Kerry Bosworth  
Nov 13, 2019  
Assignment06

# Using Functions and Classes

**Introduction**

This week we started using functions and classes (a collection of functions) in our assignment. In the next sections I will use the document this week to walk through the starter file and explain how I changed the code to complete the assignment. Using breakpoints and the debugger in PyCharm was very helpful.

**Programming Assignment**

First, does the program run at all? Yes, it does. The first function in the FileProcesser class is already written. When the program runs as-is it reads the file into a list. The ReadFileDataToList function passes two arguments, the location of the file and the reference location of a blank list. These are accepted via the corresponding parameters of the function. The items are added to the list one at a time. It returns the reference location of the list that has been updated.

When the user selects menu option 1, it calls the function ShowCurrentItemsInList in the IO class. It passes the list as the argument and the parameter reviews the argument and prints out the values line by line.   
  
**Menu Option 2**

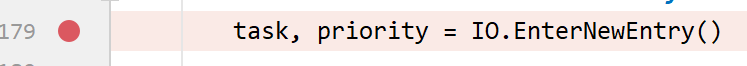
The program continues to work as I move onto menu option 2 to add an item to the list, however I encounter my first ***to do***. I need to move the code that asks the user input items as well as the code that add the items to the list into their own respective functions. Then it re-runs the ShowCurrentItemsInList to show the updated list.

I created a new function called EnterNewEntry in the IO class.

@staticmethod  
**def** EnterNewEntry():  
 *"""  
 Desc - Receives new task and priority from user* **:return***: strings  
 """* 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* **return** strTask,strPriority

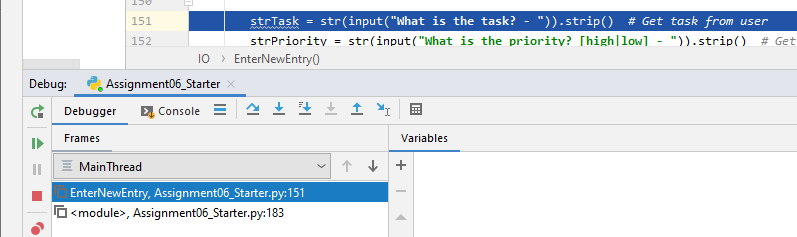
There are no arguments to pass with it, but I set it up to receive some return values gathered in the function.

task, priority = IO.EnterNewEntry()

I used a breakpoint (Figure 1) and the next step was to go up to the function to start getting the data. To start the file just has one task, sleep with a priority of high.  
  


**Figure 1:** Set breakpoint.

After that I use the step into buttons to continue with the code.



**Figure 2:** Manual progression

As it progresses, I ensure it returns the expected values (Figure 3). There was trial and error during this process. I added a task, eat with a priority of high.

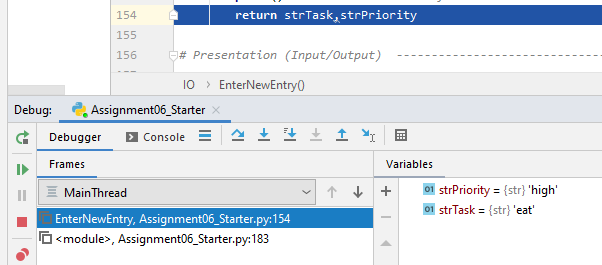


Figure 3: Return values.

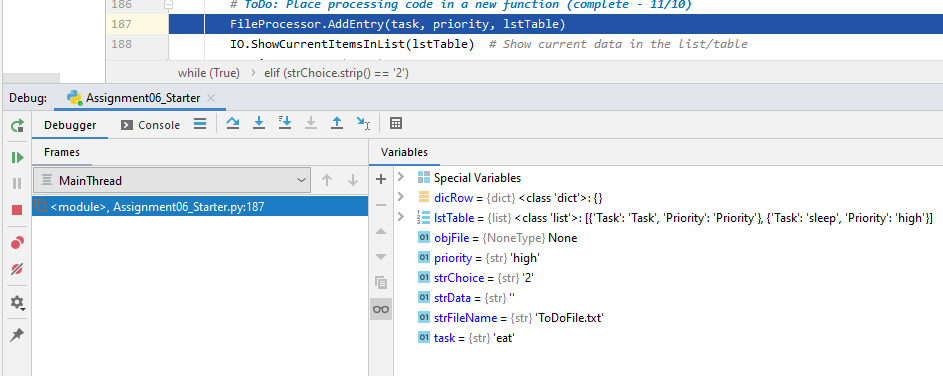
Then I used those variables to send as arguments along with the list to a function called AddEntry in the FileProcessor class.

FileProcessor.AddEntry(task, priority, lstTable)

Then created this function:

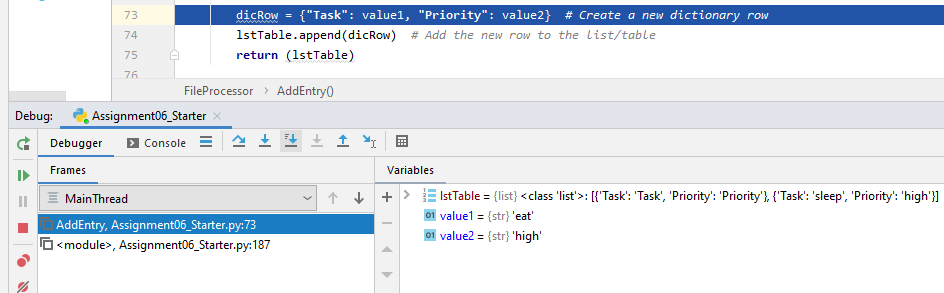
@staticmethod  
**def** AddEntry(value1, value2, lstTable):  
 *"""  
 Desc - Adds an item input by the user into the table* **:param** *value1: (string) the new task:* **:param** *value2: (string) the new priority:* **:param** *lstTable: (list) list that was added to:* **:return***: list  
 """* dicRow = {**"Task"**: value1, **"Priority"**: value2} *# Create a new dictionary row* lstTable.append(dicRow) *# Add the new row to the list/table* **return**(lstTable)

Here the debugger (Figure 4) shows it ready to pass the values collected from the user as well as the current list. The new task is collected but not yet added to the list.

****

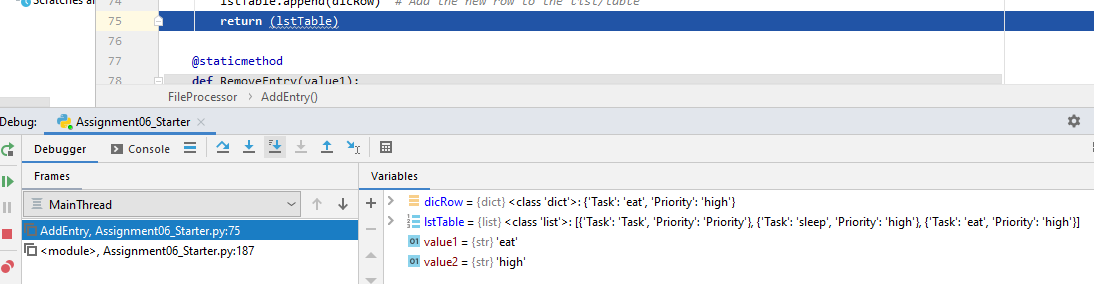
**Figure 4:** Reaching new function AddEntry.

Then the arguments that I passed are put into the function via the parameters (Figure 5). It steps through the current list and will add the new entry.

****

**Figure 5:** Parameters received the arguments.

As the function continues to process, the values are added to the list (Figure 6). Now the new task, eat|high is now I the list.

****

**Figure 6:** List is updated.

After that it immediately calls the IO.ShowCurrentItemsInList(lstTable) to display the table with the new output:

\*\*\*\*\*\*\* The current items ToDo are: \*\*\*\*\*\*\*

Task (Priority)

sleep (high) <<<<< existing value

eat (high) <<<<<< new value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The steps of evaluating each ***to do*** section combined with debugging as I went made the process go smoothly and quickly.

**Menu Option 3**

Here is the rest of the code that I updated (using the debugger, but not displaying it here). For the next menu option 3 we need to remove an entry. I called this newly created this function.

FileProcessor.RemoveEntry(strKeyToRemove)

And then defined it with the following in the FileProcessor class. The user had been asked what row they wanted deleted and that was passed as an argument and picked up by the function. The task was removed from the list. Since the table is a reference object nothing needs to be returned.

@staticmethod  
**def** RemoveEntry(value1):  
 *"""  
 Desc - Removes an item input by the user into the table* **:param** *value1: (string) the row the user wants to remove:* **:return***: nothing  
 """* 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** (value1 == 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*

**Menu Option 4**

After that in menu option 4, we need to write the new list to the file. The newly created function was established and called.

FileProcessor.WriteListDataToFile(strFileName, lstTable)

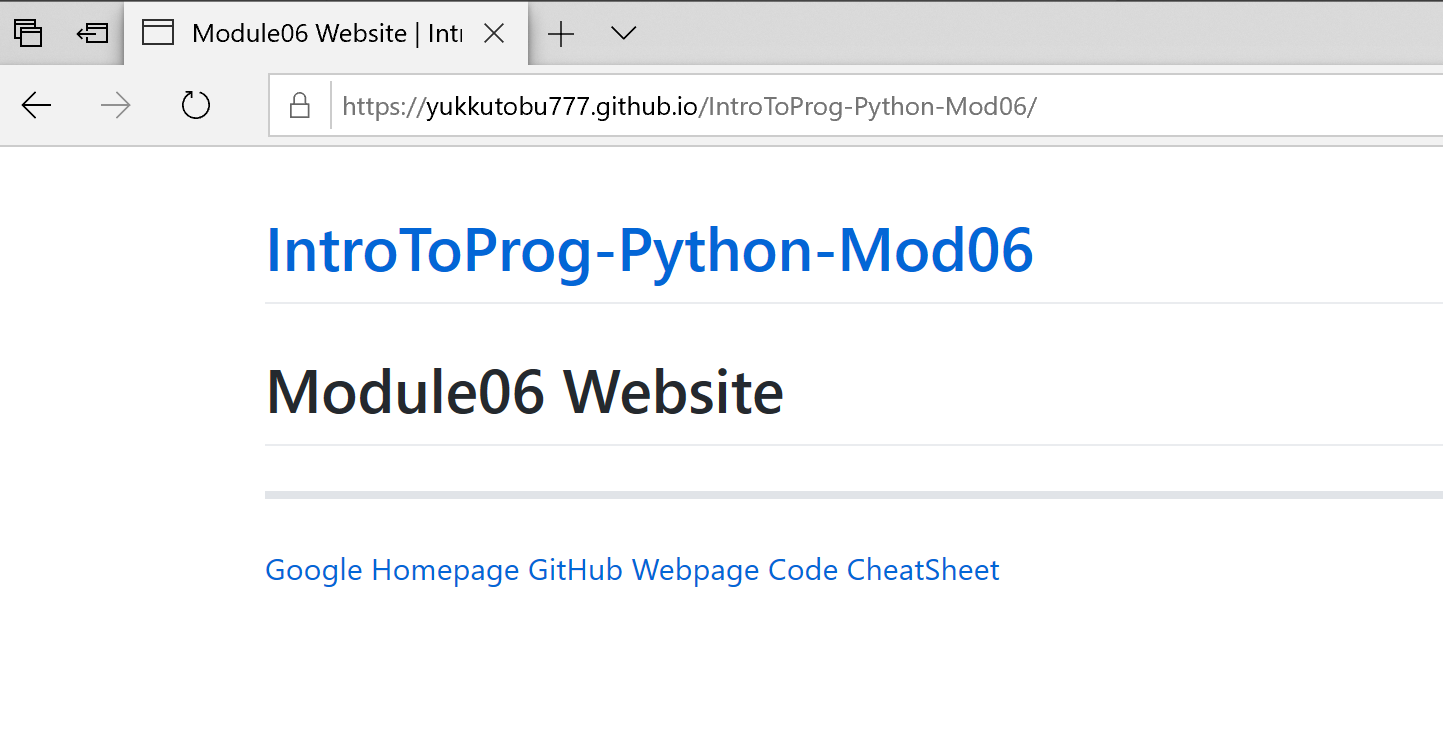
And then defined it with the following in the FileProcessor class. The file name and list were passed as arguments to the parameters and the list was written to the file and the file object was closed.

@staticmethod  
**def** WriteListDataToFile(file\_name, list\_of\_rows):  
 *"""  
 Desc - Writes data from a list into a file* **:param** *file\_name: (string) with name of file:* **:param** *list\_of\_rows: (list) you want to use data from:* **:return***: nothing  
 """* objFile = open(strFileName, **"w"**)  
 **for** dicRow **in** lstTable: *# Write each row of data to the file* objFile.write(dicRow[**"Task"**] + **","** + dicRow[**"Priority"**] + **"\n"**)  
 objFile.close()

There were no ***to do*** items in menu option 5 and 6.

**GitHub webpage**

Last week we created a GitHub repository. This week we learned how to create a new file to create a folder which creates a GitHub webpage using simple mark down language (Figure 7). The default page is index.md located at <https://yukkutobu777.github.io/IntroToProg-Python-Mod06/> . We will do more with this next week.

****

**Figure 7:** GitHub webpage.

**Summary**

In the assignment this week we used functions to organize our code. Classes contained like functions that logically broke our code into the different Scope of Concerns. We declared some global variables in the data section, worked with our file in the processing section, interacted with the user in our presentation section and then listed out the main body of the script. Within the script it called functions from the appropriate sections and returned values as needed. As stated above when using a reference object, the data in the location is changed and it does not need to be explicitly returned. We refrained from using global string or integer variables within functions. Instead they were local and existed only within the function.