

# Challenge: Longest Palindromic Subsequence

In this lesson, we will build on the problem from the last coding challenge to solve another interesting problem.

## We'll cover the following ^

- Problem statement
- Input
- Output
- Coding challenge

## Problem statement #

Given a string, find the length of the longest common palindromic subsequence in that string. Let's first review what a palindrome is. A **palindrome** is a sequence that reads the same in either direction. For example, `madam` is a famous palindromic string. If you read it from start to end or from end to start, it reads the same.

As we have seen in the last challenge, unlike substring, subsequence does not have to be contiguous. So, in a string, `racer car`, we may not have any palindromic substring of length greater than one, but we have a few palindromic subsequences. For example, `cec`, `aceca`, or the longest of them all `racecar`.

So, you need to find the length of the longest palindromic subsequence.

## Input #

Your algorithm will take a string of variable length as input.

```
str = "racer car"
```

## Output #

Your algorithm should return an integer value representing the length of the longest palindromic subsequence in the given string.

longest palindromic subsequence in the given string.

```
LPS("racer car") = 7
```




## Coding challenge #




This problem builds on the problem of the [longest common subsequence](#) we discussed in the last lesson. Try to think about how you can define this problem in terms of the longest common subsequence problem. We have provided the function to find the length of the longest common subsequence below. You may use this function in your solution. If you are able to figure out the relation of this problem with the longest common subsequence problem, it is a really easy problem. So, think carefully and then write the code. Best of luck!

main.py

lcs.py

All code files are copied to end of the page...





Hint 1 of 1

< It's a one line solution! >

In the next lesson, we will see the solution to this problem.

## Code Files Content !!!

⏏ ⏏

```

-----
|  main.py [1]
-----

from lcs import LCS
# call longest common subsequence function as follows:
#   LCS(str1, str2)
#   returns integer for length of the longest common subsequence in str1 and str2

```

```

def LPS(str):
    # write your code here
    return 0

```

```

stressTesting = True

```

```

-----
|  lcs.py [1]
-----

```

```

def LCS(str1, str2):
    n = len(str1)    # length of str1
    m = len(str2)    # length of str1

    dp = [0 for i in range(n+1)] # table for tabulation, only maintaining state of last row

    for j in range(1, m+1):      # iterating to fill table
        thisrow = [0 for i in range(n+1)] # calculate new row (based on previous row i.e. dp)
        for i in range(1, n+1):
            if str1[i-1] == str2[j-1]:    # if characters at this position match,
                thisrow[i] = dp[i-1] + 1 # add 1 to the previous diagonal and store it in this diag
            else:
                thisrow[i] = max(dp[i], thisrow[i-1]) # if character don't match, use i-th result
        dp = thisrow # after evaluating thisrow, set dp equal to this row to be used in the next
    return dp[n]

```

```

*****

```