Week 2 Day 1: Recursion

<table>
<tr><td>Conceptualize</td><td>Example 1: the movie “Inception”:
- Leonardo Dicaprio finds himself in a dream within a dream within a dream… so on
- How does he know which layer of dream he is in? → he has a totem, an item that is used to determine if he is in reality or not (in Leo’s case, the spinning top)
- What if your totem tells you you’re in reality? You’re good to go!
- What if your totem tells you you’re in a dream? You kick yourself out of that dream… and then you check your totem again (repeat as necessary)</td><td>Example 2: line
- Imagine you have been lined up with a large group of other people by height-order, and you are facing a wall → unless you are the first or last person in line, you don’t know where in the line you are
- How do you know what number in line you are? → ask the person next to you if he/she is first
- If they say ‘yes’, you now know your number!
- If they say ‘no’, you have them ask the next person… (repeat as necessary, keep track of how many people have been asked before reaching the end)</td></tr>
<tr><td>Definition</td><td>A recursive function is a function that calls/invokes itself until it reaches a <u>base case</u>
- A base case is the condition that will stop the function from calling itself again
- If there is no base case, your function will call itself indefinitely, creating an infinite loop</td><td>Base case for inception: the totem tells you you’re in reality → you no longer have to kick yourself out of a dream

Base case for the line: the person says ‘yes, I am the first in line’ → you no longer have to ask if the next person is first</td></tr>
<tr><td>Basic Example</td><td>Recursion is an alternative to iteration.

Countdown function using iteration:

function countdownIteratively(num) {
for (var i = num; i > 0; i--) {
console.log(i);
}
console.log(‘Done iterating!’);
}</td><td>Countdown function using recursion:

function countdownRecursively(num) {
console.log(num);
if (num > 0) {
countdownRecursively(num-1);
}
else {
console.log(‘Done recurring!’);
}
}</td></tr>
<tr><td>How does it work?</td><td>When a function is called within another function (the outer function known as the <u>parent function</u>), the inside function must complete before the parent function can complete.
How is this done?...</td><td></td></tr>
<tr><td>The Call</td><td>The call stack helps the computer keep track of</td><td>Example: A stack of pancakes</td></tr>
</table>

Stack	which function has to wait and which can complete. - When a function is called, it is added to the top of the stack. - When a function finishes executing, it is "popped" off the stack.	- Imagine making pancakes and placing them one on top of the other (a stack of pancakes). When you eat them, you eat the top one first and so on until you finish the last pancake. In this pattern, the last pancake you make is the first you eat. And the first pancake you make is the last you eat.
Steps to write a recursive function	1. Define the base case. Ask yourself, "When should the function end?" (e.g. Inception -- when the totem tells you you're in reality) 2. Define the recursive case. "When should the function call on itself?" 3. Modify your code as necessary.	
Tips	- When testing your code, start with small inputs - If you're stuck, try writing the function iteratively first (using a loop) - Use pen and paper to help yourself visualize and track	Example challenge: Write a recursive function that sums up all of the numbers in an array