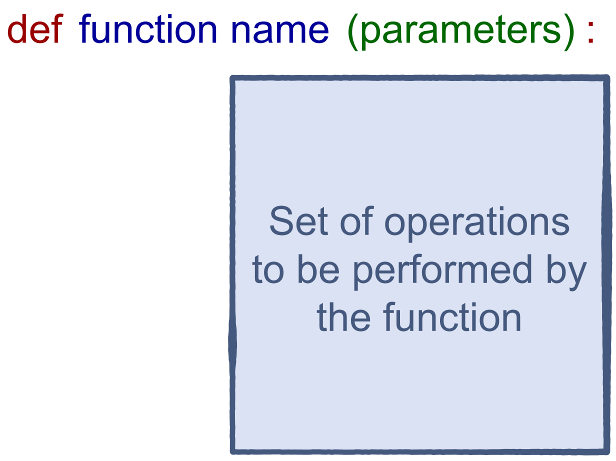
Function Creation

In this lesson, we'll learn how to create a function.

We'll cover the following

Components of a Function

We've studied what functions are and why they are used, but how do we actually make one? In Python, a function can be *defined* using the `def` keyword in the following format:



def function name (parameters) :
Set of operations to be performed by the function

The `function name` is simply the name we'll use to identify the function.

The `parameters` of a function are the inputs for that function. We can use these inputs within the function. Parameters are optional. We'll get to know more about these later.

The body of the function contains the set of operations that the function will perform. This is always indented to the right.

Implementation

Implementation

Let’s start by making a plain function that prints four lines of text. It won’t have any parameters. We’ll name it `my_print_function`. We can call the function in our code using its name along with empty parentheses:

```
def my_print_function():  # No parameters
    print("This")
    print("is")
    print("A")
    print("function")
# Function ended


# Calling the function in the program multiple times
my_print_function()
my_print_function()
```

And just like that, we’ve created our first function! We can reuse it whenever we want.

Function Parameters

Parameters are a crucial part of the function structure.

They are the means of passing data to the function. This data can be used by the function to perform a meaningful task.

When creating a function, we must define the number of parameters and their *names*. These names are only relevant to the function and won’t affect variable names elsewhere in the code. Parameters are enclosed in parentheses and separated by commas.

The actual values/variables passed into the parameters are known as **arguments**.

In the previous lesson, we saw the `min()` function. It requires two numbers as inputs and prints the smaller one.

Let’s define our own basic form of the `min()` function that simply prints the minimum. We’ll name it `minimum()`:

```
def minimum(first, second):
    if (first < second):
        print(first)
    else:
        print(second)


num1 = 10
num2 = 20

minimum(num1, num2)
```

Here, we are passing `num1` and `num2` to the function. The positions of the parameters are important. In the case above, the value of `num1` will be assigned to first as it was the first parameter. Similarly, the value of `num2` assigned to second.

If we call a function with lesser or more arguments than originally required, Python will throw an error.

A parameter can be any sort of data object; from a simple integer to a huge list.

The `return` Statement

So far, we've only defined functions that print something. They don't return anything back to us. But if we think back, functions return values all the time. Just take `len()` for example. It returns an integer which is the length of the data structure.

To return something from a function, we must use the `return` keyword. Keep in mind that once the `return` statement is executed, the compiler ends the function. Any remaining lines of code after the `return` statement will not be executed.

Let's refactor the `minimum()` method to return the smaller value instead of printing it. Now, it'll work just like the built-in `min()` function with two parameters:

```
def minimum(first, second):
    if (first < second):
        return first
    else:
        return second


num1 = 10
num2 = 20

result = minimum(num1, num2)  # Storing the value returned by the function
print(result)
```

We’ve learned how to create a function, set its parameters, provide it arguments, and return data from it. Python really does make it a simple process.

It is a good practice to define all our functions first and then begin the main code. Defining them first ensures that they can be used anywhere in the program safely.

In the next lesson, we’ll talk about function scope.