Scott Duncan

11/17/2024

IT FDN 110 A Au 24: Foundations Of Programming: Python

Module 06 – Working With Classes, Functions, and Modularity

GitHub URL: https://github.com/gsdunca/IntroToProg-Python-Mod06.git

Introduction

In this week's Module Six we had the opportunity to explore Functions, Parameters, Arguments, their differences, use cases and how they and their values are implemented in a Python script. I found this week's material to be some of the most challenging presented so far. Although the learning curve was very steep, I can see the valuable uses for this type of code organization. With the structure of implementing functions in Python code, it opens a lot of options in the function's modularity provided they are constructed in a way that makes them more applicable to being reused in future scripts.

The Separation of Concerns Pattern

One of the key points I learned in this module is the Separation of Concerns Pattern. Being able to organize a script into areas of concern, or functions allows the developer to segment the areas of code within the script to their purposes. Creating an area of concern that focuses on the data storage, access, and its creation allows the developers to concentrate on specific areas of the script's functions. Other areas of concern include the logic, or core functionality of the script, as well as the presentation to the users for input and output of this data. In creating areas of concern, it allows the developers to logically separate the imbedded or encapsulated functions for each of these areas. In our assessment this week we had two primary areas of concern, file processing, and Input/Output. These areas of concern were divided into classes. **(Figures 1.1 & 1.2)**

```
# Processing Class ---(reads and writes Data to and from a file)-----------
class FileProcessor:
    """
    A collection of processing layer functions
    - Read files into memory
    - write to files with updated data from memory
```

Figure 1.1: FileProcessor Area of Concern.

```
# Presentation Class ---(Contains various Input/Output functions for user interactions---
class IO:
    """
    A collection of presentation layer functions that manage user input and output
```

Figure 1.2: Input/Output Area of Concern

The Classes

By the use of Classes, the developers code can be separated into areas of function. As an example, a class defined as “FileAccess” could contain functions that focus on reading and writing data to and from files. In this week’s readings the example of developing a calculator function was used to explain how classes can nest or encapsulate functions. I found this very confusing due to the naming of the classes and functions were so similar only differentiated by letter casing. I found it was more palatable in my understanding after I had created the assignment where my functions were held within their appropriate classes as can be seen in **(Figure 2).**

Functions allow for greater organization and modularity

The greatest skill gained in Module Six was the ability to implement Functions, or sections and blocks of code that have a function, error checking and variable definition all encapsulated in a single function or block of code. By encapsulating a single purpose or use in a function this allows for greater modulatory between scripts. As an example, in creating this week’s assignment program, I was able to reuse functions from other scripts. With minor manipulations I was able to adapt the function to my own purpose. **(Figure 2)**


```
class FileProcessor:
    """
    A collection of processing layer functions
    - Read files into memory
    - write to files with updated data from memory

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    SDuncan,11/16/24,Updated for Assignment06
    """
    pass
    # When the program starts it loads the .json file
    # Extract the data from the file

    @staticmethod # Reads data from file into memory
    def read_data_from_file(file_name: str, student_data: list):
        """
        reads the contents of a file into memory

        ChangeLog: (Who, When, What)
        RRoot,1.3.2030,Created function
        SDuncan,11/16/24,Updated for Assignment06
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "\n ERROR:" "\n File must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 2: Example of Code Reuse and Adaptation

Although this can be seen as plagiarism or cheating, I have learned that in the realm of software development, this is common practice. I can only assume that due to the descriptive document strings embedded in the functions showing the source and change log, it provides credits to the previous developer from where the code originated. Although this Copy/Paste mentality presented a personal struggle with my moral character, it was necessary due to my inexperience with writing Python scripts from scratch.

Parameters and Arguments

Within this week's learning material, the static class was defined in the coding portion of the assignment. With the use of the static class (denoted by @staticmethod) "allows for the use of class functions directly without the need to create an object first." *(Python 110 Module 06 Notes p29)* in this week's assignment this was configured as a function with the variables directly. **(Figure 3)**

```
@staticmethod # eads data from file into memory
def read_data_from_file(file_name: str, student_data: list):
    """
    reads the contents of a file into memory
```

Figure 3: Static Function

Within the static function we passed the values for the parameters directly into the function allowing the code to reference the file name and the data contained within directly.

Global vs Local variable scopes

In this assignment two different types of variable scopes were used. For the constants (variables that didn't vary) and standard variables, what was previously configured on past assignments were moved from the root, or the global portion of the script into the functions directly. The movement of these variables into the functions changed the scope of these variables from global to local, local being they are now considered to be local to the function itself and not referenced as a global variable. Other variables such as the constants, which are available for referencing by all functions and all scopes or classes, were left at the root of the script referencing them as a global variable. **(Figure 4)**

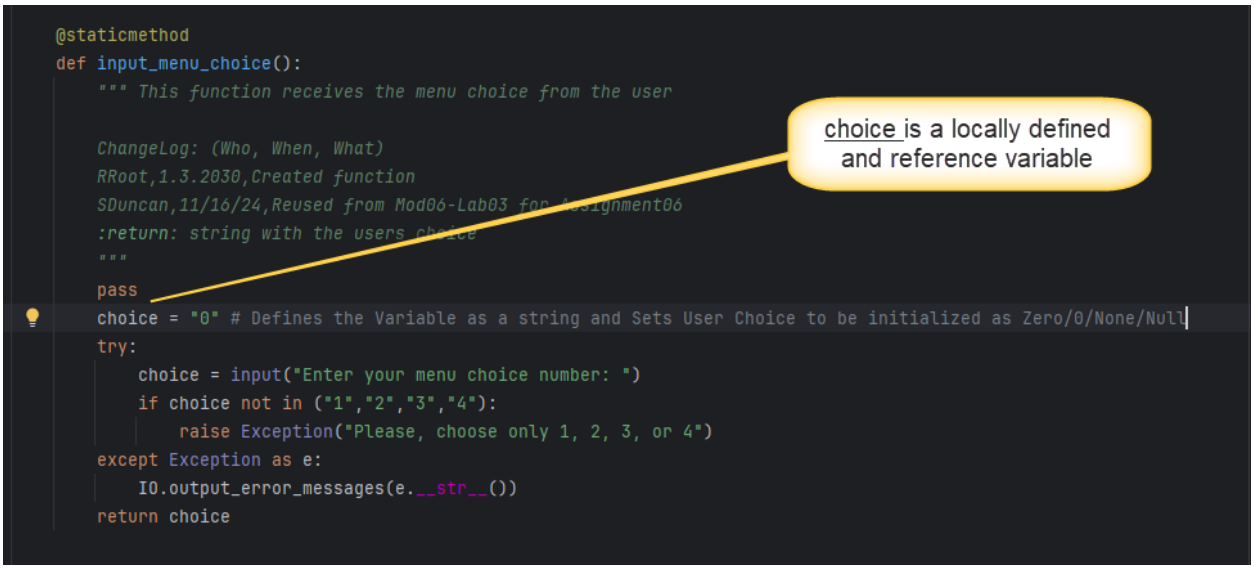
```
# ------------------------------------------------------------------------
import json

# Define the Data Constants
FILE_NAME: str = "Enroll
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------
'''
# Define the Data Variables and Constants
students: list = []  # a list of student data
menu_choice = '' # user's menu choice initialized as null/none
```

Global Scope
Constants available to all functions.

Global Scope
Variables available to all functions.

Figure 4: Global Variables and Constants

Variables referred to only the functions themselves were placed within the function changing the scope of these variables to be referred by the function they were contained within changing them from a globally referenced variable to a locally referenced variable **(Figure 5).**




```
@staticmethod
def input_menu_choice():
    """ This function receives the menu choice from the user

    ChangeLog: (Who, When, What)
    RRoot,1.3.2030,Created function
    SDuncan,11/16/24,Reused from Mod06-Lab03 for Assignment06
    :return: string with the users choice
    """
    pass
    choice = "0" # Defines the Variable as a string and Sets User Choice to be initialized as Zero/0/None/Null
    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1","2","3","4"):
            raise Exception("Please, choose only 1, 2, 3, or 4")
    except Exception as e:
        IO.output_error_messages(e.__str__())
    return choice
```

Figure 5: Locally referenced variable

This section of code was reused from a previous lab, and although it seems odd that the variable is not discreetly defined, this statement appears to define it as a local string and set the value to Zero.

Document Strings

As a developer skill it was requested in this assignment that we begin noting various functions and classes within our Python programming with document strings. These strings become useful in tracing the origon of sections of code and attempting to understand the developer's intent. Moreover, it becomes an important tool in the reuse of blocks of code and the changes or customizations made when reused in new Python programs. It becomes useful in that when a developer can hover over an area in the script and read a verbiage a previous developer has left, and for the new developer to update and leave information for the next. "*For example, if you hover over the print() function and wait (or **use ctrl + q to activate** this option in PyCharm) it will display this pop-up message using the function's docstring*." (RR, 2024, Mod06-Notes.docx pg30) **(Figure 6)**

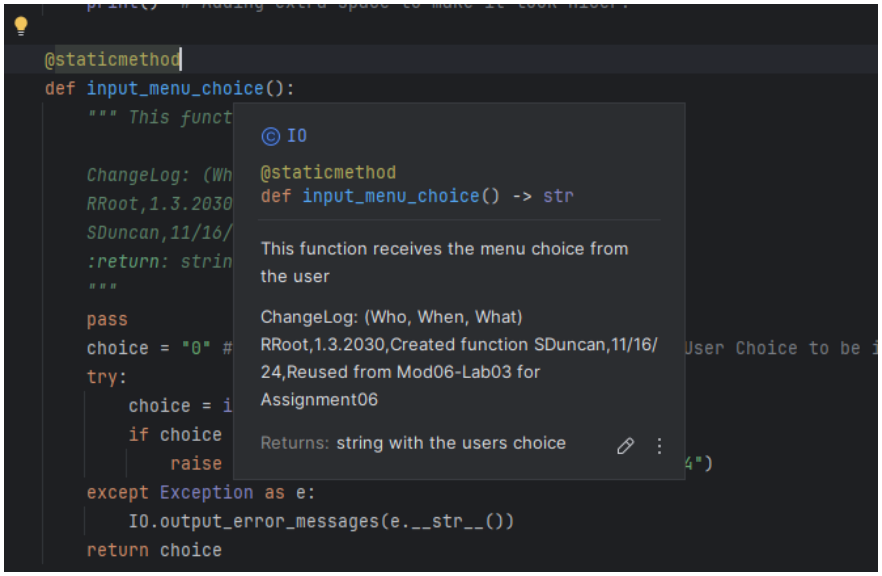


Figure 6: Document String Pop-UP

Reduced Main Body

With the development of functions, the main body of the script is greatly reduced, and the programs efficiency appears to be increased as well. What was complex in our previous learning modules appears to have become streamlined and very easy to follow (Figure 7).

```
# Beginning of the MAIN BODY of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
#students holds the List of data loaded from the file, referenced in constant FILE_NAME

# Present and Process the data
while (True):
    IO.output_menu(menu=MENU) #ties menu parameter in "output_menu" function to the constant MENU
    menu_choice = IO.input_menu_choice() #Stores the user's menu choice option/references the correct choice

    if menu_choice == "1":  # Allow the User to input new data
        IO.input_student_data(student_data=students)
        continue

    elif menu_choice == "2":# Presents the current data to the user
        IO.output_student_data(student_data=students)
        continue

    elif menu_choice == "3":# writes the contents of memory to the data to a file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    elif menu_choice == "4": # Stop the loop
        break  # out of the loop
print("Program Ended")
```

Reduced Complexity

Figure 7: Main Program Body

Online Document Post location

As a requirement for this course this document, the associated python script and data file have been uploaded to the following URL https://github.com/gsdunca/IntroToProg-Python-Mod06.git

Summary

In this latest learning module, the concept of code organization, internal code documentation, segmentation and reuse was explored. With the idea of segmenting the code into areas of concern allows the program to be separated into modular areas that are focused on specific tasks with possible reuse. The primary areas of concern centered around data storage and access, data manipulation, and presentation to the user. To achieve this modularity concept the reorganization of variables was introduced to set some variables local to the functions, while other variables and constants had a broader use and were to be considered global in nature. After the outline of the program was conceived a common practice of setting the function definitions with the keyword pass allowed for a general outline of logic to be achieved in the development of the script.

Although the concept of cut/paste from another developer's code boarders on plagiarism, I assume because the developers notes and credits are maintained in the code documentation trail. It is an accepted practice within the industry.

This practice of segmenting code for logical process organization and reuse can be a most valuable methodology in developing my own python scripts.