# Week 0

### Task 1

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

]

cold = []

mild = []

comfortable = []

for temp in temperatures:

if temp < 10:

cold.append(temp)

elif 10 <= temp < 15:

mild.append(temp)

elif 15 <= temp <= 20:

comfortable.append(temp)

classified\_temperatures = {

"Cold": cold,

"Mild": mild,

"Comfortable": comfortable

}

print(classified\_temperatures);

print("Cold temperatures:", len(cold))

print("Mild temperatures:", len(mild))

print("Comfortable temperatures:", len(comfortable))

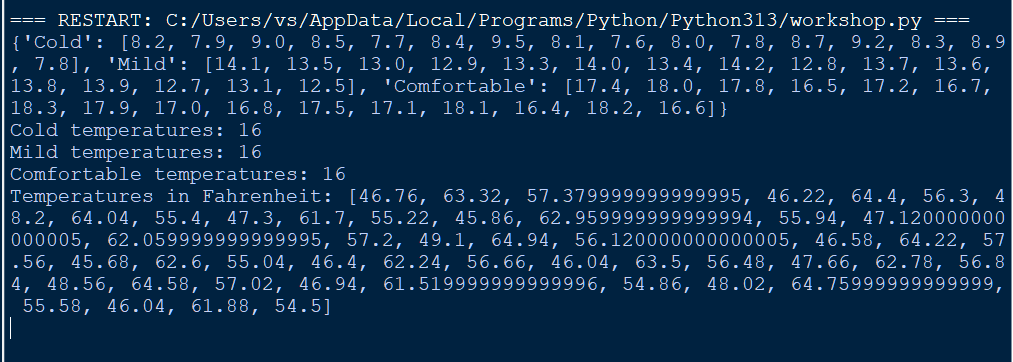
temperatures\_fahrenheit = []

for temp in temperatures:

fahrenheit = (temp \* 9/5) + 32

temperatures\_fahrenheit.append(fahrenheit)

print("Temperatures in Fahrenheit:", temperatures\_fahrenheit)

Output: 

### Task 2

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

"""

Function to find the length of the longest common subsequence (LCS)

between two strings s1 and s2 using dynamic programming.

Parameters:

s1 (str): The first string

s2 (str): The second string

Returns:

int: Length of the longest common subsequence

"""

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])

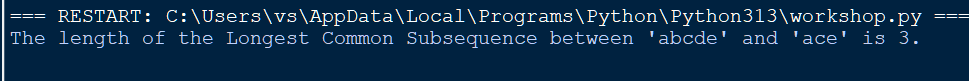
return dp[m][n]

s1 = "abcde"

s2 = "ace"

lcs\_length = longest\_common\_subsequence(s1, s2)

print(f"The length of the Longest Common Subsequence between '{s1}' and '{s2}' is {lcs\_length}.")Output:



### 

### task 3 of 8.2.2

def knapsack(weights, values, capacity):

"""

Solves the 0/1 Knapsack problem using dynamic programming.

weights (list): List of item weights.

values (list): List of item values.

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

"""

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:

include = values[i - 1] + dp[i - 1][w - weights[i - 1]]

exclude = dp[i - 1][w]

dp[i][w] = max(include, exclude)

else:

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

return dp[n][capacity]

weights = [1, 3, 4, 5]

values = [1, 4, 5, 7]

capacity = 7

max\_value = knapsack(weights, values, capacity)

print("Maximum Value:", max\_value)

Output:

