**WEEK 11 & 12**

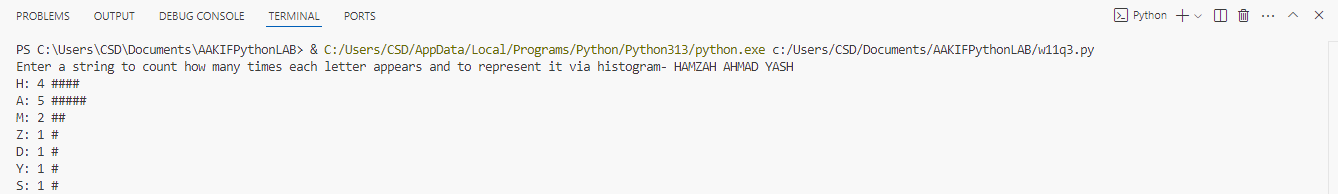
**1- Assume that the variable data refers to the dictionary {“b”:20, “a”:35}. Write the values of the following expressions:**

data = {"b": 20, "a": 35}  
print("The following expressions with the dictionary: ")  
print("data['a']:", data["a"])  
print("data.get('c', None):", data.get("c", None))  
print("len(data):", len(data))  
print("data.keys():", data.keys())  
print("data.values():", data.values())  
print("data.pop('b'):", data.pop("b", None))  
print("Updated data:", data)

**2- Write the expressions that perform the following tasks:  
Replace the value at the key “b” in data with that value’s negation. Add the key/value pair “c”:40 to data. Remove the value at key “b” in data, safely. Print the keys in data in alphabetical order.**

data = {"b": 20, "a": 35}  
print("The following expressions with the dictionary: ")  
data["b"] = -data["b"]  
data["c"] = 40  
data.pop("b", None)  
print(sorted(data.keys()))

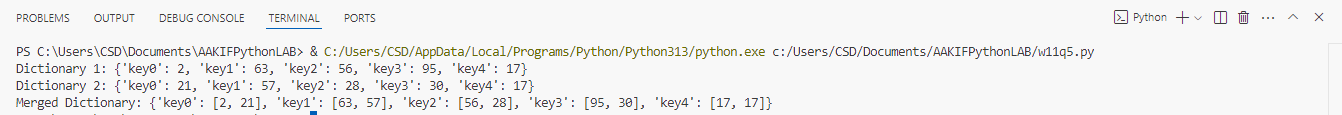
**3- Write a python program to read a string and count how many times each letter appears. (Histogram)**

string = input("Enter a string to count how many times each letter appears and to represent it via histogram- ")  
histogram = {}  
for char in string:  
 if char.isalpha():  
 histogram[char] = histogram.get(char, 0) + 1  
for char, count in histogram.items():  
 print(f"{char}: {count} {'#' \* count}")

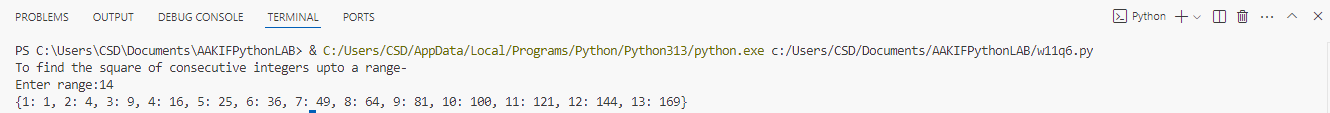
**4- Write a python program to create a dictionary, read a value from the user and search the key element (Reverse lookup).**

dict1 = {"a": 1, "b": 2, "c": 3, "d": 4}  
value = int(input("Enter a value to search for its key in the dictionary: "))  
keys = [key for key, val in dict1.items() if val == value]  
if keys:  
 print(f"Key(s) found: {', '.join(keys)}")  
else:  
 print("No key found for the given value.")

**5- Write a python program to create two dictionaries and merge them.**

import random  
dict1 = {}  
for i in range(5):   
 dict1[f'key{i}'] = random.randint(1, 100)  
print("Dictionary 1:", dict1)  
  
dict2 = {}  
for i in range(5):   
 dict2[f'key{i}'] = random.randint(1, 100)  
print("Dictionary 2:", dict2)  
  
merged\_dict = {}  
for key, value in dict1.items():  
 merged\_dict[key] = [value]   
  
for key, value in dict2.items():  
 if key in merged\_dict:  
 merged\_dict[key].append(value)   
 else:  
 merged\_dict[key] = [value]   
  
print("Merged Dictionary:", merged\_dict)

**6- Write a Python script to create a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys.**

print("To find the square of consecutive integers upto a range-")  
ran=int(input("Enter range:"))  
result = {x: x\*\*2 for x in range(1, ran)}  
print(result)

**7- Implement the Matrix Chain Multiplication problem using dynamic programming in Python.**

****def matrix\_chain\_order(p):  
 n = len(p) - 1  
 dp = [[0] \* n for \_ in range(n)]  
 for l in range(2, n + 1):  
 for i in range(n - l + 1):  
 j = i + l - 1  
 dp[i][j] = float('inf')  
 for k in range(i, j):  
 dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j] + p[i] \* p[k + 1] \* p[j + 1])  
 return dp[0][n - 1]  
  
p = [40, 20, 30, 10, 30]  
print(matrix\_chain\_order(p))

**8- Develop a Python program to find the shortest path in a graph using Dijkstra’s algorithm.**

import heapq  
def dijkstra(graph, start):  
 distances = {node: float('inf') for node in graph}  
 distances[start] = 0  
 priority\_queue = [(0, start)]  
  
 while priority\_queue:  
 current\_distance, current\_node = heapq.heappop(priority\_queue)  
  
 if current\_distance > distances[current\_node]:  
 continue  
  
 for neighbor, weight in graph[current\_node].items():  
 distance = current\_distance + weight  
 if distance < distances[neighbor]:  
 distances[neighbor] = distance  
 heapq.heappush(priority\_queue, (distance, neighbor))  
  
 return distances  
  
graph = {  
 'A': {'B': 1, 'C': 4},  
 'B': {'A': 1, 'C': 2, 'D': 6},  
 'C': {'A': 4, 'B': 2, 'D': 3},  
 'D': {'B': 6, 'C': 3}  
}  
start\_node = 'A'  
print(dijkstra(graph, start\_node))