**Name:Prakriti Khadka**

**Task 1. Classify Temperatures:**

1. Create empty lists for temperature classifications:

(a) Cold: temperatures below 10°C.

(b) Mild: temperatures between 10°C and 15°C.

(c) Comfortable: temperatures between 15°C and 20°C.

2. Iterate over the temperatures list and add each temperature to the appropriate cate-

gory.

3. Print the lists to verify the classifications.

temperatures = [8.2, 17.4, 14.1, 7.9, 18.0, 13.5, 9.0, 17.8, 13.0, 8.5,

16.5, 12.9, 7.7, 17.2, 13.3, 8.4, 16.7, 14.0, 9.5, 18.3, 13.4, 8.1,

17.9, 14.2, 7.6, 17.0, 12.8, 8.0, 16.8, 13.7, 7.8, 17.5, 13.6, 8.7,

17.1, 13.8, 9.2, 18.1, 13.9, 8.3, 16.4, 12.7, 8.9, 18.2, 13.1, 7.8,

16.6, 12.5]

n=len(temperatures)

Cold=[]

Mild=[]

Comfortable=[]

for i in range(n):

    if(temperatures[i]<10):

        Cold.append(temperatures[i])

    elif(temperatures[i]>=10 and temperatures[i]<=15):

        Mild.append(temperatures[i])

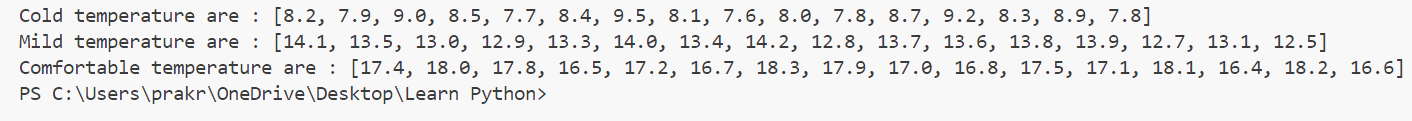
    elif(temperatures[i]>=15 and temperatures[i]<=20):

        Comfortable.append(temperatures[i])

print("Cold temperature are :",Cold)

print("Mild temperature are :",Mild)

print("Comfortable temperature are :",Comfortable)



**Task 2. Based on Data - Answer all the Questions:**

1. How many times was it mild?

(a) Hint: Count the number of items in the mild list and print the result.

2. How many times was it comfortable?

3. How many times was it cold?

temperatures = [8.2, 17.4, 14.1, 7.9, 18.0, 13.5, 9.0, 17.8, 13.0, 8.5,

16.5, 12.9, 7.7, 17.2, 13.3, 8.4, 16.7, 14.0, 9.5, 18.3, 13.4, 8.1,

17.9, 14.2, 7.6, 17.0, 12.8, 8.0, 16.8, 13.7, 7.8, 17.5, 13.6, 8.7,

17.1, 13.8, 9.2, 18.1, 13.9, 8.3, 16.4, 12.7, 8.9, 18.2, 13.1, 7.8,

16.6, 12.5]

n=len(temperatures)

Cold=[]

Mild=[]

Comfortable=[]

for i in range(n):

    if(temperatures[i]<10):

        Cold.append(temperatures[i])

    elif(temperatures[i]>=10 and temperatures[i]<=15):

        Mild.append(temperatures[i])

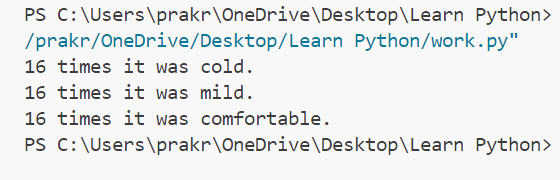
    elif(temperatures[i]>=15 and temperatures[i]<=20):

        Comfortable.append(temperatures[i])

print(len(Cold),"times it was cold.")

print(len(Mild),"times it was mild.")

print(len(Comfortable),"times it was comfortable.")



**Task 3. Convert Temperatures from Celsius to Fahrenheit**

Using the formula for temperature conversion, convert each reading from Celsius to Fahren-

heit and store it in a new list called temperatures\_fahrenheit.

1. Iterate over the temperatures list and apply the formula to convert each temperature.

2. Store the results in the new list.

3. Print the converted Fahrenheit values.

temperatures = [8.2, 17.4, 14.1, 7.9, 18.0, 13.5, 9.0, 17.8, 13.0, 8.5,

16.5, 12.9, 7.7, 17.2, 13.3, 8.4, 16.7, 14.0, 9.5, 18.3, 13.4, 8.1,

17.9, 14.2, 7.6, 17.0, 12.8, 8.0, 16.8, 13.7, 7.8, 17.5, 13.6, 8.7,

17.1, 13.8, 9.2, 18.1, 13.9, 8.3, 16.4, 12.7, 8.9, 18.2, 13.1, 7.8,

16.6, 12.5]

n=len(temperatures)

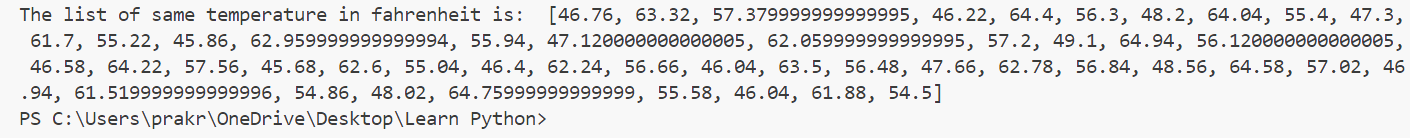
Fahrenheit\_temp=[]

for i in range(n):

    Fahrenheit = (temperatures[i] \* 9/5) + 32

    Fahrenheit\_temp.append(Fahrenheit)

print("The list of same temperature in fahrenheit is: ", Fahrenheit\_temp)



**Task 4. Analyze Temperature Patterns by Time of Day:**

Scenario: Each day’s readings are grouped as:

• Night (00-08),

• Evening (08-16),

• Day (16-24).

1. Create empty lists for night, day, and evening temperatures.

2. Iterate over the temperatures list, assigning values to each time-of-day list based on

their position.

3. Calculate and print the average day-time temperature.

temperatures = [8.2, 17.4, 14.1, 7.9, 18.0, 13.5, 9.0, 17.8, 13.0, 8.5,

16.5, 12.9, 7.7, 17.2, 13.3, 8.4, 16.7, 14.0, 9.5, 18.3, 13.4, 8.1,

17.9, 14.2, 7.6, 17.0, 12.8, 8.0, 16.8, 13.7, 7.8, 17.5, 13.6, 8.7,

17.1, 13.8, 9.2, 18.1, 13.9, 8.3, 16.4, 12.7, 8.9, 18.2, 13.1, 7.8,

16.6, 12.5]

n=len(temperatures)

Night=[]

Evening=[]

Day=[]

for i, temp in enumerate(temperatures):

    if i % 24 < 8:

        Night.append(temp)

    elif i % 24 < 16:

        Evening.append(temp)

    else:

        Day.append(temp)

if Day:

    avg\_day\_temp = sum(Day) / len(Day)

else:

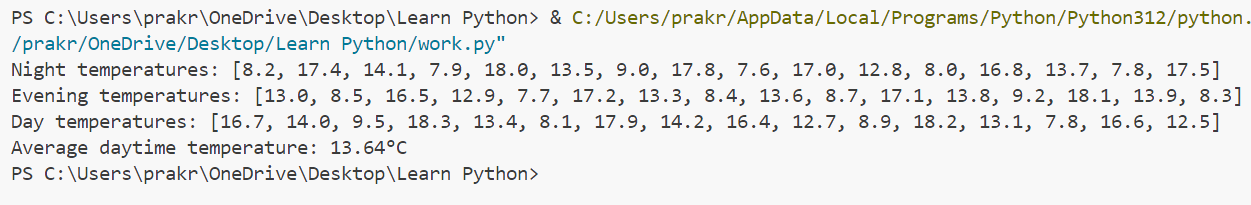
    avg\_day\_temp = 0

print("Night temperatures:", Night)

print("Evening temperatures:", Evening)

print("Day temperatures:", Day)

print(f"Average daytime temperature: {avg\_day\_temp:.2f}°C")



**8.1.1 Exercise - Recursion:**

Task 1 - Sum of Nested Lists:

Write a recursive function sum\_nested\_list(nested\_list) that:

1. Takes a nested list (a list that can contain numbers or other lists of numbers) as input.

2. Sums all numbers at every depth level of the list, regardless of how deeply nested the numbers are.

• Test the function with a sample nested list, such as nested\_list = [1, [2, [3, 4], 5], 6, [7, 8]].

The result should be the total sum of all the numbers.

def sum\_nested\_list(nested\_list):

    """

    Calculate the sum of all numbers in a nested list.

    Args:

        nested\_list (list): A list that may contain integers or other lists of integers.

    Returns:

        int: The total sum of all integers in the nested list, including those in sublists.

    Example:

    >>> sum\_nested\_list([1, [2, [3, 4], 5], 6, [7, 8]])

    36

    >>> sum\_nested\_list([1, [2, 3], [4, [5]]])

    15

    """

    total = 0

    for element in nested\_list:

        if isinstance(element, list):

            total += sum\_nested\_list(element)

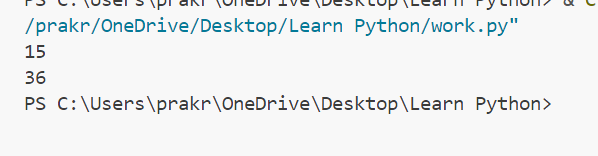
        else:

            total += element

    return total

print(sum\_nested\_list([1, [2, 3], [4, [5]]]))

print(sum\_nested\_list([1, [2, [3, 4], 5], 6, [7, 8]]))



Task 2 - Generate All Permutations of a String:

Scenario: Given a string, generate all possible permutations of its characters. This is useful

for understanding backtracking and recursive depth-first search.

Task:

• Write a recursive function generate\_permutations(s) that:

– Takes a string s as input and returns a list of all unique permutations.

• Test with strings like ”abc” and ”aab”.

print(generate\_permutations("abc"))

# Should return [’abc’, ’acb’, ’bac’, ’bca’, ’cab’, ’cba’]

def generate\_permutations(s):

    """

    Generate all unique permutations of a string.

    Args:

        s (str): The input string.

    Returns:

        list: A list of all unique permutations of the string.

    Example:

    >>> generate\_permutations("abc")

    ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']

    >>> generate\_permutations("aab")

    ['aab', 'aba', 'baa']

    """

    def backtrack(path, remaining, result):

        if not remaining:

            result.append("".join(path))

            return

        for i in range(len(remaining)):

            if i > 0 and remaining[i] == remaining[i - 1]:

                continue

            backtrack(path + [remaining[i]], remaining[:i] + remaining[i+1:], result)

    result = []

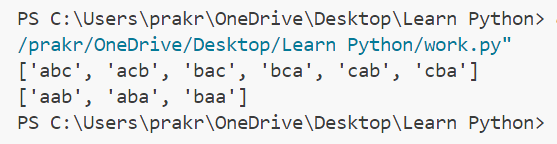
    sorted\_chars = sorted(s)

    backtrack([], sorted\_chars, result)

    return result

print(generate\_permutations("abc"))

print(generate\_permutations("aab"))



**Task 3 - Directory Size Calculation:**

Directory Size Calculation Scenario: Imagine a file system where directories can contain files

(with sizes in KB) and other directories. You want to calculate the total size of a directory,

including all nested files and subdirectories.

Task:

1. Write a recursive function calculate\_directory\_size(directory) where:

• directory is a dictionary where keys represent file names (with values as sizes in KB) or directory names (with values as another dictionary representing a subdi-rectory).

• The function should return the total size of the directory, including all nested

subdirectories.

2. Test the function with a sample directory structure.

def calculate\_directory\_size(directory):

    """

    Calculate the total size of a directory, including all nested subdirectories.

    Args:

        directory (dict): A dictionary representing the directory structure.

    Returns:

        int: Total size of the directory in KB.

    """

    total\_size = 0

    for name, value in directory.items():

        if isinstance(value, dict):

            total\_size += calculate\_directory\_size(value)

        else:

            total\_size += value

    return total\_size

directory\_structure = {

    "file1.txt": 200,

    "file2.txt": 300,

    "subdir1": {

        "file3.txt": 400,

        "file4.txt": 100

    },

    "subdir2": {

        "subsubdir1": {

            "file5.txt": 250

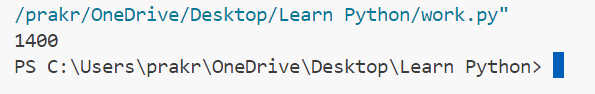
        },

        "file6.txt": 150

    }

}

print(calculate\_directory\_size(directory\_structure))



**8.2.2 Exercises - Dynamic Programming:**

Task 1 - Coin Change Problem:

Scenario: Given a set of coin denominations and a target amount, find the minimum number

of coins needed to make the amount. If it’s not possible, return - 1.

Task:

1. Write a function min\_coins(coins, amount) that:

• Uses DP to calculate the minimum number of coins needed to make up the

amount.

2. Test with coins = [1, 2, 5] and amount = 11. The result should be 3 (using coins

[5, 5, 1]).

def min\_coins(coins, amount):

    """

    Calculate the minimum number of coins needed to make the given amount.

    Args:

        coins (list): List of available coin denominations.

        amount (int): Target amount to achieve.

    Returns:

        int: Minimum number of coins needed, or -1 if it's not possible.

    """

    dp = [amount + 1] \* (amount + 1)

    dp[0] = 0

    for a in range(1, amount + 1):

        for coin in coins:

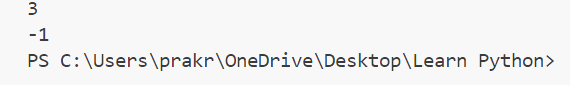
            if a >= coin:

                dp[a] = min(dp[a], dp[a - coin] + 1)

    return dp[amount] if dp[amount] != amount + 1 else -1

print(min\_coins([1, 2, 5], 11))

print(min\_coins([2],3))



**Task 2 - Longest Common Subsequence (LCS):**

Scenario: Given two strings, find the length of their longest common subsequence (LCS).

This is useful in text comparison.

Task:

1. Write a function longest\_common\_subsequence(s1, s2) that:

• Uses DP to find the length of the LCS of two strings s1 and s2.

2. Test with strings like "abcde" and "ace"; the LCS length should be 3 ("ace").

def longest\_common\_subsequence(s1, s2):

    """

    Find the length of the longest common subsequence (LCS) of two strings.

    Args:

        s1 (str): The first string.

        s2 (str): The second string.

    Returns:

        int: The length of the LCS.

    Example:

        >>> longest\_common\_subsequence("abcde", "ace")

        3

    """

    m, n = len(s1), len(s2)

    dp = [[0] \* (n + 1) for \_ in range(m + 1)]

    for i in range(1, m + 1):

        for j in range(1, n + 1):

            if s1[i - 1] == s2[j - 1]:

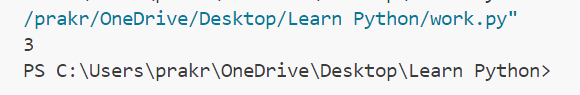
                dp[i][j] = dp[i - 1][j - 1] + 1

            else:

                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[m][n]

print(longest\_common\_subsequence("abcde", "ace"))



**Task 3 - 0/1 Knapsack Problem:**

Scenario: You have a list of items, each with a weight and a value. Given a weight capacity,

maximize the total value of items you can carry without exceeding the weight capacity.

Task:

1. Write a function knapsack(weights, values, capacity) that:

• Uses DP to determine the maximum value that can be achieved within the given

weight capacity.

2. Test with weights [1, 3, 4, 5], values [1, 4, 5, 7], and capacity 7. The result should be 9.

def knapsack(weights, values, capacity):

    """

    Solve the 0/1 Knapsack problem using dynamic programming.

    Args:

        weights (list): List of item weights.

        values (list): List of item values.

        capacity (int): Maximum weight capacity of the knapsack.

    Returns:

        int: The maximum value that can be achieved without exceeding the capacity.

    Example:

        >>> knapsack([1, 3, 4, 5], [1, 4, 5, 7], 7)

        9

    """

    n = len(weights)

    dp = [[0] \* (capacity + 1) for \_ in range(n + 1)]

    for i in range(1, n + 1):

        for w in range(1, capacity + 1):

            if weights[i - 1] <= w:

                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])

            else:

                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]

print(knapsack([1, 3, 4, 5], [1, 4, 5, 7], 7))

