Joseph Watts
March 5, 2025
IT FDN 110 A Wi 25: Foundations Of Programming Python
Assignment06
GitHub Repository: https://github.com/JBWpro/IntroToProg-Python-Mod06

Basics of Python: Classes, Functions, SoC

Introduction

As our scripts grow longer they get harder to manage. Also, we keep finding ourselves writing the same code blocks over and over again. This is where functions, classes, and arguments come in handy, as well as the concepts of Separation of Concerns.

Functions

A function is a reusable block of code that allows us to perform complex tasks with just one line of code. By defining your function, you can call on that function later in your script and it will run all the code within your function block. This allows you to create more modular scripts, meaning you can focus on having your functions complete specific tasks. This also allows you to re-use your function in other scripts if you want to accomplish the same thing. For example, each week we have typed up a similar block of code to prompt the user for input. For week four it looked something like this:

```
(student_first_name, student_last_name,course_name) = (
               input("\nEnter the student's first name: "),
               input("Enter the student's last name: "),
               input("Please enter the name of the course: "))

#Creates list, appends list to students list
student_data = [student_first_name, student_last_name, course_name]
students.append(student_data)
```

But now with functions, you can define your input statements like so:

```
def input_student_data(student_data: list):
   try:
       student_first_name = input("Enter the student's first name: ")
       if student_first_name == "":
           raise Exception("Please enter a first name")
       if not student_first_name.isalpha():
           raise ValueError("Please do not include numbers in your name.")

       student_last_name = input("Enter the student's last name: ")
       if student_last_name == "":
           raise Exception("Please enter a last name")
       if not student_last_name.isalpha():
           raise ValueError("Please do not include numbers in your name.")

       course_name = input("Please enter the name of the course: ")
       if course_name == "":
           raise Exception("Please enter a course name")
```

```
        # Process input data into dictionary
        student = {"FirstName": student_first_name,
                        "LastName": student_last_name,
                        "CourseName": course_name}
        student_data.append(student)
```

And now any time you want to prompt the user for information, you can just call upon the function like so:

```
students = IO.input_student_data(student_data=students)
```

One important thing to note on functions is that the code that defines the function must exist before it is called, meaning we need to include our definitions toward the top of the script.

Parameters

Parameters are used to pass data into your function. They act as local variables within the function and are only used when the function is running. When defining a function, your parameters are what is encapsulated in the parentheses like so:

```
def write_data_to_file(file_name: str, student_data: list):
```

Then when you call on the function, you can set both file_name and student_data to the variable you are using like so:

```
FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
```

Values assigned this way are called arguments. Using parameters and arguments make it clear what data a function needs to run. It also allows you to call the function using different file names and data without having to modify your function.

Return

Using the return statement allows your to return values from your functions like so:

```
def input_menu_choice():
   menu_choice = "0"
   try:
       menu_choice = input("Enter Menu Selection: ")
       if menu_choice not in ("1", "2", "3", "4"):
           raise Exception("Please choose either 1, 2, 3, or 4")
   except Exception as e:
       IO.output_error_messages(e.__str__())
   return menu_choice
```

```
menu_choice=IO.input_menu_choice()
```

As you can see, in the definition we return menu_choice. When we call on the function, set it equal to whatever variable you desire to capture the returned data. Using return is helpful because it makes clear what data you are returning. It is also best practice to return a new object rather than modify an existing one.

Classes

Classes are a way of further organizing your code while using functions. Classes allow you to group functions, variables and constants by the name of the class. This promotes more modularity, as you can group similar functions under the same class. One thing to remember about classes is that if your function code is not changing, you can indicate that using the @staticmethod. This will allow you to call on your functions within the class directly, where otherwise you would have to create an object first. Here is an example of defining and then using a class:

```
class FileProcessor:
   """
   Class containing functions responsible for processing file information
   """

   @staticmethod
   def read_data_from_file(file_name: str, student_data: list):
       try:
           file = open(file_name, "r")
           student_data = json.load(file)
           file.close()
       except FileNotFoundError as e:
           IO.output_error_messages("JSON file must exist before running this
script!", e)
       except Exception as e:
           IO.output_error_messages("There was a non-specific error, oh no!",
e)
       finally:
           if file.closed == False:
               file.close()
       return student_data
```

```
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)
```

You may notice the text within the “”” “””, this is called a docstring. Any text contained within will display when you hover over the called method. This allows you to provide more information to anyone using your methods!

Separation of Concerns

Separation of concerns, or SoC, is a software design principle that when followed increases the maintainability, scalability and readability of your code. The idea is that you break your code down into distinct self contained components. Having each component be responsible for a specific function makes it easier to manage and read. Using SoC, modifying a concern should not impact another, making editing large scripts easier. Three common types of concern are presentation concern, logic concern and data storage. Presentation has to do with presenting information to the user, as well as receiving information from them. Logic is the core functionality of the program, including data processing, logic and decision making. Data storage deals with how and where you are storing your data.

Running the Script

Running the script through either IDLE or the commands will first import the json module. Then it will read all the class and function definitions we have. Then it will read the file "Ennrolments.json" that sits within your directory. Using the json module, the data will be saved to the students dictionary. After the data is read a while loop starts. It will print a menu of options, and prompt the user to enter an option from the menu. If they select the first option, they will be asked to enter their first and last name as well as the course name. After that the while loop starts again. If they then pick the second option the script will print the names of all the registered students and their class, including any names you may have just entered. If the third option is selected, the data is written to a file and the data being saved is printed into the terminal. Again we are using the json module to write our dictionary to our json file. Selecting option four exits the program. You may choose any of these options as many times as you like. Entering a number not between 1 and 4 will remind the user that they must choose one of those options.

Summary

This assignment introduces functions, classes, parameters, and Separation of Concerns (SoC) to improve code organization and reusability. Functions allow for modular programming by grouping code into reusable blocks, reducing redundancy and improving readability. They take parameters as inputs and can return values using return, making them adaptable to different data. Classes further organize code by grouping related functions and variables, promoting modularity and maintainability. The @staticmethod decorator allows calling class functions without creating an instance. Separation of Concerns (SoC) enhances code structure by dividing it into distinct components—presentation, logic, and data storage—making scripts easier to manage, scale, and modify.