Pavel (Pasha) Beresnev

11/24/20

IT FDN 110A

Assignment 06

# How to Create a To-Do List in Python Using Functions

## Introduction

In this module I’ll describe how to create a to-do list program using functions. I will split this up into different code sections depending on if I am showing presentation code, processing code or the main part of the code. Finally, we will save our list to a text file to store the data entered.

## Data/Header

First I created a header to show the change log. I also declared all the variables that will be used in the program. See figure 1:

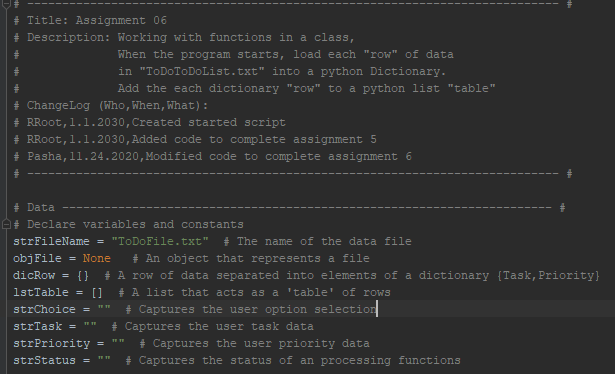


Figure : Header/data

## Processing

Next I created the processing section of the code. The first step is to create a class that I called “Processor”. I also created a document string to describe the class using triple quotes. I then created the various functions I’ll use to process the data using @staticmethod. The first function I created is “read\_data\_from\_file”. This function takes two arguments, the file name and a list of rows. I described what it does in the document string along with its parameters and the return. Then I indented the code for this function. I first clear the current data from the existing list just to make sure there is nothing extra there before we populate it with data. I create a file variable for the file object and we then open whatever file name is passed, which is the argument, when the function is called. We do this in “read” mode since we simply want to load the existing data from the file. Then for each line in the file we use a “for” loop to first unpack the two elements of each line and assign the values to the variables “task” and “priority”. Then I create a dictionary, called “row” and create two keys, “Task” and “Priority” and give them the values from the “task” and “priority” variables, stripped of any spaces. Then I append my list\_of\_rows with each row/dictionary of data. Then we close the file and return the new list of rows and the message “success”. See figure 2:

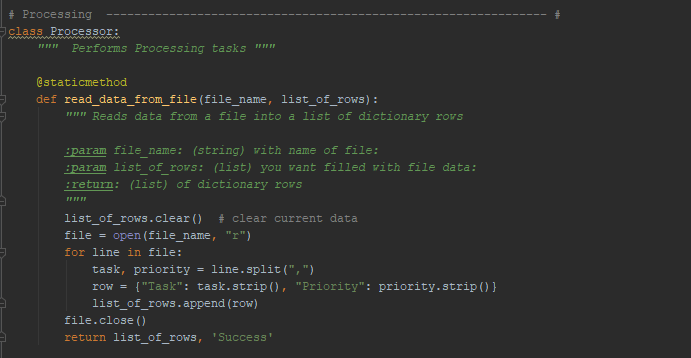


Figure : Processor Class and function to read data from file

I then create a function to add data to the list. This takes a task to add, a priority to add, and a list of rows to add to. I similarly have a document string to call out what this does, identify the parameters and identify the return, which is a list of dictionary rows with the added data. To accomplish this I first use the local variable “row”, which is only accessible within this function, and assign it to a dictionary consisting of the key “Task”, which has the value of the task argument cast as a string and stripped, followed by the key “Priority” which has the value of the priority argument cast as a string and also stripped. Then I append the row to the list of rows and return the new list of rows and the message “success”.

The next function is one to remove data from the list, which takes a task and list of rows arguments. I similarly listed the document string and show that it returns a list of dictionary rows, with the data removed. To do this I iterate over each row in the list of rows with a “for” loop. In each iteration we check if the value for the key “Task” in lowercase equals the task argument in lowercase. If so, we remove that row of data from the table. At the end we return the list of rows with the message “success”. See figure 3:

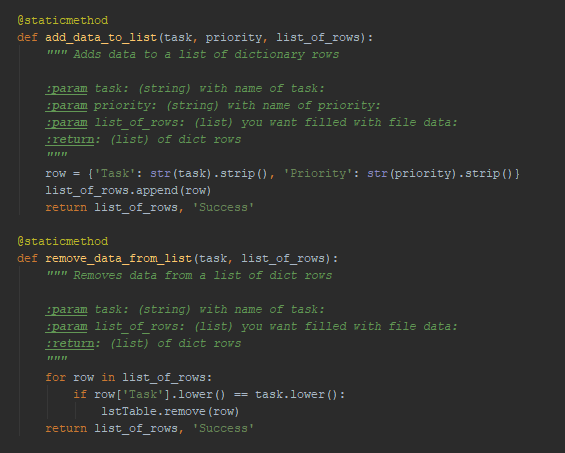


Figure : Add data to list and remove data from list functions

Finally, I create a function to write data to the file in this class. This function takes a file name and a list of rows and returns a list of dictionary rows. To do this, I first create a file variable for the file object and open it, using the file name parameter and the “write” mode. Then I iterate over each row in the list of rows. In each iteration I write to the file by taking the value for the “Task” key and concatenating it with the value for the “Priority” key using a comma in-between and adding a new line at the end. Then I close the file and return the list of rows with “success”. See Fig 4:

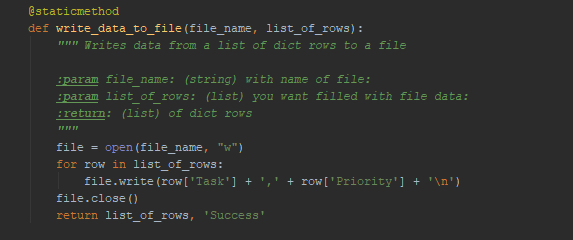


Figure : Write data to file function

## Input/output

The next section of the code is the presentation code. I create a class called “IO” and add a document string again to describe it. The first function in this class is to print the task menu, as shown by the function’s document string, which has no parameters and returns nothing. This function simply prints the menu of options along with an extra line for looks as seen in figure 5:

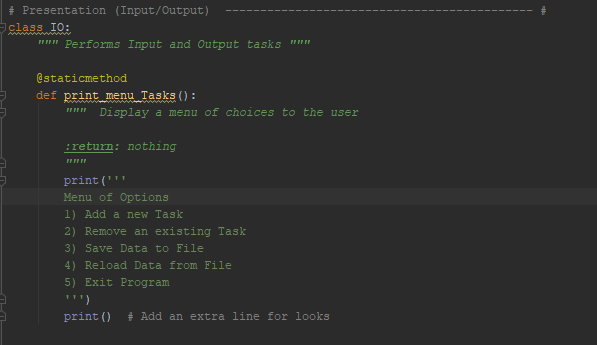


Figure : Presentation code with IO class and print menu task function

I then create a function to get a menu choice from the user and return that choice. To do this, I create a variable, “choice”, and assign it to the input from the user for which option they want to perform, cast as a string and stripped. I add an extra line for looks and return the choice.

Another function I make is one to show the current tasks in the list of dictionary rows. This takes a list of rows and returns nothing. To do this I first print a statement that the current tasks are as follows. Then I iterate over each row in the list of rows. For each iteration, I print the value for the key “Task”, concatenated with a space and parenthesis with the value for the key “Priority” with the closing parenthesis. I print another two lines for looks.

Then I created a function for getting a yes or no choice from the user. This takes a message and returns a string. To do this I simply return the input to the message stripped, in lowercase and cast as a string as shown in Fig 6:

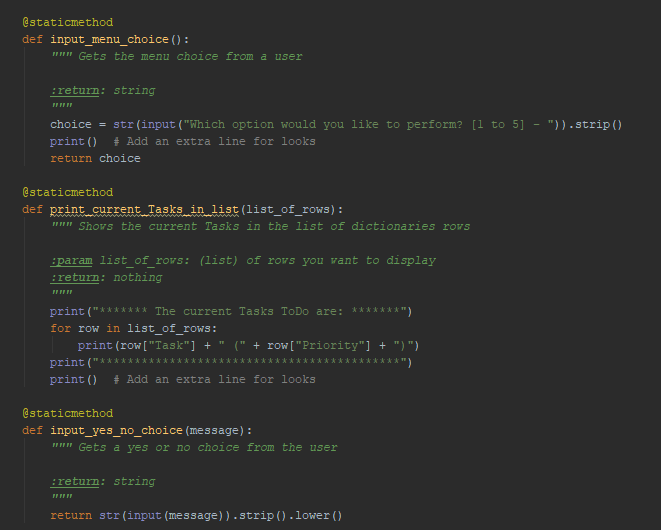


Figure : Functions for getting menu choice from user, printing the current tasks in list, and getting a yes/no choice from user

Then I created a function to pause the program and show a message before continuing. This takes an optional message, which is defaulted to blank and returns nothing. To do this I print the optional message argument and then simply use input to tell the user to press enter to continue.

Then I made a function for inputting a new task/priority. This displays the prompt to the user and returns two strings which are the task/priority. To do this I first create a local variable “task” that takes the input to a prompt to the user and strip it and cast it as a string. Then I do the same for the priority. Finally, I return the task/priority.

The last function in this section is one for inputting a task to remove. To do this I create a local variable, “task”, that is assigned the user input for the task and casts it as a stripped string. I print a blank line and return the task as shown in Fig 7:

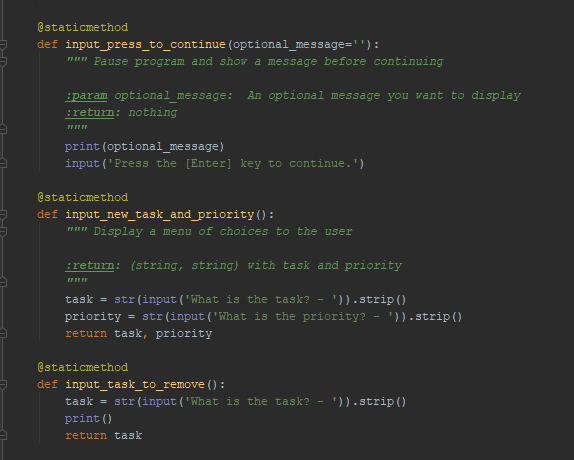


Figure : Functions to input enter to continue after a message, input task/priority and inputting task to remove

## Main body

Finally, I write the code for the main body of the script. First, we load the data from a text file. To do this, I call the read\_data\_from\_file function in the Processor class. This takes the file name, which we assigned to the variable shown. We also use the argument “lstTable” which represents our list of rows.

Then we display a menu of choices the user. This is done via a “while” loop which continues until broken out of. We first show the current data. To do so, I take the “print\_current\_Tasks\_in\_list” function from the IO class and use the list of rows as the argument. I then print the menu tasks from the IO class function and then prompt the user to enter a menu choice and assign that to the variable “strChoice”.

The next step is to process the user’s menu choice. I create an “if” statement for the first choice, along with three “elif” statements for choices 2-4 and an “else” statement for choice 5. If the user picks 1, which I identify by taking “strChoice”, stripping it and seeing if it equals 1, then I call the function to input a new task/priority from the IO class and assign the returns to two variables. Then I call the function to add data to the list using those two variables and our table of rows and assign its returns to two variables. Finally, I call the function to have the user press enter to continue with the argument for the status and continue.

If the user picks 2 to remove an existing task, I first call the function to get the user to input a task to remove in the IO class and assign its return to a variable. I then use the Processor function to remove data from the list using that variable and the list of rows and assign the returns to two variables. Once again we call the function to press enter to continue and use the str status argument and continue.

If the user picks 3 we want to save the data to a file. To do this we call the function to have the user input yes/no from the IO class using the argument of asking them if they want to save the data to a file. We capture the response in a variable. Then if that variable is “y”, we call the Processor function to write the data to a file using the file name and table. We capture those returns in two variables and then call the function in IO to input enter to continue using the argument for the status. Otherwise, we call the “enter to continue function” with the argument “save cancelled!” from the IO class and continue.

If the user picks 4 we want to reload the data from the file. To do this, we first print a message warning the user that unsaved data will be lost. Then I call the yes/no choice input function from IO and prompt the user if they’re sure they want to reload data from the file. I capture the return in a variable. Then if that matches “y” we call the read data from file Processor function using the file name and table and capture the returns in two variables. Then I call the press enter to continue function from IO and use the status as the argument. Otherwise, I call that function and say the reload was cancelled and continue.

Finally, if the user picks 5 we want to exit the program. We simply print “goodbye” and break from the loop. This is all shown in Fig 8:

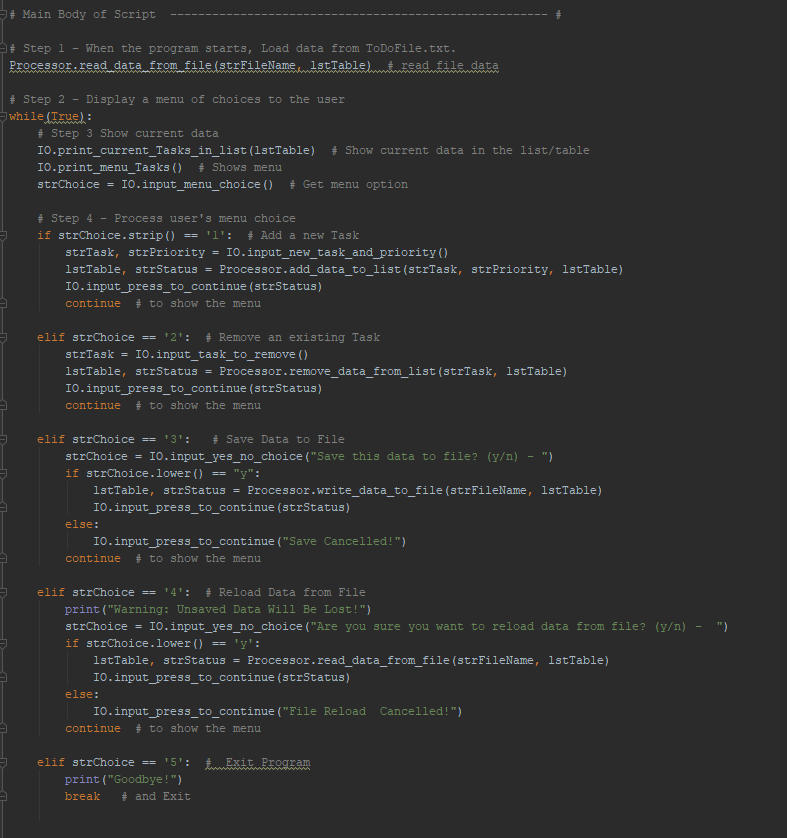


Figure : main body of script

## Results:

A sample run in PyCharm is shown in Fig 9-10. First I load the task “Laundry” and priority “low” from the existing text file. Then I add a task “homework” with “medium” priority and remove the “laundry” task. Then I save the data to the file, reload it and exit. Fig 11 shows the resulting text file.

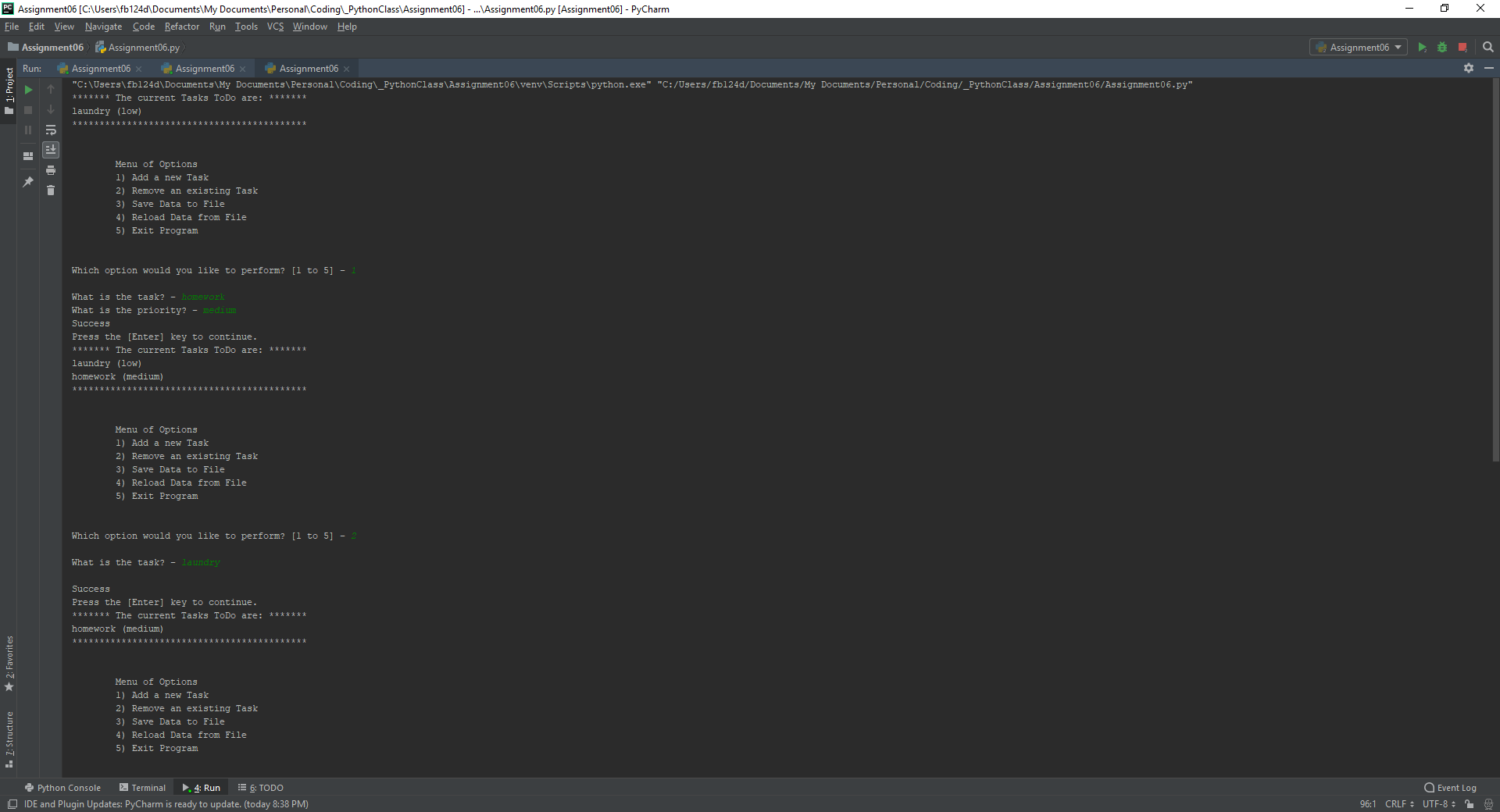


Figure : PyCharm Run

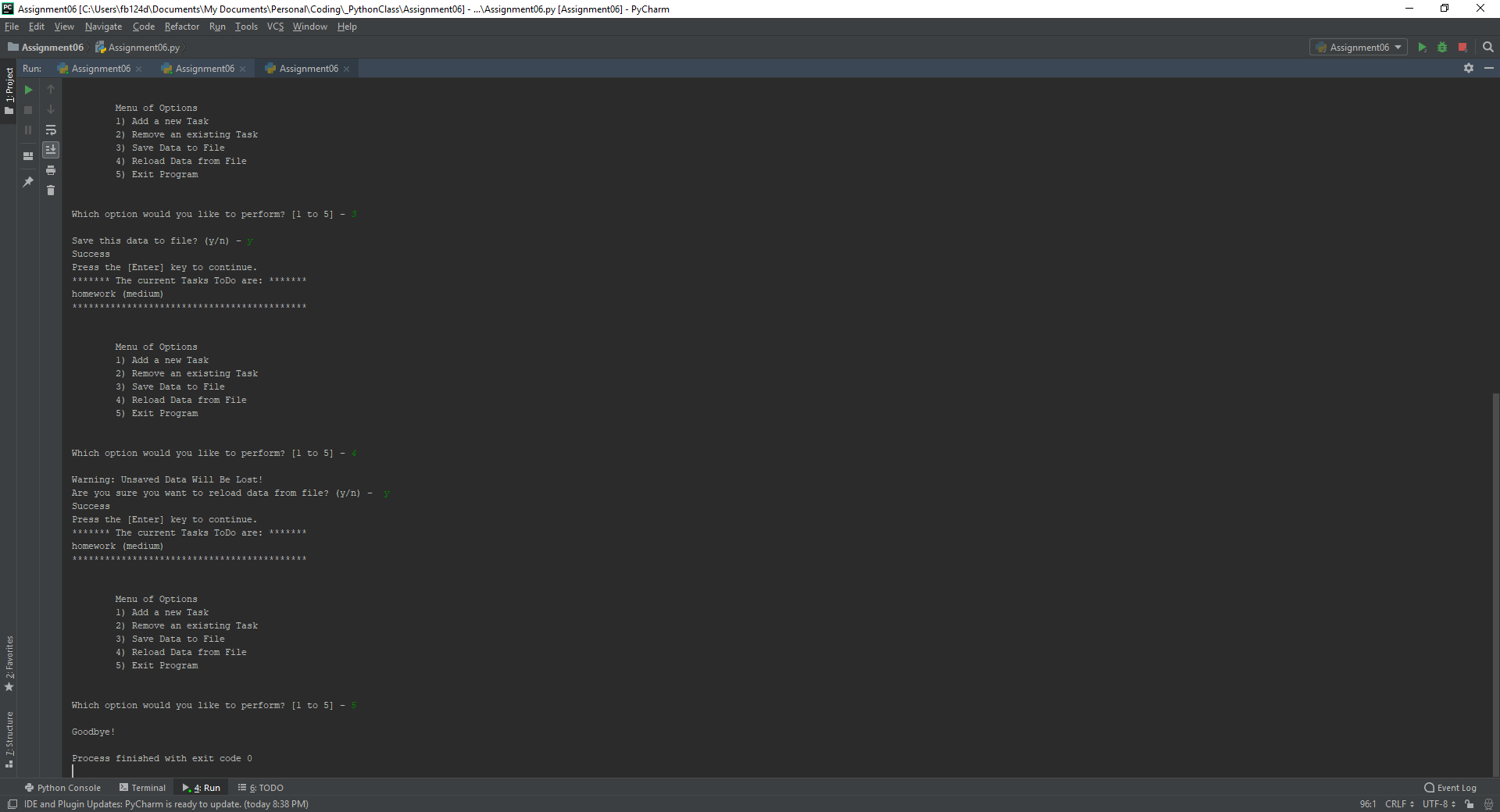


Figure : PyCharm Run continued

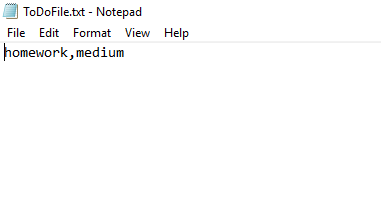


Figure : PyCharm resulting text file

Figures 12-13 shows a similar run in the Command Prompt. First I load data from the text file. Then I add a “hw” task with “high” priority. Then I remove the “laundry” task, save to the file, reload the data, and exit.



Figure : Command Prompt Run

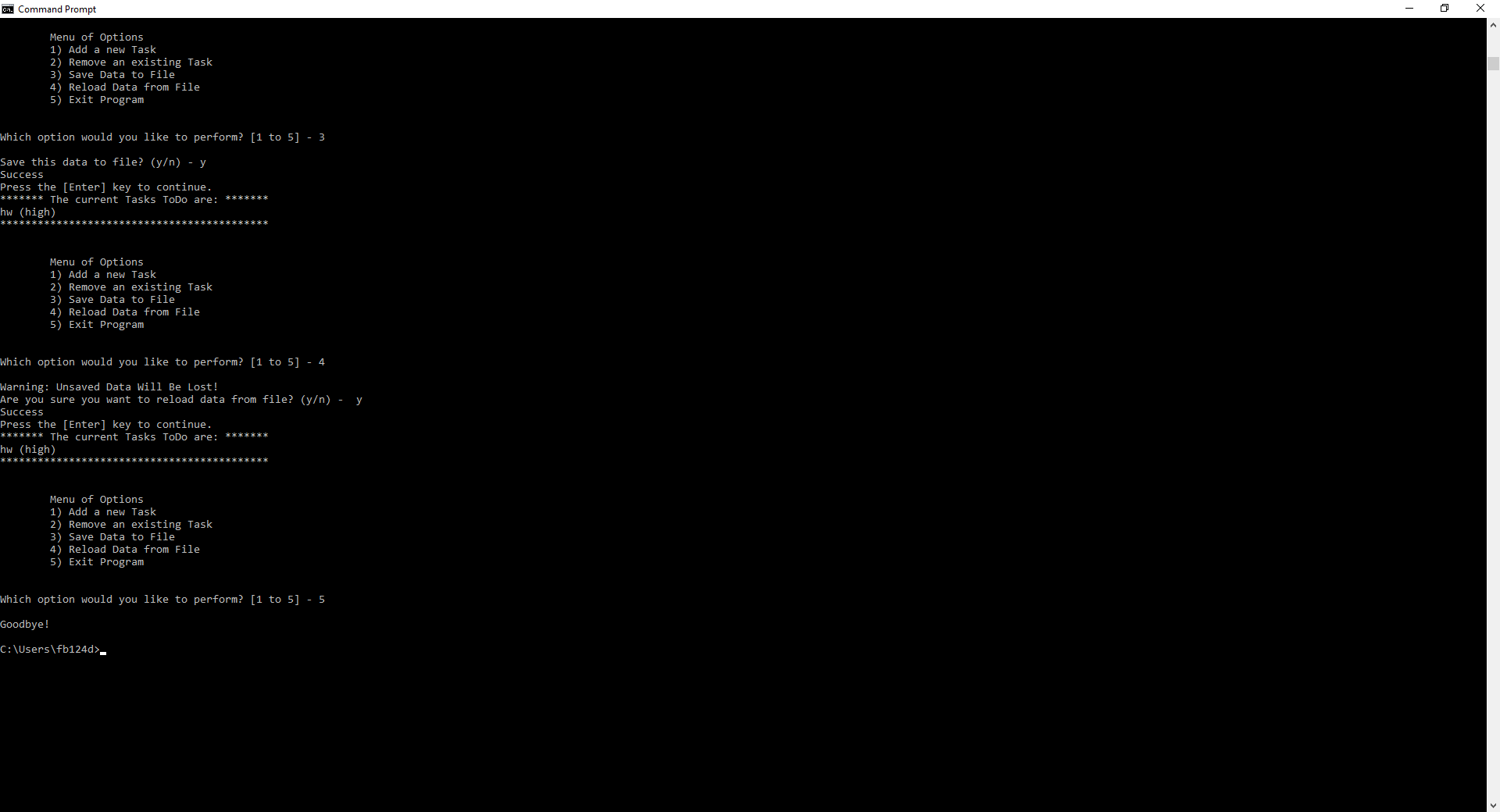


Figure : Command Prompt Run Continued

The resulting Command Prompt Text file is shown below in Fig 14:

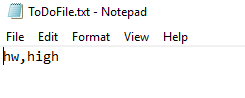


Figure : Resulting text file from Command Line run

## Conclusion

In this module I demonstrated how to create a script to read a to-do list from a text file, add data to it, remove data from it, save the new data to a text file, re-load that data, and exit the script using classes and functions. This helps keep the separation of concerns and makes for a more professional program. Finally, I showed how this looks in PyCharm and the Command Line.