David Goldberg

May 18, 2024

Foundations of Programming: Python

Assignment 06

Link to Github: https://github.com/david-goldberg26/IntroToProg-Python-Mod06.git

Functions and Classes

Introduction

This week we learned a great amount in module 06, it taught us important ways to organize our code and make our scripts more followable by the user. This week we learned the importance of functions and how to apply them to our existing code. by using functions and overarching classes. Functions allow for logic that is used multiple times throughout the script to be abstracted, allowing it to be called multiple times. We also learned the importance of classes and how we could implement them as well. It's like a function, except it groups together multiple functions inside of it. This module contains extremely important topics that will allow us to build more complicated scripts in the future.

Functions

In Python, a function is essentially a block of code that can be reused throughout a script. It can be called as many times in your script which allows your code to be organized, more readable, and manageable. An example of three functions are shown below

```
2  def read_data_from_file():
3      pass
4
5
6  def add_data_to_table(): ...
8
9
0  def write_data_to_file():
1      pass
```

Figure 1: Simple Function

The example above shows the format of the function line, where 'def' is a keyword used to define a function, 'read_data_from_file' is the name of the function, and whatever is in the parenthesis is the parameter that takes in the function. 'Pass' is a placeholder for the code you would want to insert into the function. Using variables inside functions are a bit tricky, there are

local variables and global variables, local variables are simply defined inside the function and are used to store temportary data needed for the function to perform its task, where global variables are defined outside the function and are acceessable from any function. The example below shows how global variables are used.

```
4   FILENAME = "MyLabData.json"
5   students : list[dict[str,str | float]] = []
6   file: TextIO
7
8
9
10  def read_data_from_file():
11      global file
12      global students
13      global FILENAME
```

Figure 2: Variables in Functions

To call global variables inside the function, you need to write ‘global’ in front of it to not confuse it with a local variable. Since the variables are defined outside of the function, they can be called global variables inside the function. Parameters are like local variables, except they are variables that are listed inside the parentheses in the function definition which allow you to pass information into the function when it is called, but they are also local to the function which they are defined in, so it only exists in the scope of that function.

```
45  def get_menu_option(menu:str)->str:
46      string_choice = input(menu)
47      while string_choice not in ['1', '2', '3', '4']:
48          OIProcessor.output_message("please enter and option between 1 and 4")
49          string_choice = input(menu)
50      return string_choice
```

Figure 3: Use of Parameters

In the example above we can see that ‘menu’ is a parameter that is called through out the function, but it is not returned to other parts of the code. Actually, in this case, the local variable ‘string_choice’ is returned, which it can then be used in other parts of the scrip. Return statements are also extremely useful in functions. They are used in a function to send a value back to the caller. When the function reaches a ‘return’ statement, it immediately returns that specific statement to the caller, wherever it’s located. Next, we learned about classes and how they can make our code more organized by organizing functions

Classes

Using a class is a way to group related pieces together, such as multiple functions. It is a blueprint for creating objects or instances, such as multiple variables and functions. An object is an instance of class, which holds data and can perform operations that are defined inside of the

class. Classes also use methods, which are essentially functions defined inside the class. An example of a class is shown below.

```
299  class Dog:
300      # Class attribute
301      species = "Canis familiaris"
302
303      # Initializer / Instance attributes
304      def __init__(self, name, age):
305          self.name = name
306          self.age = age
307
308      # Instance method
309      def description(self):
310          return f"{self.name} is {self.age} years old"
311
312      # Another instance method
313      def speak(self, sound):
314          return f"{self.name} says {sound}"
315
316  # Creating an instance of the Dog class
317  my_dog = Dog("Kelso", 3)
318
319  # Accessing attributes and methods
320  print(my_dog.name)          # Output: Kelso
321  print(my_dog.age)           # Output: 3
322  print(my_dog.description()) # Output: Kelso is 3 years old
323  print(my_dog.speak("Woof")) # Output: Kelso says Woof
```

Figure 4: Classes

In this example, there is an overarching class called ‘dog’ with three functions inside of it. The class attribute, which is ‘species’, is shared by all the instances of the class. Then there are the instance methods which can be called on instances of the class. The first function initializes the instance attribute, where ‘self’ refers to the specific instance of the class. Then the other methods, ‘description’ and ‘speak’ can be called on instances of the class. Line 317 then creates and uses an object, it creates and instance of the ‘dog’ class with the name being Kelso and the age being 3. Overall, classes are a great way to group pieces of objects.

Writing the Script

The Assignment below has us take module 05’s assignment and add the use of functions, classes, and the separation of concerns. The functionality of the script is identical to last week, it will read in data from a JSON file and store that to a list of dictionaries, then present an interactive menu for the student to choose form. It will ask the student to input their name and course name and will create a dictionary with those inputs, it will then write the list of dictionaries the JSON file. Taking this functionality and adding classes and function makes the script more organized. Below I will go through all the functions as well as the main body.

```python
import json
from typing import TextIO

# Define the Data Constants
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 Constants
# FILE_NAME: str = "Enrollments.csv"
FILE_NAME: str = "Enrollments.json"

# Define the Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.

```

Figure 5: Constants and Variables

As always, it is important to include your constants and variables above. It is important to include variables and constants you want to use in your main body before the usage of classes, in case you want to use global variables. You can notice there are way less variables, this is because most of the variables are used locally inside of the classes and its methods. Below is the first object called FileProcessor which is used for processing the JSON file.

```python
class FileProcessor:
    @staticmethod
    def read_data_from_file(file_name:str, student_data: list):
        '''
        Function that reads in data from a JSON file and then into a dictionary
        :param file_name: A string indicating the file name
        :param student_data: dictionary from students inputs
        :return: list of student data

        '''
        file: TextIO = None
        try:
            file = open(file_name, "r")     # open the file in read mode
            student_data = json.load(file)      # load the previous dictionaries
            print(student_data)
            file.close()
        except Exception as e:
            IO.output_error_message(message='text file not found\n', error = e)
        finally:
            if file == False:
                file.close()
        return student_data
```

```python
    @staticmethod
    def write_data_to_file(file_name:str, student_data:list):
        '''
        Function that reads the students dictionarties into the JSON File
        :param file_name: A string indicating the file name
        :param student_data: dictionary from students inputs

        '''
        try:
            file = open(file_name, "w")     # open a file to write to
            json.dump(student_data, file)       # write the dictionary/dictionaries to the json fi
            file.close()
            print("The following data was saved to file!\n")
            IO.output_student_courses(student_data=student_data)        # calling IO class to outp
        except Exception as e:      # Structured error handling if any exceptions occur
            if file.closed == False:
                file.close()
            IO.output_error_message(message='There was a problem with writing the file', error=e)
        finally:
            if file.closed == False:
                file.close()
```

Figure 6 & 7: File Processor Class

Creating a class for processing files is perfect for this assignment because we need to read in data and write data to a JSON file. Having this class allows for us to make two methods or functions inside of it. The first method is called read_data_from_file which holds two parameters, file_name and student_data. This method will read in existing data from the JSON file and return the list of dictionaries so it could be called in the main body of the script. If there are errors, a method from another class is used called output_message_error which handles all the structured errors. The next method is called write_data_to_file which uses the same parameters, will write data to the JSON file. If there are any errors it will call the IO class which has an error handling method. This creates a simpler script without having error print statements everywhere. @staticmethod is seen at the top of every method which is a method that belongs to the class but does not have access to certain instances. They are often used to create utility for functions that perform tasks related to the class. Now that the FileProcess class is complete, there is a second class called IO (input/output) which handles all the functions that deal with inputs and outputs.

```python
class IO:
    @staticmethod
    def output_error_message(message:str, error: Exception = None):
        '''
        Function displays prints error statements
        :param message: string of data containing a message
        :param error: message that pertains to the Exception

        :return: None

        '''

        print(message)
        if error is not None:       #Print error messages from the exceptions
            print('--- technical info--\n')
            print(error, error.__doc__, type(error), sep = '\n')
```

Figure 9: Error Outputs

The new class IO is shown in the figure above with its first method called output_error_message which will print any ‘messges’ that are called by this method in other methods. If it happens to

encounter an error, it will go ahead and print the exception to the user. This method can be called wherever exceptions/error can occur, such as in the FileProcess class.

```
96      @staticmethod
97      def output_menu(menu:str):
98          '''
99          Function Displays menu choices to students
100         :param menu: student menu choice
101         :return: None
102         '''
103         print('\n')
104         print(menu)     # will print the menu
105         print('\n')
106
107     @staticmethod
108     def input_menu_choice():
109         '''
110         Function allows students to input student choice
111         :return: students string choice
112         '''
113         string_choice = input("Enter a choice from the menu: ")     #allows stu
114         while string_choice not in ['1', '2', '3', '4']:        # checks if inp
115             IO.output_error_message("Please enter an option between 1 and 4")
116             string_choice = input('Enter a valid menu choice: ')        # will
117         return string_choice
```

Figure 10: Menu Output & Choice

The next two methods shown above are output_menu which has a parameter ‘menu’ which will be shown to the student when they run the script, and the next method is input_menu_choice which will ask the student for their input, inside of this method it will also make sure the input is between 1-4 where in Assignment 05 the check was done at the end of the while loop. Condensing these actions into one method makes it easier for the user who is going to look at this code next. It also calls output_error_message just incase the user selects an improper menu option.

```
119     @staticmethod
120     def input_student_data(student_data):
121         '''
122         Function allows for students to input their names and courses
123          :param student_data: list of dictionaries that are filled by students inputs
124         :return: dictionary (list)
125           '''
126         student_first_name: str = ''
127         student_last_name: str = ''
128         course_name: str = ''
129         student: list = []
130         try:
131             student_first_name = input("Enter the student's first name: ")      # student will input fi
132             if not student_first_name.isalpha():        # checks if input doesnt have a letter
133                 raise ValueError("The last name should not contain numbers.")
134             student_last_name = input("Enter the student's last name: ")        #student will input the
135             if not student_last_name.isalpha():      # checks if input doesnt have a letter
136                 raise ValueError("The last name should not contain numbers.")
137             course_name = input("Please enter the name of the course: ")
138             student = {"FirstName": student_first_name,
139                        "LastName": student_last_name,
140                        "CourseName": course_name}      # builds dictionary of students inputs
141             student_data.append(student)
142             print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
143         except ValueError as e:
144             IO.output_error_message(message= "inputted name is not the correct data type", error=e)
145         except Exception as e:
146             IO.output_error_message(message= "Error: Problem with entering your data", error=e)     # c
147         return student_data
```

Figure 11: Student Input

This method will be used to input the students first name, last name, and course name where it will also check for any possible errors using the output_message_error method. The variables that are used such as student_first_name, student_last_name, course_name, and student are defined locally in this method since they will not be used anywhere else in the script. Once the list of inputs is saved to 'students' it will then be appended to the dictionary 'student_data' which is returned at the end of this method and used in the main body of the script.

```
149     @staticmethod
150     def output_student_courses(student_data:list):
151         '''
152         Function is to output the students compelete dictionary
153         :param student_data: list of students inputs
154         '''
155         print("-" * 50)
156         for student in student_data:
157             print(f'Student {student["FirstName"]} '
158                   f'{student["LastName"]} is enrolled in {student["CourseName"]}')
159         print("-" * 50)
```

Figure 12: Output the Data

The last method used in the class IO is output_student_courses which will go through each entry in the dictionary and print the information to the student. As it is quite simple, it still makes sense to have its own method just to make the main body of the script simple and fluid. Next, lets go over the main body of this script

```
162 # Main Body
163
164 students = FileProcessor.read_data_from_file(FILE_NAME, students)
165
166 while True:
167
168     IO.output_menu(menu=MENU)       # calls IO class and a method
169     menu_choice = IO.input_menu_choice()        # calls IO class
170
171     if menu_choice == "1":
172         students = IO.input_student_data(students)      # if choi
173         continue
174
175     elif menu_choice == "2":
176         IO.output_student_courses(students)     # if choice is 2
177         continue
178
179     elif menu_choice == '3':
180         FileProcessor.write_data_to_file(FILE_NAME, students)
181         continue
182
183     elif menu_choice == '4':        # break out of the program an
184         break
185
186 print('End of program')
```

Figure 13: Main Body

Above is the main body of the script, basically unrecognizable from last weeks assignment. This is because of the classes and methods being used earlier in the script and the main method just

calls those functions. First the FileProcessor class is called with the read_data_from_file function and uses FILE_NAME and students as the parameters. Then once the while loop begins, we use the IO class with the output_menu function to output the menu to the student and also calls the input_menu_choice function to allow the student to input a menu_choice. If the student chooses 1 it will call the input_student_data function and allow for student name inputs. If 2 is selected, the function output_studend_data will be called to print the student choices. If 3 is selected, the FileProcessor class will be called with the write_data_to_file function to write data to the JSON file. Then option 4 will break the loop and exit the script.

Summary

Overall, this module and assignment were extremely important for understanding functions, classes, and separations of concerns. The use of classes and functions allows for a very clean and easy to read script where the functions contain all the functionalities of the script. This allows the main body of the script to be very straightforward and easy to understand for other users. Knowing this information will allow me to write more complex scripts in the future because these topics really helped me with script organization and process handling.