Kasia Rozanska

Nov. 24, 2020

Foundations of Programming: Python

Assignment06

<https://github.com/KasiaPaloma/IntroToProg-Python-Mod06>

Using Functions

# Introduction

In this demonstration we will convert the code we have been learning into functions. Once our programs become more complex it is good practice to break them up into smaller and more manageable chunks. We will also briefly go over classes and how both can be used to organize your code.

## Functions

A function is code that only runs when you call it. You first define it by using the keyword "def" and add however many statements you want. It will only execute when you "call the function" by using the function name followed by parenthesis (see example below).

# Define the function  
def FunctionName(Parameter01, Parameter02):  
 print("The parameters are:" + Parameter01 + "and" + Parameter02)  
  
# Call the function  
FunctionName(Argument01, Argument02)  
  
''' Output   
The parameters are Argument01 and Argument02  
'''

A parameter is a variable that the function uses as a guideline. It is the definition of the function. Arguments are the values that are passed into the function when calling it. The difference is that arguments are passed into the parameters.

You can also use a return value within a function so that the code can be separated more effectively. Return values create the results of a function within that function, so that you do not need to use arguments. Below is an example of how you would code the return value within a function.

# Define the function  
def AddValues(value1, value2):  
 fltAnswer = value1 + value2  
 return fltAnswer  
  
print(AddValues(1,2))  
  
''' Output will be  
3  
'''

## Global and Local Variables

Global variables are used in the entire script and local variables are contained within a function and cannot be accessed outside of that function. "Shadowing" is when you use global variables within functions by using the keyword "global." However, because Python is automated, if you are not careful and do not declare a variable within a function, it may use a global variable if you have reused the same naming convention.

## Classes

Functions group statements and classes group function, variables, and constants. They are incredibly flexible and have a lot of functionality, but we will get to those details in later demonstrations. For this demonstration we will build two classes, one for the functions that process the data, and one for the user input/outputs.

## Creating the Program

The full script can be found in my GitHub link but for the purpose of this demonstration we will only touch on a part of it.

Below is the code that was used in our previous demonstration to read data from a file and load it into a List. We will pull this apart and demonstrate how to create the same code but within a class and function.

# -- Data -- #  
# Declare variables and constants  
objFileName = "ToDoFile.txt" # An object that represents a file  
lstTable = [] # A list that acts as a 'table' of rows  
  
# -- Processing -- #  
objFile = open(objFileName, "r")  
for line in objFile:  
 strData = line.split(",")  
 dicRow = {"Task": strData[0].strip(), "Priority": strData[1].strip()}  
 lstTable.append(dicRow)  
objFile.close()  
  
# -- OutPut -- #  
print(lstTable)

To begin, using functions and classes allows for your code to not only look more professional, but it also allows for it to be reused in other programs. Just as above, the first thing you will want to script is the variables and constants that you will be using.

# Data ---------------------------------------------------------------------- #  
# Declare variables and constants  
strFileName = "ToDoFile.txt" # The name of the data file  
lstTable = [] # A list that acts as a 'table' of rows

Functions and classes allow for you to reuse the same code in many parts of the script without having to retype all the same logic. Below is the beginning of the class “Processor,” which begins with our function “read\_data\_from\_file.” We will only reference this function once in this demonstration, but once you look at the GitHub code you will see that I was able to reference it multiple times by simply typing, “Processor.read\_data\_from\_file().”

# Processing --------------------------------------------------------------- #  
class Processor:  
 *""" Performs Processing tasks """* @staticmethod  
 def read\_data\_from\_file(file\_name, list\_of\_rows):  
 *""" Reads data from a file into a list of dictionary rows  
  
 :param file\_name: (string) with name of file:  
 :param list\_of\_rows: (list) you want filled with file data:  
 :return: (list) of dictionary rows  
 """*

Before we add the logic to the function “read\_data\_from\_file” I wanted to quickly discuss docstrings. You probably noticed all the commented-out notes above and wondered about their specific formatting. This is known as docstring and is common practice to include at the beginning of a function. This docstring can be accessed by using **ctrl + q** in PyCharm and it will display the notes. This way you can access details regarding the parameters and returns without having to scroll back to the function itself.

Now for the actual logic of this function. Below you will notice that it is similar to our code from demo 5. The main difference here is that in the def line above we defined two parameters. Now when this code is used it can open any file, not just the file that was declared at the beginning. Go on to the next paragraph to see how.

list\_of\_rows.clear() # clear current data  
file = open(file\_name, "r")  
for line in file:  
 task, priority = line.split(",")  
 row = {"Task": task.strip(), "Priority": priority.strip()}  
 list\_of\_rows.append(row)  
file.close()  
return list\_of\_rows, 'Success'

Here the logic is that we call out the function “read\_data\_from\_file” and ask it to use the arguments “strFileName” and “lstTable” to run the code that reads the file data. It’s not obvious here but you can use other arguments further down with a new file without having to rewrite everything that is in the above snippet.

# Main Body of Script ------------------------------------------------------ #  
# When the program starts, Load data from ToDoFile.txt.  
Processor.read\_data\_from\_file(strFileName, lstTable) # read file data  
  
print(lstTable)

Try running both codes in your PyCharm instance. I have included the print statements so that you can see that both deliver the same results (see below).

[{'Task': 'Walk dog', 'Priority': 'high'}, {'Task': 'Pet dog', 'Priority': 'low'}]

[{'Task': 'Walk dog', 'Priority': 'high'}, {'Task': 'Pet dog', 'Priority': 'low'}]

# Summary

Go to the GitHub page that is at the top of this document to work through the full script yourself. Compare it to our last demonstration and you will see how beneficial it is to organize our code with function and classes.