Started on	Tuesday, 1 October 2024, 3:03 PM
State	Finished
Completed on	Tuesday, 1 October 2024, 3:13 PM
Time taken	9 mins 48 secs
Grade	**80.00** out of 100.00

Question **1**

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 bottom-up approach.

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)	bisect trisect	Longest Common Subword: isect

Answer: (penalty regime: 0 %)

Reset answer

```
def lcw(X,Y):
    m = len(X)
    n = len(Y)
    maxLength = 0
    endingIndex = m
    lookup = [[0 for x in range(n + 1)] for y in range(m + 1)]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:
                lookup[i][j] = lookup[i - 1][j - 1] + 1
                if lookup[i][j] > maxLength:
                    maxLength = lookup[i][j]
                    endingIndex = i
    return X[endingIndex - maxLength: endingIndex]

u = input()
v = input()
print("Longest Common Subword:", lcw(u,v))
```

	Test	Input	Expected	Got	
✔	lcw(u, v)	bisect trisect	Longest Common Subword: isect	Longest Common Subword: isect	✔
✔	lcw(u, v)	director conductor	Longest Common Subword: ctor	Longest Common Subword: ctor	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **2**

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):
      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]
ob1 = Solution()
```

	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 **3**

Correct

Mark 20.00 out of 20.00

To Write a Python Program to find longest common subsequence using Dynamic Programming

For example:

Input	Result
abcbdab bdcaba	bdab

Answer: (penalty regime: 0 %)

```
def lcs(u, v):
    """Return c where c[i][j] contains length of LCS of u[i:] and v[j:]."""
    c = [[-1]*(len(v) + 1) for _ in range(len(u) + 1)]
    for i in range(len(u) + 1):
        c[i][len(v)] = 0
    for j in range(len(v)):
        c[len(u)][j] = 0

    for i in range(len(u) - 1, -1, -1):
        for j in range(len(v) - 1, -1, -1):
            if u[i] == v[j]:
                c[i][j] = 1 + c[i + 1][j + 1]
            else:
                c[i][j] = max(c[i + 1][j], c[i][j + 1])
    return c

def print_lcs(u, v, c):
    """Print one LCS of u and v using table c."""
    i = j = 0
    while not (i == len(u) or j == len(v)):
        if u[i] == v[j]:
            print(u[i], end='')
```

	Input	Expected	Got	
✔	abcbdab bdcaba	bdab	bdab	✔
✔	treehouse elephant	eeh	eeh	✔

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 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.

Question **5**

Not answered

Mark 0.00 out of 20.00

Write a Python Program to find minimum number of swaps required to sort an float array given by the user.

For example:

Test	Input	Result
`minSwaps(arr)`	5 2.3 6.5 4.1 9.5 7.5	2
`minSwaps(arr)`	6 3.2 1.4 5.6 9.2 4.5 6.2	4

Answer: (penalty regime: 0 %)

```
1
```