

Thinking Recursively

In this lesson, we will go over some coding exercises with step by step explanations of recursion for you.

We'll cover the following ^

- Coding exercise:
 - Input
 - Output
- Explanation
- Coding exercise:
 - Input
 - Output

If recursion intimidates you, this lesson will break it down for you so that you never feel scared of recursion again. The aim of this lesson is to teach you how to come up with recursive solutions to problems. Let's do a small coding exercise first.

Coding exercise:

Given a string, `str`, and a character, `char`, your function `countChar` should count the number of `char` characters in the string `str`. We have written a helper function `countChar_` for you, but it only works for substring `str[1:]`. This means that you have to do processing for `str[0]` character while the rest of the answer can be found in the `countChar_` function.

Input

A string and a character:

```
str = "abacada"  
char = 'a'
```







Output


The count of `char` in `str`

```
countChar("abacada", 'a') = 4
```

You must use the helper function `countChar_`

```
def countChar(str, char):  
    '''  
    you can call helper function as countChar_(str[1:], char)  
    '''  
    return -1
```





Hint 1 of 2

< Handle the case for str being an empty string >

If you were able to complete this challenge, congratulations! You have just written a recursive algorithm. If you delete the “_” from the end of `countChar_`, it will become a recursive algorithm.

Explanation

Now, let's step back and see what the point of this exercise was.

There are two things you did in this algorithm:

- 1 - You catered to the **base case** when the string is empty
- 2 - You did computation for one step and passed the rest of the work to the **recursive call**. Here computation is simply checking if the first character of the string `str` is character `char` or not.

This is exactly how you go about writing a recursive algorithm. First, you try to find a pattern that is repeating and could be a recursive step. Then you look for the base cases, which specify some termination condition for your algorithm. Lastly,

base cases, which specify some termination condition for your algorithm. Lastly, you write code for the first step and pass the rest of the work to recursive call(s).

Coding exercise:

We are going to discuss Fibonacci numbers in the next lesson, but why don't you take a look at the problem now that you know how simple recursion actually is!

Fibonacci numbers are given by the following formulas:

$$Fib(0) = 0$$

$$Fib(1) = 1$$

$$Fib(n) = Fib(n - 1) + Fib(n - 2)$$

This results in the following sequence of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

You can already see what the base cases are and what the recursive step would be. Again, we have provided you with a function `fib_` that only works for `n-1` and `n-2`. This means you need to handle the first step of the solution again.

Input

A number `n`

```
n = 6
```

Output

Return $Fib(n)$

```
fib(6) = 8
```

Again, you must use the `fib_` function in your solution.

```
def fib(n):  
    ...  
    You can call helper function as fib_(n-1) and fib_(n-2)  
    ...  
    return -1
```





Hint 1 of 2

< Handle case for `n` being `1` or `0`



You may remove “_” in `fib_` from your solution again to see the magic.

In the next lesson, we will go over the Fibonacci numbers in depth.