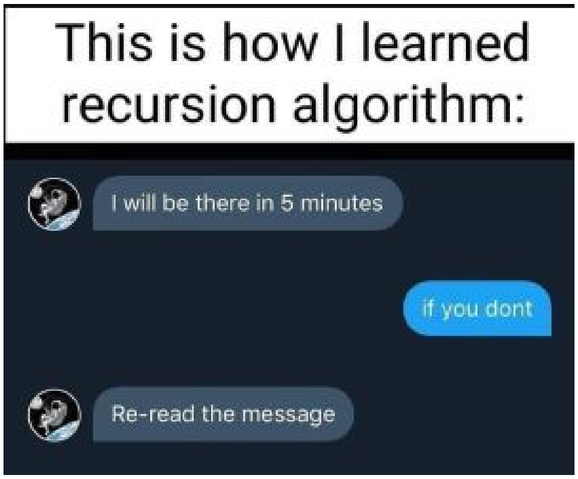
Introduction to Recursion

- In this lecture we will cover
    - Basic recursion
        - Base Case
        - Recursive Case

When you first learn recursion



This is how I learned recursion algorithm:

I will be there in 5 minutes

if you dont

Re-read the message

When you finally understand it

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:

    - Computer science has a number of concentrations.
        - General concentration is the most flexible, and even allows students to double major or minor pretty easily.
        - Software Engineering
        - Computing Systems

        - Human Centered Computing
        - Networks and Security
        - Artificial Intelligence
        - Computer Science Education.
    - Minors:
        - Minor in Computer Science - choose your own adventure minor
        - Minor in Machine Learning - popular with stats/math, and engineering
        - Minor in Bioinformatics - Biology + Computer Science

Warmup Activity: Loops

Write a method that calculates the exponent of a value. For example

```
exponent(5, 5); // returns 3,125
exponent(5, 2); // returns 25
```

You can start with an empty file, and it can be a static method

In [5]:

```
public static int exponent(int value, int exp) {
    int answer = 1;
    for(int i = 0; i < exp; i++) {
        answer *= value;
    }
    return answer;
}

System.out.println(exponent(5,5));
System.out.println(exponent(5,2));
```

```
3125
25
```

Simple Recursion

- Recursion
    - A method that calls itself
    - Another way to repeat!
- How to write it?
    - Write a base case!
    - Write the recursive case
        - the method calling itself, with a slight modification to the parameters!

Factorial Example

Take the following iterative code to determine a factorial

In [8]:

```
public static long factorialLoop(int n) {
    long fact = 1;
    for(int i = n; i > 1; i--) {
        fact *= i;
    }
    return fact;
}

System.out.println(factorialLoop(4)); // 4 * 3 * 2 * 1
System.out.println(factorialLoop(5)); // 5 * 4 * 3 * 2 * 1
System.out.println(factorialLoop(6));
```

```
24
120
720
```

Build Factorial Recursively

First think about the format / pattern to factorial

$$n! = n * (n-1)!$$

Meaning the factorial of any n is n times the factorial of $n-1$! This creates a pattern to exploit. Furthermore

$$1! = 1$$
$$0! = 1$$

Using this, we build a base case

```
if(n <= 1) return 1;
```

and then we can build the recursive call

```
return n * factorial(n-1);
```

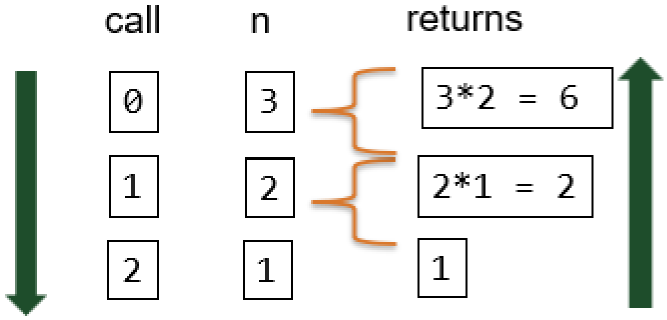
Let's put it together

In [9]:

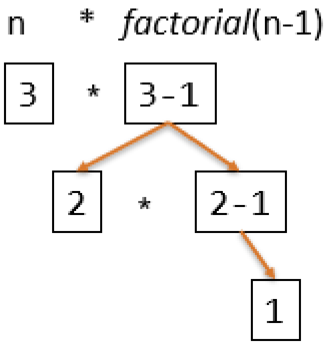
```
public static long factorial(int n) {
    if(n<=1) return 1; // base case first!
    return n * factorial(n-1);
}

System.out.println(factorial(4));
System.out.println(factorial(5));
System.out.println(factorial(6));
```

```
24
120
720
```

Breaking it down



- each method is called, and 'stored' waiting until all methods are called
- This is called the 'stack'
- As such
  - n = 3 needs to wait until n=2, and n=1 completes, and gets the return value



Student Practice

Going back to exponent, write it recursively.

Remember, the exponent can be defined by the following pattern

$n^5 = n * n^4$

In [13]:

```
public static long exponent_rec(int n, int exp) {
    if(exp <= 1) return n;
    return n * exponent_rec(n, exp-1);
}

System.out.println(exponent_rec(5, 2));
System.out.println(exponent_rec(5, 3));
System.out.println(exponent_rec(5, 5));
```

```
25
125
3125
```

More Complex: String Reverse

Take the following code.

DRAW it out on a piece of paper (you can use the tree or method call example).

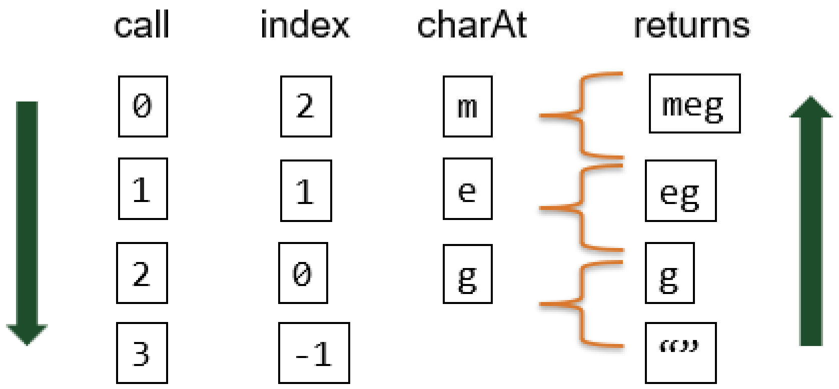
Document every method call!

In [14]:

```
public static String reverseString(String str) {
    return reverseString(str, str.length()-1);
}

public static String reverseString(String str, int index) {
    if(index < 0) return "";
    return str.charAt(index) + reverseString(str, index-1);
}

System.out.println(reverseString("gem"));
```

```
meg
```

how can it look?



call	index	charAt	returns
0	2	m	meg
1	1	e	eg
2	0	g	g
3	-1		“”

More student practice

Take an int array, and total all the values in the integer array, using *Recursion*. You may do similar as above and have overloaded method calls!

In [16]:

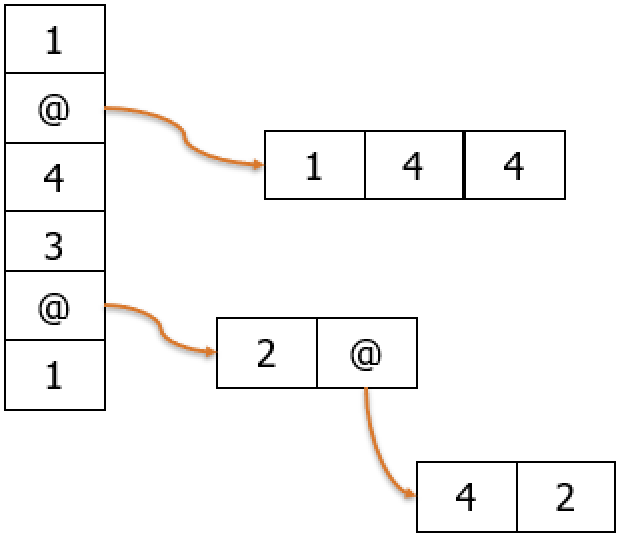
```
public static int sum(int[] values) {
    return sum(values, 0);
}

public static int sum(int[] values, int index) {
    if(index >= values.length) return 0;
    return values[index] + sum(values, ++index);
}


int[] vals = {10, 12, 13, 10, 12, 15};
```

```
System.out.println(sum(vals));
```

```
72
```

Why Recursion

- But we can just use loops!
    - For these examples yes
    - But what about more complex 'tree' like structures.
    - For example:



-
    - How many for loops do you use, if you didn't know how many other lists there were?
        - This actually is a type of 'tree' structure
        - Explored more in depth in CS 165
- Other reasons?
    - What if you only know the *next* element, and you don't have access to all the elements!
        - Recursion to the rescue!
- However, it isn't a cure all
    - can be slower
    - Have to pick and choose what is best.

Solution to the any depth of arrays summation below!

In [21]:
```
public static int sum_array(Object[] values) {
    return sum_array(values, 0); // overload, for easier initial call
}
public static int sum_array(Object[] values, int current) {
    if(current >= values.length) return 0; // past end of array, return 0
    if(values[current] instanceof Object[]) // another array!
        return sum_array((Object[])values[current], 0) + sum_array(values, current+1);
    return (Integer)values[current] + sum_array(values, current+1); // number plus son
}
```

```java
Object[] values = {1, 2, 3, new Object[]{4, 5, new Object[]{1,1}}, 10, new Integer[]{2
System.out.println(sum_array(values));
```

```
42
```