Erickson Smith
November 21, 2020
IT FDN 110 A Au 20: Foundations of Programming: Python
Assignment 06

To Do List Script

Introduction

In this week's assignment we were introduced to a new and very powerful function. Like the other Python functions len() and range() Python allows you to create custom functions. I will cover my thought process and steps I took to complete this week's assignment.

Staring the code

This week we were given a partially completed script to import and finish. That script was similar to last week's script but cleaned up by functions. In order to start the code, I needed to look at all the pieces and figure out what was missing and what needed fixing. Before I could do that I needed to look at my established variables in the Data section of the Separation of Concerns pattern. (figure 01.)

```
# 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
```

Figure 01: Declared variable prior to code body.

Declaring your variables before you code body is a good habit to get into. This process always future coders to look at your code and see what your intentions were. Another good habit to get into is updating the change log on a program. This tells future programmers as well as yourself who, when, and what was done to the code.(figure 02.)

```
# ------------------------------------------------------------------------------ #
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.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
# ESmith, 11.21.2020,Modified code to complete assignment 6
# ------------------------------------------------------------------------------ #
```

Figure 02: Change Log

As you can see. The code was established and a description of what the code is for was added in. Under that the change log begins this will give you a clear road map of where the code came from and what has been done to the code along the way.

Building the Code

With the Data section of the “Separation of Concerns” out of the way. Next comes the Processing section. In this section we establish our Class and Functions to be used later in the code. The Class is a means of bundling Data and Functions together in a nice and neat area for programmers to read. (figure 03.)

```
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
        """
        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'
```

Figure 03: Defining the Class.

Within the Class are any number of Functions that you will need throughout your code. A Function is a block of code that runs only when called on. You can pass Data, known as Parameters, into a Function. As you can see the first Function was completed for us. The next few were for us to complete. The first was a function titled add_data_to_list(figure 04.)

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    # TODO: Add Code Here! #
    new_task = input('Enter new Task: ')
    new_priority = input('What Priority level? (High, Med, Low): ')
    dicRow = {'Task': new_task, 'Priority': new_priority}
    list_of_rows.append(dicRow)
    return list_of_rows, 'Success'
```

Figure 04: A Functions that adds a Dictionary to a List.

This Function takes in three parameters. Task, Priority, and list_of_rows; these parameters capture values sent to the Function from a Function call through its arguments. The Function will ask the user to input two things. A Task and that task's

Priority, then the function will take those two values and put them in a Dictionary key,value pair and append them to our list table of key, value pairs. The next Function is remove_data_from_list(figure 05.)

```
@staticmethod
def remove_data_from_list(task, list_of_rows):
    # TODO: Add Code Here! #
    strTask = input('Which Task would you like to remove? ')
    for row in list_of_rows:
        if row['Task'] == strTask:
            list_of_rows.remove(row)
            print('Task removed')
            print(list_of_rows)
    return list_of_rows, 'Success'
```

Figure 05: A Function that removes key, value pairs from a list.

This function does the opposite of the previous one. It removes a task from the list. With the use of a for loop it will iterate through the whole list looking for the task that the user input. Once the loop finds the task it will remove it and the priority from the list. Then it will print out the message ‘Task Removed’ just to be sure. For the final Function in the Processing section we needed to finish the Function write_data_to_file. (figure 06.)

```
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    # TODO: Add Code Here! #
    objFile = open(strFileName, "w")
    for row in list_of_rows:
        objFile.write(row["Task"] + "," + row["Priority"] + "\n")
    objFile.close()
    print('Saved')
    return list_of_rows, 'Success'
```

Figure 06: A Function that writes data to a text file.

When you’ve added and removed all the Data you had you need to store it somewhere. This Function allows you to save the data from memory to the hard drive on a text file. It will open the file, format your data into a nice csv(comma separated value) and then close the file.

The next section of the “Separation of Concerns” is the Presentation or Input/Output section. This is where the user interacts with the code. Such as menus in programs (figure 07.)

```
# 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
```

Figure 07: Menu of options in the Presentation (Input/Output) section.

As you can see in this section there is another class and a whole new set of Functions to set and call from. With this function you are directly speaking with the code. (figure 08.)

```
@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
```

Figure 08: A function that asks for the users choice of menu options.

Choosing 1-5 the user can pick which option they want to use. The next function shows you the key, value pairs you have added so far. (figure 09.)

```python
@staticmethod
def print_current_Tasks_in_list(list_of_rows):
    """ 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
```

Figure 09: A Function that shows tasks in the list of dictionaries.

As with many things you may mess up on an input or change your mind about adding something that's where this next function comes into play. Simply put this function when called will ask if you really want to do something. (figure 10.)

```python
@staticmethod
def input_yes_no_choice(message):
    """ Gets a yes or no choice from the user

    :return: string
    """
    return str(input(message)).strip().lower()
```

Figure 10: A function that makes sure you want to do something.

The next function is also important because when called it stops the program in its tracks till the user wants to continue. (figure 11.)

```python
@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.')
```

Figure 11: A Function that pauses the code till a key is pressed.

The Main Code Body

Now that all the functions have been established we can write our main code body. First we need to open the file and read it in so that we can see what is in there. Next we need to display that beautiful menu we made earlier. (figure 12.)

```
# 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
```

Figure 12: Calling Functions to open the file and print the menu.

Starting with the while loop to keep the program running till the user is finished. The functions IO.print_current_Tasks_in_list(lstTable) and IO.print_menu_Tasks() are called This may look confusing but is rather simple. The (IO) part is the Class we established earlier in the code. In that class are the functions print_current_Tasks_in_list and print_menu_Tasks which print out whatever data the user has input and the menu of options. After that we move on to option 1 of the menu. (figure 13.)

```
# Step 4 - Process user's menu choice
if strChoice.strip() == '1':  # Add a new Task
    # TODO: Add Code Here
    Processor.add_data_to_list(strTask, strPriority, lstTable)
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

Figure 13: Calling Functions to add a dictionary to list.

With the user inputting option 1 the code will call the Class/Function Processor.add_data_to_list and IO.input_press_to_continue. Again Processor is the class that the function add_data_to_list is in. After option 1 is option 2. (figure 14.)

```
elif strChoice == '2':  # Remove an existing Task
    # TODO: Add Code Here
    Processor.remove_data_from_list(strTask, lstTable)
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

Figure 14: Calling a Function to remove data from a list.

Option 2 allows the user to remove data from the file by calling the Class Processor and the Function remove_data_from_list(strTask, lstTable) this function takes in two parameters first being strTask which was a variable established at the beginning of the code. lstTable is a variable as well that acts as a table for our list of dictionaries. The next option is option 3 (figure 15.)

```
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!
        Processor.write_data_to_file(strFileName, lstTable)
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("Save Cancelled!")
    continue  # to show the menu
```

Figure 15: Calling a function to write data to a text file.

One of the more complex looking sections but is rather straightforward. This section asks the user if they want to save their data with a ‘y’ or ‘n’ answer. Then the data is written to a file using the function established earlier write_data_to_file. With all that data saved and put away now you need to load it up and read it. That’s where the next section comes in. (figure 16.)

```
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!
        Processor.read_data_from_file(strFileName, lstTable)
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("File Reload  Cancelled!")
    continue  # to show the menu
```

Figure 16: Calling a function to read data from a file.

This section will call the function to read_data_from_file. This allows the code to load up all the data we had just saved into the file to memory.

Summary

Functions are an amazing tool that can make your programming life alot easier if used properly. This assignment taught me that by careful thinking and execution what you

write today can be used later on in the future. Functions and classes that are written so well could be imported and exported to many files and work perfectly.