

## CSA0888. - PYTHON PROGRAMMING

### Assignment - 3

**Name: V.MANOKARTICK**

**Reg.no: 192225077**

#### **Program 1:**

```
def maxProfit(price,n):

    profit = [0]*n

    max_price = price[n-1]

    for i in range(n-2,0,-1):

        if price[i] > max_price:
            max_price = price[i]

    profit[i] = max(profit[i+1], max_price-price[i])

    min_price = price[0]

    for i in range(1,n):

        if price[i] < min_price:
            min_price = price[i]

    profit[i] = max(profit[i-1], profit[i]+(price[i]-min_price))

    result = profit[n-1]

    return result

price = [2,7,9,56,7,4]
print("Maximum profit is",maxProfit(price,len(price)))
```

## Program 2:

```
from itertools import permutations

comb = permutations([4, 5, 6], 3)

for i in comb:

    print(i)
```

## Program. 3

```
def solve(nums):
    count=0
    n=len(nums)
    for i in range(n):
        for j in range(i+1,n):
            if nums[i] == nums[j]:
                count+=1
    return count
```

```
nums = [5,6,7,5,5,7]
print(solve(nums))
```

## Program. 4

```
def add_binary_nums(x, y):
    max_len = max(len(x), len(y))

    x = x.zfill(max_len)
    y = y.zfill(max_len)

    result = ""

    carry = 0

    for i in range(max_len-1, -1, -1):
        r = carry
        r += 1 if x[i] == '1' else 0
        r += 1 if y[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result
        carry = 0 if r < 2 else 1
```

```
if carry!=0: result = '1'+ result
```

```
return result.zfill(max_len)
```

```
print(add_binary_nums('1101','100'))
```

## Program 5

```
def minJumps(arr,n):
```

```
    if (n <= 1):  
        return 0
```

```
    if (arr[0] == 0):  
        return -1
```

```
    jump = 1
```

```
    subArrEndIndex = arr[0]
```

```
    i = 1  
    subArrFistHalfMaxSteps = 0  
    subArrSecondHalfMaxSteps = 0
```

```
    for i in range(1,n):
```

```
        subArrEndIndex = i + subArrEndIndex
```

```
        if (subArrEndIndex >= n):  
            return jump
```

```
        firstHalfMaxStepIndex = 0
```

```
        j = i
```

```
        for j in range(i, subArrEndIndex):
```

```
            stepsCanCover = arr[j] + j
```

```
            if (subArrFistHalfMaxSteps < stepsCanCover):
```

```
                subArrFistHalfMaxSteps = stepsCanCover
```

```
                subArrSecondHalfMaxSteps = 0
```

```
                firstHalfMaxStepIndex = j
```

```
            elif (subArrSecondHalfMaxSteps < stepsCanCover):
```

```
                subArrSecondHalfMaxSteps = stepsCanCover
```

```
        i = j
```

```
    if (i > subArrFistHalfMaxSteps):
```

```
        return -1
```

```
    jump += 1
```

```
        subArrEndIndex = arr[firstHalfMaxStepIndex]
        subArrFistHalfMaxSteps = subArrSecondHalfMaxSteps
    return -1
```

```
if __name__ == '__main__':
```

```
    arr = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]
    size = len(arr)
```

```
    print("Minimum number of jumps to reach end is", minJumps(arr, size))
```

### **Program 6**

```
sentence = input("Enter a sentence:")
```

```
s = sentence.split()
```

```
s.sort()
```

```
result = []
```

```
n = len(s)
```

```
for i in range(n):
    for j in range(i+1, n):
        two_words = s[i] + " " + s[j]
        result.append(two_words)
```

```
for item in result:
    print(item)
```

### **Program 7**

```
def backtrack():
    global ans, curr, visited, nums
```

```
    if (len(curr) == len(nums)):
        print(*curr)
```

```
    for i in range(len(nums)):
```

```
        if (visited[i]):
            continue
```

```
        if (i > 0 and nums[i] == nums[i - 1] and visited[i - 1] == False):
            continue
        visited[i] = True
```

```
curr.append(nums[i])
```

```
backtrack()
```

```
visited[i]  
visited[i] = False
```

```
del curr[-1]
```

```
def permuteDuplicates(nums):  
    global ans, visited, curr  
    nums = sorted(nums)
```

```
    backtrack()  
    return ans
```

```
def getDistinctPermutations(nums):  
    global ans, curr, visited
```

```
    ans = permuteDuplicates(nums)
```

```
if __name__ == '__main__':  
    visited = [False]*(5)  
    ans, curr = [], []  
    nums = [1, 2, 3]  
    getDistinctPermutations(nums)
```

## Program 8

```
from collections import Counter, defaultdict  
user_input = ["cat", "dog", "tac", "edoc", "god", "tacact",  
              "act", "code", "deno", "node", "ocde", "done", "catcat"]
```

```
def solve(words: list) -> list:
```

```
    m = defaultdict(list)
```

```
    for word in words:
```

```
        frozenset(dict(Counter('cat')).items()):
```

```
            hash(frozenset(Counter('cat'))) is equal to  
            frozenset({'c', 1}, {'a', 1}, {'t', 1})  
            m[frozenset(dict(Counter(word)).items())].append(word)
```

```
return [v for k, v in m.items()]
```

```
print(solve(user_input))
```

## Program 9

```
def finding(s, p, n, m):
    # return 1 if n and m are negative
    if n < 0 and m < 0:
        return 1

    # return 0 if m is negative
    if m < 0:
        return 0

    # return n if n is negative
    if n < 0:
        # while m is positive
        while m >= 0:
            if p[m] != '*':
                return 0
            m -= 1
        return 1

    # if dp state is not visited
    if dp[n][m] == -1:
        if p[m] == '*':
            dp[n][m] = finding(s, p, n - 1, m) or finding(s, p, n, m - 1)
            return dp[n][m]
        else:
            if p[m] != s[n] and p[m] != '?':
                dp[n][m] = 0
                return dp[n][m]
            else:
                dp[n][m] = finding(s, p, n - 1, m - 1)
                return dp[n][m]

    # return dp[n][m] if dp state is previsited
    return dp[n][m]

def isMatch(s, p):
    global dp
    dp = []

    # resize the dp array
    for i in range(len(s) + 1):
        dp.append([-1] * (len(p) + 1))
    dp[len(s)][len(p)] = finding(s, p, len(s) - 1, len(p) - 1)
```

```
return dp[len(s)][len(p)]
```

```
def main():  
    s = "baaabab"  
    p = "*****ba*****ab"
```

```
    if isMatch(s,p):  
        print("Yes")  
    else:  
        print("No")
```

```
if __name__ == "__main__":  
    main()
```

### Program. 10

```
def editDistance(str1,str2,m,n):  
  
    if m == 0:  
        return n  
  
    if n == 0:  
        return m  
  
    if str1[m-1] == str2[n-1]:  
        return editDistance(str1, str2, m-1, n-1)  
  
    return 1 + min(editDistance(str1, str2, m, n-1),  
                    editDistance(str1, str2, m-1, n),  
                    editDistance(str1, str2, m-1, n-1)  
                    )  
  
str1 = "cut"  
str2 = "cat"  
print(editDistance(str1,str2,len(str1),len(str2)))
```