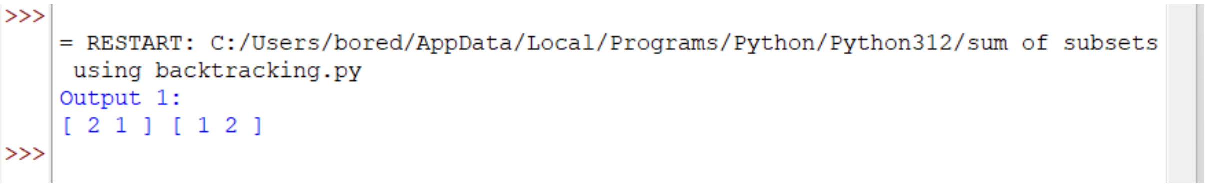
LAB-15

Sum of subsets using back tracking

CODE:

```
flag = False

def print_subset_sum(i, n, _set, target_sum, subset):

        global flag

        if target_sum == 0:

                flag = True

                print("[", end=" ")

                for element in subset:

                        print(element, end=" ")

                print("]", end=" ")

                return

        if i == n:

                return

        print_subset_sum(i + 1, n, _set, target_sum, subset)

        if _set[i] <= target_sum:

                subset.append(_set[i])

                print_subset_sum(i + 1, n, _set, target_sum - _set[i], subset)

                subset.pop()

if __name__ == "__main__":

        set_1 = [1, 2, 1]

        sum_1 = 3

        n_1 = len(set_1)

        subset_1 = []

        print("Output 1:")

        print_subset_sum(0, n_1, set_1, sum_1, subset_1)

        print()

        flag = False
```

OUTPUT:



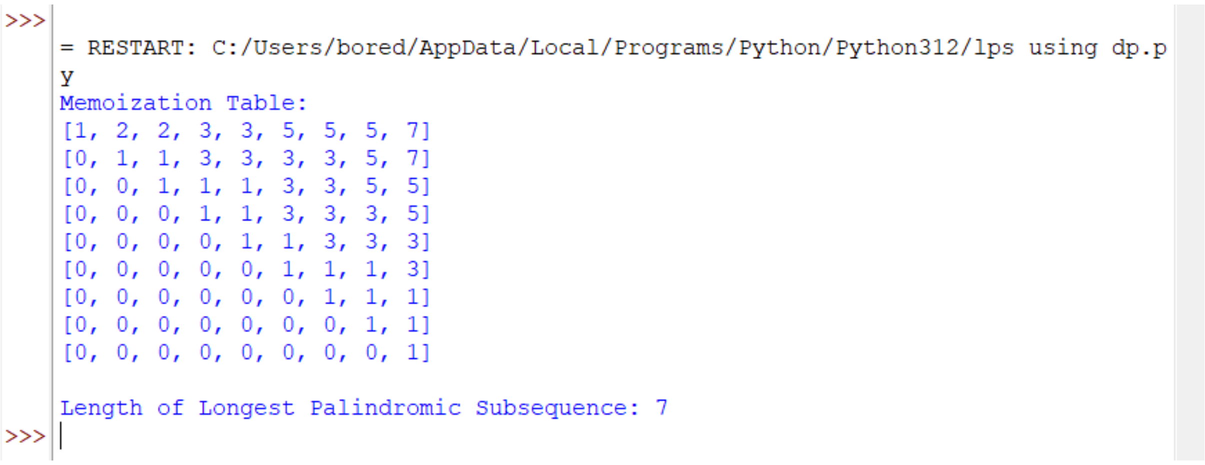
```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/sum of subsets
 using backtracking.py
Output 1:
[ 2 1 ] [ 1 2 ]
>>>
```

Lps using dp

CODE:

```
def longest_palindromic_subsequence(s):
    n = len(s)
    dp = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        dp[i][i] = 1
    for cl in range(2, n+1):
        for i in range(n - cl + 1):
            j = i + cl - 1
            if s[i] == s[j] and cl == 2:
                dp[i][j] = 2
            elif s[i] == s[j]:
                dp[i][j] = dp[i + 1][j - 1] + 2
            else:
                dp[i][j] = max(dp[i][j - 1], dp[i + 1][j])
    return dp, dp[0][n-1]
s = "bbabcbcab"
dp_table, length_of_lps = longest_palindromic_subsequence(s)
print("Memoization Table:")
for row in dp_table:
    print(row)
print("\nLength of Longest Palindromic Subsequence:", length_of_lps)
```

OUTPUT:



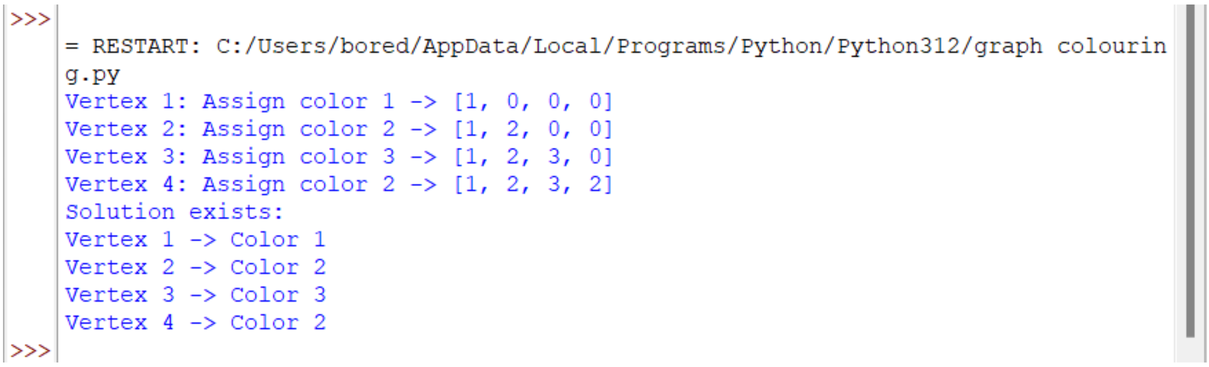
```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/lps using dp.p
    y
    Memoization Table:
    [1, 2, 2, 3, 3, 5, 5, 5, 7]
    [0, 1, 1, 3, 3, 3, 3, 5, 7]
    [0, 0, 1, 1, 1, 3, 3, 5, 5]
    [0, 0, 0, 1, 1, 3, 3, 3, 5]
    [0, 0, 0, 0, 1, 1, 3, 3, 3]
    [0, 0, 0, 0, 0, 1, 1, 1, 3]
    [0, 0, 0, 0, 0, 0, 1, 1, 1]
    [0, 0, 0, 0, 0, 0, 0, 1, 1]
    [0, 0, 0, 0, 0, 0, 0, 0, 1]

    Length of Longest Palindromic Subsequence: 7
>>> |
```

Graph colouring

CODE:

```
class Graph:

    def __init__(self, vertices):

        self.V = vertices

        self.graph = [[0 for _ in range(vertices)] for _ in range(vertices)]

    def is_safe(self, v, color, c):

        for i in range(self.V):

            if self.graph[v][i] == 1 and color[i] == c:

                return False

        return True

    def graph_color_util(self, m, color, v):

        if v == self.V:

            return True

        for c in range(1, m + 1):

            if self.is_safe(v, color, c):

                color[v] = c

                print(f"Vertex {v+1}: Assign color {c} -> {color}")

                if self.graph_color_util(m, color, v + 1):

                    return True

                color[v] = 0

        return False

    def graph_coloring(self, m):
        color = [0] * self.V
        if not self.graph_color_util(m, color, 0):
          print("Solution does not exist")
          return False
        print("Solution exists:")
        for idx, col in enumerate(color):
          print(f"Vertex {idx+1} -> Color {col}")
g = Graph(4)
g.graph = [[0, 1, 1, 1],
          [1, 0, 1, 0],
          [1, 1, 0, 1],
          [1, 0, 1, 0]]
num_colors = 3
g.graph_coloring(num_colors)
```

OUTPUT:



```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/graph colourin
    g.py
    Vertex 1: Assign color 1 -> [1, 0, 0, 0]
    Vertex 2: Assign color 2 -> [1, 2, 0, 0]
    Vertex 3: Assign color 3 -> [1, 2, 3, 0]
    Vertex 4: Assign color 2 -> [1, 2, 3, 2]
    Solution exists:
    Vertex 1 -> Color 1
    Vertex 2 -> Color 2
    Vertex 3 -> Color 3
    Vertex 4 -> Color 2
>>>
```