Tsedey Tadesse
Nov 23, 2022
IT FND 110 A AU 22
Assignment06
GitHub Link: https://github.com/tsedeytadesse/IntroToProg-Python-Mod06

Functions and Classes

Introduction

In this assignment, I have learned how large scripts can be broken down and be organized into smaller sections by creating and defining functions. Functions allow us to group statements by the tasks we want to perform and later be called in the main script to output what we want. Functions themself can be grouped into sections by creating classes. For this assignment, I modified the starter script provided in class, which focuses mainly on the use of functions and classes when writing scripts. I also expired debugging tools used in Pycharm by setting breakpoints and debug my scripts.

Functions

Functions allow us to break down a large block of script into smaller and manageable blocks. Creating and defining a function is an excellent way to write and understand scripts, especially if more than one person is working on the script. Functions can receive values through parameters and arguments.. Functions use parameters to pass values into the functions for processing, and they use arguments to receive values that are passed on to them. Functions return values by using the *return* statement then the program ends. (source: https://www.learnpython.org/en/Functions). ***Figure 1*** shows how functions are defined in Python. The parameters, value1 and value2, would later have argument values passed to them in the main body of the script. The function would return the value for "addition" when we call the function in the main body.

```
def simple_math(value1, value2):
    addition = value1 + value2
    return addition


addition = simple_math(1, 6)
print(addition)
```

Figure1*: defining functions in Pycharm*

Encapsulation of Functions

Functions are encapsulated, meaning that "no variable [we] create in a function, including its parameters, can be directly accessed outside its function." (Dawson, Michael, *Python Programming for the complete beginner, Third Edition*, Course Technology, 2010). The parameters , the return values, and global variables are the only ways of communication to the main body when the function is called in the main body. This makes the whole process of defining a function worthwhile since we don't have to worry about keeping up with all the variables in the whole script.

Global and Local Variables

Functions can have variables that are either global or local. As the names imply, global variables are variables that can be used in a function as well as outside a function. Local variables only exist within a function. The usage of local variables is preferred for a reason that we don't have to keep track of all the variables in the script. Using global variables put us at risk of creating shadow variables, which are if a global and a local variable both have the same names, the local variable shadows the global variable and the values assigned to the global variable may never be used.

Lab 6-1

Figure 3 is a script for Lab 6-1 of Module06. Note that the return statement is not used in this lab. Hence, the print() function is used within the function. For the function to output the result that we want, it must be called outside of the function. Functions don't output values. Figure 2 shows the result of the script in Figure 3.

```
/Users/yilikalademe/Documents/_PythonClass/Module06/pythonProject1/bin/python /U
The sum of the values is: 15
The difference of the values is: 5
The product of the values is: 50
The quotient of the values is: 2.0

Process finished with exit code 0
```

***Figure 2**: result of Lab 6-1*

```
# ------------------------------------------------------------------------ #
# Title: Lab6-1
# Description: Creating a function to do simple math
# ChangeLog (Who,When,What):
# TTadesse,11.20.2022,Created started script
# ------------------------------------------------------------------------ #

# Define the function
def simple_math(value1, value2):
    adding = value1 + value2
    subtracting = value1 - value2
    multiplying = value1 * value2
    dividing = value1 / value2

    print("The sum of the values is: " + str(adding))
    print("The difference of the values is: " + str(subtracting))
    print("The product of the values is: " + str(multiplying))
    print("The quotient of the values is: " + str(dividing))


# Call the function
simple_math(10, 5)
```

***Figure 3**: Lab 6-1*

Lab 6-2

Return values can have multiple values in them and this can be expressed as tuples. The tuples can be packed and unpacked to put together all the values and access them individually, respectively. As shown in the script in ***Figure 5***, the parameters can have argument values directly from the using through the input() function. ***Figure 4*** shows the result of the lab 6-2.

```
/Users/yilikalademe/Documents/_PythonClass/Module06/pythonProject1/bin/pyt
Enter Value 1: 10
Enter Value 2: 5
The sum of 10.00 and 5.00 is 15.00
The difference of 10.00 and 5.00 is 5.00
The product of 10.00 and 5.00 is 50.00
The quotient of 10.00 and 5.00 is 2.00

Process finished with exit code 0
```

***Figure 4**: result of Lab 6-2*

```python
# ------------------------------------------------------------------------ #
# Title: Lab6-2
# Description: Creating a function to do simple math
# ChangeLog (Who,When,What):
# TTadesse,11.20.2022,Created started script
# ------------------------------------------------------------------------ #

# --data code-- #
v1 = None  # first argument
v2 = None  # first argument
result1 = None  # first result of processing
result2 = None  # second result of processing
result3 = None  # third result of processing

# -- Define the function-- #


def simple_math(value1, value2):
    adding = value1 + value2
    subtracting = value1 - value2
    multiplying = value1 * value2
    dividing = value1 / value2
    all_results = adding, subtracting, multiplying, dividing  # create a tuple of all operations
    return value1, value2, all_results  # pack tuple


# -- Presenting (I/O) code-- #
v1 = float(input("Enter Value 1: "))
v2 = float(input("Enter Value 2: "))
result1, result2, result3 = simple_math(v1, v2)  # unpack tuple
print("The sum of %.2f and %.2f is %.2f  " % (result1, result2, result3[0]))
print("The difference of %.2f and %.2f is %.2f  " % (result1, result2, result3[1]))
print("The product of %.2f and %.2f is %.2f  " % (result1, result2, result3[2]))
print("The quotient of %.2f and %.2f is %.2f  " % (result1, result2, result3[3]))
```

***Figure 5**: Lab 6-2*

Classes

Classes have the same way of breaking down and organizing parts of scripts as functions do, but classes break down and group functions. Large scripts are expected to have so many functions that the functions themselves need to be categorized and grouped. This way, we can create as many functions as we want with as little as one thing to do in that function. Lab 6-3 and assignment 06 would show how classes are used in Python.

Lab 6-3

Lab 6-3 is the same exact lab as 6-2, except that each of the statements of operations that were within one function are broken down to have their own functions and these functions are grouped in a class. See ***Figure 6*** to see the whole script. The result of lab 6-3 is also the same as the result of lab -2 shown in ***Figure 4***.

```
# ------------------------------------------------------------------------ #
# Title: Lab6-3
# Description: Creating a function to do simple math
# ChangeLog (Who,When,What):
# TTadesse,11.21.2022,Created started script
# ------------------------------------------------------------------------ #

# --data code-- #
v1 = None  # first argument
v2 = None  # first argument

# -- Define the class and the functions-- #

class MathProcessor():
    """ Functions to process simple math """

    @staticmethod
    def AddValue(value1=0.0, value2=0.0):
        """ This function adds two values

        :param value1: (float) the first number to add
        :param value2: (float) the second number to add
        return: (float) sum of the two numbers
        """
        adding = float(value1 + value2)
        return adding

    @staticmethod
    def SubtractValue(value1=0.0, value2=0.0):
        """ This function subtracts two values

        :param value1: (float) the first number to subtract
        :param value2: (float) the second number to subtract
        return: (float) difference of the two numbers
        """
```

Figure 6-1: *Lab 6-3 script*

```
        subtracting = float(value1 - value2)
        return subtracting

    @staticmethod
    def MultiplyValue(value1=0.0, value2=0.0):
        """ This function multiplies two values

        :param value1: (float) the first number to multiply
        :param value2: (float) the second number to multiply
        return: (float) product of the two numbers
        """

        multiplying = float(value1 * value2)
        return multiplying

    @staticmethod
    def DivideValue(value1=0.0, value2=0.0):
        """ This function divides two values

        :param value1: (float) the first number to divide
        :param value2: (float) the second number to divide
        return: (float) quotient of the two numbers
        """

        dividing = float(value1 / value2)
        return dividing

# -- Presenting (I/O) code-- #
v1 = float(input("Enter Value 1: "))
v2 = float(input("Enter Value 2: "))

print("The sum of %.2f and %.2f is %.2f  " % (v1, v2, MathProcessor.AddValue(v1, v2)))
print("The difference of %.2f and %.2f is %.2f  " % (v1, v2, MathProcessor.SubtractValue(v1, v2)))
print("The product of %.2f and %.2f is %.2f  " % (v1, v2, MathProcessor.MultiplyValue(v1, v2)))
print("The quotient of %.2f and %.2f is %.2f  " % (v1, v2, MathProcessor.DivideValue(v1, v2)))
```

***Figure 6-2**: Lab 6-3 script*

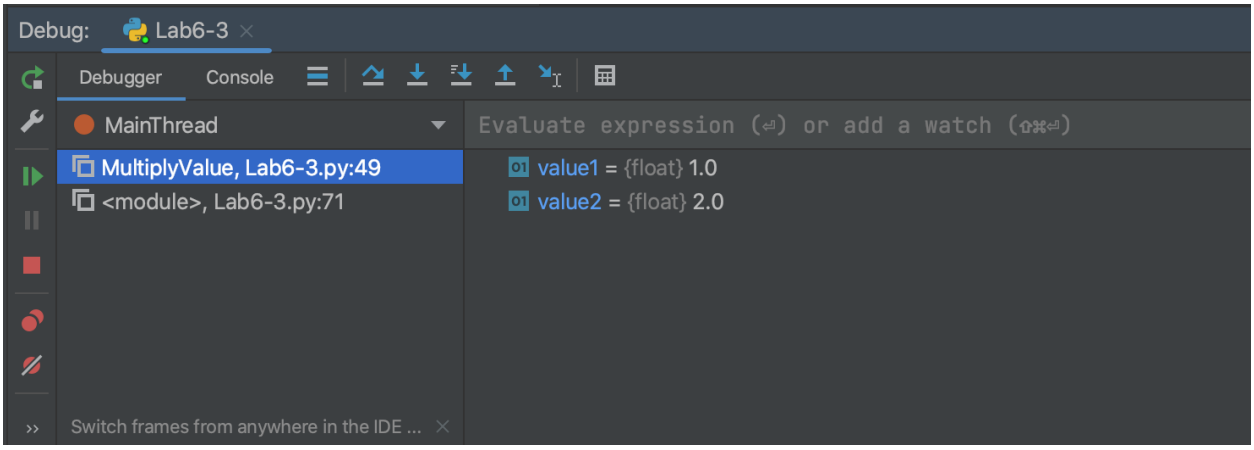
Functions in a class are called in the main body of the script by using the class name followed by a period followed by the name of the function.

In lab 6-3, I have also learned about adding document strings (docstrings) after defining functions. Docstrings are headers that explain what that specific function is doing. It is also a good practice to explain what all the parameters and the return statement are for in each function.

PyCharm Debugger

Another benefit of grouping statements in functions is that it makes it easier to debug the codes. Pycharm has its own debugging tool. Codes can be debugged by placing breakpoints where we want to start the debugging process. When we run our code in debug mode in

Pycharm, we are allowed to have two displays, one for the debugger, and one for the console. See ***Figure 7*** below to see what the debugger display looks like. The different styles of arrow buttons allow are to navigate through the section of the code we are



debugging as well as to step into the script and display what's on the console.

***Figure 7**: debugger + console in PyCharm*

Assignment06

Assignment06 prompts me to modify a script that was provided in class. The assignment is to modify a "ToDo List" by adding to the list, removing tasks from the list, and writing and reading it in a text file. The script is sectioned as data, processing, presentation and main body. Assignment06 is very similar in nature as assignment05, except that assignment06 requires the use of classes and functions to group and organize the script.

Data

The data section declares variables and constants that would later be used in the main body of the script. ***Figure 8*** below shows the data section of the assignment. along with the header for the script.

```
# Data ---------------------------------------------------------------------- #
# Declare variables and constants
file_name_str = "ToDoFile.txt"  # The name of the data file
file_obj = None  # An object that represents a file
row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection
```

***Figure 8**: assignment06 script - Data*

Processing

The processing section of the code is the largest section of the code, where codes for adding, deleting, writing, and reading data are written in different functions. This section has one class and four functions under the class. Each function is assigned a task for adding, deleting, writing, or reading data. This makes the whole script very accessible and easy to modify or debug. Each function also has docstring along with the description of all the parameters and the return values. ***Figure 9*** shows the processing section of the script.

```
# 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")
        for line in file:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):
        """ Adds data to a list of dictionary rows

        :param task: (string) with name of task:
        :param priority: (string) with name of priority:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
        # TODO: Added Code Here!
        list_of_rows.append(row)
        return list_of_rows
```

Figure 9-1*: assignment06 script - Processing*

```python
@staticmethod
def remove_data_from_list(task, list_of_rows):
    """ Removes data from a list of dictionary rows

    :param task: (string) with name of task:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    # TODO: Added Code Here!
    for row in list_of_rows:
        if row["Task"] == task:
            list_of_rows.remove(row)
            task = True
    if task == True:
        print("your is deleted from your Task list")
    else:
        print("Task not found")
    return list_of_rows

@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    # TODO: Added Code Here!
    file = open(file_name, 'w')
    for line in list_of_rows:
        file.write(line["Task"] + ',' + line["Priority"] + '\n')
    return list_of_rows
```

***Figure 9-2**: assignment06 script - Processing*

The modifications that I performed are shown under the comment "TODO: Added Code Here." The first thing I modified is in the function, add_data_to_list(), where I was provided with a dictionary of tasks and their priorities. My task was to append the dictionaries to a list of rows and return the lists of rows. See ***Figure 9-1*** to see the lines of code. My next task was to modify the remove_data_from_file() function. I write the lines of code for this task by using the *if* statement. I run into a problem that the *else* statement doesn't output "Task not found" when the task input to remove is not contained in any of the rows in the table. With the help of Assignment05 solution, I learned that nesting another *if* statement under the *for* loop solves my problem. See ***Figure 9-2*** for the lines of code. My next task was to modify the function, write_data_to_file(). I performed this task by using the open() funcion to access the text file and write() the new lines of data by using a *for* loop.

Presentation (I/O)

The presentation section of the script is embodied with a class that contains five functions to perform the task of input and output. See ***Figure 10*** for this section. This section would perform tasks like outputting the menu section, asking user input, and displaying written data.

```
# Presentation (Input/Output)  ---------------------------------------- #


class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def output_menu_tasks():
        """  Display a menu of choices to the user

        :return: nothing
        """
        print('''
        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) 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 4] - ")).strip()
        print()  # Add an extra line for looks
        return choice
```

***Figure 10-1**: assignment06 script - Presentation (I/O)*

```python
@staticmethod
def output_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_new_task_and_priority():
    """  Gets task and priority values to be added to the list

    :return: (string, string) with task and priority
    """

    pass  # TODO: Added Code Here!
    task = str(input("Enter the name of your task: "))
    priority = str(input("What is the priority for your task: 'h' for high or 'l' for low: "))
    return task, priority

@staticmethod
def input_task_to_remove():
    """  Gets the task name to be removed from the list


    :return: (string) with task
    """
    pass  # TODO: Add Code Here!
    task = str(input("Which Task would you like to remove? "))
    return task
```

***Figure 10-2**: assignment06 script - Presentation (I/O)*

The modifications that I performed for this section of the code are taking user inputs for the input_new_task_and_priority() and input_task_to_remove() functions. ***Figure 10-2*** shows the lines of codes I added to the functions.

Main Body of Script

The main body of the script is where we perform to actually output data on the console. The main body uses a *while* loop and *if-elif* statements to navigate through the menu of options for adding, removing, writing, and reading data. See ***Figure 11*** for this section of the code.

```
# Main Body of Script  ------------------------------------------------------ #

# Step 1 - When the program starts, Load data from ToDoFile.txt.
Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst)  # read file data

# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    # Step 4 - Process user's menu choice
    if choice_str.strip() == '1':  # Add a new Task
        task, priority = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '2':  # Remove an existing Task
        task = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '3':  # Save Data to File
        table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
        print("Data Saved!")
        continue  # to show the menu

    elif choice_str == '4':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop
```

Figure 11*: assignment06 script - Main Body*

As the main body of the script shows, the parameters of each function take in arguments (represented by the variables declared in the data section). This is simply achieved by using '=' sign to equate the parameters to the argument.

Assignment06 Result in Pycharm

Figure 12 shows the result of assignment06 run in Pycharm.

```
/Users/yilikalademe/Documents/_PythonClass/Assignment06/bin/python /Users/yilikalademe/Documents/_PythonClass/Ass
******* The current tasks ToDo are: *******
read a book (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program


Which option would you like to perform? [1 to 4] - 1

Enter the name of your task: cook
What is the priority for your task: 'h' for high or 'l' for low: h
******* The current tasks ToDo are: *******
read a book (h)
cook (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program


Which option would you like to perform? [1 to 4] - 2

Which Task would you like to remove? cook
your is deleted from your Task list
******* The current tasks ToDo are: *******
read a book (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program
```

Figure 12-1*: result of assignment06 script on PyCharm*

```
Which option would you like to perform? [1 to 4] - 3

Data Saved!
******* The current tasks ToDo are: *******
read a book (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program


Which option would you like to perform? [1 to 4] - 4

Goodbye!

Process finished with exit code 0
```

***Figure 12-2**: result of assignment06 script on Pycharm*

Assignment06 Result in the Terminal

Figure 13 shows the result of assignment06 run in Pycharm.

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(Assignment06) yilikals-MacBook-Pro:Assignment06 tsedeytadesse$ python3 Assignment06.py
******* The current tasks ToDo are: *******
read a book (h)
*****************************************


       Menu of Options
       1) Add a new Task
       2) Remove an existing Task
       3) Save Data to File
       4) Exit Program
```

***Figure 13-1**: result of assignment06 script on Terminal*

```
Which option would you like to perform? [1 to 4] - 1

Enter the name of your task: cook
What is the priority for your task: 'h' for high or 'l' for low: h
******* The current tasks ToDo are: *******
read a book (h)
cook (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program


Which option would you like to perform? [1 to 4] - 2

Which Task would you like to remove? clean
Task not found
******* The current tasks ToDo are: *******
read a book (h)
cook (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program

Which option would you like to perform? [1 to 4] - 3

Data Saved!
******* The current tasks ToDo are: *******
read a book (h)
cook (h)
*****************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program
```

***Figure 13-2**: result of assignment06 script on Terminal*

```
        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program


Which option would you like to perform? [1 to 4] - 4

Goodbye!
```

***Figure 13-3**: result of assignment06 script on Terminal*

Text File

Figure 14 shows the text file, ToDoFile.txt, on my drive.

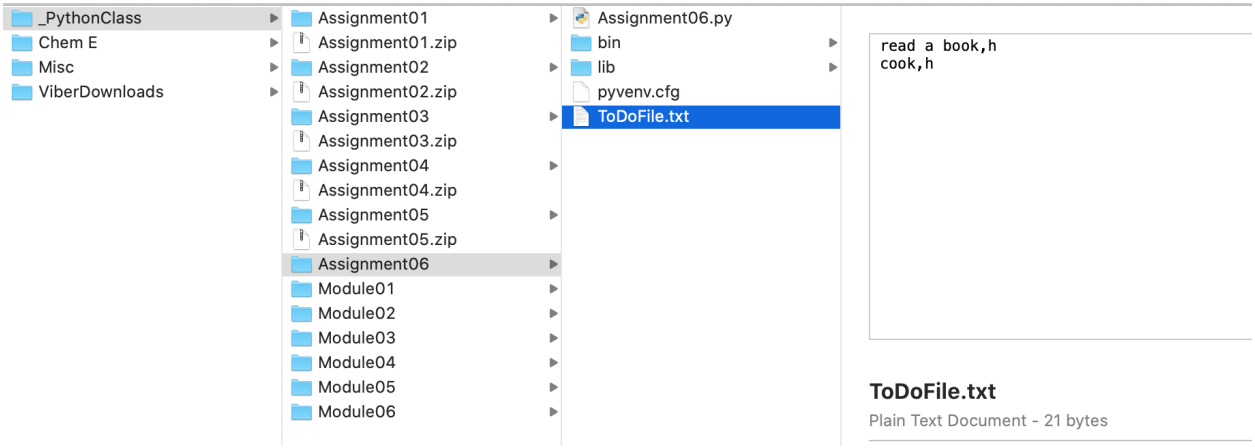


***Figure 14**: text file on drive*

Summary

In module06, with the help of the class notes, the course video, the assigned textbook reading, the supplemental video and website, I was able to learn about functions and classes. Functions help us break down and group sections of our script so that each function performs tasks with its own local variables and makes it easier for us to write, understand, and modify scripts. In turn, classes help us group functions. One script may have so many functions and it would be hard to manage. Therefore grouping them into functions based on their task is very important. Functions and classes also make it easier to debug codes, where each section could be accessed individually.