Object Oriented Programming

Introduction to OOP - Concepts

- **What is OOP?**
- **Why OOP?**
- **What are Classes?**
- **What are Objects ?**
- Advantages of OOPs

1 - What is OOP ?

- Simple Definition : A programming paradigm based on the concept of "objects."
- Real-world elements or entitites have Properties and they exhibit Behaviours. Few Examples are:

ENTITY	PROPERTIES	BEHAVIOURs
BIRD	Animal with wings, may be a Beak, Or Legs.	Birds can fly very high
STUDENTS	Human beings looking to Learn something	- Focus on their Topic, May carry or Read Books
PERSON	Has Identity like Name, Age, Gender	Exhibits Behaviour like talking, running, walking

- Object-oriented programming is a Programming Paradigm that allows us to structuring computer programs so that **properties and behavi**ors are bundled into individual **objects**.
  - **What is a Programming Paradigm?**
    - A high-level way to **conceptualize** and **structure** the implementation of a Program
    - Otherwise Programs may become **unmanagible**
    - Example of Other Paradigms - **Procedural Programming, Constraint Programming** etc.
- In OOPs Paradigm:
  - A Computer Program is mainly organized as objects
  - **Objects Contain both data structure and associated behavior**
  - Uses data structures consisting of **data fields or attributes and methods or behaviour together** with their interactions (objects) to design programs.
- **Objects bundle data (attributes) and behavior (methods).**

2 - Why we need OOPs?

  - Data Structures are basic building blocks of a Programming Language - allowing us to store and work on data in a particualr manner.

        - There are base or Primitive data structures that are parto f the Language. Example numbers, strings, and lists—are designed to represent information
        - However if we can a complex structure to hold Information, then we can use primitive structures to create our own data structure.
            - example : Employees in an organization. We need to track their Employee Id, Name, role etc...as a unit of data.
            - NOTE: If we use lists or dictionaris then it will be difficult to manage for large programs.
        - Object Oriented Programming Paradigm provides one way for us to create these so called Complex Structures
    - Business Perspective
        - Models real-world entities effectively.
    - Technical Perspective
        - Promotes key software engineering principles of
            - Code Reusability
            - Code Modularity
            - Scalability

3 - Classes and Objects

- **Definition of a Class:**
    - A class defines the **properties (attributes) and methods (actions)** that objects of that class will have.
    - A class is a **template** or **blueprint for creating objects.**
    - Defines attributes (data) and methods (functions).
    - Classes allow you to create user-defined data structures.
- **Definition of an Object:**
    - An instance of a class.
    - Contains specific data and can perform actions defined by the class.

4 - Objects

- Objects are a concrete or specific instances of a Class.
- Object will contains specific data and can perform actions defined by the class.
- Example
    - Imagine a Student() is a Class.
    - One particular Student with following details is an Object
        - Name : John
        - Age : 25
        - Hobbies: Sports, Music

OOPs Using Python : Code Examples

- Let us say we want to create an Application for the college.
- First Class to start with will be Students or Faculty or Subjects etc.

Define a Class

Python we define a class by using the `class` keyword followed by a name and a colon.

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Creating an object
student1 = Student("Alice", 20)
print(student1.name, student1.age)
```

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

- Make sure that you indent the `.__init__()` method (four spaces) and the body of the method by eight spaces.
  - **`self.name = name`** creates an attribute called `name` and assigns the value of the `name` parameter to it.
  - **`self.age = age`** creates an attribute called `age` and assigns the value of the `age` parameter to it.

Special Function : __init__()

- Each class needs to have a special function - which is Initialiser or Starting point.
- You use `.__init__()` to declare which attributes each instance of the class should have:

```
class Student:
    def __init__(self, name, age):
        self.name =  name
        self.age = age
```

Creating an object

- Creating a new object from a class is called **instantiating** a class

- We can create a new object by typing the name of the class, followed by opening and closing parentheses:
- ```
  john = Student()
  ```

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Creating an object
student1 = Student("Alice", 20)
print(student1.name, student1.age)
```

- You can pass the Parameters to the Object when instatiating
- ```
  john = Student("John", 25)
  ```
- After you create the `Dog` instances, you can access their instance attributes using **dot notation**:

```
class Student:
    def __init__(self, name, age, sports):
        self.name = name
        self.age = age
        self.sport = sport

 # Instance method
    def description(self):
        return f"{self.name} is {self.age} years old"

    # Another instance method
    def plays(self, sound):
        return f"{self.name} plays {self.sport}"
```

Instance methods

- **For the behaviour of a Class, we write functions.**
- **Instance methods** are functions that you define inside a class
- They can only call on an instance of that class.
  - `.__init__()`, an instance method always takes `self` as its first parameter.

Advantages of Object Oriented Programming

1. Modularity for easier troubleshooting.

2. Reusability through inheritance.
3. Flexibility with polymorphism.
4. Secure code with encapsulation.