Abarna Sharankumar
Feb 26,2024
Foundations of Programming: Python
Assignment 06

CLASSES AND OBJECTS

Introduction

This document will describe the concept of classes, objects, and its purpose. Each class defines a set of attributes (data) and methods (functions) that encapsulate specific behaviors. Data Classes hold the raw data—attributes that describe an object. Presentation class handle how data is presented or displayed. Processing class perform actions or computations based on the data. This assignment also includes the concept of constructors, class inheritance and overridden methods.

Classes

Classes provide a natural way to organize code by grouping functions and data needed by those functions, making code more structured and readable, especially in larger projects. Using the @staticmethod decorator Python will allow us to use the class functions directly, instead of creating an object first. To define a class, use the class keyword followed by the class name. Inside the class, define attributes (variables) and methods (functions).

Objects

objects are the fundamental building blocks of data.
Every piece of data in a Python program is represented by an object or by relationships between objects. Objects have three essential characteristics:
Identity: Each object has a unique identity.
Type: Objects belong to specific types.
Value: Objects store actual data.
An object is an instance of a class. A class acts as a blueprint, defining attributes and methods. When we create an object, we are essentially copying the class with actual values. We can access variables and methods of an object using dot notation. **Class variables** are shared among all instances of a class. **Instance variables** are specific to each object.

Constructors

Constructors initialize object attributes when you create an instance. The constructor method is called __init__().Attributes store data specific to each instance. The primary role of a constructor is to initialize the object's attributes. It assigns initial values to the object's member variables, ensuring that the object starts with a defined state. Constructors never have a return type. They set the attribute variables either globally, or more often, by using special functions designed to "set" each attributes value. These special functions are called Properties.

The Self keyword

Self keyword is used to refer to data or functions found in an object instance, and not directly in the class. We cannot pass an argument the "self" parameter, it is automatically filed with the object's referenced location in memory. When a method is used directly from the class, we leave out the "self" parameter and mark the method with the **@staticmethod** decorator as we have seen in earlier modules.

Properties

Properties are functions designed to manage attribute data. Typically, for each attribute, we create two properties—one for getting data and one for setting data. These functions are commonly known as "Getters" and "Setters". Getter property functions enables access to data while optionally applying formatting. In Python, the @property decorator indicates a function as a getter. Setter property functions allows to add validation and error handling. You create a setter like any other function, but it must include the @name_of_property.setter decorator. The Setter decorator and function name must match the name of the Getter for them to be a getter and setter pair.

Inheritance

Inheritance is a core concept in OOP where a new class can inherit data and behaviors from an existing class . "Inherited code" refers to code that is inherited or derived from another source, such as a parent class. Inherited code serves as the foundation for building new code or extending existing functionality.

```
class FileProcessor:

    @staticmethod
    def read_data_from_file(file_name):
        """
        Reads data from file
        :param file_name:
        :return
        """
        global students
        try:
            with open(file_name, "r") as file:
                data = json.load(file)
                students = [Student(item['first_name'], item['last_name'], item['course_name']) for item in data]
                for student in students:
                    print(f"{student.first_name} {student.last_name} for {student.course_name}")
        except FileNotFoundError as error:
            IO.output_error_messages("File Not Found", error)
            print("Creating a new file")
            with open(file_name, "w") as file:
                json.dump(students, file)
        except JSONDecodeError as error:
            IO.output_error_messages("Invalid JSON file", error)
            print("Creating a new file")
            with open(file_name, "w") as file:
                json.dump(students, file)
        except Exception as error:
            print(error)
            IO.output_error_messages("unhandled exception", error)
        return students

    @staticmethod
    def write_data_to_file(file_name, students):
        """
        writes data to file
        :param file_name:
        :param students:
        :return:
        """
        try:
            with open(file_name, "w") as file:
                students_dict = [student.to_dict() for student in students]
                json.dump(students_dict, file)
                for student in students:
                    print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}")
        except ValueError as error:
            IO.output_error_messages("Unhandled exception ", error)
        except Exception as error:
            IO.output_error_messages("There was an error writing to the file", error)


class IO:
    global students

    @staticmethod
    def output_error_messages(message, error: Exception = None):
        """
        displays the error messages
        :param message:
        :param error:
        """
        print(message)
        if error is not None:
            print("---Technical Information---")
            print(error, error.__doc__)
```

```
@staticmethod
def input_menu_choice(menu):
    print(menu)
    choice = input("What would you like to do: ")
    return choice

@staticmethod
def input_student_data(students):
    """
    Function to get input from the user
    :param Students
    :return: updated student data
    """
    try:
        student_first_name = input("Enter student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("First name must contain only alphabets")
        student_last_name = input("Enter student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("Last name must contain only alphabets")
        course_name = input("Enter course name: ")
        student = Student(student_first_name, student_last_name, course_name)
        students.append(student)
    except ValueError as e:
        IO.output_error_messages("User entered invalid name")
    except Exception as e:
        IO.output_error_messages("Unhandled exception", e)
    finally:
        return students

@staticmethod
def output_student_data(students):
    try:
        for student in students:
            print(f"{student.first_name} {student.last_name} registered for {student.course_name}")
    except Exception as e:
        IO.output_error_messages("unhandled exception", e)
```

```python
class Person:
    _first_name: str = ""
    _last_name: str = ""

    def __init__(self, first_name, last_name):
        """
        Constructor to initialize a Person object
        :param first_name:
        :param last_name:
        """
        self._first_name = first_name
        self._last_name = last_name

    @property
    def first_name(self):
        """
        Returns the first name of the person
        """
        return self._first_name.title()

    @first_name.setter
    def first_name(self, first_name):
        """
        Sets the first name of the person
        :param first_name
        """
        if first_name.isalpha():
            self._first_name = first_name.title()
        else:
            IO.output_error_messages("First name must contain only alphabets")

    @property
    def last_name(self):
        """
        Returns the last name of the person
        """
        return self._last_name.title()
    @last_name.setter
    def last_name(self, last_name):
        """
        Sets the last name of the person
        :param last_name
        """
        if last_name.isalpha():
            self._last_name = last_name.title()
        else:
            IO.output_error_messages("Last name must contain only alphabets")

    def __str__(self):
        """
        Returns the string representation of the person object
        :return:
        """
        return f"{self._first_name} {self._last_name}"


class Student(Person):
    """
    A student class that inherits properties from Person class
    """
    course_name: str = ""

    def __init__(self, first_name, last_name, course_name):
        """
        constructor to initialize the student object
        :param first_name
        :param last_name
        :param course_name
        """
        super().__init__(first_name, last_name)
        self.course_name = course_name

    def to_dict(self):
        return {
            'first_name': self.first_name,
            'last_name': self.last_name,
            'course_name': self.course_name
        }
```

Summary

Thus, Python program is created for course registration four main functionalities. It allows users to register a student for a course, view the current data, save the data to a file, and exit the program. On startup, it reads data from an “Enrollments.json” file into a list of dictionaries. When you add or change information in the program the program saves these changes to a file. This way, the information is not lost when you close the program, and we can see it again the next time we open the program. The program ends when the user chooses to exit.