

Tony Le

11/12/2019

IT FDN 100

Assignment 06

# Functions and classes

## Introduction

The assignment for this week is to build off of the program in Assignment 05 that displays a menu to the user that allows them to create a To-Do list. This user has options to view the data, save the data, add an item, remove an item or exit the program. The goal of this assignment is to improve what we've done before by converting our code to make use of classes and functions. We were given a starter file and use it to accomplish this task.

## Declare Variables and Constants

The script in Figure 1 begins by declaring the variables and constants to be used at a later point.

```
strFileName = "ToDoFile.txt" # The name of the data file
objFile = None # An object that represents a file
strData = "" # A row of text data from the file
dicRow = {} # A row of data separated into elements of a dictionary {Task,Priority}
lstTable = [] # A dictionary that acts as a 'table' of rows
strChoice = "" # Capture the user option selection
strTask = ""
strPriority = ""
```

Figure 1: Declaring variables

## Declaring a class

Figure 2 establishes the class that our processing functions will be placed in. We call this class **FileProcessor**.

```
class FileProcessor:
```

Figure 2: Declaring a class

## Processing functions within FileProcessor class

Within the **FileProcessor** class, we now have functions that act as the processors for this code. When called upon later in the script, these functions are executed and then return back to where they were

originally called from. The first function we establish is the **ReadFileDataToList( )** function in Figure 3. This function reads the text file **ToDoFile.txt** and reads the data within the file and loads it into a python dictionary **dicRow**. We begin by opening the file and reading the data into the memory. We follow by iterating through each entry in the txt file and extract with [0] and [1] entries to be inserted into a dictionary with keys **Task** and **Priority**. This dictionary row is then appended onto our list table.

```
@staticmethod
def ReadFileDataToList(file_name, list_of_rows):
    """
    Desc - 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
    """
    file = open(file_name, "r")
    for line in file:
        data = line.split(",")
        row = {"Task": data[0].strip(), "Priority": data[1].strip()}
        list_of_rows.append(row)
    file.close()
    return list_of_rows
```

Figure 3: Read file data function

Figure 4 shows the function that writes the data logged into the list into a text file. It uses a **for loop** to record every dictionary row for each iteration in the list.

```
@staticmethod
def WriteListDataToFile(file_name, list_of_rows):
    """
    Desc - Writes 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: nothing
    """
    pass, # TODO: Add code Here (---done---)

    objFile = open(file_name, "w")
    for dicRow in list_of_rows: # Write each row of data to the file
        objFile.write(dicRow["Task"] + "," + dicRow["Priority"] + "\n")
    objFile.close()
```

Figure 4: Write data to file function

Figure 5 shows the function that follows after a user inputs new data in the table. It takes the data, adds it to **dicRow** and **appends** it as a row to the list.

```
@staticmethod
def AddRowToList(task, priority, list_of_rows):
    """
    Desc - Reads data from a file into a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority level:
    :param list_of_rows: (list) you want filled with file data:
    :return: nothing
    """
    dicRow = {"Task": task, "Priority": priority} # Create a new dictionary row
    list_of_rows.append(dicRow) # Add the new row to the list/table
```

Figure 5: Add row to list

The last function in the processing class is shown in Figure 6. This one removes a row from the list upon request by the user. It searches the list for a match with the user input and if true, the row is then removed.

```
@staticmethod
def RemoveRow():
    intRowNumber = 0 # Create a counter to identify the current dictionary row in the loop

    # Step 3.3.b - Search though the table or rows for a match to the user's input
    while intRowNumber < len(lstTable):
        if strKeyToRemove == str(list(dict(lstTable[intRowNumber]).values())[0]): # Search current row column 0
            del lstTable[intRowNumber] # Delete the row if a match is found
            blnItemRemoved = True # Set the flag so the loop stops
            intRowNumber += 1 # Increase counter to get next row

    # Step 3.3.c - Update user on the status of the search
    if blnItemRemoved == True:
        print("The task was removed.")
    else:
        print("I'm sorry, but I could not find that task.")
    print() # Add an extra line for looks
```

Figure 6: Add row to list

## Input/output with Class IO

The first function to include in the input/output class is to create a menu that allows to user to control what the program does as shown in Figure 7. This function created is **OutputMenuItems( )**. We create

this to always call back to it after the user completes an action. The next function, **InputMenuChoice ( )** registers an input from the user to determine which action they'd like to perform from the menu. It returns the choice variable which is used later on in the script to determine which if statement to execute.

```
@staticmethod
def OutputMenuItems():
    """ Display a menu of choices to the user
    :return: nothing
    """
    print('''
    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Reload Data from File
    6) Exit Program
    ''')
    print() # Add an extra line for looks

@staticmethod
def InputMenuChoice():
    """ Gets the menu choice from a user
    :return: string
    """
    choice = str(input("Which option would you like to perform? [1 to 6] - ")).strip()
    print() # Add an extra line for looks
    return choice
```

**Figure 7: Menu script**

The next function in Figure 8 shows the user what is currently in the list. This function is used when the user would like to reference what they've already inputted into the list. It does this with the use of a **for** loop that iterates and prints out every row in the list table.

```

@staticmethod
def ShowCurrentItemsInList(list_of_rows):
    """ Shows the current items in the list of dictionaries rows

    :param list_of_rows: (list) of rows you want to display
    :return: nothing
    """

    print("***** The current items ToDo are: *****")
    for row in list_of_rows:
        print(row["Task"] + " (" + row["Priority"] + ")")
    print("*****")
    print() # Add an extra line for looks

```

Figure 8: Menu script

The last function of the IO class in Figure 9 allows the user to enter in the tasks and priorities that they would like to add to the list. We make use of the **global** command here as the **strTask** and **strPriority** variables are needed for use outside of the function script.

```

@staticmethod
def userTaskInput():
    """ Registers user input for a task and priority

    :return: nothing
    """

    global strTask
    global strPriority
    strTask = str(input("What is the task? - ")).strip() # Get task from user
    strPriority = str(input("What is the priority? [high|low] - ")).strip() # Get priority from user
    print() # Add an extra line for looks

```

Figure 9: Menu script

## Main body of the script

Now we get into the main body of the script in Figure 10 where we use all of the classes and functions that were established prior to this section of the script. The first function that is called on is **FileProcessor.ReadFileDataToList( )**. This loads data into the table that is already existing in the text file. The next function is the display the menu with **IO.OutputMenuItems( )**. The next step is to take in user input for the action to be executed and assign it to the variable **strChoice**. We use the **strChoice** variable to execute the following **if** statement. This **if** statement executes depending on the input of the user. If the user inputs "1" then we show the current items in the list table with function **IO.ShowCurrentItemsInList ( )**. With an input of "2" the user inputs a task and priority and those are inserted into the list as a dictionary row. The inputs are registered under **IO.userTaskInput( )** and processed under **FileProcessor.AddRowToList( )**. After each of these functions, the current list table is displayed. With a user input of "3," the script asks for an input of the task to be removed. That input is

then searched for in the function **FileProcessor.RemoveRow( )** and the corresponding row is removed. If the input is “4,” then it will prompt to user to determine whether they would like to save the code or not. If they do, the **FileProcessor.WriteDataToListFile( )** function is called upon and the list is saved to the **ToDoFile.txt** file otherwise the data is not saved through the use of an **else** command. An input of “5,” asks the user to reload the data. If the user decides to move on, the table is cleared and the newly cleared table is displayed on the screen. Finally, an input of “6” breaks the code and exits from there.

```
# Step 1 - When the program starts, Load data from ToDoFile.txt.
FileProcessor.ReadFileDataToList(strFileName, lstTable) # read file data

# Step 2 - Display a menu of choices to the user
while True:
    IO.OutputMenuItems() # Shows menu
    strChoice = IO.InputMenuChoice() # Get menu option

    # Step 3 - Process user's menu choice
    # Step 3.1 Show current data
    if (strChoice.strip() == '1'):
        IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table
        continue # to show the menu

    # Step 3.2 - Add a new item to the List/Table
    elif strChoice.strip() == '2':

        # Step 3.2.a - Ask user for new task and priority
        # ToDo: Place IO code in a new function (---done---)
        IO.userTaskInput()
        IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table

        # Step 3.2.b Add item to the List/Table
        # ToDo: Place processing code in a new function (---done---)
        FileProcessor.AddRowToList(strTask, strPriority, lstTable)
        IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table
        continue # to show the menu

    # Step 3.3 - Remove a new item to the List/Table
    elif strChoice == '3':

        # Step 3.3.a - Ask user for item and prepare searching while Loop
        strKeyToRemove = input("Which TASK would you like removed? - ") # get task user wants deleted
        blnItemRemoved = False # Create a boolean Flag for Loop

        # ToDo: Place processing code in a new function (---done---)
        FileProcessor.RemoveRow()

        #Step 3.3.d - Show the current items in the table
        IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table
        continue # to show the menu

    # Step 3.4 - Save tasks to the ToDoFile.txt file
    elif strChoice == '4':

        #Step 3.4.a - Show the current items in the table
        IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table

        #Step 3.4.b - Ask if user if they want save that data
        if "y" == str(input("Save this data to file? (y/n) - ")).strip().lower(): # Double-check with user

            FileProcessor.WriteListDataToFile(strFileName, lstTable)

            input("Data saved to file! Press the [Enter] key to return to menu.")
        else: # Let the user know the data was not saved
            input("New data was NOT Saved, but previous data still exists! Press the [Enter] key to return to menu.")
            continue # to show the menu

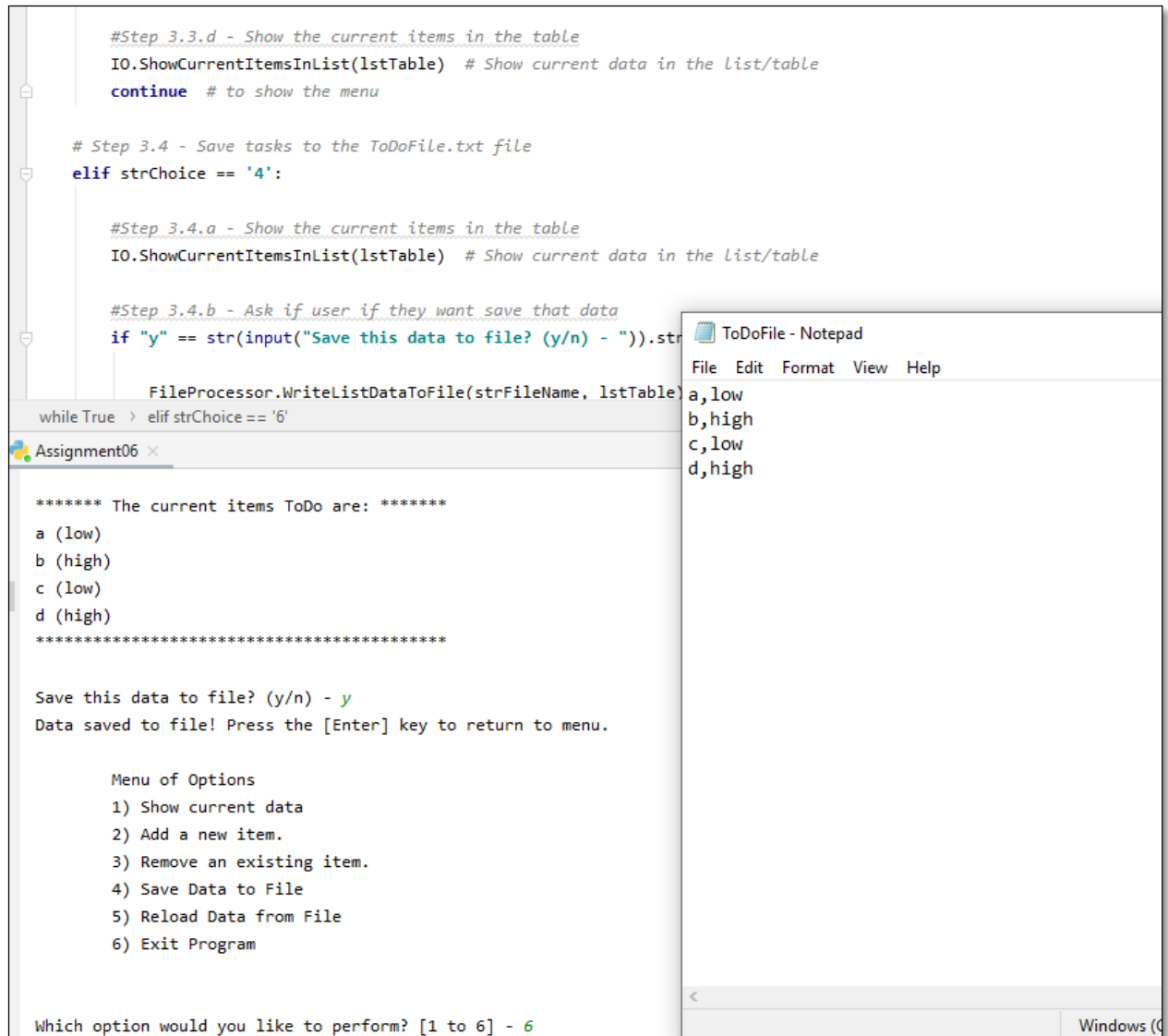
    # Step 3.5 - Reload data from the ToDoFile.txt file (clears the current data from the List/table)
    elif strChoice == '5':
        print("Warning: This will replace all unsaved changes. Data loss may occur!") # Warn user of data loss
        strYesOrNo = input("Reload file data without saving? [y/n] - ") # Double-check with user
        if strYesOrNo.lower() == 'y':
            lstTable.clear() # Added to fix bug 1.1.2030
            FileProcessor.ReadFileDataToList(strFileName, lstTable) # Replace the current list data with file data
            IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table
        else:
            input("File data was NOT reloaded! Press the [Enter] key to return to menu.")
            IO.ShowCurrentItemsInList(lstTable) # Show current data in the List/table
            continue # to show the menu

    # Step 3.6 - Exit the program
    elif strChoice == '6':
        break # and Exit
```

Figure 10: Option 1 showing current data

## Summary

Figure 11 shows the output of the script and shows that it is clearly functioning as intended with the results of the data saved in a `ToDoList.txt` file created as a result of this script. The intent of this script was to create an interactive menu and allow the user to create a to-do list of task and their respective priority values. This assignment is a build off of Assignment 05 where we are essentially completing the same goal but utilizing classes and functions while doing so. They greatly improves the organization and clarity of the script and is an essential skill for us to have moving forward in our coding careers.



```
#Step 3.3.d - Show the current items in the table
IO.ShowCurrentItemsInList(lstTable) # Show current data in the list/table
continue # to show the menu

# Step 3.4 - Save tasks to the ToDoFile.txt file
elif strChoice == '4':

    #Step 3.4.a - Show the current items in the table
    IO.ShowCurrentItemsInList(lstTable) # Show current data in the list/table

    #Step 3.4.b - Ask if user if they want save that data
    if "y" == str(input("Save this data to file? (y/n) - ")).strip():

        FileProcessor.WriteListDataToFile(strFileName, lstTable)

while True:
    elif strChoice == '6':
```

Assignment06

```
***** The current items ToDo are: *****
a (low)
b (high)
c (low)
d (high)
*****

Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

Which option would you like to perform? [1 to 6] - 6
```

ToDoFile - Notepad

```
File Edit Format View Help
a,low
b,high
c,low
d,high
```

Windows (C

Figure 11: Print out of full name entered by the user