Started on	Friday, 16 May 2025, 11:25 AM
State	Finished
Completed on	Saturday, 24 May 2025, 9:25 AM
Time taken	7 days 22 hours
Overdue	7 days 20 hours
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Create a python program to for the following problem statement.

You are given an `n x n grid` representing a field of cherries, each cell is one of three possible integers.

- `0` means the cell is empty, so you can pass through,
- `1` means the cell contains a cherry that you can pick up and pass through, or
- `-1` means the cell contains a thorn that blocks your way.

Return *the maximum number of cherries you can collect by following the rules below*:

- Starting at the position `(0, 0)` and reaching `(n - 1, n - 1)` by moving right or down through valid path cells (cells with value `0` or `1`).
- After reaching `(n - 1, n - 1)`, returning to `(0, 0)` by moving left or up through valid path cells.
- When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell `0`.
- If there is no valid path between `(0, 0)` and `(n - 1, n - 1)`, then no cherries can be collected.

For example:

Test	Result
obj.cherryPickup(grid)	5

Answer: (penalty regime: 0 %)

Reset answer

```
class Solution:
    def cherryPickup(self, grid):
        n = len(grid)
        dp = [[0]*n for _ in range(n)]
        for i in range(n-1,-1,-1):
            for j in range(n-1, -1, -1):
                if i==n-1 and j==n-1:
                    dp[i][j] = grid[i][j]
                elif i==n-1:
                    dp[i][j] = grid[i][j]+dp[i][j+1]
                elif j==n-1:
                    dp[i][j] = grid[i][j]+dp[i+1][j]
                else:
                    dp[i][j] = grid[i][j]+max(dp[i][j+1], dp[i+1][j])

        return max(0,dp[0][0])+1
obj=Solution()
grid=[[0,1,-1],[1,0,-1],[1,1,1]]
print(obj.cherryPickup(grid))
```

	Test	Expected	Got	
✔	obj.cherryPickup(grid)	5	5	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 20.00 out of 20.00

LONGEST PALINDROMIC SUBSEQUENCE

Given a sequence, find the length of the longest palindromic subsequence in it.

For example:

Input	Result
ABBDCACB	The length of the LPS is 5

Answer: (penalty regime: 0 %)

```
def Lps(X):
    n=len(X)
    dp=[[0 for _ in range(n)] for _ in range(n)]
    for x in range(n):
        dp[x][x]=1
    for l in range(2,n+1):
        for i in range(n-l+1):
            j=i+l-1
            if X[i]==X[j]:
                dp[i][j]=dp[i+1][j-1]+2
            else:
                dp[i][j]=max(dp[i+1][j],dp[i][j-1])
    return dp[0][n-1]
X=input()
print("The length of the LPS is",Lps(X))
```

	Input	Expected	Got	
✔	ABBDCACB	The length of the LPS is 5	The length of the LPS is 5	✔
✔	BBABCBCAB	The length of the LPS is 7	The length of the LPS is 7	✔
✔	cbbd	The length of the LPS is 2	The length of the LPS is 2	✔
✔	abbab	The length of the LPS is 4	The length of the LPS is 4	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Create a python program using dynamic programming for 0/1 knapsack problem.

For example:

Test	Input	Result
knapSack(W, wt, val, n)	3 3 50 60 100 120 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 220

Answer: (penalty regime: 0 %)

Reset answer

```
def knapSack(W, wt, val, n):
    if n == 0 or W == 0 :
        return 0
    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)
    else:
        return max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1), knapSack(W, wt, val, n-1))

x=int(input())
y=int(input())
W=int(input())
val=[]
wt=[]
for i in range(x):
    val.append(int(input()))
for y in range(y):
    wt.append(int(input()))

n = len(val)
print('The maximum value that can be put in a knapsack of capacity W is: ',knapSack(W, wt, val, n))
```

	Test	Input	Expected	Got	
✔	knapSack(W, wt, val, n)	3 3 50 60 100 120 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 220	The maximum value that can be put in a knapsack of capacity W is: 220	✔
✔	knapSack(W, wt, val, n)	3 3 40 50 90 110 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 160	The maximum value that can be put in a knapsack of capacity W is: 160	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

Create a python program using brute force method of searching for the given substring in the main string.

For example:

Test	Input	Result
match(str1,str2)	AABAACAADAABAABA AABA	Found at index 0 Found at index 9 Found at index 12

Answer: (penalty regime: 0 %)

Reset answer

```
import re
def match(string,sub):
    pattern=re.compile(str2)
    r=pattern.search(str1)
    while r:
        print("Found at index {}".format(r.start()))
        r=pattern.search(str1,r.start()+1)
str1=input()
str2=input()
```

	Test	Input	Expected	Got	
✔	match(str1,str2)	AABAACAADAABAABA AABA	Found at index 0 Found at index 9 Found at index 12	Found at index 0 Found at index 9 Found at index 12	✔
✔	match(str1,str2)	saveetha savee	Found at index 0	Found at index 0	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Given a 2D matrix **tsp[][]**, where each row has the array of distances from that indexed city to all the other cities and **-1** denotes that there doesn't exist a path between those two indexed cities. The task is to print minimum cost in TSP cycle.

tsp[][] = {{-1, 30, 25, 10},
{15, -1, 20, 40},
{10, 20, -1, 25},
{30, 10, 20, -1}};

Answer: (penalty regime: 0 %)

Reset answer

```
def tsp_cost(tsp):
    return min(sum(tsp[i][j] for i, j in zip(path, path[1:] + path[:1])) for path in permutations(rang

from itertools import permutations
tsp = [[-1, 30, 25, 10], [15, -1, 20, 40], [10, 20, -1, 25], [30, 10, 20, -1]]
print("Minimum Cost is :",tsp_cost(tsp))
```

	Expected	Got	
✔	`Minimum Cost is : 50`	`Minimum Cost is : 50`	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.