Welcome to 61A Lab!

We will begin at **5:10**!

Slides: **cs61a.bencuan.me**

Announcements

- Cats released!
  - Checkpoint Thurs, whole proj. next Thurs
- HW3 due Thurs.
- Regrade requests due tomorrow!
- CSM sections still open!

The Plan

- Recursion review
- Tree recursion
- Work time!
  - HW and project problems accepted

Recursion Review

What is recursion?

Recursion is **when a function calls itself**

Remember to call the function!!

```
def factorial(n):
    """Return the factorial of N, a positive integer."""
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

```
def print_all(x):
    print(x)
    return print_all
```

This is NOT recursion (no call)!

The 3 parts of a recursive function

1. **Base Case** (what's the simplest possible input?)

2. **Recursive Case** (how do we make the problem even simpler?

3. **Recursive Leap of Faith** (assume simpler problems are solved already)

A basic recursion skeleton

```
1 def canary(birb):
2     if birb is some_simple_value:        # base case
3         return another_simple_value
4     else:                                # recursive case
5         return canary(smaller_birb)
```

Tree Recursion

What is tree recursion?

- Definition: **making multiple recursive calls at one time**
  - Can be very challenging in practice!

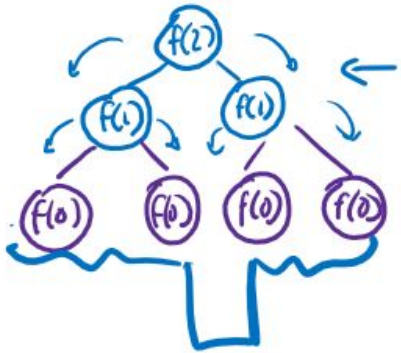
```
def f(x):
    if <base case condition>:
        return <result>

    else:
        return f(<recursive step>) + f(<another recursive step>)
```

(could be any operation)

What is tree recursion?

- Creates a branching structure:



← # of recursive calls grows exponentially

← reaches base case multiple times!

```
def f(x):
    if <base case condition>:
        return <result>

    else:
        return f(<recursive step>) + f(<another recursive step>)
```

(could be any operation)

When use tree recursion?

- When you need to try lots of possible combinations that rely on previous states
  - Fibonacci
  - Count coins
  - Coming soon(?): data structures (trees, heaps, graphs... and some sorting too ...)

Partitions

- A very common tree recursion pattern:
  - You're given two (or more) options
  - You need to combine the two options together

- Example: Line Stepper (lab q2)
  - Option 1: try going left
  - Option 2: try going right
  - Combine: add total count of two options

Lab Hints

Lab Hints

- Before you begin coding, figure out what the base case and recursive cases are!
- Recursive calls should ALWAYS move towards the base case
- Tree recursion: draw out the branching call structure
- Partition problems: ask yourself what possible moves you can make

Work Time!

go.cs61a.org/ben-queue

Want a lab partner/group?
Come to the front!