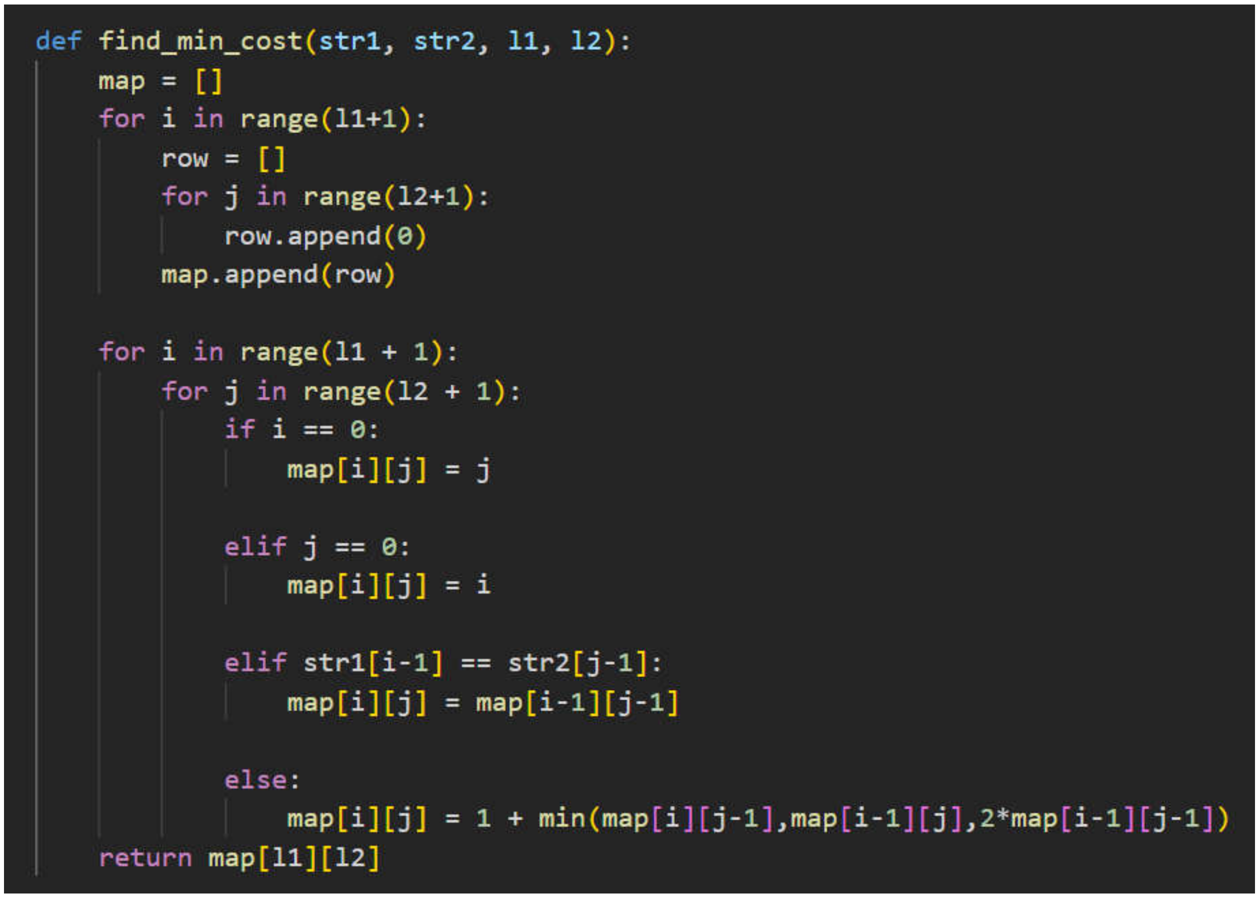
3.a.

We should model our problem as a dynamic programming problem. In this way, we should create a table with respect to length of strings as below. And assume states as location in table and our actions are remove, insert and replace. Cost of insert and delete is one but cost of replace is two.

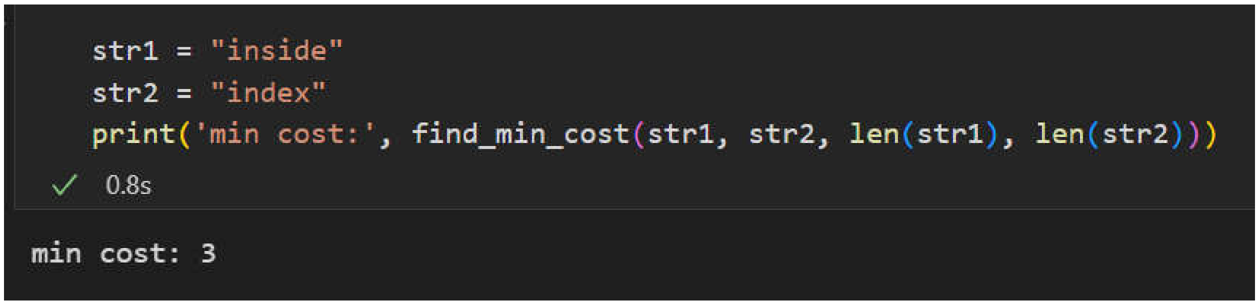
	Nothing	Str1[0]	Str1[1]	Str1[2]	Str1[3]	Str1[4]	Str1[5]
Nothing	0	1	2	3	4	5	6
Str2[0]	1	2	…	…	…	…	…
Str2[1]	2	3	…	…	…	…	…
Str2[2]	3	4	…	…	…	…	…
Str2[3]	4	5	…	…	…	…	…

The idea is to process all characters one by one starting from either from left or right sides of both strings. traverse from right corner, there are two possibilities for every pair of character being traversed. With using dynamic programming and recursive function we can implement our solution for this problem as below:



```python
def find_min_cost(str1, str2, l1, l2):
    map = []
    for i in range(l1+1):
        row = []
        for j in range(l2+1):
            row.append(0)
        map.append(row)

    for i in range(l1 + 1):
        for j in range(l2 + 1):
            if i == 0:
                map[i][j] = j

            elif j == 0:
                map[i][j] = i

            elif str1[i-1] == str2[j-1]:
                map[i][j] = map[i-1][j-1]

            else:
                map[i][j] = 1 + min(map[i][j-1],map[i-1][j],2*map[i-1][j-1])
    return map[l1][l2]
```

As we can see cost of remove and insert is 1 and cost of modifying is 2. We can see results in below example:



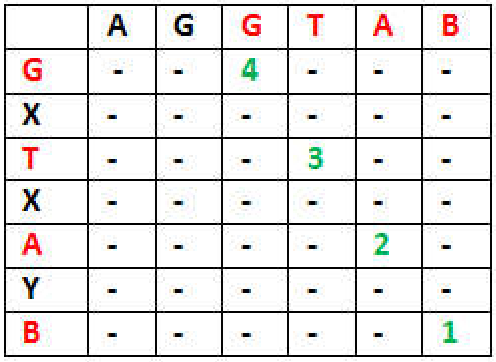
```
str1 = "inside"
str2 = "index"
print('min cost:', find_min_cost(str1, str2, len(str1), len(str2)))
```

```
min cost: 3
```

3.b.

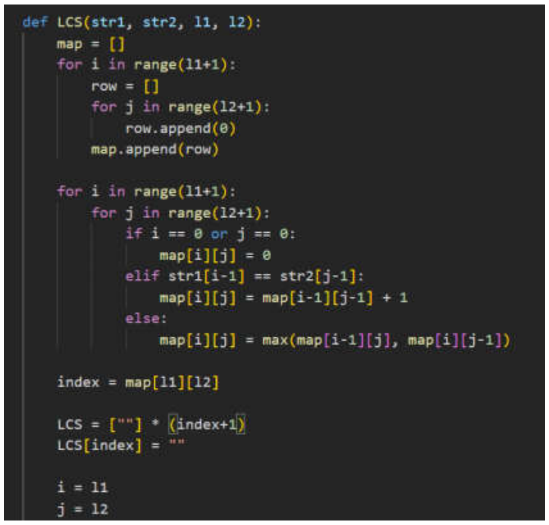
We should create a table which has alphabet of words, like the last part we just need to find subsequences of words.

Consider the input strings “AGGTAB” and “GXTXAYB”. Last characters match for the strings. So, length of LCS can be written as:

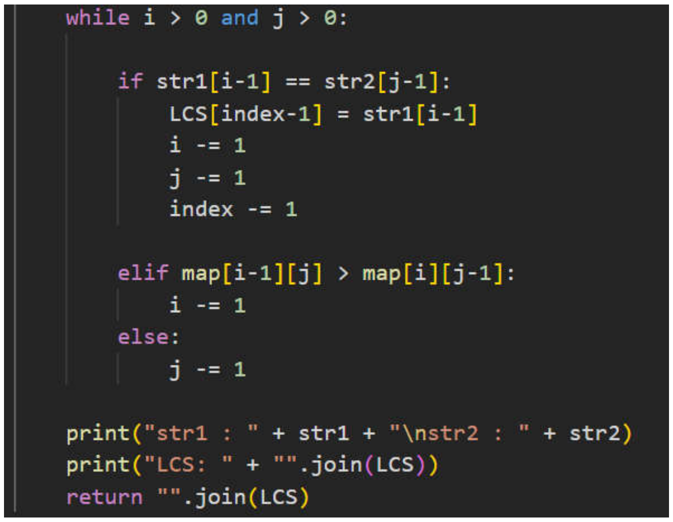


	A	G	G	T	A	B
G	-	-	4	-	-	-
X	-	-	-	-	-	-
T	-	-	-	3	-	-
X	-	-	-	-	-	-
A	-	-	-	-	2	-
Y	-	-	-	-	-	-
B	-	-	-	-	-	1

With using dynamic programming and recursive function we can implement our solution for this problem as below:

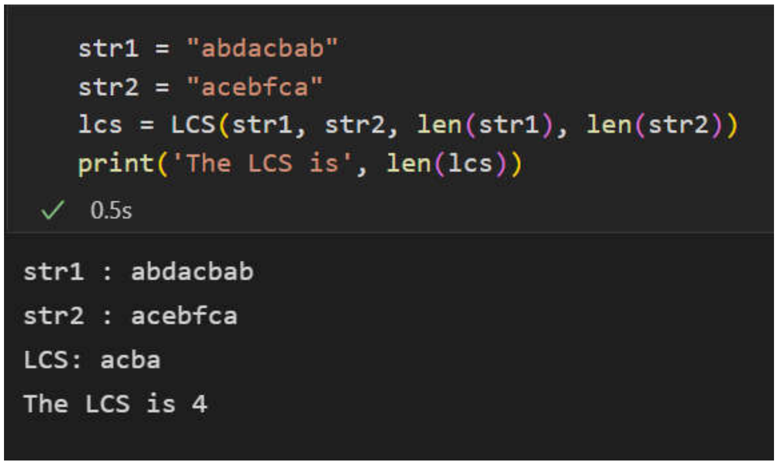


```
def LCS(str1, str2, l1, l2):
    map = []
    for i in range(l1+1):
        row = []
        for j in range(l2+1):
            row.append(0)
        map.append(row)

    for i in range(l1+1):
        for j in range(l2+1):
            if i == 0 or j == 0:
                map[i][j] = 0
            elif str1[i-1] == str2[j-1]:
                map[i][j] = map[i-1][j-1] + 1
            else:
                map[i][j] = max(map[i-1][j], map[i][j-1])

    index = map[l1][l2]

    LCS = [""] * (index+1)
    LCS[index] = ""

    i = l1
    j = l2
```



```
    while i > 0 and j > 0:

        if str1[i-1] == str2[j-1]:
            LCS[index-1] = str1[i-1]
            i -= 1
            j -= 1
            index -= 1

        elif map[i-1][j] > map[i][j-1]:
            i -= 1
        else:
            j -= 1

    print("str1 : " + str1 + "\nstr2 : " + str2)
    print("LCS: " + "".join(LCS))
    return "".join(LCS)
```

We can see results in below example:



```
str1 = "abdacbab"
str2 = "acebfca"
lcs = LCS(str1, str2, len(str1), len(str2))
print('The LCS is', len(lcs))
```

✓ 0.5s

```
str1 : abdacbab
str2 : acebfca
LCS: acba
The LCS is 4
```