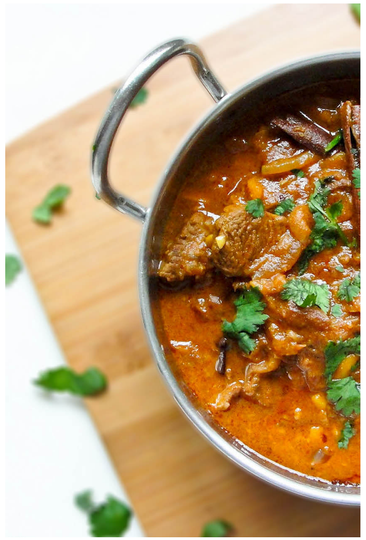
Recursion

Announcements

Curry Review

1) Partial Curry



(Demo)

2) Conditional Curry

(Demo)

cs61a.org/exam/su22/midterm/61a-su22-midterm.pdf#page=13

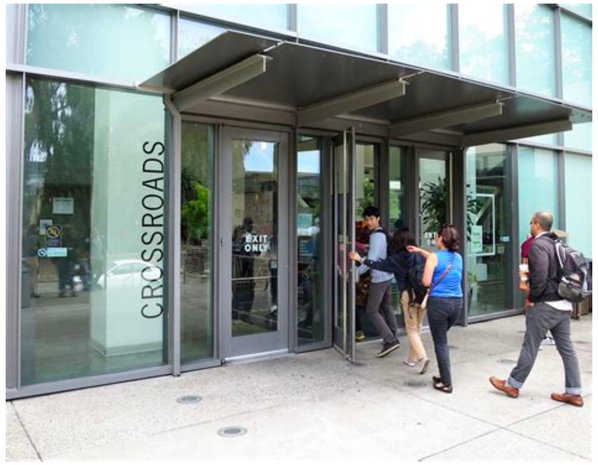
Analogy for Recursion

Analogy for Recursion

It's 6:30 PM! You rush to Crossroads Dining Hall and get in line. There are more people than you can count...

You can't step out of line because you'd lose your spot.

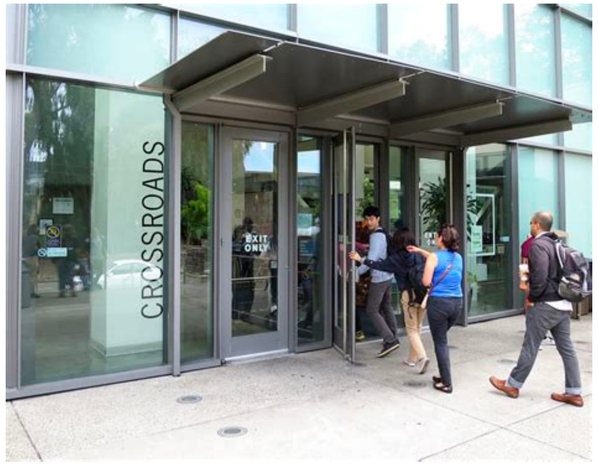
Problem: What is your position in line?



Analogy for Recursion

Iterative Solution:

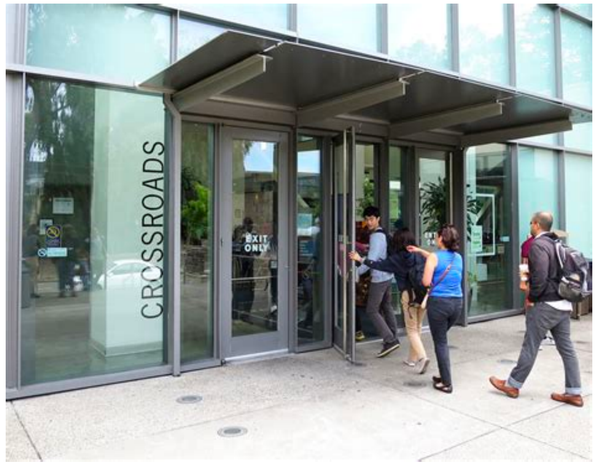
1. Ask a friend to go to the front of line.
2. Count each person in line one-by-one.
3. Then, come back and tell the answer.



Analogy for Recursion

***Recursive* Solution:**

- Person at front: Knows they're first
- Any other person: Ask the person in front of you, "what's your position in line?"
  - Once the person in front of you responds, you add one to their answer to get your position
- Result: Questions get asked from back-to-front; Answers return front-to-back.



Structure of Recursive Functions

Base case(s): the simplest instance of the problem that can be solved easily

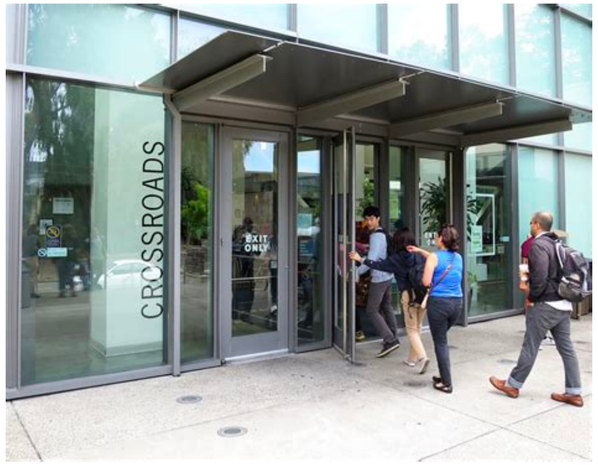
- If you're at the front of the line, you know you're first.

Recursive call: making a call to the *same* function with a *smaller* input

- Ask the person in front of you, "what's your position in line?"

Recombination: using the result of the recursive call to solve the original problem

- When the person in front of you tells you their answer, **add one to it to get the answer to your original question.**



Recursive Functions

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20fact%28n%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%20or%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20return%201%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20fact%28n-1%29%20*%20n%0A%0Afact%285%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Converting Iteration to Recursion

Recursive Nim

Rewrite play as a recursive function without a while statement.

- Do you need to define a new inner function? Why or why not? If so, what are its arguments?
- What is the base case and what is returned for the base case?

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    n = 0
    who = 0  # Player 0 goes first
    while n < goal:
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
    return who
```

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    def f(n, who):
        if n >= goal:
            return who
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
        return f(n, who)
    return f(0, 0)
```

Iteration <-> Recursion

- While loop continuation condition <-> Recursive stopping condition (base case)
- Additional variables used in iteration <-> Recursive function parameters
  - Variable initialization <-> Initial recursive call
  - Variable updates in loop <-> Argument updates in recursive call

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    n = 0
    who = 0  # Player 0 goes first
    while n < goal:
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
    return who
```

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    def f(n, who):
        if n >= goal:
            return who
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
        return f(n, who)
    return f(0, 0)
```

Converting Iteration to Recursion Practice

(Demo)