NAME-DEBAGNI BHATTACHARJEE ROLL NO-04

ENROLMENT NO-12023006015004 SEC-A

DEPARTMENT-COMPUTER APPLICATION(MCA)

1) **Write a function called check-season, it takes a month parameter and returns the season: Autumn, Winter, Spring or Summer.**

```
def findseason  (M) :
        list1 = [[12 , 1 , 2], [3 , 4 , 5],
                        [6 , 7 , 8], [9 , 10 , 11]]
        if M in list1[0] :
                print ( "WINTER" )
        elif M in list1[1] :
                print ( "SPRING" )
        elif M in list1[2] :
                print ( "SUMMER" )
        elif M in list1[3] :
                print ( "AUTUMN" )
        else :
                print ( "Invalid Month Number" )
M = 5
print("For Month number:", M);
findseason  ( M )
M = 10
print("For Month number:", M);
findseason  ( M )
```

<u>OUTPUT-</u>

```
For Month number: 5
SPRING
For Month number: 10
AUTUMN
```

2) **Write a function called calculate_slope which return the slope of a linear equation.**

```
def calculate_slope(x1, y1, x2, y2):
   if x1 == x2:
      return "Slope is undefined (vertical line)."
   slope = (y2 - y1) / (x2 - x1)
   return slope
x1, y1 = 1, 2
x2, y2 = 3, 6
slope = calculate_slope(x1, y1, x2, y2)
print(f"The slope of the line through points ({x1}, {y1}) and ({x2}, {y2}) is: {slope}")
```

OUTPUT-

```
The slope of the line through points (1, 2) and (3, 6) is: 2.0
```

3) Quadratic equation is calculated as follows: $ax^2 + bx + c = 0$.
Write a function which calculates solution set of a quadratic equation, _solve_quadratic_eqn_.

```
import cmath
def solve_quadratic_eqn(a, b, c):
    discriminant = b**2 - 4*a*c
    sol1 = (-b + cmath.sqrt(discriminant)) / (2*a)
    sol2 = (-b - cmath.sqrt(discriminant)) / (2*a)
    return sol1, sol2
a, b, c = 1, -3, 2
solutions = solve_quadratic_eqn(a, b, c)
print(f"The solutions of the quadratic equation {a}x² + {b}x + {c} = 0 are: {solutions[0]} and {solutions[1]}")
```

OUTPUT-

```
The solutions of the quadratic equation 1x² + -3x + 2 = 0 are: (2+0j) and (1+0j)
```

4) Declare a function named print_list. It takes a list as a parameter and it prints out each element of the list.

```
def print_list(input_list):
    for element in input_list:
        print(element)
example_list = [1, 2, 3, 4, 5]
print_list(example_list)
```

OUTPUT-

```
1
2
3
4
5
```

5) Declare a function named reverse_list. It takes an array as a parameter and it returns the reverse of the array (use loops).

```
def reverse_list(input_list):
    reversed_list = []
    for i in range(len(input_list) - 1, -1, -1):
        reversed_list.append(input_list[i])
    return reversed_list
example_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(example_list)
print(f"Original list: {example_list}")
print(f"Reversed list: {reversed_list}")
```

OUTPUT-

```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

6) Compute the sum up to n terms in the series

1 - 1/2 + 1/3 - 1/4 + 1/5 -... 1/n where n is a positive integer and input by user.

```
def compute_series_sum(n):
    sum = 0.0
    for i in range(1, n + 1):
        if i % 2 == 0:
            sum -= 1 / i
        else:
            sum += 1 / i
    return sum
try:
    n = int(input("Enter a positive integer n: "))
    if n <= 0:
        print("Please enter a positive integer.")
    else:
        series_sum = compute_series_sum(n)
        print(f"The sum of the series up to {n} terms is: {series_sum}")
except ValueError:
    print("Invalid input! Please enter a positive integer.")
```

OUTPUT-

```
Enter a positive integer n: 14

The sum of the series up to 14 terms is: 0.6587051837051838
```

7. Write a program to compute sin x for given x. The user should supply x and a positive integer n. We compute the sine of x using the series and the computation should use all terms in the series up through the term involving xn

sin x = x - x3/3! + x5/5! - x7/7! + x9/9! ....

```
import math

def factorial(n):

   if n == 0:

      return 1

   else:

      return n * factorial(n-1)

def compute_sin(x, n):

   sin_x = 0.0

   for i in range(n + 1):

      term = ((-1)**i * x**(2*i + 1)) / factorial(2*i + 1)

      sin_x += term

   return sin_x

try:

   x = float(input("Enter the value of x (in radians): "))

   n = int(input("Enter a positive integer n: "))

   if n < 0:

      print("Please enter a positive integer for n.")

   else:

      sin_x = compute_sin(x, n)

      print(f"The computed value of sin({x}) using the series up to the term involving x^{2*n + 1} is: {sin_x}")

      print(f"The actual value of sin({x}) using math.sin is: {math.sin(x)}")

except ValueError:
    print("Invalid input! Please enter valid numbers.")
```

OUTPUT-

```
Enter the value of x (in radians): 45
Enter a positive integer n: 24
The computed value of sin(45.0) using the series up to the term involving x^49 is:
7.54896732923731e+17
The actual value of sin(45.0) using math.sin is: 0.8509035245341184
```

8) Write a program to compute cosine of x. The user should supply x and a positive integer n. We compute the cosine of x using the series and the computation should use all terms in the series up through the term involving xn

cos x = 1 - x2/2! + x4/4! - x6/6! ....

```
import math
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
def compute_cos(x, n):
    cos_x = 0.0
    for i in range(n + 1):
        term = ((-1)**i * x**(2*i)) / factorial(2*i)
        cos_x += term
    return cos_x
try:
    x = float(input("Enter the value of x (in radians): "))
    n = int(input("Enter a positive integer n: "))
    if n < 0:
        print("Please enter a positive integer for n.")
    else:
        cos_x = compute_cos(x, n)
        print(f"The computed value of cos({x}) using the series up to the term involving x^{2*n} is: {cos_x}")
        print(f"The actual value of cos({x}) using math.cos is: {math.cos(x)}")
except ValueError:
    print("Invalid input! Please enter valid numbers.")
```

<u>OUTPUT-</u>

```
Enter a positive integer n: 12
The computed value of cos(90.0) using the series up to the term involving x^24 is: 1.2027840699335446e+23
The actual value of cos(90.0) using math.cos is: -0.4480736161291701
```

9) Print the pattern upto N Lines:

```
.
/_\          ./\
              /___\   .   /\
                           /\
                            /_____\


N=2     N=3     N=4
```

```
def print_pattern(N):
    if N < 1:
        print("Please enter a positive integer for N.")
        return
    for i in range(1, N + 1):
        print(" " * (N - i) + ".")
        for j in range(1, i + 1):
            print(" " * (N - i) + "/" + " " * (2 * j - 1) + "\\")
        print(" " * (N - i) + "/" + "_" * (2 * i - 1) + "\\")
try:
    N = int(input("Enter a positive integer N: "))
    print_pattern(N)
```

```
except ValueError:
    print("Invalid input! Please enter a positive integer.")
```

OUTPUT-

```
Enter a positive integer N: 2
 .
 /\        .
 /_\       /\
          /  \
         /___\
```

10. Print a number as a 8 segment display N Lines:

```
_
_|
|_
_
_|
_|
|_|
|
N=2            N=3            N=4
```

```
def print_segment(number, N):
    segments = {
        '0': [' _ ', '| |', '|_|'],
        '1': ['   ', '  |', '  |'],
        '2': [' _ ', ' _|', '|_ '],
        '3': [' _ ', ' _|', ' _|'],
        '4': ['   ', '|_|', '  |'],
        '5': [' _ ', '|_ ', ' _|'],
        '6': [' _ ', '|_ ', '|_|'],
        '7': [' _ ', '  |', '  |'],
        '8': [' _ ', '|_|', '|_|'],
        '9': [' _ ', '|_|', ' _|']
    }
    if N == 2:
        scale = {
            ' _ ': ' _ ',
            '|_|': '_|',
            '| |': '|_',
            '  |': ' |',
            ' _|': ' _|',
            '|_ ': '|_ ',
            '  |': '  |',
            '   ': '   '
        }
        segments = {key: [scale[seg] for seg in value] for key, value in segments.items()}
    elif N == 3:
        scale = {
            ' _ ': ' _ ',
            '|_|': ' _|',
            '| |': '|_ ',
            '  |': '  |',
            ' _|': ' _|',
            '|_ ': '|_ ',
            '  |': '  |',
            '   ': '   '
        }
        segments = {key: [scale[seg] for seg in value] for key, value in segments.items()}
    elif N == 4:
        # For N=4, the same segments are used, but you can adjust scaling if needed
        segments = {
            '0': [' _ ', '| |', '|_|'],
            '1': ['   ', '  |', '  |'],
            '2': [' _ ', ' _|', '|_ '],
            '3': [' _ ', ' _|', ' _|'],
            '4': ['   ', '|_|', '  |'],
            '5': [' _ ', '|_ ', ' _|'],
            '6': [' _ ', '|_ ', '|_|'],
            '7': [' _ ', '  |', '  |'],
            '8': [' _ ', '|_|', '|_|'],
            '9': [' _ ', '|_|', ' _|']
        }
    num_str = str(number)
    lines = ['' for _ in range(N)]
    for digit in num_str:
        seg = segments[digit]
        for i in range(N):
            lines[i] += seg[i] + ' '
    for line in lines:
        print(line)
N = int(input("Enter the number of lines (2, 3, or 4): "))
number = input("Enter the number to display: ")
print_segment(number, N)
```

<u>OUTPUT-</u>

```
Enter the number of lines (2, 3, or 4): 3
Enter the number to display: 3     _
                                   _|
                                    _|
```

11. Print the pattern upto N lines:

1 2

4 3

1 2 3

8 9 4

7 6 5

1 2 3 4

12 13 14 5

11 16 15 6

10 9 8 7

N=2 N=3 N=4

```
def print_spiral(n):
    matrix = [[0] * n for _ in range(n)]
    num = 1
    top, bottom, left, right = 0, n - 1, 0, n - 1
    while top <= bottom and left <= right:
        for i in range(left, right + 1):
            matrix[top][i] = num
            num += 1
        top += 1
        for i in range(top, bottom + 1):
            matrix[i][right] = num
            num += 1
        right -= 1
        for i in range(right, left - 1, -1):
            matrix[bottom][i] = num
            num += 1
        bottom -= 1
        for i in range(bottom, top - 1, -1):
            matrix[i][left] = num
            num += 1
        left += 1
    for row in matrix:
        print(' '.join(map(str, row)))
N = int(input("Enter the number of lines (N): "))
print_spiral(N)
```

<u>OUTPUT-</u>

```
Enter the number of lines (N): 3
1 2 3
8 9 4
7 6 5
```

12. Write a python script that displays the following table

```
1 1 1 1 1
2 1 2 4 8
3 1 3 9 27
4 1 4 16 64
5 1 5 25 125
```

```
def display_table(rows, cols):
    header = [''] + [str(i) for i in range(1, cols + 1)]
    print(' '.join(header))
    for i in range(1, rows + 1):
        row = [str(i)]  # Start with the row number
        for j in range(1, cols + 1):
            value = i ** j
            row.append(str(value))
        print(' '.join(row))
```

```
rows = 5
cols = 5
display_table(rows, cols)
```

OUTPUT-

```
1 2 3 4 5
1 1 1 1 1 1
2 2 4 8 16 32
3 3 9 27 81 243
4 4 16 64 256 1024
5 5 25 125 625 3125
```

MISCELLANEOUS

13) Write a Python program to calculate Sum & Average of an integer array.

```
def calculate_sum_and_average(arr):
    total_sum = sum(arr)
    if len(arr) > 0:
        average = total_sum / len(arr)
    else:
        average = 0
    return total_sum, average
if __name__ == "__main__":
    input_str = input("Enter integers separated by spaces: ")
    try:
        num_list = [int(x) for x in input_str.split()]
        total_sum, average = calculate_sum_and_average(num_list)
        print(f"Sum: {total_sum}")
        print(f"Average: {average:.2f}")
    except ValueError:
        print("Invalid input. Please enter only integers.")
```

OUTPUT-

```
Enter integers separated by spaces: 10 20 30 40 50

Sum: 150

Average: 30.00
```

14) Write a Python program to implement stack using array.

```
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, value):
        self.stack.append(value)
        print(f"Pushed {value} onto the stack.")
    def pop(self):
        if not self.is_empty():
            value = self.stack.pop()
            print(f"Popped {value} from the stack.")
            return value
        else:
            print("Stack is empty, cannot pop.")
            return None
    def peek(self):
        if not self.is_empty():
            value = self.stack[-1]
            print(f"Top item is {value}.")
            return value
        else:
            print("Stack is empty.")
            return None
    def is_empty(self):
        return len(self.stack) == 0
    def display(self):
        if not self.is_empty():
            print("Current stack:", self.stack)
        else:
            print("Stack is empty.”)
if __name__ == "__main__":
    stack = Stack()
    stack.push(10)
    stack.push(20)
    stack.push(30)
    stack.display()
    stack.peek()
    stack.pop()
    stack.display()
    stack.peek()
    stack.pop()
    stack.pop()
    stack.pop()
```

OUTPUT-

```
Pushed 10 onto the stack.
Pushed 20 onto the stack.
Pushed 30 onto the stack.
Current stack: [10, 20, 30]
Top item is 30.
Popped 30 from the stack.
Current stack: [10, 20]
Top item is 20.
Popped 20 from the stack.
Popped 10 from the stack.
Stack is empty, cannot pop.
```

15) Write a Python program to implement Queue using array.

```
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self, value):
        self.queue.append(value)
        print(f"Enqueued {value}.")
    def dequeue(self):
        if not self.is_empty():
            value = self.queue.pop(0)
            print(f"Dequeued {value}.")
            return value
        else:
            print("Queue is empty, cannot dequeue.")
            return None
    def peek(self):
        if not self.is_empty():
            value = self.queue[0]
            print(f"Front item is {value}.")
            return value
        else:
            print("Queue is empty.")
            return None
    def is_empty(self):
        return len(self.queue) == 0
    def display(self):
        if not self.is_empty():
            print("Current queue:", self.queue)
        else:
```

```
        print("Queue is empty”)
if __name__ == "__main__":
    queue = Queue()
    queue.enqueue(10)
    queue.enqueue(20)
    queue.enqueue(30)
    queue.display()
    queue.peek()
    queue.dequeue()
    queue.display()
    queue.peek()
    queue.dequeue()
    queue.dequeue()
    queue.dequeue()
```

OUTPUT-

```
Enqueued 10.
Enqueued 20.
Enqueued 30.
Current queue: [10, 20, 30]
Front item is 10.
Dequeued 10.
Current queue: [20, 30]
Front item is 20.
Dequeued 20.
Dequeued 30.
Queue is empty, cannot dequeue.
```

16) Write a Python program to calculate Sum of two 2-dimensional arrays.

```
def add_matrices(matrix1, matrix2):
    if len(matrix1) != len(matrix2) or any(len(row1) != len(row2) for row1, row2 in zip(matrix1, matrix2)):
        raise ValueError("Matrices must have the same dimensions.")
    result = []
    for row1, row2 in zip(matrix1, matrix2):
        result_row = [elem1 + elem2 for elem1, elem2 in zip(row1, row2)]
        result.append(result_row)
    return result
def print_matrix(matrix):
    for row in matrix:
        print(' '.join(map(str, row)))
if __name__ == "__main__":
    matrix1 = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    matrix2 = [
        [9, 8, 7],
        [6, 5, 4],
        [3, 2, 1]
    ]
    try:
        sum_matrix = add_matrices(matrix1, matrix2)
        print("Sum of the matrices:")
        print_matrix(sum_matrix)
    except ValueError as e:
        print(e)
```

OUTPUT-

```
Sum of the matrices:
10 10 10
```

```
10 10 10
10 10 10
```

17) Write a Python program to find the range of a 1D array.

```python
def find_range(arr):
    if not arr:
        raise ValueError("Array is empty.")
    min_value = min(arr)
    max_value = max(arr)
    range_value = max_value - min_value
    return range_value
if __name__ == "__main__":
    array = [5, 3, 9, 1, 6, 7]
    try:
        range_value = find_range(array)
        print(f"The range of the array is: {range_value}")
    except ValueError as e:
        print(e)
```

OUTPUT-

```
The range of the array is: 8
```

18) Write a Python program to search an element in an array.

```python
def linear_search(arr, target):
    for index, value in enumerate(arr):
        if value == target:
            return index
    return -1
if __name__ == "__main__":
    array = [10, 23, 4, 56, 12, 78, 34]
    target = int(input("Enter the element to search for:))
    index = linear_search(array, target)
    if index != -1:
        print(f"Element {target} found at index {index}.")
    else:
        print(f"Element {target} not found in the array.")
```

<u>OUTPUT-</u>

```
Enter the element to search for: 56
Element 56 found at index 3.
```

19) Write a Python program to find the sum of even numbers in an integer array.

```
def sum_of_even_numbers(arr):
    total_sum = 0
    for number in arr:
        if number % 2 == 0:
            total_sum += number
    return total_sum
if __name__ == "__main__":
    array = [10, 23, 4, 56, 12, 78, 34]
    even_sum = sum_of_even_numbers(array)
    print(f"The sum of even numbers in the array is: {even_sum}")
```

<u>OUTPUT-</u>

```
The sum of even numbers in the array is: 180
```

20) Write a Python program to find the sum of diagonal elements in a 2D array.

```
def sum_of_diagonals(matrix):
    n = len(matrix)
    if any(len(row) != n for row in matrix):
        raise ValueError("Matrix must be square.")
    primary_diagonal_sum = 0
    secondary_diagonal_sum = 0
    for i in range(n):
        primary_diagonal_sum += matrix[i][i]
        secondary_diagonal_sum += matrix[i][n - 1 - i]
    return primary_diagonal_sum, secondary_diagonal_sum
def print_matrix(matrix):
    for row in matrix:
        print(' '.join(map(str, row)))
if __name__ == "__main__":
    matrix = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    print("Matrix:")
    print_matrix(matrix)
    primary_sum, secondary_sum = sum_of_diagonals(matrix)
    print(f"Sum of primary diagonal elements: {primary_sum}")
    print(f"Sum of secondary diagonal elements: {secondary_sum}")
```

OUTPUT-

```
Matrix:
1 2 3
4 5 6
7 8 9
Sum of primary diagonal elements: 15
Sum of secondary diagonal elements: 15
```

21)Write a Python Program Reverse the elements in an array of integers without using a second array.

```
def reverse_array(arr):
    start = 0
    end = len(arr) - 1
    while start < end:
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
        end -= 1
if __name__ == "__main__":
    array = [1, 2, 3, 4, 5, 6, 7]
    print("Original array:")
    print(array)
    reverse_array(array)
    print("Reversed array:")
    print(array)
```

OUTPUT-

```
Original array:
[1, 2, 3, 4, 5, 6, 7]
Reversed array:
[7, 6, 5, 4, 3, 2, 1]
```

22) Write a Python program to enter n elements in an array and find smallest number among them.

```
def find_smallest_number(arr):
    if not arr:
        raise ValueError("Array is empty.")
    smallest = arr[0]
    for number in arr:
        if number < smallest:
            smallest = number
    return smallest
if __name__ == "__main__":
    n = int(input("Enter the number of elements: "))
    if n <= 0:
        print("The number of elements must be positive.")
    else:
        array = []
        for i in range(n):
            element = int(input(f"Enter element {i + 1}: "))
            array.append(element)
        try:
            smallest_number = find_smallest_number(array)
            print(f"The smallest number in the array is: {smallest_number}")
        except ValueError as e:
            print(e)
```

OUTPUT-

```
Enter the number of elements: 5
Enter element 1: 10
Enter element 2: 5
Enter element 3: 8
Enter element 4: 1
Enter element 5: 7
The smallest number in the array is: 1
```