Calculate String Length

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

We'll cover the following

In this lesson, we focus on the following problem:

Given a string, calculate its length.

If you are preparing for an interview or trying to understand the notion of recursion to solve a problem, I hope this lesson is helpful to you.

Python has a built-in `len()` function which returns the length of a string. This is the standard way to obtain the length of a string in Python.

```
input_str = "LucidProgramming"
print(len(input_str))
```

Iterative Approach

Now we are going to code the same functionality ourselves in Python. Let's begin with the iterative approach.

```
# Iterative length calculation: O(n)
def iterative_str_len(input_str):
    input_str_len = 0
    for i in range(len(input_str)):
        input_str_len += 1
    return input_str_len
```

On **line 3**, `input_str_len` is initialized to `0`. Then using a `for` loop on **line 4**, `input_str` is iterated character by character and `input_str_len` is incremented by `1` in each iteration on **line 5**. Finally, the final value of `input_str_len` is returned from the function on **line 6**. As the entire length of the string is traversed once, the time complexity for this solution is thus $O(n)$ where n is the length of the string.

Recursive Approach

Let's go ahead and have a look at the recursive approach:

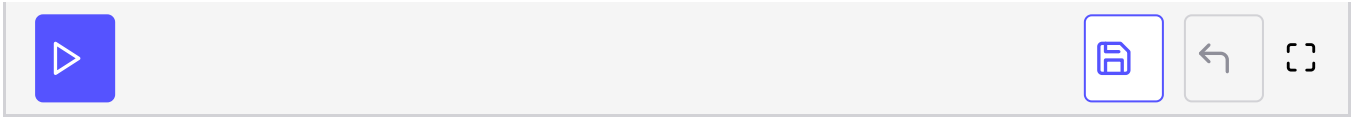
```
1  # Recursive length calculation: O(n)
2  def recursive_str_len(input_str):
3      if input_str == '':
4          return 0
5      return 1 + recursive_str_len(input_str[1:])
```

The base case for this function is when an empty string is encountered. If `input_str` is empty, `0` is returned to make a count of `0` for the empty string. Otherwise, `1` is added to whatever is returned from the recursive call on **line 5** which takes in `input_str[1:]`. The slicing notation in `input_str[1:]` indicates that all the characters except at the `0th` index are passed into the recursive call. Therefore, every recursive call keeps shortening `input_str` by one character, which is being counted in that recursive call. As there will be n recursive calls, each expending a constant amount of computational effort, the time complexity for this solution is $O(n)$ where n is the length of the string. Wasn't this pretty simple? I hope you are clear about everything that we have studied so far.

In the code widget below, you can run and play with both the iterative and recursive implementations.

```
# Iterative length calculation: O(n)
def iterative_str_len(input_str):
    input_str_len = 0
    for i in range(len(input_str)):
        input_str_len += 1
    return input_str_len

# Recursive length calculation: O(n)
```

```
def recursive_str_len(input_str):
    if input_str == '':
        return 0

    return 1 + recursive_str_len(input_str[1:])

input_str = "LucidProgramming"

print(iterative_str_len(input_str))
print(recursive_str_len(input_str))
```



Let’s have a look at another problem which we can solve using recursion in the next lesson!