1. Explain what inheritance is in object-oriented programming and why it is used.

Inheritance is a fundamental concept in object-oriented programming that allows child classes to inherit attributes and behaviors from Base Class. It enables the creation of relationship between classes, where new class can inherit attributes and methods from an existing class which can promotes code reuse, modularity, and flexibility.

2. Discuss the concept of single inheritance and multiple inheritance, highlighting their differences and advantages.

Single Inheritance:Single inheritance refers to the ability of class to inherit attributes and behaviors from a single parent class. Advantages are simplicity, Reduced complexity and minimized conflicts as there is only one direct base class.

Multiple Inheritane:Its nothing but the opposite of single inheritance i.e. it refers to ability of class to inherit attributes and behaviors from multiple parent classes. It Enhanced code reuse and flexibility by inheriting from multiple parent classes

Common differnce between these two Inheritances is single inheritance derived class can have only one direct base class, While the in other derived class can have multiple base classes.

3. Explain the terms "base class" and "derived class" in the context of inheritance.

A base class is a class that works as a parent class, providing common attributes and behaviors which are going to be inherited by it's child classes.

A derived class is a class that inherits the attribites and behaviors of the base class, allowing it to extend functionality while maintaining the properties of the base class.

4. What is the significance of the "protected" access modifier in inheritance? How does it differ from "private" and "public" modifiers?

The "protected" access modifier in inheritance allows variables and methods of a class to be accessible within the class itself, as well as by its derived classes. "Public" access modifier allows unrestricted access to the members, both within the class and by any external code. While the "private" access modifier restricts the access of members to only within the class where they are defined, making them inaccessible to derived classes.

5. What is the purpose of the "super" keyword in inheritance? Provide an example.

The "super" keyword in inheritance is used to refer to the base class from within a derived class which allows to you to invoke and access the methods and attributes of the base class. Eg. as below:

In [5]:

```python
class Vehicle:
    def __init__(self,brand):
        self.brand=brand
class Car(Vehicle):
    def __init__(self,brand,model):
        super().__init__(brand)  #super() inheriting base class attribute brand
        self.model=model
    def display_info(self):
        print(f"The Car {self.brand} of model {self.model} is the best car")

car=Car('Honda','i20')
car.display_info()
```

```
The Car Honda of model i20 is the best car
```

6. Create a base class called "Vehicle" with attributes like "make", "model", and "year". Then, create a derived class called "Car" that inherits from "Vehicle" and adds an attribute called "fuel_type". Implement appropriate methods in both classes.

In [3]:

```python
class Vehicle:
    def __init__(self,make,model,year):
        self.make=make
        self.model=model
        self.year=year
class Car(Vehicle):
    def __init__(self,make,model,year,fuel_type):
        super().__init__(make,model,year)
        self.fuel_type=fuel_type
    def diplay_info(self):
        print(f"The car {self.make}{self.model} in made in year {self.year} having
```

In [4]:

```python
car=Car('Honda','city','i20',2002)
car.diplay_info()
```

```
The car Hondacity in made in year i20 having fuel type 2002
```

7. Create a base class called "Employee" with attributes like "name" and "salary." Derive two classes, "Manager" and "Developer," from "Employee." Add an additional attribute called "department" for the "Manager" class and "programming_language" for the "Developer" class.

In [18]:

```python
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary
class Manager(Employee):
    def __init__(self,name,salary,department):
        super().__init__(name,salary)
        self.department=department
    def diplay_details(self):
        print(f"The Employee {self.name} in a {self.department} department having sa
class Developer(Employee):
    def __init__(self,name,salary,programming_language):
        super().__init__(name,salary)
        self.programming_language=programming_language
    def display_details(self):
        print(f"The Employee {self.name} is a {self.programming_language} developer
```

In [19]:

```python
manager=Manager('Prad',15000,'Operations')
manager.diplay_details()
developer=Developer('Hritik',20000,'Python')
developer.display_details()
```

```
The Employee Prad in a Operations department having salary 15000
The Employee Hritik is a Python developer having salary 20000
```

8. Design a base class called "Shape" with attributes like "colour" and "border_width." Create derived classes, "Rectangle" and "Circle," that inherit from "Shape" and add specific attributes like "length" and "width" for the "Rectangle" class and "radius" for the "Circle" class.

In [3]:

```python
class Shape:
    def __init__(self,colour,border_width):
        self.colour=colour
        self.border_width=border_width
class Rectangle(Shape):
    def __init__(self,colour,border_width,length,width):
        super().__init__(colour,border_width)
        self.len=length
        self.wid=width
    def display(self):
        print(f"The rectangle is {self.colour} coloured having border_width {self.bo
class Circle(Shape):
    def __init__(self,colour,border_width,radius):
        super().__init__(colour,border_width)
        self.rad=radius
    def display(self):
        print(f"The Circle is {self.colour} coloured having border_width {self.borde
```

In [4]:

```python
rec=Rectangle('Red','2.5 cm','10 cm','2 cm')
rec.display()

cir=Circle('Blue','2 cm','6 cm')
cir.display()
```

```
The rectangle is Red coloured having border_width 2.5 cm,length 10 cm and width 2 cm
The Circle is Blue coloured having border_width 2 cm and radius 6 cm
```

9. Create a base class called "Device" with attributes like "brand" and "model." Derive two classes, "Phone" and "Tablet," from "Device." Add specific attributes like "screen_size" for the "Phone" class and "battery_capacity" for the "Tablet" class.

In [4]:

```python
class Device:
    def __init__(self,brand,model):
        self.brand=brand
        self.model=model
class Phone(Device):
    def __init__(self,brand,model,screen_size):
        super().__init__(brand,model)
        self.screen_size=screen_size
    def display_info(self):
        print(f"Phone {self.brand} with model {self.model} have screen size {self.s
class Tablet(Device):
    def __init__(self,brand,model,battery_capacity):
        super().__init__(brand,model)
        self.battery_capacity=battery_capacity
    def display_info(self):
        print(f"Tablet {self.brand} with model {self.model} have battery capacity o
```

In [5]:

```python
phone=Phone('realme','3 pro','6 inches')
phone.display_info()
tab=Tablet('Apple','Ipad','8 Hours')
tab.display_info()
```

```
Phone realme with model 3 pro have screen size 6 inches
Tablet Apple with model Ipad have battery capacity of 8 Hours
```

10. Create a base class called "BankAccount" with attributes like "account_number" and "balance." Derive two classes, "SavingsAccount" and "CheckingAccount," from "BankAccount." Add specific methods like "calculate_interest" for the "SavingsAccount" class and "deduct_fees" for the "CheckingAccount" class.

In [20]:

```python
class BankAccount:
    def __init__(self, account_number, balance):
        self.acc_num=account_number
        self.bal=balance
    def deposit(self, amount):
        self.bal+=amount
    def withdraw(self, amount):
        if amount<=self.bal:
            self.bal-=amount
    def display_balance(self):
        print(f"Account Number: {self.acc_num}")
        print(f"Balance: {self.bal}")
class SavingsAccount(BankAccount):
    def calculate_interest(self, rate):
        interest=self.bal*rate
        self.deposit(interest)

class CheckingAccount(BankAccount):
    def deduct_fees(self, fee):
        if self.bal>=fee:
            self.withdraw(fee)
```

In [21]:

```python
savings = SavingsAccount("000123", 1000)
savings.display_balance()
```

```
Account Number: 000123
Balance: 1000
```

In [22]:

```python
savings.calculate_interest(0.05)
savings.display_balance()
```

```
Account Number: 000123
Balance: 1050.0
```

In [26]:

```python
checking = CheckingAccount("000789", 500)
checking.display_balance()
```

```
Account Number: 000789
Balance: 500
```

In [27]:

```python
checking.deduct_fees(100)
checking.display_balance()
```

```
Account Number: 000789
Balance: 400
```