Name: Gaurang Sharma

Matriculation number: 11012446

GitHub link: https://github.com/GaurangSharma44/Big-Data-Programming

----

Exercise 1:

Thought Process:

Started with working on a logic for the problem on paper. First thought was looping and traversing both strings. Keep incrementing the second string until a match is found (push it in a different list (S3)) and continue if not. Then S3 is your longest common subsequence.

Algorithm \ Pseudocode for Exercise 1:

```
1. Input strings S1 and S2
        - No special characters or numbers are allowed
        - Strings and result can be case insensitive
  Convert strings to list
2. If S1[i] matches with S2[j],
        add S2[j] to S3,
        increment S2 by 1
   else,
        continue
3. Convert S3 to string and print the Longest Common Subsequence
```

References:

1. https://www.geeksforgeeks.org/python-convert-a-list-of-multiple-integers-into-a-single-integer/
2. https://www.w3schools.com/python/ref_list_append.asp
3. https://www.geeksforgeeks.org/python-string-isalpha-application/

Code:

```
def LongestCommonSubsequece():

    take_input = True
    while(take_input):
        S1 = input("Enter your first string: ")
        S2 = input("Enter your second string: ")
        if not S1.isalpha() or not S2.isalpha():
            print("Please use only alpha characters")
            take_input = True
        else:
            take_input = False

    s1_array = list(S1)
    s2_array = list(S2)

    temp = []
    index = 0
    for i in range(len(s1_array)):
        for j in range(len(s2_array))[index:]:
            if s1_array[i].lower() == s2_array[j].lower():
                temp.append(s2_array[j])
                index = j+1
            else:
                continue
            break
    S3 = temp
    print ("The Longest Common Subsequence is")
    for i in S3:
        print(i, end="")

if __name__ == "__main__":
    LongestCommonSubsequece()
```

Exercise 2:

Thought Process:

For this problem I started with the code first, maybe not a good practice but couldn’t find any solution earlier.

I started with restricting the user to only input alpha characters and/or ‘.’,’*’ for the second string.

Then I wrote a program for simple substring matching.

I then thought of replacing ‘.’ with the desired character and ignoring the rest (since substring is to be found and not exact match).

It was really tricky with ‘*’,

> first, I replaced * with previous character,
>
> then for 0 occurrences, I thought to remove the star itself by checking if the previous character in S2 matches with the desired character of S1,
>
> for n number of ‘*s’ I thought to check if the next character in S1 matches with the previous in S2, if yes keep looping.
>
> for the special case of ‘. *’, which meant n number of ‘*s’. I checked if the previous char of a ‘*’ is a ‘.’ Then replace it with the desired character.
>
> This is where I figured to push the pattern into a new list so to check for the above condition and not interfere with the pattern.

Algorithm / Pseudocode for Exercise 2:

*My assumption and what my code is based upon:

- dot and star in succession (.*) means either n number of dots or at least 1 dot
- an alphabet followed by * (c*) means at least 1 occurrence of the alphabet but the star can be ignored (c)
- we have to find substring and not exact match

1. Input strings S1 and S2

    - No special characters or numbers are allowed in S1

```
        - No numbers or special characters except '.' and '*' are allowed in S2
  Convert strings to lists
2. If S1[0] and S2[0] are equal,
        add S1[0] to S3,
        and increment both lists by 1
        else,
                else if S2 has '.',
                        add S1[i] to S3,
                        and increment both lists by 1
                else if S2 has '*',
                        if the previous character in S2 is equal with S1[i],
                                add S1[i] to S3,
                                if S1[i+1] also equals to the previous character of S2,
                                        add S1[i] to S3
                                else,
                                        increment only S1 by 1
                        if the previous character in S2 is a '.',
                                add S1[i] to S3,
                                increment only S1 by 1
                        else,
                                increment only S2 by 1
                else,
                        add S2[j] to S3,
                        and increment S2 by 1
                        increment only S2 by 1
3. Convert S1 and S3 lists to string, search for S1 in S3
        if yes, return True
        else, return False
```

References:

1. https://docs.python.org/3.3/howto/regex.html
2. https://stackoverflow.com/questions/252703/what-is-the-difference-between-pythons-list-methods-append-and-extend
3. https://stackoverflow.com/questions/6576962/python-regular-expressions-return-true-false

Code:

```
import re

def PatternMatching():

    take_inputS1 = True
    while(take_inputS1):

        S1 = input ("Enter your first string: ")
        m = re.match('^[a-z]+$',S1)
        if m:
            take_inputS1 = False
            #print("String accpeted")
            #print (S1)
        else:
            print("Please enter only alphabets")
            take_inputS1 = True

    take_inputS2 = True
    while(take_inputS2):

        S2 = input ("Enter your second string: ")
        o = re.findall("[a-z.*]", S2)
        if o:
            take_inputS2 = False
            #print("String accpeted")
            #print (S2)
        else:
            print("Second string can only contain alphabets, . and *")
            take_inputS2 = True

    S2_array = list(S2)
    S1_array = list(S1)
    temp = []

    print ("S1 array: ",S1_array)
    print ("S2 array: ",S2_array)

    index1 = 0
    index2 = 0
    for j in range(len(S2_array))[index2:]:
        for i in range(len(S1_array))[index1:]:
            if S1_array[i] == S2_array[j]:
                temp.extend(S1_array[i])
                index1 = i+1
                index2 = j+1
                break
            else:
                #'*' can only be the previous character, either 0 or more
occurrences
                if S2_array[j] == '*':
                    if S2_array[j-1] == S1_array[i]:
                        temp.extend(S1_array[i])
                        index2 = j+1
                        #n number  of occurrences
                        k = i+1
                        if len(S1_array) > k:
                            if S2_array[j-1] == S1_array[k]:
                                temp.extend(S1_array[k])
                                #index1 = i+1
                                break
                            else:
                                index1 = i+1
                                break
                        else:
                            break
                        break
                    #if '.' is the previous character of '*', it can be either
1 '.' or n number of '.'
                    elif S2_array[j-1] == '.':
                        temp.extend(S1_array[i])
                        index1 = i+1
                        break
                    #0 occurrences of *
                    else:
                        index2 = j+1
                        break
                #'.' can be any character but only 1 occurrence
                if S2_array[j] == '.':
                    temp.extend(S1_array[i])
                    index1 = i+1
                    index2 = j+1
                    break
                else:
                    temp.extend(S2_array[j])
                    index2 = j+1
                    break
    S3_array = temp
    str1 = ''.join(S1_array)
    str3 = ''.join(S3_array)
    print ("S1: ",str1)
    print ("S3: ",str3)

    if bool(re.search(str1,str3))==True: #Searching for a substring
        print ('True')
    else:
        print ('False')

if __name__ == "__main__":
    PatternMatching()
```
