Chad Conklin

11/13/2023

IT FDN 110 B Au 23: Foundations Of Programming: Python

Assignement 06

<https://github.com/cconk/IntroToProg-Python-Mod06>

Python Programming

# Introduction

# In the realm of Python programming, understanding foundational concepts such as functions, parameters, arguments, and classes is crucial for writing effective and maintainable code. These elements serve as the building blocks of Python programming, allowing developers to create structured, efficient, and reusable code. The way these components interact and are utilized significantly impacts the organization, readability, and scalability of software projects. Below, we explore these key concepts in detail, shedding light on their definitions, differences, and the roles they play in Python programming.

# Functions in Python are fundamental constructs that enable code reuse and logical segmentation. They allow programmers to encapsulate specific tasks or logic within defined blocks, making the code more readable and maintainable. Parameters and arguments, closely associated with functions, determine how data is passed to and used within these functions. Understanding the distinction between these two is important for proper function implementation. Furthermore, Python's support for both global and local variables provides flexibility in data management but requires careful consideration to avoid potential conflicts and maintain code clarity. Classes, another pivotal aspect of Python, offer a way to bundle data and functionality, promoting object-oriented programming principles. They provide a means to organize functions methodically and foster a more structured approach to problem-solving. The concept of "Separation of Concerns," integral to software design, is naturally supported by the use of functions, as it encourages dividing a program into distinct sections, each handling a specific part of the functionality. This approach not only simplifies the development process but also enhances code quality and sustainability. Let's delve deeper into each of these concepts to understand their roles and applications in Python programming by responding to each of the following questions as seen below.

# Topic 1

1. What are functions?
   1. Functions in Python are blocks of reusable code that perform a specific task. They help in breaking down complex problems into smaller, manageable parts. A function is defined using the def keyword, followed by the function name and parentheses.
2. What are parameters?
   1. Parameters are variables declared in the function definition. They act as placeholders for the data that can be passed into the function. Parameters specify what kind of arguments a function expects to receive when it is called.
3. What are arguments?
   1. Arguments are the actual values or data passed to a function when it is called. These values are assigned to the function's parameters and used within the function.
4. What is the difference between parameters and arguments?
   1. The difference lies in their role and timing of use. Parameters are the variables defined in the function's declaration to accept certain values, while arguments are the real values passed to the function when it is invoked. In simple terms, parameters receive arguments.
5. What are return values?
   1. Return values are the results that a function sends back to the part of the program that called it. A function can return a value using the return statement. If no return statement is used, the function returns None by default.
6. What is the difference between a global and a local variable?
   1. A global variable is declared outside of all functions and is accessible throughout the entire script, including inside functions. A local variable, on the other hand, is declared within a function and is only accessible within that function. It cannot be accessed from outside its function.
7. How do you use functions to organize your code?
   1. Functions allow you to encapsulate code into discrete sections, each performing a specific task. This leads to better organization by enabling code reuse, improving readability, and making maintenance easier. It also helps in breaking down complex problems into simpler tasks.
8. What is the difference between a function and a class?
   1. A function is a block of code that performs a specific task, while a class is a blueprint for creating objects (a particular data structure that can contain both data and functions). Classes support inheritance and polymorphism, enabling more complex behaviors. Functions within a class are known as methods and they operate on the data within the object (instance of the class).
9. How do you use classes to organize your functions?
   1. Classes allow you to group related functions (methods) and data (attributes) together. This encapsulation makes it easier to model real-world problems and relationships. Functions within classes (methods) can interact with the class's internal state, providing a cohesive way to represent and manipulate data.
10. How do functions help you program using the “Separation of Concerns" principle?
    1. By using functions, you can separate different aspects or concerns of your program. Each function deals with a specific task or concern, making the code more modular, easier to understand, maintain, and test. This principle is fundamental in software engineering for managing complexity and improving code quality.

# Topic 2

For the final task included in the assignment document concerning creating a program using constants, variables, and print statements, I developed and successfully ran the following program both in the command prompt and IDLE. Additionally, I successfully used Visual Studio Code to develop the solution. The following code, including the comments therein, encompasses the solution I was able to provide. These are the steps I completed and focus points based on the starter code:

1. Script Header: Updated the script header with my name and current date.
2. Constants: Several of the constants were already defined like MENU and FILE\_NAME as constants, in the starter code.
3. Variables: Similar to the constants several of the variables were defined int the starter code as well like menu\_choice, students, and other necessary variables. These are used within functions and classes.
4. Classes and Functions: Create two classes, FileProcessor and IO, each with their own methods. Here's a brief outline:
   * FileProcessor: This class handles file operations like reading and writing data.
   * IO: This class handles input/output operations like displaying menus and capturing user inputs.
5. Implementing Functions:
   * output\_error\_messages(message: str, error: Exception = None): This function will print error messages.
   * output\_menu(menu: str): This function will display the menu.
   * input\_menu\_choice(): This function captures and returns the user's menu choice.
   * output\_student\_courses(student\_data: list): This function prints the current student data.
   * input\_student\_data(student\_data: list): This function captures new student data from the user.
   * read\_data\_from\_file(file\_name: str, student\_data: list): This function reads data from a file.
   * write\_data\_to\_file(file\_name: str, student\_data: list): This function writes data to a file.
6. Using the @staticmethod decorator
   * The @staticmethod decorator in Python is used to define a method within a class as a static method, which means that it can be called on the class itself without needing an instance of the class. This has several implications and uses:
     1. Instance Independence: A static method does not have access to the instance (self) of the class or any of its instance attributes. It behaves like a regular function but is enclosed within the class's namespace.
     2. Class State Independence: Static methods also do not have access to the class attributes (cls) unless explicitly passed. They are independent of the state of the class.
     3. Usage:
        1. Static methods are used when some functionality is related to the class but does not need to access or modify the class's state.
        2. They are often used to create utility functions within a class.
     4. Syntax and Calling:
        1. Defined using the @staticmethod decorator above the method definition.
        2. Can be called using either the class name or an instance of the class, e.g., ClassName.static\_method() or class\_instance.static\_method().
     5. Comparison with Class Methods:
        1. Unlike static methods, class methods take a cls parameter that points to the class and not the instance when the method is called. They are defined with the @classmethod decorator.
        2. Class methods can modify the class state that applies across all instances of the class, while static methods cannot.

# ------------------------------------------------------------------------------------------ #

# Title: Assignment06\_Starter

# Desc: This assignment demonstrates using functions

# with structured error handling

# Change Log: (Who, When, What)

#   RRoot,1/1/2030,Created Script

#   Chad Conklin,11/14/2023, Modified script to use functions and structured error handling

# ------------------------------------------------------------------------------------------ #

import json

import os

from typing import IO

# 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.json"

# Data Variables

students: list = []  # a table of student data

#Classes

class FileProcessor:

    """Handles file processing tasks"""

    @staticmethod

    def read\_data\_from\_file(file\_name: str, student\_data: list):

        """ Reads data from a file into student\_data list """

        if not os.path.exists(file\_name):

                # File does not exist, so we start with an empty list

                return

        try:

                with open(file\_name, "r") as file:

                    # Ensure the file is not empty

                    file\_content = file.read()

                    if file\_content:

                        student\_data.extend(json.loads(file\_content))

                    else:

                        # File is empty, so we start with an empty list

                        return

        except json.JSONDecodeError as e:

                IO.output\_error\_messages("Error decoding JSON from file", e)

        except Exception as e:

                IO.output\_error\_messages("Error reading file", e)

    @staticmethod

    def write\_data\_to\_file(file\_name: str, student\_data: list):

        try:

            with open(file\_name, "w") as file:

                json.dump(student\_data, file)

        except Exception as e:

            IO.output\_error\_messages("Error writing to file", e)

class IO:

    """ Handles Input/Output tasks """

    @staticmethod

    def output\_error\_messages(message: str, error: Exception = None):

        print(message)

        if error:

            print("-- Technical Error Message --")

            print(error)

    @staticmethod

    def output\_menu(menu: str):

        print(menu)

    @staticmethod

    def input\_menu\_choice() -> str:

        return input("What would you like to do: ")

    @staticmethod

    def output\_student\_courses(student\_data: list):

        print("-" \* 50)

        for student in student\_data:

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

        print("-" \* 50)

    @staticmethod

    def input\_student\_data() -> dict:

        try:

            first\_name = input("Enter the student's first name: ")

            if not first\_name.isalpha():

                raise ValueError("The first name should not contain numbers.")

            last\_name = input("Enter the student's last name: ")

            if not last\_name.isalpha():

                raise ValueError("The last name should not contain numbers.")

            course\_name = input("Please enter the name of the course: ")

            return {"FirstName": first\_name, "LastName": last\_name, "CourseName": course\_name}

        except ValueError as e:

            IO.output\_error\_messages(str(e))

            return {}

# Main processing

FileProcessor.read\_data\_from\_file(FILE\_NAME, students)

while True:

    IO.output\_menu(MENU)

    menu\_choice = IO.input\_menu\_choice()

    if menu\_choice == "1":

        student\_info = IO.input\_student\_data()

        if student\_info:

            students.append(student\_info)

            print(f"You have registered {student\_info['FirstName']} {student\_info['LastName']} for {student\_info['CourseName']}.")

    elif menu\_choice == "2":

        IO.output\_student\_courses(students)

    elif menu\_choice == "3":

        FileProcessor.write\_data\_to\_file(FILE\_NAME, students)

        print("Data saved to file.")

    elif menu\_choice == "4":

        break

    else:

        print("Please choose a valid option (1-4).")

print("Program Ended")

# Summary

In conclusion, I continued to learn a lot about Python throughout this module and I am looking forward to the rest of class.