1 Functions

Functions are (or should be) short sections of code that may be called from within the main programme, or from another function. They usually take some inputs (arguments) and return one or more results.

Functions may be used for two purposes:

1) To avoid repeating code when the same code needs to be run in two or more locations in the main programme.

2) To structure the code, breaking the code down into smaller pieces. In functional programming the whole programme would be broken down into functions

If a function is more than 10 lines long you may want to think whether it could be broken down further, with some of the code pulled out as a separate function.

Ideally a function should do just one thing; it should be as simple as possible, and easy for another person to follow.

A docstring may be used at the start of a function to provide help to the user - in this case if the user types help(function_name) the docstring will be returned. These docstrings may also give examples of the function.

Any variables defined within functions are destroyed when the function closes. Functions are not usually used to change global variables (those variables defined in the main programme code). If a global variable is passed to a function any changes to that variable made by the function are lost when the function ends (and so global variables are protected from being changed by functions). There is a way to force the function to change the global variable, which we will cover in a later lesson, but this should usually be avoided if possible.

1.1 Defining a function

A function is defined with the def statement. It is followed by the function's inputs (arguments or 'args') which may also be given a default value. The user may enter the inputs either by name, or in the order in which the function expects them.

It is good practice to use verbs to define functions, to describe what they do.

Let's do a very simple example, raising a number to the power of another number, and we will set a default where, if no other number is given, we will raise to the power of zero (which always returns a value of 1). Note that we define a default value with the = sign.

```
def raise_to_power (number, power = 0):
    result = number ** power
    return result
```

Once the function has been defined we can call it:

```
print (raise_to_power(2,6))
```

```
OUT:
```

```
64
```

We could describe the inputs by their names if we wanted to:

```
print (raise_to_power(number = 2, power = 6))
```

```
OUT:
```

```
64
```

If we define the inputs by name, the order becomes unimportant (without the names the function expects inputs in the order defined in the function):

```
print (raise_to_power(power = 6, number = 2))

OUT:

64
```

And remember that we set a default of power to zero, so if we only pass the first argument ('number'), the power defaults to zero:

```
print (raise_to_power(2))

OUT:

1
```

1.2 Adding help to a function with a docstring

A help docstring may be added to a function using triple quotes (' or ") after the function has been defined. This may be called later by the user using help.

```
def raise_to_power (number, power = 0):
    """
    This functions takes two numbers, and raises the first number to the power of the second.
    If no second number, or power, is given then power defaults to zero
    """

    result = number ** power
    return result
```

Calling help:

```
help (raise_to_power)

OUT:

Help on function raise_to_power in module __main__:

raise_to_power(number, power=0)
    This functions takes two numbers, and raises the first number to the power of the second.
    If no second number, or power, is given then power defaults to zero
```

1.3 Returning two or more results

Sometimes we may wish to return more than one value. There are two ways of doing this:

1) Return the values separately (the code that calls the argument must also refer to both results:

```
def calculate_sum_and_product(a, b):
    """Return sum and product of two numbers"""

    my_sum = a + b
    my_product = a * b
    return my_sum,my_product

# When calling the function we need to put both results generated by the function

calculated_sum, calculated_product = calculate_sum_and_product(5,6)
print (calculated_sum)
print (calculated_product)
```

```
OUT:

11
30
```

2) A more common way would be to return a single result that is a container (commonly a list or a tuple) of the two results. Here we return a tuple of the two results (remember that tuples are like lists, but they cannot be changed). We can tell that the returned value is a tuple by the use of curved brackets (a list could also be used, which we would spot by square brackets.

```
def calculate_sum_and_product(a, b):
    """Return sum and product of two numbers"""
    my_sum = a + b
    my_product = a * b
    result = (my_sum, my_product) # this is a tuple
    return results

results = calculate_sum_and_product(5,6)

# Individual results are obtained by their index:

print (results[0])
print (results[1])

OUT:

11
30
```