Count Consonants in String

In this lesson, you will learn how to count consonants of a string using both an iterative and recursive approach in Python.

In this lesson, we focus on the following problem:

Given a string, calculate the number of consonants present.

The vowels are the following letters:

```
a e i o u
```

Any other letter that is not a vowel is a consonant.

Let's have a look at an example:

```
Welcome to Educative!
```

has `9` consonants.

Before you dive into the implementation, consider the edge cases such as spaces and exclamation marks. This implies that if a character is not a vowel, it has to be a letter to be considered a consonant.

Iterative Approach

Check out the iterative approach in the snippet below:

```
vowels = "aeiou"

```

```
3  def iterative_count_consonants(input_str):
4      consonant_count = 0
5      for i in range(len(input_str)):
6          if input_str[i].lower() not in vowels and input_str[i].isalpha():
7              consonant_count += 1
8      return consonant_count
```

iterative_count_consonants(input_str)

On **line 1**, we define `vowels` globally to be `"aeiou"`. `input_str` is the given string passed to `iterative_count_consonants` function which we have to process to calculate the number of consonants. `consonant_count` is initialized to `0` on **line 4** and then the `input_str` is traversed character by character using a `for` loop on **line 5**. Note that `input_str[i]` is changed to lowercase using `lower()` because `vowels` contains all the vowels in lowercase so that we can compare them correctly on **line 6**. As discussed before, we check if `input_str[i].lower()` is present in `vowels` or not. Additionally, `input_str[i].isalpha()` should also be `True` for the condition on **line 6** to be `True`. Thus, if `input_str[i]` is not a vowel and an alphabet, then `consonant_count` is incremented by `1` on **line 7**. After the condition on **line 6** is evaluated for every character in `input_str`, the final count of `consonant_count` is returned from the function on **line 8**.

The running time complexity for the iterative approach is $O(n)$ because, for each character in the input string of length n, we spend a constant amount of effort.

Recursive Approach

The implementation of the recursive approach will be very similar to the recursive implementations we have covered in the previous lessons. Check it out below:

```
vowels = "aeiou"

def recursive_count_consonants(input_str):
    if input_str == '':
        return 0

    if input_str[0].lower() not in vowels and input_str[0].isalpha():
        return 1 + recursive_count_consonants(input_str[1:])
    else:
        return recursive_count_consonants(input_str[1:])
```

recursive_count_consonants(input_str)

Here, `vowels` is again defined globally on **line 1**. Now let's come to `recursive_count_consonants` function on **line 3**. If `input_str` is empty, `0` is returned from the function. This is the base case for `recursive_count_consonants`.

On **line 7**, we have the same condition as in the iterative implementation. If the condition on **line 7** evaluates to `True`, we add `1` in addition to the count returned from the recursive call where a truncated string is passed using the slice notation (**line 8**). Otherwise, if the condition on **line 7** is not `True` and `input_str[0]` is not a consonant, `1` is not added to the count, but a recursive call is made by passing `input_str[1:]` on **line 10**. At every recursive call, the first character is evaluated while the rest of the characters are passed into the next recursive calls.

The running time complexity for the recursive approach is $O(n)$ because, for each character in the input string of length n, we make a recursive call and each recursive call has constant time complexity.

In the code widget below, you can find both the implementations. Go ahead and play around with them using your test cases.

```
vowels = "aeiou"

def iterative_count_consonants(input_str):
    consonant_count = 0
    for i in range(len(input_str)):
        if input_str[i].lower() not in vowels and input_str[i].isalpha():
            consonant_count += 1
    return consonant_count


def recursive_count_consonants(input_str):
    if input_str == '':
        return 0

    if input_str[0].lower() not in vowels and input_str[0].isalpha():
        return 1 + recursive_count_consonants(input_str[1:])
    else:
        return recursive_count_consonants(input_str[1:])

input_str = "abc de"
print(input_str)
print(iterative_count_consonants(input_str))
input_str = "LuCiDPrograMMiNG"
print(input_str)
print(recursive_count_consonants(input_str))
```

I hope you have by now grasped the logic of how we have solved problems in this chapter using recursion! Let’s check that in the coding challenge waiting for you in the next lesson.