Started on	Tuesday, 27 May 2025, 1:38 PM
State	Finished
Completed on	Tuesday, 27 May 2025, 2:03 PM
Time taken	24 mins 58 secs
Grade	**80.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Create a python program to find the maximum value in linear search.

For example:

Test	Input	Result
`find_maximum(test_scores)`	10 88 93 75 100 80 67 71 92 90 83	`Maximum value is  100`

Answer: (penalty regime: 0 %)

Reset answer

```python
def find_maximum(lst):
    max=None
    for i in lst:
        if(max==None or i>max):
            max=i
    return max

test_scores = []
n=int(input())
for i in range(n):
    test_scores.append(int(input()))
print("Maximum value is ",find_maximum(test_scores))
```

	Test	Input	Expected	Got	
✓	`find_maximum(test_scores)`	10 88 93 75 100 80 67 71 92 90 83	`Maximum value is  100`	`Maximum value is  100`	✓

	Test	Input	Expected	Got	
✓	`find_maximum(test_scores)`	5 45 86 95 76 28	`Maximum value is  95`	`Maximum value is  95`	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **2**

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

```python
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`	✓

	Test	Input	Expected	Got	
✓	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 **3**

Correct

Mark 20.00 out of 20.00

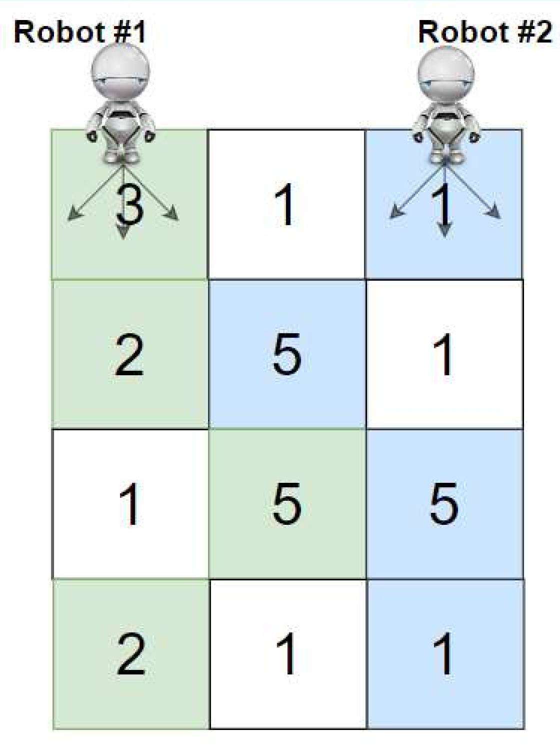
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

You have two robots that can collect cherries for you:

- **Robot #1** is located at the **top-left corner** `(0, 0)`, and
- **Robot #2** is located at the **top-right corner** `(0, cols - 1)`.

Return *the maximum number of cherries collection using both robots by following the rules below*:

- From a cell `(i, j)`, robots can move to cell `(i + 1, j - 1)`, `(i + 1, j)`, or `(i + 1, j + 1)`.
- When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.
- When both robots stay in the same cell, only one takes the cherries.
- Both robots cannot move outside of the grid at any moment.
- Both robots should reach the bottom row in `grid`.



3	1	1
2	5	1
1	5	5
2	1	1

For example:

Test	Result
`ob.cherryPickup(grid)`	24

Answer: (penalty regime: 0 %)

Reset answer

```
class CherryPickup:
    def cherryPickup(self, grid):
        from functools import lru_cache
        rows, cols = len(grid), len(grid[0])

        @lru_cache(None)
        def dp(r, c1, c2):
            if c1 < 0 or c1 >= cols or c2 < 0 or c2 >= cols:
                return 0
            cherries = grid[r][c1]
            if c1 != c2:
                cherries += grid[r][c2]
            if r != rows - 1:
                max_cherries = 0
                for new_c1 in [c1-1, c1, c1+1]:
                    for new_c2 in [c2-1, c2, c2+1]:
                        max_cherries = max(max_cherries, dp(r+1, new_c1, new_c2))
                cherries += max_cherries
            return cherries

        return dp(0, 0, cols - 1)

```

	Test	Expected	Got	
✓	ob.cherryPickup(grid)	24	24	✓

Passed all tests! ✓

Correct

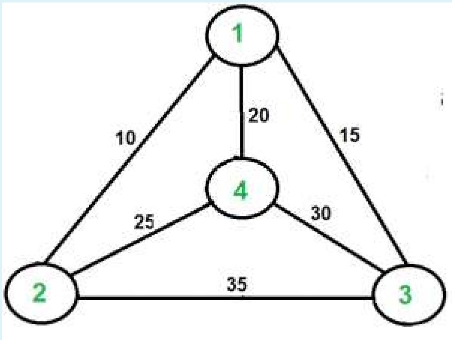
Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

Solve Travelling Sales man Problem for the following graph



Answer: (penalty regime: 0 %)

Reset answer

```
import sys
from itertools import permutations

def tsp(graph, s):
    V = len(graph)
    vertex = [i for i in range(V) if i != s]

    min_path = sys.maxsize
    next_permutation = permutations(vertex)

    for perm in next_permutation:
        current_pathweight = 0
        k = s
        for j in perm:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)

    print(min_path)


```

	Expected	Got	
✓	80	80	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Incorrect

Mark 0.00 out of 20.00

Create a Python program to find longest common substring or subword (LCW) of two strings using dynamic programming with top-down approach or memoization.

Problem Description

A string r is a substring or subword of a string s if r is contained within s. A string r is a common substring of s and t if r is a substring of both s and t. A string r is a longest common substring or subword (LCW) of s and t if there is no string that is longer than r and is a common substring of s and t. The problem is to find an LCW of two given strings.

For example:

Test	Input	Result
`lcw(u, v)`	`potato` `tomato`	`Longest Common Subword: ato`

Answer: (penalty regime: 0 %)

Reset answer

```python
def lcw(s1, s2):
    from functools import lru_cache
    n, m = len(s1), len(s2)
    dp = [[0] * (m + 1) for _ in range(n + 1)]
    max_len = 0
    end_pos = 0

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
                if dp[i][j] > max_len:
                    max_len = dp[i][j]
                    end_pos = i

    print("Longest Common Subword:", s1[end_pos - max_len:end_pos])

lcw("potato", "tomato")
```

	Test	Input	Expected	Got	

✗	`lcw(u, v)`	`potato`
		`tomato`
			`Longest Common Subword: ato`	`Longest Common Subword: ato`
				
				`***Run error***`
				`Traceback (most recent call last):`
				`  File "__tester__.python3", line 41, in <module>`
				`    lcw(u, v)`
				`NameError: name 'u' is not defined`
					✗

Testing was aborted due to error.
Your code must pass all tests to earn any marks. Try again.

Show differences

Incorrect

Marks for this submission: 0.00/20.00.