1. Create a class with attributes brand and model. Initialize these attributes using the constructor and create an instance of the class to display the car's details.

→
```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print("Car Brand:", self.brand)
        print("Car Model:", self.model)

car1 = Car("Toyota", "Corolla")
car1.display_info()
```

OUTPUT - Q.1

Car Brand : Toyota

Car Model : Corolla

2. Create a class Student with attributes name and age. Implement a method greet() that prints a greeting message with the student's name. Create an instance of the class and call the method.

→

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

    def greet(self):
        print(f"Hello, my name is {self.name} and I am
                {self.age} years old")

Student1 = Student("Sanjay", 20)
Student1.greet()
```

OUTPUT - Q.2

Hello, My name is Sanjay and I am 20 years old

3. Create a Book class with attributes title and author. Implement a method show_details() to display the book's title and author. Create an instance and call the method.

→
```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def show_details(self):
        print("Book Title:", self.title)
        print("Author:", self.author)

Book1 = Book("To Kill a Mockingbird", "Harper Lee")
Book1.show_details()
```

OUTPUT - Q.3

Book Title : To Kill a MockingBird

Author : Harper Lee

4. Create a Rectangle class with attributes length and width. Implement a method area() to calculate and return 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

    def area():
        return self.length * self.width

rectangle1 = Rectangle(5, 3):
print("Area of Rectangle:", rectangle1.area())
```

Q.4 - OUTPUT

Area of Rectangle: 15

5. Create a Person class with an attribute name. Implement a method introduce() that prints a greeting message with the person's name. Create an instance and call the method.

→
```
Class Person:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        print(f"Hi, my name is {self.name}.")

Person1 = Person("Sanjay")
Person1.introduce()
```

OUTPUT - Q5

Hi, My name is Sanjay.

6. Write a Python class called Car that has attributes like make, model, and year. Create an object of this class and print the values of these attributes.

→
```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

car1 = Car("Toyota", "Corolla", 2020)
print(car1.make, car1.model, car1.year)
```

OUTPUT- Q.6

Toyota Corolla 2020

7. Create a class Student with an instance method display_student_info that prints the name, age, and grade of the student. Then, create an object of the student class and display the student's information.

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

    def display_student_info(self):
        print(f"name: {self.name}, Age: {self.age}. Grad:
                                                {self.grade}")

Student1 = Student("Sanjay", 15, "A")
Student1.display_student_info()
```

OUTPUT - Q.7

Name: Sanjay , Age: 15 , Grade: A

8. write a Python class Employee that has a class variable company_name. Define an instance method that displays the employee's name and company name. Show how the class variable can be accessed both by an instance and directly from the class.

→
```
class Employee:
    Company_name = "TechCorp"

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

    def display_employee_info(self):
        print(f"Employee Name: {self.name}, company:
                                {Employee.company_name}")

emp1 = Employee("John")
emp1.display_employee_info()
print(Employee.company_name)
```

OUTPUT- Q.8

Employee Name: John, Company: TechCorp
TechCorp

9. Create two classes, Person and Address. In Person, store the name and age, and in Address, store the address details. Use composition to pass a Person object into the Address class and display both person's and address's details.

→

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

class Address:
    def __init__(self, person, street, city):
        self.person = person
        self.street = street
        self.city = city

    def display_full_info(self):
        print(f"Name: {self.person.name}, Age: {self.person.age}")
        print(f"Address: {self.street}, {self.city}")

person = Person("Alice", 20)
address1 = Address(person1, "123 Main St", "New York")
address1.display_full_info()
```

OUTPUT – Q.9

Name: Alice, Age: 20

Address: 123 Main St, New York

10. Write a Python class Book that has a class variable category. Create an instance method to set the title and author of the book. Demonstrate how an instance can modify the class variable and how it can be accessed via the instance and the class.

→

```
class Book:
    category = "Fiction"

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def display_book_info(self):
        print(f"Title : {self.title}, Author : {self.author},
              Category : {Book.category}")

book1 = Book("The Great Gatsby", "F. Scott Fitzgerald")
book1.display_book_info()
Book.category = "Classic"
book.display_book_info()
```

OUTPUT – Q.10

Title: The Great Gatsby, Author: F. Scott Fitzgerald, Category: Fiction

Title: The Great Gatsby, Author: F. Scott Fitzgerald, Category: Classic

11. Create a class Person with attributes name and age, and a subclass Employee that adds an attribute job_title. Display the details of an employee.

→

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

class Employee(Person):
    def __init__(self, name, age, job_title):
        super().__init__(name, age)
        self.job_title = job_title

    def display_info(self):
        print("Name:", self.name)
        print("Age:", self.age)
        print("Job Title:", self.job_title)

employee = Employee("Alice", 30, "Software Engineer")
employee.display_info()
```

OUTPUT - Q.11

Name: Alice

Age: 30

Job Title: Software Engineer

12. Create a class Person with attributes name and age, and another class Address with attributes city and country. The subclass Employee should inherit both Person and Address. Display all details.

→

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
class Address:
    def __init__(self, city, country):
        self.city = city
        self.country = country
class Employee(Person, Address):
    def __init__(Self, name, age, city, Country, job-title):
        Person.__init__(Self, name, age)
        Address.__init__(Self, city, Country)
        Self.job-title = job-title
def display-info(Self):
        Print("Name:", Self.name)
        Print("Age:", Self.age)
        Print("City:", Self.city)
        Print("Country:", Self.country)
        Print("Job Title:", Self.job-title)

employee = Employee("Bob", 40, "New York", USA", "Manager")
employee.display-info()
```

OUTPUT- Q.12

Name: Bob

Age: 40

City: New York

Country: USA

Job Title: Manager

13. Create a Superclass Animal with a method make_Sound, and a Subclass Dog that overrides this method. Call the method on both the superclass and subclass objects.

→
```
class Animal:
    def make_sound(self):
        print("Same generic animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Bark")

animal = Animal()
dog = Dog()

animal.make_sound()
dog.make_sound()
```

OUTPUT
Q. 13 → Same generic animal sound
Bark

~~Q. 14~~ Create an abstract class Shape with an abstract method area. Then, create two subclasses Rectangle and Circle that implement the area method. Print the area for both shapes.

Q. 14 Create a class Person with a constructor, and a subclass Employee that calls the superclass constructor using super(). Display the details of the employee.

```
class Person:
    def __init__(self, name):
        self.name = name
        print("Person's Name:", self.name)

class Employee(Person):
    def __init__(self, name, salary):
        super().__init__(name)
        self.salary = salary
        print("Employee's Salary:", self.salary)

employee = Employee("John", 50000)
```

OUTPUT
Q. 14 →
Person's Name: John
Employee's Salary: 50000

15. Create an abstract class Shape with an abstract method area. Then, create two subclasses Rectangles and Circle that implement the area method. Print the area for both shapes.

→
```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius * self.radius
rect = Rectangle(5, 3)
print("Rectangle of Area:", rect.area())
circle = Circle(7)
print("Circle Area:", circle.area())
```

OUTPUT :- Q.15

```
Rectangle of Area: 15
Circle Area: 153.86
```

Q.16 Define an abstract class Vehicle with an abstract method fuel_efficiency. Create subclasses Car and Truck that implement the fuel_efficiency method. Instantiate both and print their fuel efficiencies.

```
from abc import ABC, abstractmethod
class Vehicle(ABC):
    @abstractmethod
    def fuel_efficiency(self):
        pass

class Car(Vehicle):
    def __init__(self, mileage):
        self.mileage = mileage
    def fuel_efficiency(self):
        return self.mileage

class Truck(Vehicle):
    def __init__(self, mileage):
        self.mileage = mileage
    def fuel_efficiency(self):
        return self.mileage
car = Car(15)
print("Car Fuel Efficiency: ", car.fuel_efficiency(), "Km/I")
truck = Truck(8)
print("Truck Fuel Efficiency: ", truck.fuel_efficiency(), "Km/I")
```

OUTPUT Q.16 →

```
Car Fuel Efficiency: 15 Km/I
Truck Fuel Efficiency: 8 Km/I
```

Q.17 Create an interface Playable using an abstract class that has an abstract method play. then, create classes Musicplayer and videoplayer that implement ~~that~~ the play method. Instantiate both and demonstrate playing music and video.

→
```
from abc import ABC, abstractmethod
  class playable (ABC):
        @abstractmethod
        def play (self):
         pass
class Musicplayer (playable):
      def play (self):
      Print (" playing music")
class videoplayer (playable):
    def play (self):
    Print (" playing video")
music - player = musicplayer()
music - player. play()
video - player = videoplayer()
  video - player. play()
```

OUTPUT

Q.17→ Playing Music
Playing video

OUTPUT

Q.18 →

Hello
Hello
Hello
Hello
Hello
World
World
World
World
World
Don

Q.18 Create two threads. one should Print "Hello" 5 times, and the other should Print "World" 5 times. use join() to wait for both Threads to finish before Printing "Done".

→
```
import threading
  def Print - hello():
      for - in range(5):
        Print ("Hello")
def print - World():
   for - in range(5):
   Print ("World")

  Creating threads
thread1 = threading.Thread(target = Print - hello)
thread2 = threading.thread(target = Print - World)

  Starting threads
     thread1.start()
     thread2.start()
  Waiting for threads to finish
    thread1.join()
    thread2.join()

     Print ("Done")
```

Q.19 Create two threads, each incrementing a shared Counter 100 times. Use a Lock to avoid race conditions and ensure correct Counter updates.

```
import threading
Counter = 0
lock = threading.Lock()
def increment_Counter():
    global Counter
    for _ in range(100):
        with Lock:
            Counter += 1

thread1 = threading.Thread(target=increment_counter)
thread2 = threading.Thread(target=increment_counter)

thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("Final Counter:", Counter)
```

OUTPUT - Q.19

Final Counter: 200

Q.20 Demonstrate a deadlock scenario by creating two threads. Each thread tries to acquire two locks in reverse order, causing both threads to wait indefinitely.

→
```
import threading
lock1 = threading.Lock()
lock2 = threading.Lock()
def thread1_function():
    lock1.acquire()
    print("Thread 1: Lock1 Acquired")
    lock2.acquire()
    print("Thread 1: Lock 2 Acquired")

def thread2_function()
    lock2.acquire()
    print("Thread 2: Lock 2 acquired")
    lock1.acquire()
    print("Thread 2: Lock 1 Acquired")
thread1 = threading.Thread(target=thread1_function)
thread2 = threading.Thread(target=thread2_function)
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("Exection finished")
```

OUTPUT :- Q.20

```
Thread 1: Lock1 Acquired
Thread 1: Lock2 acquired
```