Introduction to Recursion

In this lecture we will go over the basics of Recursion.

What is Recursion?

There are two main instances of recursion. The first is when recursion is used as a technique in which a function makes one or more calls to itself. The second is when a data structure uses smaller instances of the exact same type of data structure when it represents itself. Both of these instances are use cases of recursion.

Recursion actually occurs in the real world, such as fractal patterns seen in plants!

Why use Recursion?

Recursion provides a powerful alternative for performing repetitions of tasks in which a loop is not ideal. Most modern programming languages support recursion and recursion serves as a great tool for building out particular data structures.

We'll start this introduction with an example of recursion- a factorial function.

Factorial Example

In this part of the lecture we will explain recursion through an example excercise of creating the factorial function. The factorial function is denoted with an exclamation point and is defined as the product of the integers from 1 to *n*. Formally, we can state this as:

$$n! = n \cdot (n-1) \cdot (n-2) \ldots 3 \cdot 2 \cdot 1$$

Note, **if n = 0, then n! = 1**. This is important to take into account, because it will serve as our *base case*.

Take this example:

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24.$$

So how can we state this in a recursive manner? This is where the concept of **base case** comes in.

Base case is a key part of understanding recursion, especially when it comes to having to solve interview problems dealing with recursion. Let's rewrite the above equation of 4! so it looks like this:

$$4! = 4 \cdot (3 \cdot 2 \cdot 1) = 24$$

Notice that this is the same as:

$$4! = 4 \cdot 3! = 24$$

Meaning we can rewrite the formal recursion definition in terms of recursion like so:

$$n! = n \cdot (n-1)!$$

Note, **if n = 0, then n! = 1**. This means the **base case** occurs once n=0, the *recursive cases* are defined in the equation above. Whenever you are trying to develop a recursive solution it is very important to think about the base case, as your solution will need to return the base case once all the recursive cases have been worked through. Let's look at how we can create the factorial function in Python:

In [1]:
```
def fact(n):
    '''
    Returns factorial of n (n!).
    Note use of recursion
    '''
    # BASE CASE!
    if n == 0:
        return 1

    # Recursion!
    else:
        return n * fact(n-1)
```

Let's see it in action!

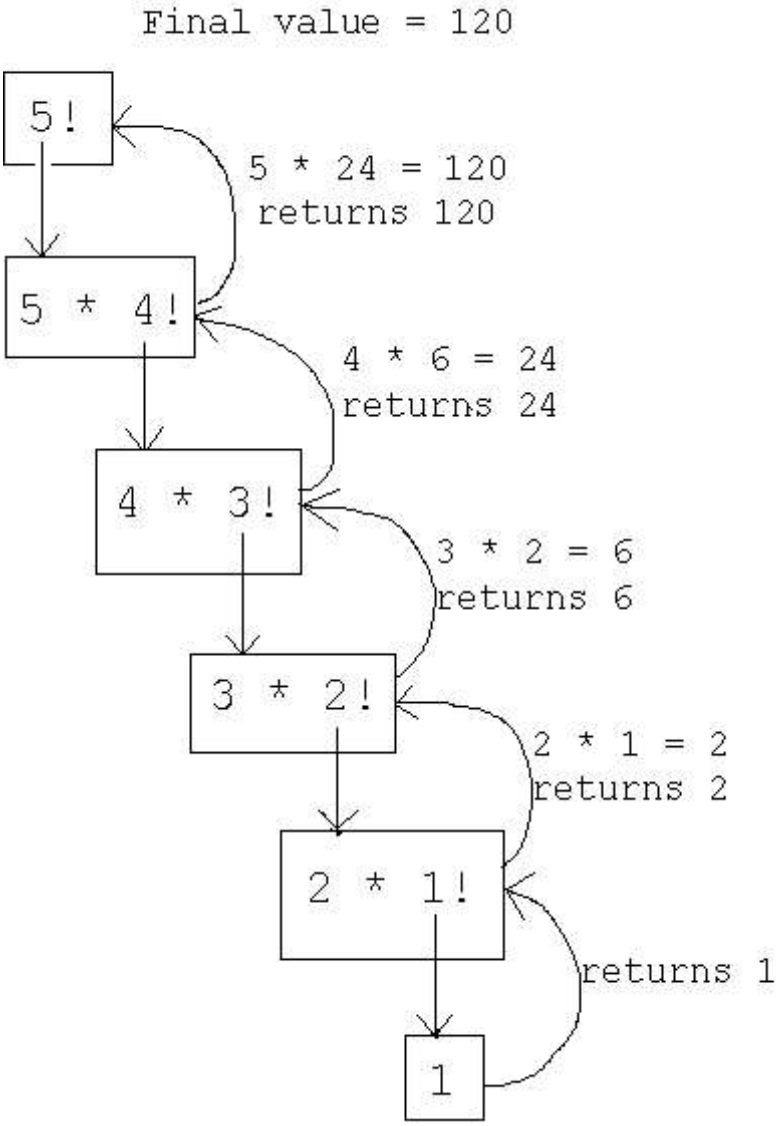
In [2]:
```
fact(5)
```

Out[2]:
```
120
```

Note how we had an if statement to check if a base case occured. Without it this function would not have successfully completed running. We can visualize the recursion with the following figure:

In [3]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= 'http://faculty.cs.niu.edu/~freedman/241/241notes/recur.gif')
```

Out[3]:



We can follow this flow chart from the top, reaching the base case, and then working our way back up.

Conclusion

Recursion is a powerful tool, but it can be a tricky concept to implement. In the next lectures we will go over a few more example problems for recursion. Then afterwards you'll be faced with some real interview questions involving recursion!