OBJECT ORIENTED PROGRAMMING----->>>>> It 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's OOP capabilities to design elegant and efficient solutions to complex problems.

In Python object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of object-oriented Programming (OOPs) or oops concepts in Python is to bind the data and the functions that work together as a single unit so that no other part of the code can access this data.

OOPs Concepts in Python Class in Python Objects in Python Polymorphism in Python Encapsulation in Python Inheritance in Python Data Abstraction in Python

TYPES OF VARIABLES--->> 1.GLOBAL VARIABLES 2.LOCAL VARIABLES

PYTHON CLASSES-->>

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

class Dog: pass

PYHTON OBJECTS---->>>

In object oriented programming Python, The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string "Hello, world" is an object, a list is an object that can hold other objects, and so on. You've been using objects all along and may not even realize it.

1. Define a simple `Car` class with attributes for make, model, and year. Create an object of this class and print its attributes.

```
class car:
  def __init__(self,make,model,year):
    self.make=make
    self.model=model
    self.year=year
mob1=car("toyota","camry",1987)
```

```
mob2=car("honda","hoda city",2004)
print(mob1.make,mob1.model,mob1.year)
print(mob2.make,mob2.model,mob2.year)
```

```
toyota camry 1987
honda hoda city 2004
```

2. Create a Person class with attributes for name and age. Add a method to print a greeting message. Create an object and call the method.

```
class person:
  def __init__(self,name,age):
    self.name=name
    self.age=age
per1=person("mehak",19)
print(per1.name,per1.age)
```

```
mehak 19
```

3. Define a Rectangle class with attributes for length and width. Add a method to calculate the area of the rectangle. Create an object and print the area.

```
class rectangle:
  def __init__ (self,length,width):
    self.length=length
    self.width=width
rec1=rectangle(10,5)
print(rec1.length,rec1.length)
```

```
10 10
```

4. Create a Student class with attributes for name and grades (a list of numbers). Add a method to calculate the average grade. Create an object and print the average grade.

```
class student:
  def __init__(self,name,grades):
    self.name=name
    self.grades=grades
  def purchase(self):
    print(self.name,self.grades)

mob1=student("mehak",19)

mob1.purchase()
```

```
mehak 19
```

5. Define a Book class with attributes for title, author, and pages. Add a method to display the book's details. Create an object and call the method.

```
class book:
  def __init__(self,title,author,pages):
    self.title=title
    self.author=author
    self.pages=pages
  def purchase(self):
    print(self.title,self.author,self.pages)
mob1=book("mehak",19,22)
mob1.purchase()
```

```
mehak 19 22
```

6. Create a `Dog` class with attributes for name and breed. Add a method to make the dog bark (print a bark message). Create an object and call the method.

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

    def bark(self):
        print(f"Name of dog is {self.name} and breed is {self.breed}")

dog1 = Dog("Tommy", "pug")
dog1.bark()
```

```
Name of dog is Tommy and breed is pug
```

7. Define a `BankAccount` class with attributes for account number and balance. Add methods to deposit and withdraw money. Create an object and test the methods.

```
class Bank_account:
  def __init__(self,account_number,balance):
     self.account_number=account_number
     self.balance=balance
  def account(self):
    print(f"Account detailes are -> {self.account_number} and
{self.balance}")

mob1=Bank_account(12345,10000)
mob1.account()
```

```
Account detailes are -> 12345 and 10000
```

8. Create a `Circle` class with attributes for radius. Add a method to calculate the circumference. Create an object and print the circumference.

```
import math as m
class circle:
```

```
    def __init__ (self,radius):
        self.radius=radius
    def circumference(self):
        print(2*m.pi*self.radius)
mob1=circle(10)
mob1.circumference()
```

```
62.83185307179586
```

9. Define a Laptop class with attributes for brand and price. Add a method to apply a discount to the price. Create an object and test the method.

```
class laptop:
  def __init__ (self,brand,price):
    self.brand=brand
    self.price=price
  def price_after(self):
      discount=0.05
      self.price=+self.price-self.price*discount
      print(self.brand,self.price)
mob1=laptop("hp",10000)
mob1.price_after()
```

```
hp 9500.0
```

10. Create a Employee class with attributes for name and salary. Add a method to give a raise (increase the salary). Create an object and test the method.

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def give_raise(self, amount):
        self.salary += amount
        print(self.salary)
mob1=Employee("Rahul",10000)
mob1.give_raise(10)
```

```
10010
```

11. Define a Point class with attributes for x and y coordinates. Add a method to calculate the distance from another point. Create two objects and print the distance between them.

```
class point:
  def __init__(self,x,y):
    self.x=x
    self.y=y
  def distance(self,other_point):
```

```
    x_diff=self.x-other_point.x
    y_diff=self.y-other_point.y
    distance=(x_diff**2+y_diff**2)**0.5
    print(distance)

mob1=point(0,0)
mob2=point(9,4)
mob1.distance(mob2)
```

```
9.848857801796104
```

2. Create a Movie class with attributes for title, director, and release year. Add a method to display the movie's information. Create an object and call the method.

```
class movie:
  def __init__(self,title,director,release_year):
    self.title=title
    self.director=director
    self.release_year=release_year
  def  display(self):
    print(f"The movie was creted by {self.director} of title
{self.title} in year {self.release_year} ")
mob1=movie("Tarzan","abcd",2004)
mob1.display()
```

```
The movie was creted by abcd of title Tarzan in year 2004
```

13. Define a Product class with attributes for name and price. Add a method to calculate the price after tax (assume a fixed tax rate). Create an object and print the price after tax.

```
class product:
  def __init__(self,name,price):
    self.name=name
    self.price=price
  def purchase(self,tax_rate):
    self.price=self.price+tax_rate
    print(self.name,self.price)
mob1=product("mehak",1000)
mob1.purchase(10)
```

```
mehak 1010
```

14. Create a Player class with attributes for name and score. Add a method to update the score. Create an object and test the method.

```
class player:
  def __init__(self,name,score):
    self.name=name
    self.score=score
```

```
  def update(self,new_score):
    self.score+=new_score
    print(f"{self.name} has score {self.score}")
mob1=player("Rahul",50)
mob1.update(50)
```

```
Rahul has score 100
```

15. Define a House class with attributes for address and number of rooms. Add a method to display the house's details. Create an object and call the method.

```
class house:
  def __init__(self,address,number_of_rooms):
    self.address=address
    self.number_of_rooms=number_of_rooms
  def display(self):
    print(f"The house is located at {self.address} and has
{self.number_of_rooms} rooms")
mob1=house("123 Main St",5)
mob1.display()
```

```
The house is located at 123 Main St and has 5 rooms
```

16. Create a Shape class with attributes for color and filled (a boolean). Add a method to display the shape's properties. Create an object and call the method.

```
class shape:
  def __init__ (self,color,filled):
    self.color=color
    self.filled=filled
  def display(self):
    print(f"The shape is {self.color} and {self.filled}")
mob1=shape("red",True)
mob1.display()
```

```
The shape is red and True
```

17. Define a Vehicle class with attributes for type and speed. Add a method to accelerate (increase the speed). Create an object and test the method.

```
class vehicle:
  def __init__(self,type,speed):
    self.type=type
    self.speed=speed
  def accelerate(self,new_speed):
    self.speed+=new_speed
    print(self.speed)
mob1=vehicle("car",100)
mob1.accelerate(10)
```

```
110
```

Practice Questions: 1.Given a class for a BasicPlan, write the classes for StandardPlan and PremiumPlan which have class attributes of the following:

BasicPlan StandardPlan Premium Plan ✓ ✓ ✓ can_stream ✓ ✓ ✓ can_download ✓ ✓ ✓ has_SD ✓ ✓ has_HD Examples BasicPlan.has_SD → True

PremiumPlan.has_SD → True

BasicPlan.has_UHD → False

BasicPlan.price → "$8.99"

PremiumPlan.num_of_devices → 4 ✓ has_UHD 1 2 4 num_of_devices $8.99 $12.99 $15.99 price

```
# prompt: Practice Questions: 1.Given a class for a BasicPlan, write
the classes for StandardPlan and PremiumPlan which have class
attributes of the following:
# BasicPlan     StandardPlan    Premium Plan
# ✓               ✓                             ✓    can_stream
# ✓                ✓                        ✓     can_download
# ✓                ✓                        ✓     has_SD
#                      ✓                      ✓

class BasicPlan:
    can_stream = True
    can_download = True
    has_SD = True
    price = "$8.99"

class StandardPlan(BasicPlan):
    has_HD = True
    num_of_devices = 2
    price = "$12.99"

class PremiumPlan(StandardPlan):
    has_UHD = True
    num_of_devices = 4
    price = "$15.99"

print(BasicPlan.has_SD)  # True
print(PremiumPlan.has_SD)  # True
print(PremiumPlan.has_UHD)  # False
print(BasicPlan.price)  # "$8.99"
print(PremiumPlan.num_of_devices)  # 4

True
True
True
```

```
$8.99
4
```

2. Create a method in the Person class which returns how another person's age compares. Given the objects p1, p2 and p3, which will be initialised with the attributes name and age, return a sentence in the following format:

{other_person} is {older than / younger than / the same age as} me.

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

  def compare_age(self, other):
    if self.age > other.age:
      return f"{other.name} is younger than me."
    elif self.age < other.age:
      return f"{other.name} is older than me."
    else:
      return f"{other.name} is the same age as me."

p1 = Person("Alice", 25)
p2 = Person("Bob", 30)
p3 = Person("Carol", 25)

print(p1.compare_age(p2))  # "Bob is older than me."
print(p1.compare_age(p3))  # "Carol is the same age as me."
```

3. Create methods for the Calculator class that can do the following:

Add two numbers. Subtract two numbers. Multiply two numbers. Divide two numbers.

```
# prompt: 3. Create methods for the Calculator class that can do the
following:
# Add two numbers.
# Subtract two numbers.
# Multiply two numbers.
# Divide two numbers.

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

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

  def subtract(self):
```

```
    return self.num1 - self.num2

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

  def divide(self):
    if self.num2 == 0:
      raise ZeroDivisionError("Cannot divide by zero.")
    return self.num1 / self.num2

# Create a calculator object
calc = Calculator(10, 5)

# Add two numbers
print(calc.add())  # Output: 15

# Subtract two numbers
print(calc.subtract())  # Output: 5

# Multiply two numbers
print(calc.multiply())  # Output: 50

# Divide two numbers
print(calc.divide())  # Output: 2.0
```