Python Test

In [98]:
```python
# Que1. Define a class called Vehicle. Include attributes like make, model, and year.
# Add an initialization method to set these attributes.

# class vehicle
class Vehicle:
    def __init__(self,make, model, year):
        self.make = make
        self.model = model
        self.year = year



# instantiating
car = Vehicle(2012,'Polo',2024)
```

In [99]:
```python
# Que2. Add a method called display_info to the Vehicle class that prints out the vehicl
# information in a nicely formatted string.

# class vehicle
class Vehicle:
    def __init__(self,make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display(self):
        return f"Made in year: {self.make}\nModel: {self.model}\nYear: {self.year}"


# instantiating
car = Vehicle(2012,'Polo',2024)
print(car.display())
```

```
Made in year: 2012
Model: Polo
Year: 2024
```

In [100…
```python
# Que3. Create a subclass of Vehicle called Car that includes additional attributes like
# Override the display_info method to include these new attributes.

# subclass
class Car(Vehicle):

    def __init__(self, make, model, year,door,engine_type):
        super().__init__(make, model, year)
        self.door = door
        self.engine_type = engine_type

    # display
    def display(self):
        return f"{super().display()}\nDoor : {self.door}\nEngine Type : {self.engine_typ
```

```
Car1 = Car(2001,"Polo",2023,4,"V2")
print(Car1.display())
```

```
Made in year: 2001
Model: Polo
Year: 2023
Door : 4
Engine Type : V2
```

In [101…

```
# Que4. Write a method in the Car class to start_engine.
# This method should print a message that says the engine is starting, using the engine_

class Car(Vehicle):

    def __init__(self, make, model, year,door,engine_type):
        super().__init__(make, model, year)
        self.door = door
        self.engine_type = engine_type

    # display
    def display(self):
        return f"{super().display()}\nDoor : {self.door}\nEngine Type : {self.engine_ty

    # start_engine
    def start_engine(self):
        return "Engine is starting"


Car1 = Car(2001,"Polo",2023,4,"V2")
print(Car1.display())
print(Car1.start_engine())
```

```
Made in year: 2001
Model: Polo
Year: 2023
Door : 4
Engine Type : V2
Engine is starting
```

In [102…

```
# Que5. Implement a Bicycle subclass that inherits from Vehicle.
# Include additional attributes such as gear_count and bicycle_type.

class Bicycle(Vehicle):

    def __init__(self, make, model, year,gear_count,bicycle_type):
        super().__init__(make, model, year)
        self.gear_count = gear_count
        self.bicycle_type = bicycle_type
```

In [103…

```
# Que6. Create a Python class Circle. Use the __init__ method to set the radius
# and a method to calculate the area (use pi value from the math module).
import math

class Circle:

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

    def area(self):
```

```
        pi = math.pi
        return f"Area : {pi * (self.radius ** 2)}"
```

In [104…
```
# Que7. Define a class Rectangle with attributes length and width. Include methods to c

class Rectangle:

    def __init__(self,length,width):
        self.length = length
        self.width = width

    def area(self):
        return f"Area : {self.length * self.width}"


    def perimeter(self):
        return f"Perimeter : {(self.length * self.width)*2}"
```

In [105…
```
# Que8. Create a class Employee with properties name and employee_id.
# Include a method to print an email address, assuming it's employee_id@company.com.

class Employee:

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


    def display(self):
        return f"Name : {self.name}\nEmployee No. : {self.employee_id}"

    def displayMail(self):
        return f"{(self.employee_id).lower()}@company.com"

#
emp1 = Employee("Sanket","ReBIT2428")

print(emp1.displayMail())
```

```
rebit2428@company.com
```

In [106…
```
# Que9. Extend the Employee class with a subclass Manager
# that has an additional attribute department and a method to print department details.

class Manager(Employee):

    def __init__(self, name, employee_id,department):
        super().__init__(name, employee_id)
        self.department = department

    def displayDepartment(self):
        return f"Department : {self.department}"

# object
dep1 = Manager('chirantan','rebit2200',"Training")
print(dep1.display())
print(dep1.displayDepartment())
```

```
Name : chirantan
Employee No. : rebit2200
Department : Training
```

In [107…

```python
# Que10. Define a class ComplexNumber to represent complex numbers.
# Include methods to add and multiply two complex numbers.

class ComplexNumber:

    def __init__(self,real, imaginary):
        self.real = real
        self.imaginary = imaginary

    def __add__(self,other):
        return ComplexNumber(self.real + other.real, self.imaginary + other.imaginary)

    def __mul__(self,other):
        real_part = self.real * other.real - self.imaginary * other.imaginary
        imag_part = self.real * other.imaginary + self.imaginary * other.real
        return ComplexNumber(real_part, imag_part)

    def __str__(self):
        return f"{self.real} + {self.imaginary}i" if self.imaginary >= 0 else f"{self.r


# object
comp1 = ComplexNumber(3,4)
comp2 = ComplexNumber(1,2)

sum_result = comp1 + comp2
print(f"sum result : {sum_result}")

multi_result = comp1 * comp2
print(f"Multiplication result : {multi_result}")
```

```
sum result : 4 + 6i
Multiplication result : -5 + 10i
```

In [108…

```python
# Que11. Create a class Book with attributes title, author, and price.
# Add a method to apply a discount to the price and another to format the book details c

class Book:

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

    def Discount(self):
        percent = int(input("Enter Discount Percentage: "))

        total = self.price - (self.price / 100) * percent
        return f"Total after discount : {total}"

    def __str__(self):
        return f"Name : {self.title}\nAuthor : {self.author}\nPrice : {self.price}"

# object
```

```
book1 = Book('Magic',"Rhonda Burn", 399)
print(str(book1))
```

```
Name : Magic
Author : Rhonda Burn
Price : 399
```

In [109…

```
# Que12. Implement a class Flight that can
# keep track of airline, flight number, and the list of passengers (use passenger names)

class Flight:

    def __init__(self,airline,flight_number):
        self.airline = airline
        self.flight_number = flight_number
        self.passanger = []


    def __str__(self):
        return f"Airline : {self.airline}\nFlight Number : {self.flight_number}"


# object
flg = Flight('Air India',"AI-5511")
print(flg)
```

```
Airline : Air India
Flight Number : AI-5511
```

In [110…

```
#Que13. Add methods to the Flight class to add a passenger to the flight and to remove a

class Flight:

    def __init__(self,airline,flight_number):
        self.airline = airline
        self.flight_number = flight_number
        self.passanger = []

    def add_passanger(self,name):
        if name not in self.passanger:
            self.passanger.append(name)
            return "Passanger added successfully"

    def remove_passanger(self,name):
        if name in self.passanger:
            self.passanger.remove(name)
        else:
            return "Passanger removed successfully"

    def get_passanger(self):

        if len(self.passanger) == 0:
            return "No Passanger"
        else:
            print("List of passanger")
            for names in self.passanger:
                print(names)
            return "Success"
```

```
    def __str__(self):
        return f"Airline : {self.airline}\nFlight Number : {self.flight_number}"


# object
flg = Flight('Air India',"AI-5511")
print(flg)

# get passanger0
print(flg.get_passanger())
print()

# add passanger
flg.add_passanger("Sanket")
flg.add_passanger("Siddharth")
flg.add_passanger("Pratham")
flg.add_passanger("Durga")
print()

# get passanger
print(flg.get_passanger())
print()

# remove passanger
flg.remove_passanger("Pratham")
print()

# get passanger
print(flg.get_passanger())
```

```
Airline : Air India
Flight Number : AI-5511
No Passanger


List of passanger
Sanket
Siddharth
Pratham
Durga
Success


List of passanger
Sanket
Siddharth
Durga
Success
```

In [111…

```
# Que14. Create a class Animal with an attribute species. Add a method make_sound that p

class Animal:

    def __init__(self,species):
        self.species = species
```

```
    def sound(self):
        return 'sound of animal'
```

In [112…

```
# Que15. Subclass Animal with Dog and Cat classes.
# Override the make_sound method to print "Woof" for dogs and "Meow" for cats.

class Dog(Animal):

    def __init__(self, species):
        super().__init__(species)

    def sound(self):
        return "Woof Woof!!"

class Cat(Animal):

    def __init__(self, species):
        super().__init__(species)

    def sound(self):
        return "Meow Meow!!"
```

In [113…

```
# Que16. Develop a class Calculator with methods for basic operations: add, subtract, mu

class Calc:

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

    def __add__(self,other):
        return self.number + other.number

    def __sub__(self,other):
        return self.number - other.number

    def __mul__(self,other):
        return self.number * other.number

    def __truediv__(self, other):
        try:
            if other.number == 0:
                raise ZeroDivisionError("Division by zero is not allowed.")

            div = self.number / other.number
            return div
        except ZeroDivisionError as e:
            print(e)


# object
cal1 = Calc(50)
cal2 = Calc(32)
cal3 = Calc(0)

print(f"Sum : {cal1 + cal2}")
print(f"Substraction : {cal1 - cal2}")
print(f"Multiplication : {cal1 * cal2}")
```

```
print(f"Division : {cal1 / cal2}")
print(f"Division : {cal1 / cal3}")
```

```
Sum : 82
Substraction : 18
Multiplication : 1600
Division : 1.5625
Division by zero is not allowed.
Division : None
```

In [114…

```
# Que17. Define a Polygon class with methods to calculate perimeter and area.
# This class should be designed to be subclassed by specific polygon types like triangle

class WeatherForecast:

    def __init__(self):
        self.temp = []

    def addTemp(self, temperature):
        self.temp.append(temperature)

    def avgTemp(self):
        avg = sum(self.temp) / len(self.temp)
        return f"Average temperature is {avg} Celcius"

# Object
t = WeatherForecast()

t.addTemp(34)
t.addTemp(24)
t.addTemp(29)
print(t.avgTemp())
```

```
Average temperature is 29.0 Celcius
```

In [115…

```
# Que18. Define a Polygon class with methods to calculate perimeter and area.
# This class should be designed to be subclassed by specific polygon types like triangle

class Polygon:

    def __init__(self,*args):
        self.args = args

    def Perimeter(self):
        pass

    def Area(self):
        pass
```

In [116…

```
# Que19. Implement a subclass of Polygon for Triangle.
# Use attributes for the sides and appropriate methods to compute area (using Heron's fo

class Triangle(Polygon):

    def __init__(self, side1,side2,side3):
        self.side1 = side1
        self.side2 = side2
```

```
        self.side3 = side3

    def Area(self):
        semi_peri = (self.side1 + self.side2 + self.side3) / 2
        area = (semi_peri * (semi_peri - self.side1) * (semi_peri - self.side2) * (semi_
        return f"Area of Trianlge is {area:.2f}"


# object
obj1 = Triangle(5,8,10)

print(obj1.Area())
```

```
Area of Trianlge is 19.81
```

In [117…

```
# Que20. Write a Square subclass of Polygon with a method override for area calculation
class Square(Polygon):

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

    def area(self):
        ar = self.side * self.side
        return f"Area of Square {ar}"


# object
obj1 = Square(25)
print(obj1.area())
```

```
Area of Square 625
```

In [119…

```
# Que21. Create a class Timer that can be used to time operations in Python.
# Use the time module for tracking start and end times.
import time

class Timer:

    def __init__(self):
        self.start_time = None
        self.end_time = None

    def start(self):
        self.start_time = time.time()
        print("Time started")

    def end(self):
        self.end_time = time.time()
        total_time = self.end_time - self.start_time
        return f"Total time taken by operation {total_time:.2f} seconds"

# object
t = Timer()

t.start()

a = 0
while a < 100000000:
    a += 1
```

```
t.end()
```

```
Time started
```

Out[119… 'Total time taken by operation 10.91 seconds'

In [ ]:
```python
# Que22. Design a Queue class using lists. Implement methods for enqueue, dequeue, and v

class Queue:

    def __init__(self):
        self.items = []

    def enque(self,element):
        self.items.append(element)
        print(f"element {element} added successfully")

    def deque(self):
        self.items.pop(0)
        print(f"element {self.items[0]} removed successfully")

    def display(self):
        if len(self.items) == 0:
            return "Empty list"
        else:
            return f"Queue Items: {self.items}"


# object
obj1 = Queue()

print(obj1.display())
obj1.enque(5)
obj1.enque(10)
obj1.enque(23)
obj1.deque()
obj1.enque(59)

print(obj1.display())
```

```
Empty list
element 5 added successfully
element 10 added successfully
element 23 added successfully
element 10 removed successfully
element 59 added successfully
List Items: [10, 23, 59]
```

In [130…
```python
# Que 23. Create a class Stack with methods for pushing, popping, and checking the size

class Stack:

    def __init__(self):
        self.items = []

    def push(self,element):
        self.items.append(element)
```

```
    def pop(self):
        self.items.pop()

    def size(self):
        return f"Size of stack : {len(self.items)}"

    def display(self):
        return f"Stack Elements : {self.items}"

# object
obj1 = Stack()

# operations
print(obj1.size())
print(obj1.display())

obj1.push(2)
obj1.push(95)
obj1.push(19)
obj1.push(75)

obj1.pop()
obj1.pop()

print(obj1.display())
print(obj1.size())
```

```
Size of stack : 0
Stack Elements : []
Stack Elements : [2, 95]
Size of stack : 2
```

In [131…

```
# Que24. Develop a MusicPlayer class with methods to play, stop, and load music tracks (

class MusicPlayer:

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

    def play(self):
        return f"Playing {self.song_name} song..."

    def stop(self):
        return f"Pausing song {self.song_name}"

    def loadMusic(self,new_song):
        return f"Playing new song {new_song}"

# object
song = MusicPlayer("skyfall")
print(song.play())
print(song.stop())
print(song.loadMusic("Night Changes"))
```

```
Playing skyfall song...
Pausing song skyfall
Playing new song Night Changes
```

In [139…

```python
# Que25. Implement a Database class that simulates a simple database interaction
# with methods to connect, disconnect, and execute a query (simulated with print stateme

class Database:

    def __init__(self,db_name):
        self.db_name =db_name
        self.connection = False

    def connect(self):

        if self.connection:
            return f"{self.db_name} already connected"
        else:
            self.connection = True
            return f"{self.db_name} connected now"

    def disconnect(self):

        if not self.connection:
            return f"{self.db_name} is not connected"
        else:
            self.connection = False
            return f"Disconnecting now {self.db_name}"

    def query(self,query):
        if self.connection:
            return f"Executing : {query}"
        else:
            return f"Query not executed \n{self.db_name} is not connected"

# object
db = Database("School")

print(db.connect())
print(db.disconnect())
print(db.query("this is query"))
print(db.connect())
print(db.query("this is query"))
```

```
School connected now
Disconnecting now School
Query not executed
School is not connected
School connected now
Executing : this is query
```

In [ ]: