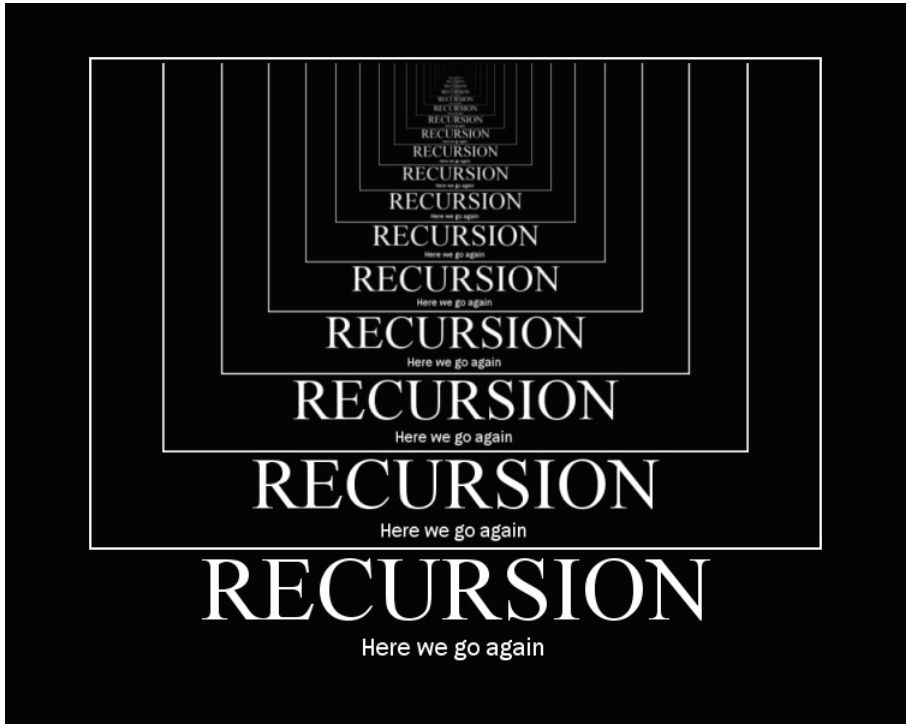
Recursion EXPLAINED!

So we had this recursive method:

```
def recurse_me(times)
   if times == 0
     puts "Base Case 0!"
   else
     puts "Before: #{times}"
     recurse_me(times-1)
     puts "After: #{times}"
   end
end
```



But how does that really work?

let's call recurse me: recurse_me(3)

when times is 3

```
if times == 0
  puts "Base Case 0!"
else
  puts "Before: #{3}"
  recurse_me(3-1)
  puts "After: #{3}"
end
```

when times is 2

```
if times == 0
  puts "Base Case 0!"
else
  puts "Before: #{2}"
  recurse_me(2-1)
  puts "After: #{2}"
end
```

when times is 1

```
if times == 0
  puts "Base Case 0!"
else
  puts "Before: #{1}"
  recurse_me(1-1)
  puts "After: #{1}"
end
```

when times is 0

```
if times == 0
  puts "Base Case = 0!"
else
  puts "Before: #{times}"
  recurse_me(times-1)
  puts "After: #{times}"
end
```

When the call to recurse_me happens, we pause the current method invocation to call it with the next number (shown above with the bracket). It keeps calling and pausing until a base case is reached, then comes back up via a return and continues the method where it was paused. It keeps returning and continuing up the chain till it reaches the top most call.

Another way to visualize the execution of recurse_me(3)

with times = 3

recurse_me(3)

```
if 3 == 0
    puts "Base Case 0!"
else
    puts "Before: #{3}"

    if 2 == 0
        puts "Base Case 0!"
    else
        puts "Before: #{2}"

        if 1 == 0
            puts "Base Case 0!"
        else
            puts "Before: #{1}"

            if 0 == 0
                puts "Base Case 0!"
            else
                # this never gets executed
                # because BASE CASE

            end

            puts "After: #{1}"
        end

        puts "After: #{2}"

    end

    puts "After: #{3}"
end
```

recurse_me(2)

recurse_me(1)

recurse_me(0)

If we fill in the calls to recurse_me with the code inside the method... we can visualize it as one big long method.