M.Bruce
Aug 14, 2022
IT FDN 110 (B)
Assignment 06
Link: https://github.com/Jane2024/IntroToProg-Python-Mod06 (GitHub repository)
https://jane2024.github.io/IntroToProg-Python-Mod06/ (GitHub Page)

# Functions and Classes in Python

## Introduction

This week covered Functions- how to define them, what are parameters vs. arguments and returning function values. The practice of abstraction and encapsulation was covered and demonstrated using local and global variables which sets the scope of their visibility. The role of classes and how they differ from functions rounded out this week's agenda. The assignment for the week focused on adding functions and organizing the code segments that were used in previous assignment05 for handling of loading and managing data from and to a file utilizing a Python list of dictionary objects. This knowledge document will cover the above topics and capture how assignment06 was coded in each step as well as lessons learned.

## Functions

In Python, functions are designed to perform a specific operation and are comprised of one or more statements. There are built-in functions and user-defined custom functions. Custom functions must be defined before use. When the program first launches, it reads each function definition but does not run the function. The function will only be run if called in the main body of the script. A function definition contains a function header- which is the first line with a "def <function name>(list of parameters)". This tells the computer the block of code that follows is a function. A function always ends with its return statement. Using docstrings (documentation strings) helps in documenting what the function's purpose is. Docstrings need to be in the first line of code following the header and encapsulated with three quotes.

> Example first two lines of a custom function with a docstring would be:
>
> *def add_two_numbers(n1, n2)*
> *"""Adds two provided numbers and returns their sum """*

### Parameters and arguments

Functions receive values via their parameters which are comma separated variables within the parentheses of a function header. They catch the values provided in the arguments of the function call from main.
Note:
***Parameters:*** *single or multiple variables listed within the parentheses in the function header in function definition*
***Arguments****: single or multiple variables listed within the parentheses in the function's call from main body*

The number of arguments passed to the function needs to match the number of parameters defined for the function or an error will be generated. There are no limits on how many parameters/arguments a function can have.

### Lab 6-1 Working with functions

This lab creates a function that prints the Sum, Difference, Product, and Quotient of two numbers. The code to accomplish this is seen in Listing 6.1 below.

```
# ---------------------------------------------- #
# Title: Lab6-1_MBruce
# Description: Calling a function with multiple parameters
# ChangeLog: mbruce, 8-15-2022, added statements to calc and print the sum, difference, product and quotient
# RRoot,1.1.2030,Created Script
# ---------------------------------------------- #

# Define the function
def CalcValues(value1, value2):
    """ Calculates mathematical outcomes using values of two arguments"""
    fltSum = value1 + value2
    fltDiff = value1 - value2
    fltMult = value1 * value2
    fltDev = value1 / value2

    print("The Sum of the values is: ", fltSum)
    print("The Difference of the values is: ", fltDiff)
    print("The Product of the values is: ", fltMult)
    print("The Quotient of the values is: ", fltDev)


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

LISTING 6.1 FOR Lab6-1 script

*The function is defined with two parameters. A docstring follows. Then the two provided variables via function call from main body with two arguments are used to perform several mathematical operations. Print statements handle output of results as seen in figure 6.1 below.*

```
C:\_PythonClass\Assignment06>python.exe C:\_PythonClass\Assignment06\Lab6-1_MBruce.py
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

C:\_PythonClass\Assignment06>
```

FIGURE 6.1 Lab6.1 - shell output

## Returning values

Where functions can be defined with any number of paraments, they can also return multiple values, and be assigned to variables in main body, or used immediately without placing in a variable. This is referred to as evaluating a function as an expression.

In the example function call as an ***expression*** below, the returned value is used immediately in the print statement.

*Print(MyFunction())*

But, returning values in a function and assigning to variables in main body allows for reuse of the results and saves on duplicated function calls. It also helps separate out code into the three layers of concern- data, processing and presentation.
If multiple values are returned by a function, the values need to be placed into a collection. Tuple packing and unpacking aids in this effort. Lab 6-2 demonstrates this.

## Lab 6-2 Returning Tuples

For Lab 6-2, create a function that returns the Sum, Difference, Product, and Quotient of two numbers as a tuple **Listing 6.2** below is the code for Lab6-2.

```
# ------------------------------------------------ #
# Title: Lab6-2_MBruce
# Description: Calling a function with multiple parameters-return tuple
# ChangeLog: mbruce, 8-15-2022, calc and print the sum, difference, product and quotient. Return as tuple.
# RRoot,1.1.2030,Created Script
# ------------------------------------------------ #

######## Data code ########

fltV1 = None                # Variable for user input 1
fltV2 = None                # Variable for user input 2
tupResults = ()             # Tuple to hold the results of calculations

######## function definition ########

def CalcValues(value1, value2):
    """ Calculates mathematical outcomes using values of two arguments"""
    fltSum = value1 + value2
    fltDiff = value1 - value2
    fltMult = value1 * value2
    fltDev = value1 / value2

    return (fltSum,fltDiff, fltMult, fltDev)       #create a tuple to return values

######## presentation code ########

fltV1 = float(input("Please enter value 1: "))     # get value 1 from user
fltV2 = float(input("Please enter value 2: "))     # get value 2 from user
tupResults = CalcValues(fltV1, fltV2)             # call function with values and return a tuple of results

# Print results
print("The Sum of %.2f and %.2f is %.2f" % (fltV1, fltV2, tupResults[0]))
print("The difference of %.2f and %.2f is %.2f" % (fltV1, fltV2, tupResults[1]))
print("The product of %.2f and %.2f is %.2f" % (fltV1, fltV2, tupResults[2]))
print("The quotient of %.2f and %.2f is %.2f" % (fltV1, fltV2, tupResults[3]))
```

LISTING 6.2 Lab6-2 script

In testing the Lab 6-2 script, the two inputs are provided from user input and assigned to float variables. The function call is done providing the two inputs as arguments. In the function the two values are used for calculations and the results are stored in a tuple “tupResults”. In the print out statements, the variables and tuple

indexes are used to print the results from the function call. In **figure 6.2** below, the Lab6-2 code listed above is run and output in PyCharm.

```
C:\_PythonClass\Assignment06>python.exe C:\_PythonClass\Assignment06\Lab6-2_MBruce.py
Please enter value 1: 5
Please enter value 2: 10
The Sum of 5.00 and 10.00 is 15.00
The difference of 5.00 and 10.00 is -5.00
The product of 5.00 and 10.00 is 50.00
The quotient of 5.00 and 10.00 is 0.50
```

FIGURE6.2 Lab6-2 output in cmd shell

## Types of Arguments

Arguments are the variables supplied in the parentheses when a function is called in a program. A function can have zero to many arguments. Here are some options in Python for arguments***: Positional vs keyword arguments, Default parameters, Overloading functions with arguments and None keyword.***

### Positional vs Named arguments

When calling a function, the order in which the arguments (and their values) are listed reflects the order that they will be assigned to parameters in the function. First argument in the function call gets assigned to the first parameter in the function, 2nd to 2nd parameter and so on. That is ***positional arguments***. If the name of the parameter is listed as the argument and its value is explicitly assigned in the argument, then that is a ***keyword argument***. Keywords help clarify what is being passed to parameters in the function. They can also be listed in any order, since the parameter assignment has already been defined via the passed argument.

### Default parameters

When a function is defined, the listed parameters can be assigned default values within the parentheses in the function header. These values will be used each time the function is called, when arguments are not supplied in the function call. Also, when one parameter gets a default parameter assigned, then all parameters in the function need to have default values assigned.

### Overloading Functions

When listing the parameter name and assigning a value to it in the argument of the function call, it will replace any value previously assigned in the function definition header. This is called overloading the function. If values are provided, it will also replace a previous defined parameter value.

### None keyword

None can be used to indicate the absence of a parameter value. Its value is literally “None”’

### Returning data by reference

None can

## Variable scope (local and global)

Scope in python refers the accessibility or visibility of its variables. If a variable is defined within a function, then it has local scope. It can only be seen and accessed within the function. If it is defined outside the function or with a "global" keyword, then it is a global variable and can be seen and accessed outside and inside the function.

## Abstraction and encapsulation

Abstraction and encapsulation deal with similar concepts when it comes to functions. Abstraction refers to the use of functions to simplify the program. You can call a print function for instance and pass it what you want printed, without worrying about the details of how it handles the printing. Only the name of the function and the arguments are seen. With encapsulation, scope values are set for variables to essentially hide and prevent access to them from outside the function, limiting their scope to within the function. There can be two variables of the same name, one local and one global, but only the local variable will be changed within the function. Outside the function, the global variable will remain unchanged.

## Classes and Functions

Classes are a way of grouping functions of similar actions or operations together.

### Lab 6-3 creating class of functions

For Lab 6-3, create a class that includes four functions, each returning either the Sum, Difference, Product, and Quotient of two numbers as a float result. See **Listing 6.3** for Lab6-3 code defining its class and functions.

```
### Class and Function definitions ###

class MathProcessor():

    """ List of functions for processing math calculations """

    @staticmethod
    def AddValues(value1=0, value2=0):

        """ This function adds two values and returns their sum """

        fltSum = value1 + value2
        return fltSum

    @staticmethod
    def SubtractValues(value1=0, value2=0):

        """ This function subtracts two values and returns their difference """

        fltDifference = value1 - value2
        return fltDifference

    @staticmethod
    def MultiplyValues(value1=0, value2=0):

        """ This function multiplies two values and returns their product """

        fltProduct = value1 * value2
        return fltProduct

    @staticmethod
    def DivideValues(value1=0, value2=0):

        """ This function divides two values and returns their quotient"""

        fltQuotient = value1 / value2
        return fltQuotient
```

LISTING 6.3 Lab6-3 script code segment for class and function definitions

In the script, the four functions were created taking in two float arguments (user input), performing a math-based calculation with them and then returning the resulting calculation. All the functions were defined under the MathProcessor class, which means function calls get the prefix of MathProcessor.<functionName>.
In **listing 6.4** below, the lab6-3 main body of the script shows the functions are evaluated as expressions. They are called and the returned values are used within a print statement without assigning to a separate variable.

```
#####  Main  #####

fltV1 = float(input("Enter value 1: "))
fltV2 = float(input("Enter value 2: "))
print("The Sum of %.2f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.AddValues(fltV1, fltV2)))
print("The Difference of %.2f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.SubtractValues(fltV1, fltV2)))
print("The Product of %.2f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.MultiplyValues(fltV1, fltV2)))
print("The Quotient of %.2f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.DivideValues(fltV1, fltV2)))
```

LISTING 6.4 Lab6-4 main body with function calls

In **figure 6.3** below, the lab6-3 script listed above is run and output in the cmd prompt.

```
c:\_PythonClass\Assignment06>python.exe C:\_PythonClass\Assignment06\Lab6-3_MBruce.py
Enter value 1: 5
Enter value 2: 10
The Sum of 5.00 and 10.00 is 15.00
The Difference of 5.00 and 10.00 is -5.00
The Product of 5.00 and 10.00 is 50.00
The Quotient of 5.00 and 10.00 is 0.50
```

FIGURE6.3 Lab6-3 output in cmd shell

## Asignment06

This assignment was to modify an existing script template that manages a "ToDoFile" list containing data on "Tasks" and their "Priority." The data is loaded into Python Dictionary objects, representing a row of data, which is then added to a list object to create a table of data. This is similar to assignment05, but in this assignment, the work is separated into functions under new classes.

### Create Script

1) Opened PyCharm and created folder and project named "Assignment06"
2) Added script named **Assignment06_Starter_updated.py** to project and rename it at:
   *C:/_PythonClass/Assignment06/* **Assignment06.py**
3) Updated header info, and define variables to be used in the program

Two Classes were created to handle the processing and input/output of the program:
- *Class Processor*
- *Class IO*

The "Processor" class separates and contains all the functions that handle data processing:
*-read_data_from_file()*
*-write_data_from_file()*
*-add_data_to_list()*
*-remove_data_from_list()*

The “IO” class separates and contains all the functions that handle input from the user and output to display:
*-output_menu_tasks()*
*-input_menu_choice()*
*-output_current_tasks_in_list()*
*-input_new_task_and_priority()*
*-input_task_to_remove()*
*-output_task_removed()*
*-output_task_not_in_list()*

## Load contents of ToDoFile.lst, Display Menu, show current data and get user choice

When the program launches, it automatically reads the tasks in the “ToDoFile.txt” and loads them into a list of dictionary objects via function call to ***Processor.read_data_from_file(),*** which passes two arguments for the file name and the list. The function returns the list of rows. The list of rows is passed to the function: ***IO.output_current_tasks_in_list()*** to display the current tasks in the list to the user, followed by a function call to ***IO.output_menu_tasks()*** to show the user a list of options for actions they can perform. The user is prompted for their action choice via a function call to ***IO.input_menu_choice().*** This returns a menu item choice in the form of a string variable “choice_str” which is tested for in a series of “if” /”elif” statements.

Each If condition is associated with an action corresponding to a menu item choice in a while loop.

First menu option is add new task handled by function call to IO.input_new_task_and_priority(). This prompts the user for a task name and a priority. This is returned by the function and passed in two arguments along with the ***list of rows***, to Processor.add_data_to_list() function. After appending new task to list, it returns the updated list. The code for both these is seen in ***figures 5.4 and 5.5***

```
@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
    """
    # TODO: Add Code Here!

    strTask = input("Please input a new task name:")                    # Prompt user for new task- assign to string "strTask"
    strPriority = input("Please input its priority:")                   # Prompt user for priority- assign to string "strPriority"

    return (strTask, strPriority)                                       # Return user entered task and priority values
```

FIGURE5.4 **Assignment06 code** for function **“**IO.input_new_task_and_priority().”

Here is the code for adding the new task to the list in ***Figure 5.5***

```
@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: Add Code Here!

    list_of_rows.append(row)                                    # Add it to the current task list of dictionary objects- "lstTable"

    return list_of_rows
```

FIGURE5.5 **Assignment06 code** for function **“**Processor.add_data_to_list()”

The output to the user for this portion of the program is seen in ***figure 5.6*** below.

```
c:\_PythonClass\Assignment06>python.exe C:\_PythonClass\Assignment06\Assigment06.py
******* The current tasks ToDo are: *******
Prevent Global Warming (#1)
Solve World Hunger (#2)
World Peace (#3)
*****************************************


        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

Please input a new task name:Reduce waste
Please input its priority:#4
******* The current tasks ToDo are: *******
Prevent Global Warming (#1)
Solve World Hunger (#2)
World Peace (#3)
Reduce waste (#4)
*****************************************
```

FIGURE5.6 **Assignment06 cmd shell output** for ***add new task*** and ***add data to list***, then it returns back into the top of the while loop to print out current list and eventually print out the menu again and prompt the user for a new menu option entry.

## Remove task from list option

The second menu action begins with a function call to ***IO.input_task_to_remove()***. This prompts the user for a task name to be removed from the list. This function returns the task name and passes it along with the ***list of rows***, to function ***Processor.remove_data_from_list().*** These functions are documented in ***Figure 5.7 below and 5.8 below.***

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

    :return: (string) with task
    """
    #TODO: Add Code Here!

    strRemoveTask = input("Enter name of task you want to remove?")      # Prompt user for task to be removed

    return strRemoveTask                                                 # Return user entered task value
```

FIGURE5.7 **Assignment06 code for function** IO_input_task_to_remove()

The removal of the task is done by first creating a new list of just task values to be searched for a match to the entered "task" by the user. If a match is found, it removes it and calls ***IO.output_task_removed()*** to tell the user it has been removed. If no match is found, it calls function ***IO.output_task_not_in_list()*** to notify user the task was not in the current list.
***Figure 5.8*** shows the initial function call to remove data from the list

```
@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: Add Code Here!
    lst_of_all_values = [task for elem in list_of_rows for task in elem.values()]  # Used List comprehension to create list of
                                                                                    # only task values for easier task matching

    if task not in lst_of_all_values:                           # If this evaluates to true (task not in list)
        IO.output_task_not_in_list(task=task)                   # Inform user task in not in list- call function to print message

    else:
        for row in list_of_rows:                                # Loop through lstTable to find task match
            if row["Task"].lower() == task.lower():             # Check for match
                list_of_rows.remove(row)                        # When match is found, Remove task
                IO.output_task_removed(task=task)               # Inform user task is removed- call function to print out message


    return list_of_rows
```

FIGURE5.8 Assignment06 option 2 -code for function *Processor. remove_data_from_list().*

***Figure 5.9*** below shows code to inform the user the task was removed via function IO.output_task_removed()

```
@staticmethod
def output_task_removed(task):
    """  outputs the task name that was removed from the list

    :return: nothing
    """

    print("The task: '" + task + "' has been removed")              # Inform user task is remove
    print("*******************************************")            # decoration
    print()                                                         # Add an extra line for looks
```

FIGURE5.9 Assignment06 code for- *IO.output_task_removed()*

*Figure 5.10* below shows the cmd shell output when user selects option #2 to remove a task.

```
******* The current tasks ToDo are: *******
Prevent Global Warming (#1)
Solve World Hunger (#2)
World Peace (#3)
Reduce waste (#4)
*******************************************


        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

Enter name of task you want to remove?Reduce waste
The task: 'Reduce waste' has been removed
*******************************************

******* The current tasks ToDo are: *******
Prevent Global Warming (#1)
Solve World Hunger (#2)
World Peace (#3)
*******************************************
```

FIGURE5.10 Assignment06 cmd shell output for option "Remove an existing task".

*Figure 5.11* below shows the code for function *IO.output_task_not_in_list(),* which is called if task was not found.

```
@staticmethod
def output_task_not_in_list(task):
    """  outputs that task name that was not found in the list

    :return: nothing
    """

    print("The task: '" + task + "' is not in current list")        # Inform user task was not in list
    print("*******************************************")            # decoration
    print()                                                         # Add an extra line for looks
```

FIGURE5.11 Assignment06 - code for function to display message task not in current list

*Figure 5.12* below shows the cmd shell output when the task is not in the current list.

```
*******************************************


       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

Enter name of task you want to remove?reduce pollution
The task: 'reduce pollution' is not in current list
*******************************************

******* The current tasks ToDo are: *******
Prevent Global Warming (#1)
Solve World Hunger (#2)
World Peace (#3)
Reduce waste (#4)
*******************************************
```

FIGURE5.12 Assignment06 cmd shell output

## Save Data to file option

If menu item # "3" was chosen, then function ***Processor.write_data_to_file()*** is called with arguments for file name to be written to and "list of rows" for the data to be written. ***Figures 5.13 and 5.14*** show the code and the output of this function.

```
@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: Add Code Here!
    objFile = open(file_name, "w")                          # Open file ToDoList.txt for write
    for row in list_of_rows:                                # Loop through dictionary objects (row) in list table
        objFile.write(row["Task"] + "," + row["Priority"] + "\n")   # Write each one to file
    objFile.close()
    return list_of_rows                                     # Need to return the list_of_rows after write to file, else task list will not print
```

FIGURE5.13 Assignment06 -code for function Processor.write_data_to_file()

```
*******************************************


       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: *******
Prevent Global Warming (#1)
Solve World Hunger (#2)
World Peace (#3)
*******************************************
```

FIGURE5.14 Assignment06 -cmd shell output for function Processor.write_data_to_file()

## Final step- Exit program

If user selects option # 4, the program exits via a break statement. User is informed program is exiting. See output in **Figure 5.15.**

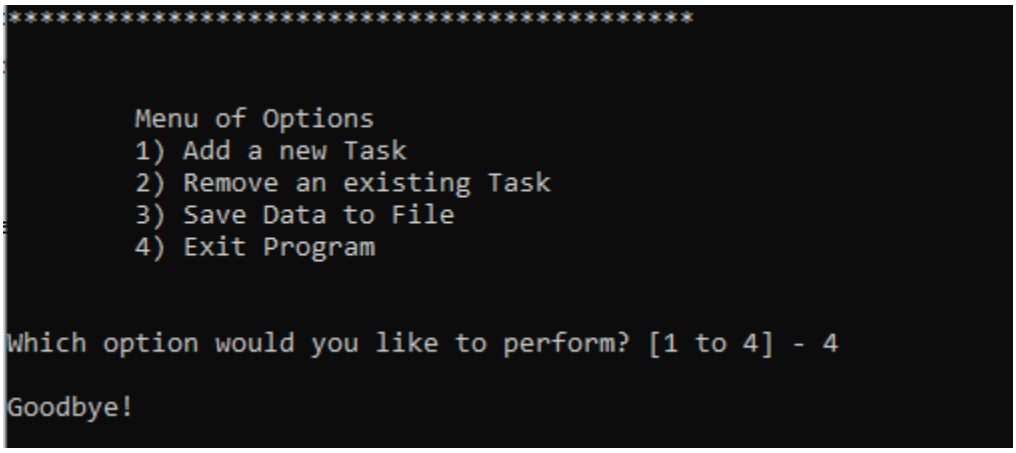


FIGURE5.15 Assignment06 – Option #4 program exit in the cmd shell

## Summary

In the module06 the concentration was on learning to use classes and functions to employ abstraction and encapsulation methodologies. The classes help group together the similar operations and the functions help to separate the code into more manageable smaller chucks than our previous assignments. It also allowed for code reuse for repeated operations like the display of the main menu options. In addition to the work with functions and classes, using the PyCharm debugger was covered and how to make GitHub pages to document associated code changes checked into GitHub.