Started on	Saturday, 10 May 2025, 3:00 PM
State	Finished
Completed on	Saturday, 10 May 2025, 3:05 PM
Time taken	40 mins 13 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

```

def find_maximum(lst):
    if len(lst)==0:
        return 0
    max=lst[0]
    for i in lst:
        if 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):
    ########## Add your code here #########
    if n==0 or W==0:
        return 0
    if wt[n-1]>W:
        return knapSack(W,wt,val,n-1)
    else:
        inc=val[n-1]+knapSack(W-wt[n-1],wt,val,n-1)
        exc=knapSack(W,wt,val,n-1)
        return max(inc,exc)

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()))

```

	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**

Not answered

Mark 0.00 out of 20.00

Write a Python program to sort unsorted numbers using Multi-key quicksort

For example:

Test	Input	Result
`quick_sort_3partition(nums, 0, len(nums)-1)`	5 4 3 5 1 2	Original list: [4, 3, 5, 1, 2] After applying Random Pivot Quick Sort the said list becomes: [1, 2, 3, 4, 5]
`quick_sort_3partition(nums, 0, len(nums)-1)`	6 21 10 3 65 4 8	Original list: [21, 10, 3, 65, 4, 8] After applying Random Pivot Quick Sort the said list becomes: [3, 4, 8, 10, 21, 65]

Answer: (penalty regime: 0 %)

Question **4**

Correct

Mark 20.00 out of 20.00

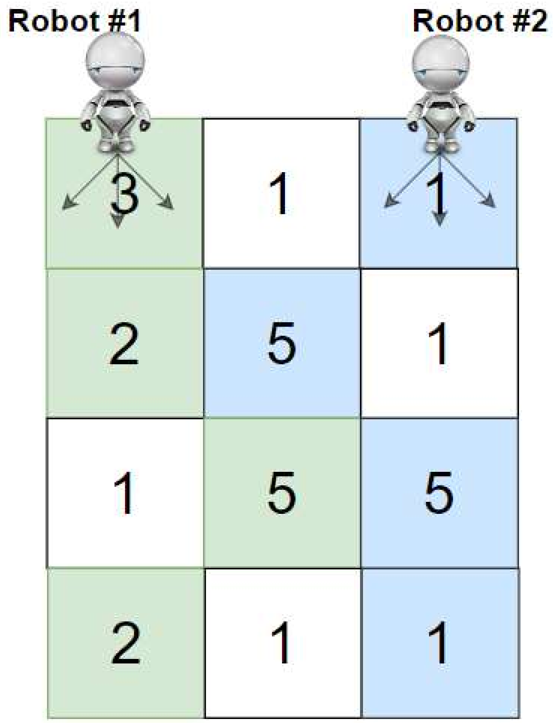
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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 Solution(object):
    def cherryPickup(self, grid):
        rows = len(grid)
        cols = len(grid[0])
        memo={}
        def dp(r,c1,c2):
            if r==rows or c1<0 or c1==cols or c2<0 or c2==cols:
                return 0
            if (r,c1,c2) in memo:
                return memo[(r,c1,c2)]
            cherries=grid[r][c1]+(grid[r][c2] if c1!=c2 else 0)
            maxcherries=0
            for dc1 in [-1,0,1]:
                for dc2 in [-1,0,1]:
                    maxcherries=max(maxcherries,dp(r+1,c1+dc1,c2+dc2))
            result=cherries+maxcherries
            memo[(r,c1,c2)]=result
            return result
            #########  Add your code here ##########

```

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

Passed all tests! ✓

Correct

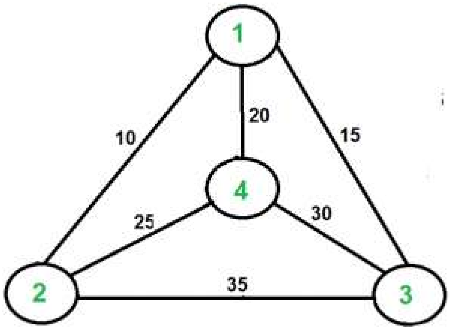
Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Solve Travelling Sales man Problem for the following graph



Answer: (penalty regime: 0 %)

Reset answer

```

from sys import maxsize
from itertools import permutations
V = 4


def travellingSalesmanProblem(graph, s):
    vetex=[]
    cur=0
    minpath=maxsize
    for i in range(V):
        if i!=s:
            vetex.append(i)
    # k=s
    nextper=permutations(vetex)
    for i in nextper:
        cur=0
        k=s
        for j in i:
            cur+=graph[k][j]
            k=j
        cur+=graph[k][s]
```

	Expected	Got	
✓	80	80	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.