Started on	Friday, 16 May 2025, 11:30 AM
State	Finished
Completed on	Friday, 16 May 2025, 7:25 PM
Time taken	7 hours 55 mins
Overdue	5 hours 55 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 palindromic substring using Brute force method in a given string.

For example:

Input	Result
mojologiccigolmojo	logiccigol

Answer: (penalty regime: 0 %)

Reset answer

```
def printSubStr(str, low, high):

    for i in range(low, high + 1):
        print(str[i], end = "")

def longestPalindrome(str):
    n=len(str)
    max_len=0
    start=0
    for i in range(n):
        for j in range(1,n):
            s=str[i:j+1]
            if s==s[::-1]:
                cur=j-i+1
                if cur>max_len:
                    max_len=cur
                    start=i
    printSubStr(str, start, start + max_len - 1)

if __name__ == '__main__':

    str = input()
```

	Input	Expected	Got	
✔	mojologiccigolmojo	logiccigol	logiccigol	✔
✔	sampleelpams	pleelp	pleelp	✔

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 the longest common subsequence using Memoization Implementation.

For example:

Input	Result
AGGTAB GXTXAYB	Length of LCS is 4

Answer: (penalty regime: 0 %)

```
def lcs(x,y,m,n,dp):
    if m==0 or n==0:
        return 0
    if dp[m][n]!=-1:
        return dp[m][n]
    if x[m-1]==y[n-1]:
        dp[m][n]=1+lcs(x,y,m-1,n-1,dp)
        return dp[m][n]
    dp[m][n]=max(lcs(x,y,m,n-1,dp),lcs(x,y,m-1,n,dp))
    return  dp[m][n]
x=input()
y=input()
dp=[[-1]*(len(y)+1) for i in range(len(x)+1)]
print("Length of LCS is",lcs(x,y,len(x),len(y),dp))
```

	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 **3**

Correct

Mark 20.00 out of 20.00

Create a Naive recursive python program to find the minimum number of operations to convert str1 to str2

For example:

Input	Result
Python Peithen	Edit Distance 3

Answer: (penalty regime: 0 %)

Reset answer

```
def ed(x,y,m,n):
    if m==0:
        return n
    if n==0:
        return m
    if x[m-1]==y[n-1]:
        return ed(x,y,m-1,n-1)
    return 1+min(ed(x,y,m-1,n-1),ed(x,y,m,n-1),ed(x,y,m-1,n))
x=input()
y=input()
print("Edit Distance",ed(x,y,len(x),len(y)))
```

	Input	Expected	Got	
✔	Python Peithen	Edit Distance 3	Edit Distance 3	✔
✔	food money	Edit Distance 4	Edit Distance 4	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

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

For example:

Test	Input	Result
mergesort(li)	5 3.2 1.5 1.6 1.7 8.9	[1.5, 1.6, 1.7, 3.2, 8.9]
mergesort(li)	6 3.1 2.3 6.5 4.5 7.8 9.2	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]

Answer: (penalty regime: 0 %)

```
def mergesort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = mergesort(arr[:mid])
    right_half = mergesort(arr[mid:])

    return merge(left_half, right_half)

def merge(left, right):
    merged = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1

```

	Test	Input	Expected	Got	
✔	mergesort(li)	5 3.2 1.5 1.6 1.7 8.9	[1.5, 1.6, 1.7, 3.2, 8.9]	[1.5, 1.6, 1.7, 3.2, 8.9]	✔
✔	mergesort(li)	6 3.1 2.3 6.5 4.5 7.8 9.2	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]	✔

	Test	Input	Expected	Got	
✔	mergesort(li)	4 3.1 2.3 6.5 4.1	[2.3, 3.1, 4.1, 6.5]	[2.3, 3.1, 4.1, 6.5]	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Not answered

Mark 0.00 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):
    c = [[-1]*(len(v) + 1) for _ in range(len(u) + 1)]
    lcw_i = lcw_j = -1
    length_lcw = 0
    for i in range(len(u)):
        for j in range(len(v)):
            temp = lcw_starting_at(u, v, c, i, j)
            if length_lcw < temp:
                length_lcw = temp
                lcw_i = i
                lcw_j = j
    return length_lcw, lcw_i, lcw_j
def lcw_starting_at(u, v, c, i, j):
    ############ Add your code here ############
    return ---------------


u = input()
v = input()
length_lcw, lcw_i, lcw_j = lcw(u, v)
print('Longest Common Subword: ', end='')
if length_lcw > 0:
```