**ASSIGNMENT-3**

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

def check\_season(month):

if month in [12, 1, 2]:

return 'Winter'

elif month in [3, 4, 5]:

return 'Spring'

elif month in [6, 7, 8]:

return 'Summer'

elif month in [9, 10, 11]:

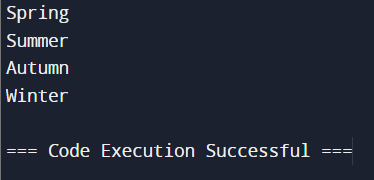
return 'Autumn'

print(check\_season(3)) # Spring

print(check\_season(7)) # Summer

print(check\_season(11)) # Autumn

print(check\_season(1)) # Winter



1. Write a function called calculate\_slope which return the slope of a linear equation.

def calculate\_slope():

x1, y1 = map(float, input("Enter the coordinates of the first point (x1 y1): ").split())

x2, y2 = map(float, input("Enter the coordinates of the second point (x2 y2): ").split())

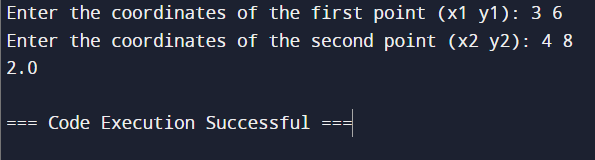
if x1 == x2:

return 'Undefined'

return (y2 - y1) / (x2 - x1)

# Example usage:

print(calculate\_slope())



1. Quadratic equation is calculated as follows: ax² + bx + c = 0. Write a function which calculates solution set of a quadratic equation, \_solve\_quadratic\_eqn\_.

import cmath

def solve\_quadratic\_eqn():

a = float(input("Enter coefficient a: "))

b = float(input("Enter coefficient b: "))

c = float(input("Enter coefficient c: "))

discriminant = cmath.sqrt(b\*\*2 - 4\*a\*c)

sol1 = (-b + discriminant) / (2 \* a)

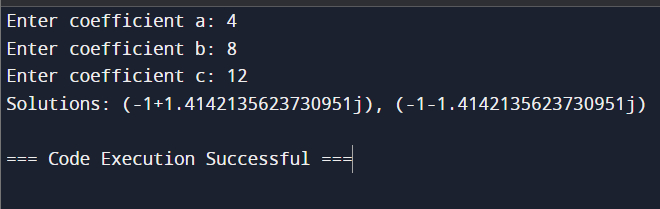
sol2 = (-b - discriminant) / (2 \* a)

return sol1, sol2

# Example usage:

solutions = solve\_quadratic\_eqn()

print(f"Solutions: {solutions[0]}, {solutions[1]}")



1. 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(lst):

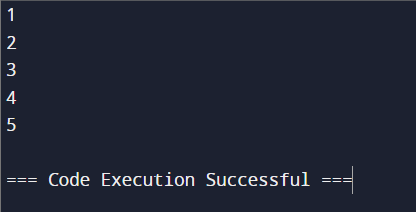
for element in lst:

print(element)

# Example usage:

example\_list = [1, 2, 3, 4, 5]

print\_list(example\_list)



1. 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():

array = input("Enter elements of the array separated by spaces: ").split()

reversed\_array = []

for i in range(len(array) - 1, -1, -1):

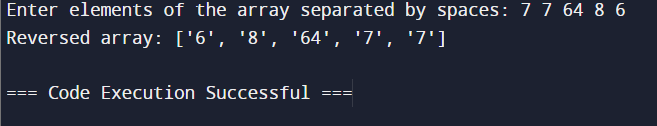
reversed\_array.append(array[i])

return reversed\_array

# Example usage:

reversed\_array = reverse\_list()

print("Reversed array:", reversed\_array)



1. 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 sum\_series():

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

series\_sum = 0

for i in range(1, n + 1):

term = 1 / i if i % 2 != 0 else -1 / i

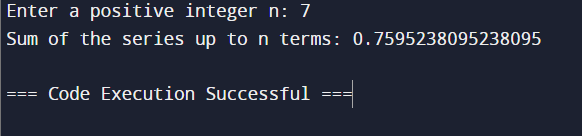
series\_sum += term

return series\_sum

# Example usage:

result = sum\_series()

print(f"Sum of the series up to n terms: {result}")



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 compute\_sin():

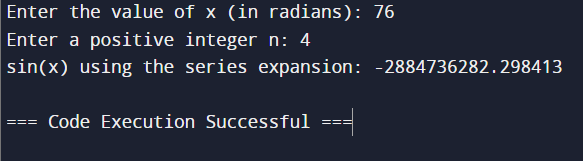
x = float(input("Enter the value of x (in radians): "))

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

return sum((-1)\*\*i \* x\*\*(2\*i+1) / math.factorial(2\*i+1) for i in range(n))

# Example usage:

print(f"sin(x) using the series expansion: {compute\_sin()}")



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 cosine\_series(x, n):

cos\_x = 0

for i in range(n + 1):

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

cos\_x += term

return cos\_x

# User input

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

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

# Check if n is positive

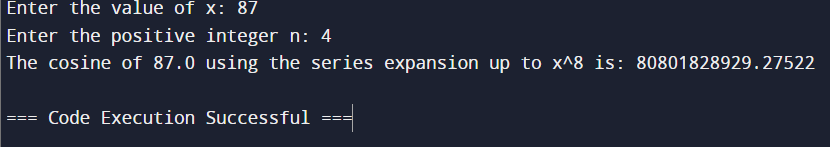
if n < 0:

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

else:

result = cosine\_series(x, n)

print(f"The cosine of {x} using the series expansion up to x^{2\*n} is: {result}")



9. Print the pattern upto N Lines:

.

/\_\

.

/ \

/\_\_\_\

.

/ \

/ \

/\_\_\_\_\_\

N=2 N=3 N=4

def print\_pattern(N):

for i in range(1, N + 1):

# Print first line of the pattern

print(" . ")

# Print the middle lines of the pattern

for j in range(1, i + 1):

print("/" + " " \* (2 \* j - 2) + "\\")

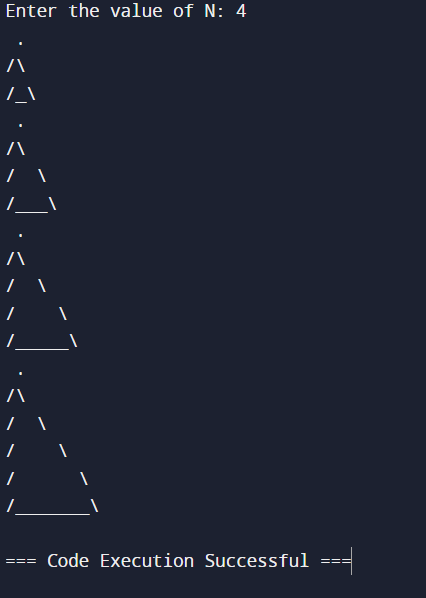
# Print the last line of the pattern

print("/" + "\_" \* (2 \* i - 1) + "\\")

# User input

N = int(input("Enter the value of N: "))

print\_pattern(N)



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

\_

\_|

|\_

\_

\_|

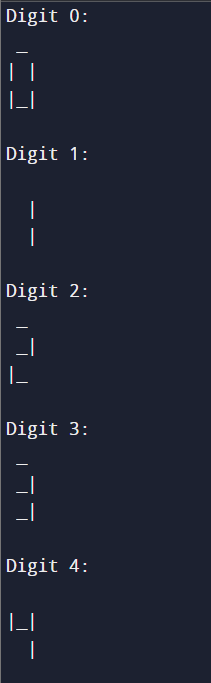
\_|

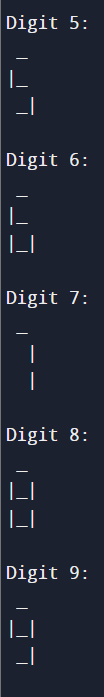
|\_|

|

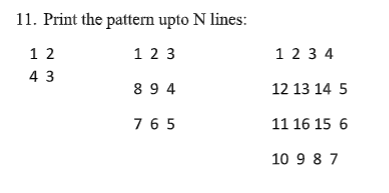
N=2 N=3 N=4

print() # New line for separation





11.



N=2 N=3 N=4

def print\_pattern(n):

matrix = [[0] \* n for \_ in range(n)]

num, top, bottom, left, right = 1, 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

print('\n'.join(' '.join(map(str, row)) for row in matrix))

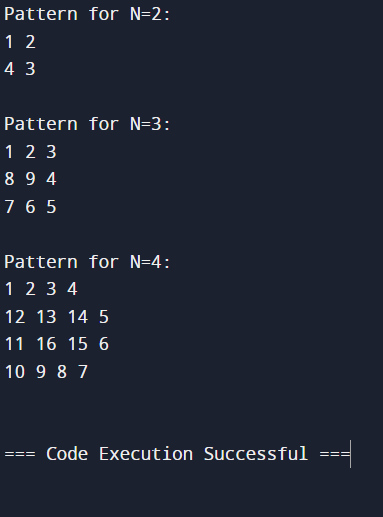
# Example usage:

for N in [2, 3, 4]:

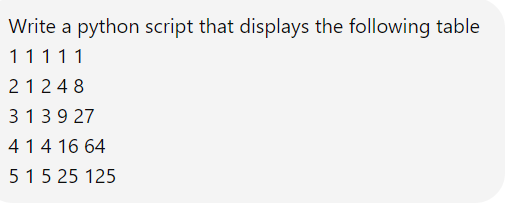
print(f"Pattern for N={N}:")

print\_pattern(N)

print() # New line for separation



12.



def print\_table(rows, cols):

# Generate the table

table = [[(i + 1) \*\* j for j in range(cols)] for i in range(rows)]

# Print the table

for row in table:

print(" ".join(map(str, row)))

# Define number of rows and columns

rows, cols = 5, 5

# Print the table

print\_table(rows, cols)

