Functions

Functions are a powerful programming construct. They allow you to create code that takes inputs, performs a computation, and produces an output. You can then use these functions over and over to compute results based upon different inputs. This is an example of an *abstraction*. Abstraction is a fundamental concept in computer science, whereby you abstract away the details of something. Effectively, the caller of a function only needs to know how to call the function and *what* it does. The caller does not need to know *how* the function does it!

Anatomy of a Function in Python

Let's consider a simple function that returns the sum of the squares of two inputs:

def sum\_of\_squares(num1, num2):

    """

    Return the sum of the squares of the two inputs.

    """

    sq1 = num1 \* num1

    sq2 = num2 \* num2

    sumsq = sq1 + sq2

    return sumsq

Let's break down the elements of the function:

1. **def**: All function definitions must begin with the Python keyword \color{red}{\verb|def|}def (short for "define"). Note that def is not indented at all. In general, this will be the case. As you learn more Python, you will find that there are exceptions to this, but at this point, your functions will always start at the beginning of a line.
2. **name**: After the keyword def, your function must have a name. In this case, it is \color{red}{\verb|sum\_of\_squares|}sum\_of\_squares. Function names must begin with a letter. Function names can include letters, numbers, and underscore (\_) characters. By convention in Python function names should not use uppercase letters and words should be separated by underscores. You should choose clear, understandable function names that indicate what the function does.
3. **parameters**: The parameters, or inputs, of the function appear directly after the name. They are surrounded by parentheses. A function in Python can have zero or more parameters. The parameters should be separated by commas. In the above example, there are two parameters,\color{red}{\verb|num1|}num1 and \color{red}{\verb|num2|}num2.
4. **colon**: After the parameters, there must be a colon. Python requires this colon to indicate that what follows will be the code for the function.
5. **indentation**: Note that the remainder of the function must be indented. In some languages, indentation is optional and is used simply to help the person reading the code to understand what is and what is not a part of the function. In Python, this indentation is **required**. By convention, the code of the function should all be indented exactly 4 spaces.
6. **docstring**: The line immediately following the colon should be a docstring (short for "documentation string"). The docstring for this function is on lines 2 through 4. This is a multiline string that describes, in English, what the function does. You should not be describing how the function performs the computation. Rather, this is a description of how to use the function. The docstring can be as long as necessary to explain the behavior of the function. However, it should be clear and concise so as not to confuse the reader.
7. **body**: After the docstring, is the body of the function. This is the actual code that will be executed when the function is called. The code for this function is on lines 5 through 8. A function can contain any valid Python code. This function includes arithmetic expressions and variable assignments. Note that you can even call other functions.
8. **return**: Functions often return values to the caller. This is not strictly required however. The function may be performing actions (such as printing messages) that do not require it to return anything to the caller. However, more often than not, a function is computing some result that should be returned to the caller. The return keyword is used to indicate what should be returned from the function. Whatever appears after the return keyword is what will be returned. This can be any expression, which will first be evaluated and then the result will be returned. In this case, the value of the variable \color{red}{\verb|sumsq|}sumsq will be returned. If you omit the return statement from your function, Python will automatically return the special value \color{red}{\verb|None|}None from the function.

All functions should have these elements (except possibly the return statement). You may define as many functions as you need to perform the computation of your program.

## Calling Functions in Python

In order to call a function, you must use the function's name, pass it any arguments it needs, and then optionally do something with the return value. For example, you could call the sum\_of\_squares function as follows:

number = 3

value = sum\_of\_squares(number, 5)

print(value)

Here, the function \color{red}{\verb|sum\_of\_squares|}sum\_of\_squares is actually being called on line 2. Two arguments are being passed to the function: \color{red}{\verb|number|}number and 5. Note that you can write any Python expression as an argument. It will first be evaluated and then the result will be passed to the function. So, the variable number will be evaluated to 3, and then 3 and 5 will be passed to the function. These values will be assigned to the parameters, \color{red}{\verb|num1|}num1 and \color{red}{\verb|num2|}num2, of the function. When the function returns, that return value will then be assigned to the variable \color{red}{\verb|value|}value. In this case, the number 34 will be assigned to the variable \color{red}{\verb|value|}value. Finally, line 3 will print this result, so 34 will be printed in the output console.

Functions can be called as many times as you would like with different arguments. This is a key element of the function abstraction. You now have a tool to compute the sum of squares of two numbers. Whenever you need to do that, you just call the function \color{red}{\verb|sum\_of\_squares|}sum\_of\_squares with the appropriate arguments and do not need to worry about the code necessary to compute the result. Each time it will produce the appropriate result given the arguments it was passed.

## Local Variables within Functions

Let's return to the definition of our function again:

def sum\_of\_squares(num1, num2):

    """

    Return the sum of the squares of the two inputs.

    """

    sq1 = num1 \* num1

    sq2 = num2 \* num2

    sumsq = sq1 + sq2

    return sumsq

There are two types of *local* variables that exist in this function: the parameters and variables that are defined within the function body. These local variables are newly created each time the function is called and then they only exist during the execution of the function. This is important to understand. You cannot, for instance, refer to the variable \color{red}{\verb|sq1|}sq1 outside of this function. It does not exist! It is created during the execution of the function and is destroyed as soon as the function returns. Similarly, you cannot refer to the parameter num1 outside of the function. When the function is called num1 is assigned the value of the first argument that was passed to the function. You can then use this variable within the body of the function. But, again, it is destroyed as soon as the function is returned.

It is a common mistake to try to refer to parameters and other local variables of a function outside of the body of the function. This will not work. Furthermore, if you create variables with the same names as those within the function, they are *different* variables that have nothing to do with the variables you have created inside the function. When you call functions, you should only try to interact with functions by passing them arguments and using their return values. The caller of a function should not try to use any of the local variables of a function outside of that function.