Ironhack Student Portal

Lesson Goals

- Learn the difference between procedural and functional programming style.
- Learn about the components of functions and how to write them in Python.
- Learn the difference between global and local variables and how they apply to functions.
- Learn how to apply functions to Pandas data frames.

Introduction

Up until this point, we have been writing Python code in a procedural manner and using functions and methods that are either in Python's standard library or part of some other library such as Numpy or Pandas. This has worked fine for us thus far, but in order to truly graduate from beginner to intermediate Python programmer, you must have a basic understanding of functional programming and be able to write your own functions.

In this lesson, we will cover how to write Python code in a functional style and how to construct and call user-defined functions.

Procedural vs. Functional Programming

Procedural code is code that is written line by line and executed in the order it is written. It is the style in which most people learn to program due to its straightforward sequence of actions to solve a problem - do this, then do this, then do this.

More mature programmers frame problems a little differently. They take more of a functional approach to problem solving, where they create functions consisting of multiple lines of code that accomplish some task and often return a result that can be used by another function.

You can think of a function as a tool - something that helps you perform a job. In a sense, functional programming allows you to create your own tools. Some of these tools will be designed specifically for the current set of tasks you need to perform while other functions you write will be more general and able to be used for other projects. Imagine if a chef were able to construct their own set of knives or a carpenter were able to construct their own set of saws that were perfect for the jobs they needed to perform. That is the power that functional programming gives you as a programmer. Over time, you will develop a personal library of functions that allow you to get your work done more efficiently.

Writing Functions in Python

Functions in Python are defined by the `def` keyword followed by the function name and one or more inputs the function will need enclosed in parentheses. Within the function, you write code that performs the task you want the function to execute followed by a `return` statement if there is anything you would like the function to return. Below is an example of a `list2string` function that converts lists to strings.

```
def list2string(lst):
    string = ' '.join(lst)
    return string

to_string = ['John', 'was', 'a', 'man', 'of', 'many', 'talents']
list2string(to_string)
```

```
'John was a man of many talents'
```

In this example, we defined our `list2string` function and told it to expect one input. Within the function, we used the `join` method to convert the list to a string consisting of the list elements joined together by spaces. We then had the function return the string of space-separated elements.

You can include all the concepts we have covered in past lessons (loops, conditional logic, list comprehensions, etc.) inside a function.

Global vs. Local Variables

When writing functional code, it is important to keep track of which variables are global and which variables are local. Global variables are variables that maintain their value outside of functions. Local variables maintain their value within the function where they are defined and then cease to exist after we exit that function. In the example below, the variables `a` and `c` are global variables and variable `b` is a local variable that does not exist outside of the `multiply` function.

```
a = 9

def multiply(number, multiplier=2):
    b = number * multiplier
    return b

c = multiply(a)

print(a, c)
```

```
9 18
```

The `multiply` function above accepts a number and a multiplier (that defaults to 2), creates a local variable `b` that has a value of the number times the multiplier, and the function returns that value. We can see from the results of the example that global

variable `a` (9) did get multiplied, returned, and assigned to another global variable `c`, so we know that `b` did exist within the function and was returned. However, now that the function has completed its job, we receive an error when we try to print `b` because it does not exist in the global variable namespace.

```
print(b)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-59-c173745d9bfd> in <module>()
      8
      9 print(a, c)
---> 10 print(b)

NameError: name 'b' is not defined
```

We can enter any number into the function above and also override the default multiplier and our function will return the product of those two numbers.

```
multiply(100, multiplier=5)
```

```
500
```

Applying Functions to Data Frames

Now that we have a basic understanding of how functions and variables work, let's look at how we can apply a function that we have created to a Pandas data frame. For this example, we will import Pandas and the vehicles data set we have been working with in this module. We will then write a `get_means` function that identifies all the numeric columns in the data set and returns a data frame containing each column name and its mean.

```
import pandas as pd

data = pd.read_csv('./data sets/vehicles/vehicles.csv')

def get_means(df):
    numeric = df._get_numeric_data()
    means = pd.DataFrame(numeric.mean()).reset_index()
    means.columns = ['Column', 'Mean']
    return means

mean_df = get_means(data)
mean_df
```

	Column	Mean
0	Year	2000.716400
1	Engine Displacement	3.338493
2	Cylinders	5.765076
3	Fuel Barrels/Year	17.609056
4	City MPG	17.646139
5	Highway MPG	23.880646
6	Combined MPG	19.929322
7	CO2 Emission Grams/Mile	475.316339
8	Fuel Cost/Year	1892.598465

Note that we performed several steps with a single function call:

- Getting the numeric columns
- Calculating the means
- Resetting the indexes
- Setting the data frame's column names

Also note that the only input necessary for this function was a data frame. The function took care of everything else. That means that it should work for any data frame that you pass it, as long as the data frame has numeric columns. We have just created a tool by writing code once that we can use again and again.

Challenge: Import another data set that contains numeric variables and run the `get_means` function on it. It should return the means for the numeric columns in that data set.

Summary

In this lesson, we learned about functions in Python. We began the lesson by looking at the differences between writing procedural code and functional code. We then covered how to construct functions in Python and the differences between global and local variables. We ended the chapter by learning how functions can be applied to a Pandas data frame, computing and returning the means of all the numeric columns in a data frame. Now that you have learned about functions, you are well on your way to building a customized set of tools that can help you perform analytical work as efficiently as possible!