Define Class and Syntax of Class

- **Definition:**

1. A **class** is a **blueprint or template** for creating objects.
2. It defines **data (variables) and functions (methods)** that operate on that data.
3. A class helps in **code organization and reusability** in object-oriented programming.
4. **Syntax:**

```
class ClassName:
    def __init__(self):  # Constructor
        self.variable = "value"
    def method(self):  # Method
        print("Hello from Class")
```

Define Object and Syntax of Object

- **Definition:**

1. An **object** is a **real-world entity** created using a class.
2. Objects **store class data and can use class methods**.
3. Multiple objects of the same class can have **different values**.
4. **Syntax:**

```
obj = ClassName()  # Creating an object
obj.method()  # Calling a method
```

Marathi Tip (मराठी टीप)

- **"A class is a design, and an object is its real-world usage!"**
- **Example:** If "Car" is a class, then its objects can be **Honda, Tata, BMW** – different cars built using the same blueprint!

Q. Design a class student with data members : name, roll no., department, mobile no, address. Create suitable methods for reading and printing student information ? (Most Imp)

Ans: -

```
# Define the Student class
class Student:
    # Constructor to initialize the student data
    def __init__(self):
        self.name = ""
        self.roll_no = ""
        self.department = ""
        self.mobile_no = ""
        self.address = ""

    # Method to read student details
    def read_student_info(self):
        self.name = input("Enter Student Name: ")
        self.roll_no = input("Enter Roll No: ")
        self.department = input("Enter Department: ")
        self.mobile_no = input("Enter Mobile No: ")
        self.address = input("Enter Address: ")

    # Method to print student details
    def print_student_info(self):
        print("\n--- Student Information ---")
        print(f"Name       : {self.name}")
        print(f"Roll No    : {self.roll_no}")
        print(f"Department : {self.department}")
        print(f"Mobile No  : {self.mobile_no}")
        print(f"Address    : {self.address}")
```

```
# Create an object of the Student class
student1 = Student()

# Read student details
student1.read_student_info()

# Print student details
student1.print_student_info()
```

OUTPUT:

```
Enter Student Name: Rajesh Patil
Enter Roll No: 101
Enter Department: Computer Engineering
Enter Mobile No: 9876543210
Enter Address: Pune

--- Student Information ---
Name       : Rajesh Patil
Roll No    : 101
Department : Computer Engineering
Mobile No  : 9876543210
Address    : Pune
```

❖Method Overloading in Python (Most Imp)

1. Method Overloading allows multiple methods in the same class with the **same name** but **different numbers of parameters**.
2. In Python, method overloading is **not directly supported**, but we can achieve it using **default arguments** or *args.
3. The method behaves differently based on the number/type of arguments passed.
4. It improves **code readability and reusability** by handling multiple cases with the same method name.

Real-World Example

Example: Calculator App

- A **calculate_area()** method can be used for **circle, rectangle, or square** by overloading it.

Python Program for Method Overloading

```
class MathOperations:
    def add(self, a=0, b=0, c=0):  # Default values
        return a + b + c  # Returns sum of given numbers

# Create object
obj = MathOperations()

print(obj.add(10, 20))      # 2 arguments → Output: 30
print(obj.add(10, 20, 30))  # 3 arguments → Output: 60
print(obj.add(10))          # 1 argument → Output: 10
```

Output:

```
30
60
10
```

❖Method Overriding in Python(Asked once)

1. Method Overriding occurs when a **child class redefines a method** of its parent class using the **same name and parameters**.
2. It allows a subclass to provide a **specific implementation** of a method defined in the parent class.
3. The overridden method in the child class **replaces** the method from the parent class when called through an object of the child class.
4. It follows **runtime polymorphism**, meaning the method call is decided at runtime.

Real-World Example

Example: Banking System

- A **withdraw()** method in BankAccount can be overridden in SavingsAccount and CurrentAccount classes to implement different withdrawal rules.

Python Program for Method Overriding

```
# Parent class
class Animal:
    def sound(self):
        print("Animals make different sounds")

# Child class overriding the sound method
class Dog(Animal):
    def sound(self):
        print("Dog barks!")

# Child class overriding the sound method
class Cat(Animal):
    def sound(self):
        print("Cat meows!")

# Create objects
dog = Dog()
cat = Cat()

# Call overridden methods
```

```
dog.sound()  # Output: Dog barks!
cat.sound()  # Output: Cat meows!
```

Output:

```
Dog barks!
Cat meows!
```

Marathi Tip (मराठी टीप)

- **"Method Overloading म्हणजे एकाच नावाची अनेक कार्ये, तर Method Overriding म्हणजे पालक वर्गाची कार्ये नवीन रीतीने परिभाषित करणे!"**

उदाहरण:

- **Method Overloading** → एका गणना पद्धतीद्वारे क्षेत्रफळ मोजणे (वर्तुळ, चौकोन, त्रिकोण)
- **Method Overriding** → प्राणी वर्गातील sound() पद्धतीला वेगवेगळ्या प्राण्यांसाठी वेगवेगळे आवाज देणे!

🔹 Key Differences Summary

Feature	Method Overloading	Method Overriding
Definition	Same function name, different parameters	Same function name, same parameters, different behavior
Where it occurs?	In the **same class**	Between **parent and child classes**
Purpose	Multiple ways to call a function	Change the behavior of an inherited method
Implementation	Uses **default arguments** / `*args`	Uses **inheritance (Parent-Child class relationship)**
Example	`add(a, b)`, `add(a, b, c)`	`sound()` in `Animal` overridden in `Dog`

❖Data Hiding in Python (Ask 2 time for 2marks)

- Data hiding is a technique used to **restrict direct access** to data members of a class.
- It is an important concept in **Object-Oriented Programming (OOP)** to protect **private data**.
- It **prevents accidental modification** of important values by outside code.
- Data hiding is achieved using **private variables** (prefixing __ before variable names).

Advantages of Data Hiding

1. **Data Security** – Protects sensitive data from unauthorized access.
2. **Encapsulation** – Combines data and functions into a single unit (class).
3. **Avoids Data Corruption** – Prevents accidental modification of critical values.
4. **Reduces Complexity** – Limits external dependencies, making the system more robust.

Marathi Tip (मराठी टीप)

- **Data Hiding म्हणजे डेटाला लपवणे आणि संरक्षित करणे.**
- **उदाहरण:** बँकेच्या खात्याचा **balance** थेट बाहेरून बदलता येऊ नये!
- **कस काम करतं?** __variable_name वापरून डेटा लपवता येतो.

महत्वाचे: डेटा **सुरक्षित ठेवण्यासाठी** आणि **अनावश्यक बदल टाळण्यासाठी** Data Hiding वापरले जाते!

❖Data Abstraction in Python (Ask 1 time for 2marks)

- Data abstraction is a technique in **Object-Oriented Programming (OOP)** that **hides unnecessary details** from the user.
- It only shows **essential features** and keeps the internal implementation hidden.
- The main goal of abstraction is to **simplify complex systems** by exposing only relevant parts.
- In Python, **abstraction is achieved using abstract classes and methods** (from abc module).

Advantages of Data Abstraction

1. **Hides Complexity** – Users only interact with essential details.
2. **Better Code Maintainability** – Easier to update and manage large programs.
3. **Increases Security** – Internal implementation remains protected.
4. **Enhances Code Reusability** – Abstract classes can be used as blueprints for multiple implementations.

Marathi Tip (मराठी टीप)

- **Data Abstraction म्हणजे वापरकर्त्याला आवश्यक माहितीच दाखवणे आणि उर्वरित लपवणे.**
- **उदाहरण:** गाडी सुरू करायची आहे? फक्त start() कॉल करा – इंजिन कसं चालतं ते माहित नसले तरी चालेल!
- **कस काम करतं?** abstract class आणि abstract methods चा उपयोग करून abstraction साधता येतो.

महत्वाचे: Data Abstraction **प्रोग्रॅमिंग सोपी व सुरक्षित बनवते!**

❖ Inheritance in Python (Most IMP)

What is Inheritance?

- **Inheritance** is an **Object-Oriented Programming (OOP)** feature that allows a **child class to acquire properties and behaviors of a parent class**.
- It helps in **code reusability** and **reduces redundancy**.
- The **parent class (superclass)** provides attributes and methods that can be used by the **child class (subclass)**.
- The child class **can also override or extend** the functionality of the parent class.

◆ Types of Inheritance in Python

1 Single Inheritance

- A child class inherits from one parent class.
- **Real-life example**:
  Father (Parent Class) → Son (Child Class)
- **Example**:

```
class Animal: # Parent class
    def sound(self):
        print("Animals make sounds.")

class Dog(Animal): # Child class
    def bark(self):
        print("Dog barks.")

d = Dog() # Object creation
d.sound()  # ✅ Output: Animals make sounds.
d.bark()   # ✅ Output: Dog barks.
```

- **Explanation:** The Dog class inherits sound() method from Animal and adds its own method bark().

---

2 Multiple Inheritance

- A child class inherits from more than one parent class.
- **Real-life example:**
  A child inherits both father's and mother's traits.
- **Example:**

```
class Father:  # Parent class 1
    def work(self):
        print("Father is working.")

class Mother: # Parent class 2
    def care(self):
        print("Mother is caring.")

class Child(Father, Mother): # Child class inheriting both parents
    def play(self):
        print("Child is playing.")

c = Child()   # Object creation
c.work()  # ✅ Output: Father is working.
c.care()  # ✅ Output: Mother is caring.
c.play()  # ✅ Output: Child is playing.
```

- **Explanation:** Child inherits from both Father and Mother, meaning it gets both work() and care() methods.

---

3 Multilevel Inheritance

- A class inherits from another class, which itself inherits from another class (like a chain).
- **Real-life example:**
  Grandfather → Father → Son
- **Example:**

```
class Grandparent:  # Grandparent class
    def house(self):
        print("Grandparent owns a house.")

class Parent(Grandparent): # Parent class
    def car(self):
        print("Parent owns a car.")

class Child(Parent): # Child class
    def bike(self):
        print("Child owns a bike.")

c = Child()   # Object creation
c.house()  # ✅ Output: Grandparent owns a house.
c.car()    # ✅ Output: Parent owns a car.
c.bike()   # ✅ Output: Child owns a bike.
```

- **Explanation:**
  - Grandparent class provides house().
  - Parent inherits house() and adds car().
  - Child inherits both and adds bike().

---

4 Hierarchical Inheritance

- Multiple child classes inherit from the same parent class.
- **Real-life example:**
  Different vehicles inherit properties of a general vehicle.
- **Example:**

```
class Vehicle:  # Parent class
    def fuel(self):
        print("Vehicle uses fuel.")

class Car(Vehicle):  # Child class 1
    def wheels(self):
        print("Car has 4 wheels.")

class Bike(Vehicle): # Child class 2
    def wheels(self):
        print("Bike has 2 wheels.")

c = Car()   # Object creation
b = Bike()
c.fuel()   # ✅ Output: Vehicle uses fuel.
c.wheels() # ✅ Output: Car has 4 wheels.
b.fuel()   # ✅ Output: Vehicle uses fuel.
b.wheels() # ✅ Output: Bike has 2 wheels.
```

- **Explanation:**
  - Car and Bike **both inherit** fuel() from Vehicle.
  - But they have **their own implementations** for wheels().

---

5 Hybrid Inheritance

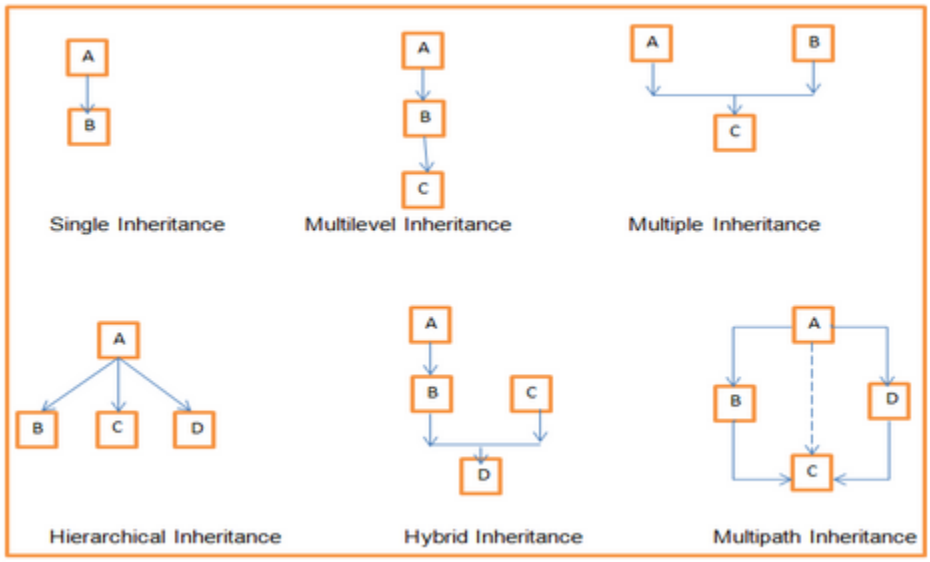
- Combination of two or more types of inheritance.
- **Real-life example:**
  A child inherits from both parents, and their parents inherit from grandparents.
- **Example:**

```
class A:  # Parent class
    def method_A(self):
        print("Class A method.")

class B(A):  # Child class 1 (Single Inheritance)
    def method_B(self):
        print("Class B method.")

class C:  # Child class 2 (Multiple Inheritance)
    def method_C(self):
        print("Class C method.")

class D(B, C):  # Grandchild class (Combining Multiple & Multilevel Inheritance)
    def method_D(self):
        print("Class D method.")

d = D()  # Object creation
d.method_A()  # ✅ Output: Class A method.
d.method_B()  # ✅ Output: Class B method.
d.method_C()  # ✅ Output: Class C method.
d.method_D()  # ✅ Output: Class D method.
```

- **Explanation:**
  - D **inherits from B and C**, which means it has access to methods of both A and C.
  - **This combines multiple and multilevel inheritance**.

- **Advantages of Inheritance:**
  1. **Code reusability:** No need to rewrite the same code again and again.
  2. **Improved maintainability:** Changes in the parent class automatically reflect in child classes.
  3. **Faster development:** Less time is spent writing duplicate code.
  4. **Encapsulation:** Provides a clear structure to the program.

- **Diagram :**



Marathi Tip (मराठी टीप)

- **Inheritance म्हणजे पुन्हा पुन्हा कोड न लिहता वर्गाचे गुणधर्म दुसऱ्या वर्गाकडून घेणे!**
  - **Single Inheritance:** आईचा स्वभाव मुलांमध्ये येतो.
  - **Multiple Inheritance:** मूल आई-वडिलांपासून गुणधर्म घेते.
  - **Multilevel Inheritance:** आजोबा → वडील → मुलगा.
  - **Hierarchical Inheritance:** एकाच वर्गाचे अनेक मुले असणे (प्रत्येकाचे वेगळे वैशिष्ट्य).
  - **Hybrid Inheritance:** दोन प्रकारांचे मिश्रण.

Q. Create a parent class named Animals and a child class Herbivorous which will extend the class Animal. In the child class Herbivorous over side the method feed ( ). Create a object.

Answer:-

```
# Parent class
class Animal:
    def feed(self):
        print("Animals eat food.")

# Child class inheriting Animal class
class Herbivorous(Animal):
    def feed(self):  # Overriding the feed() method
        print("Herbivorous animals eat only plants.")

# Object creation
herb = Herbivorous()
herb.feed()  # ✅ Output: Herbivorous animals eat only plants.
```

Q. Write a Python program to create a class 'Diploma' having a method 'getdiploma' that prints "I got a diploma". It has two subclasses namely 'CO' and 'IF' each having a method with the same name that prints "I am with CO diploma" and 'I am with IF diploma' respectively. Call the method by creating an object of each of the three classes..

Answer :-

```
# Parent class
class Diploma:
    def getdiploma(self):
        print("I got a diploma.")

# Subclass CO (Child of Diploma)
class CO(Diploma):
    def getdiploma(self):  # Overriding method
        print("I am with CO diploma.")

# Subclass IF (Child of Diploma)
class IF(Diploma):
    def getdiploma(self):  # Overriding method
        print("I am with IF diploma.")

# Creating objects
diploma_obj = Diploma()
co_obj = CO()
if_obj = IF()

# Calling methods
diploma_obj.getdiploma()  # Calls parent class method
co_obj.getdiploma()       # Calls overridden method in CO
if_obj.getdiploma()       # Calls overridden method in IF
```

Output

```
I got a diploma.
I am with CO diploma.
I am with IF diploma.
```

❖ Summer 2022

1. Define class and object in python. **(2marks)**
2. Write a program to create class EMPLOYEE with ID and NAME and display its contents. **(4marks)**
3. Explain method overloading in python with example. **(6marks)**
4. Write a program to implement the concept of inheritance in python. **(6marks)**

❖ Winter 2022

1. Write syntax of defining class in Python. **(2marks)**
2. Illustrate with example method over loading. **(4marks)**
3. Design a class student with data members; Name, roll number address. Create suitable method for reading and printing students details. **(6marks)**
4. Create a parent class named Animals and a child class Herbivorous which will extend the class Animal. In the child class Herbivorous over side the method feed ( ). Create a object. **(6marks)**

❖ Summer 2023

1. With neat example explain default constructor concept in Python. **(2marks)**
2. Write a Python program to find the factorial of a number provided by the user. **(4marks)**
3. Write a Python Program to check if a string is palindrome or not. **(6marks)**
4. Design a class student with data members : name, roll no., department, mobile no. Create suitable methods for reading and printing student information. **(6marks)**
5. With suitable example explain inheritance in Python. **(6marks)**

❖ Winter 2023

1. Define Data Hiding concept ? Write two advantages of Data Hiding. **(2marks)**
2. Explain method overloading and overriding in python. **(4marks)**
3. Write a program to create class student with Roll no. and Name and display its contents. **(6marks)**
4. Write program to implement concept of inheritance in python. **(6marks)**

❖ Summer 2024

1. What is data obstruction and data hiding. **(2marks)**

2. Write a python program to create class student with roll-no and display its contents. **(4marks)**
3. Write a Python program to create a class 'Diploma' having a method 'getdiploma' that prints "I got a diploma". It has two subclasses namely 'CO' and 'IF' each having a method with the same name that prints "I am with CO diploma" and 'I am with IF diploma' respectively. Call the method by creating an object of each of the three classes.. **(6marks)**
4. Explain multiple inheritance and write a python program to implement it. **(6marks)**

❖ Winter 2024

1. Define class and object. **(2marks)**
2. List different object oriented features supported by Python. **(2marks)**
3. Describe the concept of inheritance in Python with example . **(4marks)**
4. Illustrate with example method overloading. **(4marks)**
5. Design a class student with data members : Name , Roll no , Address. Create suitable methods for reading and printing students details **(6marks)**