Chantal Van den bussche

3/5/2025

Foundations of Programming: Python

Assignment06

https://github.com/chantevan/IntroToProg-Python-Mod06

Functions and Classes

Introduction

This week, I learned about functions and classes which are ways to organize code so that it is more maintainable for the long-term. I also learned about Separation of Concerns which is a specific pattern to organizing classes and functions based on their purpose.

Functions

Functions are useful due to their modularity and reusability. They are a reusable grouped set of programming statements that perform a specific task and can be recalled later in the main body of the program by the function name. In this way, you are able to keep your program clean and also avoid rewriting the same set of instructions over and over in your program. Instead, you can just call the function.

In my Assignment06, I created a function (**output_error_messages**) for printing error messages (Example 1). This function was then called in other blocks of code to display an error message to the user if an error occurred while running the program (Example 2).

Example 1:

```
def output_error_messages(message: str, error: Exception = None):
    """
    A function that handles error messaging

    :param message:
    :param error:
    :return:

    ChangeLog: (Who, When, What)
    Chantal Van den bussche, 3/5/2025, Created Function
    """
    print(message, end="\n\n")
```

```
   if error is not None:
       print("-- Technical Error Message -- ")
       print(error, error.__doc__, type(error), sep='\n')
```

Example 2:

```
try:
   file = open(FILE_NAME, "r")
   student_data = json.load(file)
   file.close()
except Exception as e:
   # calling the function output_error_messages
   IO.output_error_messages(message= "ERROR! There was a problem "
                                     "reading your file.", error= e)
```

Global vs Local Variables

Variables are categorized as either global or local depending on the location of their declaration within the code. Global variables are declared outside of any function (Example 3). They are accessible and usable throughout the entire script (Example 4). In contrast, local variables are declared within a function and can only be accessed and used within that specific function (Example 5).

Example 3:

```
# Define the Global Data Variables
menu_choice: str  # Hold the choice made by the user.
```

Example 4:

```
# Present and Process the data
while (True):

   # Present the menu of choices
   IO.output_menu(menu=MENU)
   # Global Variable is used in main body
   menu_choice = IO.input_menu_choice()

   # Input user data
```

```
    # Global variable is used again in main body
    if menu_choice == "1":  # This will not work if it is an integer!
```

Example 5:

```
def input_menu_choice():
    """
    A function that takes user input for their menu choice

    :return:

    ChangeLog: (Who, When, What)
    Chantal Van den bussche, 3/5/2025, Created Function
    """
    # choice is a Local Variable
    choice = "0" # Needs to be string!!
    try:
        choice = input("Enter your menu choice as a number: ")
        if choice not in ("1","2","3","4"):
            raise Exception("Please enter only 1, 2. 3, or 4.")
    except Exception as e:
        IO.output_error_messages(e.__str__()) # Avoiding technical error message
    return choice
```

Parameters

Data can be passed into a function by creating parameters. This is generally a safer and better way to pass data into a function than using a global variable within the function. This is because by using parameters, you encapsulate (bundle data and processes that work on that data) the data which reduces the likelihood of errors. Parameters also increases flexibility (you can call the function with different messages for example) and predictability (global variables' values could change). In Example 6, **messages** and **error** are my parameters. In Example 7, I demonstrate parameter flexibility by providing custom argument data for **messages** when I call the function.

Example 6:

```
def output_error_messages(message: str, error: Exception = None):
```

Example 7:

```
except Exception as e:
    # calling the function output_error_messages
    IO.output_error_messages(message= "ERROR! There was a problem "
                                      "reading your file.", error= e)
```

Return Values

To capture the value(s) that functions can return, you "return values". By doing this, you make the function act as if it is an expression. In this way, you are able to use the returned value of the function without storing it in a variable.

You can also choose to store the returned value in a variable. For example, in the function **input_menu_choice**, I store the returned value of the user's chosen menu choice in the local variable **choice** (Example 8). This makes it easy to reuse the return value of the function throughout your script. In Example 9, **menu_choice** is equal to the function **input_menu_choice**. This effectively means that **menu_choice** is equal to the return value of **choice**.

Example 8:

```
def input_menu_choice():
    """
    A function that takes user input for their menu choice

    :return:

    ChangeLog: (Who, When, What)
    Chantal Van den bussche, 3/5/2025, Created Function
    """
    choice = "0" # Needs to be string!!
    try:
        choice = input("Enter your menu choice as a number: ")
        if choice not in ("1","2","3","4"):
            raise Exception("Please enter only 1, 2. 3, or 4.")
    except Exception as e:
        IO.output_error_messages(e.__str__()) # Avoiding technical error message
    return choice # return value stored in a local variable
```

Example 9:

```
# input_menu_choice returns choice

# Therefore menu_choice = choice

menu_choice = IO.input_menu_choice()
```

Using return values has several benefits compared to modifying mutable objects within the function. Using return values promotes clarity, predictability, encapsulation, immutability, reusability, error handling, and chaining.

Classes

Classes group functions, variables, and constants. It is a way of organizing your script to keep it modular, readable, and clean. Whereas a function groups code to perform a specific task, you can think of classes as a way of grouping function, variables, and constants by their overall “big picture” goals. More on this in the Separation of Concerns section.

Static Classes

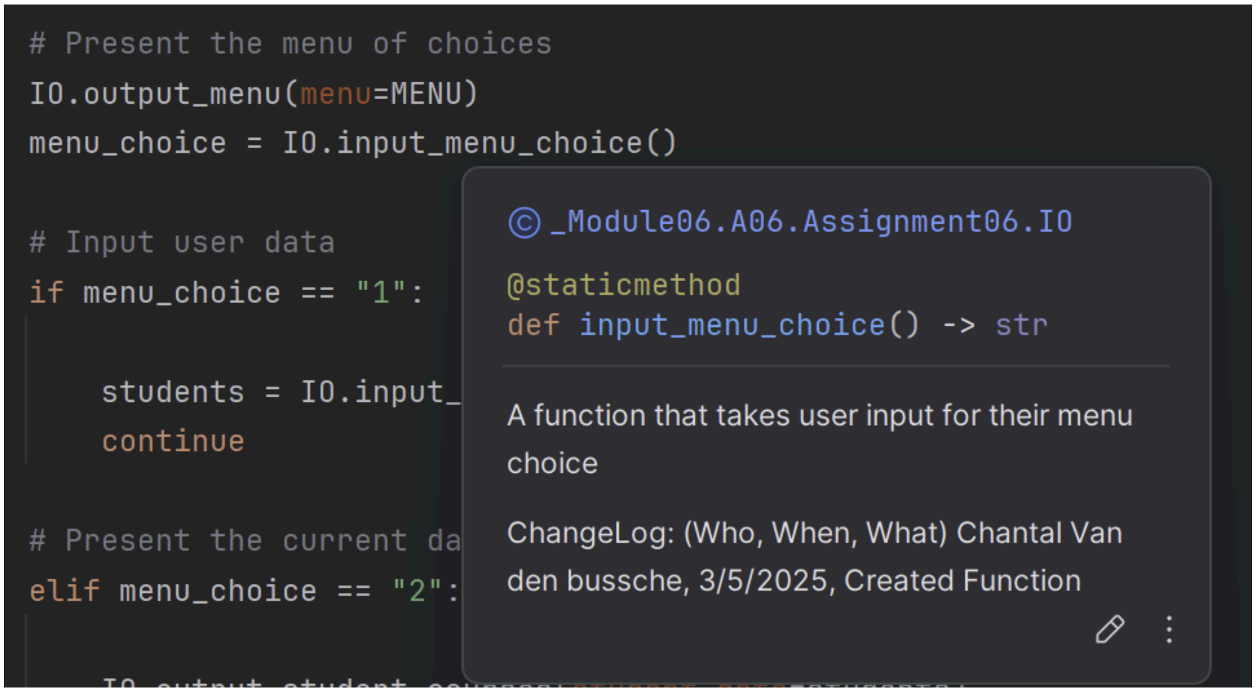
If a function code never changes, it is “static”. You can therefore use the @staticmethod decorator which will allow you to forgo creating an object first and instead use class functions directly. In my Assignment06, I used the @staticmethod decorator on my functions to inform Python they were static functions (Example 10).

Example 10:

```
@staticmethod

def output_student_courses(student_data: list):

   # The following comment in green is my docstring

   """

   A function that display registered students to their enrolled courses


   :param student_data:

   :return:


   ChangeLog: (Who, When, What)

   Chantal Van den bussche, 3/5/2025, Created Function

   """

   # Process the data to create and display a custom message

   print("-" * 50)

   for student in student_data:

       print(f'Student {student["FirstName"]} '
            f'{student["LastName"]} is enrolled in {student["CourseName"]}')
   print("-" * 50)
```

Document Strings (docstrings)

Example 10 not only displays the use of the @staticmethod decorator but also displays the creation of docstring. Docstring is a way for developers to include additional notes about their functions and classes. This way, you can hover over the class or function name and learn more about it (Figure 1). This can be very useful if you have a long script. That way you do not constantly have to scroll up and down to remind yourself what this or that function/class does.



```
# Present the menu of choices
IO.output_menu(menu=MENU)
menu_choice = IO.input_menu_choice()

# Input user data
if menu_choice == "1":

    students = IO.input_
    continue

# Present the current da
elif menu_choice == "2":
```

```
_Module06.A06.Assignment06.IO
@staticmethod
def input_menu_choice() -> str
```

A function that takes user input for their menu choice

ChangeLog: (Who, When, What) Chantal Van den bussche, 3/5/2025, Created Function

Figure 1: docstring informing what the function input_menu_choice does

Separation of Classes (SoC)

SoC is a pattern of organizing classes, functions, and variables by their overarching purpose within the code. By following the SoC, you promote modularity, reusability, and maintainability. Presentation concern (anything to do with user interface elements), Logic concern (anything to do with the core functionality of my script), and Data Storage concern (anything to do with storing my data) are common types of software development concerns. I used these concerns to organize my code into three separate layers: Presentation layer (anything to do with Presentation concern), Processing layer (anything to do with Logic concern), and Data Storage layer (anything to do with Data Storage concern). Example 11 showcases my Data Storage layer.

Example 11:

```
# -- Data Storage -- #
# 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.
-----------------------------------------
'''
# FILE_NAME: str = "Enrollments.csv"
FILE_NAME: str = "Enrollments.json"

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

Summary

This week, I learned how to make my code more modular for a more organized and easier to maintain script. First, I learned about functions, a way to group code by their specific task. Then, I learned about classes, a way to group functions and variables by their overarching purpose. And I learned the Separation of Concerns pattern which organizes classes, functions, and variables by the specific software development concern they address.