Set 13: Functions Part 2

Skill 13.1: Differentiate between built-in and user-defined functions

Skill 13.1 Concepts

There are two distinct categories for functions in the world of Python. What we have been learning about in the last lesson are called *user-defined Functions* - functions that are written by users (like us!).

There is another category called *built-in* functions - functions that come built into Python for us to use. Remember when we were using *print* or *str*? Both of these functions are built into the language for us, which means we have been using built-in functions all along!

There are lots of different built-in functions that we can use in our programs. Take a look at this example of using the *len* function,

```
destination_name = "Venkatanarasimharajuvaripeta"


# Built-in function: len()
length_of_destination = len(destination_name)


# Built-in function: print()
print(length_of_destination)
```

Here we are using a total of two built-in functions in our example: *print*, and *len*.

And yes, if you're wondering, that is a real railway station in India!

There are even more obscure built-in functions like *help* where Python will print a link to documentation for us and provide some details,

Code

```
help("string")
```

Output

```
Help on module string:

NAME
        string - A collection of string constants.

MODULE REFERENCE
        https://docs.python.org/3.11/library/string.html

        The following documentation is automatically generated from the Python
        source files.  It may be incomplete, incorrect or include features that
        are considered implementation detail and may vary between Python
        implementations.  When in doubt, consult the module reference at the
        location listed above.
...
```

Check out all the latest built-in functions in the official Python docs.

Let's practice using a few of them. You will need to rely on the provided Python documentation links to find your answers!

Skill 13.1 Exercise 1

Skill 13.2: Interpret variable scope

Skill 13.2 Concepts

As we expand our programs with more functions, we might start to ponder, where exactly do we have access to our variables? To examine this, let's revisit a modified version of a function we've seen previously.

```
def class_welcome(name1, name2):
  print("Welcome to Python!")
  print("Get to work, " + name1)
  print(name2 + " sit!")
```

What if we wanted to access the variable destination outside of the function? Could we use it? Take a second to think

Check the result below!

Code

```
def class_welcome(name1):
  print("Welcome to Python!")
  print("Get to work, " + name1)
class_welcome("Reid")
print(name1)
```

Output

```
Welcome to Python!
Get to work, Reid
NameError: name 'name1' is not defined
```

Notice when we run the code above, we get a *NameError*, telling us that *name1* is not defined. The variable *name1* has only been defined inside the space of a function, so it does not exist outside the function.

We call the part of a program where *name1* can be accessed its *scope*. The *scope* of *name1* is only inside the *class_welcome*.

Let's consider another example,

Code

```
name1 = "Reid"
def class_welcome():
  print("Welcome to Python!")
  print("Get to work, " + name1)
class_welcome()
print(name1)
```

name1 is accessible everywhere because it is global

Output

```
Welcome to Python!
Get to work, Reid
Reid
```

In the above example, we were able to access the *name1* variable both inside the *class_welcome* function as well as our *print* statement. If a variable lives outside of any function it can be accessed anywhere in the file.

Skill 13.2 Exercise 1

Skill 13.3: Apply the global keyword to variables

Skill 13.3 Concepts

Recall that in Python there is no command for declaring variables. That is, a variable is created the moment you first assign a value to it.

Consider the following example,

Code	Explanation

`player = "x"`
`computer = "o"`
 	
`def start_game():`
`  player = input("Do you want to be 'x' or 'o'?")`
 	
`start_game()`
`print(player)`
`print(computer)`
 	
Even though these variables have the same name, they are different because they are declared in different scopes.
	The variable player is created and assigned the value "x"
	
	The variable player is created again in the body of the function (different scope) and assigned the value that the user inputs. This variable is different than the player variable declared previously.

Output	**Explanation**
x o	x and o is printed for *player* and *computer*, respectively, regardless of what the user types.

The *global* keyword in Python can be used within the function to indicate that the variable being referenced or modified within that function refers to a global variable (defined at the module level), rather than the local variable with the same name.

Code	Explanation

`player = "x"`
`computer = "o"`
 	
`def start_game():`
`  global player`
`  player = input("Do you want to be 'x' or 'o'?")`
 	
`start_game()`
`print(player)`
`print(computer)`
 	
Indicates we want to use the player in the function *start_game*
	The variable player is created and assigned the value "x"
	
	The keyword *global* is used to indicate that we want to use the global version of *player* in the context of the *start_game* function.

Output	**Explanation**
o x	The correct values for x and o are printed because we have modified the global variables.

Skill 13.3 Exercise 1

Skill 13.4: Use the *return* statement in a function

Skill 13.4 Concepts

At this point, we have been using *print()* to help us visualize the output of our functions. Functions can also *return* a value to the program so that this value can be modified or used later. We use the Python keyword *return* to do this.

Here's an example of a program that will return a converted currency for a given location a user may want to visit.

Code

```
def calculate_exchange_usd(us_dollars, exchange_rate):
  return us_dollars * exchange_rate


new_zealand_exchange = calculate_exchange_usd(100, 1.4)


print("100 dollars in US currency would give you " + str(new_zealand_exchange) + " New Zealand dollars")
```

Output

```
100 dollars in US currency would give you 140 New Zealand dollars
```

Saving our values returned from a function like we did with *new_zealand_exchange* allows us to reuse the value (in the form of a variable) throughout the rest of the program.

When there is a result from a function that is stored in a variable, it is called a *returned function value*.

Skill 13.4 Exercises 1 and 2

Skill 13.5: Use the *return* statement to return multiple values

Skill 13.5 Concepts

Sometimes we may want to return more than one value from a function. We can return several values by separating them with a comma. Take a look at this example of a function that allows users in to check the upcoming week's weather (starting on Monday),

Code

```
weather_data = ['Sunny', 'Sunny', 'Cloudy', 'Raining', 'Snowing']


def threeday_weather_report(weather):
  first_day = " Tomorrow the weather will be " + weather[0]
  second_day = " The following day it will be " + weather[1]
  third_day = " Two days from now it will be " + weather[2]
  return first_day, second_day, third_day

weather_data = threeday_weather_report(weather_data)
print(weather_data)
```

Output

(' Tomorrow the weather will be Sunny', ' The following day it will be Sunny', ' Two days from now it will be Cloudy')

The function above returns a *tuple* datatype. Recall that a *tuple* is like a list except you cannot edit the values or the tuple itself by adding or removing items. Like a list, however, we can access the values of the tuple as follows,

Code

```
weather_data = threeday_weather_report(weather_data)
print(weather_data[0])
print(weather_data[1])
print(weather_data[2])
```

Output

Tomorrow the weather will be Sunny
The following day it will be Sunny
Two days from now it will be Cloudy

Skill 13.5 Exercise 1