```
#1. Write a program to create a class that represents Complex numbers containing real
#and imaginary parts and then use it to perform complex number addition, subtraction,
#multiplication and division.

class Complex:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def display(self):
        if self.imag >= 0:
            print(f"{self.real} + {self.imag}i")
        else:
            print(f"{self.real} - {abs(self.imag)}i")

    def add(self, other):
        result = Complex(self.real + other.real, self.imag + other.imag)
        return result

    def subtract(self, other):
        result = Complex(self.real - other.real, self.imag - other.imag)
        return result

    def multiply(self, other):
        real_part = self.real * other.real - self.imag * other.imag
        imag_part = self.real * other.imag + self.imag * other.real
        result = Complex(real_part, imag_part)
        return result

    def divide(self, other):
        denominator = other.real**2 + other.imag**2
        if denominator == 0:
            print("Division by zero is not allowed!")
            return None
        real_part = (self.real * other.real + self.imag * other.imag) / denominator
        imag_part = (self.imag * other.real - self.real * other.imag) / denominator
        result = Complex(real_part, imag_part)
        return result

# Example usage
c1 = Complex(4, 5)
c2 = Complex(2, 3)

print("First Complex Number:")
c1.display()

print("\nSecond Complex Number:")
c2.display()

print("\nAddition:")
result_add = c1.add(c2)
result_add.display()

print("\nSubtraction:")
result_sub = c1.subtract(c2)
result_sub.display()
```

```
print("\nMultiplication:")
result_mul = c1.multiply(c2)
result_mul.display()

print("\nDivision:")
result_div = c1.divide(c2)
if result_div:
    result_div.display()
```

```
First Complex Number:
4 + 5i

Second Complex Number:
2 + 3i

Addition:
6 + 8i

Subtraction:
2 + 2i

Multiplication:
-7 + 22i

Division:
1.7692307692307692 - 0.15384615384615385i
```

```
#2. Write a program that implements a Matrix class and performs addition, multiplication
#and transpose operations on 3x3 matrices.
class Matrix:
    def __init__(self, data):
        if len(data) == 3 and all(len(row) == 3 for row in data):
            self.data = data
        else:
            raise ValueError("Matrix must be 3x3!")

    def display(self):
        for row in self.data:
            print(row)
        print()

    def add(self, other):
        result = []
        for i in range(3):
            row = []
            for j in range(3):
                row.append(self.data[i][j] + other.data[i][j])
            result.append(row)
        return Matrix(result)

    def multiply(self, other):
        result = []
        for i in range(3):
            row = []
            for j in range(3):
                sum = 0
                for k in range(3):
                    sum += self.data[i][k] * other.data[k][j]
                row.append(sum)
```

```
                result.append(row)
            return Matrix(result)

        def transpose(self):
            result = []
            for i in range(3):
                row = []
                for j in range(3):
                    row.append(self.data[j][i])
                result.append(row)
            return Matrix(result)

    # Example usage
    m1 = Matrix([
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ])

    m2 = Matrix([
        [9, 8, 7],
        [6, 5, 4],
        [3, 2, 1]
    ])

    print("Matrix 1:")
    m1.display()

    print("Matrix 2:")
    m2.display()

    print("Addition of Matrices:")
    result_add = m1.add(m2)
    result_add.display()

    print("Multiplication of Matrices:")
    result_mul = m1.multiply(m2)
    result_mul.display()

    print("Transpose of Matrix 1:")
    transpose_m1 = m1.transpose()
    transpose_m1.display()

    print("Transpose of Matrix 2:")
    transpose_m2 = m2.transpose()
    transpose_m2.display()
```

```
Matrix 1:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Matrix 2:
[9, 8, 7]
[6, 5, 4]
[3, 2, 1]

Addition of Matrices:
[10, 10, 10]
```

```
[10, 10, 10]
[10, 10, 10]

Multiplication of Matrices:
[30, 24, 18]
[84, 69, 54]
[138, 114, 90]

Transpose of Matrix 1:
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]

Transpose of Matrix 2:
[9, 6, 3]
[8, 5, 2]
[7, 4, 1]
```

```
#3. Write a program to create a class that can calculate the surface area and volume of
#a solid. The class should also have a provision to accept the data relevant to the soli

class Solid:
    def __init__(self):
        self.shape = None
        self.data = {}

    def accept_data(self):
        self.shape = input("Enter the shape (cube/sphere/cylinder): ").lower()

        if self.shape == "cube":
            self.data['side'] = float(input("Enter the side length of the cube: "))

        elif self.shape == "sphere":
            self.data['radius'] = float(input("Enter the radius of the sphere: "))

        elif self.shape == "cylinder":
            self.data['radius'] = float(input("Enter the radius of the cylinder: "))
            self.data['height'] = float(input("Enter the height of the cylinder: "))

        else:
            print("Unsupported shape!")

    def surface_area(self):
        if self.shape == "cube":
            side = self.data['side']
            return 6 * (side ** 2)

        elif self.shape == "sphere":
            radius = self.data['radius']
            return 4 * 3.14159 * (radius ** 2)

        elif self.shape == "cylinder":
            radius = self.data['radius']
            height = self.data['height']
            return 2 * 3.14159 * radius * (radius + height)

        else:
            return None

    def volume(self):
```

```
        if self.shape == "cube":
            side = self.data['side']
            return side ** 3

        elif self.shape == "sphere":
            radius = self.data['radius']
            return (4/3) * 3.14159 * (radius ** 3)

        elif self.shape == "cylinder":
            radius = self.data['radius']
            height = self.data['height']
            return 3.14159 * (radius ** 2) * height

        else:
            return None

# Example usage
solid = Solid()
solid.accept_data()

area = solid.surface_area()
volume = solid.volume()

if area is not None and volume is not None:
    print(f"Surface Area of {solid.shape.capitalize()}: {area:.2f}")
    print(f"Volume of {solid.shape.capitalize()}: {volume:.2f}")
else:
    print("Calculation could not be performed.")
```

```
Enter the shape (cube/sphere/cylinder): cube
Enter the side length of the cube: 2
Surface Area of Cube: 24.00
Volume of Cube: 8.00
```

```
#4. Write a program to create a class that can calculate the perimeter/circumference
#and area of a regular shape. The class should also have a provision to accept the data
#relevant to the shape.
class RegularShape:
    def __init__(self):
        self.shape = None
        self.data = {}

    def accept_data(self):
        self.shape = input("Enter the shape (square/rectangle/circle/triangle): ").lower

        if self.shape == "square":
            self.data['side'] = float(input("Enter the side length of the square: "))

        elif self.shape == "rectangle":
            self.data['length'] = float(input("Enter the length of the rectangle: "))
            self.data['width'] = float(input("Enter the width of the rectangle: "))

        elif self.shape == "circle":
            self.data['radius'] = float(input("Enter the radius of the circle: "))

        elif self.shape == "triangle":
            self.data['a'] = float(input("Enter side a: "))
```

```
            self.data['b'] = float(input("Enter side b: "))
            self.data['c'] = float(input("Enter side c: "))

        else:
            print("Unsupported shape!")

    def perimeter(self):
        if self.shape == "square":
            side = self.data['side']
            return 4 * side

        elif self.shape == "rectangle":
            length = self.data['length']
            width = self.data['width']
            return 2 * (length + width)

        elif self.shape == "circle":
            radius = self.data['radius']
            return 2 * 3.14159 * radius

        elif self.shape == "triangle":
            return self.data['a'] + self.data['b'] + self.data['c']

        else:
            return None

    def area(self):
        if self.shape == "square":
            side = self.data['side']
            return side * side

        elif self.shape == "rectangle":
            length = self.data['length']
            width = self.data['width']
            return length * width

        elif self.shape == "circle":
            radius = self.data['radius']
            return 3.14159 * radius * radius

        elif self.shape == "triangle":
            a = self.data['a']
            b = self.data['b']
            c = self.data['c']
            s = (a + b + c) / 2
            return (s * (s - a) * (s - b) * (s - c)) ** 0.5  # Heron's formula

        else:
            return None

# Example usage
shape = RegularShape()
shape.accept_data()

peri = shape.perimeter()
ar = shape.area()
```

```
if peri is not None and ar is not None:
    print(f"Perimeter/Circumference of {shape.shape.capitalize()}: {peri:.2f}")
    print(f"Area of {shape.shape.capitalize()}: {ar:.2f}")
else:
    print("Calculation could not be performed.")
```

```
Enter the shape (square/rectangle/circle/triangle): square
Enter the side length of the square: 5
Perimeter/Circumference of Square: 20.00
Area of Square: 25.00
```

```
#5. Write a program that creates and uses a Time class to perform various time arithmeti
class Time:
    def __init__(self, hours=0, minutes=0, seconds=0):
        self.hours = hours
        self.minutes = minutes
        self.seconds = seconds
        self.normalize()

    def normalize(self):
        # Adjust the time to proper format
        if self.seconds >= 60:
            self.minutes += self.seconds // 60
            self.seconds = self.seconds % 60

        if self.minutes >= 60:
            self.hours += self.minutes // 60
            self.minutes = self.minutes % 60

    def add(self, other):
        new_hours = self.hours + other.hours
        new_minutes = self.minutes + other.minutes
        new_seconds = self.seconds + other.seconds
        return Time(new_hours, new_minutes, new_seconds)

    def subtract(self, other):
        # Convert both times into seconds
        total_self = self.hours * 3600 + self.minutes * 60 + self.seconds
        total_other = other.hours * 3600 + other.minutes * 60 + other.seconds
        if total_self < total_other:
            total_self, total_other = total_other, total_self  # Always positive result

        diff_seconds = total_self - total_other

        hours = diff_seconds // 3600
        minutes = (diff_seconds % 3600) // 60
        seconds = (diff_seconds % 3600) % 60

        return Time(hours, minutes, seconds)

    def display(self):
        print(f"{self.hours:02d}:{self.minutes:02d}:{self.seconds:02d}")

# Example usage
print("Enter first time:")
h1 = int(input("Hours: "))
m1 = int(input("Minutes: "))
```

```
s1 = int(input("Seconds: "))

print("\nEnter second time:")
h2 = int(input("Hours: "))
m2 = int(input("Minutes: "))
s2 = int(input("Seconds: "))

time1 = Time(h1, m1, s1)
time2 = Time(h2, m2, s2)

print("\nFirst Time:", end=' ')
time1.display()
print("Second Time:", end=' ')
time2.display()

# Perform addition
sum_time = time1.add(time2)
print("\nSum of Times:", end=' ')
sum_time.display()

# Perform subtraction
diff_time = time1.subtract(time2)
print("Difference of Times:", end=' ')
diff_time.display()
```

```
Enter first time:
Hours: 5
Minutes: 30
Seconds: 00

Enter second time:
Hours: 10
Minutes: 00
Seconds: 00

First Time: 05:30:00
Second Time: 10:00:00

Sum of Times: 15:30:00
Difference of Times: 04:30:00
```

```
#6. Write a program to create a class Date that has a list containing day, month and yea
#Define an overloaded == operator to compare two Date objects.

class Date:
    def __init__(self, day, month, year):
        self.date = [day, month, year]

    def __eq__(self, other):
        if isinstance(other, Date):
            return self.date == other.date
        return False

    def display(self):
        print(f"{self.date[0]:02d}/{self.date[1]:02d}/{self.date[2]}")

# Example usage
print("Enter first date:")
d1 = int(input("Day: "))
```

```
m1 = int(input("Month: "))
y1 = int(input("Year: "))

print("\nEnter second date:")
d2 = int(input("Day: "))
m2 = int(input("Month: "))
y2 = int(input("Year: "))

date1 = Date(d1, m1, y1)
date2 = Date(d2, m2, y2)

print("\nFirst Date:", end=' ')
date1.display()
print("Second Date:", end=' ')
date2.display()

# Compare the two dates
if date1 == date2:
    print("\nThe two dates are equal.")
else:
    print("\nThe two dates are not equal.")
```

```
Enter first date:
Day: 1
Month: 2
Year: 2025

Enter second date:
Day: 1
Month: 2
Year: 2030

First Date: 01/02/2025
Second Date: 01/02/2030

The two dates are not equal.
```

```
#7. Create a class Weather that has a list containing weather parameters.
#Define an overloaded in operator that checks whether an item is present in the list.
 #(Hint: define the function __contains__( )in a class.)

class Weather:
    def __init__(self, parameters):
        self.parameters = parameters

    def __contains__(self, item):
        return item in self.parameters

    def display(self):
        print("Weather parameters:", ", ".join(self.parameters))

# Example usage
weather_today = Weather(["Temperature", "Humidity", "Wind Speed", "Pressure", "Visibilit

weather_today.display()

param_to_check = input("\nEnter a parameter to check if it is present: ")

if param_to_check in weather_today:
```

```
    print(f"\nYes, '{param_to_check}' is present in the weather parameters.")
else:
    print(f"\nNo, '{param_to_check}' is NOT present in the weather parameters.")
```

```
Weather parameters: Temperature, Humidity, Wind Speed, Pressure, Visibility

Enter a parameter to check if it is present: Humidity

Yes, 'Humidity' is present in the weather parameters.
```

```
#8. Implement a String class containing the following functions:
#a. Overloaded += operator function to perform string concatenation
#b. Method toLower() to convert upper case letters to lower case.
#c. Method toUpper() to convert lower case letters to upper case.

class String:
    def __init__(self, content=""):
        self.content = content

    def __iadd__(self, other):
        if isinstance(other, String):
            self.content += other.content
        elif isinstance(other, str):
            self.content += other
        else:
            raise TypeError("Can only concatenate String or str types.")
        return self

    def toLower(self):
        lower_content = ""
        for ch in self.content:
            if 'A' <= ch <= 'Z':
                lower_content += chr(ord(ch) + 32)
            else:
                lower_content += ch
        self.content = lower_content

    def toUpper(self):
        upper_content = ""
        for ch in self.content:
            if 'a' <= ch <= 'z':
                upper_content += chr(ord(ch) - 32)
            else:
                upper_content += ch
        self.content = upper_content

    def __str__(self):
        return self.content

# Example usage
str1 = String("Hello")
str2 = String(" World")

print("Initial Strings:")
print(str1)
print(str2)
```

```
str1 += str2
print("\nAfter concatenation (+=")
print(str1)

str1.toLower()
print("\nAfter converting to lowercase:")
print(str1)

str1.toUpper()
print("\nAfter converting to uppercase:")
print(str1)
```

```
Initial Strings:
Hello
 World

After concatenation (+=
Hello World

After converting to lowercase:
hello world

After converting to uppercase:
HELLO WORLD
```