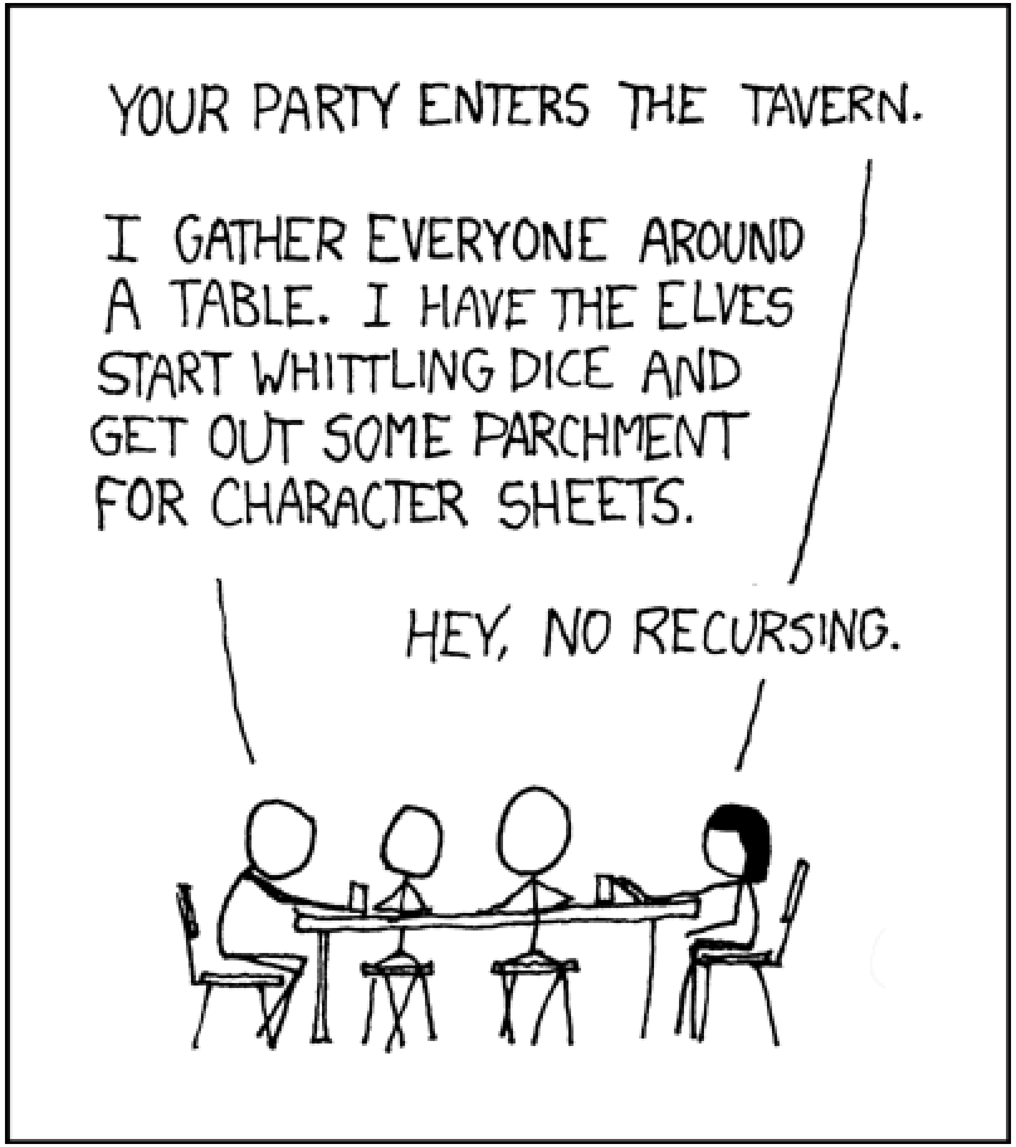
CS 161, Lecture 13: Recursion


YOUR PARTY ENTERS THE TAVERN.
I GATHER EVERYONE AROUND A TABLE. I HAVE THE ELVES START WHITTLING DICE AND GET OUT SOME PARCHMENT FOR CHARACTER SHEETS.
HEY, NO RECURSING.

What is Recursion?

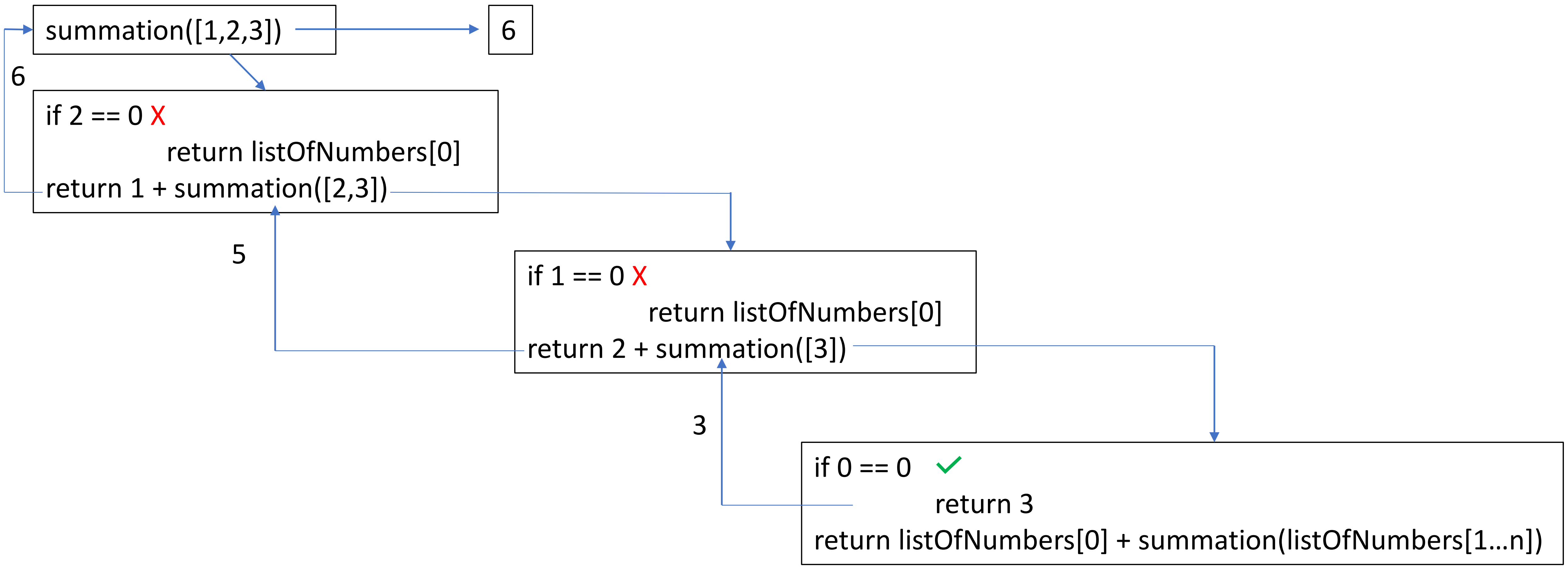
- When a function calls itself one or more times (directly or indirectly)
- Form of repetition
- Typically used to perform same operation on a smaller subset and then build the result based on what is returned from the smaller case
- Typically has at least one base case for stopping
- Based on inductive logic

Iteration vs. Recursion

- Anything that can be done iteratively can be do recursively and vice versa
  - Not always a good idea, some problems naturally lend themselves to one mode of thinking or the other

How it works on a high level

```
summation(listOfNumbers[0…n])
        if n == 0
                return listOfNumbers[0]
        return listOfNumbers[0] + summation(listOfNumbers[1…n])
```



summation([1,2,3])

6

```
if 2 == 0 X
        return listOfNumbers[0]
return 1 + summation([2,3])
```

```
if 1 == 0 X
        return listOfNumbers[0]
return 2 + summation([3])
```

```
if 0 == 0 ✓
        return 3
return listOfNumbers[0] + summation(listOfNumbers[1…n])
```

Pros and Cons

- Pros
  - Readable
  - Sometimes easier to conceptualize for problems that have many moving parts
- Cons
  - Efficiency
  - Memory usage
    - Each call to the function makes a new function stack frame (see previous slide)

Example: Factorial

- The product of an integer and all that come before it
- n! = n * (n-1) * (n-2) * … * (n-(n-1)) * 1 for all n > 0
- Base Case: 0! = 1

Iterative Factorial

```
int factorial(int n) {
    int fact;
    if (n == 0)
        fact = 1;
    else
        for (fact = n; n > 1; n--)
            fact = fact * (n-1);
    return fact;
}
```

Recursive Factorial

```
int factorial (int n) {
        if (n == 0)
                return 1;
        return n * factorial(n-1);
}
```

Code Demo