# Lab 8 - Python Functions

When building programs, one of the most critical steps is identifying the necessary inputs, processing, and output components of the program. For more complex programs it may be helpful to break the program down into smaller tasks that can operate independently with their own set of input, processing, and output. We do this by defining functions to represent these smaller tasks that we can then call to execute the set of steps.

## Function components

There are several components to be aware of when building a function. Let’s look at the example function below to identify the components.

A screenshot of a computer

Description automatically generated with medium confidence

On line 1, you have the def keyword. This tells python you are defining a function. After a space we have the name of the function “sample\_function”. Within the parenthesis that follow we can define the inputs for our function. Inputs for a function are called parameters and we have defined them as parameter1 and parameter2 to reinforce what they are. After the parenthesis, we have a colon. This indicates the following indented lines are part of the function declared in the header.

After line 1, we see the next two lines are indented. This means that they are part of the function. Every time the function is called, the indented steps will be executed. In line 2 we see some calculation being performed. This is the processing step of the function. Whenever we write a function, we need to decide how to take the inputs we are given and use them to generate the desired output. In this case, the calculation is as simple as adding the 2 inputs together, but we can obviously design more complicated functions.

On line 3, we see a return statement. This is where we specify the output of our function. Whenever a function is called, the return statement will send the value or set of values after it to the calling statement.

On line 5, outside of the function, we see the function being called. The returned\_value variable is being set to the value of the output variable in the return statement. The next line prints this out and the result of the program execution is shown below.



## Function Headers

In the following programs, write the appropriate header for a function. They should include the def keyword, the function name and any of the requested parameters.

**Example**: Write the header of a function called greeting() that accepts a parameter called name.

**Answer**: def greeting(name):

**Question 1**: Write the header of a function called user\_lookup() that accepts a parameter called username.

**Answer**: def user\_lookup(username):

**Question 2**: Write the header of a function called get\_total() that accepts 2 parameters called group1 and group2.

**Answer**: def get\_total(group1, group2):

**Question 3**: Write the header of a function called compare\_results() that accepts 2 parameters called result1 and result2.

**Answer**: def compare\_results(result1,result2):

## Processing Steps in Functions

In the following examples, you will be provided a header and a return statement along with high level instructions on what steps must be taken to generate the requested output based on the provided parameters.

**Example**: The **suggested\_tip()** function should receive the **meal\_cost** and **tip\_percent** as inputs. The processing step should multiply them together and divide by 100 to return an appropriate tip amount based on those inputs. In the answer section, write the missing line.

def suggested\_tip(meal\_cost, tip\_percent):

# What goes here?

return tip\_amount

**Answer**: tip\_amount = meal\_cost \* tip\_percent / 100

**Question 4**: The **calculate\_wages()** function should receive **hours\_worked** and **hourly\_wages** as inputs. The processing step should multiply them together to return wages based on those inputs. In the answer section, write the missing line.

def calculate\_wages(hours\_worked, hourly\_wages):

# What goes here?

return wages

**Answer**: wages = hours\_worked \* hourly\_wages

**Question 5**: The **sales\_tax()** function should receive **pretax** as the only input. The function should multiply the pretax amount by 0.0688 to get the tax amount. In the answer section, write the missing line.

def sales\_tax(pretax):

# What goes here?

return tax\_amount

**Answer**: tax\_amount = pretax \* .0688

**Question 6**: The **total\_with\_tax()** function should receive **pretax** as the only input. The function should multiply the pretax amount by 1.0688 to get the total. In the answer section, write the missing line.

def total\_with\_tax(pretax):

# What goes here?

return total

**Answer**: total = pretax \* 1.0688

**Question 7**: The **get\_average()** function should receive 2 parameters **value1** and **value2** as the input. The function should add them together and divide the sum by 2 to get the average. In the answer section, write the missing line.

def get\_average(value1, value2):

# What goes here?

return average

**Answer**: average = (value1 + value2)/2

**Question 8**: The **get\_rectangle\_area()** function should receive **a** and **b** as inputs. Set **area** equal to the product of **a** and **b**. In the answer section, write the missing line.

def get\_rectangle\_area(a, b):

# What goes here? Area = a \*b

return area

**Answer**:

## Return Statements

In the following examples, you will be provided a function header and the required processing steps. You will need to write the proper return statement.

**Example**: The **cube\_volume()** function receives input parameters of **length**, **width** and **height**. It multiplies them together to return the volume.

def cube\_volume(length, width, height):

volume = length \* width \* height

# What goes here?

**Answer**: return volume

**Question 9**: The **rectangle\_perimeter()** function receives input parameters of **length**, and **width**. It multiplies each of them by 2 and adds the result together to return the perimeter.

def rectangle\_perimeter(length, width):

perimeter = length \*2 + width \* 2

# What goes here?

**Answer**: return perimeter

**Question 10**: The **circle\_area()** function receives the input parameter **radius**. The function squares the radius and multiplies it by pie to get the area.

def circle\_area(radius):

area = radius\*\*2 \* 3.14

# What goes here?

**Answer**: return area

**Question 11**: The **join\_text()** function receives the input parameters of **string1** and **string2**. The function concatenates the strings to return a concatenated string.

def join\_text(string1, string2):

concatenated = string1 + string2

# What goes here?

**Answer**: return concatenated

## Calling Functions

Now that we have gone over the main steps to define a function, let’s practice calling a function. One of the great things about function is they allow us to compartmentalize our work. Once they are created, we don’t need to worry about how they work. When calling a function, we just need to consider the following:

* What is the function name?
* What does the function accept as arguments?
* What does the function return?

**Example**: Earlier we defined a function called suggested\_tip(), it accepted arguments for **meal\_cost** and **tip\_percent**. It returned a **tip\_amount**. How would you call this function with a **meal\_cost** of 100 and a **tip**\_**percent** of 15 and store the result in a variable called **tip\_amount**?

**Answer**: tip\_amount = suggested\_tip(100, 15)

**Question 12**: Using the same **suggested\_tip**() from the example above. How would you call this function with a **meal\_cost** of 200 and a **tip\_percent** of 20 and store the result in a variable called **tip\_amount**?

**Answer**: tip\_amount = suggested\_tip(200,20)

**Question 13**: Using the same **suggested\_tip**() from the example above. How would you call this function with a **meal\_cost** of x and a **tip\_percent** of y and store the result in a variable called **tip\_amount**?

**Answer**: tip\_amount = suggested\_tip(x,y)

**Question 14**: Earlier we defined a function called **get\_average**(), it accepted arguments for **value1** and **value2** and returned the average of the 2 values. How would you call this function with inputs of **5** and **7** and store the result in a variable called **the\_average**?

**Answer**: the\_average = get\_average(5,7)

**Question 15**: Earlier we defined a function called **get\_average**(), it accepted arguments for **value1** and **value2** and returned the average of the 2 values. How would you call this function with inputs of **x** and **y** and store the result in a variable called **the\_average**?

**Answer**: the\_average = get\_average(x,y)

## All Together now

Now let’s try and apply the concepts we have practiced to build and call a few simple functions.

**Example**: Write a function called **triangle\_perimeter**(), it should accept 3 parameters for **side1**, **side2** and **side3** as inputs. It should add these together to calculate the **perimeter** and return it. After defining the function, call it with values of 3, 4, and 5 and store the result in a variable called perimeter.

**Answer**:

def triangle\_perimeter(side1, side2, side3):

perimeter = side1 + side2 + side3

return perimeter

perimeter = triangle\_perimeter(3,4,5)

**Question 16**: Write a function called square\_area(). It should accept one parameter called **side** for input. The function should calculate the **area** as **side** squared and return the area. After defining the function, call it with the value of 5 and store the result in a variable called area.

**Answer**:

def square\_area(side):

area = side \*\* 2

return area

area = square\_area(5)

**Question 17**: Write a function called square\_perimeter(). It should accept one parameter called **side** for input. The function should calculate the perimeter as **side** times 4 and return the perimeter. After defining the function, call it with the value of 3 and store the result in a variable called perimeter.

**Answer**:

def square\_perimeter(side):

perimeter = side \* 4

return perimeter

perimeter = square\_perimeter(3)