimal number. To create a variable of type 'float', use the assignment operator and set the variable to any floating point value. For example;

```
1 x = 4.55555678
2 y = 5.0
3 z = 5.67e-07
```

A complex number can be thought of as a composite number comprising a real part and an imaginary part. Imaginary numbers arise when we take the square root of negative number. To create a complex number, use the assignment operator and set the variable to any complex number. The imaginary part has the letter 'j' appended to it. For example

```
1 x = 4.3+6j
```

Using the interactive Python shell complete the following exercises:

18. What instruction will create a variable called `k` and assign it the value of 42?

19. What is the result of typing the following (assuming you have created a variable as in the previous question)?

```
type(k)
```

What instruction will print the value of `k` ?

20. Create a variable called `pi` and assign it an approximate value of π, What is the type of `y` ?

```
y = 42 + pi
```

22. What is the instruction to create a variable of type complex with a real part equal to 3 and an imaginary part equal to 8 ?

23. What number type in Python would be most suited to storing the value of 1/3 ?

24. What number type in Python would be most suited to storing the number of customers of a business?

25. What number type in Python would be most suited to storing the cost of a pint of your favourite beverage?

26. What number type in Python would be most suited to storing the *roots* of the following equation. Give a Python instruction that will create a variable and set its value to one of the roots of the equation. (There are two roots and the code for calculating either root is very similar, so just choose one.)

$$x^2 + 2x + 10 = 0$$

Now we know about the different ways of representing numbers in Python let us look at operators on them. Table 1.2 outlines the operations defined on number types.

Operator	Operation
$x + y$	Addition of x and y
$x - y$	Subtraction of y from x
$x * y$	Multiplication of x and y
x / y	True division of x by y. The result of this operation is always a floating point number
$x // y$	Floor division of x by y. The result of this operation is the integer part of the division
$x\%y$	Modulo. The result of this operation is the remainder when x is divided by y
$-x$	Negation of x
$x ** y$	Has the value of x raised to the power of y — i.e. x^y.
$x == y$ $x < y$ $x <= y$ $x > y$ $x >= y$	Comparisons

Table 1.2: Operators and their meaning on Number types

The precedence of operators in Python is the same as in mathematics (Brackets, Order, Division, Multiplication, Addition and Subtraction). To force the evaluation of an expression in a certain way or if you are unsure about the precedence of operators use round brackets.

27. What instruction would you type to calculate the remainder of 7 divided by 5?

28. What is the type of the result of adding two integer values?

29. What is the type of the result of adding a float to an int?

30. What is the type of the result of multiplying a float with a complex number?

31. What instruction would type to calculate the following expression (you should get the answer 65536)?

$$2^{2^{2^2}}$$

32. Using the interactive Python shell, what instruction calculates the following? what is the type of the result (you should get the answer 177147)?

$$\frac{(4+5)^5}{9^{-0.5}}$$

33. Using the interactive Python shell, what instruction calculates the following? what is the type of the result (you should get the answer 14)?

$$\frac{4^3+6}{9^{0.5}+2}$$

34. Using the interactive Python shell calculate the cost per person of a meal if the total was £56.70 and there were 3 people eating, What instruction calculates the cost?

35. From the above question; how much does it cost? What is the type of the result?

36. What instruction would you use to calculate 67% of 143? What is the type of the answer?

37. Convert the following formulae into Python syntax

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

where a, b and c are variables.

38. Write a program that calculates the force between two objects. The program should prompt the user for the distance between the objects, in metres, and the mass of the objects in kilograms. To calculate the force the equation is;

$$F = G\left(\frac{M_1 M_2}{d^2}\right)$$

where G is the universal gravitational constant equal to $6.67 \times 10^{-11} (m^2 kg^{-1} s^{-2})$.
Your program should print the appropriate unit of force next to the result.

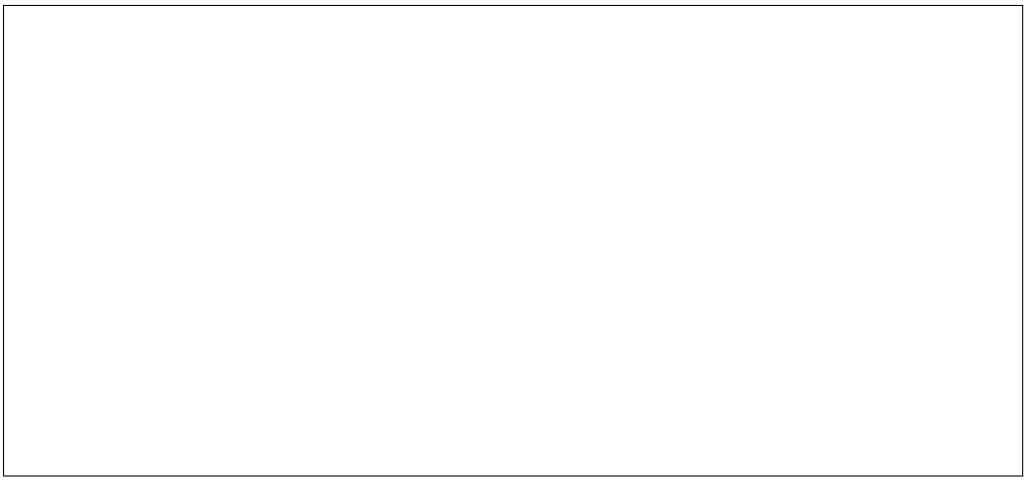
39. Write a program that calculates the cost of a pizza per square centimetre. The program should prompt the user for the radius, in centimetres, and the price and print out the cost of the pizza per square centimetre.

40. Write a program that calculates the distance a car can travel without refuelling. The program should prompt the user for the capacity of the tank, in litres, and the number of kilometres the car can travel per litre.

41. Write a program that can convert seconds into hours, minutes and seconds. The program should prompt the user for the number of seconds and should print the result in the following format “HH:MM:SS”.

42. What is the type of the result of the comparison operators?

43. From the table above, work out the meaning of the comparison operators (Write down what values you tried and the output).



44. How is a floating point number represented by a computer? (Provide an example with your answer)

Strings

Python represents sequences of characters as strings. To create a variable containing a string the assignment operator is used. Strings are identified in Python by the use of speech marks or single quotation marks. For example

```
s = "This is a String"
```

The result of this instruction is that `s` will be a variable of type string (denoted *str*) with the value “This is a String". Using the interactive Python interpreter complete the following exercises.

45. What command creates a variable called ‘name’ containing a string whose value is your name?

Python represents its strings as a sequence of characters, each character can be indexed starting from 0 to the length of the string minus 1. For example the string "Hello world!!!" is represented as follows;

H	e	l	l	o	␣	w	o	r	l	d	!	!	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13

The + operator concatenates two strings together. The elements of the sequence can be accessed using square brackets and an index.

- Using the interactive Python shell type the following snippet.

```
s = "This is a String"
print(s[0])
```

46. What is the result of the above code?

47. What is the type of the result of accessing an element of a string?

In Python you can use negative numbers as the index to an element of a string. With a negative index the index counts from the end of the string towards the first element.

- Using the interactive Python shell type the following snippet.

```
s = "Hello World!!!"
print(s[-7])
```

The output of these instructions will be the letter 'o' as it is the seventh letter from the end of the string.

Assume that `s = "Hello world!!!"`, answer the following questions.

48. Using the interactive Python shell, what is the result of `s[4]`?

---

49. What are the four possible indices one could use to get a letter 'o' ?

---

50. Using the interactive Python shell, what is the result of `s[-4]`?

---

51. Using the interactive Python shell, what is the result of `s[-0]`?

In Python the colon ':' allows the square brackets to take two numbers, called parameters. The two parameters are the indices of the first and last elements of a *slice*. Using the interactive Python shell type the following.

```
s = "Hello World!!!"
print(s[3:7])
```

The above snippet will print the string "lo W". Assuming that `s = "Hello World!!!"` complete the following exercises using the interactive Python shell.

52. What is the output of the following snippet?

```
print(s[:6])
```

---

53. What is the output of the following snippet?

```
print(s[9:-7])
```

---

54. What is the output of the following snippet?

```
print(s[-7:9])
```

---

55. What instruction would result in the string '!!!'?

---

56. What instruction would result in the string 'World'?

---

57. What is the type of a slice of a string?

Strings in Python have a number of methods that can be invoked to get results. Methods may be invoked using the following syntax;

```
<<variable_name>>.<<method_name>>(<<parameter_list>>)
```

Lets look at a concrete example, the following snippet demonstrates how to use the `islower` method. The `islower` method will result is `True` if all characters are lower case.

```
str = "Hello world"
str.islower()
```

Table 1.3 outlines some of the commonly used methods on strings. The full list of methods can be found on the Python documentation website. Have a look at the documentation for the string data type:

https://docs.python.org/3.2/library/stdtypes.html#string-methods.

Method name	Description
`lower()`	Converts the string to be lower case
`count(x)`	Counts the occurrence of x
`index(x)`	Finds the first occurrence of x
`replace(x,y)`	Replaces all instance of x with y

Table 1.3: Useful method on strings

Using the interactive Python shell type the following snippet;

```
1 s = "HeLlO wOrLd"
2 print(s.capitalize())
3 print("The letter l occurs ", s.count('l'), " times")
4 print("The first w is at index", s.find('w'))
```

examples/snippet1–3.py

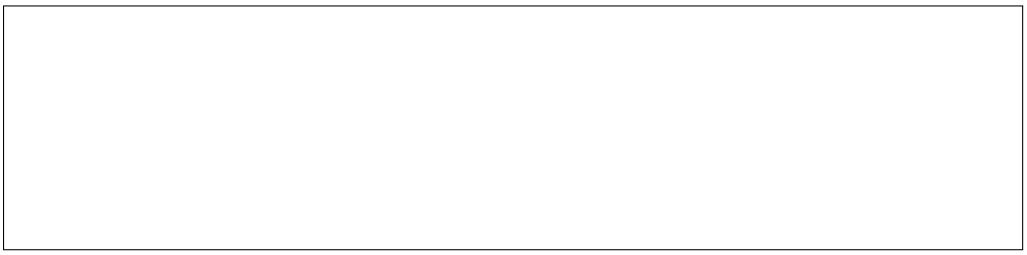
58. What does line 2 of snippet1-3 print?

59. What does line 3 of snippet1-3 print?

60. What does line 4 of snippet1-3 print?

61. Using the Python documentation for string, what method will swap the case of the string? What is the result of invoking the method on the variable `s` from snippet1-3?

62. Using what you have learned from the earlier sections write a program that prompts the user for their firstname and their lastname and prints a personalised salutation to them where the first letter of their first and last name is capitalized.



63. What method did you use to capitalise the name?

Lists

A list in Python is an ordered sequence of elements, an element can be of any type (including a list itself). The notation for constructing a list is;

```
<<variable_name>> = [<<element_1>>,...,<<element_n>>]
```

Let's have a look at a concrete example.

```
#Create a list with elements of different types
list = [1,2,3,"Hello",[1,"world"]]

#Loop through the elements of the list printing each element
    and its type
for item in list:
    print(item, " is of type ", type(item))
```

examples/snippet1–4.py

Note that the types of the elements of a list can be different.

64. Using snippet1-4 as a starting point, make a list that contains the first 5 integers starting at 0 then using a `for` loop print the square of the elements of the list.

65. Using snippet1-4 as a starting point, make a list containing the first 5 integers starting at 0 then using a `for` loop sum the elements of the list and print the result (Hint: you will need an extra variable to store the total).

Lists are *mutable*, that is, it is possible to add, remove and edit elements of a list. Just like strings the elements of a list can be accessed using square brackets. The following snippet shows some of the common methods used on lists. The first element of a list has index 0.

```
#Create a list with elements of different types
list = [1,2,3,"Hello",[1,"world"]]

#Remove the element "Hello" from the list
list.remove("Hello")

#Insert a new element at a specific index
list.insert(3, 2009)

#Append a new element to the end of the list
list.append("Python is great")

#Prints the element at index 1
print(list[1])
```

examples/snippet1–5.py

It is possible to *sort* (i.e. re-order into a standard ordering) a list in Python by using the `sort` method. Below is an example of how to use the `sort` method.[1]

```
list = [12,4,7,12,99,1000,1827684]
list.sort()
print(list)
```

The online Python documentation for lists can be found here:

https://docs.python.org/3.2/library/stdtypes.html#mutable-sequence-types

and contains a full list of methods that are available on the list data type.

66. Using the interactive Python shell, What is the result of the following snippet? What does the * operator do on lists?

```
[1,2,3] * 5
```

67. Using the interactive Python shell, What is the result of the following snippet?

```
[1,2,3] + [4,5,6]
```

68. By experimenting with the snippet above work out what the + operator does on lists?

69. What instruction would generate a list containing ten integer elements whose values are 0?

Python supports a concept called *list comprehension* which is a neat and concise way of constructing lists. Although it is not strictly required, for those that are interested a Python tutorial on list comprehension can be found at;
https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions.

Try completing the following exercises.

70. Using list comprehension, what instruction constructs a list containing the first 10 even numbers start at 0.

71. Using list comprehension, what instruction constructs the list `["1a","2a","3a","4a"]`.

72. Using list comprehension, what instruction constructs a list containing the first 100 square numbers.

---

[1] Note that `L.sort()` actually changes the value of the list referred to by `L`. Alternatively, you can make a sorted copy of a list by using a command of the form `S = sorted(L)`. This will set variable `S` to refer to a new sorted list with the same elements as `L`, but will not change the value of `L` itself.

Tuples

Tuples in Python are much like lists except that they are *immutable*. Once a tuple has been created it is not possible to add or delete elements, or modify the value of its elements. This immutability may seem like a restriction that makes them useless however the property of being immutable is beneficial when tuples are used as keys in dictionaries (dictionaries will be covered in the next section). Tuples may be created directly or converted from a list. Using the interactive Python shell enter the following snippet.

```
#Create a list with some elements of different types
l = [1,'a',[1,2,3,4]]

#Create a tuple with some elements of different types
t = (1,'a',[1,2,3,4])

#Construct a tuple from a list
TupleFromList = tuple(l)

#Construct a list from a tuple
ListFromTuple = list(t)
```

examples/snippet1–6.py

73. What is the type of `t`?

74. What is the type of `ListFromTuple`?

75. What is the type of element at index 2 in `t`?

76. Using the notation from lists, try and append an element to a tuple. What error do you get?

77. What is the result of trying to modify a tuple?

Dictionaries

A dictionary in Python is a collection of unordered values and associated keys. Unlike with lists where elements are accessed by an index, dictionaries use a key of almost any type to address an element stored in the dictionary. In other languages dictionaries are called Maps and Hashtables. There are a number of ways of constructing dictionaries.

```
#Create an empty dictionary
dict1 = {}

#Create a dictionary containing two element
dict2 = {"one":1, "two": 2}

#Adding an element to a dictionary
dict2["three"] = 3

#Accessing elements of a dictionary
print(dict2["one"])

#Looping through the keys of a dictionary
for key in dict2:
    print(key , " ", dict2[key])

#Looping through the values of a dictionary
for value in dict2.values():
    print(value)

#Looping through the key/value pair tuples of a dictionary
for item in dict2.items():
    print("Key: ", item[0], " Value: ", item[1])

#Get the number of items in a dictionary
len(dict2)

#Remove a key from a dictionary
dict2.pop("three")

#Delete all element in a dictionary
dict2.clear()
```

examples/snippet1–7.py

Using the interactive Python shell complete the following tasks then answer the exercises.

- Create an empty dictionary and assign it to a variable called 'phonebook'.
- Add some elements where the key is a name and the value is a phone number.
- Change the phone number of the first person you added.

Complete the following questions.

78. What is the type of the variable 'phonebook'?

79. What is the most suitable data type for a phone number?

80. What is the instruction to remove the element associated with the first item you added?

81. Using previous exercises as an example of how to use a `for` loop, what instruction would you type to construct a dictionary called 'squares' that contains the integers 1, 2, 3, 4 and 5 as keys associated with the value equal to the square of each of the key?

Dictionaries have a number of methods that can be invoked on them, the Python documentation for dictionaries can be found at

https://docs.python.org/3/tutorial/datastructures.html#dictionaries.

Boolean

The boolean type stores the value `True` or `False` and is used for logical expressions. Booleans are the result of logical operations and are used in loops and conditional statement. Let's see how to use a boolean and look at some of the operations. Using the interactive Python shell type in the following lines;

```
1  0 == 1
2  "a" == "a"
3  [1,2,3] >= [1,2,3]
4  [1,2,3] > [1,2,3]
5  (1,2,3) <= (1,2)
6  (1,2,3) <= [1,2,3]
7  len((1,2,3,4)) <= len([1,2,3])
8  (1,2) <= (1,1)
9  (1,2,3) < (4,1)
10 (1,1,1) <= (2)
```

examples/snippet1–8.py

82. What is the result of line 3 of snippet1-8?

83. What is the result of line 4 of snippet1-8?

84. Describe how Python compares tuples (You may wish to consult the Python online documentation)

85. What is the result of line 6 of snippet1-8 and suggest a cause for this result?

86. Describe what line 7 of snippet1-8 is doing

Python has a set of keywords which are used to form more complex boolean expressions. A keyword is a word that is reserved by Python, so it can not be used as a variable name as it has a special meaning. The keywords concerning boolean expressions are '*not*', '*and*' and '*or*' which as their names suggest compute the negation, conjunction and disjunction (respectively). The order of precedence of the logical operators is as with traditional boolean logic (however logical operators have lower precedence than all other operators in Python with the exception of *lambda* expressions).

87. What is the order of precedence of 'or','not' and 'and' in traditional logic?

Boolean expressions are used to make decisions in programs and control the flow of execution.

Extended Question: Supermarket

Consider a supermarket that sells alcohol, there are a number of restrictions on the sale of alcohol, the supermarket wishes to create some boolean expressions to make the decision if they can sell alcohol to a customer. If the boolean expression evaluates to `True` then the supermarket will sell alcohol to the customer otherwise the customer will be refused. Complete the following tasks using an interactive Python shell and answer the questions below;

- Create a variable called 'age' and assign it the value of your age.
- Create a variable called 'legalAge' and assign it the value of legal drinking age.
- Create a variable called 'paidByCard' and assign it the value `False`.
- Create a variable called 'challenge25' and assign it the value `True`.
- Create a variable called 'ageOnID' and assign it the value 16.

88. What boolean expression would result in the value True if and only if `age` is greater than or equal to `legalAge`?

89. What boolean expression would result in the sale of alcohol if and only if the customer is at least as old as the legal age or has paid by card?

90. Some supermarkets operate the scheme 'Challenge 25'. The Challenge 25 scheme forces the supermarket to check the ID of any customer attempting to buy alcohol who looks under 25 years of age. What boolean expression would result in the value `True` if and only if the following conditions are satisfied:

    - The supermarket operates Challenge 25 and the age on the ID is greater than or equal to the legal age;

      or,

    - the supermarket does not operate Challenge 25 and the age of the customer is greater than or equal to the legal age.

91. Assume the following boolean expression is being used, complete the table below:[2]

```
((␣(challenge25␣and␣not␣(paidByCard)␣)
␣␣␣or
␣␣␣(␣(not␣challenge25)
␣␣␣␣␣␣and
␣␣␣␣␣␣((facial_hair_length_cm==0)␣or␣(age_on_card<legalAge))
␣␣␣)
␣)
␣and
␣facial_hair_length_cm␣<␣7
)
```

legalAge	paidByCard	challenge25	ageOnID	facial hair len	Answer
18	True	True	16	0cm	
18	True	False	25	0.1cm	
18	True	False	25	0.3cm	
26	True	True	25	0cm	
26	False	True	25	6cm	
26	True	True	25	5cm	
18	False	True	25	17cm	
18	False	True	25	2cm	

[2] Note that, although in most situations Python requires code in any 'code block' to be formatted with a fixed left margin, position, this requirement is not enforced within *parentheses* (i.e. brackets, including square or curly brackets also). This is very useful for setting out complex Boolean conditions (such as the example shown here), list structures and dictionaries in a neat and readable way.

Unlike many stricter languages Python has a few strange behaviours where the boolean type is used. Any of the types we have mentioned above (Numbers, String, List, Tuples and Dictionaries) can be used where Python would expect to find a boolean type. Python will interpret the types as follows:

True	False
`True`	`False`
`1`	`0`
non-empty String	empty String or `None`
non-empty Dictionary	empty Dictionary
non-empty List or Tuple	empty List or Tuple

It is worth noting that evaluation of boolen operators in Python has a short-cut behaviour to avoid unnecessarily evaluation. An 'or' operation will only evaluate the second argument if the first one is `False`. This is because when evaluating an expression of the form `«test1» or «test2»`, as long as `«test1»` is `True` then the whole expression must be true, so there is no need to evaluate `«test2»`. Similarly with the 'and' operator, if the first argument is false then the value `False` is immediately returned otherwise the second argument is evaluated and its value returned. So, the 'and' operator will only evaluate the second argument if the first argument value is `True`.

92. *Warning: This question concerns some rather odd behaviour of Python. It is not recommended that you program using this kind of syntax. Generally, it is sensible to confine the use of logical operators to operate only on Boolean valued expressions. However, the question does illustrate the fact that the result of executing a code expression depends on the exact rules that are built into the language's interpreter or compiler. In some cases this may give results that are neither intuitive nor useful.*

— Using the interactive Python Shell type in the following

```
1 'a' == ('a' or 'b')
2 'b' == ('a' or 'b')
3 'a' == ('a' and 'b')
4 'b' == ('a' and 'b')
```

Using your knowledge about how Python boolean operators work explain the behaviour of the above snippet

93. Python uses a system called *duck-typing* to determine the type of an object. Research what duck-typing is, how it affects Python and what are the alternatives. Include a comparison of advantages and disadvantages of duck-typing to its alternatives.

94. Python is a strongly typed language. Research what a strongly typed language is, what are the alternatives?

Graphics in Python

There are a number of ways of producing graphics using Python, one method which has gained a large following is the library tkinter. The module tkinter allows you to create graphical user interfaces in Python. We will be using it to draw graphics to illustrate and practice things we have learnt in Python. Let's looks at a simple example that creates an empty window;

```
1 #Import␣tkinter
2 from␣tkinter␣import␣*
3
4 #Create␣a␣main␣window
5 master␣=␣Tk()
6 #Create␣a␣Canvas␣in␣the␣main␣window␣with␣a␣given␣size
7 w␣=␣Canvas(master,␣width=200,␣height=100)
```

examples/tk–snippet1–1.py

The tkinter module is part of the standard Python libraries. Line one imports the library so that you can use it. Line three creates a main window and assigns it to the variable 'master'. The variable 'master' can be thought of as a handle to the main window. Line five creates a Canvas *attached* to the main window, the Canvas allows you to draw on the main window. The Canvas is a special kind of type, which is not part of the basic types provided by Python but is imported from a module (you will cover this later on in the course). We call the Canvas an object and the object has a set of actions you can do to it. Just to illustrate the point of objects and actions, consider a feline (a cat). A feline can be modelled as an object and there are a number of actions you can do to it, such as feed it, stroke it, play with it and talk to it. This type of programming where objects are the primary concern is called object-oriented programming. Although you will not learn object-oriented programming until later in the module we will be using object without knowing it. The actions that can be performed on an object are called methods.

The methods that we will use on the Canvas object are;

- `create_line` - accepts a list of Cartesian points and draws lines between them. A line will be drawn between consecutive points in the list.
- `create_arc` - accepts two Cartesian points defining the top left and bottom right corners of the bounding rectangle of which the ellipse is inside. The optional arguments 'start' and 'extent' give the start and end position of the arc. Starting at 'start' extending anticlockwise to 'extent'.
- `create_text` - accepts a Cartesian point and a keyword argument 'text' and draws the text at the specified point.
- `create_oval` - accepts two Cartesian points defining the top left and bottom right corners of the bounding rectangle of which the oval is inside.
- `create_rectangle` accepts two Cartesian points defining the top left and bottom right corners of a rectangle
- `create_polygon` - accepts a list of Cartesian points which define the vertices of a polygon.

Let's see a basic example of using these methods on the Canvas object.

```
from tkinter import *

main = Tk()

# create a canvas object with a blue background
canvas = Canvas(main, bg="blue", height=250, width=300)

# create a red arc
coord = 10, 50, 240, 210
arc = canvas.create_arc(coord, start=0, extent=150, fill="red")

# create two line segments
canvas.create_line(0,0,100,100,150,290)

# create a yellow oval
canvas.create_oval(100,100,200,200, fill="yellow")
# create a green polygon
canvas.create_polygon(100,10,160,10,160,60,100,60, fill="green"
    )
# creates a text object at the specified position
canvas.create_text(200,90, text="Hello World")

canvas.pack()

main.mainloop()
```

examples/tk–snippet1–2.py

The pack method tells Python to update the windows and adjust the windows to ensure all of the objects fit. Remember to always call pack after drawing on a canvas or making alterations to an object that will affect its visual appearance.

- Create a file containing snippet 'tk-snippet1-2' and run the program. Try and relate the instructions in the file with what you see on the screen.

95. Using tkinter, write a program that will draw a picture on a canvas object. Your picture should include at least two of the following features:

    - a house,
    - a tree,
    - the sun,
    - a mountain.

Key Points:

▷ Idle can be started by typing into the terminal:

```
idle3 &
```

▷ The Python instruction to get input from the user is:

```
«string_variable» = input()
```

▷ The # symbol denotes a comment in python code.

▷ Python is case-sensitive

▷ Python is whitespace sensitive

▷ The Python operators (+, /, **,%) on numbers are Addition, Division, Exponentiation and Modulo.

▷ Variables in Python are untyped

▷ Number in Python are represented by the data types

- `int`
- `float`
- `complex`

▷ Sequences in Python are represented by the List and Tuple data types

▷ Dictionaries provide associative map and are implemented as hash tables

▷ The boolean data type in Python has the value `True` or `False`.

▷ Python interprets data type as boolean values according to;

True	False
`True`	`False`
`1`	`0`
Non-empty String	Empty String or 'None'
Non-empty Dictionary	Empty Dictionary
Non-empty List or Tuple	Empty List or Tuple

Notes

Reflection

What did you learn in this section?

Did you encounter any problems? How did you resolve them?

I need to revise

1.
2.
3.

....
Module Staff Signature Date

Week 2

Control structures

AIMS:

- to understand the syntax and structure of `for` and `while` loops
- to understand the syntax and structure of `if` statements
- to be able to use logical operators to construct control conditions

Loops

A loop allows a piece of code to be executed multiple times. Loops in Python come in two different varieties: `for` loops and `while` loops. We will cover both types of loops providing you with a variety of examples of different ways that loops can be used. These should also help you understand which type of loop is most appropriate for a given task. (In general it is always possible to use either type of loop, but one will often result in shorter and clearer code than the other.)

`for` loops

The general form of a Python `for` loop is:

```
for␣<<variable>>␣in␣<<iterable>>:
␣␣␣␣<<block>>
```

In which:

- «`variable`» is any valid variable name (we call this the *control variable*);
- «`block`» is one or more lines of code (a 'code block') each of which will be *indented* relative to the line in which the preceeding `for` keyword occurs (typically, though not always, the control variable occurs one or more times within the code block);
- «`iterable`» can be one of several types of code construct that refer to an object that can be used to generate a *sequence* of values.

The behaviour of a `for` loop is to successively take each value generated by «`iterable`», assign «`variable`» to have this value, then execute «`block`». Thus, «`block`» gets run as many

times as there are values generated by «iterable».[1] Each time, the result of executing «block» can be different, because the value of «variable» may be different.

Let us see a concrete example of how to use a `for` loop:

```
1 x = [1,3,8,2,5,6,33]
2 for item in x:
3     print(item)
4
5 for k in range(1,10):
6     print("Hello ", k)
```

examples/snippet2–1.py

On line 1 we construct a list containing seven integers. Lines 2-3 contains the `for` loop. The variable 'item' is the `for` loop's control variable. The loop block (in this case just the instruction `print(item)`) will be successively run with `item` assigned to have the value of each element of the list denoted by `x`. So, in this case it will be run seven times with `item` taking the value of each of the elements of `x`. Notice how line 3 is indented, this indicates that this code is within the *body* of the `for` loop. The `for` loop body consists of the subsequent lines that are at a higher level of indentation than the `for` instruction.

1. What will lines 1-3 print out in the interactive Python Shell?

An alternative method of using a `for` loop is to use the `range` function. Instead of in line 1, where we explicitly construct a list, the `range` function will generate a sequence of numbers, according to the parameters given to it. `range(start, stop)` will generate all numbers from `start` up to but *not* including `stop`.

2. Type in snippet2-1, paying close attention to the `range` instruction. How might you change it to count from 7 to 10?

The `range` instruction also allows you to provide a step size which allows you to alter the difference in the elements. The following snippet demonstrates this;

```
for x in range(10,-10,-2):
    print(x)
```

3. Write a program that prints out the numbers between 1 and 1000.

4. Write a program that prints out the numbers between 1 and 100 in multiples of 5.

5. Write a program that sums the numbers 1 to 10 and prints the result.

6. Write a program that uses `for` loop that prints the 6 times table in the following format.
   1 x 6 = 6
   2 x 6 = 12
   ⋮
   12 x 6 = 72

7. Write a program that asks the user to enter a number then prints the times table of the entered number.

8. Write a program that calculates and prints out the conversion between degrees Celsius and degrees Fahrenheit for all integer values between 0 and 100. The print out should have a new line for each entry and a header row at the top. The conversion formula is provided below.

$$\text{Fahrenheit} = \frac{9}{5} \times \text{Celsius} + 32$$

It is also possible to use list comprehension to construct a list for a `for` loop to iterate over. Using list comprehension complete the following exercises.

9. Print out the first 10 square numbers that are even

10. Write a program that calculates and prints the first 5 Pythagorean triples (http://en.wikipedia.org/wiki/Pythagorean_triple)

[1] Though, as we shall see later, there are ways to instruct Python to terminate the loop before using all values generated by the iterator, and there are also instructions that may result in skipping some lines of the block.

While loops

A *while* loop is another way of repeating code. A `while` loop continues to loop while a boolean expression evaluates to `True`. Indentation is used in the same way as in `for` loops. The general form of a `while` loop is;

```
while␣<<boolean_expression>>␣:
␣␣␣␣<<block>>
```

Let's see a concrete example of how to use a `while` loop.

```
1 x␣=␣10
2 while␣x␣>␣0␣:
3 ␣␣␣␣print(x)
4 ␣␣␣␣x␣=␣x-1
```

As soon as the boolean expression, in the example above $x > 0$, evaluates to `False` the loop will terminate and the interpreter will continue to execute the program.

11. Write a program that uses a `while` loop that prints the even numbers from 0 to 100.

12. Write a program that uses a `while` loop that prints the numbers from 0 to 100 that are multiples of 3.

13. Write a program that loops continuously asking the user to input their name and prints a salutation to them.

Breaking and continuing

Python provides a way of exiting a loop (either a `for` loop or a `while` loop) early. The *break* instruction will exit the loop.

Type the following snippet into an interactive Python shell (or into a program editing window):

```
1 x␣=␣10
2 while␣x␣>␣0␣:
3 ␣␣␣␣print(x)
4 ␣␣␣␣break
5 ␣␣␣␣x␣=␣x-1
6 ␣␣␣␣print(x)
```

14. What is printed by the snippet?

To begin the next iteration of the loop the instruction `continue` is used. This instruction causes the Python interpreter to continue to the next iteration without executing any of the remaining instructions of the loop.

Type the following snippet into an interactive shell or program window.

```
x = 10
while x > 0 :
    x = x-1
    continue
    print(x)
```

15. What is printed by the snippet?

Now that we have seen loops let's see what interesting things we can do with them. Two dimensional lists are common in programming for storing data such as matrices. The following snippet demonstrates how to construct a two dimensional list:

```
#Create a variable called a of type list
a = []


for i in range(5):
    #Append an empty list to a
    a.append([])
    for j in range(5):
        #Append 0 to the ith list of a
        a[i].append(0)

#Print a
print(a)
```

examples/snippet2–2.py

16. Using snippet2-2, write a program that will print the values of a two dimensional list. Each nested list should be printed on a separate row.

17. Write a program that constructs a two dimensional list containing 5 lists each containing 5 elements. The value of the ith element in the jth list should be $j \times 5 + i$. You may want to use the code you wrote in the previous question to print the values of the two dimensional list.

18. Adapt the program from the previous question so that it asks the user for two dimensions, then constructs a two dimensional list with the specified dimensions with the values as described in the previous questions (When printed it should look like a grid of numbers).

19. Write a program that asks the user to enter 10 numbers, then prints out the sum of the numbers and the mean average.

If statements

Conditional statements allow a section of code to be executed when a condition is met. In many programming languages conditional statements take the form of `if` statements. These take a boolean expression as an operand. The general form of an `if` statement is;

```
if <<boolean_expression>>:
    <<block>>
else:
    <<block>>
```

Let's see a concrete example of how to use a `if` statement.

```
1  print("*** Lucky Number Guesser ***")
2
3  nstring = input("Enter an integer: ")
4  n = int(nstring)
5
6  if (n == 7):
7      print("You guessed my lucky number.")
8      print("Give yourself a pat on the back.")
9
10 if not(n == 7):
11     print("Unlucky :( --- Better luck next time.")
```

examples/snippet2–3.py

If the user enters the number 7 the instructions on lines 7-8 are executed, if the user enters any that is not 7 then the instruction on line 11 is executed.

— Create a file and type the code from snippet2-3 and test that it runs correctly. Consider what values you should try to ensure that the code runs as expected.

— Add comments to snippet2-3 so that you understand what it does.

20. Modify snippet2-3 so that the lucky number is 42

21. Using snippet2-3 for inspiration write a program that allows the user to enter a number and prints an output indicating whether the number is odd or even.

22. Modify your program from the previous question so it loops indefinitely repeatedly classifying the input from the user as odd or even.

As with the grammatical constructs in the English language that use 'if' it is common to find an 'else' clause. In programming languages the `else` is a block of code that is executed if the `if` condition evaluates to `False`. The block of code for the `if` statement and the block of code in the `else` statement are mutually exclusively executed, that is only one of them is executed. We may rewrite snippet2-3 to use an `else` clause;

```
print("*** Lucky Number Guesser ***")

nstring = input("Enter an integer: ")
n = int(nstring)

if (n == 7):
    print("You guessed my lucky number.")
    print("Give yourself a pat on the back.")

else:
    print("Unlucky :( --- Better luck next time.")
```

examples/snippet2–4.py

FizzBuzz

The game FizzBuzz is a group game to teach children about division. Players take it in turn to count incrementally, replacing any number that is divisible by three with the word "Fizz" and any number divisible by five by the word "Buzz". If the number is divisible by both three and five the number should be replaced by the word "FizzBuzz".

23. Write a program that counts from 0 to 100 playing the game FizzBuzz.

24. Modify your program from the previous question so a user can play with the computer, taking alternate turns. The computer should not continue playing if the user makes a mistake.

    The following code generates a pseudo-random number between 0 and 10.

    ```
    import random
    print(random.randint(0,10))
    ```

25. Using the snippet above, modify your program so the computer makes a mistake with a 20% chance (the 'import random' instruction must be placed at the top of your file)

26. Modify your program to introduce a new rule that if the number is divisible by seven then the number should be replaced also by "Woof". If the number is divisible by three and seven then the number should be replaced with "FizzWoof". If the number is divisible by five and seven then the number should be replaced with "BuzzWoof" and If the number is divisible by three, five and seven then the number should be replaced with "FizzBuzzWoof".

Number Guessing Game

The number guessing games is a one player game where the player is required to guess a secret random number (between 0 and 100) chosen by the computer. The game will give the player a clue each time the player gets the number wrong, either printing "too high" or "too low". In the event of the player guessing the number correctly the game will congratulate the player.

27. Write the number guessing game.

28. Modify your game so the player only gets 5 guesses.

29. What is the minimum number of guesses that are required so that the player can always win (explain how you came to your answer)?

Extended Question - CityParcels

CityParcels is a parcel delivery company which provides a variety of delivery options. The company wishes to have a program that will correctly classify the parcel type and cost. The parcel carrier provides the following parcel types.

Type	Max weight	Max length	Max width	Max height
Letter	0.1kg	240mm	165mm	5mm
Large letter	0.75kg	353mm	250mm	25mm
Small parcel	2kg	450mm	350mm	80mm
Small parcel	2kg	350mm	250mm	160mm
Medium parcel	20kg	610mm	460mm	460mm
Large parcel				

Table 2.1: Dimensions of parcel types.

A large parcel is any parcel which does not fit into any of the smaller parcel types. A parcel cannot exceed any of the maximum dimension however a parcel may weigh less than the weight specified in the table.

30. Write a program that asks the user to enter the dimensions (including the weight) of the parcel and outputs the smallest (cheapest) parcel type that includes parcels of that size.

The cost of sending a parcel is shown in the Table 2.2, for every 0.1kg the parcel is over the basic weight an additional fee is charged (also shown in the table).

Type	Price	Basic weight	Price per 0.1kg
Letter	£1.72	0.1kg	
Large letter	£2.03	0.1kg	£0.10
Small parcel	£4.30	1kg	£0.40
Medium parcel	£6.75	10kg	£0.90
Large parcel	£15	10kg	£1.56

Table 2.2: Costs of parcel types.

31. Modify your program from the previous exercise to calculate the cost of sending the customer's parcel.

32. Modify your program so that the user is prompted if they want to classify another parcel, if they do your program should loop otherwise your program should print out the total cost of all the parcels classified and exit gracefully.

33. Modify your program so that it always classifies a parcel correctly independent of the order that the dimensions are provided.

More graphics in Python

It is possible to use the programming constructions from this section in tkinter to create more complex and interesting graphics. The loops can be used to create repetitive patterns. If you wish to create more colourful graphics tkinter provides a set of predefined colours such as `green`, `yellow` and `red`. To be able to chose any colour you wish you can also specify the colour as an *RGB* value. *RGB* stands for red, green and blue value, the numbers are usually expressed as hexadecimal values between 0-255 and are prefixed with a '#' symbol, for example red has the RGB value ''`#FF0000`''. Building on the simple graphics program you used last week try the following exercises.

34. Write a program that creates a chessboard effect.

35. Write a program that draws a grid showing the numbers 1 to 100 (in 10 rows of 10). To help players play FizzBuzzWoof, the squares of the grid should be coloured according to the game rules. Choose different colours to indicate what the player should say.

Key Points:

▷ Loops allow blocks of code to be repeated.

▷ The general form of a `for` loop is:

```
for <<variable>> in <<iterator>>:
    <<block>>
```

▷ The general form of a `while` loop is:

```
while <<boolean_expression>>:
      <<block>>
```

▷ `if` statements allow blocks of code to be executed depending on a condition.

▷ The general form of a `if` statement is:

```
if <<boolean_expression>>:
   <<block>>
else:
   <<block>>
```

Notes

Notes

Reflection

What did you learn in this section?

Did you encounter any problems? How did you resolve them?

I need to revise

1.
2.
3.

....
Module Staff Signature Date

Week 3

Functions[1]

AIMS:

- to know what a function is in Python
- to be able to define a function in Python
- to be able to define a recursive function in Python

A function is a block of code that has a name and can be executed by *calling* it. We have come across functions in previous weeks, for example `type`, `len` and `input`. A function may be passed some some *arguments* (values that control its behaviour), and may also return a value. Arguments are passed to a function by including them in round brackets. The order of the arguments matters. The return value of a function is a value that is returned from the function when the function stops executing and the execution is passed back to the main program.

Functions play an important role in programming and improve the readability and maintainability of code. The general form for a function definition is:

```
def␣<<function_name>>(<<paramter_list>>):
␣␣␣␣<<block>>
```

The body of a function is indented. This indentation denotes what code will be executed when the function is called. Any variables that are defined in the function or occur in the parameter list are called *local* variables. The *scope* of a local variable is restricted to the indented section and any nested indented sections. This means that the variable cannot be accessed outside of the function. Nevertheless, the same variable name can be used in different functions or outside the function. In this case Python treats it as a different variable, although it has the same name. (This enables functions to be modular, in that, when naming variables in one function, we do not need to worry about what names are used in other functions.)

[1] In this workbook, the word 'function' will be used to cover both functions that return values and functions that do not, which some people (including BB) prefer to call 'procedures'. In Python documentation the word function tends to be used for both cases. It is also worth noting that even if a 'function' does not contain a `return` command, it will still return the value `None`; so, on that basis, it could be argued that it is a function rather than a procedure.

Let's see a concrete example of how to define a `function`.

```
1 def␣sayHello():
2 ␣␣␣␣print("Hello")
```

examples/snippet3–1.py

When the function is called it will print "Hello".

A function is 'called' by giving its name, followed by round brackets, which may contain argument values that will be passed to the function. In this case, no parameters, so to call the function we would simply use the following code:

```
1 sayHello()
```

— Type snippet3-1 into the interactive Python shell and call the function `sayHello()`.

The rules for naming functions in Python are the same as for naming variables.
We may define a function that 'accepts' arguments as follows;

```
1 def␣add(arg1,␣arg2):
2 ␣␣␣␣print(arg1␣+␣arg2)
3
4 add(3,4)
5 add([1,2,3],[4,5])
```

examples/snippet3–2.py

1. What is the output from line 4?

2. What is the output from line 5?

3. What is the return type of line 4? (Recall from previous weeks, we may use the function `type(«xyz»)` to get the type of a value.)

To return a value from a function the keyword `return` is used. The operand after the keyword `return` is returned to the point in the code where the function was called. The effect is that function call instruction takes the value returned by the function.

It is possible to return any kind of data from a function: lists, tuples, integers, dictionaries and even functions themselves (although this is far more advanced than we will cover here). The following function is similar to the function in snippet3-2; however, instead of *printing* the sum of the two operands, this function *returns* the sum.

```
1 def␣add(arg1,␣arg2):
2 ␣␣␣␣return␣arg1␣+␣arg2
3
4 add(3,4)
5 print(add(3,4))
```

examples/snippet3–3.py

4. How many arguments does the function in snippet3-3 accept?

5. What is the output from line 4?

6. what is the output from line 5?

7. What is the return type of line 4?

8. Briefly explain the difference between the functions in snippet3-2 and snippet3-3 mentioning what each function does and the return type.

The use of functions is essential for producing readable and maintainable code, for all but the simplest tasks. Once you get the hang of them, they are also a great help in problem analysis and program design and implementation. This is because they enable you to tackle a problem in terms of smaller sub-problems, which then get combined together to produce a solution to the problem as a whole.

For the following questions you should write a function that meets the specification and ensure your function behaves as expect by calling the function of a few sensible inputs. You should write these functions in a file (i.e. use an Idle file window) and in the same file you should include additional lines of code that call the function with some suitable parameter values and print out the results obtained. You may put each function in a separate file or put several functions together in one file. For the questions that say 'Write a program ...', you will need to have one or more functions plus some additional code in the same file to achieve the required effect.

9. Write a function called 'minus' that accepts two arguments and subtracts the first argument from the second printing out the result.

10. Write a function called 'sayHelloTo' that accepts an argument and prints the salutation "Hello <*argument*>".

11. Write a function that calculates and *returns* the area of a rectangle from its height and width. (Note: the function must *return* rather than print the area, so the function definition must include a `return` statement.)

12. Write a *program* that asks the user to enter the width and height of a rectangle and then uses the function that you defined in answer to Question 11

13. Write a function called 'CeliusToFahrenheit' that accepts an argument and converts the value into Fahrenheit. (The conversion formula can be found in the Week 2 section.)

14. Write a function called 'FahrenheitToCelius' that calculates the opposite of the above question.

15. Write a function that accepts a list of integers and returns the mean (average) of the list.

16. Write a function that accepts a list of integers and returns the median of the list. (See `http://en.wikipedia.org/wiki/Median`.)

17. Write a program that asks the user to enter a sequence of numbers, terminating by entering the string `"end"`. (We suggest that you collect these numbers by appending them to a list.) Once the numbers have been entered the program should calculate the mean and median of the numbers using the functions that you defined in answering Questions 15 and 16. The program should then print out the mean and median values and also print out which is the highest and what is the difference between the two values.

Recursion

Recursion is an important part of programming and is available in most (not all) programming languages. In computer science recursion is a method where the solution to a problem is computed from solutions of smaller instances of the same problem. Recursion allows us to easily express some programs which would be really hard to write without recursion. An interesting note is that there exists programming languages which only have recursion, they don't have 'for' or 'while' loops. It has been shown that if a program uses recursion it is also possible to express the program without the use of recursion. In other words recursion is not necessary but it is a very useful tool and often makes it easy to write a program.

Let's look at an application where recursion is used. Consider adding the elements of a list together, this could easily be achieved using a `for` loop;

```
def sumList(a):
    sum = 0
    for i in a:
        sum = sum + i
    return sum
```

The function `sumList`, sums together the elements of the list and returns the value. This could alternatively be expressed using recursion. In recursion the function will call itself and use the return value to compute the value. When you write a recursive function there are three golden rules to remember;

- A recursive function must have a **base case**
- A recursive function must change the arguments and move towards the base case
- A recursive function must call itself

Let's look at how we may implement the above example using recursion.

```
def sumListRec(a):
    if len(a) == 0:
    #base case, the smallest instance of the problem
        return 0
    else:
        #how to combine subproblems together to the the bigger
            solution
        return a[0] + sumListRec(a[1:])
```

— Type in the above examples and test that they do what you expect them to. You may wish to add print instructions so that you can see what values are being passed to the function.

There are many well known problems that are easily solved using recursion. Complete the following exercises using recursion. You should not use any Python libraries.

18. Copy the following snippet into a Python editing window. Run it, and test it for some different values of x and y. Describe what it does.

```
def func(x,y):
    if y==0:
        return 0
    else:
        return x + func(x,y-1)
```

19. What is the base case in the snippet above?

20. Write a function that uses recursion to compute the factorial of a number. The factorial of a positive number n is the product of all the numbers between 1 and n. In terms of equations, the factorial of n can be expressed as follows:[2]

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 1 \\ n \cdot \text{fact}(n-1) & \text{if } n > 1 \end{cases}$$

21. Write a function that uses recursion to compute the nth Fibonacci number. The nth Fibonacci number can be computed as follows:

$$\text{fibonacci}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & \text{if } n > 0 \end{cases}$$

[2] For reasons we won't go into here, mathematicians also define factorial(0) = 1.

22. Write a function that uses recursion to compute the number of ways of selecting k items from a bag of n item (this function is often denoted in mathematics as $\binom{n}{k}$).The formulae for calculating $\binom{n}{k}$ is

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \text{ where } 0 < k < n$$

and

$$\binom{n}{0} = \binom{n}{n} = 1$$

23. How are arguments passed to functions in Python? What other ways are there?

Extended Question: ROT13

ROT13 is a simple encryption system that was used in the first century BC. The encryption is a subsitution based technique. To encrypt a string each character of the string is considered individually and is substitued with its respective character as per the table below.

a	b	c	d	e	f	g	h	i	j	k	l	m
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
n	o	p	q	r	s	t	u	v	w	x	y	z

24. Write a function called `substituteCharacter` that takes an alphabetical character (i.e. a one letter string) as an argument and returns the corresponding encoded letter according to the substitution table above. (One approach is to represent the substitution table using a dictionary.) You may assume you only need to deal with lower case letters (or handle upper case as well if you wish).

25. Write a function called `encryptROT13` that accepts a string and uses `substitueCharacter` to encrypt the string. The encrypted string should be returned by the function (spaces are substitued with a space).

26. Write a function called `decryptROT13` that accepts an encrypted string and returns the decrypted string

Extended Question: Beetle Drive

The Beetle game is a simple luck-based game where you aim to complete a beetle. A beetle consists of: a body, a head, six legs, two eyes, two antennae and a tail. The player rolls a dice, each number on the dice represents a body part of the beetle. When the play has all the body parts for a beetle the player wins and the game is over. The number on the dice represent the following beetle body parts:

1. Head
2. Eye
3. Antenna
4. Tail
5. Led
6. Body

It is necessary to roll the correct number for the body before any other part can be had. To the body, you must attach a head before you can have eyes and the antennae. A full description of the game can be found at http://en.wikipedia.org/wiki/Beetle_(game).

The game has been partially implement. You will find the implementation in the file `BeetleDriveTask.py`. Which you can download from the module VLE page, in the Semester 1 directory of the Labs area.

7. Read the comments at the top of each function and implement the functionality.

Animation with Python

Using the Canvas object it is possible to create animation. The Canvas object has a method that lets you move something that has been drawn on the canvas and let you update the canvas so that the changes can be seen. The following example shows how these methods can be used;

```
from tkinter import *

width=500
height=500
x_velocity = 5

# function that changes the position of the ball
def animate(canvas):
    canvas.after(50) # delay 50ms before next canvas command
    canvas.move('ball', x_velocity, 0)
    canvas.update()

# Initialises the main window
main = Tk()
canvas = Canvas(main, width = width, height = height)
canvas.create_oval(20,220,40,240,fill="red", tag='ball')
canvas.pack()

#Infinitely loops calling the animate function
while True:
    animate(canvas)
```

examples/animate–move.py

The `create_oval` method on the canvas object can be given a 'tag' which allows you to refer to it later, this tag is used by the `move` method. The `move` method on the canvas object accepts three arguments the tag of the drawn object you wish to move, a change in x value and a change in y value. The `update` method redraws the canvas so the ball is drawn in its new position.

— Create a file and type in `animate-move.py`. Try changing the numbers and see what happens.

8. Write a program that makes the ball move from the top left hand corner diagonally to the bottom right hand corner.

9. Write a program that makes the ball move around the circumference of a circle.

10. Write a program that gives the appearance of the ball bouncing off the sides of the window.

    Hint: how should the ball's x coordinate change as it collides with the left and right sides of the window? How should the ball's y coordinate change as it collides with the top and bottom of the window?

Key Points:

- ▷ Functions allow a block of code with a specific purpose to be extracted, allowing the code to be called elsewhere in the program.
- ▷ The general form for a function definition is:

```
def <<function_name>>(<<parameter_list>>):
    <<block>>
```

- ▷ The scope of a variable in a function is restricted to within that function
- ▷ Functions may accept values, these values are called arguments.
- ▷ Functions may return values, the keyword `return` is used to return a value and return execution to the calling function.
- ▷ A function that calls itself is called a recursive function.
- ▷ A recursive function must:
  - A recursive function must have a **base case**.
  - A recursive function must change the arguments and move towards the base case.
  - A recursive function must call itself.

Notes

Reflection

What did you learn in this section?

Did you encounter any problems? How did you resolve them?

I need to revise

1.
2.
3.

....
Module Staff Signature Date

Week 4

File I/O and Exception Handling

AIMS:

- to be able to read from and write to files in Python
- to be able to *parse* data from a file to use in Python
- to be aware of *exceptions* in Python
- to be able to handle exceptions appropriately in Python

Working with files is an important skill to learn. Often you will be required to process data from a file and write the results out to another file. To read and write files, we must create a file object. This can be thought of as getting the file from a filing cabinet and placing it on your desk to start working on it. The file object has methods which allow you to manipulate the file and its contents. To create a file object, we use a Python function called `open`. Let's have a look how this is done:

```
fileObject = open(<<filename>>,<<mode>>)
```

The `filename` argument allows you to specify the name and the path to the file you wish to manipulate. The `mode` argument is an optional argument (you may omit it) which specifies whether the file will be used to read, write or append data. There are a number of modes you may specify when you open a file. These are summarised in Table 4.1.

Character	Meaning
`r`	open for reading (default)
`w`	open for writing, truncating the file first
`a`	open for writing, appending to the end of the file if it exists
`b`	binary mode
`t`	text mode (default)
`+`	open a disk file for updating (reading and writing)
`U`	universal newline mode (backwards compatibility only)

Table 4.1: Different modes to open files in Python

Writing to files

Let's see how to write to a file.

```
1 # Creates a new file object
2 fileObject = open("my_newfile", "w")
3 # Writes a string to the file
4 fileObject.write("This string will be written to the file.\n")
5 # Closes the file
6 fileObject.close()
```

On line two we create a file object for the file called '`my_newfile`' in the directory where Python is running from. The file is opened in 'w' mode meaning that we can write to it. The `write` method on line four allows you to write a string to the file. The `\n` at the end of line four tells the `write` method to write a new line character. The `close` method on a file object ensures that all the content that is to be written to storage is actually written. The operating system optimises disk writing so not every write instruction gets written to storage immediately. The `close` command ensures that everything is written to the storage and the resources associated with the file are released. (The `flush` method of a file object ensures that everything has been written to storage without releasing the resources associated with the file object.)

There are a number of special character sequences in Python that allow you to represent characters such as tab and new line. These *escape sequences* all begin with '\' (the escape character) followed by another symbol. Table 4.2 shows the set of escape sequences in Python:

Character	Meaning
\newline	Ignores the newline (used for program code layout)
\\	Backslash (\) This is for when you want the actual \ symbol.
\'	Single quote (')
\"	Double quote (")
\b	Backspace
\f	Formfeed
\n	Linefeed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Table 4.2: Non-printable characters

— Type in the code from the above snippet and check that it works correctly by locating and opening the newly created file and checking its contents.

1. Experiment with what happens if you open a file in mode ‘w’ that already contains text and write a string to it? What happens if you open a file in mode ‘a’ and write a string to it?

2. Write a program that creates a file called ‘squares’ and writes to the file the first 100 square numbers, one on each line. Each line should consist of the number followed by a comma, followed by its square.

An alternative way of writing to a file which ensures that the file resources are released is to use the `with` notation. The general form of this notation is:

```
with␣open(<<filename>>,␣<<mode>>)␣as␣<<variable>>:
␣␣␣␣<<block>>
```

The `with` notation ensures that when execution leaves the `with` block Python will automatically release the resources associated with the file object. This means you do not need to use the `close` method when you have finished using the file (and thus prevents the file being accidentally left open). To illustrate this, the following snippet produces the same result as above.

```
with␣open(‘my_newfile’,␣’w’)␣as␣fileObject:
␣␣␣␣#Writes␣the␣string␣to␣the␣file
␣␣␣␣fileObject.write(’This␣string␣will␣be␣written␣to␣the␣file
    .\\n’)
```

3. Modify the code you wrote for Question 2 to use the `with` notation.

Reading files

Let's see how to read from a file.

```
# Creates a new file object
fileObject = open('my_newfile', 'r')
# Loop indefinitely reading a line from the file
while True:
    # read a line from the file
    line = fileObject.readline()
    # if the line is empty break the while loop
    if line == "":
        break
    else:
        # print the line
        print(line)
fileObject.close()
```

The `readline` method on the file object will read one line from the file at a time. In Python a line is a sequence of characters ending with the 'newline' character.

4. What will happen when Python encounters the end of the file?

— Create a file called 'grades' and enter a set of numbers one per line.

5. Write a program that opens the file 'grades', reads in and adds up all the grade values in the file and prints the sum of all the grades (you will have to convert the string into an integer, look back in your notes to find out how to do this)

6. Write a program that opens the file 'grades' and counts the number of lines in the file

7. Write a *function* called 'sumGrades' that opens the file called 'grades' and sums all the grades in the file.

8. Write a *function* called 'numberOfGrades' that opens the file called 'grades' and counts the number of grades in the file.

9. Using the functions from the previous question write a *function* that calculates the mean (average) of the values in the file 'grades'.

10. How might we modify the above functions to make them more useful? (For example, consider a case where we want to access grades stored in various different files.)

The `with` notation works for reading files as well. The following snippet illustrates the use of the `with` notation for reading.

```
1 with␣open('my_newfile','r')␣as␣fileObject:
2 ␣␣␣␣while␣True:
3 ␣␣␣␣␣␣␣␣line␣=␣fileObject.readline()
4 ␣␣␣␣␣␣␣␣if␣line␣==␣"":
5 ␣␣␣␣␣␣␣␣␣␣␣␣break
6 ␣␣␣␣␣␣␣␣else:
7 ␣␣␣␣␣␣␣␣␣␣␣␣print(line)
```

In some situations you may wish to read the entire file at once, the method `readlines` does just that. The method returns a list, each element of the list is a line in the file.

Comma separated values files

Comma separated value files (CSV) are common in practice for transferring data between systems. Python provides a module for working with CSV files, the module provides a number of useful functions but we are only interested in reading and writing CSV files. The CVS reader provided by the module returns a list for each line in the file it is reading, the elements of the list are the values that were delimited by the commas in the file (you may specify an alternative delimiting character when you construct the reader). The CSV writer provided in the module writes lists of values to a file, applying the correct formatting.

Reading CSV files

Let's look at a minimal example of how to read a CSV file:

```
1 import␣csv
2 with␣open(<<filename>>)␣as␣f:
3 ␣␣␣␣reader␣=␣csv.reader(f)
4 ␣␣␣␣for␣row␣in␣reader:
5 ␣␣␣␣␣␣␣␣print(row)
```

examples/snippet4–0.py

The file is opened using the `with` notation. On line 3 a new CSV reader is created and the file we wish to read is passed to it. Then we iterate over the lines in the file.

— Create a new file and type in the above snippet. Create a CSV file (or you could use the `novels.csv` file on the VLE in the Labs area). Modify the code so that it reads your CSV file.

11. Provide a line by line description of snippet4-0.

Writing CSV files

Let's look at a minimal example of how to write to a CSV file:

```
import csv
montypython = [['Title', 'Release Date'],
['And Now For Something Completely Different', '1971'],
['Monty Python And The Holy Grail', '1975'],
["Monty Python's Life Of Brian", '1979'],
['Monty Python Live At The Hollywood Bowl', '1982'],
["Monty Python's The Meaning Of Life", '1983']]

with open('test.csv','w') as file:
    writer = csv.writer(file)
    for film in montypython:
        writer.writerow(film)
```

examples/snippet4–1.py

The `writerow` method accepts a list and writes a comma separated value entry to the file.

— Type snippet4-1 into a file and test that it does what you expect.

12. Provide a line by line description of snippet4-1.

Error Handling

There are three types of errors that you are likely to encounter when programming in Python.

- Logic errors — These are errors that occur from badly implemented code, where the code does not behave as expected but does not cause either of the following two types of errors.
- Syntactic errors — These are errors that occur because the Python interpreter was unable to interpret the code because it is not valid Python code.
- Runtime errors — These are errors that occur due to some event that happened during execution of code.

During the course of completing the exercises you have probably encountered all of these types of error.

13. In the following example state which (if any) type of error will occur. (Assume that all variable have been given values of an appropriate type.)

```
#Multiply x by y and store the value in z
z = x ** y
```

```
Print(numberOfCows)
```

```
l = [1,2,3]
print(l[4])
```

```
d = {'a':3 , 'b':4 , 'c':5}
d['d']
```

```
f = 42/0
```

One way to handle errors is to avoid them to start with. This can be done by providing explicit checks, in the following example the function `div` accepts two arguments and prints the result of dividing the first argument by the second argument. Before the division takes place the function checks if the second arguments is not 0. In the following snippet of code it is not possible to get a divide by 0 error.

```
def␣div(arg0,␣arg1):
␣␣␣␣if␣not␣(arg1␣==␣0):
␣␣␣␣␣␣␣␣print(arg0␣/arg1)
```

Sometimes it is not possible to avoid errors, these types of errors occur when an abnormal condition happens such as a file being unavailable or when the program receives an input it did not expect. These types of errors are handled using the `try/except` notation. The general form of the `try/except` notation is:

```
try:
␣␣␣␣<<block_that_might_cause_an_error>>
catch␣<<exception_type>>:
␣␣␣␣<<block_to_handle_an_error>>
finally:
␣␣␣␣<<block_that_is_always_executed>>
```

If an error occurs in the code in the `try` block and the error matches that of the exception type of the `except` statement then Python will execute the code in the `except` block. In all situations the code in the `finally` block is executed. The `finally` block is normally used to release any resources that were being used in the `try` block.

Let's see a concrete example of using `try/except` notation.

```
(x,y)␣=␣(5,0)
try:
␣␣␣␣z␣=␣x/y
except␣ZeroDivisionError:
␣␣␣␣print("divide␣by␣zero")
finally␣:
␣␣␣␣print("All␣done")
```

examples/snippet4–2.py

14. What are the values of the variables x and y after line 1 is executed in snippet4-2?

15. Describe what will happen when snippet4-1 is executed. State what causes the exception and how Python handles the exception including what the program will print.

16. Using the Python online documentation find out what errors might be raised while trying to open a file. What are the possible errors that may occur?

17. Modify your code from Question 5 of this section to add error handling for the file reading operations.

— Download the files 'ClassA.txt' and 'ClassB.txt' from the VLE Labs page Use these file to test your answers to the following questions.

18. Write a function that accepts a file name and returns a list. Each line of the file should be read and added to the list as an element.

19. Write a function that accepts two lists and returns the intersection of the lists (i.e. a list of those elements that are in both of the lists).

20. Write a program which incorporates the functions you wrote in answer to questions 18 and 19, and by utilising these functions prints out all lines that occur in both of the files `ClassA.txt` and `ClassB.txt`.

Extended Question: Payroll

A company has a payroll system which is used to calculate the weekly pay of its employees. The payroll system works off two files one called `RateOfPay.csv` and one called `Timesheet.csv`. The file `RateOfPay.csv` is a CSV file containing the employee number, employee name and their rate of pay — one employee per line. The `Timesheet.csv` file contains the employee number and the number of hours they have worked that week. Sample files are provided on the VLE (on the Labs page).

The company wishes to automate the payroll system. The system will calculate and print out the amount each employee has earned.

21. Write a function called `getRateOfPay` that reads the content of `RateOfPay.csv` into a dictionary, using the employee number as the key and a tuple containing employee name and the rate of pay as the value.

22. Write a program that uses `getRateOfPay` and calculates the weekly pay for each employee, you should read the number of hours from `Timesheet.csv`. The results should be printed in a similar format to:

Employee Number	Name	Hourly rate	Number of hours	Total earning
00001	Joe Bloggs	7.89	10	78.9
00002	Fred Daniels	4.00	40	160

23. Modify your code so that the employees pay is increased by 50% for any hours worked over 40 hours.

24. Extend your program to create a file containing those employees that are in 'Timesheet.csv' but are not in 'RateOfPay.csv'(Hint: to test that your code works, you will need to add one or more extra entries to 'Timesheet.csv')

Key Points:

▷ The `open` function creates a file object.

▷ The mode parameter of the `open` function specifies the purpose for opening the file, the options are summarised below:

Character	Meaning
`r`	open for reading (default)
`w`	open for writing, truncating the file first
`a`	open for writing, appending to the end of the file if it exists
`b`	binary mode
`t`	text mode (default)
`+`	open a disk file for updating (reading and writing)
`U`	universal newline mode (backwards compatibility only)

▷ The `write` method on a file object will write a string to the file.

▷ The `readline` method on a file object will read a line (up to a new line character) and return a string.

▷ The `with` notation ensures that resources are released after the resource is no longer required.

▷ The general notation for `with` is:

```
with open(<<filename>>, <<mode>>) as <<variable>>:
     <<block>>
```

▷ Error handling allow you to manage what happens when an unpredicted situation occurs.

▷ The `try/except` notation allows errors to be handled.

▷ The general `try/except` notation is:

```
try:
   <<block_that_might_cause_an_error>>
except <<exception_type_A>>:
   <<block_to_handle_an_error_of_type_A>>
          :  :
except <<exception_type_Z>>:
   <<block_to_handle_an_error_of_type_Z>>
else:
   <<block_to_handle_if_there_is_no_error>>
finally:
   <<block_that_is_always_executed>>
```

Notes

Notes

Reflection

What did you learn in this section?

Did you encounter any problems? How did you resolve them?

I need to revise

1.

2.

3.

....
Module Staff Signature Date

Week 5

Program Decomposition and Testing

AIMS:

- to practise decomposing a program into functions
- to be able to design a test plan
- to be able to document code using `docstring`

As we mentioned in previous weeks, functions allow programs to be broken down into small blocks of code. Breaking a program down in this way is called program decomposition and helps in the maintainability, reusability and readability of code. Although the skill of program decomposition can only be gained through experience, there are a few general rules which will help:

- **Don't Repeat Yourself** (also known as the DRY principle) — if you have a block of code that is repeated several time in a program it is a likely candidate to be extracted into a function.
- **Single Responsibility Principle** — a function should have a single responsibility. If a function is doing more than one thing it should probably be decomposed into smaller functions.
- **Reduction of complexity** — the decomposition of a program should reduce the conceptual complexity of a program.
- **Global state is bad** — where possible avoid using global variables. Instead pass the required information via function arguments.

The concepts of program decomposition and modular design is a detailed area of study and passes by far the time requirements we have here to do it justice. Topics in modular design are covered in more detail in book such as Refactoring: Improving the design of existing code (Martin Fowler) and Design patterns : elements of reusable object-oriented software (Gamma et al).

Docstrings

When a program has been decomposed it is essential that you provide good documentation for your functions. In previous weeks we have used the # symbol to create comments in code. A further way of providing documentation is to use *docstrings*. A docstring is a string that occurs as the first statement in a module, function, method or class definition. Docstrings differ from comments in that comments normally describe how your code works whereas docstrings describe what your code does. A docstring is defined using triple quote marks, as shown below.

```
def add(arg0, arg1):
    """
    returns the sum of two objects
    """
    return arg0 + arg1
```

The docstring of an object can be accessed by calling the `help` function from the interactive Python shell. Assuming that the above definition has previously been executed by Python, the docstring along with some other information about the function can be displayed in the Python shell by using Python's `help` function as follows:

```
>>> help(add)
Help on function add in module __main__:

add(arg0, arg1)
    returns the sum of two objects
```

Testing

Testing is the process of ensuring that a program meets its specification. To successfully test a program it is required that the specification is known and is well defined. Testing can be as simple as executing your code with a set of test cases and printing the result or can done using a formal software development process (e.g. Unit Testing). Having a set of tests is not good enough to say that your program is working to specification, the test cases should cover all eventualities. This concept is called *test coverage*. To ensure that your tests have good coverage you should consider the following aspects:

- *Execution Path Coverage* — where a function has conditional statements, there will be various different routes that the execution can take through the code according to whether certain tests come out true or false. To ensure correctness, all possible routes should be tested. Similar considerations apply to loop constructs, where it is advisable to test cases where the loop block is executed once or several times. If it is possible that the loop is not run at all (e.g. `while` loop where the condition is initially `False` or `for` loop over an empty list). Keeping functions concise and clear will make this kind of testing much easier than it is with logically complex, highly nested functions.
- *Parameter Value Coverage* — Where a function has parameters, you need to be sure that the program will behave correctly and return the right values for any possible combination of input parameters. Normally it will be infeasible to explicitly test every possible combination, so the function should be tested for a range of *typical* parameter values and also for any *special case* values that need to be handled (e.g. a function that can process arbitrary length lists will need to behave sensibly when operating on the *empty* list).
- *External Dependencies* — If code behaves differently depending on the state of the program, the relevant states should be identified and your tests should test the behaviour of your code in each state. (It is generally advisable to avoid external dependencies — e.g. by not using global variables — so that this kind of testing is not needed.)

Before writing code you should carefully consider the purpose of the code and write a specification, from the specification you can define a set of tests then write the code. This style of development is called Test driven development (TDD). Let's have a look at an example.

Threshold function specification

We require a function that will return `True` or `False` depending if two floats are within a certain distance of each other. The function should return `True` if the two floats are within a distance and `False` otherwise. The third argument should be non-negative.

Threshold function test plan

Unit	Test	Precondition	Expected	Actual	Passed
`threshold`	`arg0` is not a float	none	ValueError		
`threshold`	`arg1` is not a float	none	ValueError		
`threshold`	`arg2` is not a float	none	ValueError		
function `threshold`	All parameters are floats	none	Output `True/False`		
`threshold`	`arg2` is negative	none	ValueError		
`threshold`	`arg2` is not negative	none	Output `True/False`		
`threshold`	The two numbers within tolerance (`arg0` = 4.3, `arg1` = 4.5, `arg2` = 0.3) (`arg0` = 0, `arg1` = 0, `arg2` = 0.1)	none	Output `True`		
`threshold`	The two numbers not within tolerance (`arg0` = 3.0, `arg1` = 4.0, `arg2` = 0.5) (`arg0` = 4.3, `arg1` = 4.4, `arg2` = 0.05)	none	Output `False`		
`threshold`	The two numbers are identical (`arg0` = 0.0, `arg1` = 0.0, `arg2` = 0.0) (`arg0` = 1.0, `arg1` = 1.0, `arg2` = 1.0) (`arg0` = -1, `arg1` = -1, `arg2` = 0.1)	none	Output `True`		

The snippet below defines a function that meets the specification above.

```
def threshold(arg0, arg1, arg2):
    """
    (float, float,float) -> boolean

    Determines if arg0 is close to arg1 (within a distance of
    arg2)

    Retuns True if |arg0- arg1| < arg2
    and False otherwise
    """
    if not (isinstance(arg0, (float,int)) and isinstance(arg1,
    (float,int)) and isinstance(arg2, (float,int)) ):
        raise ValueError('parameters are not all floats')

    dist = abs(arg0-arg1)
    if dist <= arg2:
        return True
    else :
        return False
```

1. Create a file and type in the `threshold` function. Using the test plan in the table above see if the function passes the tests. If not make modifications so that it does.

2. Using the same process as in the example above, design, implement and test a function that meets the following specification. Use the table on the following page to write your test plan (continue on to a separate page if needed).

   We require a function that given a list of integers, returns a tuple containing two elements. The first element of the tuple should be the minimum value in the list and the second element should be the maximum value in the list. The function should not accept a list with more than 7 elements.

Unit	Test	Precondition	Expected	Actual	Passed

3. You are given a function called `getIntersection` which takes as parameters two tuples. The tuples represent the equation of a straight line. A straight line can be represented mathematically as.

$$y = mx + c$$

The tuples are equivalent to `(m,c)`. If the lines do not intersect the function should return `None`, else the function should return a tuple that represents the intersection point. If the lines are coincident (they are the same line) then the function should return the first parameter. What test cases would you write to ensure the function behaves as expected (indicate if the test is a Size, Polychotomy, Boundary or Order test)? There is no programming required for this question.

Vending machine

4. Implement a vending machine change dispenser. Your program should;

   - Prompt the user for the amount of change required
   - Print the quantity of each coin the machine will dispense or notify the user if the machine can't service the request
   - The vending machine should dispense the minimum number of coins possible
   - Keep a track of the current number of each type of coins the vending machine has

Describe how your program works.

6. Write a test plan for your vending machine change dispenser (You may need a separate sheet of paper).

Extended question: Terminal Phonebook

Create a copy of the file in Appendix B. The file contains the start of a program for a terminal based phonebook. The program will store names and telephone numbers, the user will be able to add, view, search and delete contacts.

7. Complete the following exercises

   - Implement the functions `addContact`, `viewAllContacts`, `searchContact` and `deleteContact`, use a dictionary to store the names and telephone numbers. The functions should prompt the user for input.
   - Modify your program so it checks the length of the telephone number, if the telephone number is less than 6 characters then it should prompt the user to re-enter the number.
   - Modify your program so when the program exists it writes the content of the phonebook to a file called 'phonebook.csv' in CSV format.
   - Modify your program so when it starts the program checks to see if the file 'phonebook.csv' exists, if it does it should read the content else it should create a new empty phonebook.
   - Modify your program so that you can't add two contacts with the same name.

Extend the menu to add an option to amend a contact's details. Implement the functionality to allow the user to amend a contacts details.

8. Add docstrings to all functions in the phone book program

10. Describe how you might test the Phone book program.

11. Modify your code so that it catches any error that might be raised.
12. Write a test plan for the Phone book application.

Key Points:

- ▷ When writing test cases you should consider:
  - — Possible execution paths
  - — Parameter values (both typical and special cases)
  - — External dependencies (which are best avoided)
- ▷ The design of functions is key to program decomposition.
- ▷ To design a function you should follow the function design recipe;
  - — Determine the function signature and stub
  - — Define type contract
  - — Define the header
  - — Produce documentation
  - — Implement body
  - — Test/debug
- ▷ Docstrings provide documentation for Python classes, methods and function.
- ▷ The notation for docstrings is;

```
"""<<documentation>>"""
```

Notes

Reflection

What did you learn in this section?

Did you encounter any problems? How did you resolve them?

I need to revise

1.
2.
3.

....
Module Staff Signature Date