Started on	Friday, 3 January 2025, 3:17 PM
State	Finished
Completed on	Friday, 3 January 2025, 3:34 PM
Time taken	16 mins 14 secs
Grade	**80.00** out of 100.00

Question **1**

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

```python
def match(string,sub):
    l = len(string)
    ls = len(sub)
    start = sub[0]
    for i in range(l-ls+1):
        if string[i:i+ls]==sub:
            print(f"Found at index {i}")

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 **2**

Correct

Mark 20.00 out of 20.00

Create a python program for 0/1 knapsack problem using naive recursion method

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):
    K=[[0 for x in range(W+1)] for x in range(n+1)]
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w]=0
            elif wt[i-1]<=w:
                K[i][w]=max(val[i-1]+K[i-1][w-wt[i-1]],K[i-1][w])
            else:
                K[i][w]=K[i-1][w]
    return K[n][w]


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	✓
✓	knapSack(W, wt, val, n)	3 3 55 65 115 125 15 25 35	The maximum value that can be put in a knapsack of capacity W is: 190	The maximum value that can be put in a knapsack of capacity W is: 190	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Incorrect

Mark 0.00 out of 20.00

Write a recursive python function to perform merge sort on the unsorted list of values.

For example:

Test	Input	Result
`mergesort(li)`	5 21 31 47 9 6	`[6, 9, 21, 31, 47]`
`mergesort(li)`	6 84 21 56 3 2 7	`[2, 3, 7, 21, 56, 84]`

Answer: (penalty regime: 0 %)

```
mergesort(li)
{

}
```

	Test	Input	Expected	Got	

✗	`mergesort(li)`	5
		21
		31
		47
		9
		6
			`[6, 9, 21, 31, 47]`	`***Run error***`
				`Traceback (most recent call last):`
				`  File "__tester__.python3", line 1, in <module>`
				`    mergesort(li)`
				`NameError: name 'mergesort' 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.

Question **4**

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

```
from sys import maxsize
from itertools import permutations
V = 4


def travellingSalesmanProblem(graph, s):

   #Write your code
    v=[]
    for i in range(V):
        if i!=s:
            v.append(i)
    mp=maxsize
    np=permutations(v)
    for i in np:
        k=s
        cp=0
        for j in i:
            cp+=tsp[k][j]
            k=j
        cp+=tsp[k][s]
        mp=min(cp,mp)
```

	Expected	Got	
✓	Minimum Cost is : 50	Minimum Cost is : 50	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

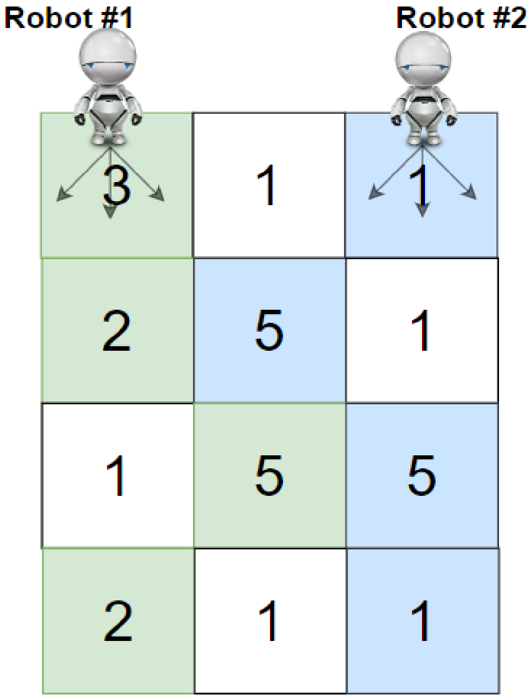
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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

```python
class Solution(object):
    def cherryPickup(self, grid):
        def dp(i, j, k):
            if (i, j, k) in memo:
                return memo[(i, j, k)]
            if i == ROW_NUM - 1:
                return grid[i][j] + (grid[i][k] if j != k else 0)
            cherries=grid[i][j] + (grid[i][k] if j != k else 0)
            max_cherries=0
            for dj in [-1, 0, 1]:
                for dk in [-1, 0, 1]:
                    next_j, next_k = j + dj , k + dk
                    if 0 <= next_j < COL_NUM and 0 <= next_k < COL_NUM:
                        max_cherries= max(max_cherries ,dp(i + 1, next_j, next_k))
            memo[(i, j, k)]= cherries + max_cherries
            return memo[(i, j, k)]


        ROW_NUM = len(grid)
        COL_NUM = len(grid[0])
        memo={}
```

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

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.