# MACHINE LEARNING

ARTIFICIAL INTELLIGENCE – E

ASSIGNMENT 1

SYED ABRAR C AIE22147

NITHESH NAIR PS AIE22133

## INTRODUCTION:

In this report, we have explained how we did our python programs using funciton's. We have 4 python problems such as determining the range of the given list, Extracting the pair of elements in the list which sum is equal to the targeted sum given by the user, Highest occurrence of a character in the string and Matrix Multiplication.

## PROBLEMS:

1) This program is to identify pairs of elements in a list whose sum equals a specified target. The find_pairs_with_sum function systematically traverses each element in the list, examining potential pairs by iterating through subsequent elements. Pairs that satisfy the target sum condition are collected in a list, which is then returned by the function.

The main function sets up an empty list to gather user input. It prompts the user for the number of elements, individual element values, and the target sum. Subsequently, it employs the find_pairs_with_sum function to detect pairs meeting the target sum criterion. The script then presents the results, showcasing the identified pairs, or communicates if no such pairs are found.

PSEUDOCODE:

```
function find_pairs_with_sum(arr, target):

  pairs = empty list
```

```
    for i from 0 to length of arr - 1:
        for j from i + 1 to length of arr - 1:
            if arr[i] + arr[j] equals target:
                add (arr[i], arr[j]) to pairs
    return pairs
function main():
    my_list = empty list
    num_elements = get user input for the number of elements in the list
    for i from 0 to num_elements - 1:
        element = get user input for the element at index i
        add element to my_list
    target_sum = get user input for the target sum
    pairs = call find_pairs_with_sum with my_list and target_sum
    if pairs is empty:
        print "No pairs found in the given list for the target sum."
    else:
        print "Pairs with the sum", target_sum, "are:", pairs
call main()
```

2) This program is to calculates the range between the largest and smallest numbers in a user-provided list. The `range_of_list` function first checks if the number of elements in the list is less than or equal to 3. If so, it concludes that it's not possible to determine the range with such a small number of elements. Otherwise, it prompts the user to input each element of the list through the `user_list_input` function. Subsequently, the script calculates the minimum and maximum values within the list and returns the range, which is the difference between the largest and smallest numbers.

PSEUDOCODE:

```
Function range_of_list(user_list, m):
    if m <= 3:
        return "It's not possible to determine the range."
    else:
        user_list_input(user_list, m)
        minimum = min(user_list)
        maximum = max(user_list)
return maximum - minimum
Function user_list_input(user_list, m):
    For i in range(0, m):
        element = get_user_input()
        append element to user_list
    Return user_list
Function get_user_input():
    return integer value entered by the user
# Main program
num_elements = get_user_input()
user_list = empty list
range_result = range_of_list(user_list, num_elements)
Display "The range between the largest and smallest number is", range_result
```

3) This program is to matrix exponentiation, allowing users to input a square matrix, specify its dimension, and raise it to a given power. The functions used: **matrix_multiplication,**

**identity_matrix, and power_of_matrix.** The **matrix_multiplication** function computes the product of two matrices. It initializes an empty result matrix and iterates through each row and column, calculating the dot product for each element. The **identity_matrix** function generates an identity matrix of a given size, where diagonal elements are 1, and others are 0. The **power_of_matrix** function raises a square matrix to a specified power using binary exponentiation. It checks if the input matrix is square, initializes the result matrix as an identity matrix, and iteratively multiplies the result matrix by the original matrix, updating the power using binary representation until the desired power is reached.

PSEUDOCODE:

```
Function matrix_multiplication(A, B):
  result_matrix = []
  For i in range(length(A)):
    row = []
    For j in range(length(B[0])):
      element = sum(A[i][k] * B[k][j] for k in range(length(A[0])))
      append element to row
    append row to result_matrix
  Return result_matrix
Function identity_matrix(size):
  Return a 2D list with 1 in diagonal, 0 elsewhere, size x size
Function power_of_matrix(matrix, m):
  If length(matrix) is not equal to length(matrix[0]):
    Raise ValueError('Input matrix should be a square matrix!')
  result_matrix = identity_matrix(length(matrix))
  current_power = m
  While current_power > 0:
        If current_power is odd:

            result_matrix = matrix_multiplication(result_matrix, matrix)

        matrix = matrix_multiplication(matrix, matrix)

        current_power //= 2

    Return result_matriX

# Main program

n = get_user_input("Matrix Dimension: ")

matrix = create_matrix(n)

m = get_user_input("Enter the power of matrix: ")

result_matrix = power_of_matrix(matrix, m)

For row in result_matrix: print(row)
```

4) This program is to determine a user-provided string to identify the character with the highest occurrence and its count. The functions used: **clean_string, count_characters, find_highest_occurrence, and count_highest_occurrence.**

The **clean_string** function removes non-alphabetic characters from the input string and converts the remaining characters to lowercase. This ensures a standardized representation for counting occurrences.The **count_characters** function tallies the occurrences of each character in the cleaned string, utilizing a dictionary (char_count) to store the counts.The **find_highest_occurrence** function identifies the character with the highest occurrence by extracting the maximum count and its corresponding character from the **char_count** dictionary. The **count_highest_occurrence** function integrates the previous functions. It cleans the input string, counts the character occurrences, and then determines and returns the character with the highest count.

PSEUDOCODE:

```
Function count_characters(input_string):

    char_count = {}

    For each character i in input_string:
        If i is in char_count:
            Increment the count for i in char_count
        Else:
            Set the count for i in char_count to 1
    Return char_count
 Function find_highest_occurrence(char_count):
    max_char = character with maximum count in char_count
    max_count = count of max_char in char_count
    Return max_char, max_count
Function count_highest_occurrence(input_string):
    cleaned_string = clean_string(input_string)
    char_count = count_characters(cleaned_string)
    max_char, max_count = find_highest_occurrence(char_count)
    Return max_char, max_count
Function clean_string(input_string):
    Return a string containing only alphabetic characters in lowercase from input_string
# Main program
input_string = get_user_input("Enter the string: ")
max_char, max_count = count_highest_occurrence(input_string)
Display f"The maximum occurring character: '{max_char}' \n Occurrence count: {max_count}."
```