Question 4:
Given a string, find the length of the longest repeating subsequence, such that the two subsequences don't have the same string character at the same position, i.e. any ith character in the two subsequences shouldn't have the same index in the original string.

Examples:

```
Input: str = "abc"
Output: 0
There is no repeating subsequence

Input: str = "aab"
Output: 1
The two subsequence are 'a'(first) and 'a'(second).
Note that 'b' cannot be considered as part of subsequence
as it would be at same index in both.

Input: str = "aabb"
Output: 2

Input: str = "axxxy"
Output: 2
```

To solve this problem, I have used dynamic programming.
Why dynamic programming?
It is an efficient technique to solve complex problems
While solving a problem it breaks down the problem into subproblems and solves each subproblem only once.
Generally problems solves by dynamic programming is used in optimization problems where we will have to find the best solution / efficient solution (time and space efficiency)
It is used to reduce redundancy, to solve graph problems, string problems etc.

Coming to the description of the program:

```
str = input('enter the string:')
```
his line of code is used to take a string input. 'str' is the variable name in which the string is stored

```
n = len(str)
```
'len' is a python inbuilt function which is used to calculate the length of the given string
Here n = len(str), the length of the inputted string is stored in a variable names 'n'

```
def repeating_string(str):
```
Here I have defined an user defined function named "repeating_string" which takes an argument named "str"

```
a = [[0 for j in range(n+1)] for i in range(n+1)]
```
The above statement is used to initialize a 2D list named 'a' with dimension (n+1)*(n+1) and initializes all of its elements to zero
[0 for j in range(n+1)] ---- creates a list of 'n+1' zeroes
The outer list creates a list of 'n+1' lists

```
    for i in range(1, n+1):
        for j in range(1, n+1):
            if i != j and str[i-1] == str[j-1]:
                a[i][j] = a[i-1][j-1] + 1
            else:
                a[i][j] = max(a[i-1][j], a[i][j-1])
    return a[n][n]
```

The main logic lies here:
Initialise integer ‘i’ from 1 to n+1
Initialize integer ‘j’ from 1 to n+1
Check condition: i != j AND str[i-1] == str[j-1] , if TRUE: a[i][j]=a[i-1][j-1] + 1
Else if FALSE: a[i][j] = **max(**a[i-1][j],a[i][j-1]
Finally return value a[n][n]

```
print(repeating_string(str))
```

This is the print statement within which the ‘repeating_string’ is called, finally output is printed in the console with the help of this statement