LONGEST COMMON SUBSEQUENCE

```
import numpy as np
```
[22] ✓ 0.0s

```
def LCS_Length(X,Y):

    # prepending a dummy character to X and Y so that the characters
    # of import begin at indexing 1
    X = '.' + X
    Y = '.' + Y

    # the new lengths of X and Y will be the number of rows in and cols in 2D array
    rows = len(X)
    cols = len(Y)

    # initializing an empty 2D array of zeros that will be used to store answers for
    # our subproblems
    c = np.zeros((rows,cols))

    for i in range(1,rows):
        for j in range(1, cols):

            if( X[i] == Y[j]):
                c[i,j] = c[i-1,j-1] + 1
            else:
                c[i,j] = max(c[i-1,j],c[i,j-1])

    # returning the lencth of the longest common subsequence in X and Y
    return c[rows-1,cols-1]
```
[23] ✓ 0.0s

```
print(LCS_Length('ABCB', 'BDCAB'))
```
[24] ✓ 0.0s

```
3.0
```

KNAPSACK PROBLEM (1/0)

```
import numpy as np
```
[4] ✓ 0.0s

```
def knapsackMaxValue(bag_capacity, items):

    # adding 1 to the max capacity of the bag and the total number of the items so that 2D
    # array indexing can begin from 0 with inital row and column of all 0's
    W = bag_capacity + 1
    I = len(items) + 1

    # creating the memoization 2D array
    c = np.zeros((I,W))

    # iterating to the max capacity of the bag, calculating the max possible value for each weight
    # (solving and stroing the solutions for each subproblem)
    for w in range (1,W):
        for i in range(1,I):
            if (w >= items[i-1][0]):
                c[i,w] = max( c[i-1,w], c[i-1, w - items[i-1][0]] + items[i-1][1])
            else:
                c[i,w] = c[i-1,w]

    return c[I-1,W-1]
```
[5] ✓ 0.0s

+ Code + Markdown

```
items = [(2,3),
         (3,4),
         (4,5),
         (5,8),
         (9,10)]
maxWeight = 15

print(knapsackMaxValue(maxWeight, items))
```
[6] ✓ 0.0s

```
20.0
```