# **Nazneen Mohammed Ameen**

# **Feb 29, 2020**

# **Programming with Python**

# **Assignment06**

**FUNCTIONS AND CLASS**

**INTRODUCTION**

Every program you’ve written so far has been one large, continuous series of instruction. Once your program reaches a certain size or level of complexity, it becomes hard to work with them this way. Fortunately, there are ways to break up big programs into smaller, manageable chunks of code. In this module you will learn one way of doing this by creating your own functions. you can put group of functions in class to be more manageable and organized. Functions work with global and local variables.

**Functions**

A function is a set of statements that take inputs, do some specific computation and produces output. The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs. Python provides built-in functions like print (), etc. but we can also create your own functions. These functions are called *user-defined functions*. The. To defining a function use word def with name of the function is shown below

**def function\_name():**

this line tells the computer that the block of code that follows is to be used together as the function. Example of function below.

|  |
| --- |
| # A simple Python function to check whether x is even or odd  def evenOdd( x ):      if (x % 2 == 0):          print "even"      else:          print "odd"    # Driver code  evenOdd(2)  evenOdd(3) |

**The output for this is:**

Even

odd

**Parameters and Arguments in a Function**

A parameter is a variable in a method definition. When a method is called, the arguments are the data you pass into the method's parameters. Parameter is variable in the declaration of function. Argument is the actual value of this variable that gets passed to function.

**Pass by Reference or Pass by Value?**

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is same as reference passing in Java. As shown in the example below

|  |
| --- |
| # Here x is a new reference to same list lst  def myFun(x):     x[0] = 20    # Driver Code (Note that lst is modified  # after function call.  lst = [10, 11, 12, 13, 14, 15]  myFun(lst);  print(lst) |

**Output: is modified**

[20, 11, 12, 13, 14, 15]

When we pass a reference and change the received reference to something else, the connection between passed and received parameter is broken. For example, consider below program.

|  |
| --- |
| def myFun(x):       # After below line link of x with previous     # object gets broken. A new object is assigned     # to x.     x = [20, 30, 40]    # Driver Code (Note that lst is not modified  # after function call.  lst = [10, 11, 12, 13, 14, 15]  myFun(lst);  print(lst) |

**Output: is not modified**

[10, 11, 12, 13, 14, 15]

|  |
| --- |
|  |

[**Default Arguments:**](https://www.geeksforgeeks.org/default-arguments-in-python/)

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

|  |
| --- |
| # Python program to demonstrate  # default arguments  def myFun(x, y=50):      print("x: ", x)      print("y: ", y)  # Driver code (We call myFun() with only  # argument)  myFun(10) |
|  |

**Output:**

('x: ', 10)

('y: ', 50)

**Keyword Arguments:**

The idea is to allow caller to specify argument name with values so that caller does not need to remember order of parameters, As shown below for example.

|  |
| --- |
| # Python program to demonstrate Keyword Arguments  def student(firstname, lastname):       print(firstname, lastname)      # Keyword arguments  student(firstname ='Geeks', lastname ='Practice')  student(lastname ='Practice', firstname ='Geeks') |
|  |

**Output:**

('Geeks', 'Practice')

('Geeks', 'Practice')

|  |
| --- |
|  |

**Global and Local Variables in Python**

Global variables are the one that are defined and declared outside a function and we need to use them inside a function. For example

|  |
| --- |
| # This function uses global variable s  def f():      print s    # Global scope  s = "I love Geeksforgeeks"  f() |

Output:

I love Geeksforgeeks

If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.

|  |
| --- |
| This function has a variable with  # name same as s.  def f():      s = "Me too."      print s    # Global scope  s = "I love Geeksforgeeks"  f()  print s |

Output:

Me too.

I love Geeksforgeeks.

The variable s is defined as the string “I love Geeksforgeeks”, before we call the function f(). The only statement in f() is the “print s” statement. As there is no local s, the value from the global s will be used. The question is, what will happen, if we change the value of s inside of the function f()? Will it affect the global s as well? We test it in the following piece of code:

|  |
| --- |
| def f():      print s        # This program will NOT show error      # if we comment below line.      s = "Me too."        print s    # Global scope  s = "I love Geeksforgeeks"  f()  print s |

Output:

Line 2: undefined: Error: local variable 's' referenced before assignment

To make the above program work, we need to use “global” keyword. We only need to use global keyword in a function if we want to do assignments / change them. global is not needed for printing and accessing. Why? Python “assumes” that we want a local variable due to the assignment to s inside of f (), so the first print statement throws this error message. Any variable which is changed or created inside of a function is local, if it hasn’t been declared as a global variable. To tell Python, that we want to use the global variable, we must use the keyword **“global”**, as can be seen in the following example: <https://www.geeksforgeeks.org/global-local-variables-python/>

|  |
| --- |
| This function modifies global variable 's'  def f():      global s      print s      s = "Look for Geeksforgeeks Python Section"      print s    # Global Scope  s = "Python is great!"  f()  print s |

Now there is no ambiguity.  
Output:

Python is great!

Look for Geeksforgeeks Python Section.

Look for Geeksforgeeks Python Section.

**Classes and Functions**

Classes are a way of grouping functions, variables, and constants.

**Class Method vs Static Method in Python**

**Class Method**

The @classmethod decorator, is a built-in [function decorator](https://www.geeksforgeeks.org/function-decorators-in-python-set-1-introduction/) that is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your function definition. A class method receives the class as implicit first argument, just like an instance method receives the instance.

**Syntax:**

**class C(object):**

**@classmethod**

**def fun(cls, arg1, arg2, ...):**

....

**fun:** function that needs to be converted into a class method

**returns:** a class method for function.

* A class method is a method which is bound to the class and not the object of the class.
* They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
* It can modify a class state that would apply across all the instances of the class. For example, it can modify a class variable that will be applicable to all the instances.

**Static Method**

A static method does not receive an implicit first argument.

**Syntax:**

**class C(object):**

**@staticmethod**

**def fun(arg1, arg2, ...):**

**...**

**returns:** a static method for function fun.

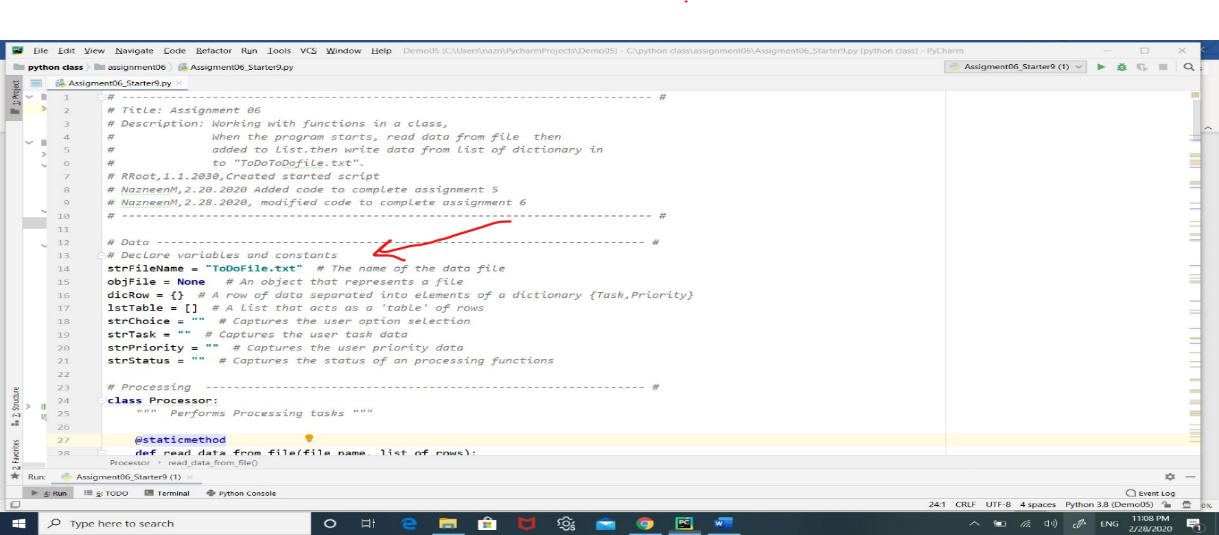
* A static method is also a method which is bound to the class and not the object of the class.
* A static method can’t access or modify class state.
* It is present in a class because it makes sense for the method to be present in class.

**Class method vs Static Method**

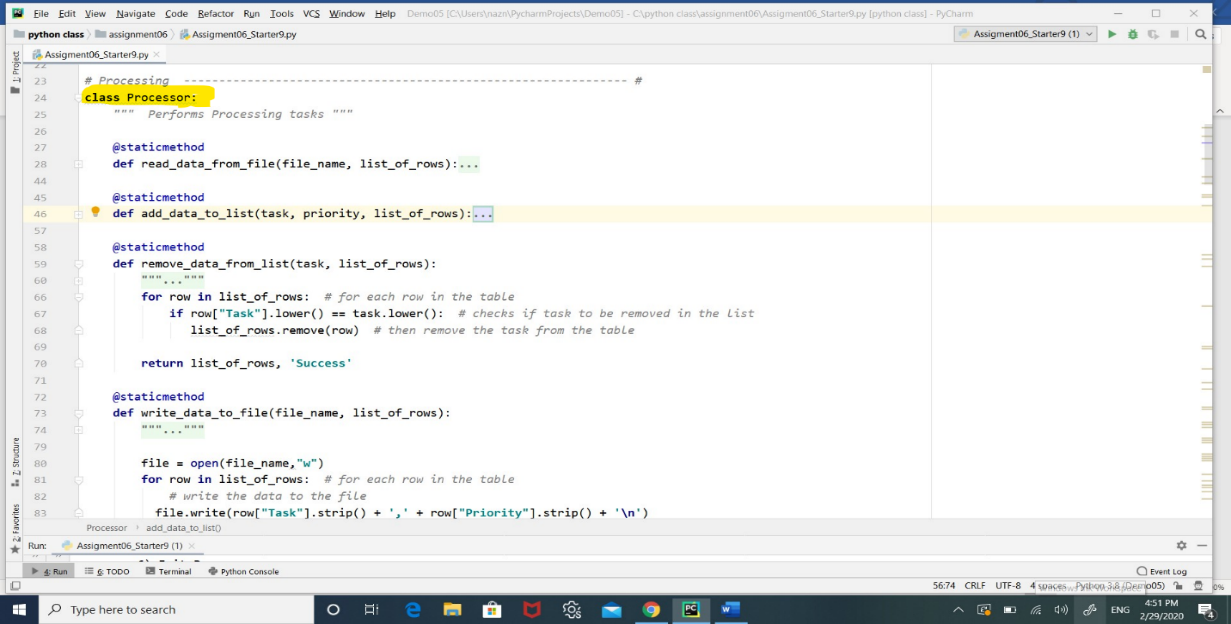
* A class method takes cls as first parameter while a static method needs no specific parameters.
* A class method can access or modify class state while a static method can’t access or modify it.
* In general, static methods know nothing about class state. They are utility type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as parameter.
* We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

<https://www.geeksforgeeks.org/python-classes-and-objects/>

The code below example for class of functions which is used @statisticmethod for doing some processing in the( tasks and its priority) .in this example there is menu of option to (display current task and priority [high/low]), where read data from TODofile.txt then add task, remove task, save changes in to file(ToDoFile.txt), reload data again from file (ToDoFile.txt) and then exit . I divided this code to data, processor and presentation section to be more organized and easier to figure mistakes. the first part is declaration part (Data). Define variables, lists of table, dictionary and file. As shown in figure 6-1.

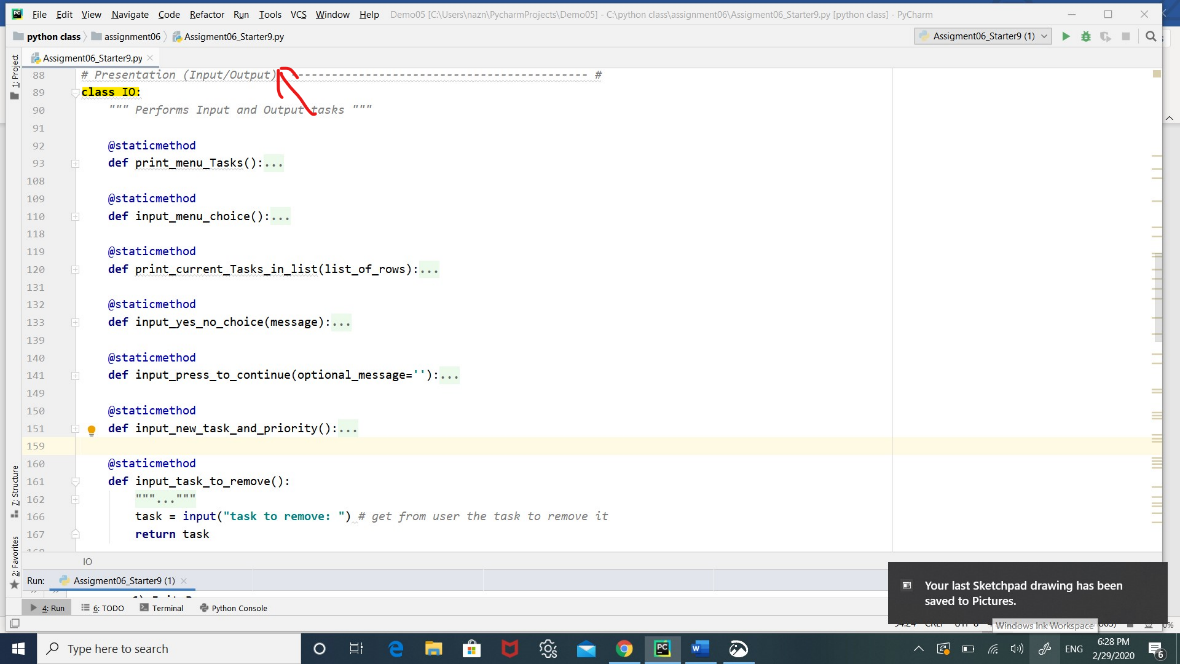
****

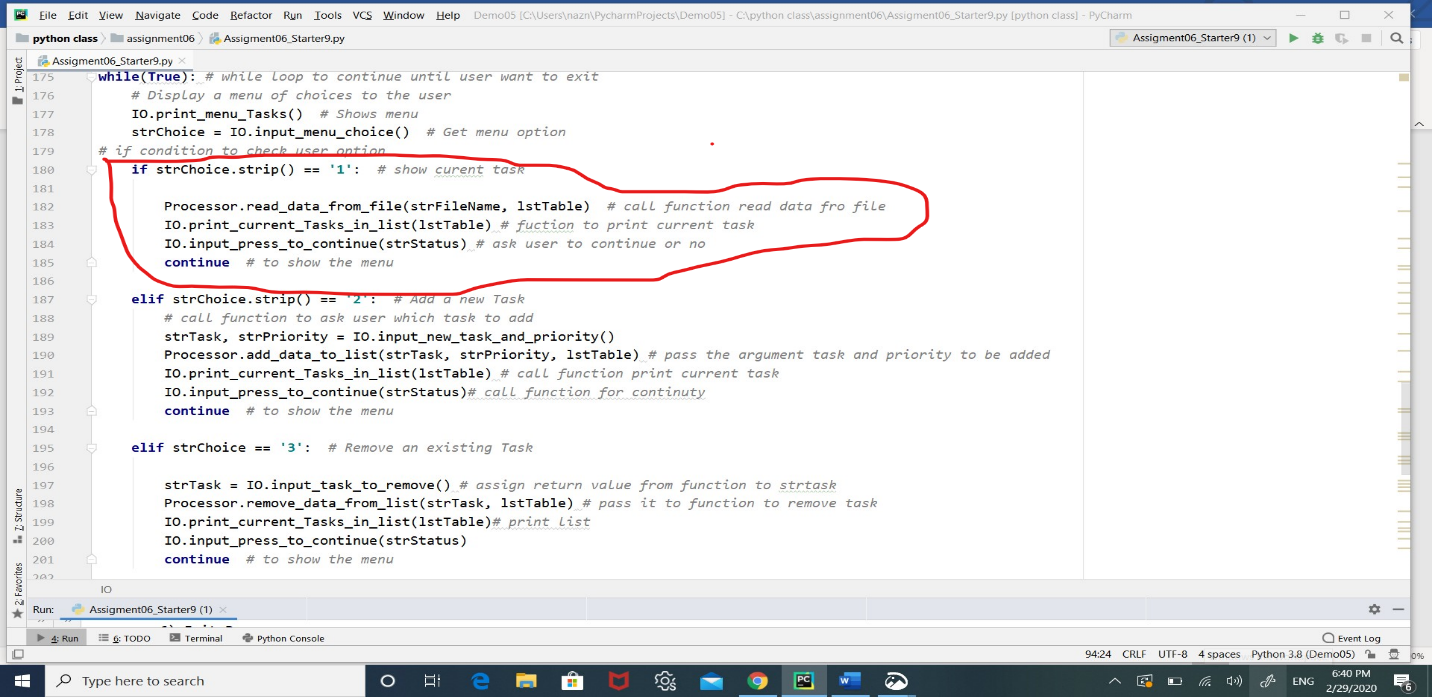
**Figure 6-1 shows declaration of variables.**

Then I grouped functions in classes , first class which is the processor class includes functions that processing data like (read\_data\_from\_file, add\_data\_to\_list, remove \_ data \_from\_list, write \_data\_to\_file) as shown below in figure 6-2

**Figure 6-2 shows processor class**

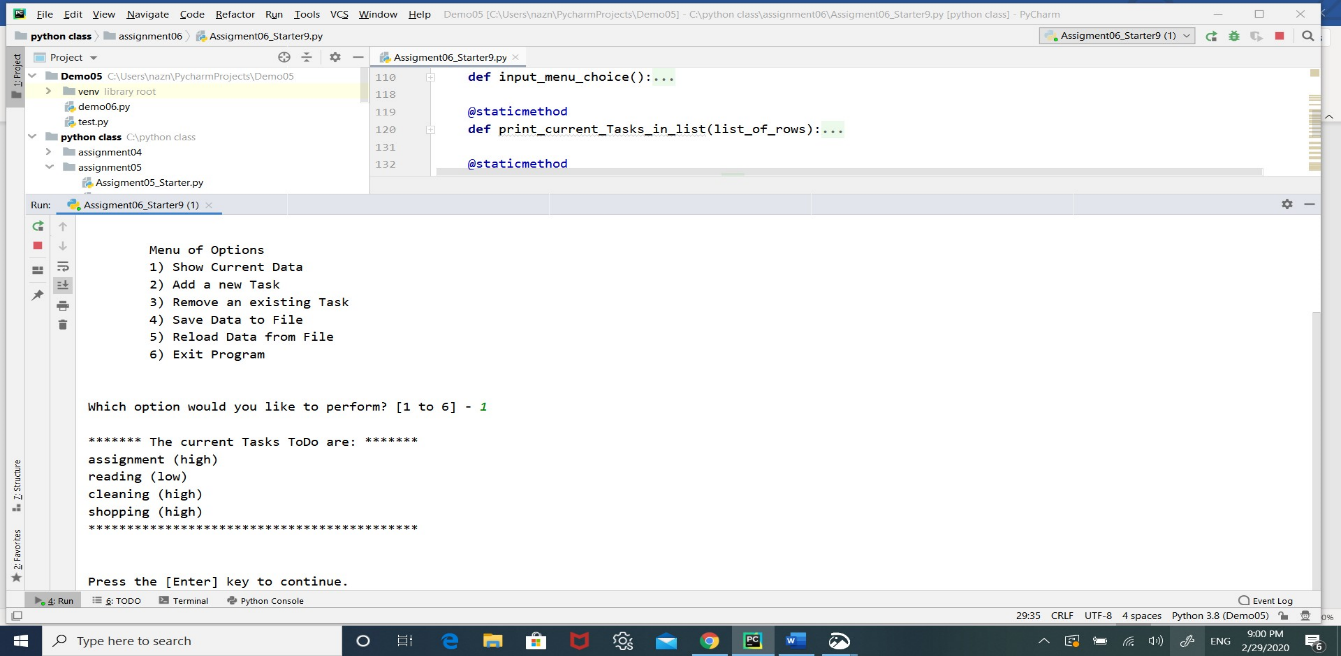
In the presentation part I put (IO class) to include functions that prints the main menu function, and so on, or functions that need users input . as shown in figure 6-3.

 **Figure 6-3 shows presentation section includes class IO**

In the main body I put while loop for menu choices and for each choice I call the functions that need to perform this choice. some functions called more than one time this is a good idea to use function.as shown below in figure 6-4 for example the part that circled with red, shows the functions called to show the current task.

**figure 6-4 main body part calling functions to do specific task**

In the figure 6-4 above after while (true), I call function print\_menu\_task() to display the menu of options for user and then I called function (input\_menu\_choice) and I assigned the result of the function in to StrChoice variable. Capturing the result in a variable allows you to use the variables of result multiple times without having to call function again.

In the figure 6-5 below shows the output in case the user choose option 1.

**Figure 6-5 shows the output for choice 1**

So for each choice I called functions that related to the choice, below is the scripts of the code in the main body .

*Main Body of Script -------------------------------#  
  
# Step 1 - When the program starts, Load data from ToDoFile.txt.*Processor.read\_data\_from\_file(strFileName,lstTable)  
IO.print\_current\_Tasks\_in\_list(lstTable) *# Show current data in the list/table***while**(**True**): *# while loop to continue until user want to exit  
 # Display a menu of choices to the user* IO.print\_menu\_Tasks() *# Shows menu* strChoice = IO.input\_menu\_choice() *# Get menu option  
# if condition to check user option* **if** strChoice.strip() == **'1'**: *# show curent task* Processor.read\_data\_from\_file(strFileName, lstTable) *# call function read data fro file* IO.print\_current\_Tasks\_in\_list(lstTable) *# fuction to print current task* IO.input\_press\_to\_continue(strStatus) *# ask user to continue or no* **continue** *# to show the menu* **elif** strChoice.strip() == **'2'**: *# Add a new Task  
 # call function to ask user which task to add* strTask, strPriority = IO.input\_new\_task\_and\_priority()  
 Processor.add\_data\_to\_list(strTask, strPriority, lstTable) *# pass the argument task and priority to be added* IO.print\_current\_Tasks\_in\_list(lstTable) *# call function print current task* IO.input\_press\_to\_continue(strStatus)*# call function for continuty* **continue** *# to show the menu* **elif** strChoice == **'3'**: *# Remove an existing Task* strTask = IO.input\_task\_to\_remove() *# assign return value from function to strtask* Processor.remove\_data\_from\_list(strTask, lstTable) *# pass it to function to remove task* IO.print\_current\_Tasks\_in\_list(lstTable)*# print list* IO.input\_press\_to\_continue(strStatus)  
 **continue** *# to show the menu* **elif** strChoice == **'4'**: *# Save Data to File* strChoice = IO.input\_yes\_no\_choice(**"Save this data to file? (y/n) - "**) *# make sure to user if want to save* **if** strChoice.lower() == **"y"**: *# if answer y* Processor.write\_data\_to\_file(strFileName, lstTable) *# call function write to file* IO.input\_press\_to\_continue(strStatus) *# ask for continue* **else**:  
 IO.input\_press\_to\_continue(**"Save Cancelled!"**)  
 **continue** *# to show the menu* **elif** strChoice == **'5'**: *# Reload Data from File* print(**"Warning: Unsaved Data Will Be Lost!"**) *# notice to user* strChoice = IO.input\_yes\_no\_choice(**"Are you sure you want to reload data from file? (y/n) - "**)  
 **if** strChoice.lower() == **'y'**:  
 Processor.read\_data\_from\_file(strFileName,lstTable) *# call function first to read data from file  
 # return list of table* IO.print\_current\_Tasks\_in\_list(lstTable) *# print it so user can see it* IO.input\_press\_to\_continue(strStatus) *# ask for continue* **else**:  
 IO.input\_press\_to\_continue(**"File Reload Cancelled!"**) *# else reload cancel* **continue** *# to show the menu1* **elif** strChoice == **'6'**: *# Exit Program* print(**"Goodbye!"**)  
 **break** *# and Exit*

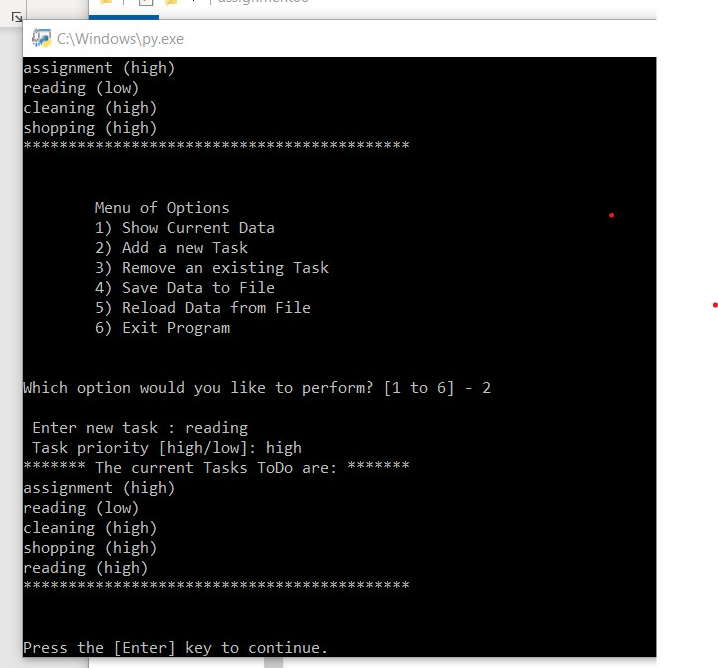
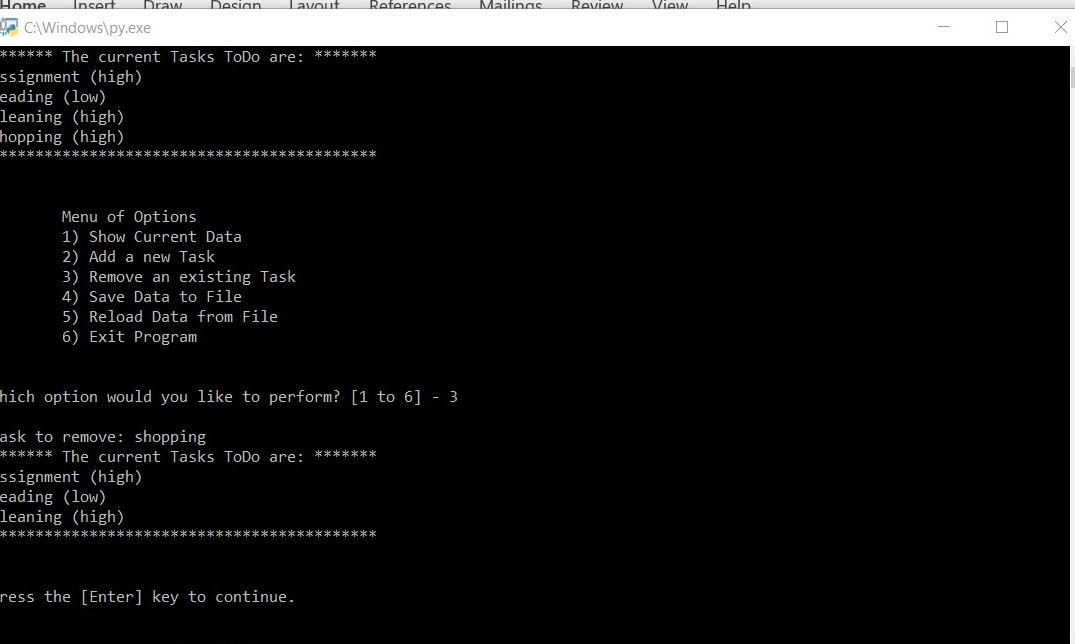
If user choose 2 that add new task and priority as showed in above script the output will be as shown if figure 6-6.

Figure 6-6 when adding task

If the user wants to remove item then should choose option 4 and the output for it shown below in fig 6-7

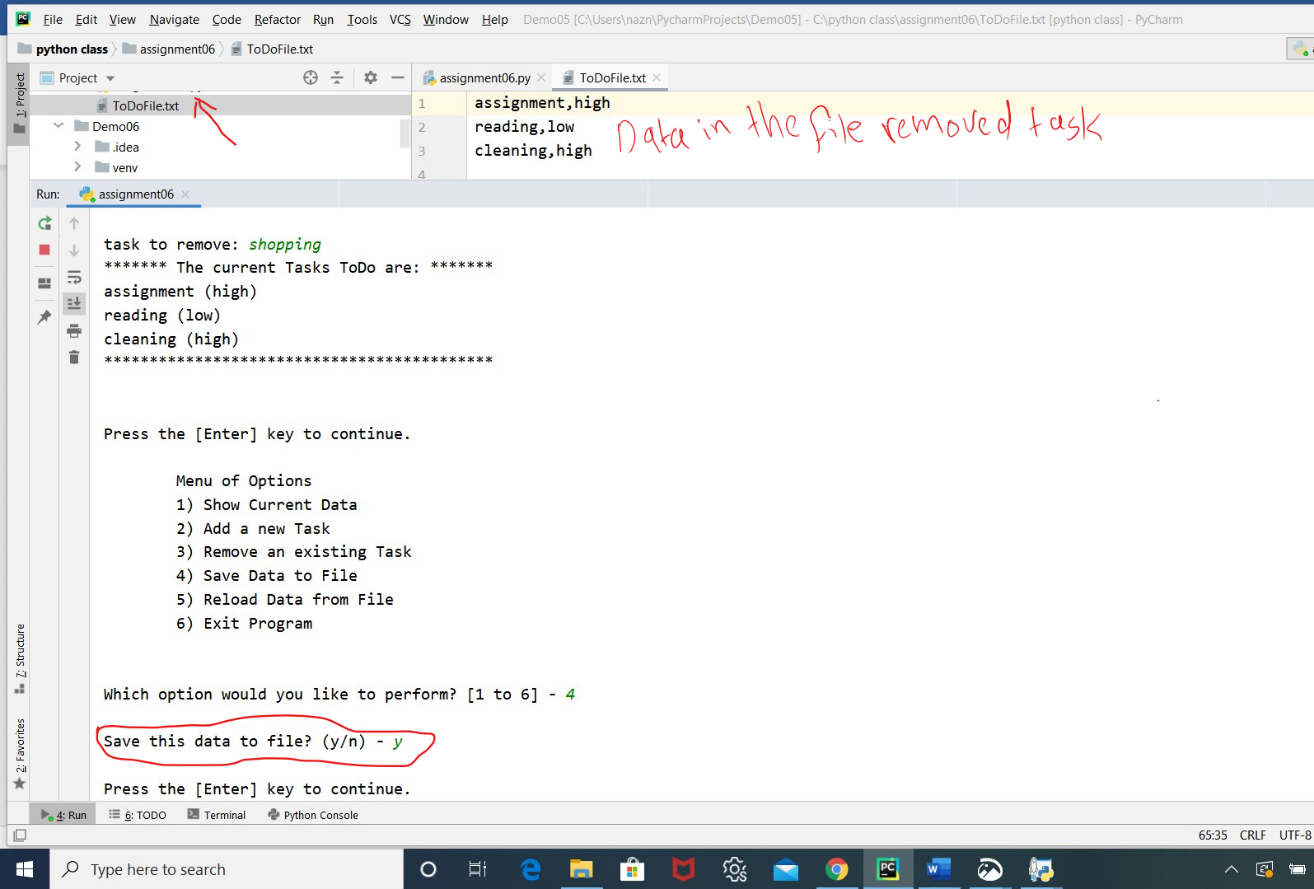


**Figure 6-7**

**Shows**

**Remove task**

These changes that made to the tasks is stored in memory and will go just by existing the task menu. So, to save the changes we should save the list of table to a file again this can done by choose option number 4 to save data to file.in the figure 6-8 shows how to save to file after removing a task.

Figure 6-8 shows the file contents after changing.

To declare how to save changes in the code I used a global variable which is lstTable to save the list of row in it .any change happened in the each part of the program will save to lstTable then I passed it to function print data to file then this function will write the data to the file. below the script code for the rest of the program.

*# 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  
  
# Processing --------------------------------------------------------------- #***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"**) *# read data from text file* **for** line **in** file: *# for each line in the file* task, priority = line.split(**","**) *# look for ',' end of task then assign it  
 # to task the other word to priority.* dicRow = {**"Task"**: task. strip(), **"Priority"**: priority.strip()} *# put it in dictionary* list\_of\_rows.append(dicRow) *# make list of dictionary* file.close() *# close the file* **return** list\_of\_rows, **'Success'** *# this function return list of dictionary* @staticmethod  
 **def** add\_data\_to\_list(task, priority, list\_of\_rows):  
  
 *""" add task and priority to list of row* **:param** *task: task from user to be added* **:param** *priority: priority of task to be added to list/Table* **:param** *list\_of\_rows: list of table that new task add to it* **:return***: list of Row after adding  
 """* row = {**"Task"**: task.strip(), **"Priority"**: priority.strip()} *# put new task and priority in dic row* list\_of\_rows.append(row) *# add new dic. to end of the list* **return** list\_of\_rows, **'success'** *# return updated lis of dictionary* @staticmethod  
 **def** remove\_data\_from\_list(task, list\_of\_rows):  
 *""" remove task from list of row* **:param** *task: task to be removed* **:param** *list\_of\_rows:list of Table that task removed from* **:return***: list of dic after task removed  
 """* **for** row **in** list\_of\_rows: *# for each row in the table* **if** row[**"Task"**].lower() == task.lower(): *# checks if task to be removed in the list* list\_of\_rows.remove(row) *# then remove the task from the table* **return** list\_of\_rows, **'Success'** @staticmethod  
 **def** write\_data\_to\_file(file\_name, list\_of\_rows):  
 *""" save data to the file* **:param** *file\_name: file to save data in it* **:param** *list\_of\_rows: list of table to save it in file\_name* **:return***:  
 """* file = open(file\_name,**"w"**)  
 **for** row **in** list\_of\_rows: *# for each row in the table  
 # write the data to the file* file.write(row[**"Task"**].strip() + **','** + row[**"Priority"**].strip() + **'\n'**)  
 file.close()  
  
  
*# 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) Show Current Data  
 2) Add a new Task  
 3) Remove an existing Task  
 4) Save Data to File   
 5) Reload Data from File  
 6) Exit Program  
 '''**)  
 print() *# Add an extra line for looks* @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 6] - "**)).strip() *# get choice from user* print() *# Add an extra line for looks* **return** choice *# return user choice* @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* @staticmethod  
 **def** input\_yes\_no\_choice(message):  
 *""" Gets a yes or no choice from the user* **:return***: string  
 """* **return** str(input(message)).strip().lower() *# return string* @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.'**)  
  
 @staticmethod  
 **def** input\_new\_task\_and\_priority():  
 *""" gets task and priority from user* **:return***:task and priority to be added  
 """* task = input(**" Enter new task : "**) *# get from user task to be added* priority = input(**" Task priority [high/low]: "**)*# get priority from user* **return** task, priority  
  
 @staticmethod  
 **def** input\_task\_to\_remove():  
 *""" gets task from user* **:return***: task to be removed  
 """* task = input(**"task to remove: "**) *# get from user the task to remove it* **return** task

Summary

In this module you learned about function which is away of grouping one or more statements. And learned about parameters and arguments. Also you got idea about class and how is important to bound group of function by using @staststicmethod. Finally you saw how the function important to break part a big code .