Recursion

This lesson will explain the concept of recursion in Python.

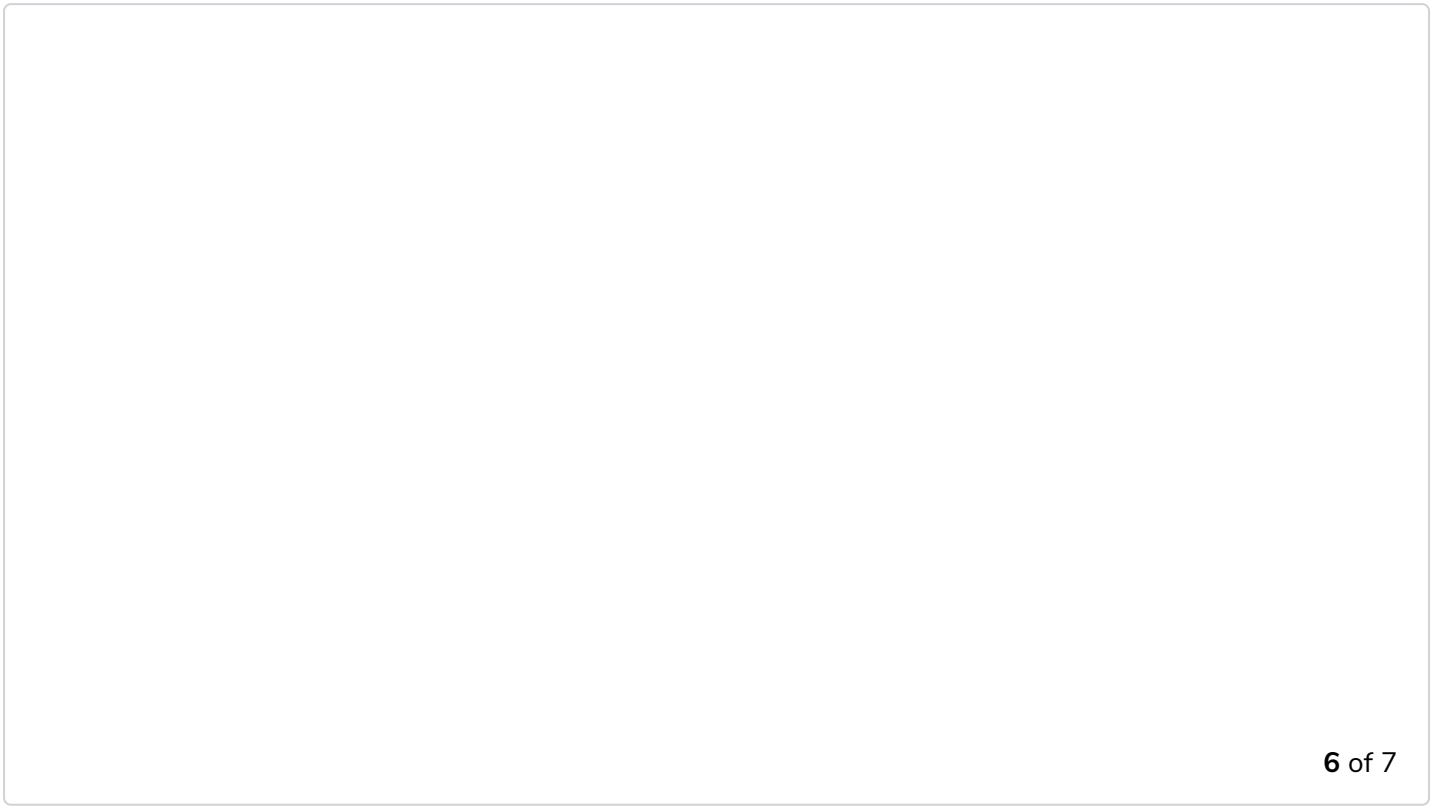
We'll cover the following

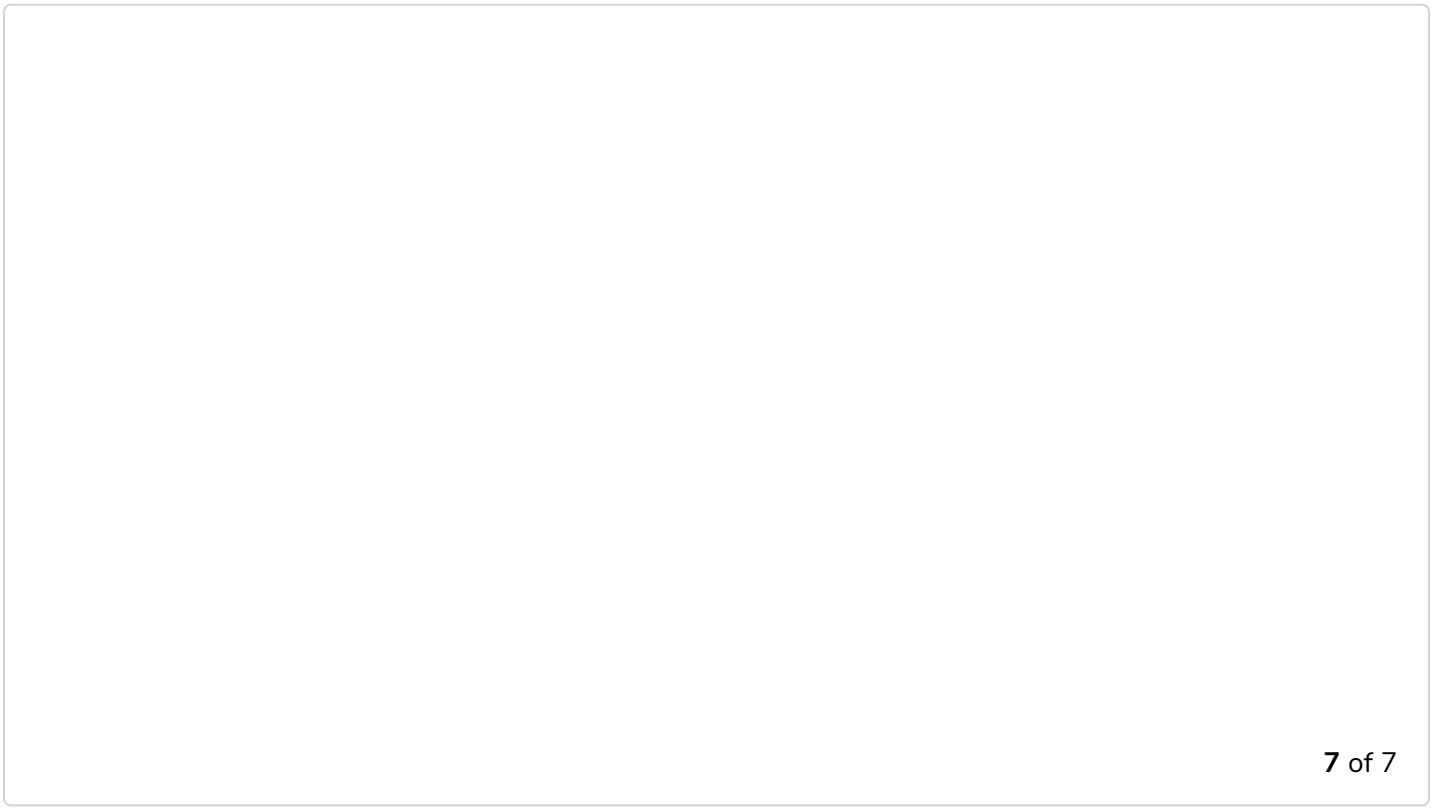
Definition

Recursion is the process in which a function calls itself during its execution. Each recursive call takes the program one scope deeper into the function.

The recursive calls stop at the **base case**. The base case is a check used to indicate that there should be no further recursion.

Imagine recursive calls as nested boxes where each box represents a function call. Each call makes a new box. When the base case is reached, we start moving out of the boxes one by one:





As we can see in the illustration above, the function is called repeatedly for a specified number of times.

A Simple Example

Let’s write a function which decrements a number recursively until the number

becomes `0`:

```
def rec_count(number):
    print(number)
    # Base case
    if number == 0:
        return 0
    rec_count(number - 1)  # A recursive call with a different argument
    print(number)


rec_count(5)
```

This is fairly easy to understand. In each call, the value of the `number` variable is printed. We then check whether the base case has been fulfilled. If not, we make a recursive call to the function with the current value decremented.

One thing to notice is that an outer call cannot move forward until all the inner recursive calls have finished. This is why we get a sequence of `5` to `0` to `5`.

Why Use Recursion?

Recursion is a concept which many find difficult to grasp at first, but it has its advantages. For starters, it can significantly reduce the runtime of certain algorithms, which makes the code more efficient.

Recursion also allows us to easily solve many problems related to **graphs** and **trees**, things you may study in the future. It is also important in search algorithms.

However, we need to be careful when using recursion. If we don't specify an appropriate base case or update our arguments as we recurse, the program will reach **infinite recursion** and crash. The arguments passed to our recursive function are updated in each recursive call so that the base case can eventually be reached.

A Complex Example

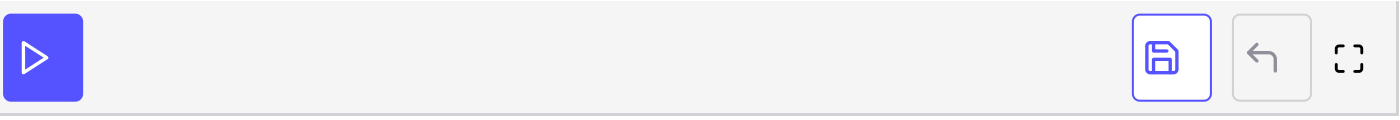
The Fibonacci sequence is a popular series of numbers in mathematics, where every number is the sum of the two numbers before it. The first two terms in the series are `0` and `1`:

```
0 1 1 2 3 5 8 13
```

Let's write a function which takes in a number, `n`, and returns the **nth** number in the Fibonacci sequence. So, if `n == 6`, the function will return `5`:

```
def fib(n):
    # The base cases
    if n <= 1:  # First number in the sequence
        return 0
    elif n == 2:  # Second number in the sequence
        return 1
    else:
        # Recursive call
        return fib(n - 1) + fib(n - 2)


print(fib(6))
```



First, we handle our base cases. We know that the first two values are always `0` and `1`, so that is where we can stop our recursive calls.

If `n` is larger than `2`, then it will be the sum of the two values before it.

That brings us to the end of our discussion on functions. We are now equipped with all the tools needed to write meaningful programs in Python.

In the next section, we'll take a look at Python **loops**. Be sure to check out the quiz and exercises on function before you move on.