Started on	Tuesday, 15 April 2025, 2:37 PM
State	Finished
Completed on	Tuesday, 15 April 2025, 6:11 PM
Time taken	3 hours 33 mins
Overdue	1 hour 33 mins
Grade	**80.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Create a python program to find the longest common subsequence using Memoization Implementation.

For example:

Input	Result
AGGTAB GXTXAYB	Length of LCS is 4

Answer: (penalty regime: 0 %)

```
def lcs(str1, str2):
    m, n = len(str1), len(str2)
    table = [[0] * (n+1) for _ in range(m+1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if str1[i-1] == str2[j-1]:
                table[i][j] = 1 + table[i-1][j-1]
            else:
                table[i][j] = max(table[i-1][j], table[i][j-1])
    lcs = ""
    i, j = m, n
    while i > 0 and j > 0:
        if str1[i-1] == str2[j-1]:
            lcs = str1[i-1] + lcs
            i -= 1
            j -= 1
        elif table[i-1][j] > table[i][j-1]:
            i -= 1
        else:
            j -= 1
    return lcs
str1=input()
```

	Input	Expected	Got	
✔	AGGTAB GXTXAYB	Length of LCS is 4	Length of LCS is 4	✔
✔	SAMPLE SAEMSUNG	Length of LCS is 3	Length of LCS is 3	✔
✔	saveetha sabeetha	Length of LCS is 7	Length of LCS is 7	✔

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 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

```
def lcw(u, v):
    m, n = len(u), len(v)
    table = [[0] * (n+1) for _ in range(m+1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if u[i-1] == v[j-1]:
                table[i][j] = 1 + table[i-1][j-1]
            else:
                table[i][j] = max(table[i-1][j], table[i][j-1])
    lcw = ""
    i, j = m, n
    while i > 0 and j > 0:
        if u[i-1] == v[j-1]:
            lcw = u[i-1] + lcw
            i -= 1
            j -= 1
        elif table[i][j] >= table[i][j]:
            i -= 1
        else:
            j -= 1
    return lcw
u=input()
```

	Test	Input	Expected	Got	
✔	lcw(u, v)	potato tomato	Longest Common Subword: ato	Longest Common Subword: ato	✔
✔	lcw(u, v)	snakegourd bottlegourd	Longest Common Subword: egourd	Longest Common Subword: egourd	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 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 ##############
        dp = [[False for i in range(len(s))] for i in range(len(s))]
        for i in range(len(s)):
            dp[i][i] = True
        max_length = 1
        start = 0
        for l in range(2,len(s)+1):
            for i in range(len(s)-l+1):
                end = i+l
                if l==2:
                    if s[i] == s[end-1]:
                        dp[i][end-1]=True
                        max_length = l
                        start = i
                else:
                    if s[i] == s[end-1] and dp[i+1][end-2]:
                        dp[i][end-1]=True
                        max_length = l
                        start = i
        return s[start:start+max_length]
```

	Test	Input	Expected	Got	
✔	ob1.longestPalindrome(str1)	ABCBCB	BCBCB	BCBCB	✔
✔	ob1.longestPalindrome(str1)	BABAD	ABA	ABA	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

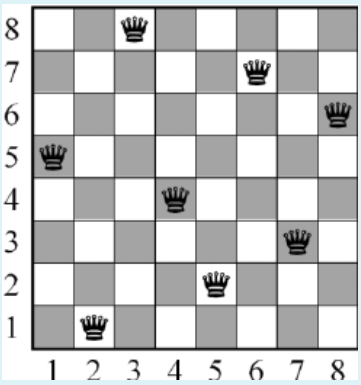
Question **4**

Not answered

Mark 0.00 out of 20.00

You are given an integer **N**. For a given **N** x **N** chessboard, find a way to place **'N'** queens such that no queen can attack any other queen on the chessboard.

A queen can be attacked when it lies in the same row, column, or the same diagonal as any of the other queens. **You have to print one such configuration**.



Note :

Get the input from the user for N . The value of N must be from 1 to 8

If solution exists Print a binary matrix as output that has 1s for the cells where queens are placed

If there is no solution to the problem print "Solution does not exist"

For example:

Input	Result
5	1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0

Answer: (penalty regime: 0 %)

```
1
```

Question **5**

Correct

Mark 20.00 out of 20.00

Create a python program to find the Edit distance between two strings using dynamic programming.

For example:

Input	Result
Cats Rats	No. of Operations required : 1

Answer: (penalty regime: 0 %)

Reset answer

```
def LD(s, t):
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    if s[-1] == t[-1]:
        cost = 0
    else:
        cost = 1
    res = min([LD(s[:-1], t)+1, LD(s, t[:-1])+1, LD(s[:-1], t[:-1]) + cost])
    return res
str1=input()
str2=input()
print("No. of Operations required :",LD(str1,str2))
```

	Input	Expected	Got	
✔	Cats Rats	No. of Operations required : 1	No. of Operations required : 1	✔
✔	Saturday Sunday	No. of Operations required : 3	No. of Operations required : 3	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.