Hitakshi Verma

August 8, 2020

Foundations of Programming: Python

Assignment 05

**Create a script using list, dictionaries, and text files**

**Introduction:**

Under the guidance of my professor, Randal Root, I have now started creating professional python programs with the use of well-known sequences like lists, strings and tuples. In this *Module05*, I deep dived into more complex learnings like reading from a text file, dictionaries, functions, concept of improving scripts with an introduction to GitHub.

In *Assignment05*, I am going to create a python script that manages “To Do list” which has two columns; named “Task” and “Priority”. This script loads these columns into the *dictionary object* (as row of data) and further loads the rows into *lists* *object* to create a table of data.

Furthermore, I will post my python script on source control software called *GitHub* in public mode for other’s visibility. To start off with this actual assignment, I want the reader to know the concepts that I have learned and actually used to create this python script.

* Lists: Storing and loading list data to and from a file
* Dictionaries: Storing and loading dictionary data into a list (table of rows) to and from a file
* Improving Python scripts
* GitHub

**Lists:**

In previous Module04, we all know what lists are. Let’s recall it together,

*“The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.”*

For example - list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5]; list3 = ["a", "b", "c", "d"]

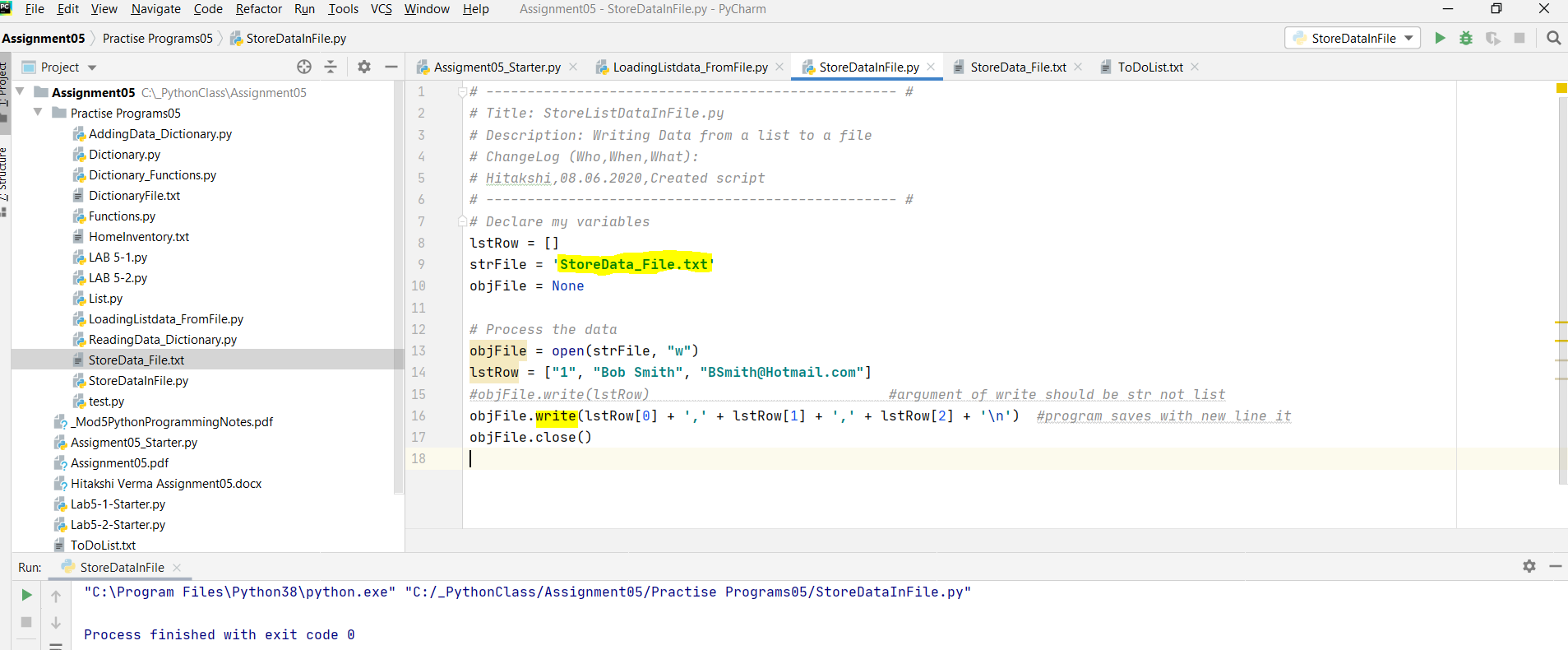
Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on. To access values in lists, use the square brackets for slicing along with the *index or indices(number)* to obtain value available at that index.

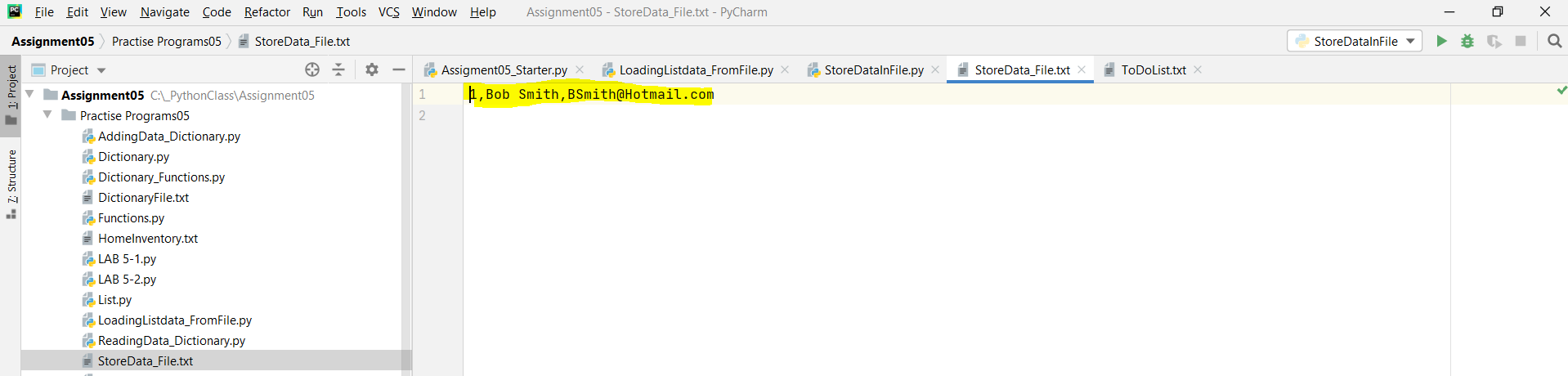
print "list1[0]: ", list1[0]

**Storing Lists Data in a file:**

For a program to run, it has to be loaded in computer’s memory. Data and its instructions store till its execution time. But once the application or program closes, all its associated data gets lost. To avoid this situation, “we” as “developers” started storing data in a text file for its persistent usage and availability.

Below is the snippet that shows how I used write () function to store list data object in a text file.

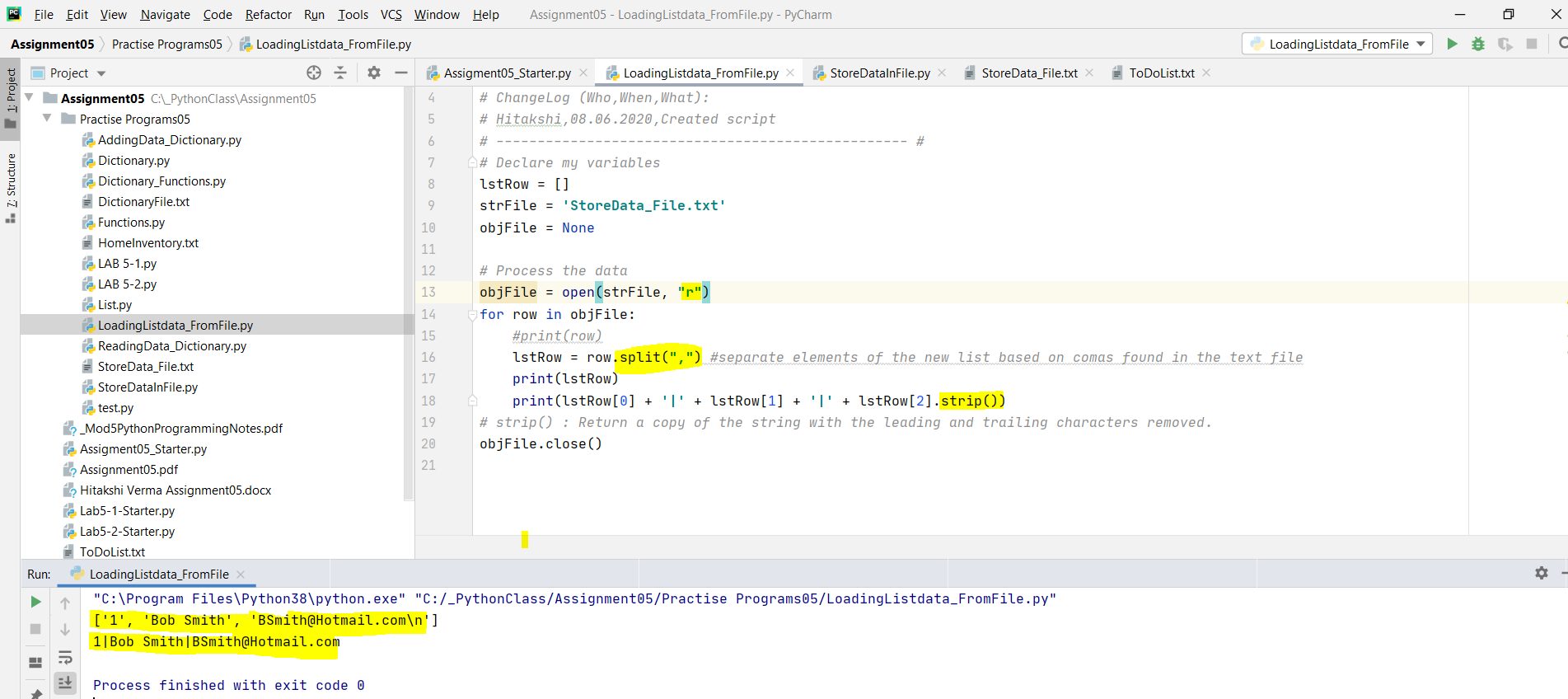
*Figure 1: Script demonstrates list data object stores in a text file using write function where indexes are being used to access the individual list values*

*Figure 2: Snippet shows storage of list data object in a StoreData\_File.txt*

**Loading List Data from a file:**

Data in a file must be loaded back into memory if the user wants to continue using this stored data. *Below is the code snippet where I am using two string functions here; strip() and split().*

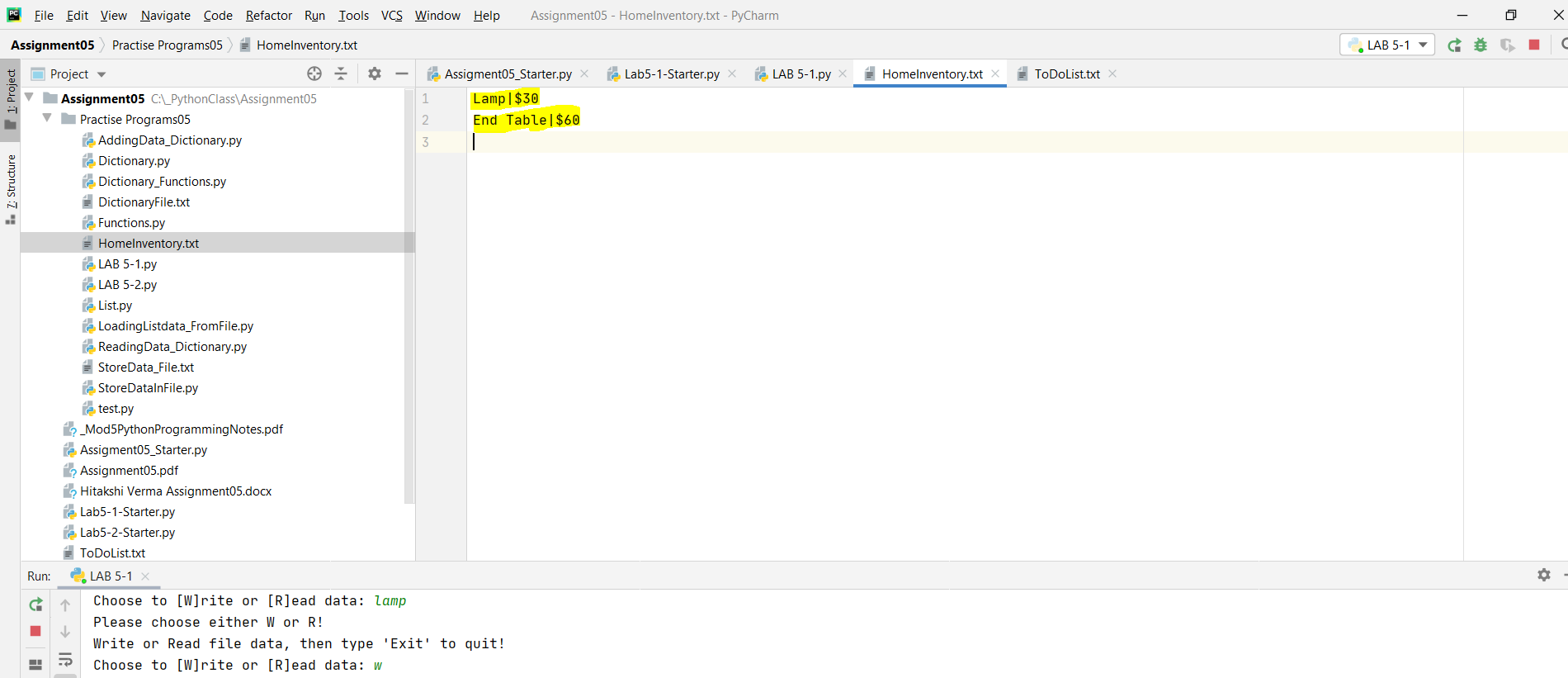
strip() function removes the unwanted carriage return and split() function is used to separate the elements of string separated by a delimiter.

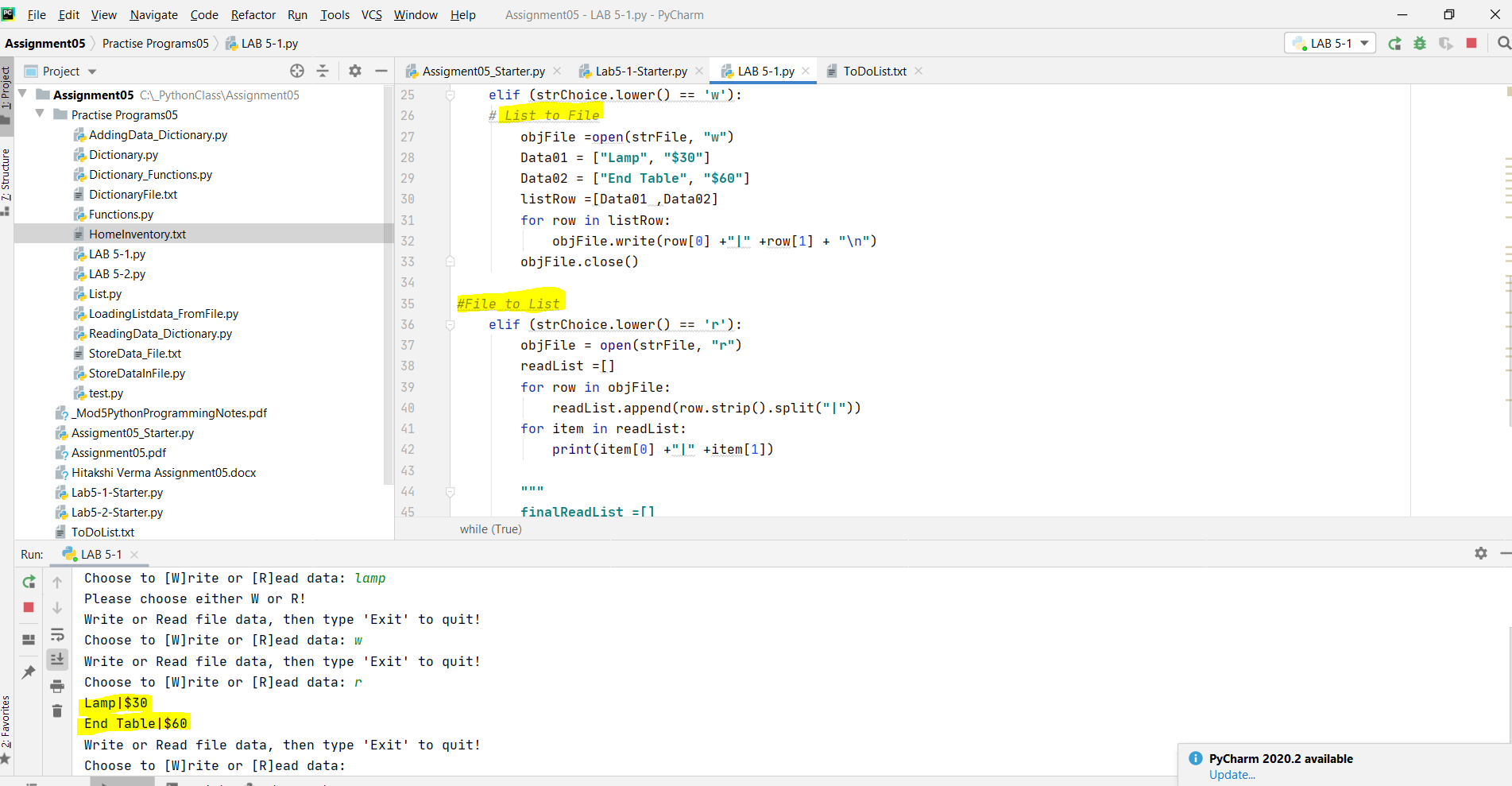
*Figure 3: Script shows the loading of list data object from ‘StoreData\_File.txt using split and strip function*

**Working on LAB 5-1:**

**Here I am writing and reading the following list data in and from a HomeInventory.txt file**

***Lamp|$30, End Table|$60***

*Figure4: Scripts demonstrates the list data gets written in “HomeInventory.txt” file*

 *Figure 5: Script shows reading data from “HomeInventory.txt” file*

**Dictionaries:**

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

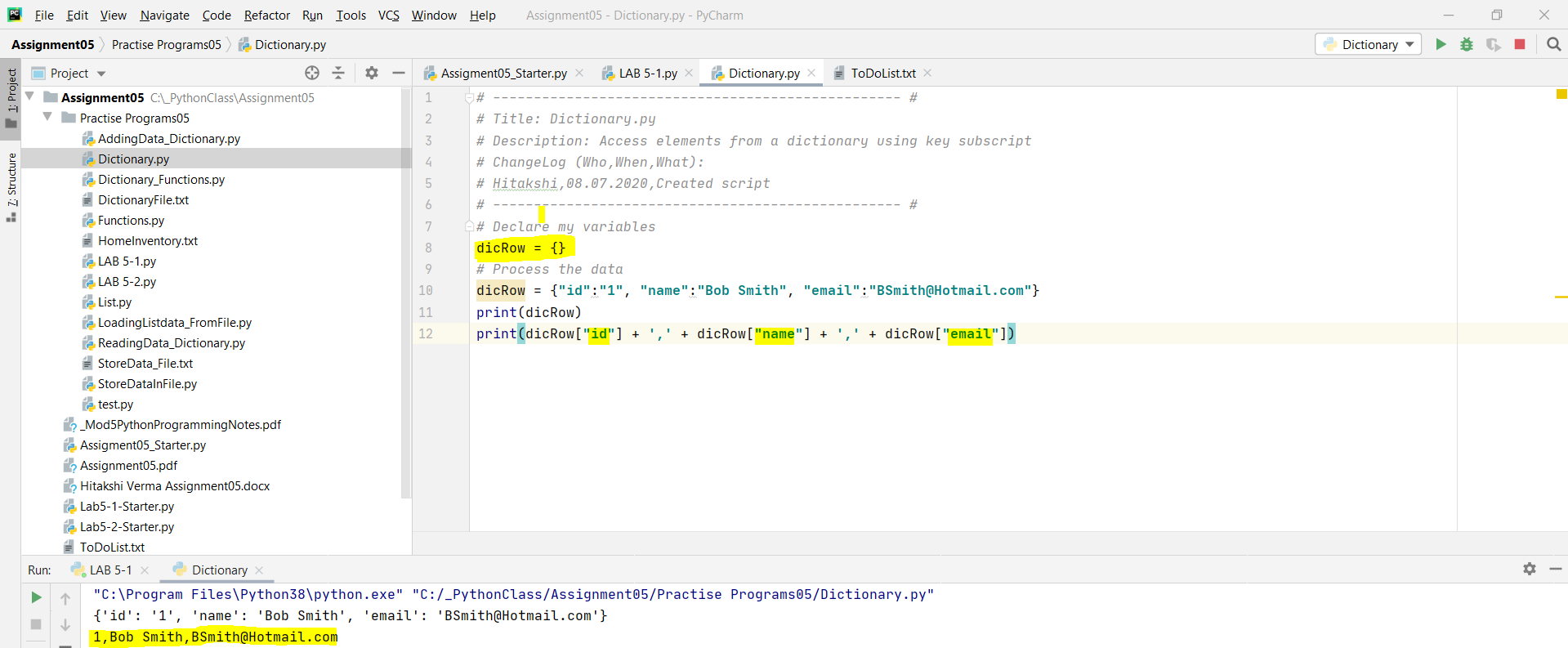
For Example:

thisDict = {"brand":"Ford","model": "Mustang”, “year": 1964}

We can access the items of a dictionary by referring to its key name, inside square bracket.

x = thisdict["model"]

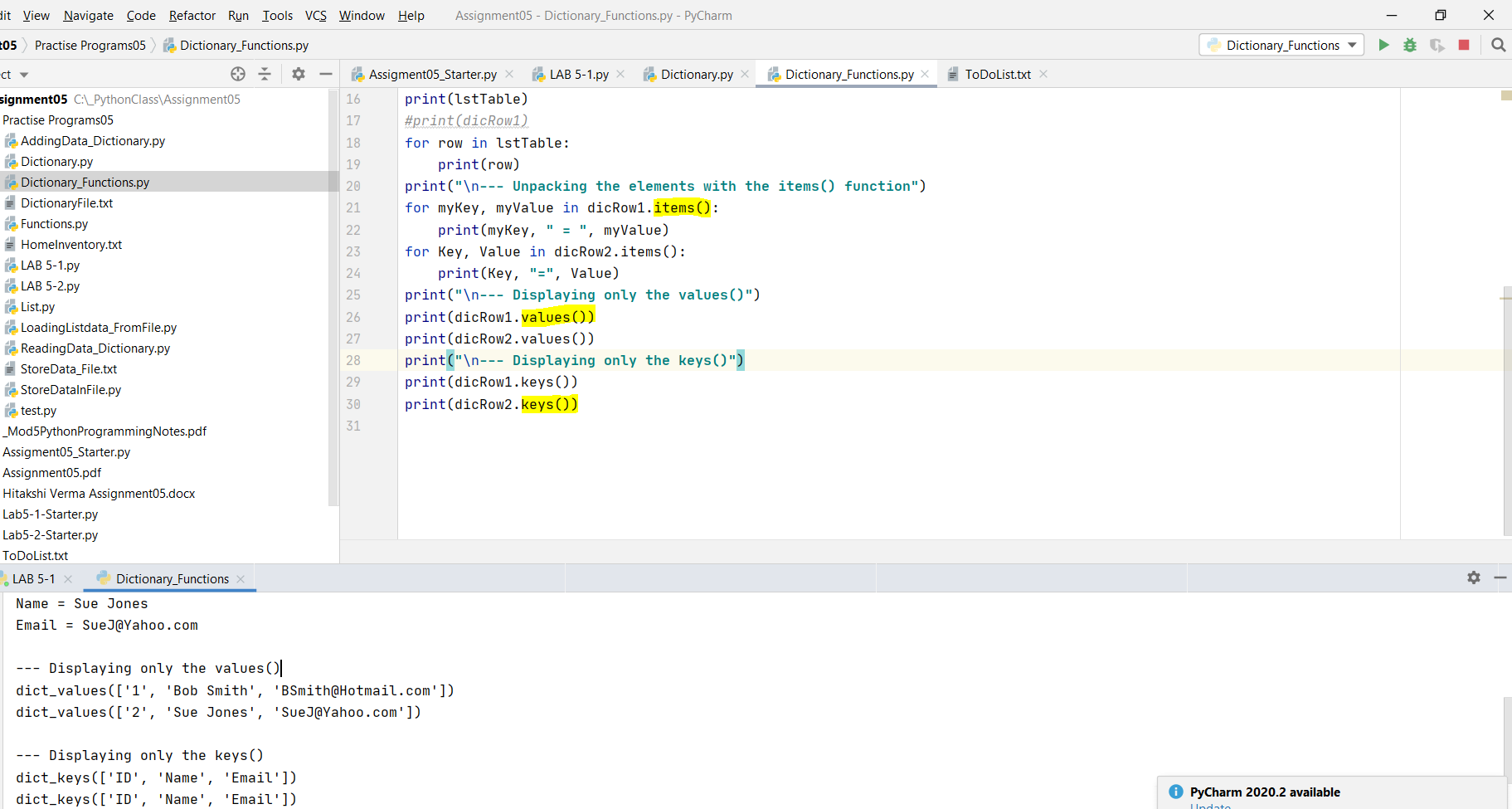
Below is the script that demonstrates the dictionary sequence:

*Figure 6: Script demonstrates the accessibility of dictionary elements using key as subscript*

Dictionaries are like row of data and can be added as collection in list to form a table. Dictionaries have its own in -built functions like items(), keys() and values().

* items() function is used to unlock the dictionary object into items
* keys() function gives the key value
* values() function display only values not keys

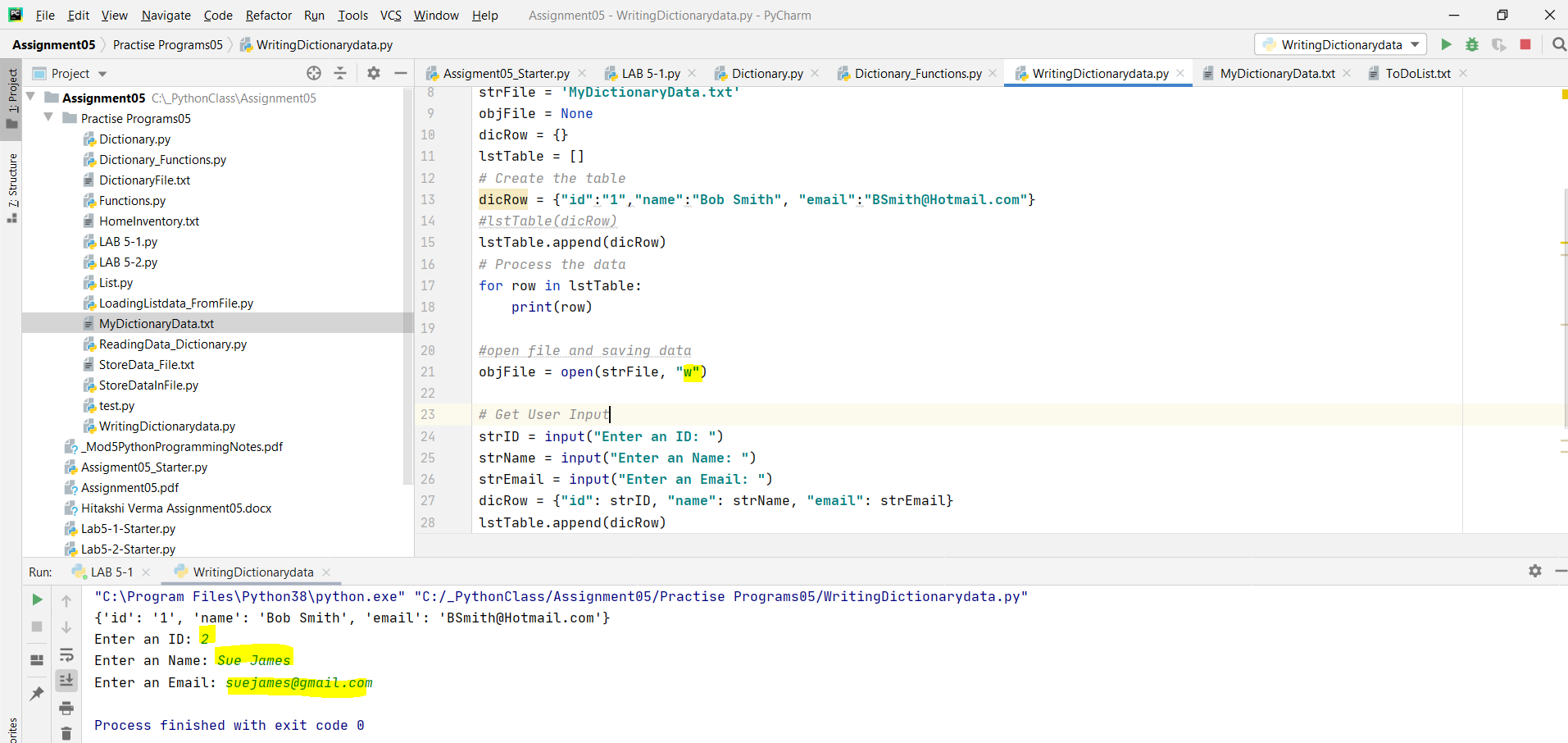
Below is the code snippet that demonstrates the dictionary in built function:



*Figure7: Script demonstrates storage of dictionary object into list object and dictionary in-built functions*

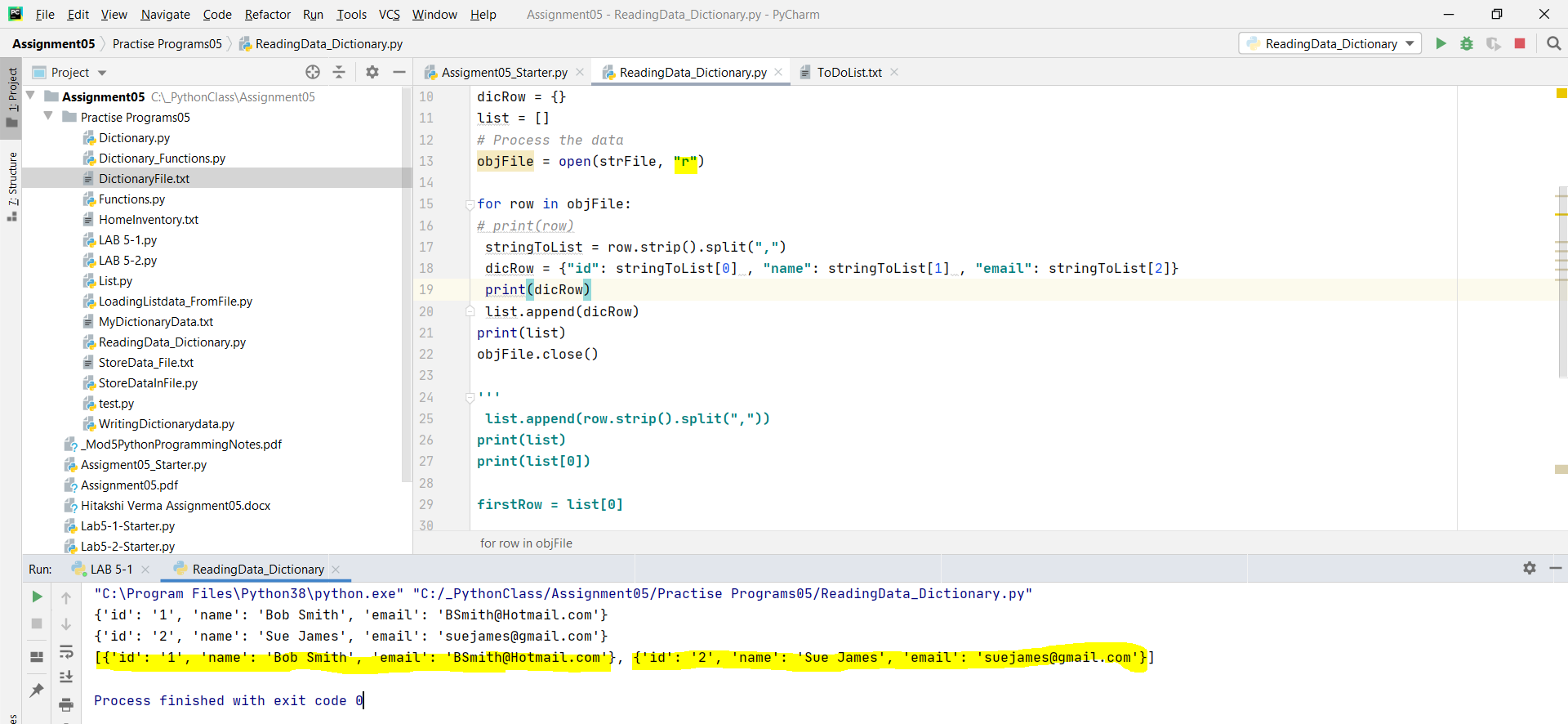
**Storing/ Writing dictionary (row of data) in a text file:**

Dictionary data as row is first appended to list of table and then data is written in a text file called “MyDictionaryData.txt”.

*Figure8: Script demonstrates the addition of dictionary row in list table and writing dictionary data in “MyDictionaryData.txt” text file*

**Loading/Reading file data into List of dictionaries:**

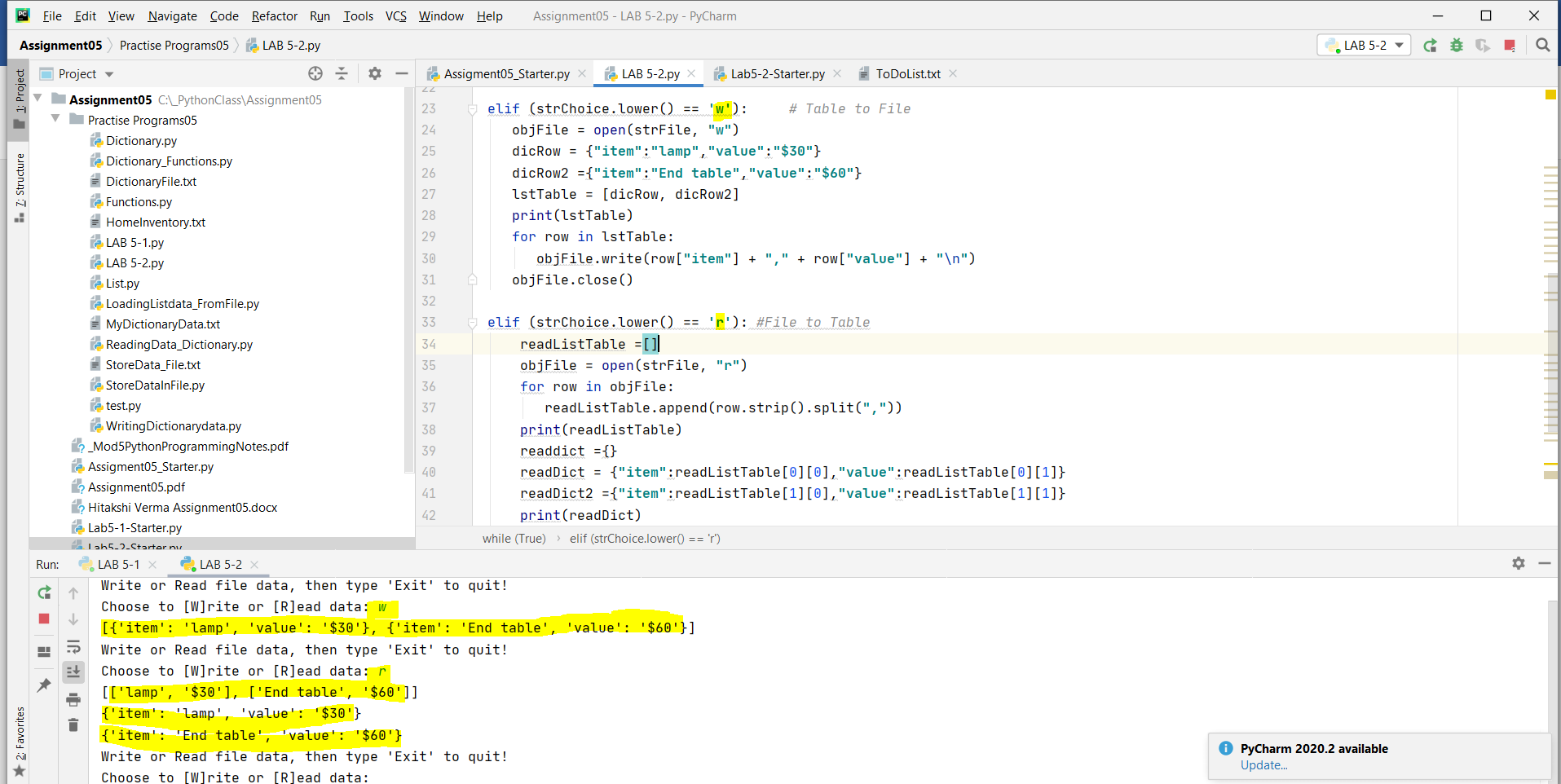
Now, we are back to reading “MyDictionaryData.txt” file data and storing into a list.



*Figure9: Script demonstrates the reading of “MyDictionaryData.txt” text file data into list.*

**Working on LAB 5-2:**

Below is the code snippet of writing and reading home inventory data in a text file from and to a list of dictionary objects.



*Figure10: Script demonstrates storage and loading of list of dictionary objects to and from “HomeInventory.txt” file*

**Improving Python Scripts:**

Organization and neatness of code plays an important role specially when we are digging into creation of more professional programming.

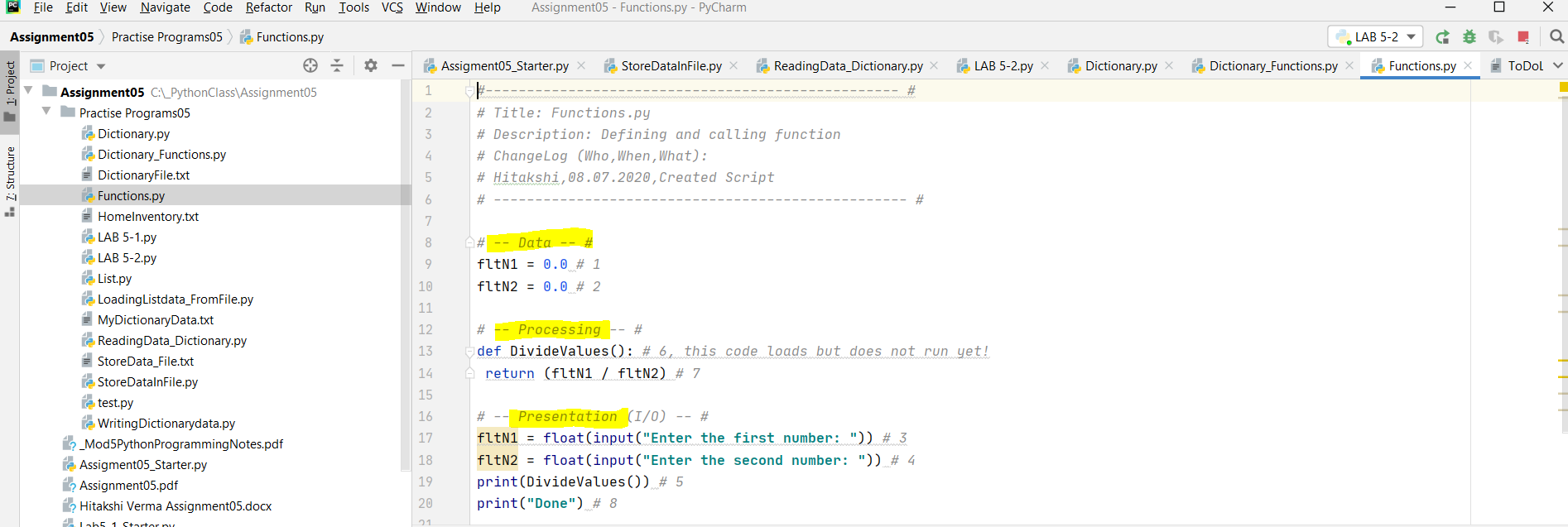
There are 4 techniques that helps us to improve our scripts:

1. Separation of Concerns
2. Functions
3. Script Templates
4. Structured Error Handling

***Separation of Concerns:***

Separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, so that each section addresses a separate concern. Most programs can be divided into three different sections*; Data, Processing, and Presentation* (or Input-Output).

*That’s what we have seen in my above code snippets. I am sharing a small code snippet which demonstrates the above-mentioned sections:*

*Figure11: Snippet demonstrates the division of the Data, Processing and Presentation sections*

***Functions:***

Functions allow us to group a set of programming statements and later reference them by a given name. In Python, function must be defined in a script before they can be called. The statements of a function run later in your program by “calling” the function. When the script starts, it loads the function into memory but waits to run its statements.

*Function makes it easier to separate the presentation and processing code. Above code snippet shows DivideValue() function where in the first instance, function was defined and in second instance, function was called.*

***Script Templates:***

Script templates inform the user about the details of the program. It highlights the script creation details, purpose of its creation and program sectional details like Data, Processing and Presentation.

Below is the Script template:

*#-------------------------------------------------- #  
# Title: Functions.py  
# Description: Defining and calling function  
# ChangeLog (Who,When,What):  
# Hitakshi,08.07.2020,Created Script  
# -------------------------------------------------- #*

*# -- Data -- #*

*# -- Processing -- #*

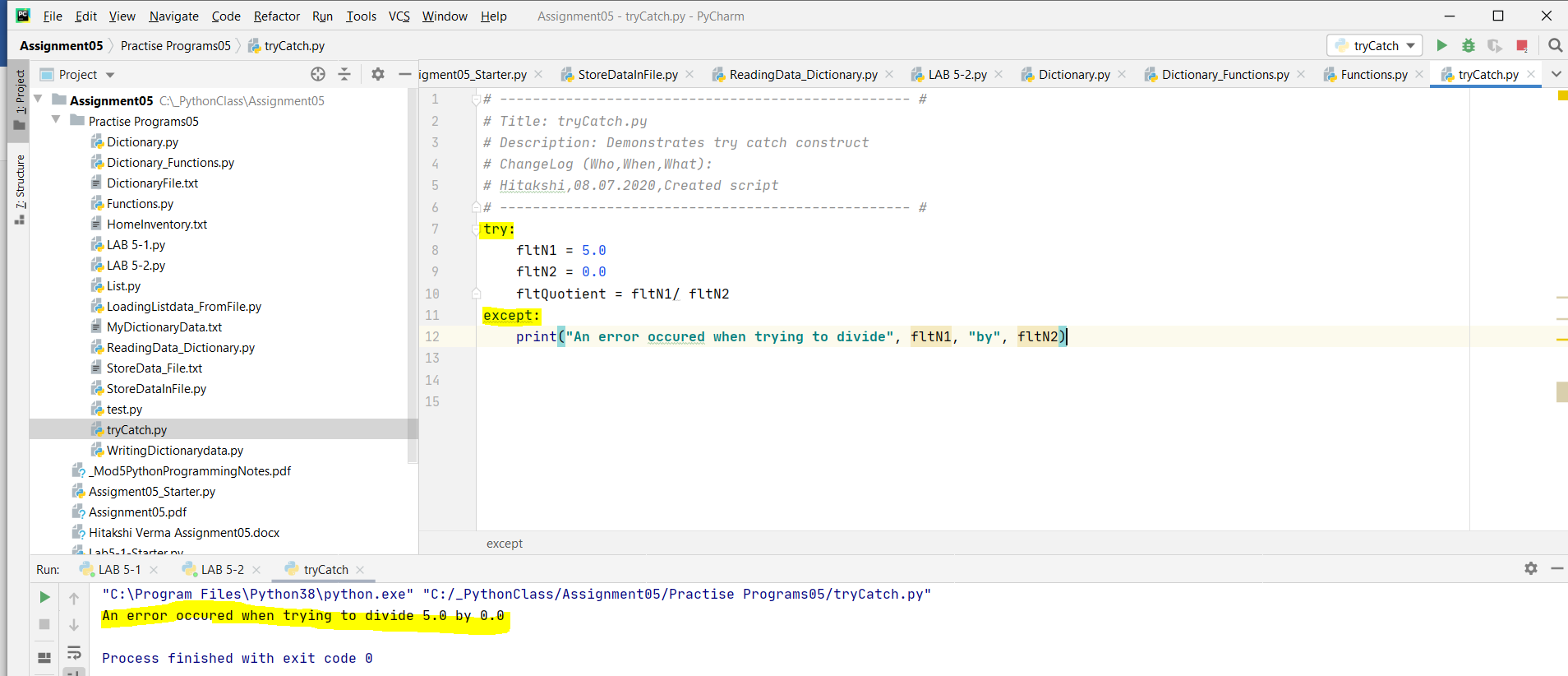
*# -- Presentation (I/O) -- #*

***Error Handling:***

Error Handing improves our scripts by managing errors we may not have control over in any other way.

We can trap errors in your programs using a *try-except construct*.

* *Advantages of using Try- Catch construct:* Ability to provide more general and user-friendly error messages.
* Allows for a simple, organized way of grouping statements to be processed.
* If an error occurs in the grouped statements, Python’s construct helps the interpreter to automatically move to another set of statements.



*Figure 12: Script demonstrates try – catch construct*

**GitHub:**

To make backups of your code files and to make them available for others to access, I am using one of the most popular *source control software is Git*, which can store your files on the Internet at GitHub.com.

**Assignment 5 – Modify data in a TodoList.txt file using lists and dictionary**

In this assignment, I have created a “Menu” which allows user to perform the below operations in an already existing text file.

**1) Show current data  
2) Add a new item.   
3) Remove an existing item.  
4) Save Data to File  
5) Exit Program**

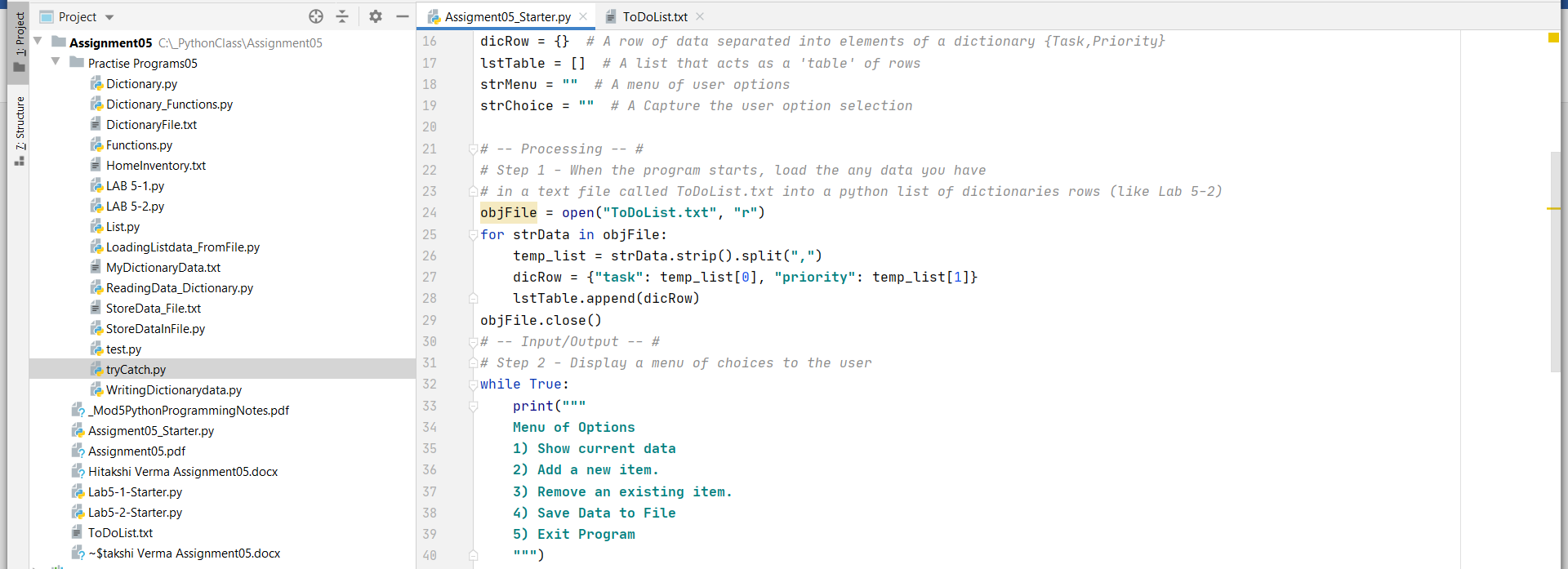
A while loop was used which would provide the above options to the user until he selected “Exit Program”. Here, I have assumed that a *ToDoList text file* already exists. This file contains 2 columns of data – *Task* and *Priority*. While *Task is a String* and can contain multiple characters, *priority* can only contain 3 values – *“high, medium or low”.*

I have used the *Assignment05\_Starter file* which was provided by our professor. I divided the program in multiple steps as follow –

**Step 1:**

This is the processing section of the script, where I loaded the data which was already present in ToDoList text file. I opened the file in “read” mode and iterated over the file using a *for loop*. Within the loop, I have sanitized the data by removing characters using *strip function* and also used *split function* to divide the string into a temporary list. This list contains the name of the task and its priority.

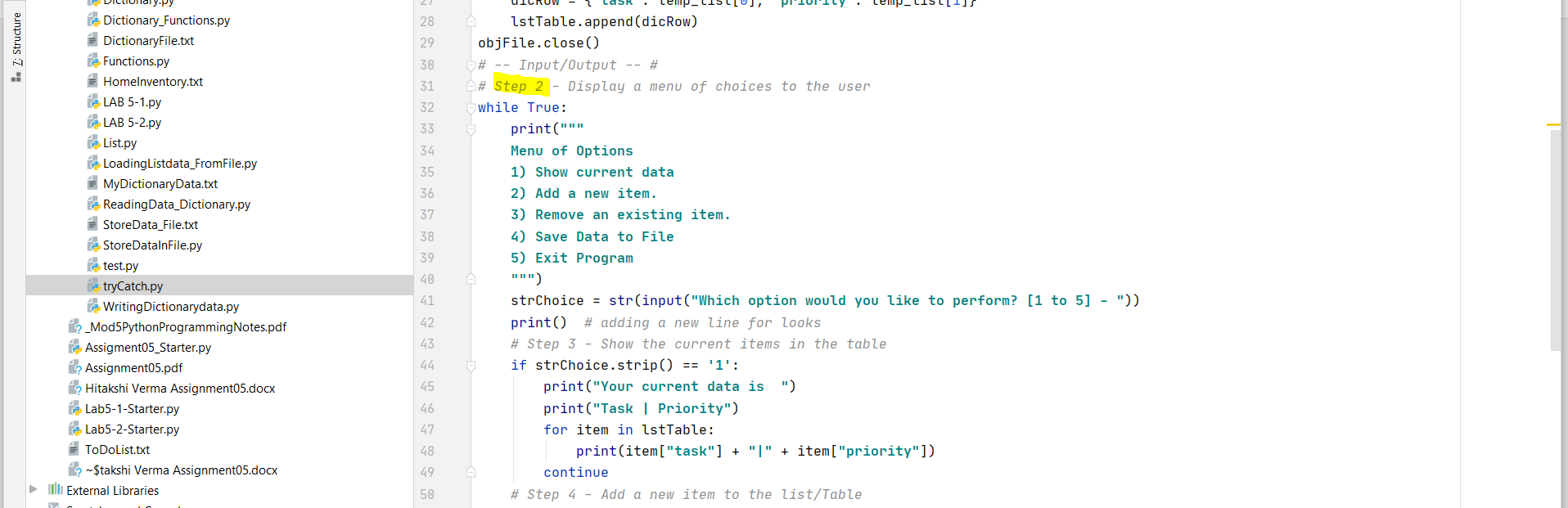
Next, I added these values into a dictionary and stored it in a variable called *dicRow*. This dictionary was added as an individual item in a table called *lstTable*. Going forward, in this program, I will perform operations like add/ delete onto *lstTable*. I closed the file after the *for loop*.



*Figure 1: Loading data from the file and showing the menu of options to user*

**Step 2:**

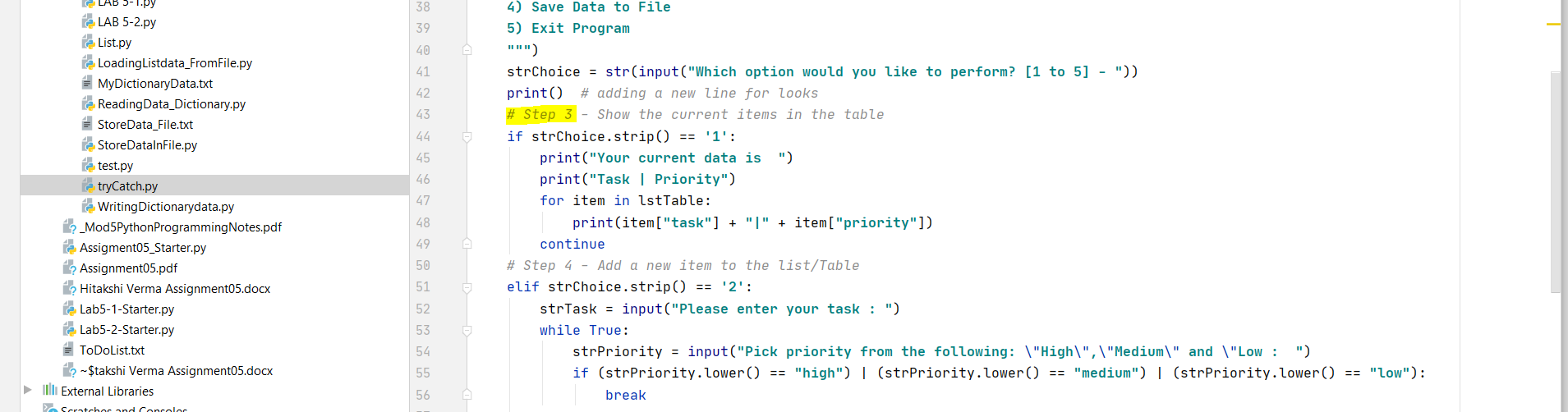
I asked the user to choose any of the 5 options as shown on the screen. I used a string to capture user input. If the user input did not match any of the items, the code prompted the user to input again.



*Figure 2: Displaying the menu to user*

**Step 3:**

If the user selected “1”, the code would display the user all the data which was present in the lstTable. I used *for loop* and display the task and its priority which was separated by a “|” for better readability.



*Figure 3: Display all the tasks and priority on the screen*

**Step 4:**

Once the user selected “Add an item”, I would ask the user to enter the task name and stored it in a variable called *strTask.* I assumed that the user will enter a valid task and there were no input validations added.

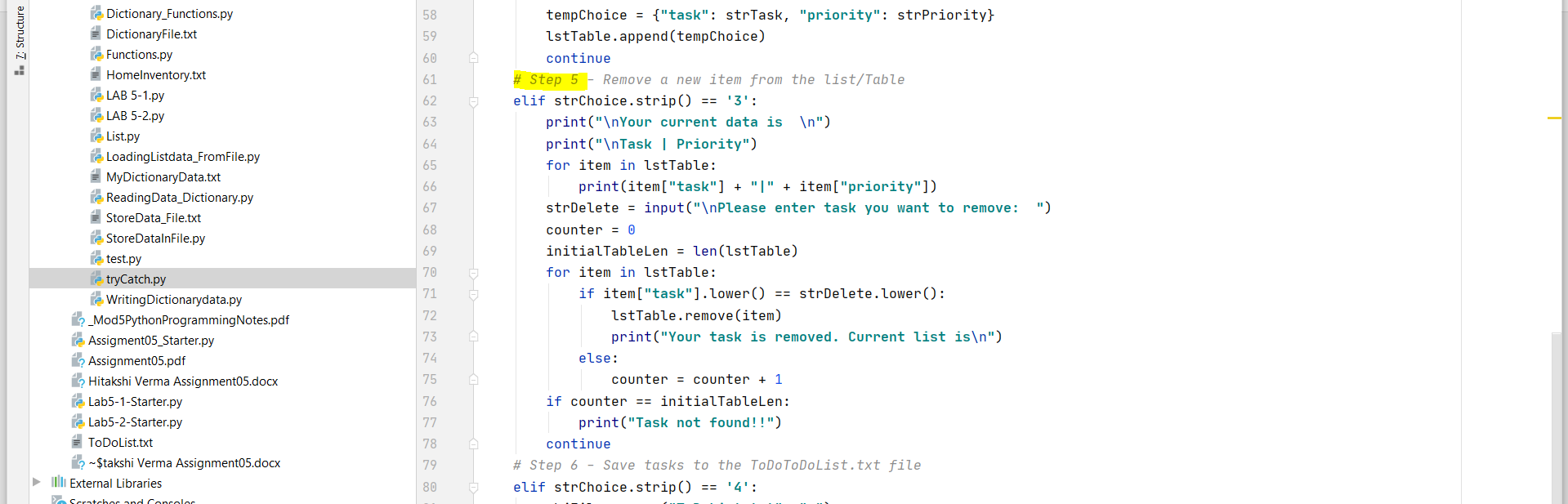
For priority, the code will ask the user to choose between “high”, “medium” or “low”. *I did a string comparison using if statement to ensure that no other value was provided as an input else the user would be prompted again*. I stored this data into a new dictionary *tempChoice* and appended this to *lstTable*.



*Figure 4: Add a new task to the existing list of tasks*

**Step 5:**

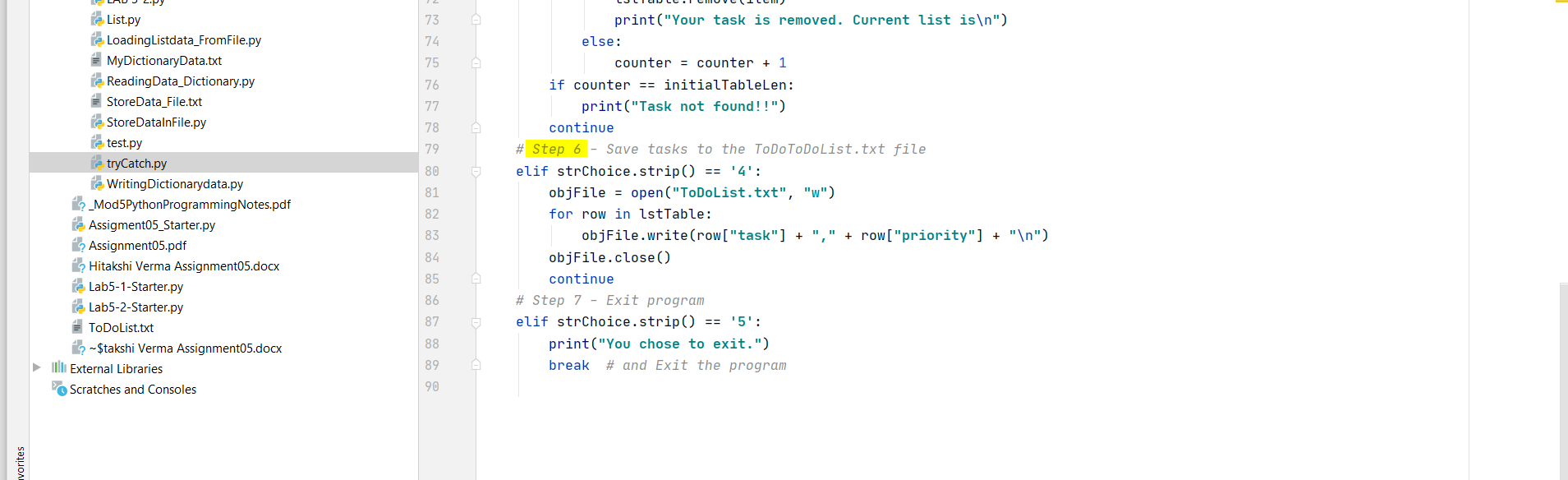
If the user wanted to remove an item, I added a code to first display all the items to user and then asked user to enter the task he wanted to remove. I used the same code which was used in *Step3* to show data to the user. I used a new variable *strDelete* which would store the task name; user wanted to remove. I iterated using *for loop* over the lstTable. I did a “task” comparison between the user input string and the “task” stored in the lstTable. *If it matched, the whole item was removed from the lstTable; else a message was shown to the user – “task not found”*. This was achieved using a *counter variable* which would increment each time the user input did not match the task in the list. At the end of the loop, if the counter value was equal to the initial list length, it would imply that the task was not found.



*Figure 5: Remove a task from the list of tasks*

**Step 6:**

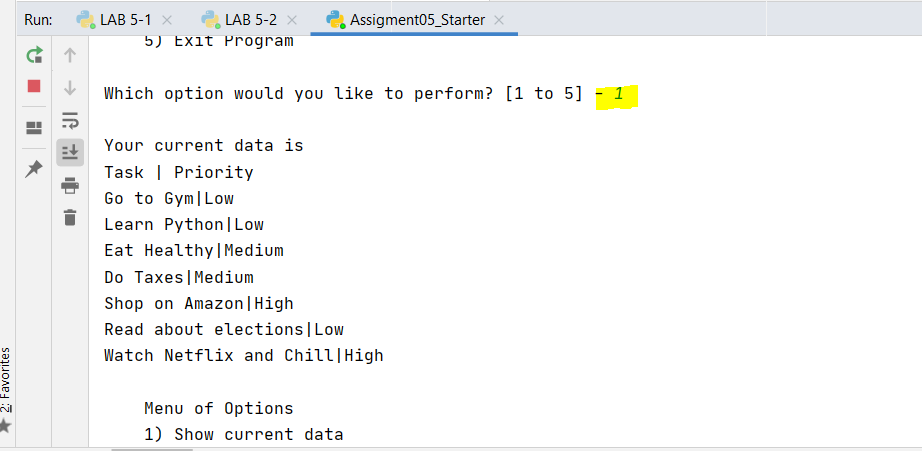
This was a straight forward step, where I saved the data in lstTable into the file. I opened the file in “write” mode and iterated over the *lstTable*. For each row in lstTable, I would write the data into the file. Each row in the file represented a comma separated task and priority. Once, all the rows were added, I closed the file using the close() function.



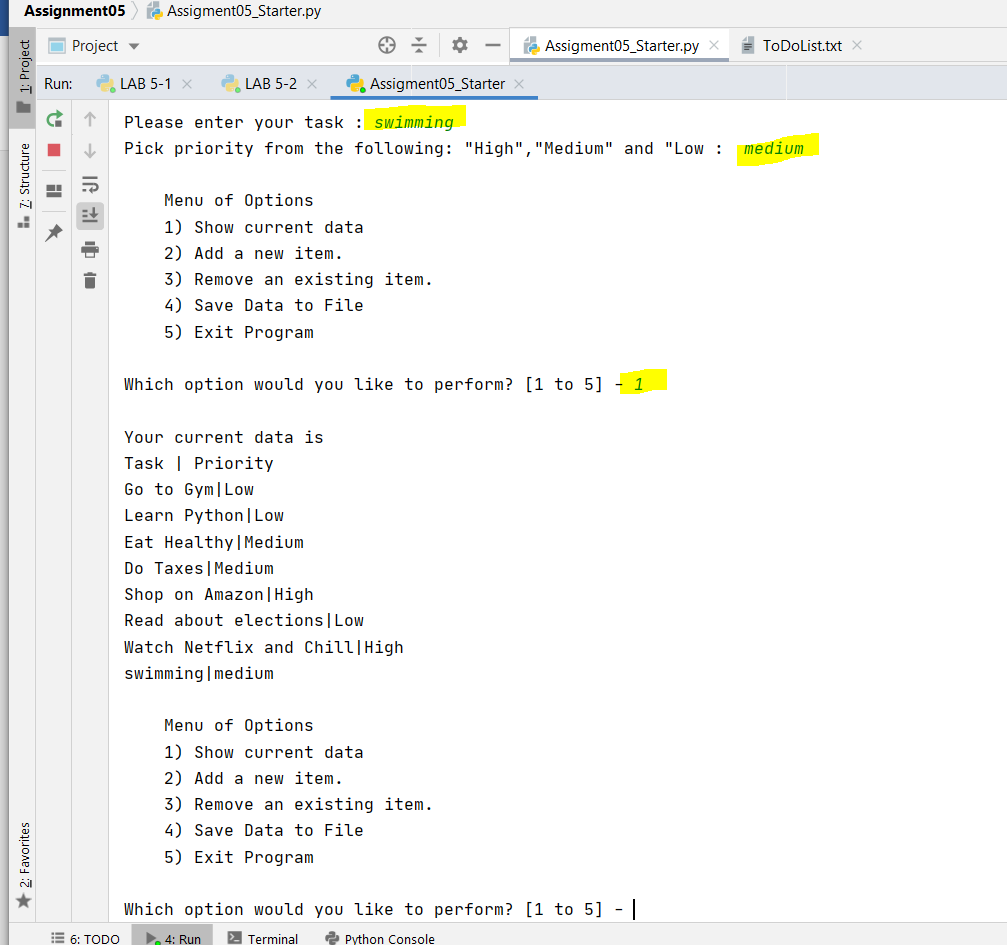
*Figure 6: Save the current list of tasks to the text file*

**Step 7:**

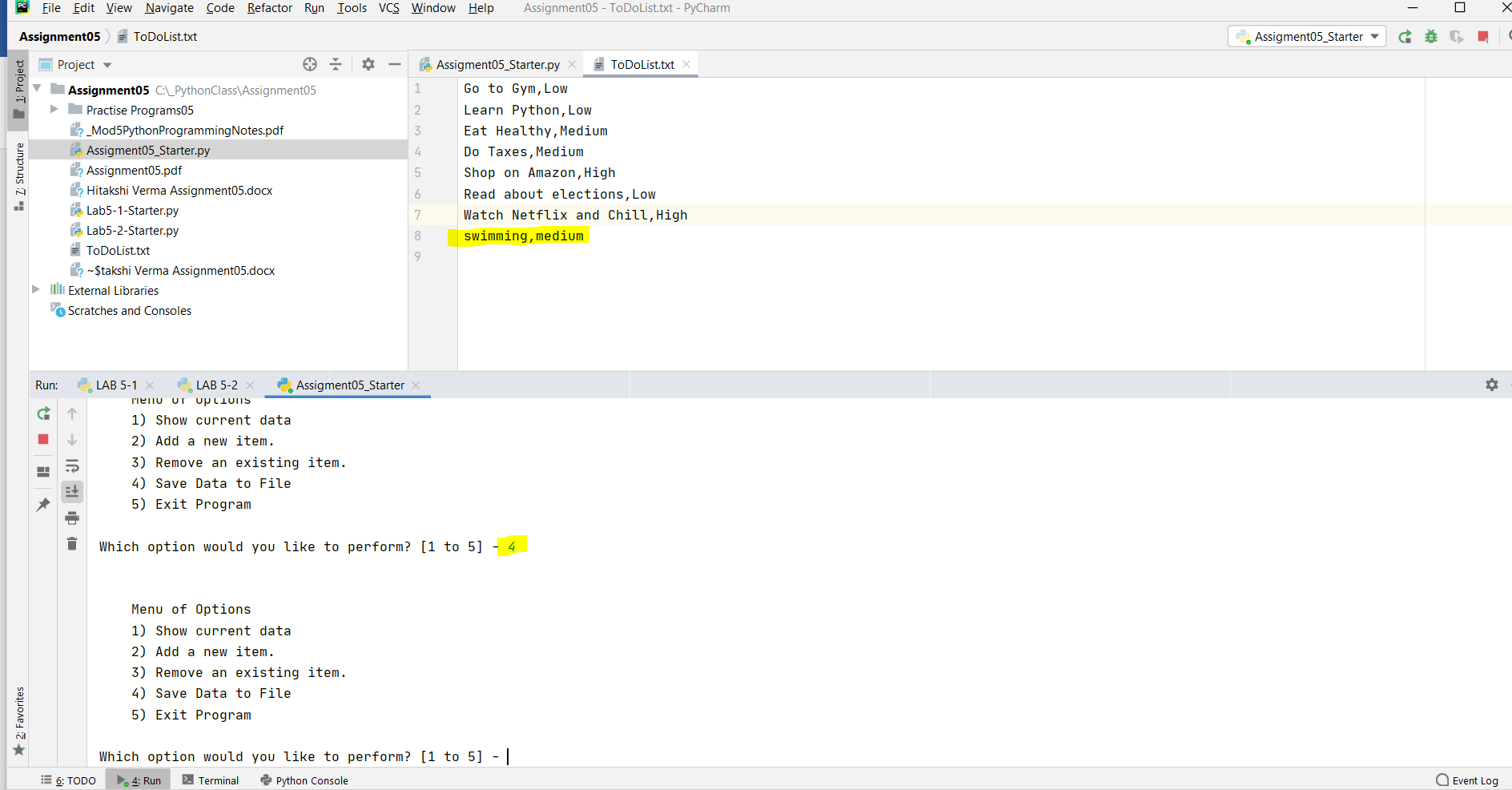
This was the last option in the menu. If the user selected “exit” option, it would simply break from the loop and end the program.



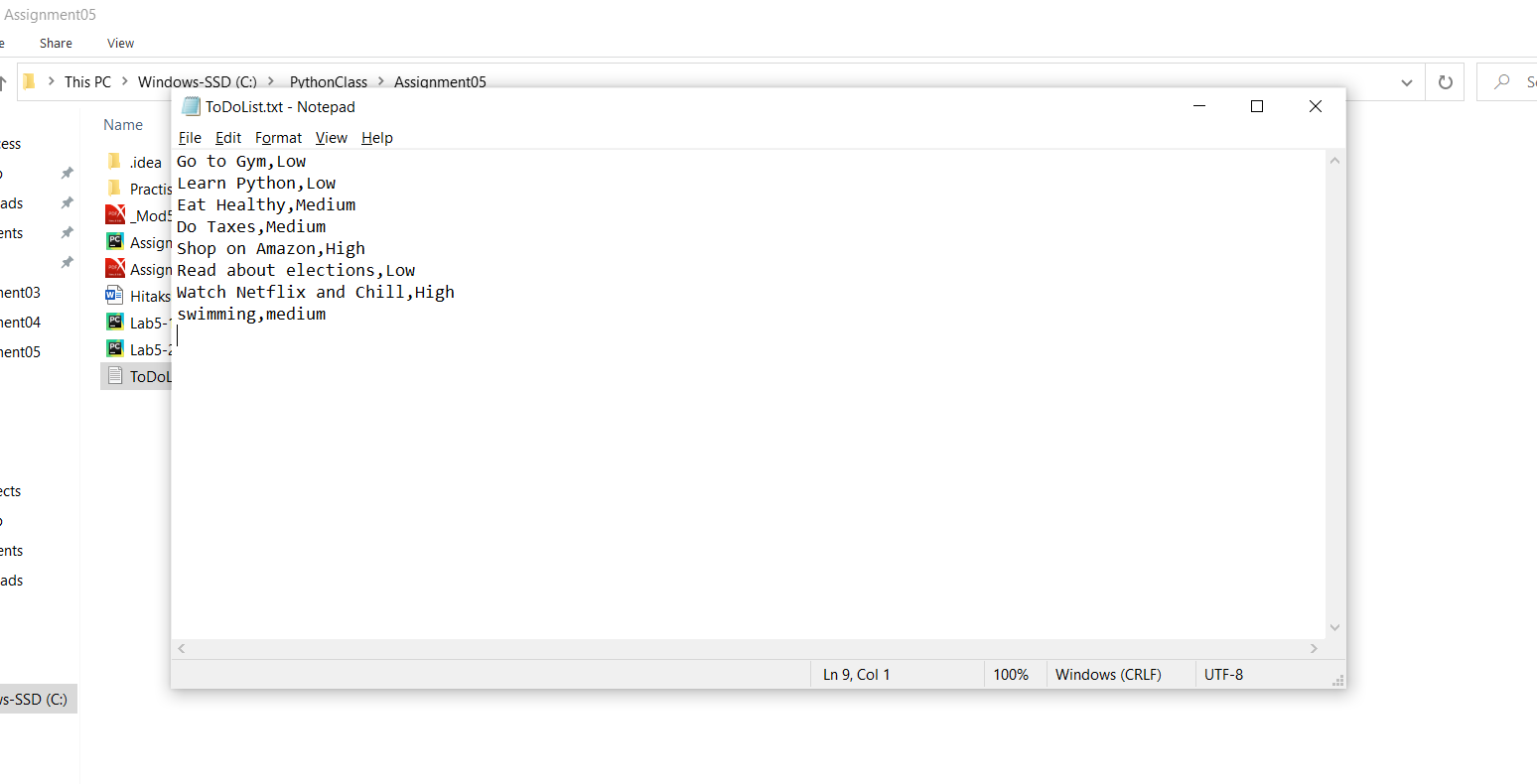
*Figure 7: Running code from PyCharm*



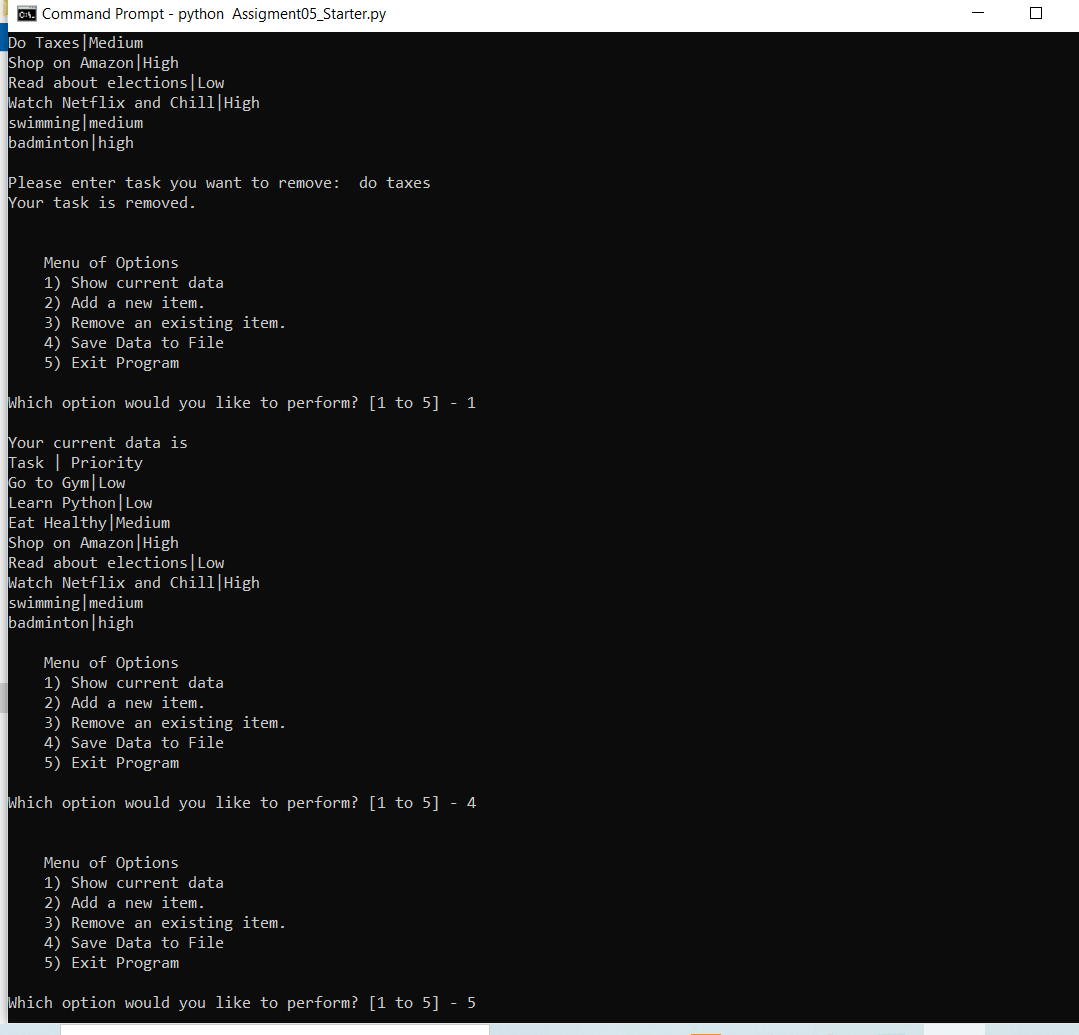
*Figure 8: Choosing individual menu items*



*Figure 9: Saving a new task to the list*



*Figure 9: ToDoList text file*



*Figure 10: Running code from command line*

**Summary:**

In the continuation of understanding the concepts of sequences like lists, tuples and string; I dig deep into another python’s important sequence called as “Dictionaries”; practically understood several concepts like How to store and load lists and dictionary object into and from a text file using important functions like split, strip, items, values, keys etc. Learnt about functions and error handling by creating python script and doing lab work. I have also realized how importance it is for us to adopt best coding practices to maintain our code specially when it’s of great length. I have adopted separation of concerns and script template for better readability and professional programming. Finally, I also got a chance to get familiar with GitHub (source control software) where I pushed and commit code for its persistent storage.