→ "Object Oriented Programming in Python" :→

Object oriented programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core OOP principles (classes, objects, inheritance, encapsulation, polymorphism and abstraction) programmers can leverage the full potential of Python OOP capabilities to design elegant and efficient solutions to complex problems.

OOPs is a way of organizing code that uses objects & classes to represent real-world entities and their behaviours. In OOPs object has attributes that has specific data and can perform certain actions using methods.

"OOPs concepts in Python" :→

1) Classes in Python.
2) Objects in Python
3) Polymorphism in Python.
4) Encapsulation in Python.
5) Inheritance in Python.
6) Data Abstraction in Python.

1) Python Class :→ A class is a collection of objects. Classes are blueprints for creating objects. A class defines a set of attributes and methods that the created objects (instances) can have.

Note! :→ ↳ Classes are created by keyword class.

↳ Attributes are the variables that belong to a class.

↳ Attributes are always public and can be accessed using the dot (.) operator.

Example :→ MyClass.Myattribute.

2) Python objects :→

An object is an instance of a class. It represents a specific implementation of the class & holds its own data.

An object consists of :

a) State :→ It is represented by the attributes and reflects the properties of an object.

b) Behaviour :→ It is represented by the methods of an object and reflects the response of an object to other objects.

c) Identity :→ It gives a unique name to an object and enables one object to interact with other object.

3) Python Inheritance :→

Inheritance allows a class (child class) to acquire properties and methods of another class (parent class). It supports hierarchical classification & promotes code reuse.

↳ Types :→

1) Single-Inheritance :→ A child class inherits from a single parent class.

2) Multiple-Inheritance :→ A child class inherits from more than one parent class.

3) Multilevel-Inheritance :→ A child class inherits from a parent class, which in turn inherits from another class.

4) Hierarchical-Inheritance :→ Multiple child classes inherit from a single parent class.

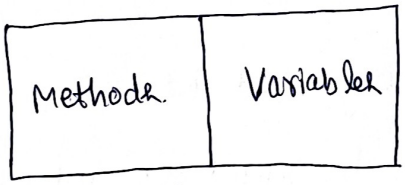
5> Hybrid - Inheritance :> A combination of two or more types of inheritance.

4> Python - Polymorphism :> Polymorphism allows methods to have the same name but behave differently based on the object's context. It can be achieved through method overriding or overloading.

5> Python - Encapsulation :>

Encapsulation is defined as bundling of data (attributes) & methods (functions) within a class, restricting access to some components to control interactions.

A class is an example of encapsulation. as it encapsulates all the data, that is member functions, variables, etc.



Class

"Types of Encapsulation" :>

1> Public-members :> Accessible from anywhere.

2> Protected-Members :> Accessible within the class & it's subclasses.

3> Private-Members :> Accessible only within the class.

6> Data Abstraction :> Abstraction hides the internal implementation details while exposing only the necessary functionality. It helps focus on "what to do" rather than "how to do it.".

[1.10]

Types of Abstraction :->

↳ "Partial-Abstraction" :- Abstract class contains both abstract and concrete methods.

↳ "Full-Abstraction" :- Abstract class contains only abstract methods (like interfaces).

Difference between Object Oriented Programming & Procedural Programming :-

-> OOP (Object Oriented Programming) :- It is a programming paradigm that organizes code into objects, which are instances of classes. OOP is designed to model real-world entities and promote code modularity, reusability and maintainability.

Advantages of OOP :->

1) Modularity :- Code is organized into classes & objects.
2) Reusability :- Code can be reused via inheritance.
3) Maintainability :- Encapsulation makes the system easier to maintain.
4) Scalability :- New features can be added without affecting existing code.

-> Procedural Programming :- It is a programming paradigm that follows a step by step approach using functions (procedures) to execute tasks. It focuses on breaking down a program into a set of instructions, executed in a linear and structured manner.

Key characteristics of procedural programming :→

1. Follows a top-down approach :→ The program executes from top to bottom.
2. Uses procedures (functions) :→ Code is divided into reusable functions.
3. Uses global & local variables :→ Data is often stored in variables that functions modify.
4. Focuses on logic & flow control :→ Uses loops (for, while), conditionals (if-else) & function calls.
5. Code is executed step by step :→ Each function is executed when called.

"Feature"	"Object oriented Programming (OOP)"	"Procedural Programming (PP)"
Definition	Organizes code using objects & classes.	Organizes code as a sequence of procedures or functions.
Approach	Focuses on modeling real-world entities with objects.	Focuses on step-by-step procedures to perform tasks.
Key Concepts	Uses classes, objects, inheritance polymorphism & encapsulation	Uses functions, loops & control structures.
Data Handling	Data and behavior are encapsulated within objects.	Data is often shared & modified globally.
Code Reusability	Promotes reusability through inheritance & polymorphism	Reusability is achieved through function calls & modularization.
Maintenance	Easier to maintain and extend due to encapsulation & abstraction.	Can become difficult to manage as the codebase grows.

[P.T.O]

Eg. of OOP :→ Java, C++, Python (oop features), C# (C Sharp)

Eg. of procedural programming :→ C, Pascal, older versions of BASIC

Let's see how to create a python class and object :

```
class Parrot:

    name = ""
    age = 0

parrot1 = Parrot()
parrot1.name = "Blu"
parrot1.age = 10

parrot2 = Parrot()
parrot2.name = "Woo"
parrot2.age = 15

print(f"{parrot1.name} is {parrot1.age} years old")
print(f"{parrot2.name} is {parrot2.age} years old")
```

Output :

```
Blu is 10 years old
Woo is 15 years old
```

In the above example, we created a class with the name Parrot with two attributes : name and age.

Then, we create instances of the Parrot class. Here Parrot1 and Parrot2 are references to our new objects.

We then accessed & assigned different values to the instance attributes using the object name and the "." notation.

Creating a student class to store name & age of students:

```
Class Student:
    name = "Raghav"
    age = 20

student1 = Student()
student2 = Student()
print(student1.name, student1.age)
print(student2.name, student2.age)
```

These are hard coded values. Each object of this class will take same values.

O/p: Raghav, 20
Raghav, 20

Now let's create a constructor to initialize the values whenever an object of this class is created.

```
Class Student:
    def __init__(self, full_name, class_grade):
        self.name = full_name
        self.age = class_grade

student1 = Student("Raghav", 10)
student2 = Student("Rahul", 11)
print(student1.name, student1.age)
print(student2.name, student2.age)
```

O/p: Raghav, 10
Rahul, 11

Let's now create a class Student to store name & age of students.
Note:- Objects created must be stored in an array & using array index, we must be able to access the name & age of an individual student.

P.T.O

```
Class Student:
    def __init__(self, full_name, stud_age):
        self.name = full_name
        self.age = stud_age

Testarray = [ ]
student1 = Student("Raghav", 10)
student2 = Student("Rahul", 11)
Testarray.append(student1)
Testarray.append(student2)
print(student1.age, student1.name)
print(student2.age, student2.name)
print(Testarray[1].name).
```

→ Class to store records of n students.

```
Class Student:
    def __init__(self, full_name, student_age):
        self.name = full_name
        self.age = student_age

record = int(input("How many student records you want to create?\n"))
print("Records to be created are", record)
students = [ ]
for i in range(1, record+1):
    name = input(f"Enter name of student {i}.\n")
    age = int(input(f"Enter age of student {i}.\n"))
    students.append(Student(name, age))

for index, student in enumerate(students):
    print(f"Name of student {index+1} is {student.name}\n")
    print(f"Age of student {index+1} is {student.age}\n")
```

→ Class to store records of students. Use function to get & print the details.

```
Class Student:
    name
    age
    def get_details(self):
        self.name = input("Enter student name.\n")
        self.age = int(input("Enter student age.\n"))

    def print_details(self):
        print(self.name)
        print(self.age)

student1 = Student()
student1.get_details()
student2 = Student()
student2.get_details()
student1.print_details()
student2.print_details()
print(student1.name, student1.age)
print(student2.name, student2.age)
```

→ Class to create a simple arithmetic program.

```
Class Calculator:
    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2

    def add(self):
        return self.num1 + self.num2
```

P.T.O

```
    def subtract(self):
        return self.num1 - self.num2

    def multiply(self):
        return self.num1 * self.num2

    def divide(self):
        if self.num2 == 0:
            return "Error! Division by zero."
        return self.num1 / self.num2

calc = Calculator(10, 5)
print("Addition:", calc.add())
print("Subtraction:", calc.subtract())
print("Multiplication:", calc.multiply())
print("Division:", calc.divide())
```

→ "OOPs: Object Oriented Programming"

It is a way of organizing code by creating "blueprints" called classes. to represent real-world things like a student, car or house. These blueprints help us to create objects (individual examples of those things) and define their behavior.

Class: A class is a blueprint or template for creating objects. It defines the properties (attributes) and actions/behaviors (methods) that objects of this type will have.

Object: An object is a specific instance of a class. It has actual data based on the blueprint defined by the class.

Example: "Constructing a building".

Class: Blueprint for a floor.

Object: Actual house built from the blueprint. Each house (object) can have different features, like paint color or size, but follows the same blueprint.

Why OOPs?

1> Models real-world problems:

Mimics real-world entities for easier understanding.

2> Code Reusability:

Encourages reusable, modular and organized code.

[P.T.O]

3> Easier Maintenance:

OOP organizes code into small, manageable parts (classes & objects). Changes in one part don't impact others, making it easier to maintain.

4> Encapsulation:

Encapsulation protects data integrity & privacy by bundling data & methods within objects.

5> Flexibility & Scalability:

OOP makes it easier to add new features without affecting existing code.

OOPs Programs Examples :->

-> Python program to create a class representing a circle. Include methods to calculate its area and perimeter.

```
import math

class Circle:

    def __init__(self, radius):
        self.radius = radius

    def calculate_circle_area(self):
        return math.pi * self.radius**2

    def calculate_circle_perimeter(self):
        return 2 * math.pi * self.radius

radius = float(input("Enter the radius of the circle: "))
circle = Circle(radius)
area = circle.calculate_circle_area()
perimeter = circle.calculate_circle_perimeter()

print("Area of the circle:", area)
print("Perimeter of the circle:", perimeter)
```

-> Python program to store records and marks of n students using classes and objects.

```
Class student:
```

P.T.O

```
def __init__(self, name, roll_no, marks):
    self.name = name
    self.roll_no = roll_no
    self.marks = marks

def total_marks(self):
    return sum(self.marks)

def percentage(self):
    return self.total_marks() / len(self.marks)

def display_info(self):
    print(f"\n Student Name : {self.name}")
    print(f"\n Roll Number : {self.roll_no}")
    print(f"\n Total Marks : {self.total_marks()}")
    print(f"\n Percentage : {self.percentage():.2f} %")

n = int(input("Enter the number of students : "))
students = []

for i in range(n):
    name = input(f"\n Enter name of student {i+1} : ")
    roll_no = input(f" Enter roll number of student {i+1} : ")
    marks = list(map(int, input("Enter marks in 5 subjects (separated
            by space) : ").split()))

    student = Student(name, roll_no, marks)
    students.append(student)

print("\n Student Records : ")
for student in students:
    student.display_info()
```