Martin Stallworth

19 August 2020

Foundations of Programming, Python

Assignment 06

Working with Functions and Classes

Introduction

During last week's module we got a small taste of how to use functions to group statements. This week we dived deeper into this concept. One of the things we broke down was the different properties that make up a function like arguments and return values. We also discussed how local and global variables interact with and within a function. Lastly, we discussed how to create a header for a function, or a docstring, and how to group functions into classes.

Getting the Background

Last week we received a built preview into learning how to incorporate functions into our programs but this week we went more in depth into the topic. Functions are another way to group statements with the added versatility that you can call the function when you actually need it. Within a function you can add parameters, or arguments, to pass in different values for processing. This is then used to get a return value. Initially, I had some confusion on the difference between what an argument and return value does but the Saravji's Hut Part 1 video did a great job in explaining the role that each plays within a function. To get some clarification I also read some of the book because I typically find the example programs the book provides very helpful to solidify the information that I pick up in the module. The 6th chapter actually has a bunch a functions back-to-back that really helped me see the different constructs that you can keep inside of a function (Dawson 176-186). Some of these were very similar to what we have been doing in previous weeks so its cool to see that you can pack a lot of the program into the function and then use a simple call to get all of that to run.

One of the next big topics that we got into were global and local variables. I heard mention of these types of variables previously during the lectures but I had no idea what they actually were. It seems as though the difference is mostly relevant when you want to have a shadow variable, a variable inside the function that has the same name as a variable outside of the function. From my understanding of the module a global variable is one that is available to the entire program but also including the function. You would likely see a global variable located in the "DATA" section of the program. Meanwhile, a local variable is a variable that is set inside of the function and only works within the confines of that function. This means that if you have multiple functions a local variable would only work in the function it is defined in. Next we discussed docstrings which serve as a header for functions. I can see the usefulness that a docstring may have for a program because it allows you to think about what you want to put in and get out of your function before you even put the data inside. Lastly, we got a brief preview into using classes to group functions. This seemed like it was making things more complex but once I started working on the assignment I got to see how useful it was to group functions into classes.

Working through Labs and Assignment

The labs for this week continued the more straightforward feel that the labs from last week had. I enjoyed working with a program that was built off of one of the first assignments. It also helps to modify an existing script instead of making one from scratch because you have a better idea of what direction to take. Using the preceding example in the module I was able to work through Lab A relatively quickly. The only difficulty that I had was that initially I was trying to shove all of the return values into one variable before printing and it did

not come out looking great appearance wise. I then changed to create a variable for the return value of each operation and then organized all of these into one print statement which came out looking a lot better. An example of this is provided below.

```
1.  intNumA = int(input('Please enter 1st Number:'))
2.  intNumB = int(input('Please enter 2nd Number:'))
3.  print('\nUsing the numbers:', intNumA, 'and', intNumB)
4.  one_answer = getSum(intNumA, intNumB)
5.  two_answer = getDif(intNumA, intNumB)
6.  three_answer = getPro(intNumA, intNumB)
7.  four_answer = getQuo(intNumA, intNumB)
8.  print('\nThe result is:\n', one_answer, '\n', two_answer, '\n', three_answer, '\n', four_answer)
```

Listing 1: Ex of using variables to call a function

Lab B seemed like a logical follow-up to Lab A. Instead of having four different functions for each of the operations we had to put them all in one function and return the four different values. I essentially took the same approach that I took in the first lab and copied that into the same function. I used a list to hold all of the return values and then continued to use four variables to call the return values for each corresponding operation. The difference this time is that I had to unpack the list first. There was also another approach that I could have taken but I used this one to keep consistency with my first lab.

```
1.  intNumA = int(input('Please enter 1st Number:'))
2.  intNumB = int(input('Please enter 2nd Number:'))
3.  print('\nUsing the numbers:', intNumA, 'and', intNumB)
4.  lstResult = getResults(intNumA, intNumB)
5.  one_answer = lstResult[0]
6.  two_answer = lstResult[1]
7.  three_answer = lstResult[2]
8.  four_answer = lstResult[3]
9.  print('\nThe result is:\n{}\n{}\n{}\n{}'.format(one_answer, two_answer, three_answer, four_answer))
```

Listing 2: Ex of unpacking list hold multiple return vals

Lab C was a follow-up to the previous two labs as well as a good precursor to the assignment. This was a great introduction into how to group functions under a class as well as good practice for creating a docstring for a function. An example of my class and a docstring is provided below.

```
1.  class SimpleMath():
2.      """A collection of simple math processing functions"""
3.      @staticmethod
4.      def get_sum(value1, value2):
5.          """Function for adding two values
6.
7.
8.          Args:
9.              value1: the first number to add
10.             value2: the second number to add
11.
12.
13.         Returns:
14.             A float corresponding to the sum of value1 and value2
15.         """
16.         return float(value1 + value2)
```

Listing 3: Ex of grouping a function under a class and a docstring

Once I started to work on the assignment it was very intimidating at first because this was probably the most lines of code I had ever seen to this point. However, the lecture definitely helped to clarify things that I was confused about. The description of what each of the TODOs meant was helpful especially. I initially thought

that I had to create code from scratch inside each function but once Dirk went through it I saw that I had to move some of the code from the presentation portion of the program into only some of the functions. It was almost like working backwards because I had the information that was supposed to go into the docstrings before I even created the docstrings. This did make it easier to figure out what went into each section of the docstrings. I then added the 'IO' functions to the main loop to call the functions that I had to create. I liked the structure of the portion that was already in the program for loading the data so I decided to add this to my delete function to add a little more to my code. This is the portion of the assignment that I had the most difficulty with because I had to figure out how to properly incorporate it with the code that I already had in this function. After getting some help from Douglas I was able to come to this result.

```
    def del_data(intIDDel, lstTbl):
        """Function to delete data in memory
        deletes ID, CD Title, and Artist Name using the ID #


        Args:
            intIDDel (integer): integer for ID that represents entry to be deleted
            lstTbl (list of dic): 2D data structure (list of dicts) that holds the data during runtim
e


        Return:
            None.
        """
        intRowNr = -1
        blnCDRemoved = False
        for row in lstTbl:
            intRowNr += 1
            if row['ID'] == intIDDel:
                print('WARNING: If you continue ID', intIDDel,'will be deleted from the memory!')
                strYESNO = input('type \'yes\' to continue and delete row from the memory. Otherwise
 the deletion will be cancelled.' )
                if strYESNO.lower() == 'yes':
                    print('\ndeleting...')
                    del lstTbl[intRowNr]
                    blnCDRemoved = True
                else:
                    input('cancelling...Entry not deleted. Press [ENTER] to continue to the menu.')
                break
        if blnCDRemoved:
            print('The CD was removed')
            print()
        else:
            print('\nCould not find this CD!')
            print()

```

Listing 4: Example of del function in assignment

Once I got this down I was happy with the rest of my program and made sure all parts of the menu worked properly in both Spyder and the terminal and upload my code onto GitHub. Photos of this delete function working are provided below.

```
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Damn. (by:Kendrick Lamar)
2       Blueprint (by:Jay-Z)
3       Blonde (by:Frank Ocean)
4       Lemonade (by:Beyonce)
=====================================

Which ID would you like to delete? 4
WARNING: If you continue ID 4 will be deleted from the memory!

type 'yes' to continue and delete row from the memory. Otherwise the deletion will be cancelled.yes

deleting...
The CD was removed

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Damn. (by:Kendrick Lamar)
2       Blueprint (by:Jay-Z)
3       Blonde (by:Frank Ocean)
=====================================
```

Figure 1: Ex of del function working in Spyder

```
Which operation would you like to perform? [l, a, i, d, s or x]: d

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       The Big Wheel (by:Runrig)
2       Bad (by:Michael Jackson)
3       Lemonade (by:Beyonce)
4       Damn. (by:Kendrick Lamar)
5       Blonde (by:Frank Ocean)
6       Blueprint (by:Jay-Z)
=====================================
Which ID would you like to delete? 1
WARNING: If you continue ID 1 will be deleted from the memory!
type 'yes' to continue and delete row from the memory. Otherwise the deletion will be cancelled.yes

deleting...
The CD was removed

======= The Current Inventory: =======
ID      CD Title (by: Artist)

2       Bad (by:Michael Jackson)
3       Lemonade (by:Beyonce)
4       Damn. (by:Kendrick Lamar)
5       Blonde (by:Frank Ocean)
6       Blueprint (by:Jay-Z)
```

Figure 2: Ex of del function working in terminal

Summary

In this assignment we got a more in depth look into how to group statements using functions compared to the introduction that we got last week. We used the labs and the assignments to break down the different properties that a part of a function. This could include things like: arguments, return values, docstrings, etc.. We also were able to work on grouping functions themselves using classes which it appears we will dive deeper into next module. I can see the usefulness for using functions for organization purposes and making a code look more professional. I am excited to learn new ways I give my codes a better presentation and interactivity with users.

Appendix

Using planetb's webpage (external reference). Retrieved 2020-Aug-19

Using Saravji's Hut Youtube (external reference). Retrieved 2020-Aug-19

Dawson, Michael. Python Programming for the Absolute Beginner: 3rd Edition. Kindle ed., Cengage Learning, 2010

Link for GitHub: https://github.com/mstallworth513/Assignment_06

Listing for LAB06_A.py

```
#-------------------------------------------#
# Title: LAB06_A.py
# Desc: Script demonstrating using attributes to pass values into functions and using return values
# Change Log: MStallworth, 2020-Aug-17, added attributes and return values to functions and created variables outside of function to hold these return values.
# MStallworth, 2020-Aug-17, Created File
#-------------------------------------------#

# -- DATA -- #
intNumA = None
intNumB = None
one_answer = None
two_answer = None
three_answer = None
four_answer = None

# -- PROCESSING -- #
def getSum(value1, value2):
    Sum = 'Sum: {}'.format(value1 + value2)
    return Sum

def getDif(value1, value2):
    Difference = 'Difference: {}'.format(value1 - value2)
    return Difference

def getPro(value1, value2):
    Product = 'Product: {}'.format(value1 * value2)
    return Product

def getQuo(value1, value2):
    Quotient = 'Quotient: {}'.format(value1 / value2)
    return Quotient


# -- PRESENTATION (Input/Output) -- #
# Get user input data
print('Basic Math script. Calculating the sum, difference, product, and quotient of two numbers.\n')

intNumA = int(input('Please enter 1st Number:'))
intNumB = int(input('Please enter 2nd Number:'))
print('\nUsing the numbers:', intNumA, 'and', intNumB)
one_answer = getSum(intNumA, intNumB)
two_answer = getDif(intNumA, intNumB)
three_answer = getPro(intNumA, intNumB)
four_answer = getQuo(intNumA, intNumB)
print('\nThe result is:\n', one_answer, '\n', two_answer, '\n', three_answer, '\n', four_answer)
```

Listing for LAB06_B.py

```
#-------------------------------------------#
# Title: LAB06_B.py
# Desc: Script demonstrating creating a function with multiple return values
# Change Log: MStallworth, 2020-Aug-17, moved all operations into one function with multiple return values and group the return into a list)
# MStallworth, 2020-Aug-17, Created File>
#-------------------------------------------#

# -- DATA -- #
intNumA = None
```

```
intNumB = None
one_answer = None
two_answer = None
three_answer = None
four_answer = None
lstResult = None

# -- PROCESSING -- #
def getResults(value1, value2):
    Sum = 'Sum: {}'.format(value1 + value2)
    Difference = 'Difference: {}'.format(value1 - value2)
    Product = 'Product: {}'.format(value1 * value2)
    Quotient = 'Quotient: {}'.format(value1 / value2)
    return [Sum, Difference, Product, Quotient]


# -- PRESENTATION (Input/Output) -- #
# Get user input data
print('Basic Math script. Calculating the sum, difference, product, and quotient of two numbers.')
intNumA = int(input('Please enter 1st Number:'))
intNumB = int(input('Please enter 2nd Number:'))
print('\nUsing the numbers:', intNumA, 'and', intNumB)
lstResult = getResults(intNumA, intNumB)
one_answer = lstResult[0]
two_answer = lstResult[1]
three_answer = lstResult[2]
four_answer = lstResult[3]
print('\nThe result is:\n{}\n{}\n{}\n{}'.format(one_answer, two_answer, three_answer, four_answer))
```

Listing for LAB06_C.py

```
#-------------------------------------------#
# Title: LAB06_C.py
# Desc: Script demonstrating the use of docstrings in functions and grouping functions in a class
# Change Log: MStallworth, 2020-Aug-
18, Added docstrings to LAB A functions and placed functions under a class. Used the classname to cal
l each function.)
# MStallworth, 2020-Aug-18, Created File
#-------------------------------------------#

# -- DATA -- #
intNumA = None
intNumB = None
one_answer = None
two_answer = None
three_answer = None
four_answer = None

# -- PROCESSING -- #
class SimpleMath():
    """A collection of simple math processing functions"""
    @staticmethod
    def get_sum(value1, value2):
        """Function for adding two values


        Args:
            value1: the first number to add
            value2: the second number to add


        Returns:
            A float corresponding to the sum of value1 and value2
        """
        return float(value1 + value2)

    @staticmethod
```

```
    def get_difference(value1, value2):
        """Function for subtracting two values


        Args:
            value1: the number to subtract from
            value2: the number to subtract


        Returns:
            A float corresponding to the difference of value1 and value2
        """
        return float(value1 - value2)

    @staticmethod
    def get_product(value1, value2):
        """Function for multiplying two values


        Args:
            value1: the first number to multiply
            value2: the second number to multiply


        Returns:
            A float corresponding to the product of value1 and value2
        """
        return float(value1 * value2)

    @staticmethod
    def get_quotient(value1, value2):
        """Function for dividing two values


        Args:
            value1: the number to divide from
            value2: the number to divide


        Returns:
            A float corresponding to the product of value1 and value2
        """
        return float(value1 / value2)



# -- PRESENTATION (Input/Output) -- #
# Get user input data
print('Basic Math script. Calculating the sum, difference, product, and quotient of two numbers.\n')

      intNumA = int(input('Please enter 1st Number:'))
      intNumB = int(input('Please enter 2nd Number:'))
      print('\nThis script is calculated using the numbers', intNumA, 'and', intNumB)
      print('\nThe results are:\n')
      print('Sum:\t\t',SimpleMath.get_sum(intNumA, intNumB))
      print('Difference:\t',SimpleMath.get_difference(intNumA, intNumB))
      print('Product:\t',SimpleMath.get_product(intNumA, intNumB))
      print('Quotient:\t',SimpleMath.get_quotient(intNumA, intNumB))
```

Listing for CDInventory.py

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Working with classes and functions.
# Change Log: MStallworth, 2020-Aug-
19, Moved code from main menu into the functions, added docstrings to added functions, created blank
file, added IO's to menu to connect to corresponding functions.
```

```
# DBiesinger, 2030-Jan-01, Created File. MStallworth, 2020-Aug-19, Appened File
#------------------------------------------#

# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
strFileName = 'CDInventory.txt'  # data storage file
objFile = None  # file object


# -- PROCESSING -- #
class DataProcessor:
    # TODone add functions for processing here
    """Processing the data"""

    @staticmethod
    def add_data(strID, strTitle, strArtist, lstTbl):
        """Function to add data in memory using a list of dictionaries
        Adds the ID, CD Title, and Artist Name


        Args:
            strID (string): # for identifying each entry
            strTitle (string): Name of CD
            strArtist (string): Name of artist
            lstTbl (list of dic): 2D data structure (list of dicts) that holds the data during runtim
e


        Returns:
            None.
        """

        intID = int(strID)
        dicRow = {'ID': intID, 'Title':strTitle, 'Artist':strArtist}
        lstTbl.append(dicRow)

    @staticmethod
    def del_data(intIDDel, lstTbl):
        """Function to delete data in memory
        deletes ID, CD Title, and Artist Name using the ID #


        Args:
            intIDDel (integer): integer for ID that represents entry to be deleted
            lstTbl (list of dic): 2D data structure (list of dicts) that holds the data during runtim
e


        Return:
            None.
        """
        intRowNr = -1
        blnCDRemoved = False
        for row in lstTbl:
            intRowNr += 1
            if row['ID'] == intIDDel:
                print('WARNING: If you continue ID', intIDDel,'will be deleted from the memory!')
                strYESNO = input('type \'yes\' to continue and delete row from the memory. Otherwise
the deletion will be cancelled.' )
                if strYESNO.lower() == 'yes':
                    print('\ndeleting...')
                    del lstTbl[intRowNr]
                            blnCDRemoved = True
                        else:
                            input('cancelling...Entry not deleted. Press [ENTER] to continue to the me
nu.')
```

```
                    break
            if blnCDRemoved:
                print('The CD was removed')
                print()
            else:
                print('\nCould not find this CD!')
                print()


    class FileProcessor:
        """Processing the data to and from text file"""

        @staticmethod
        def read_file(file_name, table):
            """Function to manage data ingestion from file to a list of dictionaries

            Reads the data from file identified by file_name into a 2D table
            (list of dicts) table one line in the file represents one dictionary row in table.

            Args:
                file_name (string): name of file used to read the data from
                table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

            Returns:
                None.
            """
            table.clear()  # this clears existing data and allows to load data from file
            objFile = open(file_name, 'r')
            for line in objFile:
                data = line.strip().split(',')
                dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
                table.append(dicRow)
            objFile.close()

        @staticmethod
        def write_file(file_name, lstTbl):
            """Function to write in memory data into a file
            Saves the data into identified file name from a 2D table


            Args:
                file_name (string): name of the file used to save the data
                lstTbl: 2D data structure used to save the in memory data


            Returns:
                None.
            """
            # TODone Add code here
            objFile = open(strFileName, 'w')
            for row in lstTbl:
                lstValues = list(row.values())
                lstValues[0] = str(lstValues[0])
                objFile.write(','.join(lstValues) + '\n')
            objFile.close()


    # -- PRESENTATION (Input/Output) -- #

    class IO:
        """Handling Input / Output"""

        @staticmethod
        def print_menu():
            """Displays a menu of choices to the user

            Args:
```

```
                    None.

                Returns:
                    None.
                """

                print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory
    ')
                print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')

            @staticmethod
            def menu_choice():
                """Gets user input for menu selection

                Args:
                    None.

                Returns:
                    choice (string): a lower case sting of the users input out of the choices l, a, i,
     d, s or x

                """
                choice = ' '
                while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
                    choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: '
    ).lower().strip()
                print()  # Add extra space for layout
                return choice

            @staticmethod
            def show_inventory(table):
                """Displays current inventory table


                Args:
                    table (list of dict): 2D data structure (list of dicts) that holds the data during
     runtime.

                Returns:
                    None.

                """
                print('======= The Current Inventory: =======')
                print('ID\tCD Title (by: Artist)\n')
                for row in table:
                    print('{}\t{} (by:{})'.format(*row.values()))
                print('======================================')

            @staticmethod
            def add_entry():
                """Adds user input into the in memory inventory
                using .add_data() from Data Processing Class


                Args:
                    None.


                Returns:
                    strID (string): ID for data entry
                    strTitle (string): Input for CD Title
                    strArtist (string): Input for Artist Name
                """

                strID = input('Enter ID: ').strip()
                strTitle = input('What is the CD\'s title? ').strip()
                strArtist = input('What is the Artist\'s name? ').strip()
                return strID, strTitle, strArtist
```

```
234.
235.
236.            # TODone add I/O functions as needed
237.        # create a blank file
238.        objFile = open(strFileName, 'a')
239.        objFile.close()
240.        # 1. When program starts, read in the currently saved Inventory
241.        FileProcessor.read_file(strFileName, lstTbl)
242.
243.        # 2. start main loop
244.        while True:
245.            # 2.1 Display Menu to user and get choice
246.            IO.print_menu()
247.            strChoice = IO.menu_choice()
248.
249.            # 3. Process menu selection
250.            # 3.1 process exit first
251.            if strChoice == 'x':
252.                print('\nGoodbye.')
253.                break
254.            # 3.2 process load inventory
255.            if strChoice == 'l':
256.                print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-
    loaded from file.')
257.                strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will
     be canceled')
258.                if strYesNo.lower() == 'yes':
259.                    print('reloading...')
260.                    FileProcessor.read_file(strFileName, lstTbl)
261.                    IO.show_inventory(lstTbl)
262.                else:
263.                    input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the
    menu.')
264.                    IO.show_inventory(lstTbl)
265.                continue  # start loop back at top.
266.            # 3.3 process add a CD
267.            elif strChoice == 'a':
268.                # 3.3.1 Ask user for new ID, CD Title and Artist
269.                # TODone move IO code into function
270.                ID, Title, Artist = IO.add_entry()
271.
272.                # 3.3.2 Add item to the table
273.                DataProcessor.add_data(ID, Title, Artist, lstTbl)
274.                # TODone move processing code into function
275.
276.                IO.show_inventory(lstTbl)
277.                continue  # start loop back at top.
278.            # 3.4 process display current inventory
279.            elif strChoice == 'i':
280.                IO.show_inventory(lstTbl)
281.                continue  # start loop back at top.
282.            # 3.5 process delete a CD
283.            elif strChoice == 'd':
284.                # 3.5.1 get Userinput for which CD to delete
285.                # 3.5.1.1 display Inventory to user
286.                IO.show_inventory(lstTbl)
287.                # 3.5.1.2 ask user which ID to remove
288.                intIDDel = int(input('Which ID would you like to delete? ').strip())
289.                    # 3.5.2 search thru table and delete CD
290.                DataProcessor.del_data(intIDDel, lstTbl)
291.                IO.show_inventory(lstTbl)
292.                continue  # start loop back at top.
293.
294.                # TODone move processing code into function
295.
296.            # 3.6 process save inventory to file
297.            elif strChoice == 's':
298.                # 3.6.1 Display current inventory and ask user for confirmation to save
```

```
299.                IO.show_inventory(lstTbl)
300.                strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
301.                # 3.6.2 Process choice
302.                if strYesNo == 'y':
303.                    # 3.6.2.1 save data
304.                    # TODone move processing code into function
305.                    FileProcessor.write_file(strFileName, lstTbl)
306.                else:
307.                    input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

308.                continue  # start loop back at top.
309.            # 3.7 catch-
    all should not be possible, as user choice gets vetted in IO, but to be save:
310.            else:
311.                print('General Error')
```