Function Scope

This lesson will highlight some important things to know about a function's scope.

We'll cover the following

The scope of a function means the extent to which the variables and other data items made inside the function are accessible in code.

In Python, the function scope is the function's body.

Whenever a function runs, the program moves into the function scope. It moves back to the outer scope once the function has ended.

Data Lifecycle

In Python, data created inside the function cannot be used from the outside unless it is being returned from the function.

Variables in a function are isolated from the rest of the program. When the function ends, they are released from memory and cannot be recovered.

The following code will never work:

```
def func():
    name = "Stark"


func()
print(name)  # Accessing 'name' outside the function
```

This will show an error that 'name' is not defined

As we can see, the `name` variable doesn't exist in the outer scope, and Python lets

us know.

Similarly, the function cannot access data outside its scope unless the data has been passed in as an argument.

```
name = "Ned"


def func():
    name = "Stark"


func()
print(name)  # The value of 'name' remains unchanged.
```

Altering Data

When **mutable** data is passed to a function, the function can modify or alter it. These modifications will stay in effect outside the function scope as well. An example of mutable data is a list.

In the case of **immutable** data, the function can modify it, but the data will remain unchanged outside the function's scope. Examples of immutable data are numbers, strings, etc.

Let's try to change the value of an integer inside a function:

```
num = 20


def multiply_by_10(n):
    n *= 10
    num = n  # Changing the value inside the function
    print("Value of num inside function:", num)
    return n


multiply_by_10(num)
print("Value of num outside function:", num)  # The original value remains unchanged
```

So, it's confirmed that immutable objects are unaffected by the working of a function. If we really need to update immutable variables through a function, we

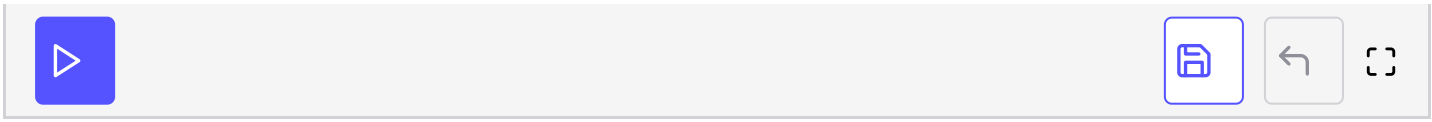
can simply assign the returning value from the function to the variable.

Now, we'll try updating a mutable object through a function:

```
num_list = [10, 20, 30, 40]
print(num_list)


def multiply_by_10(my_list):
    my_list[0] *= 10
    my_list[1] *= 10
    my_list[2] *= 10
    my_list[3] *= 10


multiply_by_10(num_list)
print(num_list)  # The contents of the list have been changed
```



We passed `num_list` to our function as the `my_list` parameter. Now, any changes made to `my_list` will reflect in `num_list` outside the function. This would not happen in the case of an immutable variable.

By this point, we have a better understanding of the scope of a function.

In the next lesson, we'll look at some of Python's built-in functions.