Jeff Wichmann

May 25, 2020

IT FDN 100 A Sp 20: Foundations of Programming: Python

Assignment 6

<https://github.com/jwichmann51/-IntroToProg-Python-Mod06>

Python: Add Functions in To Do File

# Introduction

This paper outlines the steps and process to modify a To Do List programming script in Python whereby functions are added. A starting program template was provided by Randal Root (Root R., Assignment06 IT FDN 100 A Sp 20: Foundations of Programming: Python, University of Washington, 2020) and included prompts where code needs to be added. The assignment calls for taking code from Assignment05 and building functions. My goal is to create a document that a Python user can utilize to build functions as necessary to complete this coding exercise. Additionally, I can use this as a reference to refresh my memory in the future, if necessary. A subsequent draft of this assignment will be completed with the corrected script. I am using Mac OS.

# Functions

Functions are used to break up large and complex programs by grouping a block of code into a single programming statement. The entire block of code can be run by calling the single function. Functions are also good because if there is typo in block of code within a function, the typo only needs to be fixed once. Alternatively if functions are not being used, each individual instance of the code would need to be fixed. Figure 1 shows two example functions inspired by Randal Root (Root R., Assignment06 IT FDN 100 A Sp 20: Foundations of Programming: Python, University of Washington, 2020).

A screenshot of a cell phone

Description automatically generated

**Figure 1: Example Python code inspired by Randal Root**

To call function01, simply write the name function01(). To call function02, simply write the name function02(). Figure 2 shows the output of booth functions being run in PyCharm.

A screenshot of a cell phone

Description automatically generated

**Figure 2: Output in PyCharm**

# Function With Parameters

Functions can also have parameters. Parameters “allow you to pass values into the function for processing” (Root R., Assignment06 IT FDN 100 A Sp 20: Foundations of Programming: Python, University of Washington, 2020). Values included in parameters are called *arguments* but these are also referred to as *parameters*. There is no limit how many parameters can be included. An example function with parameters is shows in figure 3.

A screenshot of a cell phone

Description automatically generated

**Figure 3: Example Python code used in Randal Root’s LAB 6-1**

Figure 4 shows the output of the above function with parameters.

A screenshot of a cell phone

Description automatically generated

**Figure 4: Output in PyCharm**

# Class

A class is a named group of functions. Figure 5 shows two functions within a single class and one of the two functions being run.

A screenshot of a cell phone

Description automatically generated

**Figure 5: Example Python code created by Randal Root**

Figure 6 shows the output after function04 is run.

A screenshot of a cell phone

Description automatically generated

**Figure 6: Output in PyCharm**

# The assignment – saving a to do list

The script assignment for Module 6 is to modify the file Assigment06\_Starter.py where prompted. The prompt was marked in the program # TODO: Add Code Here. The assignment involves pulling data from Assignment05 programming script and dropping it into the right location in the Assigment06\_Starter.py file. The code also needed to be sorted between the processing and presentation portions of the programming script (Root R., Assignment06 IT FDN 100 A Sp 20: Foundations of Programming: Python, University of Washington, 2020). Processing and presentation are two of the three layers of concern. The third is data. This script follows in figure 7.

# ---------------------------------------------------------------------------- #  
# Title: Assignment 06  
# Description: Working with functions in a class,  
# When the program starts, load each "row" of data  
# in "ToDoList.txt" into a python Dictionary.  
# Add the each dictionary "row" to a python list "table"  
# ChangeLog (Who,When,What):  
# RRoot,1.1.2030,Created started script  
# RRoot,1.1.2030,Added code to complete assignment 5  
# JWichmann, 5/22/2020, Reviewed code ahead of the Friday evening Zoom with Dave  
# JWichmann, 5/24/2020, Modified code to complete assignment  
# JWichmann, 5/25/2020, Final code review before submission  
# ---------------------------------------------------------------------------- #  
  
# Data ---------------------------------------------------------------------- #  
# Declare variables and constants  
strFileName = "ToDoFile.txt" # The name of the data file  
objFile = None # An object that represents a file  
dicRow = {} # A row of data separated into elements of a dictionary {Task,Priority}  
lstTable = [] # A list that acts as a 'table' of rows  
strChoice = "" # Captures the user option selection  
strTask = "" # Captures the user task data  
strPriority = "" # Captures the user priority data  
strStatus = "" # Captures the status of an processing functions  
  
# Processing --------------------------------------------------------------- #  
class Processor:  
 *""" Performs Processing tasks """* @staticmethod  
 def read\_data\_from\_file(file\_name, list\_of\_rows):  
 # This is a function with two parameters:  
 # i) file\_name (strFileName which is ToDoFile.txt)  
 # and  
 # ii) list\_of\_rows (lstTable)  
 # This function is activated in Step 1 below via the code Processor.read\_data\_from\_file(strFileName, lstTable)  
 *""" 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  
 """* 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'  
  
 @staticmethod  
 def add\_data\_to\_list(strTask, strPriority, list\_of\_rows):  
 # This is a function with three parameters:  
 # i) strTask which is the task inputted by the user  
 # ii) strPriority which is the priority inputted by the user  
 # iii) list\_of\_rows (lstTable)  
 # This is the part of the code that does the processing. The Presentation section below is where the user enters the data.  
 # This function is activated by Processor.add\_data\_to\_list(strTask, strPriority, lstTable)  
 # *TODO: Add Code Here! ++2) Add a new item ++ DONE* strTask = str(input(" Enter a Task: "))  
 strPriority = str(input(" Enter the Priority: "))  
 dicRow = {"Task": strTask, "Priority": strPriority}  
 lstTable.append(dicRow)  
 #lstTable.append({"Task": strTask, "Priority": strPriority})  
 return list\_of\_rows, 'Success'  
  
 @staticmethod  
 def remove\_data\_from\_list(strTask, list\_of\_rows):  
 # This is a function with two parameters:  
 # i) task (strTask)  
 # ii) list\_of\_rows (lstTable)  
 # This code is run with the command Processor.remove\_data\_from\_list(strTask, lstTable)  
 # *TODO: Add Code Here! ++3) Remove an existing item++ DONE* for row in lstTable:  
 if row["Task"].lower() == strTask.lower():  
 lstTable.remove(row)  
 print("row removed")  
 print(lstTable, '<< List with Dictionary objects')  
 else:  
 print("row not found")  
 print(lstTable, '<< List with Dictionary objects')  
 return list\_of\_rows, 'Success'  
  
 @staticmethod  
 def write\_data\_to\_file(file\_name, list\_of\_rows):  
 # This code will be called with the following code below Processor.write\_data\_to\_file(strFileName, lstTable)  
 # *TODO: Add Code Here! ++4) Save data to file ++ DONE* objFile = open("ToDoFile.txt", "w")  
 for row in lstTable:  
 objFile.write(str(row["Task"]) + ',' + str(row["Priority"] + '\n'))  
 objFile.close()  
 return list\_of\_rows, 'Success'  
  
# Presentation (Input/Output) -------------------------------------------- #  
class IO:  
 *""" Performs Input and Output tasks """* @staticmethod  
 def print\_menu\_Tasks():  
 *""" Display a menu of choices to the user  
  
 :return: nothing  
 """* print('''  
 Menu of Options  
 1) Add a new Task  
 2) Remove an existing Task  
 3) Save Data to File   
 4) Reload Data from File  
 5) Exit Program  
 ''')  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def input\_menu\_choice():  
 *""" Gets the menu choice from a user  
  
 :return: string  
 """* choice = str(input("Which option would you like to perform? [1 to 5] - ")).strip()  
 print() # Add an extra line for looks  
 return choice  
  
 @staticmethod  
 def print\_current\_Tasks\_in\_list(list\_of\_rows):  
 # This is a function with one parameter:  
 # i) list\_of\_rows (lstTable)  
 # This function is activated by Processor.print\_current\_Tasks\_in\_list(lstTable)  
 *""" Shows the current Tasks in the list of dictionaries rows  
  
 :param list\_of\_rows: (list) of rows you want to display  
 :return: nothing  
 """* print("\*\*\*\*\*\*\* The current Tasks ToDo are: \*\*\*\*\*\*\*")  
 for row in list\_of\_rows:  
 print(row["Task"] + " (" + row["Priority"] + ")")  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def input\_yes\_no\_choice(message):  
 *""" Gets a yes or no choice from the user  
  
 :return: string  
 """* return str(input(message)).strip().lower()  
  
 @staticmethod  
 def input\_press\_to\_continue(optional\_message=''):  
 *""" Pause program and show a message before continuing  
  
 :param optional\_message: An optional message you want to display  
 :return: nothing  
 """* print(optional\_message)  
 input('Press the [Enter] key to continue.')  
  
 @staticmethod  
 def input\_new\_task\_and\_priority():  
 pass # *TODO: Add Code Here! DONE* print(" Type in a Task and Priority")  
 # strTask = str(input(" Enter a Task: "))  
 # strPriority = str(input(" Enter the Priority: "))  
 # return task, priority  
  
 @staticmethod  
 def input\_task\_to\_remove():  
 pass # *TODO: Add Code Here! DONE* strTask = input("Task to Remove: ")  
 # return task  
  
# Main Body of Script ------------------------------------------------------ #  
  
# Step 1 - When the program starts, Load data from ToDoFile.txt.  
Processor.read\_data\_from\_file(strFileName, lstTable) # read file data  
  
# Step 2 - Display a menu of choices to the user  
while(True):  
 # Step 3 Show current data  
 IO.print\_current\_Tasks\_in\_list(lstTable) # Show current data in the list/table  
 IO.print\_menu\_Tasks() # Shows menu  
 strChoice = IO.input\_menu\_choice() # Get menu option  
   
 # Step 4 - Process user's menu choice  
 if strChoice.strip() == '1': # Add a new Task  
 # *TODO: Add Code Here DONE* IO.input\_new\_task\_and\_priority()  
 Processor.add\_data\_to\_list(strTask, strPriority, lstTable)   
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu  
  
 elif strChoice == '2': # Remove an existing Task  
 # *TODO: Add Code Here DONE* IO.input\_task\_to\_remove()  
 Processor.remove\_data\_from\_list(strTask, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu  
  
 elif strChoice == '3': # Save Data to File  
 strChoice = IO.input\_yes\_no\_choice("Save this data to file? (y/n) - ")  
 if strChoice.lower() == "y":  
 # *TODO: Add Code Here! DONE* Processor.write\_data\_to\_file(strFileName, lstTable)  
 print(" Data was saved!")  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue("Save Cancelled!")  
 continue # to show the menu  
  
 elif strChoice == '4': # Reload Data from File  
 print("Warning: Unsaved Data Will Be Lost!")  
 strChoice = IO.input\_yes\_no\_choice("Are you sure you want to reload data from file? (y/n) - ")  
 if strChoice.lower() == 'y':  
 # *TODO: Add Code Here! DONE* IO.print\_current\_Tasks\_in\_list(lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue("File Reload Cancelled!")  
 continue # to show the menu  
  
 elif strChoice == '5': # Exit Program  
 print("Goodbye!")  
 break # and Exit

**Figure 7: Example Python Script**

# Images showing the script working

Since the program is so long, the functional code is broken into multiple pieces. Note that I start with the ToDoFile.txt shown in figure 18.

A screenshot of a cell phone

Description automatically generated

**Figure 8: Example text file**

# Show current data

The image below shows current data being displayed (Figure 9). This data was read from ToDoFile.txt file when the program began.

# A screenshot of a cell phone Description automatically generated

**Figure 9: Output in PyCharm**

# 1) Add a new task

Figure 10 shows an item being added (code; high) and the new list.

A screenshot of a cell phone

Description automatically generated

**Figure 10: Output in PyCharm**

# 2) Remove an existing item

Figure 11 shows that I was not able to get the code to remove the item “paint”.

A screenshot of a cell phone

Description automatically generated

**Figure 11: Output in PyCharm**

# 3) Save Data to File

Figure 12 shows the output in PyCharm and the new text file (Figure 13).

A screenshot of a cell phone

Description automatically generated

**Figure 12: Output in PyCharm**

The newly saved ToDoFile.txt file (Figure 13) which includes the new item (code; high).

A screenshot of a cell phone

Description automatically generated

**Figure 13: Example text file**

# 4) Reload data from file

Figure 14 shows the output in PyCharm when the data is reloaded from the file.

A screenshot of a cell phone

Description automatically generated

**Figure 14: Output in PyCharm**

# 5) Exit Program

As show in figure 15, the program exited properly.

A screenshot of a cell phone

Description automatically generated

**Figure 15: Output in PyCharm**

# Running the script in OS command

Figure 16 shows the first third of the script running in OS command.

A screenshot of a social media post

Description automatically generated

**Figure16: Output in OS command**

Figure 17 shows the second third of the script running in OS command.

**A screenshot of a cell phone

Description automatically generated**

**Figure 17: Output in OS command**

Figure 18 shows the last third of the script running in OS command.

**A picture containing bird

Description automatically generated**

**Figure 18: Output in OS command**

# Summary

In summary, this document outlines the steps and process to create a programming script. The goal was to modify a starting template to build function. I used the code that I wrote Assignment05. The document also explained functions, functions with parameters, and classes. This document acts as guide to a Python user or for me, if I need a refresher at some point in the future.