**Functions**

Functions are the basic building blocks of programming, so an understanding of how functions work is vital to understanding programming. Each function has three components: its **inputs**, its **processing** and its **outputs**. Once defined, the function can be used throughout the program.

**Defining Functions**

Functions are defined once in the program. This definition has a number of aspects.

Consider the following annotated example of a function:

def add1(number\_one, number\_two)2:

 return4 number\_one + number\_two3

1. Functions need a name. This name should be short and meaningful. Since functions are designed to perform an action most programmers will use verbs as the function name where possible.
2. Functions need data to work with. This is the **inputs**. The best way for this data to enter the function is as a parameter of the function, this allows the function to be self-contained and re-usable in other programs. During the calling phase (outlined below) the actual data (called arguments) is substituted into the function in the same order as they are defined. The parameters are defined in the normal way that variables are defined within the language (without any scoping definition since the scope is implicitly defined as the function). In Python this simply means giving the variable a name to be used throughout the function. Since Python is a dynamic language the parameter can be replaced with any variable type when the function is called so care needs to be taken in the body of the function to handle all variable types that could be used.
3. Functions change data. This is the action or **processing** of the function and is a block of code which may call other functions. Best practise is for this processing to be as short as possible.
4. Functions produce data. This is the **output** of the function and can be used elsewhere in the program. This is usually done using the keyword **return** in most languages including Python. There can be multiple returns in the one function and the function will exit when the first return is encountered (and the rest of the function processing will be ignored).

In the above example the function is given the name add and requires two parameters. Inside the function these parameters are added together and the result is send from the function by the return keyword.

There are a number of points to note in the definition of functions:

* Best practise is for functions to be as short as possible and to do one task. This means that they can then be tested and you can have confidence that the function works in all cases with all possible data. One clue that I often follow is that if a function takes more than 10 lines of code, it is probably doing more than one task and should be broken up into multiple functions. This makes the function a building block for more complex functions and programs and allows the function to be used in a large number of different programs. Programmers like to build up groups of these well tested functions which they call libraries. Often programmers release these libraries so that other programmers can use them. All languages come with libraries of functions that are ready for use.
* In well designed programs almost all of the code is inside functions. The only code outside of functions are variables that are required to be used throughout the code, the small code used to start the whole program and comments. If you want to look at well constructed code the language libraries usually offer good examples of how code should be written.
* This current example is designed to add two numbers, however it will not work with three numbers. You would need to make a separate function for that. This is not very useful in programming and some languages solve this by allowing **overloading** of function names. **Overloading** means defining the same function name again with a different set of parameters. There is no limit to how many overloaded functions you have, just so long as the functions have different parameters. Python does not allow for overloading but solves the problem using default and keyword parameters.
* Default parameters are defined by giving the parameter a value during the definition. This value is used in the function if the function is called without a corresponding argument, but if the function is given an argument that value is use (not the default value). The added advantage of default parameters is that when the function is called the arguments do not need to be in the same order, provided that they are simply named.
* Keyword parameters can be used by declaring the arguments not as simple variable but also using the parameter name from the original declaration. This enables you to declare the variable in a different order to the original function definition. If you use a \*\*before the parameter name in the definition this allows the argument to be a dictionary of keyword arguments, meaning that the function can take an unlimited number of arguments. A single \* before a parameter means that the argument will be a tuple of unlimited length. The normal ways that these are defined is \*args, \*\*kwargs.
* An example of this in Python is the print() function which can have various number of arguments, each separated by commas. Eg print(1), print(1,2), print(1,2,3)
* Some languages require you to clearly define what type of data will be accepted as a parameter and then throw an error if the function is called without the correct type of data. Other languages trust the caller of the function to provide the correct type of data. Python uses the second approach so it is often a good idea to check the data that comes into a function when you are creating library functions.
* All language allow functions to call themselves, this is called **Recursion** and is a powerful alternative to looping. Recursion is a whole topic in itself and well beyond the scope of this course.
* In some language functions are allowed to take in other functions as a parameter of the function and also allow you to allocate a function to a variable. These languages are said to treat functions as **first class citizens**. Doing this can lead to highly complex functions which are very efficient (ie run very fast). Python allows this and also to return functions as the output of the function (not just data) however this is also a topic well beyond the scope of this course.
* Some languages allow you to create function inside of functions. This enables functions to use **Closure,** which is a means by which data can be hidden inside the function but its processing can be exposed. This is done by returning the internal function of the function, not the data. Closure is a whole topic in itself and well beyond the scope of this course. Python allows for closure of functions. Python also allows for Decorator functions, which is a special type of Closure and for Monad pattern of functional structure.
* Some languages allow you to create functions without a name. These types of functions are often called **Lambda.** Python allows for Lambda function. Lambda is a whole topic in itself and well beyond the scope of this course.

**Using Functions**

Once a function has been defined (either directly or by importing the library where the function has been defined) it can be used. This is called **calling** the function. When you call the function you must use the function name and provide the arguments (data) that the function requires. The function will then process this data and usually provide you back with data that you can use.

The arguments you give a function can be actual data, sometime called literals, or variables that you have defined elsewhere or the results of other functions which return data to you. For example:

a = add(1,add(2,3))

print(a, add(a,6))

These statements are correct in Python. The function add takes two parameters which it assumes to be integer and returns data. (In reality this function will also work with Floats, Stings and Lists since all can be added together using the + sign).