Anya Pryor
11/17/2021

Foundations of Programming
Python

Instructors: Randall Root
Grace Nam

Python Module Six

Introduction

In this module we are now surveying the use of functions and parameters, distinguishing between parameters and arguments. Return values are defined in this module. We will also look at global and local variables and the difference between a function and a class.

Functions

Functions are saved processes. You can pass a parameter value into a function, making them more flexible in use. Variables with no prefix, and using all lower case and underscores (snake casing) is the standard for variable names. You can chain together multiple operations within a function.

```
# ---------------------------------------------------------------------------- #
# Title: Functions lab
# Description: Using a function with multiple operations and parameters
# ChangeLog (Who,When,What):
# Anya Pryor 11/17/21 Modified code following the example of the video
# ---------------------------------------------------------------------------- #
def CalculateValues(value1, value2):
    fltaddvalues= value1+value2
    fltdifference= abs(value1-value2)
    fltproduct = value1 * value2
    fltanswer = value1 / value2
    print("The sum of values is: " + str(fltaddvalues))
    print("The difference of the values is: " + str(fltdifference))
    print("The product of the values is: " + str(fltproduct))
    print("The quotient of the values is: " + str(fltanswer))
CalculateValues(10,5)
```

Arguments are the values passed into the parameter. It’s a good practice to declare your variables first and then pass arguments into them.

In a function you can use the keyword return. This allows you to separate out the processing from the presentation tasks in the query. Here is the code from the second lab. Unlike the answer in the video I did use prefixes on the variable names. This was very similar to the previous listing exercise where we used the command return to generate a list instead of a tuple.

```
#--------------------------------------------------#
# Title: Lab 6 02
# Description: Returning multiple values as a tuple with a
function that does multiple processes
# ChangeLog: (Who, When, What)
# RRoot, 01.01.2030, Created Script
#Anya Pryor 11/19/21 added to the function
#--------------------------------------------------#

#data
fltvalue1=None #first value
fltvalue2=None #second value
fltResult1=None #first result of processing
fltResult2=None #second results of processing
fltResult3=None #third result of processing
fltResult4=None #fourth result
fltresult5=None
fltresult6=None
#processing
def Calculatevalues (fltvalue1,fltvalue2):
    fltAddValues=fltvalue1+fltvalue2

    fltDifference=abs(fltvalue1-fltvalue2)

    fltProduct=fltvalue1*fltvalue2

    fltQuoitent=fltvalue1/fltvalue2

    return fltvalue1,fltvalue2, fltAddValues,fltDifference,
fltProduct, fltQuoitent #pack the tuple

#presentation and input/output

fltvalue1=float(input("Enter value 1: "))

fltvalue2=float(input("Enter value 2: "))

fltResult1, fltResult2, fltResult3,fltResult4, fltresult5,
fltresult6=Calculatevalues(fltvalue1,fltvalue2) #unpack the
tuple

print("The sum of %.2f and %.2f is %.2f" % (fltResult1,
fltResult2, fltResult3)) #present the tuple elements

print("The difference of %.2f and %.2f is %.2f" %
(fltResult1, fltResult2, fltResult4))

print("The product of %.2f and %.2f is %.2f" % (fltResult1,
fltResult2, fltresult5))

print("The quoitent of %.2f and %.2f is %.2f" %
(fltResult1, fltResult2, fltresult6))
```

Arguments

In Dawson's Python Programming for The Absolute Beginner he defines a relationship between a parameter and an argument as "Parameters catch the values sent to the function from a function call through its arguments."(p. 164). So far we have been setting up the parameters and then replacing them with arguments. This means that the position in which they appear in the call of the function can impact the results. These are positional arguments. Another way to use parameters and arguments is to assign them explicitly in the function call. These are named arguments. Here is an example:

```
#-------------------------------------------------#
# Title: Listing 08
# Description: Using explict parameter names
# ChangeLog: (Who, When, What)
# RRoot, 01.01.2030, Created Script
#Anya Pryor, 11/21/21 added some notes
#-------------------------------------------------#
#data
fltvalue1=None #first value
fltvalue2=None #second value
fltResult1=None #first result of processing
fltResult2=None #second results of processing
fltResult3=None #third result of processing
#processing
def AddValues(value1,value2):
    fltanswer=value1+value2
    return value1, value2, fltanswer
#presentation
fltvalue1=float(input("Enter value 1: "))
fltvalue2=float(input("Enter value 2; "))
fltResult1, fltResult2, fltResult3
=AddValues(value1=fltvalue1, value2=fltvalue2) #here is
where the parameter names are called
```

```
print("The result of %.2f and %.2f is %.2f"%
(fltResult1,fltResult2,fltResult3))
```

Value Type and Reference Type

Reviewing this material from module 2, some types of variables act as references, where they hold the most recent value sent to them, and others act as references, where the initial assigned value persists. Simple variables act as values, as in this example:

```
x = 1

y = x

x = 3
```

print(y) Returns the value of 1 as the initial value persists.

More complex data types, such as lists, will act as references and update dynamically:

```
x = [1, 2] # This is a Python List

y = x

x[0] = 3

print(y) # Prints [3,2], as expected of a reference!
```

Lists and dictionaries act as reference types. Here is an example from listing 13:

```
#-------------------------------------------------#

# Title: Listing 13

# Description: Python's version of by val and by ref

# ChangeLog: (Who, When, What)

# RRoot, 01.01.2030, Created Script
```

```
#Anya Pryor 11/21/21 added some notes and renamed the
answers variable as all_answers
#--------------------------------------------------#

# -- data code -- #
v1 = 5
v2 = 10
lstData = [0, 0, 0, 0]
dicData = {"sum": 0, "diff": 0, "prod": 0, "quot": 0}

# -- processing code -- #
def CalcListValues(value1, value2, all_answers):
        all_answers[0] = value1 + value2
        all_answers[1] = value1 - value2
        all_answers[2] = value1 * value2
        all_answers[3] = value1 / value2

def CalcDictionaryValues(value1, value2, all_answers):
#dictionary allows you to use a key instead of the index
position
        all_answers["sum"] = value1 + value2
        all_answers["diff"] = value1 - value2
        all_answers["prod"] = value1 * value2
        all_answers["quot"] = value1 / value2
```

```
# -- presentation (I/0) code -- #
CalcListValues(value1 = v1, value2 = v2, all_answers =
lstData) #all_answers receives the reference or address for
the values
print(lstData)
```

```
CalcDictionaryValues(value1 = v1, value2 = v2, all_answers
= dicData)
print(dicData)
```

Global and Local Variables

Variables nested inside of functions are considered local, as they can only be called within the processing of the function. Global variables are those that can be used anywhere in the body of the script, this is the default state. It's important to name variables as such in order to attempt to reduce confusion. Python will assume that two variables with the same name are intended to be local instead of global. In this situation the global variable is said to shadow the local one, and it's likely to produce unintended results. The principle referenced here to guide programmers is encapsulation, or the effort to make code as simple and modular as possible. It's a best practice to name your variables to indicate if they are to be global or local.

Classes

Using a class command allows you to combine functions together. Here is the lab code I came up with:

```
# --------------------------------------------------#
# Title: Listing 17
# Description: Using classes
# ChangeLog: (Who, When, What)
# RRoot, 01.01.2030, Created Script
# ----------------------------------------------------#

# -- processing code -- #
class MathProcessor():
    """ functions for processing simple math """
    @staticmethod
    def AddValues(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 two numbers
        """
        return float(value1 + value2)

    @staticmethod
    def SubtractValues(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) sum of two numbers
        """
        return float(value1 - value2)

    @staticmethod
    def ProductValues(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 two numbers
        """
        return float(value1 * value2)

    @staticmethod
    def QuotientValues(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 two numbers
        """
        return float(value1 / value2)
```

```
# -- presentation (I/0) code -- #
print(MathProcessor.AddValues(5,10))
print(MathProcessor.SubtractValues(5,10))
print(MathProcessor.ProductValues(5,10))
print(MathProcessor.QuotientValues(5,10))
```

Debugging

To assist programmers in problem solving the makers of the program Python and the editing tool PyCharm have error messages, but PyCharm also has a debugging tool. It's accessed by putting in breakpoints by left clicking in the margin where you want the debugging tool to be active. Breakpoints need to be at a point where some action is being performed. The other commonly used commands in the debugging tool are step into my code and step over. Step into my code allows you to only review the code you have written and not the underlying Python code.

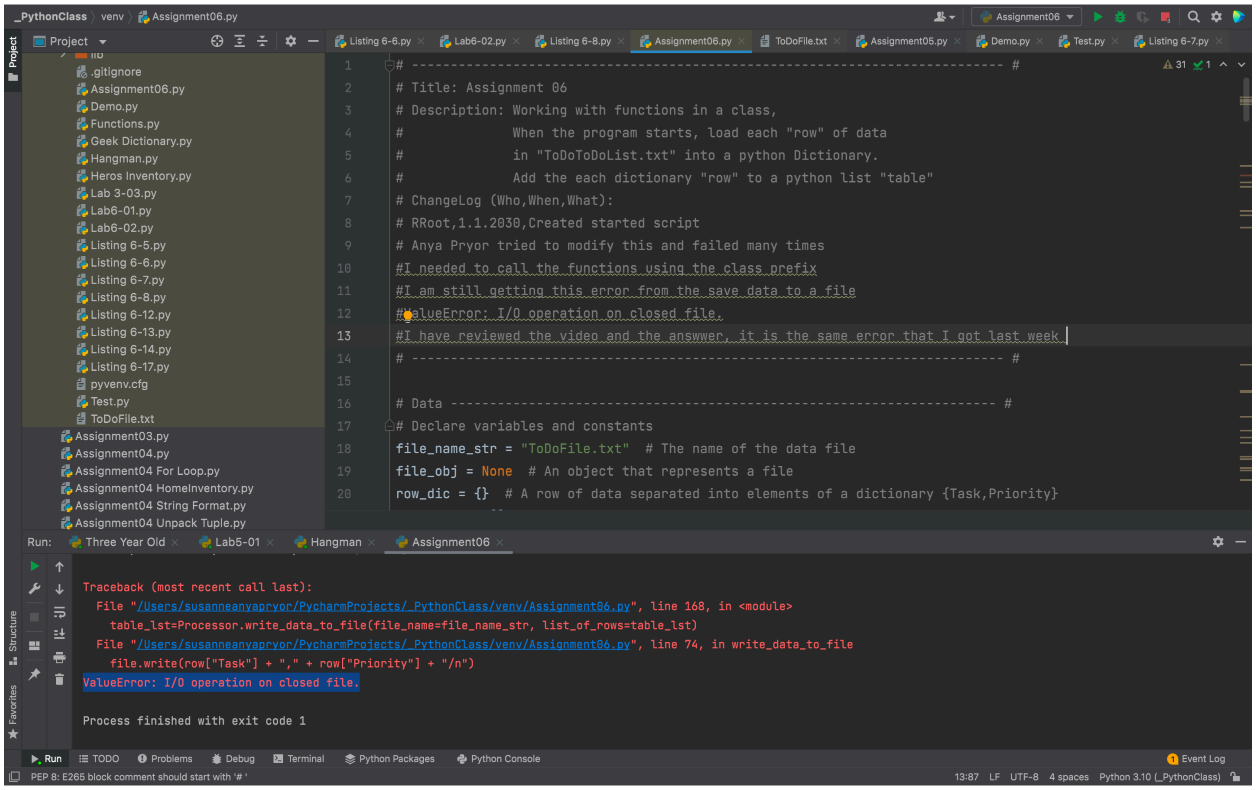
Git Hub

To begin using Git Hub you need to create a repository. Use the Create New File icon to create a folder. You can separate the file name from the folder name with a forward slash. Documents on Git Hub are usually in the format md. Once you create a file you commit it to add it to the repository.

To add files to a web page you need to navigate to Git Hub pages and point to the folder and file you have created. Here is my Git Hub page https://anyapryor.github.io/IntroToProg-Python-Mod06/

Assignment 06

I am still getting an error when I try to write to a file that this cannot be performed on a closed file. I have reviewed the answer posted and as this is the same error that I found last week I reviewed the video of that as well and I am not sure what is causing this. I also have still some trouble in distinguishing between arguments and parameters, as I am not sure which is which in this code. Here is a screen shot of the error:



```
# ---------------------------------------------------------------------------- #
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.txt" into a python Dictionary.
#              Add the each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# Anya Pryor tried to modify this and failed many times
#I needed to call the functions using the class prefix
#I am still getting this error from the save data to a file
#ValueError: I/O operation on closed file.
#I have reviewed the video and the answwer, it is the same error that I got last week
# ---------------------------------------------------------------------------- #

# 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}
```

```
Traceback (most recent call last):
  File "/Users/susanneanyapryor/PycharmProjects/_PythonClass/venv/Assignment06.py", line 168, in <module>
    table_lst=Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
  File "/Users/susanneanyapryor/PycharmProjects/_PythonClass/venv/Assignment06.py", line 74, in write_data_to_file
    file.write(row["Task"] + "," + row["Priority"] + "/n")
ValueError: I/O operation on closed file.

Process finished with exit code 1
```

PEP 8: E265 block comment should start with '# '

Here is the code:

```
#
-------------------------------------------------------
------------------------- #
# Title: Assignment 06
# Description: Working with functions in a class,
#               When the program starts, load each
"row" of data
#               in "ToDoToDoList.txt" into a python
Dictionary.
```

```
#               Add the each dictionary "row" to a
python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# Anya Pryor tried to modify this and failed many
times
#I needed to call the functions using the class
prefix
#I am still getting this error from the save data
to a file
#ValueError: I/O operation on closed file.
#I have reviewed the video and the answwer, it is
the same error that I got last week
#
---------------------------------------------------
------------------------- #

# 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



# 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 into a list of dictionary
rows
        :param task (string) task to add:
        :param priority (string) priority to add:
        :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()}
        list_of_rows.append(row)
        return list_of_rows

    @staticmethod
    def remove_data_from_list(task, list_of_rows):
        """ Removes data from a list of dictionary
rows
               :param task (string) task to remove:
               :param list_of_rows: (list) you want
to remove the task from:
               :return: (list) of dictionary rows
               """
        for row in list_of_rows:
            if row["Task"].lower==task.lower:
                list_of_rows.remove(row)
        return list_of_rows

    @staticmethod
    def write_data_to_file(file_name,
list_of_rows):
        """ Writes 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

              """

        file = open(file_name, "w")

        for row in list_of_rows:

            file.write(row["Task"] + "," +
row["Priority"] + "/n")

            file.close()

        return list_of_rows



# 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 5] - ")).strip()
        print()  # Add an extra line for looks
        return choice
```

```
    @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():
        """  Gather input from the user

        :parameter: task, priority
        :return: string
        """
        task = str(input("Enter a task: ")).strip()
        priority = str(input("Enter a Priority
(high, medium, low): ")).strip()
        return task, priority

    @staticmethod
    def input_task_to_remove():
        """  Gather input from the user

        :parameter: task
        :return: string
        """
        task= str(input("What item would you like
to remove? ")).strip()
```

```
        return task

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

# Step 1 - When the program starts, Load data from
ToDoFile.txt.
Processor.read_data_from_file(file_name_str,
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(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,prio
rity=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=fi
le_name_str, list_of_rows=table_lst)
        print("Data Saved!")
        continue  # to show the menu

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