Recursion

This lesson explains the concept of recursion using the factorial example

We'll cover the following

What is Recursion?

Recursion is a method of function *calling* in which a function calls *itself* during execution.

A recursive function must comprise of at least one base case i.e. a condition for termination of execution.

The function keeps on converging to the defined base case as it continuously calls itself.

Example

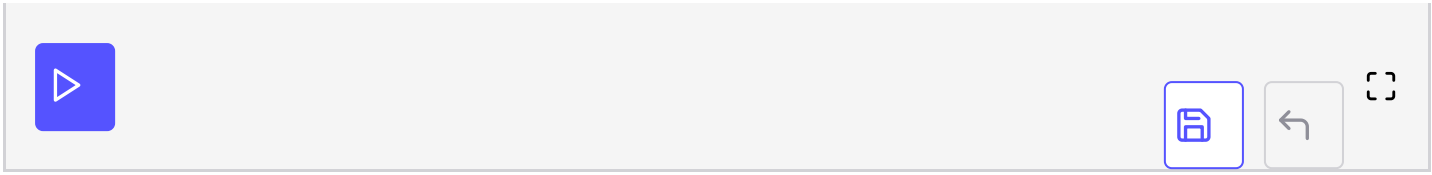
Let's start by showing an example and then discussing it.

```
<?php
function factorial($n)
{
    if ($n == 1 || $n == 0)
    { //base case
        return 1;
    }
    else
    {
        return $n * factorial($n - 1); //function calls itself

    }
}

echo factorial(4); //calling the function with 4 as the argument

?>
```



This is a classic application of *recursion*. This function calculates the **factorial** of a number.

> **Note:** The *factorial* of **n**, written **n!**, is the product of every number from **1 to n**. So we can say that **4!** = **4×3×2×1**.

Solution Explained

Let's step through what happens in our *function* when we call `factorial(4)`.

- In **line 11**, `factorial(4)` is called
  - Inside the `factorial` function, since, **n=4** we take the `else` path. We `return` `4×factorial(n-1)`, **line** 7.
  - `factorial(3)` is called
    - Since **n=3**, we take the `else` path. We return `3×factorial(n-1)`, **line** 7.
    - `factorial(2)` is called
      - Since **n=2**, we take the `else` path. We return `2×factorial(n-1)`, **line** 7.
      - `factorial(1)` is called
        - Since **n=1**, we take the *first* path, **line 3**, and finally return **1** to the *previous* function.
      - `factorial(1)` returns **1** so `factorial(2)` can return **2×1...2**.
    - `factorial(2)` returns **2** so `factorial(3)` can return **3×2...6**.
  - `factorial(3)` returns **6** so `factorial(4)` can return **4×6...24**.

Many times, a *recursive* solution to a problem is very easy to program.

- The drawback of using *recursion* is that there is a lot of *overhead*.

Every time a function is called, it is placed in *memory*. Since you don't **exit** the

`factorial` function until **n** reaches **1**, **n** functions will reside in *memory*. This isn't a

problem with the simple `factorial(4)`, but other functions can lead to serious memory requirements.

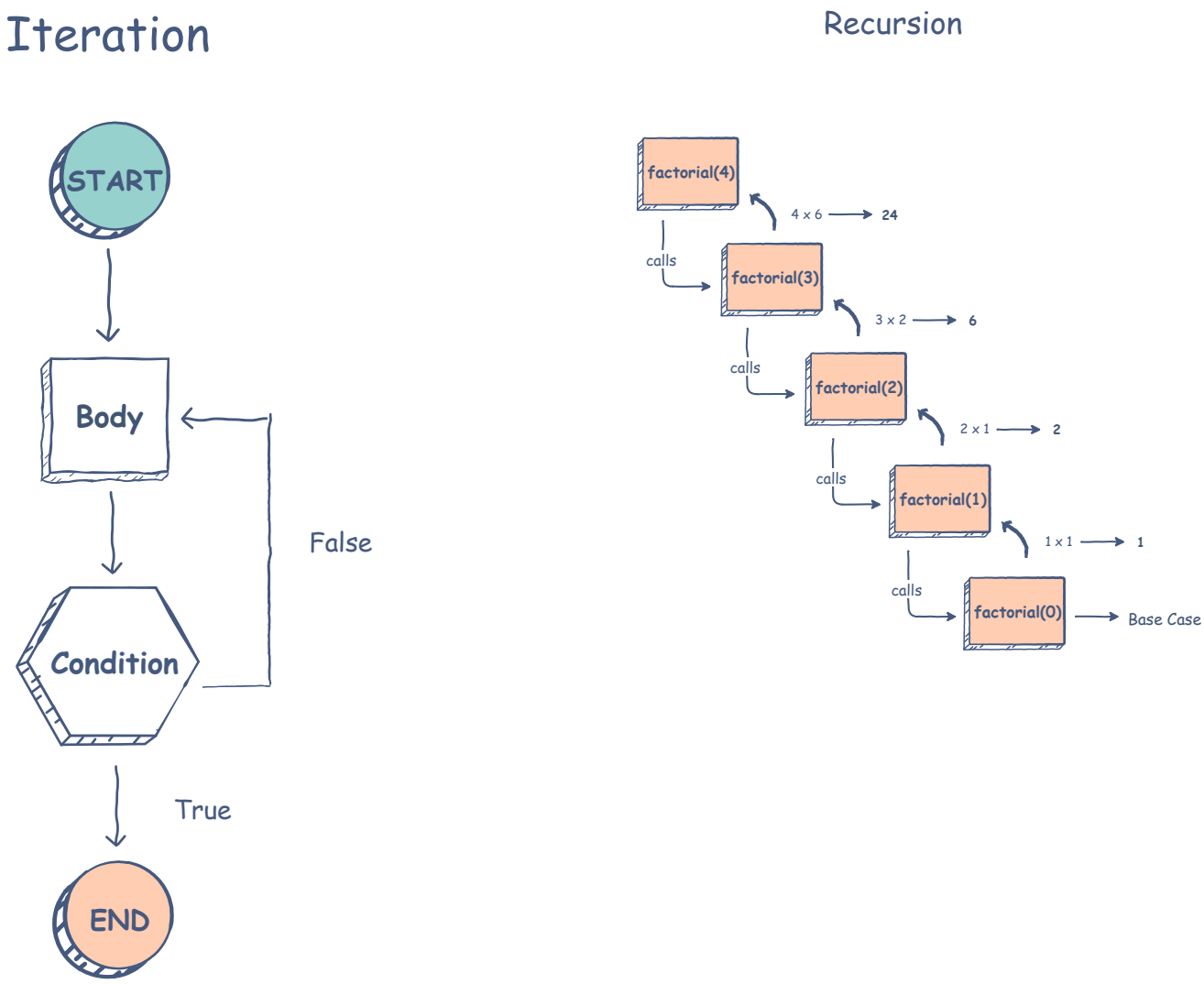
Recursion versus Iteration

> **Iteration** is a sequence of steps that are executed *repeatedly* until a certain condition is met.

The steps may or may not be enclosed in a function.

However, in recursion, the function itself is called repeatedly as done in the `factorial` example above.

The following figure illustrates the difference between *recursion* and *iteration*:



Well, that was all on *recursion*. Practice your concepts by solving a challenge in the next lesson.