Ever wonder why programming languages such as *Haskell*, *Scala*, *JavaScript*, etc are so in demand nowadays? This is because these languages are based on **Functional Programming**, meaning the system is built around the concept of recursion. Entire languages are now being based on recursion. Interesting, right?

In general, almost any problem that can be solved with a non-recursive code can also be solved by recursion. In fact, most problems that can't be solved by a non-recursive code can even be solved by recursion. This doesn't mean, however, that you should always use recursion to solve the problem. This is exactly what we will be discussing in this lesson.

Using recursion to solve problems may generate a lot of extra effort and time. The recursive solution might actually be an inefficient way of solving the problem. In those cases, it is better to stick to the conventional iterative way of writing your code. But, how you do you know when to use recursion or not? Sadly, there is no definitive answer to this question, but we do have a few signs that indicate when to use recursion to solve them.

These are some of the noticeable signs in the problem that can help you determine whether or not you should solve the problem recursively.

Use where appropriate#

Recursion should be used where you feel like it is appropriate or where it just feels natural. You will most likely encounter problems where you cannot come up with an iterative solution, and that is where recursion fits in nicely.

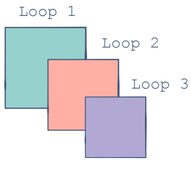
Problem breaks down into smaller similar subproblems

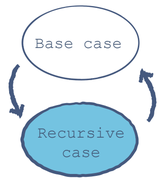
The most obvious sign to use recursion is when a problem can be broken down into smaller subproblems. When you encounter a problem and observe a pattern of it breaking down into similar subproblems, then it can likely be solved using recursion.

Problem requires an arbitrary number of nested loops#

There are problems that require you to use an arbitrary number of nested loops, that is loops within loops, but when you start using them - the problem remains unsolved or becomes more complicated. As long as you know the number of loops that have to be nested, there is no problem, and you can use the iterative approach. But some problems require an arbitrary number of nested loops, and these are more easily solved using recursion.

For example, iterating through a graph or a tree, finding all permutations of a string, etc.





Easier to solve it Recursively rather than Iteratively

This is what it boils down to: use the method that makes it is easier to solve the problem. We know that most of the problems can be solved using both these approaches, but it depends on what is more comfortable for you when solving the problem.

Now that we have a foundational idea of when recursion should be used to solve problems, we will go through what should be expected from this course.