Started on	Wednesday, 14 May 2025, 3:14 PM
State	Finished
Completed on	Wednesday, 14 May 2025, 4:19 PM
Time taken	1 hour 4 mins
Grade	**80.00** out of 100.00

Question **1**

Incorrect

Mark 0.00 out of 20.00

Given a string s, return *the longest palindromic substring* in s.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

For example:

Test	Input	Result
ob1.longestPalindrome(str1)	ABCBCB	BCBCB

Answer: (penalty regime: 0 %)

Reset answer

```
class Solution(object):
   def longestPalindrome(self, s):
       ############  Add your code here ##############
ob1 = Solution()
str1=input()
print(ob1.longestPalindrome(str1))
```

Syntax Error(s)

```
Sorry: IndentationError: expected an indented block (__tester__.python3, line 4)
```

Incorrect

Marks for this submission: 0.00/20.00.

Question **2**

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 **3**

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)
        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

```

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

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 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

```
1
2
3  def knapSack(W, wt, val, n):
4      if W==0 or n==0:
5          return 0
6      if wt[n-1]>W:
7          return knapSack(W,wt,val,n-1)
8      else:
9          inc=val[n-1]+knapSack(W-wt[n-1],wt,val,n-1)
10         exc=knapSack(W,wt,val,n-1)
11         return max(inc,exc)
12
13
14 x=int(input())
15 y=int(input())
16 W=int(input())
17 val=[]
18 wt=[]
19 for i in range(x):
20     val.append(int(input()))
21 for y in range(y):
22     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 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 **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

```
from typing import DefaultDict

INT_MAX = 2147483647

def findMinRoute(tsp):
    sum = 0
    counter = 0
    j = 0
    i = 0
    min = INT_MAX
    visitedRouteList = DefaultDict(int)

    visitedRouteList[0] = 1
    route = [0] * len(tsp)

    while i < len(tsp) and j < len(tsp[i]):
        if counter >= len(tsp[i]) - 1:
            break
        if j != i and visitedRouteList[j] == 0:
            if tsp[i][j] < min:
```

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

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.