## Important Reminder

* We'll meet at ***9 am Tuesday next week for the first day in the office***. Please make sure you sign up on Eden before the weekend.

## Key Points

### **Recursion**

def say\_hey(num\_times)  
 return if num\_times <= 0   
  
 puts "Hey y'all!"  
 say\_hey(num\_times - 1)  
end

Recursion is a fun and crazy topic that gets easier as you solve more problems with recursion! We will revisit recursion next Wednesday for additional practice.

Recursion is when a method calls itself; we can used this idea to solve problems that would otherwise be very difficult to solve iteratively. When creating a recursive method, there are two important things to keep in mind (otherwise you will get an infinite loop, and eventually a stack overflow error). Your recursive method will need:

1. **Base case:**this is when the method will return (it will stop calling itself). You'll need at least 1 base case, or condition. Think of this like the i < 100 in a while loop.
2. **Recursive** (or inductive) **step:**this is where the method calls itself, going towards a base case. Think of this like the i += 1 in a while loop.

In the say\_hey method above:

* The base case is when num <= 0, we will return and stop calling itself.
* The inductive step is when we call the method itself, going towards the base case: say\_hey(num\_times - 1) (we are subtracting by one until we eventually hit 0, the base case)

To illustrate what happens when we call: say\_hey(3)

* num\_times is 3. Is 3 <= 0? false
* puts "Hey y'all"
* calls say\_hey(3 - 1). What does this evaluate to? Let's find out by exploring say\_hey(2):
  + num\_times is 2. Is 2 <= 0? false
  + puts "Hey y'all"
  + calls say\_hey(2 - 1). What does this evalutate to? Let's find out by exploring say\_hey(1):
    - num\_times is 1. Is 1 <= 0? false
    - puts "Hey y'all"
    - calls say\_hey(1 - 1). What does this evalutate to? Let's find out by exploring say\_hey(0):
      * num\_times is 0. Is 0 <= 0? false so let's return
      * We now know what say\_hey(0) is!
    - We now know what say\_hey(1) is => *Hey y'all*
  + We now know what say\_hey(2) is => Hey y'all
* We now know what say\_hey(3) is! => Hey y'all
* So our final output should be   
  Hey y'all  
  Hey y'all  
  Hey y'all

A lot of the exercises today were review of the material you've learned over the past two weeks. There was some new stuff though!

### **Spaceship operator <=>**

The spaceship operator is out of this world, and quite useful. It can be used to compare if an object is less than, equal to, or greater than another object; all in one go! For x <=> y, it will return:

* -1 if x < y
* 0 if x == y
* 1 if x > y

### **Truthy and Falsey values**

In Ruby, everything has a truthiness value. All objects are truthy, except false and nil.

### || operator

When we think about logical OR, we typically read as "If either or both are true, return true" and "If neither are true, return false". This is logically correct, but with ruby we get the last object evaluated back: For banana || mango

* If banana is true, return banana
* If mango is false, return mango

With this knowledge, we can do some cool stuff, like lazy assignment: fruit = fruit || "banana"

* This will set fruit to "banana" if fruit hasn't been defined (the value will be nil, which is falsey)
* Syntactic sugar allows us to write it as fruit ||= "banana"
* This pattern is super useful for lazy initialization of instance variables in classes