SSN College of Engineering

Department of Information Technology

UIT2201 — Programming and Data Structures

2022 – 2023

Exercise — 03

U. Pranaav | IT-B | 3122225002093

PART-A

1.

I. AIM:

To create a Vector class that can create a vector object on which arithmetic operations such as addition, subtraction, scalar multiplication can be performed. The Vector class creates a vector object of a sequence if a sequence is passed, or creates a Vector object of a certain input length comprised of 0s.

II. ALGORITHM:

1. Start the algorithm:
2. Define a class called Vector.
3. Inside the class, define an init method that takes a single argument, val.
4. In the init method, use an if-elif-else block to check if the type of val is either an integer or a list.
5. If val is an integer, set the instance variable dimension to val, and set the instance variable list to a list of zeros with length val.
6. If val is a list, set the instance variable dimension to the length of the list, and set the instance variable list to the list.
7. If val is neither an integer nor a list, raise a TypeError with the message "Enter only an integer or list."
8. Define a len method that returns the instance variable dimension.
9. Define a getitem method that takes an index as an argument and returns the value of the list at that index.
10. Define a setitem method that takes an index and a value as arguments and sets the value of the list at that index to the given value.
11. Define an add method that takes another Vector object as an argument and returns a new Vector object whose values are the sum of the corresponding values of the two vectors. Raise a ValueError if the dimensions of the two vectors are different.
12. Define a sub method that takes another Vector object as an argument and returns a new Vector object whose values are the difference of the corresponding values of the two vectors. Raise a ValueError if the dimensions of the two vectors are different.

13. Define a mul method that takes another Vector object as an argument and returns a new Vector object whose values are the product of the corresponding values of the two vectors. Raise a ValueError if the dimensions of the two vectors are different.
14. Define a truediv method that takes another Vector object as an argument and returns a new Vector object whose values are the quotient of the corresponding values of the two vectors. Raise a ValueError if the dimensions of the two vectors are different. Raise a ZeroDivisionError if any of the elements of the other Vector object are 0.
15. Define a str method that returns the string representation of the list instance variable.
16. End of algorithm.

III. CODE:

```
# -*- coding: utf-8 -*-


'''
This module provides a class used for creating a Vector class
Object. This is a part of the excercises given under the course
UIT2201 (Programming and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 19 2023

Revised on Wed Apr 22 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

'''


import random


#making a class for creating a Vector
class Vector:

    '''
    The given class stores the coordinates of a point and
    performs functions such as finding distance between
    two points adding two points as well as subtracting
    two points.

    The input data is not modified in any way and there are
    no side effects.

    methods:
        __init__: the constructor

        __setitem__: for setting a certain value at an index

        __getitem__: for getting a value at an index

        __len__: for finding the length of matrix
```

```
        __iter__: to create an iterable object

        __next__: to iterate through the iterable object

        __add__: for using the '+' operation on class objects

        __sub__: for using the '-' operation on class objects

        __mul__: for using the '*' operation on class objects

        __str__: for displaying class objects in human readable
        form.

    '''

    def __init__(self,val):

        '''
        The constructor takes in two arguments and creates a
     Vector object based on the input.

        Input is not modified in any way and there are no side
        effects.

        args:
            self: the object
            val: the integer or collection using which Vector
            object is created

        Returns:
         Vector dimension variable dim and initialized Vector.

        '''

        if not isinstance(val,(int,float)):
            self.vec = val
            self.dim = len(val)
        else:
            if isinstance(val,int):
                self.vec = [0]*val
                self.dim = val
            else:
                raise ValueError("Cannot enter a float value")


    def __setitem__(self,index,val):

        '''
        Allows the Vector objects to set a value at a certain
        index.

        args:
            self: the object
            index: position at which value needs to be added
            val: the value to be added

        Returns:
            None

        '''

        self.vec[index] = val
```

```
def __getitem__(self,index):

    '''
    Allows the Vector objects to get a value at a certain
    index.

    args:
        self: the object
        index: position at which value needs to be returned

    Returns:
        Value at given index.

    '''

    return self.vec[index]


def __len__(self):

    '''
    Returns the length of the Vector object.

    args:
        self: the object

    Returns:
        Length of Vector object.

    '''
    return self.dim


def __iter__(self):

    '''
    Creates an iterator object that can be used to
    iterate over.

    args:
        self: the object

    Returns:
        The object instance as an iterator object.

    '''

    self.current_index = 0
    return self


def __next__(self):

    '''
    Returns the next item in the iteration sequence.

    Raises:
        StopIteration: If there are no more items to iterate over.

    args:
```

```
            self: the object

        Returns:
            The next item in the iteration sequence.

        '''

        if self.current_index >= len(self.vec):
            raise StopIteration

        else:
            current_element = self.vec[self.current_index]
            self.current_index += 1
            return current_element


    def __add__(self,other):

        '''
        This function takes in two Vectors as input and calculates
        the sum of Vectors and returns a Vector object.

        The input is not modified and there are no side effects.

        args:
            self: first object
            other: second object

        Returns:
            The sum of two Vectors as a Vector object.

        '''

        if self.dim != other.dim:
            raise ValueError("Dimensions do not match")

        sum_vec = Vector(len(self))

        for i in range(len(self)):
            sum_vec[i] = (self[i]+other[i])

        return sum_vec


    def __sub__(self,other):

        '''
        This function takes in two Vectors as input and calculates
        the difference between the Vectors and returns a Vector
        object.

        The input is not modified and there are no side effects.

        args:
            self: first object
            other: second object

        Returns:
            The difference between the two Vectors as a Vector
            object.

        '''
```

```
        if self.dim != other.dim:
            raise ValueError("Dimensions do not match")

        sub_vec = Vector(len(self))

        for i in range(len(self)):
            sub_vec[i] = (self[i]-other[i])

        return sub_vec


    def __mul__(self,other):

        '''
        This function takes in two Vectors as input and calculates
        the product of the Vectors and returns a Vector object.

        The input is not modified and there are no side effects.

        args:
            self: first object
            other: second object

        Returns:
            The product of the two Vectors as a Vector object.

        '''

        if self.dim != other.dim:
            raise ValueError("Dimensions do not match")

        mul_vec = Vector(len(self))

        for i in range(len(self)):
            mul_vec[i] = (self[i]*other[i])

        scalar_prod = 0

        for i in mul_vec:
            scalar_prod += i

        return scalar_prod


    def __str__(self):

        '''
        This function takes in only the object and returns the
        given object as a str data type.

        The input is not modified in any way and there are no side
        effects

        args:
            self: the object to be displayed

        Returns:
            An object of the str data type.

        '''
```

```
        vec_final = '<'
        for i in range(len(self)):
            vec_final += str(self[i])
            if i != (len(self)-1):
                vec_final += ','
        return vec_final+'>'
#end of class Vector


#driver code
if __name__ == '__main__':
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    #generating two random vectors
    vec_1 = Vector(5)
    vec_2 = Vector(5)

    for i in range(5):
        vec_1[i] = random.randint(-1000,1000)

    for i in range(5):
        vec_2[i] = random.randint(-1000,1000)

    print(f"Vector 1 is: \n{vec_1}\nVector 2 is: \n{vec_2}\n")

    print("Sum of vectors is:\n",vec_1 + vec_2)
    print()  #for spacing between lines

    print("Difference of vectors is:\n",vec_1 - vec_2)
    print()

    print("Scalar product of vectors is:\n",vec_1 * vec_2)
    print()

    print(f"Length of vector 1 \n{vec_1}\n is: ",len(vec_1))
    print()

    print(f"Length of vector 2 \n{vec_2}\n is: ",len(vec_2))
    print()
```

III. OUTPUT:

Vector 1 is:
<953,-499,413,332,-320>
Vector 2 is:
<714,-444,-835,-390,-796>

Sum of vectors is:
<1667,-943,-422,-58,-1116>

Difference of vectors is:
<239,-55,1248,722,476>

Scalar product of vectors is:

682383

Length of vector 1
<953,-499,413,332,-320>
is: 5

Length of vector 2
<714,-444,-835,-390,-796>
is: 5

2.

I. AIM:

To create a Matrix class that creats a matrix object on which arithmetic operations such as addition, subtraction, scalar multiplication as well as finding its determinant and transpose. The Matrix object can be initialized using 2 integers for number of rows and columns and sets each element to a value of 0.

III. ALGORITHM:

1. Start the algorithm.
2. Import the required modules: random and vectorclass.
3. Define a class Matrix with the following methods:
   a. **__init__()** method to initialize the instance variables of the Matrix class.
   b. **__getitem__()** method to get the value of a specific index in a matrix.
   c. **__setitem__()** method to assign a value to a particular index of a matrix.
   d. **__add__()** method to add two matrices.
   e. **__sub__()** method to subtract two matrices.
   f. **__mul__()** method to multiply two matrices.
   g. **__str__()** method to print matrix object.
   h. **transpose()** method to return the transpose of a matrix.
   i. **det()** method to find determinant of a matrix.
4. Define the **det()** method inside the Matrix class.
   a. Check if the matrix is square, if not raise a ValueError with the message "The matrix must be square."
   b. If the matrix is 2x2, calculate and return the determinant using the formula prod1-prod2.
   c. If the matrix is larger than 2x2, use recursion to calculate the determinant by creating submatrices and computing their determinants.
   d. Return the final determinant value.
5. Define the **transpose()** method inside the Matrix class.
   a. Create a new matrix object with the number of rows and columns reversed from the original matrix.
   b. Iterate through the original matrix and assign the values to the new matrix with the indices swapped.
   c. Return the new matrix object.
6. Define the **__mul__()** method inside the Matrix class.

a. Check if the number of columns in the first matrix is equal to the number of rows in the second matrix, if not raise a ValueError with the message "The number of columns in the first matrix must be equal to the number of rows in the second matrix."
b. Create a new matrix object with the number of rows from the first matrix and the number of columns from the second matrix.
c. Use nested for loops to iterate through the rows of the first matrix and the columns of the second matrix.
d. Calculate the dot product of the row from the first matrix and the column from the second matrix using a nested for loop.
e. Assign the dot product to the corresponding index in the new matrix.
f. Return the new matrix object.

7. Define the **__add__()** method inside the Matrix class.
   a. Check if the dimensions of the two matrices are the same, if not raise a ValueError with the message "Incorrect dimensions".
   b. Create a new matrix object with the same dimensions as the two matrices.
   c. Use nested for loops to iterate through the rows and columns of the matrices.
   d. Add the corresponding elements of the two matrices and assign the sum to the corresponding index in the new matrix.
   e. Return the new matrix object.
8. Define test cases for the Matrix class: a. Create two matrices of 3x3. b. Assign random values to each row and col of the two matrices. c. Add the two matrices and print the result. d. Subtract the two matrices and print the result. e. Multiply the two matrices and print the result. f. Find the determinant of the first matrix and print the result. g. Find the transpose of the first matrix and print the result.
9. Execute the test cases if the file is not imported.
10. End of algorithm

III. CODE:

```
# -*- coding: utf-8 -*-


'''
This module provides a class used for creating a Matrix by
importing a module with a class Vector for creating vectors.
This is a part of the excercises given under the course
UIT2201 (Programming and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 19 2023

Revised on Wed Apr 22 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

'''
```

```python
from vector import Vector
import random


#making a class for creating a matrix
class Matrix:

    '''
    The given class stores a matrix with each row as a Vector
    object and performs functions such as addition, multiplication
    subtraction, finding determinant, finding transpose.

    The input data is not modified in any way and there are
    no side effects.

    methods:
        __init__: the constructor

        __setitem__: for setting a certain value at an index

        __getitem__: for getting a value at an index

        __len__: for finding the length of matrix

        __iter__: to create an iterable object

        __next__: to iterate through the iterable object

        __str__: for displaying class objects in human readable
        form.

        __add__: for using the '+' operation on class objects

        __sub__: for using the '-' operation on class objects

        __mul__: for using the '*' operation on class objects

        det: for finding the determinant of the matrix object

        transpose: for finding the transpose of the matrix object

    '''

    def __init__(self,r,c):
        self.row = r
        self.col = c
        self.mat =  [Vector(r) for x in range(c)]


    def __setitem__(self,index,val):

        '''
        Allows the matrix objects to set a value at a certain
        index.

        The input is not modified and there are no side effects.

        args:
            self: the object
            index: position at which value needs to be added
            val: the value to be added
```

```
        Returns:
            None

        '''

        self.mat[index[0]][index[1]] = val


    def __getitem__(self,index):

        '''
        Allows the matrix objects to get a value at a certain
        index.

        The input is not modified and there are no side effects.


        args:
            self: the object
            index: position at which value needs to be returned

        Returns:
            Value at given index.

        '''

        try:
            return self.mat[index[0]][index[1]]
        except Exception:
            return self.mat[index]


    def __len__(self):

        '''
        Returns the length of the matrix object.

        The input is not modified and there are no side effects.

        args:
            self: the object

        Returns:
            Length of matrix object.

        '''

        return self.col


    def __iter__(self):

        '''
        Creates an iterator object that can be used to
        iterate over.

        args:
            self: the object

        Returns:
            The object instance as an iterator object.
```

```
        '''

        self.current_index = 0
        return self


    def __next__(self):

        '''
        Returns the next item in the iteration sequence.

        Raises:
            StopIteration: If there are no more items to iterate over.

        args:
            self: the object

        Returns:
            The next item in the iteration sequence.

        '''

        if self.current_index >= len(self):
            raise StopIteration
        else:
            current_element = self.mat[self.current_index]
            self.current_index += 1
            return current_element


    def __str__(self):

        '''
        This function takes in only the object and returns the
        given object as a str data type.

        The input is not modified in any way and there are no side
        effects

        args:
            self: the object to be displayed

        Returns:
            An object of the str data type.

        '''

        to_return = '['
        for i in range(len(self)):
            for j in range(self.row):
                to_return += str(self[i,j])
                if j != (self.row-1):
                    to_return += ','

            if i!= (self.col-1):
                to_return += '\n'
        to_return += ']'

        return to_return


    def __add__(self,other):
```

```
        '''
        This function takes in two matrices as input and calculates
        the sum of matrices and returns a matrix object.

        The input is not modified and there are no side effects.

        args:
            self: first object
            other: second object

        Returns:
            The sum of two matrices as a matrix object.

        '''

        if len(self) != len(other) or self.row != other.row:
            raise ValueError("Incorrect dimensions")

        sum_mat = Matrix(self.row, self.col)

        for i in range(len(self)):
            for j in range(self.row):
                sum_mat[i,j] = (self[i,j]+other[i,j])

        return sum_mat


    def __sub__(self,other):

        '''
        This function takes in two matrices as input and calculates
        the difference between the matrices and returns a matrix
        object.

        The input is not modified and there are no side effects.

        args:
            self: first object
            other: second object

        Returns:
            The difference between the two matrices as a matrix
            object.

        '''

        if len(self) != len(other) or self.row != other.row:
            raise ValueError("Incorrect dimensions")

        sub_mat = Matrix(self.row, self.col)

        for i in range(len(self)):
            for j in range(self.row):
                sub_mat[i,j] = (self[i,j]-other[i,j])

        return sub_mat


    def __mul__(self,other):

        '''
```

```
        This function takes in two matrices as input and calculates
        the product of the matrices and returns a matrix object.

        The input is not modified and there are no side effects.

        args:
            self: first object
            other: second object

        Returns:
            The product of the two matrices as a matrix object.

        '''

        if self.col != other.row:
            raise ValueError("The number of columns in the first matrix must be equal to the number of row in the
second matrix.")
        result = Matrix(self.row, other.col)
        for i in range(result.row):
            for j in range(result.col):
                dot_product = 0
                for k in range(self.col):
                    dot_product += self[i,k] * other[k,j]
                result[i,j] = dot_product
        return result


    def det(self):

        '''
        This function takes in a matrix as input and calculates
        the determinant of the given matrix.

        The input is not modified and there are no side effects.

        args:
            self: the object

        Returns:
            The determinant of the given matrix.

        '''

        if self.row != self.col:
            raise ValueError("The matrix must be square.")

        if self.row == 2:
            prod1 = self[0,0]*self[1,1]
            prod2 = self[0,1]*self[1,0]
            return prod1-prod2

        det_val = 0
        for i in range(self.col):
            sub_matrix = Matrix(self.row-1, self.col-1)
            for j in range(1, self.row):
                for k in range(self.col):
                    if k < i:
                        sub_matrix[j-1, k] = self[j, k]
                    elif k > i:
                        sub_matrix[j-1, k-1] = self[j, k]

            det_val += ((-1)**i) * self[0, i] * sub_matrix.det()
```

```
        return det_val


    def transpose(self):

        '''
        This function takes in a matrix as input and returns
        the transpose of the given matrix.

        The input is not modified and there are no side effects.

        args:
            self: the object

        Returns:
            The transpose of the given matrix.

        '''

        trans_mat = Matrix(len(self),len(self[0]))
        for i in range(len(self[0])):
            for j in range(len(self)):
                trans_mat[i,j] = self[j,i]

        return trans_mat
#end of class matrix


#driver code
if __name__ == '__main__':
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    #generating two random matrices
    mat_1 = Matrix(4,4)
    mat_2 = Matrix(4,4)

    for i in range(4):
        for j in range(4):
            mat_1[i,j] = random.randint(-1000,1000)

    for i in range(4):
        for j in range(4):
            mat_2[i,j] = random.randint(-1000,1000)

    print(f"Matrix 1 is: \n{mat_1}\n\nMatrix 2 is: \n{mat_2}\n")

    print("Sum of matrices is:\n",mat_1 + mat_2)
    print()

    print("Difference of matrices is:\n",mat_1 - mat_2)
    print()

    print("Product of matrices is:\n",mat_1 * mat_2)
    print()

    print(f"Determinant of matrix 1 \n{mat_1}\n\n is :\n\n",mat_1.det())
    print()

    print(f"Determinant of matrix 2 \n{mat_2}\n\n is :\n\n",mat_2.det())
    print()
```

```
print(f"Transpose of matrix 1 \n{mat_1}\n\n is :\n\n",mat_1.transpose())
print()

print(f"Transpose of matrix 2 \n{mat_2}\n\n is :\n\n",mat_2.transpose())
print()
```

III. OUTPUT:

Matrix 1 is:
[427,257,-103,936
264,839,559,-558
-378,-124,-887,611
-77,103,-752,532]

Matrix 2 is:
[408,841,501,-128
495,-946,-10,689
778,-533,-559,577
9,-57,-675,-486]

Sum of matrices is:
[835,1098,398,808
759,-107,549,131
400,-657,-1446,1188
-68,46,-1427,46]

Difference of matrices is:
[19,-584,-604,1064
-231,1785,569,-1247
-1156,409,-328,34
-86,160,-77,1018]

Product of matrices is:
[229721,117532,-362866,-391910
952897,-837811,188043,1138010
-900191,237350,-104730,-845797
-560699,208297,21661,-611633]

Determinant of matrix 1
[427,257,-103,936
264,839,559,-558
-378,-124,-887,611
-77,103,-752,532]

is :

128762988245

Determinant of matrix 2
[408,841,501,-128
495,-946,-10,689
778,-533,-559,577
9,-57,-675,-486]

is :

-297510785616

Transpose of matrix 1
[427,257,-103,936
264,839,559,-558
-378,-124,-887,611
-77,103,-752,532]

is :

[427,264,-378,-77
257,839,-124,103
-103,559,-887,-752
936,-558,611,532]

Transpose of matrix 2
[408,841,501,-128
495,-946,-10,689
778,-533,-559,577
9,-57,-675,-486]

is :

[408,495,778,9
841,-946,-533,-57
501,-10,-559,-675
-128,689,577,-486]