Functions and variables

Previously, you focused on working with multiple parameters and arguments in functions and returning information from functions. In this reading, you'll review these concepts. You'll also be introduced to a new concept: global and local variables.

Working with variables in functions

Working with variables in functions requires an understanding of both parameters and arguments. The terms parameters and arguments have distinct uses when referring to variables in a function. Additionally, if you want the function to return output, you should be familiar with return statements.

Parameters

A **parameter** is an object that is included in a function definition for use in that function. When you define a function, you create variables in the function header. They can then be used in the body of the function. In this context, these variables are called parameters. For example, consider the following function:

```
def remaining_login_attempts(maximum_attempts, total_attempts):

    print(maximum_attempts - total_attempts)
```

This function takes in two variables, *maximum_attempts* and *total_attempts* and uses them to perform a calculation. In this example, *maximum_attempts* and *total_attempts* are parameters.

Arguments

In Python, an **argument** is the data brought into a function when it is called. When calling *remaining_login_attempts* in the following example, the integers *3* and *2* are considered arguments:

```
remaining_login_attempts(3, 2)
```

These integers pass into the function through the parameters that were identified when defining the function. In this case, those parameters would be *maximum_attempts* and *total_attempts*. *3* is in the first position, so it passes into *maximum_attempts*. Similarly, *2* is in the second position and passes into *total_attempts*.

Return statements

When defining functions in Python, you use return statements if you want the function to return output. The *return* keyword is used to return information from a function.

The *return* keyword appears in front of the information that you want to return. In the following example, it is before the calculation of how many login attempts remain:

def remaining_login_attempts(maximum_attempts, total_attempts):

    return maximum_attempts - total_attempts

Note: The *return* keyword is not a function, so you should not place parentheses after it.

Return statements are useful when you want to store what a function returns inside of a variable to use elsewhere in the code. For example, you might use this variable for calculations or within conditional statements. In the following example, the information returned from the call to *remaining_login_attempts* is stored in a variable called *remaining_attempts*. Then, this variable is used in a conditional that prints a "*Your account is locked*" message when *remaining_attempts* is less than or equal to *0*. You can run this code to explore its output:

```
def remaining_login_attempts(maximum_attempts, total_attempts):
    return maximum_attempts - total_attempts
remaining_attempts = remaining_login_attempts(3, 3)
if remaining_attempts <= 0:
    print("Your account is locked")
```

```
Your account is locked
```

In this example, the message prints because the calculation in the function results in *0*.

Note: When Python encounters a *return* statement, it executes this statement and then exits the function. If there are lines of code that follow the *return* statement within the function, they will not be run. The previous example didn't contain any lines of code after the *return* statement, but this might apply in other functions, such as one containing a conditional statement.

Global and local variables

To better understand how functions interact with variables, you should know the difference between global and local variables.

When defining and calling functions, you're working with local variables, which are different from the variables you define outside the scope of a function.

Global variables

A **global variable** is a variable that is available through the entire program. Global variables are assigned outside of a function definition. Whenever that variable is called, whether inside or outside a function, it will return the value it is assigned.

For example, you might assign the following variable at the beginning of your code:

device_id = "7ad2130bd"

Throughout the rest of your code, you will be able to access and modify the *device_id* variable in conditionals, loops, functions, and other syntax.

Local variables

A **local variable** is a variable assigned within a function. These variables cannot be called or accessed outside of the body of a function. Local variables include parameters as well as other variables assigned within a function definition.

In the following function definition, *total_string* and *name* are local variables:

```
def greet_employee(name):

    total_string = "Welcome" + name

    return total_string
```

The variable *total_string* is a local variable because it's assigned inside of the function. The parameter *name* is a local variable because it is also created when the function is defined.

Whenever you call a function, Python creates these variables temporarily while the function is running and deletes them from memory after the function stops running.

This means that if you call the *greet_employee()* function with an argument and then use the *total_string* variable outside of this function, you'll get an error.

Best practices for global and local variables

When working with variables and functions, it is very important to make sure that you only use a certain variable name once, even if one is defined globally and the other is defined locally.

When using global variables inside functions, functions can access the values of a global variable. You can run the following example to explore this:

```
username = "elarson"
def identify_user():
    print(username)
identify_user()
```

```
elarson
```

The code block returns "*elarson*" even though that name isn't defined locally. The function accesses the global variable. If you wanted the *identify_user()* function to accommodate other usernames, you would have to reassign the global username variable outside of the function. This isn't good practice. A better way to pass different values into a function is to use a parameter instead of a global variable.

There's something else to consider too. If you reuse the name of a global variable within a function, it will create a new local variable with that name. In other words, there will be both a global variable with that name and a local variable with that name, and they'll have different values. You can consider the following code block:

```
username = "elarson"
print("1:" + username)
def greet():
    username = "bmoreno"
    print("2:" + username)
greet()
print("3:" + username)
```

```
1:elarson
2:bmoreno
3:elarson
```

The first print statement occurs before the function, and Python returns the value of the global *username* variable, "*elarson*". The second print statement is within the function, and it returns the value of the local *username* variable, which is "*bmoreno*". But this doesn't change the value of the global variable, and when *username* is printed a third time after the function call, it's still "*elarson*".

Due to this complexity, it's best to avoid combining global and local variables within functions.

Key takeaways

Working with variables in functions requires understanding various concepts. A parameter is an object that is included in a function definition for use in that function, an argument is the data brought into a function when it is called, and the *return* keyword is used to return information from

a function. Additionally, global variables are variables accessible throughout the program, and local variables are parameters and variables assigned within a function that aren't usable outside of a function. It's important to make sure your variables all have distinct names, even if one is a local variable and the other is a global variable.

Revision #1
Created 2023-12-21 11:36:43 UTC by naruzkurai
Updated 2023-12-27 11:35:24 UTC by naruzkurai