Recursion – Is when a function calls itself. It is used to evaluate a term in a sequence, by creating a function that defines the pattern.

Requirement: In order to create a recursive function, you must understand the subproblem. A solution to a complex problem can be found by breaking it down into repeatable subproblems, and repeatedly calling until the base case is found. Once the base case has been solved, all subproblems can be solved, by using the results of the base case.

3 Requirements for Recursion:

1. **Base Case** (end condition) – The base case is normally used to return a value once the end term in a sequence has been reached. No other case should return a value, or the recursive calls will stop immediately.
2. **Recursive Call** - Where the function calls itself. The recursive call must have an element that makes the element size smaller, so it can converge to the base case.
    a. Recursive calls should not address smaller overlapping components.

3. **Mathematical Equation - The goal is to come up with a mathematical equation that resolves the problem, by breaking the main problem into a sequence of smaller recursive calls that converge to the base case.** Once the base case is reached, the results will be plugged back into the larger mathematical equation to solve for the unknown. Note: It is like solving for X.

Recursion has nothing to do with iteration (Loops), it is how a program repeats by calling itself:

Strategy for Solving:

1. First determine what data you are iterating over.
2. You must figure out how to narrow the element size, so you get closer to the base case.
3. Base case will always return a result.
4. Determine your base case, sometimes it is easier to think of this last.

Common Mistakes:

1. **Simple Recursion:**

    a. If you are adding data together from recursive calls you must return it in an equation like fibonacci. Use **return fib(n-2) +fib(n-1);**

2. **Binary Tree and Recursion:**
    a. When traversing through a tree, you set the node parameter to the current node's child during the recursive call. Use **validate(node.left, min, max);**
    b. **For data to be used to evaluate the base condition, it must be passed to the recursive call. This will allow the data to converge until the base case is true.** Use **validate(node.left, min, node.data);**

//For great example see Stephen Grider- UDEMY - The Coding Interview Bootcamp - Problems.

The Coding Interview Bootcamp - Examples:

1. **validateBSTOrder (best)**
2. Stairs.

3. Spiral.
4. linkedList

FullStory Example:

1. Math