EXERCISE -!

CUSTOM OBJECT DESIGN

CODE:

```
class Calculator:
    def __init__(self):
        self.num1 = None
        self.num2 = None

    def set_num1(self):
        try:
            self.num1 = float(input("Enter the first number: "))
        except ValueError:
            print("Please provide a valid number.")

    def set_num2(self):
        try:
            self.num2 = float(input("Enter the second number: "))
        except ValueError:
            print("Please provide a valid number.")

    def add(self):
        return self.num1 + self.num2

    def sub(self):
        return self.num1 - self.num2

    def mul(self):
        return self.num1 * self.num2

# Sample usage
```

```
calculator = Calculator()


# Getting input from user
calculator.set_num1()
calculator.set_num2()


# Performing operations
print("Addition result:", calculator.add())
print("Subtraction result:", calculator.sub())
print("Multiplication result:", calculator.mul())
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-01.py
Enter the first number: 10.5
Enter the second number: 3.2
Addition result: 13.7
Subtraction result: 7.3
Multiplication result: 33.6
```

EXERCISE-2

i) Relationship between objects

A) INHERITANCE

CODE:

```
from abc import ABC, abstractmethod


# Base Pet Class
class Pet(ABC):
    def __init__(self, name, color, pet_type, cost):
        self.name = name
        self.color = color
        self.type = pet_type
        self.cost = cost

    def display_name(self):
        return self.name

    def display_color(self):
        return self.color

    def display_type(self):
        return self.type

    def display_cost(self):
        return self.cost

# Derived Dog Class
class Dog(Pet):
    def __init__(self, name, color, breed, cost):
        super().__init__(name, color, "dog", cost)
```

```
        self.breed = breed

    def display_breed(self):
        return self.breed

# Derived Cat Class
class Cat(Pet):
    def __init__(self, name, color, breed, cost):
        super().__init__(name, color, "cat", cost)
        self.breed = breed

    def display_breed(self):
        return self.breed

# Function to create a Dog
def create_dog():
    name = input("Enter dog's name: ")
    color = input("Enter dog's color: ")
    breed = input("Enter dog's breed: ")
    cost = int(input("Enter dog's cost: "))
    return Dog(name, color, breed, cost)

# Function to create a Cat
def create_cat():
    name = input("Enter cat's name: ")
    color = input("Enter cat's color: ")
    breed = input("Enter cat's breed: ")
    cost = int(input("Enter cat's cost: "))
    return Cat(name, color, breed, cost)

# Main function to get input and display pet details
```

```
def main():
    pet_type = input("Do you want to add a Dog or a Cat? ").lower()
    if pet_type == 'dog':
        dog = create_dog()
        print(f"\nDog Details:")
        print(f"Name: {dog.display_name()}")
        print(f"Color: {dog.display_color()}")
        print(f"Breed: {dog.display_breed()}")
        print(f"Cost: {dog.display_cost()}")
    elif pet_type == 'cat':
        cat = create_cat()
        print(f"\nCat Details:")
        print(f"Name: {cat.display_name()}")
        print(f"Color: {cat.display_color()}")
        print(f"Breed: {cat.display_breed()}")
        print(f"Cost: {cat.display_cost()}")
    else:
        print("Invalid pet type. Please enter either 'Dog' or 'Cat'.")
# Call the main function
if __name__ == "__main__":
    main()
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-02-i.py
Do you want to add a Dog or a Cat? cat
Enter cat's name: kitty
Enter cat's color: black
Enter cat's breed: Ragdoll
Enter cat's cost: 10000

Cat Details:
Name: kitty
Color: black
Breed: Ragdoll
Cost: 10000
```

B) COMPOSITION

CODE:

```
class Page:
    def __init__(self, pg_number: int, contents: str):
        self.pg_number = pg_number
        self.contents = contents

    def get_content(self) -> str:
        return self.contents

class Document:
    def __init__(self, title: str, author: str):
        self.title = title
        self.author = author
        self.pages = []

    def add_page(self, pg_number: int, contents: str) -> None:
        page = Page(pg_number, contents)
        self.pages.append(page)
        print(f"Page {pg_number} added to the document.")

    def remove_page(self, pg_number: int) -> None:
        self.pages = [page for page in self.pages if page.pg_number != pg_number]
        print(f"Page {pg_number} removed from the document.")

    def get_page_contents(self, pg_number: int) -> str:
        for page in self.pages:
            if page.pg_number == pg_number:
                return page.get_content()
        return "Page not found"
```

```
# Example usage
doc = Document("My Document", "John Doe")
print(f"Document '{doc.title}' by {doc.author} created.")

# Add pages to the document
doc.add_page(1, "This is the content of page 1.")
doc.add_page(2, "This is the content of page 2.")
doc.add_page(3, "This is the content of page 3.")

# Access content of a specific page
print(f"Contents of Page 1: {doc.get_page_contents(1)}")
print(f"Contents of Page 2: {doc.get_page_contents(2)}")

# Remove a page from the document
doc.remove_page(2)

# Try accessing the removed page
print(f"Contents of Page 2: {doc.get_page_contents(2)}")

# Check remaining pages
print(f"Contents of Page 1: {doc.get_page_contents(1)}")
print(f"Contents of Page 3: {doc.get_page_contents(3)}")
```

OUTPUT :

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-02/ex-02-ib.py
Document 'My Document' by John Doe created.
Page 1 added to the document.
Page 2 added to the document.
Page 3 added to the document.
Contents of Page 1: This is the content of page 1.
Contents of Page 2: This is the content of page 2.
Page 2 removed from the document.
Contents of Page 2: Page not found
Contents of Page 1: This is the content of page 1.
Contents of Page 3: This is the content of page 3.
```

C) AGGREGATION

CODE:

```
# Part (Book) class
class Book:
    # Can exist independently of the whole
    def __init__(self, title):
        self.title = title

    def get_title(self):
        return self.title

# Whole (Library) Contains Books
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def show_books(self):
        for book in self.books:
            print(book.get_title())

# Main function to test aggregation
def main():
    # Creating books
    book1 = Book("To Kill a Mockingbird")
    book2 = Book("1984")

    # Creating a library
```

```
    library = Library()

    # Adding books to the library
    library.add_book(book1)
    library.add_book(book2)

    # Displaying books in the library
    library.show_books()

if __name__ == "__main__":
    main()
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-02/ex-02-ic.py
To Kill a Mockingbird
1984
```

EXERCISE-2

ii) Relationship between classes

CODE:

```
import math

# 1. Point Class
class Point:
    def __init__(self, x, y):
        self.xcod = x
        self.ycod = y

    def getpoint(self):
        return self.xcod, self.ycod

    def showpoint(self):
        print(f'{self.xcod},{self.ycod}')

# 2a. Circle Class
class Circle:
    def __init__(self, r=0):
        self.radius = r
        self.center = None
        self.point = None
        self.area = None

    def getcircle(self):
        while True:
            try:
                # Taking input and splitting based on comma
                a = input('Enter center co-ordinates(x,y): ').strip().split(',')
```

```
            b = input('Enter point co-ordinates(x,y): ').strip().split(',')

            # Ensure that both inputs are in the correct format
            if len(a) != 2 or len(b) != 2:
                raise ValueError("Please enter two comma-separated values.")

            # Converting string values to float
            self.center = Point(float(a[0]), float(a[1]))
            self.point = Point(float(b[0]), float(b[1]))

            # Calculate radius and area
            self.radius = ((self.center.xcod - self.point.xcod) ** 2 +
                           (self.center.ycod - self.point.ycod) ** 2) ** 0.5
            self.area = 3.1415 * (self.radius) ** 2
            print(f"Radius: {self.radius}")
            break  # Exit the loop after successful input and calculation

        except ValueError as e:
            print(f"Error: {e}. Please try again.")  # Show error and retry input

    def getradius(self):
        return self.radius

    def getarea(self):
        return self.area

# 2b. Cone Class (Inherits from Circle)
class Cone(Circle):
    def __init__(self):
        super().__init__()
        self.apex = None
```

```
        self.volume = None

    def getcone(self):
        self.getcircle()
        while True:
            try:
                # Taking apex input and splitting it
                apex_x, apex_y = map(float, input("Enter the apex of the cone (x, y): ").strip().split(','))
                self.apex = Point(apex_x, apex_y)

                # Calculate height
                height = math.sqrt((self.apex.xcod - self.center.xcod) ** 2 +
                                   (self.apex.ycod - self.center.ycod) ** 2)

                # Calculate volume
                self.volume = (1/3) * 3.1415 * (self.radius ** 2) * height
                break  # Exit the loop after successful input and calculation
            except ValueError:
                print("Error: Please enter valid numbers for the apex coordinates.")

    def getvolume(self):
        return self.volume

# 3a. Regular_Polygon Class
class Regular_Polygon:
    def __init__(self):
        self.lop = []
        self.num_sides = 0

    def getdetails(self):
        self.num_sides = int(input("Enter the number of sides: "))
```

```
        for i in range(self.num_sides):
            while True:
                try:
                    x, y = map(float, input(f"Enter point {i+1} co-ordinates (x, y): ").strip().split(','))
                    self.lop.append(Point(x, y))
                    break
                except ValueError:
                    print("Error: Please enter valid coordinates (x, y) in numeric form.")

# 3b. Square Class (Inherits from Regular_Polygon)
class Square(Regular_Polygon):
    def __init__(self):
        super().__init__()

    def calculate_area(self):
        side_length = math.sqrt((self.lop[0].xcod - self.lop[1].xcod) ** 2 +
                                (self.lop[0].ycod - self.lop[1].ycod) ** 2)
        return side_length ** 2

    def calculate_perimeter(self):
        side_length = math.sqrt((self.lop[0].xcod - self.lop[1].xcod) ** 2 +
                                (self.lop[0].ycod - self.lop[1].ycod) ** 2)
        return 4 * side_length

    def calculate_volume(self):
        return 0

# Testing the classes
cone = Cone()
cone.getcone()
print("Volume of the cone: ", cone.getvolume())
```

```
print()


square = Square()

square.getdetails()

print("Area of the square: ", square.calculate_area())

print("Perimeter of the square: ", square.calculate_perimeter())

print("Volume of the square: ", square.calculate_volume())
```

OUTPUT :

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-02/ex-02-ii.py
Enter center co-ordinates(x,y): 2.5
Enter point co-ordinates(x,y): 4.0
Error: Please enter two comma-separated values.. Please try again.
Enter center co-ordinates(x,y): 2.5,4.0
Enter point co-ordinates(x,y): 4.0,6.0
Radius: 2.5
Enter the apex of the cone (x, y): 2.5,8.0
Volume of the cone:  26.179166666666664

Enter the number of sides: 4
Enter point 1 co-ordinates (x, y): 0.0,0.0
Enter point 2 co-ordinates (x, y):  2.0,0.0
Enter point 3 co-ordinates (x, y): 2.0,2.0
Enter point 4 co-ordinates (x, y): 0.0,2.0
Area of the square:  4.0
Perimeter of the square:  8.0
Volume of the square:  0
```

EXERCISE -3

MODULES AND PACKAGES

1)

CODE:

api.py

```
import requests
from datetime import datetime

class DateTimeApi:
    def __init__(self):
        self.timeIndia = requests.get('http://worldtimeapi.org/api/timezone/Asia/Kolkata')
        self.timeIndia = self.timeIndia.json()
        self.onlydate = self.timeIndia['datetime'][0:10]
        self.onlydateformated = datetime.strptime(self.onlydate, '%Y-%m-%d')
        self.onlytime = self.timeIndia['datetime'][11:19]
```

convert.py

```
class Convert:
    def convert_hrs_days(self, hours):
        return hours / 24

    def convert_days_hours(self, days):
        return days * 24

    def convert_man_hrs_days(self, hours):
        return hours / 8
```

Date.py

```
from datetime import datetime
```

```
from api import DateTimeApi

class Date(DateTimeApi):
    def __init__(self):
        super().__init__()

    def currentDate(self):
        '''returns current date in dd-mm-yyyy format'''
        return self.onlydateformated.strftime("%d-%m-%Y")

    def currentDateFormatMDY(self):
        '''returns current date in mm-dd-yyyy format'''
        return self.onlydateformated.strftime("%m-%d-%Y")

    def currentTime(self):
        '''returns current time in hh:mm:ss format'''
        return self.onlytime

    def createDate(self, year, month, day):
        '''returns date in dd-mm-yyyy format'''
        return datetime(year, month, day).strftime("%d-%m-%Y")

    def currentDatestr(self):
        '''returns current date in "Month Day, Year" format'''
        return datetime.strptime(self.onlydate, '%Y-%m-%d').strftime("%B %d, %Y")
```

difference.py

```
from api import DateTimeApi
from datetime import datetime, timedelta

class Difference(DateTimeApi):
```

```
    def __init__(self):
        super().__init__()

    def difference_with_current(self, date):
        date = datetime.strptime(date, '%Y-%m-%d')
        return (date - self.onlydateformated).days

    def difference(self, date1, date2):
        date1 = datetime.strptime(date1, '%Y-%m-%d')
        date2 = datetime.strptime(date2, '%Y-%m-%d')
        return (date2 - date1).days

    def days_after(self, days):
        return (self.onlydateformated + timedelta(days=days)).strftime("%Y-%m-%d")

    def days_before(self, days):
        return (self.onlydateformated - timedelta(days=days)).strftime("%Y-%m-%d")

    def month_after(self, months):
        return (self.onlydateformated + timedelta(days=months*30)).strftime('%Y-%m-%d')

    def month_before(self, months):
        return (self.onlydateformated - timedelta(days=months*30)).strftime('%Y-%m-%d')
```

Validity.py

```
from datetime import datetime

class Validity:
    def is_valid_time(self, time):
        try:
            datetime.strptime(time, '%H:%M:%S')
```

```
        return True
    except ValueError:
        return False

def is_valid_date(self, date):
    try:
        datetime.strptime(date, '%Y-%m-%d')
        return True
    except ValueError:
        return False
```

main.py

```
from Difference import Difference
from Validity import Validity
from Convert import Convert
from Date import Date

# Date functionalities
date = Date()
print(date.currentDate()) # Output: Current date in dd-mm-yyyy format
print(date.currentDateFormatMDY()) # Output: Current date in mm-dd-yyyy format
print(date.currentTime()) # Output: Current time in hh:mm:ss format
print(date.currentDatestr()) # Output: Current date in 'Month Day, Year' format

# Conversion functionalities
convert = Convert()
print(convert.convert_hrs_days(48)) # Output: 2.0 (days)
print(convert.convert_days_hours(2)) # Output: 48 (hours)
print(convert.convert_man_hrs_days(16)) # Output: 2.0 (man-days)

# Validity check functionalities
```

```
validity = Validity()

print(validity.is_valid_time('12:00:00')) # Output: True

print(validity.is_valid_date('2024-08-28')) # Output: True


# Date difference functionalities

difference = Difference()

print(difference.difference_with_current('2024-08-28')) # Output: Number of days between current date and '2024-08-28'

print(difference.difference('2024-08-28', '2024-08-30')) # Output: 2

print(difference.days_after(2)) # Output: Date 2 days after current date

print(difference.days_before(2)) # Output: Date 2 days before current date

print(difference.month_after(1)) # Output: Date 1 month after current date

print(difference.month_before(1)) # Output: Date 1 month before current date
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-03/Q1/Main.py
01-10-2024
10-01-2024
21:07:18
October 01, 2024
2.0
48
2.0
True
True
-34
2
2024-10-03
2024-09-29
2024-10-31
2024-09-01
```

2)

Centroid.py

```
# Centroid.py
class Centroid:
    def __init__(self, *coordinates):
        self.coordinates = coordinates

    def centroid_distance(self, other):
        if len(self.coordinates) == len(other.coordinates):
            square = 0
            for a, b in zip(self.coordinates, other.coordinates):
                diff = a - b  # Using the __sub__ method of Vector
                square += sum(coord ** 2 for coord in diff.coordinates)
            return square ** 0.5
        else:
            raise ValueError("Centroids must have the same number of dimensions")
```

Complex.py

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

    def complex_distance(self, other):
        return (((self.real - other.real) ** 2 + (self.imaginary - other.imaginary) ** 2) ** 0.5)
```

Number.py

```
class Number:
    def __init__(self, value):
        self.value = value
```

```
    def number_distance(self, other):
        return abs(self.value - other.value)
```

String.py

```
class String:
    def __init__(self, value):
        self.value = value

    def string_distance(self, other):
        if len(self.value) < len(other.value):
            return String.string_distance(other, self)

        if len(other.value) == 0:
            return len(self.value)

        min_length = min(len(self.value), len(other.value))
        diff_count = sum(1 for i in range(min_length) if self.value[i] != other.value[i])
        length_diff = abs(len(self.value) - len(other.value))
        return diff_count + length_diff
```

Vector.py

```
class Vector:
    def __init__(self, *coordinates):
        self.coordinates = coordinates

    def __sub__(self, other):
        if len(self.coordinates) == len(other.coordinates):
            return Vector(*(a - b for a, b in zip(self.coordinates, other.coordinates)))
        else:
            raise ValueError("Vectors must have the same number of dimensions")
```

```
    def __str__(self):
        return str(self.coordinates)

    def vector_distance(self, other):
        if len(self.coordinates) == len(other.coordinates):
            square = sum((a - b) ** 2 for a, b in zip(self.coordinates, other.coordinates))
            return square ** 0.5
        else:
            raise ValueError("Vectors must have the same number of dimensions")
```

Main.py

```
import Centroid
import Complex
import Number
import String
import Vector

def distance(obj1, obj2):
    if isinstance(obj1, Centroid.Centroid) and isinstance(obj2, Centroid.Centroid):
        return obj1.centroid_distance(obj2)
    elif isinstance(obj1, Complex.Complex) and isinstance(obj2, Complex.Complex):
        return obj1.complex_distance(obj2)
    elif isinstance(obj1, Number.Number) and isinstance(obj2, Number.Number):
        return obj1.number_distance(obj2)
    elif isinstance(obj1, Vector.Vector) and isinstance(obj2, Vector.Vector):
        return obj1.vector_distance(obj2)
    elif isinstance(obj1, String.String) and isinstance(obj2, String.String):
        return obj1.string_distance(obj2)
    else:
        raise TypeError("Incompatible types for distance calculation")
com1 = Complex.Complex(4, 3)
```

```
com2 = Complex.Complex(1, 0)
com_diff = distance(com1, com2)
print("Complex Difference:", com_diff)

n1 = Number.Number(5)
n2 = Number.Number(12)
n_diff = distance(n1, n2)
print("Number Difference:", n_diff)

v1 = Vector.Vector(1, 7, 5, 2)
v2 = Vector.Vector(3, 1, 4, 7)
v_diff = distance(v1, v2)
print("Vector Difference:", v_diff)

c1 = Centroid.Centroid(v1, v2)
c2 = Centroid.Centroid(v2, v1)
c_diff = distance(c1, c2)
print("Centroid Difference:", c_diff)

s1 = String.String("Hello")
s2 = String.String("Hi")
s_diff = distance(s1, s2)
print("String Difference:", s_diff)
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-03/Q2/Main.py
Complex Difference: 4.242640687119285
Number Difference: 7
Vector Difference: 8.12403840463596
Centroid Difference: 11.489125293076057
String Difference: 4
```

EXERCISE-4

ABSTRACT CLASS (TREE)

CODE:

Binary_Tree Folder :

node.py

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

Binarytree.py

```
from abc import ABC, abstractmethod
from .Node import Node

class BinaryTree(ABC):
    def __init__(self):
        self.root = None

    @abstractmethod
    def create_tree(self, elements):
        pass

    @abstractmethod
    def insert(self, value):
        pass

    @abstractmethod
    def traverse(self, order):
        pass
```

BinarysearchTree.py

```
from .BinaryTree import BinaryTree
from .Node import Node

class BinarySearchTree(BinaryTree):
    def create_tree(self, elements):
        for element in elements:
            self.insert(element)

    def insert(self, value):
        if self.root is None:
            self.root = Node(value)
        else:
            self._insert_recursive(self.root, value)

    def _insert_recursive(self, current_node, value):
        if value < current_node.value:
            if current_node.left is None:
                current_node.left = Node(value)
            else:
                self._insert_recursive(current_node.left, value)
        else:
            if current_node.right is None:
                current_node.right = Node(value)
            else:
                self._insert_recursive(current_node.right, value)

    def traverse(self, order):
        if order == 'inorder':
            return self._inorder_traversal(self.root, [])
```

```
        elif order == 'preorder':
            return self._preorder_traversal(self.root, [])
        elif order == 'postorder':
            return self._postorder_traversal(self.root, [])
        else:
            raise ValueError("Traversal order must be 'inorder', 'preorder', or 'postorder'")

    def _inorder_traversal(self, node, result):
        if node:
            self._inorder_traversal(node.left, result)
            result.append(node.value)
            self._inorder_traversal(node.right, result)
        return result

    def _preorder_traversal(self, node, result):
        if node:
            result.append(node.value)
            self._preorder_traversal(node.left, result)
            self._preorder_traversal(node.right, result)
        return result

    def _postorder_traversal(self, node, result):
        if node:
            self._postorder_traversal(node.left, result)
            self._postorder_traversal(node.right, result)
            result.append(node.value)
        return result
```

OUTSIDE FOLDER

App.py

```
from BinaryTree.BinarySearchTree import BinarySearchTree
```

```
class NewsApplication:
    def __init__(self):
        self.political_news_tree = BinarySearchTree()
        self.sports_news_tree = BinarySearchTree()

    def add_news(self, category, date, news):
        if category == 'Political':
            self.political_news_tree.insert((date, news))
        elif category == 'Sports':
            self.sports_news_tree.insert((date, news))
        else:
            raise ValueError("Category must be 'Political' or 'Sports'")

    def display_news_on_date(self, date, category):
        if category == 'Political':
            tree = self.political_news_tree
        elif category == 'Sports':
            tree = self.sports_news_tree
        else:
            raise ValueError("Category must be 'Political' or 'Sports'")

        news_list = tree.traverse('inorder')
        for news_date, news in news_list:
            if news_date == date:
                print(news)

    def display_news_between_dates(self, start_date, end_date, category):
        if category == 'Political':
            tree = self.political_news_tree
        elif category == 'Sports':
```

```
            tree = self.sports_news_tree
        else:
            raise ValueError("Category must be 'Political' or 'Sports'")

        news_list = tree.traverse('inorder')
        for news_date, news in news_list:
            if start_date <= news_date <= end_date:
                print(news)

# Example usage
if __name__ == "__main__":
    app = NewsApplication()
    app.add_news('Political', '2024-09-01', 'Political news 1')
    app.add_news('Sports', '2024-09-02', 'Sports news 1')
    app.add_news('Political', '2024-09-03', 'Political news 2')
    app.add_news('Sports', '2024-09-04', 'Sports news 2')

    print("News on 2024-09-01 (Political):")
    app.display_news_on_date('2024-09-01', 'Political')

    print("\nNews between 2024-09-01 and 2024-09-03 (Political):")
    app.display_news_between_dates('2024-09-01', '2024-09-03', 'Political')
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-04/App.py
News on 2024-09-01 (Political):
Political news 1

News between 2024-09-01 and 2024-09-03 (Political):
Political news 1
Political news 2
```

EXERCISE - 5

POLYMORPHISM

i)

```
class Vector(list):
    def __init__(self, *args):
        # Ensure that only integers are allowed
        for arg in args:
            if not isinstance(arg, int):
                raise TypeError(f"Only integers are allowed. '{arg}' is not an integer.")
        super().__init__(args)

    def append(self, value):
        if not isinstance(value, int):
            raise TypeError(f"Only integers are allowed. Invalid value: {value}")
        super().append(value)

    def insert(self, index, value):
        if not isinstance(value, int):
            raise TypeError(f"Only integers are allowed. Invalid value: {value}")
        super().insert(index, value)

    def __setitem__(self, index, value):
        if not isinstance(value, int):
            raise TypeError(f"Only integers are allowed. Invalid value: {value}")
        super().__setitem__(index, value)

    def __add__(self, other):
        if isinstance(other, Vector):
            # Merge and remove duplicates
            merged_vector = sorted(set(list(self) + list(other)))
            return Vector(*merged_vector)
```

```
        else:
            raise TypeError("Addition is supported only between Vector objects.”)

    def __sub__(self, other):
        if isinstance(other, Vector):
            # Compute symmetric difference
            difference_vector = sorted(list(set(self).symmetric_difference(set(other))))
            return Vector(*difference_vector)
        else:
            raise TypeError("Subtraction is supported only between Vector objects.")

class EmptyVectorError(Exception):
    """Custom exception for empty vectors."""
    pass

def GetRatios(Vec1, Vec2):
    # Check if both vectors are empty
    if len(Vec1) == 0 and len(Vec2) == 0:
        raise EmptyVectorError("Both vectors are empty.")

    # Check if the vectors have different sizes
    if len(Vec1) != len(Vec2):
        raise ValueError("Vectors must be of the same size.")

    Ratio = []
    for i in range(len(Vec1)):
        try:
            if Vec2[i] == 0:
                Ratio.append('NaN')  # Handle division by zero
            else:
                Ratio.append(Vec1[i] / Vec2[i])
```

```
        except ZeroDivisionError:
            Ratio.append('NaN')  # Just a safeguard for zero division
        except Exception as e:
            raise e
    return Ratio

# Example Usage:
V1 = Vector(2, 3, 4)
V2 = Vector(1, 2)

# Add two vectors
V3 = V1 + V2
print(f"V1 + V2 = {V3}")

# Subtract two vectors
V4 = V1 - V2
print(f"V1 - V2 = {V4}")

# Get ratios
V5 = Vector(10, 20, 30)
V6 = Vector(2, 0, 5)
ratios = GetRatios(V5, V6)
print(f"Ratios: {ratios}")
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-05/Q1.py
V1 + V2 = [1, 2, 3, 4]
V1 - V2 = [1, 3, 4]
Ratios: [5.0, 'NaN', 6.0]
```

ii)

```
class Employee:
    # Constructor to initialize first and last names (Instance members)
    def __init__(self, first_name, last_name):
        self.first_name = first_name  # Instance member
        self.last_name = last_name      # Instance member

    @classmethod
    def from_string(cls, name_string):
        first_name, last_name = name_string.split(' ')
        return cls(first_name, last_name)  # Return a new Employee object

emp1 = Employee('John', 'Doe')
print(f"Employee 1: First Name: {emp1.first_name}, Last Name: {emp1.last_name}")
# Output: Employee 1: First Name: John, Last Name: Doe

emp2 = Employee.from_string('Seetha Raman')
print(f"Employee 2: First Name: {emp2.first_name}, Last Name: {emp2.last_name}")
# Output: Employee 2: First Name: Seetha, Last Name: Raman
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-05/Q2.py
Employee 1: First Name: John, Last Name: Doe
Employee 2: First Name: Seetha, Last Name: Raman
```

iii)

```
class Movie:
    def __init__(self, name, genre):
        self.name = name
        self.genre = genre

    def __str__(self):
        return f"Movie(Name: {self.name}, Genre: {self.genre})

class MovieList(list):
    def __init__(self, genre):
        self.genre = genre
        super().__init__()  # Inherit the initialization from list

    def append(self, movie):
        """Overrides the append method to ensure only movies of the same genre are added."""
        if not isinstance(movie, Movie):
            raise TypeError("Only Movie objects can be added to MovieList.")

        if movie.genre != self.genre:
            raise ValueError(f"Movie genre does not match. Expected genre: {self.genre}, but got:
{movie.genre}.")
        # Append the movie to the list if it matches the genre
        super().append(movie)

    def __add__(self, other):
        """Overload the + operator to return the list with more movies."""
        if not isinstance(other, MovieList):
            raise TypeError("Can only add MovieList objects.")
        return self if len(self) >= len(other) else other
```

```
    def __str__(self):
        """Override string representation to show the movie list."""
        movie_names = [movie.name for movie in self]
        return f"MovieList(Genre: {self.genre}, Movies: {', '.join(movie_names)})"

movie1 = Movie("Inception", "thriller")
movie2 = Movie("The Dark Knight", "thriller")
movie3 = Movie("Interstellar", "sci-fi")
movie4 = Movie("Memento", "thriller")
movie5 = Movie("Blade Runner", “sci-fi”)

thriller_list = MovieList("thriller")
sci_fi_list = MovieList(“sci-fi”)

thriller_list.append(movie1)  # Added
thriller_list.append(movie2)  # Added
thriller_list.append(movie4)  # Added
sci_fi_list.append(movie3)    # Added
sci_fi_list.append(movie5)    # Added

print(thriller_list)
print(sci_fi_list)
result_list = thriller_list + sci_fi_list
print("List with more movies:", result_list)
```

OUTPUT:

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-05/Q3.py
MovieList(Genre: thriller, Movies: Inception, The Dark Knight, Memento)
MovieList(Genre: sci-fi, Movies: Interstellar, Blade Runner)
List with more movies: MovieList(Genre: thriller, Movies: Inception, The Dark Kni
ght, Memento)
```

EXERCISE - 5a (even question)

```
def Average(*args):
    if len(args) == 0:
        return "Length should be more than one."
    total = sum(args)
    leng = len(args)
    return total / leng
print(Average(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))  # Output: 5.5

# 2. Using **kwargs
def ShowOff(**kwargs):
    # Check if 'mark' is present in the keyword arguments
    if 'mark' in kwargs:
        # Get the value of 'mark'
        mark = kwargs['mark']
        # Check if 'mark' is greater than 60
        if mark > 60:
            # Get the value of 'name' if it exists
            name = kwargs.get('name', 'Unknown')
            # Return the name and marks
            return f"Name: {name}, Marks: {mark}"
        else:
            name = kwargs.get('name', 'Unknown')
            sub_name = kwargs.get('sub_name')
            teacher_name = kwargs.get('teacher_name')
            return f'''
Name: {name}
Subject Name: {sub_name}
Teacher Name: {teacher_name}
Marks are 60 or below'''
```

```
    else:
        return "Marks are not provided"


# Example usage for ShowOff
print(ShowOff(name="Alice", mark=75))  # Output: Name: Alice, Marks: 75
print(ShowOff(name="Bob", mark=50, sub_name='Maths', teacher_name='Bob'))  # Output: Marks are 60 or below
print(ShowOff(name="Charlie"))  # Output: Marks are not provided
```

OUTPUT :

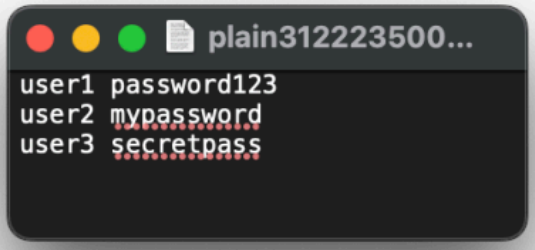
```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-05a.py
5.5
Name: Alice, Marks: 75

Name: Bob
Subject Name: Maths
Teacher Name: Bob
Marks are 60 or below
Marks are not provided
```

EXERCISE-6

STRINGS AND SERIALISATION

1)ENCODING AND DECODING

plain3122235002072.txt



Ex-6.py

```
class EncryptDecrypt:
    def __init__(self, reg_num, shift=3):
        self.reg_num = reg_num
        self.shift = shift
        self.plain_file = f"plain{reg_num}.txt"
        self.crypt_file = f"crypt{reg_num}.txt"
        self.reconstructed_file = f"reconstructed{reg_num}.txt"

    def encrypt_password(self, password):
        encrypted = ""
        for char in password:
            encrypted += chr((ord(char) + self.shift) % 128)  # Shift character by
'shift'
        return encrypted

    def decrypt_password(self, encrypted_password):
        decrypted = ""
        for char in encrypted_password:
            decrypted += chr((ord(char) - self.shift) % 128)  # Reverse the shift
        return decrypted

    def encrypt_file(self):
        with open(self.plain_file, "r") as plain_file:
            lines = plain_file.readlines()

        encrypted_lines = []
        for line in lines:
```

```
            username, password = line.strip().split()
            encrypted_password = self.encrypt_password(password)
            encrypted_lines.append(f"{username} {encrypted_password}\n")

        # Write encrypted data to crypt file
        with open(self.crypt_file, "w") as crypt_file:
            crypt_file.writelines(encrypted_lines)

    def decrypt_file(self):
        with open(self.crypt_file, "r") as crypt_file:
            encrypted_lines = crypt_file.readlines()

        decrypted_lines = []
        for line in encrypted_lines:
            username, encrypted_password = line.strip().split()
            decrypted_password =
self.decrypt_password(encrypted_password)
            decrypted_lines.append(f"{username} {decrypted_password}\n")

        # Write decrypted data to reconstructed file
        with open(self.reconstructed_file, "w") as reconstructed_file:
            reconstructed_file.writelines(decrypted_lines)

    def compare_files(self):
        with open(self.plain_file, "r") as plain_file:
            original_lines = plain_file.readlines()

        with open(self.reconstructed_file, "r") as reconstructed_file:
            reconstructed_lines = reconstructed_file.readlines()

        match_count = 0
        total_count = len(original_lines)

        for orig_line, recon_line in zip(original_lines, reconstructed_lines):
            if orig_line.strip() == recon_line.strip():
                match_count += 1

        correctness_percentage = (match_count / total_count) * 100
        print(f"Correctness: {correctness_percentage}%")

    def process(self):
        self.encrypt_file()  # Encrypt plain file and create crypt file
        self.decrypt_file()  # Decrypt crypt file and create reconstructed file
        self.compare_files()  # Compare plain file and reconstructed file

# Instantiate the class and process
```

```
reg_num = "3122235002072"
enc_dec = EncryptDecrypt(reg_num)
enc_dec.process()
```

crypt3122235002072.txt

crypt3122...

```
user1 sdvvzrug456
user2 p|sdvvzrug
user3 vhfuhwsdvv
```

reconstructed3122235002072.txt

reconstructe...

```
user1 password123
user2 mypassword
user3 secretpass
```

OUTPUT :

```
============= RESTART: /Users/mohit_reddy/Documents/ex-06/ex-6.py =============
Correctness: 100.0%
>>>
```

2) ENCRYPTION AND DECRYPTION

stock.txt

stock.txt

Company	Rate	Date	Time
HCL	1125.00	28.11.2022	10.12 AM
Infosys	1640.25	28.11.2022	10.13 AM
Wipro	404.85	28.11.2022	10.14 AM

stock_greeter.py

```
# Part 1: Greeting in Native Language Script
def greet_user():
    name = input("What's your name? ")

    # Dictionary of greetings in various languages in their native scripts
    greetings = {
        'tamil': 'வணக்கம்',        # Vanakkam
        'hindi': 'नमस्ते',         # Namaste
        'arabic': 'مرحبا',          # Marhaba
        'japanese': 'こんにちは',    # Konnichiwa
        'chinese': '你好',          # Nǐ hǎo
        'russian': 'Привет',        # Privet
        'french': 'Bonjour',        # Bonjour
        'german': 'Hallo',          # Hallo
        'spanish': 'Hola',          # Hola
        'english': 'Hello'          # English
    }

    language = input("Which language do you prefer? ").lower()
    # Use the greeting if it exists, else default to 'Hello' in English
    greeting = greetings.get(language, 'Hello')
```

```
    print(f"{greeting}, {name}!")  # Display the greeting


# Part 2: Stock Information Display
class Stock:
    def __init__(self, company, rate, date, time):
        self.company = company
        self.rate = rate
        self.date = date
        self.time = time

    def display(self):
        print(f"Company: {self.company}, Rate: {self.rate}, Date: {self.date},
Time: {self.time}")


def display_stock_information():
    try:
        with open('stock.txt', 'r') as file:
            next(file)  # Skip the header line
            print("Details:\n")
            for line in file:
                data = line.strip().split()
                company, rate, date, time = data[0], data[1], data[2], data[3]
                stock = Stock(company, rate, date, time)
                stock.display()
    except FileNotFoundError:
        print("Error: stock.txt file not found!")


# Run the greeting function and display stock information
greet_user()
display_stock_information()
```

OUTPUT :

```
========= RESTART: /Users/mohit_reddy/Documents/ex-06/stock_greeter.py =========
What's your name? Mohit R
Which language do you prefer? Tamil
வணக்கம், Mohit R!
Details:

Company: HCL, Rate: 1125.00, Date: 28.11.2022, Time: 10.12
Company: Infosys, Rate: 1640.25, Date: 28.11.2022, Time: 10.13
Company: Wipro, Rate: 404.85, Date: 28.11.2022, Time: 10.14
```

EXERCISE-7

Reg Ex

```
import re


# Class for password validation
class PasswordValidator:
    def validate(self, password):
        """
        Validates the password based on the following criteria:
        - At least 8 characters long
        - Contains at least one uppercase letter
        - Contains at least one lowercase letter
        - Contains at least one digit
        - Contains at least one special character (@$!%*?&)
        """
        pattern = r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]
{8,}$'
        if re.match(pattern, password):
            return "Valid Password."
        else:
            return "Invalid Password."


# Class for variable name validation
class VariableValidator:
    def validate(self, variable):
        """
        Validates variable names:
        - Must start with a letter or underscore
```

```
        - Can contain letters, digits, and underscores
        """
        pattern = r'^[A-Za-z_][A-Za-z0-9_]*$'
        if re.match(pattern, variable):
            return "Valid"
        else:
            return "Invalid"

# Class for binary number validation
class BinaryNumberValidator:
    def validate(self, number):
        """
        Validates if the input string is a binary number.
        Only '0' and '1' are allowed.
        """
        pattern = r'^[01]+$'
        if re.match(pattern, number):
            return "Valid"
        else:
            return "Invalid"

# Class for sentence validation
class SentenceValidator:
    # Defined verbs, subjects, and objects for sentence structure
    verbs = ["cut", "cuts", "sing", "sings", "dance", "dances", "fell", "falls", "beat", "beats",
"ate", "eats", "drink", "drinks"]
    subjects = ["He", "She", "People", "boys", "girls", "Ram", "Mohan", "A child", "Milk"]
    objects = ["tree", "Kavin", "Lollypop", "milk", "cat", "the tree", "the milk"]
```

```
    def validate(self, sentence):
        """
        Validates if the sentence follows the pattern:
        Subject + Verb + Object + '.'
        """
        subject_pattern = r'|'.join(re.escape(subject) for subject in SentenceValidator.subjects)
        verb_pattern = r'|'.join(re.escape(verb) for verb in SentenceValidator.verbs)
        object_pattern = r'|'.join(re.escape(obj) for obj in SentenceValidator.objects)

        # Sentence pattern: Subject Verb Object.
        pattern = rf'^({subject_pattern})\s+({verb_pattern})\s+({object_pattern})\.$'
        if re.match(pattern, sentence, re.IGNORECASE):
            return "Valid"
        else:
            return "Invalid"

# Testing the validators
if __name__ == "__main__":
    # 1. Password validation
    password = input("Enter the password: ")
    password_validator = PasswordValidator()
    print(password_validator.validate(password))

    # 2. Variable validation
    variable = input("Enter the variable name: ")
    variable_validator = VariableValidator()
    print(variable_validator.validate(variable))
```

```
# 3. Binary number validation
binary_number = input("Enter the binary number: ")
binary_validator = BinaryNumberValidator()
print(binary_validator.validate(binary_number))

# 4. Sentence validation with predefined test cases
test_sentences = [
    "Ram Cuts the tree.",    # Valid
    "Mohan beat Kavin.",     # Invalid
    "A child ate Lollypop.", # Valid
    "Drink milk cat.",       # Invalid
    "Milk drinks cat."       # Valid (Syntactically Valid)
]

sentence_validator = SentenceValidator()
for sentence in test_sentences:
    print(f'{sentence}: {sentence_validator.validate(sentence)}')
```

OUTPUT :

```
= RESTART: /Users/mohit_reddy/Library/Mobile Documents/com~apple~CloudDocs/sem_3/
design pattern/designPatternLab/ex-07.py
Enter the password: @Password321
Valid Password.
Enter the variable name: golf
Valid
Enter the binary number: 0301
Invalid
Ram Cuts the tree.: Valid
Mohan beat Kavin.: Valid
A child ate Lollypop.: Valid
Drink milk cat.: Invalid
Milk drinks cat.: Valid
```