Leetcode questions for Strings (python):

REMOVE OUTMOST BRACKET:

Welcome | removeParenthesis.py | valid_parenthesis

```python

def remove_outermost_bracket (s):
    result=[]
    open_count=0

    for char in s:
        if char == '(' and open_count>0:
            result.append(char)
        elif char == ')' and open_count >1:
            result.append(char)

        if char == '(':
            open_count+=1
        elif char == ')':
            open_count-=1

    return ''.join(result)

print (remove_outermost_bracket("(()())"))
```

To reverse the words in a string:

```python
def reverse_words(s):
    words = [word for word in s.split()]
    return ' '.join(words[::-1])
print(reverse_words("the sky is black"))
```

To return longest common prefix:

```python
def longest_common_prefix(s):
    if not s:
        return ""

    for i in range(len(s[0])):
        char = s[0][i]
        for j in range(1, len(s)):
            if i == len(s[j]) or s[j][i] != char:
                return s[0][:i]
    return s[0]

s= ["flower","glower","glightwer"]
print(longest_common_prefix(s))
```

To print the largest odd:

```python
def largest_odd(s):
    for i in range (len(s)-1,-1,-1):
        if int(s[i])%2==1:
            return s[:i+1]
    return ""
#it would print the largest odd in between the whole number
print(largest_odd("51"))
print(largest_odd("51246"))
print(largest_odd("51789"))
```

Longest common substring:

```
def longest_common_subsequence(str1, str2):
    m = len(str1)
    n = len(str2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if str1[i - 1] == str2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
    lcs = ""
    i, j = m, n
    while i > 0 and j > 0:
        if str1[i - 1] == str2[j - 1]:
            lcs = str1[i - 1] + lcs
            i -= 1
            j -= 1
        elif dp[i - 1][j] > dp[i][j - 1]:
            i -= 1
        else:
            j -= 1
    return lcs

# First time calling the lcs function
lcs = longest_common_subsequence(strs[0], strs[1])
```

```
    # Iterate through the remaining strings and update the LCS
    for i in range(2, len(strs)):
        lcs = longest_common_subsequence(lcs, strs[i])
        if not lcs: #if at any point the lcs is empty, then return ""
            return ""

    return lcs

# Example Usage
s = ["abcdefg", "acecfgh", "abcdegh"]
res = longest_common_subsequence_multiple_strings(s)
print(res)  # Output: aceg
```

Longest common substring:

```
def longest_common_substring(words):
    if not words:
        return ""
    shortest = min(words, key=len)  #-------Pick the shortest word to optimize search
    longest_substr = ""
    for i in range(len(shortest)):  #-------starting of substring
        for j in range(i + 1, len(shortest) + 1): #--------ending of substring
            substr = shortest[i:j]  #----------- Extract substring from shortest word
            if all(substr in word for word in words):  #--------- Check all words
                if len(substr) > len(longest_substr):
                    longest_substr = substr
    return longest_substr
```

3461. Check If Digits Are Equal in String After Operations I

Easy | Topics | Companies | Hint

You are given a string `s` consisting of digits. Perform the following operation repeatedly until the string has **exactly** two digits:

- For each pair of consecutive digits in `s`, starting from the first digit, calculate a new digit as the sum of the two digits **modulo** 10.
- Replace `s` with the sequence of newly calculated digits, *maintaining the order* in which they are computed.

Return `true` if the final two digits in `s` are the **same**; otherwise, return `false`.

Example 1:

Input: `s = "3902"`

Output: `true`

Explanation:

- Initially, `s = "3902"`
- First operation:
  - `(s[0] + s[1]) % 10 = (3 + 9) % 10 = 2`
  - `(s[1] + s[2]) % 10 = (9 + 0) % 10 = 9`
  - `(s[2] + s[3]) % 10 = (0 + 2) % 10 = 2`
  - `s` becomes `"292"`
- Second operation:
  - `(s[0] + s[1]) % 10 = (2 + 9) % 10 = 1`
  - `(s[1] + s[2]) % 10 = (9 + 2) % 10 = 1`
  - `s` becomes `"11"`
- Since the digits in `"11"` are the same, the output is `true`.

Code

Python3 Auto

```python
class Solution:
    def hasSameDigits(self, s: str) -> bool:
        temp=s
        while len(temp)>2:
            k=""
            for i in range(len(temp)-1):
                k+=str((int(temp[i])+int(temp[i+1]))%10)
            temp=k
        return temp[0]==temp[1]

```