A function is a block of organized, reusable code used to perform a single, related action. They provide better modularity for your program and offer a high degree of code reuse. You might already be familiar with some functions in Python, such as print() and input(). You can also create your own functions with their own purposes, which are known as user-defined functions.

To make a function, you first have to define one using the def keyword, followed by whatever name you decide to give your function. You should always try to name your function something that relates to what it does in order to make its purpose easily understandable. After the function name, you have to append an open and closing parenthesis, which may or may not contain parameters within them depending on how you want to make your function. Finally, add a colon to signify where the contents of your function begins.

An example user-defined function might look as follows:

Def myFunction(i, j):

The contents of our function can then be written under this header.

The final important keyword related to function definitions is the *return* keyword, which exits a function and optionally passes a parameter of your choosing back to the caller. Alternatively, you have the option of just using the return keyword on its own without supplying an expression, in which case it will pass back Nothing or Null.

Let’s create a simple user-defined function that gets the sum of two integers.

Def MySum(I, j):

a = i + j

return a

Let’s analyze this. We first define our function using the def keyword, followed by our function name (MySum), followed by two parameters that we pass into our open parenthesis. These two variables, I and j, are the values that will be added together when we called our function from our main script. On the following line, we create a variable *a* to act as the sum of I and j. Finally, this value *a* is returned, as signified by the return statement.

We can now test our function by asking the user to input two numbers, and then print the result after we pass those two numbers into our function.

def MySum(i,j):

a = i + j

return a

x = int(input("Enter first number:"))

y = int(input("Enter second number:"))

total = MySum(x, y)

print("Sum:",total)

So how does this calculate the sum of our two numbers? Well, first our x and y value are assigned to the values that we give them. Then the line following our inputs, where MySum is called, accepts our x and y as inputs. The values we entered for x and y are passed as the values for I and j respectively when the function runs. This is known as passing by value, and essentially what that means, is a copy of our values are passed and given to the parameters in the function header. So when our function runs, i is assigned the value that was in x, and j is assigned the value that was in y. The reason why I goes to x and j goes to y is a matter of the order in which we passed them into the function. Think of it like a contract, where our function definition states the number of parameters required for us to use it, and the order in which they must be defined. So MySum expects two parameters to be passed in, and then those parameters will be added and returned. We fulfill this contract by passing in exactly two parameters of integer value. If we were to only pass one of our values (x or y) into the function, it would give us an error. Now, once the values for I and j have been established, the rest of the function runs, and so the line that sums the two values are run, and stores the result in a local variable *a*. Finally, *a* is returned by the function, and this value is then assigned to our total variable in our main script.

So why are functions useful? Obviously, if we wanted do something as simple as find the sum of two numbers, we could just add them in our code, or use the built-in *sum* function from the very beginning. For trivial cases like these, using a function may not be the best option. If however, we wanted to solve a more complicated problem, then using a function could be greatly beneficial. Consider the following function:

def CountVowels(str):

counter = 0

vowels = ['a','e','i','o','u','A','E','I','O','U']

for c in str:

if c in vowels:

counter = counter + 1

return counter

As its name would suggest, this function counts and returns the number of occurrences of a vowel in a given string. If we wanted to count the number of vowels in three separate strings, we’d have to copy and paste this code segment several times, once for each string we wish to parse. Putting this segment in a user-defined function, however, makes managing our program much easier, since now we only have to pass in the strings that we wish to parse without having to rewrite the same code for each string. Let’s add the following segment to our script:

phrases = ["The quick brown fox jumps over the lazy dog",

"The five boxing wizards jump quickly.",

"Pack my box with five dozen liquor jugs."]

for p in phrases:

numVowels = CountVowels(p)

print("In '",p,"'")

print("Number of vowels:",numVowels)

If we run this script, the number of vowels will be displayed for each of our three phrases in our *phrase* list. But why is that? Let’s take it from the top. First we define a user-defined function, CountVowels, which returns the number of times any vowel occurs in a given string. The function accepts one string parameter and returns and integer value representing the number of vowel occurrences. Below our function is our list of phrases, which is initialized to contain three different values. Finally, we write the main code of our program that will be run. Note that even though our function definition is written above our main code, the function will not be run unless the main code calls it. The CountVowels function is called on the first line under our For Loop. For each element in our list of phrases, pass the element into our CountVowels function. The number of vowels per phrase is returned and assigned to variable *numVowels*. From here, we print the current element that we’re analyzing, and then finally print the number of vowels in that phrase. We do this for each element in our list, calling the CountVowels function a total of three times. We have effectively determined the number of vowels in each phrase without having to rewrite the logic in our CountVowels function. We could continue to add more phrases to our list, and our program would still count the number of vowels for every item without having to change any of the logic.

To summarize, functions are a powerful tool that allow for code to be modularized. They can be used to create blocks of reusable code. Functions can be made to return different types of values, such as integers and strings, and these values can be assigned to variables just like any normal value. The amount and type of parameters a function requires should be treated like a contract that must be fulfilled in order for the function to be used. Functions do not always have to return a value, and can instead return nothing if the *return* keyword is used on its own.