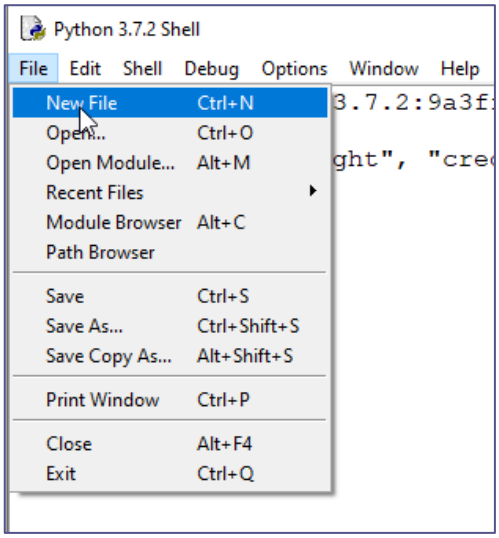
4. FUNCTIONS

Up until now, all the work we have been doing with Python has been completed in the interactive mode, where the result of our program code was immediately displayed but we have not saved anything.



We will now use the text editor or script mode to create and save all our programs:

Step 1: File >> New File/Window (depending on your version of IDLE)

Step 2: When the new window opens, choose File >> Save As and save the file as 'print_a_message.py' in a suitable folder in your drive.

We are now going to start writing our own **functions**.

4.1 WHAT ARE FUNCTIONS?

Functions are blocks of organised code that perform specific tasks; we can reuse them to make our code more EFFICIENT. Python has many ready-made functions that we can use. These are called built-in functions, and you will already have used the print() and input() functions.

Code Efficiency

What does this mean?

At GCSE level it means:

- using the most suitable variable names and data types
- reusing code wherever possible
    - NOT copying and pasting but using a function several times to complete a task
- avoiding unnecessary code
    - by using loops to perform tasks
    - by using if/elif statements to test conditions

Why should we use functions?

- They make your code easier to read as you can see what tasks each block of code performs (providing you use a suitable name).

Functions can be reused instead of repeating code.

- If changes are needed, they can be made to just one block of code.
- It is easier to organise the order in which the program runs.

FUNCTIONS VS PROCEDURES

In Python, the blocks of code we write are ALL called functions. In programming terms, you will need to know the difference between a function and a procedure.

- A procedure is a self-contained block of code that performs a task, e.g. calculating the average of two numbers, printing out a list of numbers. The procedure MAY return a value but it does not have to.
- A function is also a self-contained block of code that performs a task, e.g. asking for data input from a user and checking that the input is valid. A function will ALWAYS return a value. (See Returning a value)

4.2 WRITING A FUNCTION

Here is an example of a very simple function:

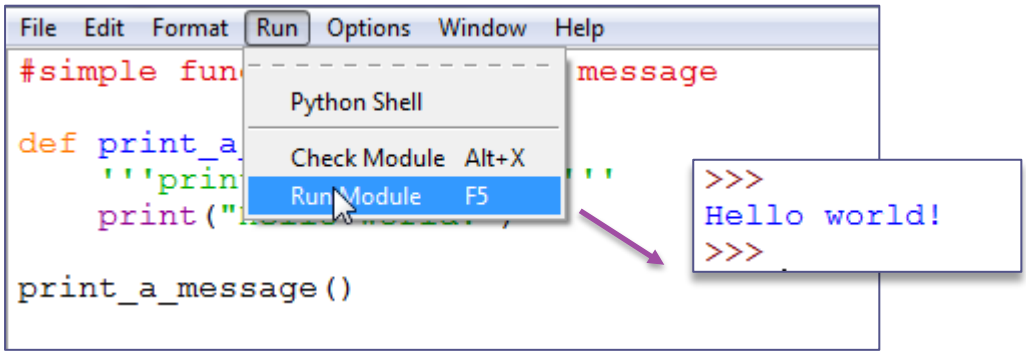
```
#simple function to print a message

def print_a_message():
    '''prints out a string '''
    print("Hello world!")
```

- We use the 'def' statement to tell Python we are defining a function.
- The name should be what the function *does* – keep it simple and use underscores between words.
- The name is followed by two brackets (parentheses); this will hold any variables we want the function to use. These are called the function parameters.
- After the brackets we add a colon.
- The body of the function, the code statements we want to execute, are then indented below – usually by four spaces. Simply press enter after the colon and IDLE will correctly indent for you.

In order to execute or 'run' the function and see the result of the code in the interpreter window, we must 'call' the function.

4.3 CALLING THE FUNCTION

The function will not run until we instruct Python to execute the code by 'calling' the function and EITHER pressing the F5 key OR choosing Run >> Run Module.



4.4 DOCSTRING AND COMMENTS

One of the most important parts of programming is making your code easy for yourself and others to understand at a later date. Using one-line comments and docstrings in your user-defined functions helps with this.

Docstrings are used to explain what the function does and one-line comments are used to add additional explanation to your code.

File Edit Format Run Options Window Help

```
#simple function to print a message

def print_a_message():
    '''prints out a string '''
    print("Hello world!")

print_a_message()
```

Comments are shown by using a hash symbol at the start; these can be added to the end of a line of code to explain what it does. The docstrings should be added on the first line of the function, enclosed by three single OR three double quotation marks.

More guidance on how to maintain your code can be found in ***Chapter 12 – Defensive Design***.

EXERCISE 04: BASIC FUNCTION

We are now saving each exercise.

1. Open IDLE and choose New File/Window and save as BasicFunction1.py
2. Write seperate functions to do the following:

Function 1	Write a function to add two numbers together and print the answer. Use the following variable names and values: x = 17 y = 22
Function 2	Write a function to multiply x and y together and divide by z. Use the following variables in your function: x = 6 y = 4 z = 8

   For each function:
   a. Ensure you have a docstring
   b. Use at least one one-line comment

3. Call each function and ensure that it works

4.5 EXTENDING THE BASIC FUNCTION

So far the functions we have written are very basic; we will now look at extending our functions to use parameters or arguments.

Look at this example:

```
def add_numbers(x,y):
    '''Add variables x& y, print the result'''
    result = x + y  # this produces the result
    print(result)

add_numbers(15,22)

add_numbers(37,98)
```

Functions are 'called' with arguments supplied instead of placeholders

Parameters are the 'placeholders' used when the function is defined. In this example these are x and y.

Arguments are the actual values we use when we 'call' the function.

EXERCISE 05: EXTEND BASIC FUNCTION

1. Develop Function 2 from the previous exercise to use parameters and supply the following arguments:

   x = 15
   y = 13
   z = 5

2. Write a function that will take in any string and print it out followed by 'You're welcome'. Supply the following argument to the function:

   'There are only 10 types of people in the world.'

The next step is to develop our function to RETURN values. A function can take inputs as parameters or not, it can also return a value or not.

4.6 RETURNING A VALUE

When we return a value from a function we do this so we can use that value as an argument for another function or part of the program code.

Look at this example using two more built-in functions:

```
#ask for the user name and print a welcome message

def get_name():
    '''get name input and return'''
    name = input("Please enter your name: ")
    return name

def print_greeting(name):
    '''prints greeting with name variable'''
    print("Hello {0}, welcome to my greeting program.".format(name))

name = get_name()
print_greeting(name)
```

This time when we call the get_name() function, we must tell Python where the variable 'name' is created *before* we use it as an argument in the print_greeting() function.

This example includes two more built-in functions:

- input() – this gets input from the user into our program.
  - Unless specifically instructed otherwise, Python treats the values entered as string data types.
  - it is important to use the correct data types to avoid errors (see ***Chapter 3 – Data Types*** for more on this.
  - This means we must 'cast' the data type to the one we want to use.

```
shoe_size = float(input("Please enter your shoe size: "))

exam_mark = int(input("Please enter your exam mark: "))
```

- format() – this helps with the presentation of outputs.
  - We can use this method to include variables in our text string. Look at 3.3 Strings for more information.

    Simple example:

```
print("GCSE number formats you need to know:
          \n\t1.{0}\n\t2.{1}".format('integer','float'))
```

  - Placeholders are inserted into the text where the arguments are to be inserted. These must use curly braces { } and start at 0.
  - The arguments are placed inside the format() functions brackets.

4.7 RETURNING MULTIPLE VALUES

Sometimes you may write a function where you need to return more than one value; this can be achieved by using a data structure called a tuple (see ***Chapter 9: Tuples***). A tuple is a sequence of values separated by commas. Tuples are sometimes surrounded by parentheses (brackets), but they don't have to be.

Look at this example:

```
1   # return multiple values
2
3   # this function will be called twice with a different prompt each time
4
5
6   def get_name(p):
7       """asks for name string and returns"""
8       n = input("Please enter {0}: ".format(p))
9       return n
10
11
12  def find_names():
13      """asks for two name string and returns"""
14      y_name = get_name("your name")
15      f_name = get_name("your first friend's name")
16      s_name = get_name("your second friend's name")
17      return y_name, f_name, s_name
18
19
20  def print_message(y, f, s):
21      """prints message using two name variables"""
22      print("Hello {0}, your best friends are {1} and {2}".format(y, f, s))
23
24
25  def main():
26      """ runs all programs"""
27      y_name,f_name, s_name = find_names()
28      print_message(y_name, f_name, s_name)
29
30
31  main()
```

PyCharm IDE shown for line numbers

4.8 REUSING FUNCTIONS

One of the advantages of creating user-defined functions – writing your own functions – is that you can design them so that they can be reused.

In the example code on the previous page, there is a 'helper' function on line 6. We can pass different prompts to the function to ask for different names. Although we have called the return value 'n', we can assign a different name to this variable each time we call the function on lines 14, 15 and 16. This makes the code much more efficient as we are reusing the function three times, supplying different **arguments** each time we **call** the function.

We are then using a tuple to return the three string variables on line 17, which are passed into the function on line 20 so they can be inserted into the printed message.

Finally, it is good practice to use a main function to control the order in which your functions are called in your program. As you can see, the variables which are returned from each function need to be listed in the same order as when they are returned in the design of the function.

A common error when passing several variables into a function is that they are passed into the function in the wrong sequence.

Finally, the main function is called on line 31 so that the functions called inside it will run in the **sequence** specified.

EXERCISE 06: RETURNING VALUES

Write functions that will:

a. Ask for the student name and return it.

b. Ask for the name of their Computer Science teacher and return it.

c. Ask for marks out of 10 for the last four homework tasks and return an average mark. *Hint: you will need to calculate the average inside the function.*

d. Display the following message, depending on the average mark:
   i. average >= 8. 'Well done, X, Y is very pleased with your effort'
   ii. average >= 6 <8. 'A good effort, X .Y thinks you should check your work carefully'
   iii. average <=5. 'X this is very poor. Y has asked you to try harder'

Hint: X is the name of the student; Y is the name of the teacher. Remember to call functions correctly and declare variables being returned from each function.

