Recursion

Announcements

Recursive Functions

Recursive Analogy

It's 12:30 PM. You just finished listening very intently to CS61A recursion lecture (you are extra happy, as recursion is a fascinating topic). Immediately, you begin sprinting to the Golden Bear Cafe. Oh no! An incredibly long line has sprung up in front of it GBC. It’s so long that you can’t even begin to tell how long it will take to get through or if you’ll be able to make your 1:00 PM class. You want to find out how long this line is.

You can’t leave the line or else you’ll lose your spot.

How can you figure out the line length?

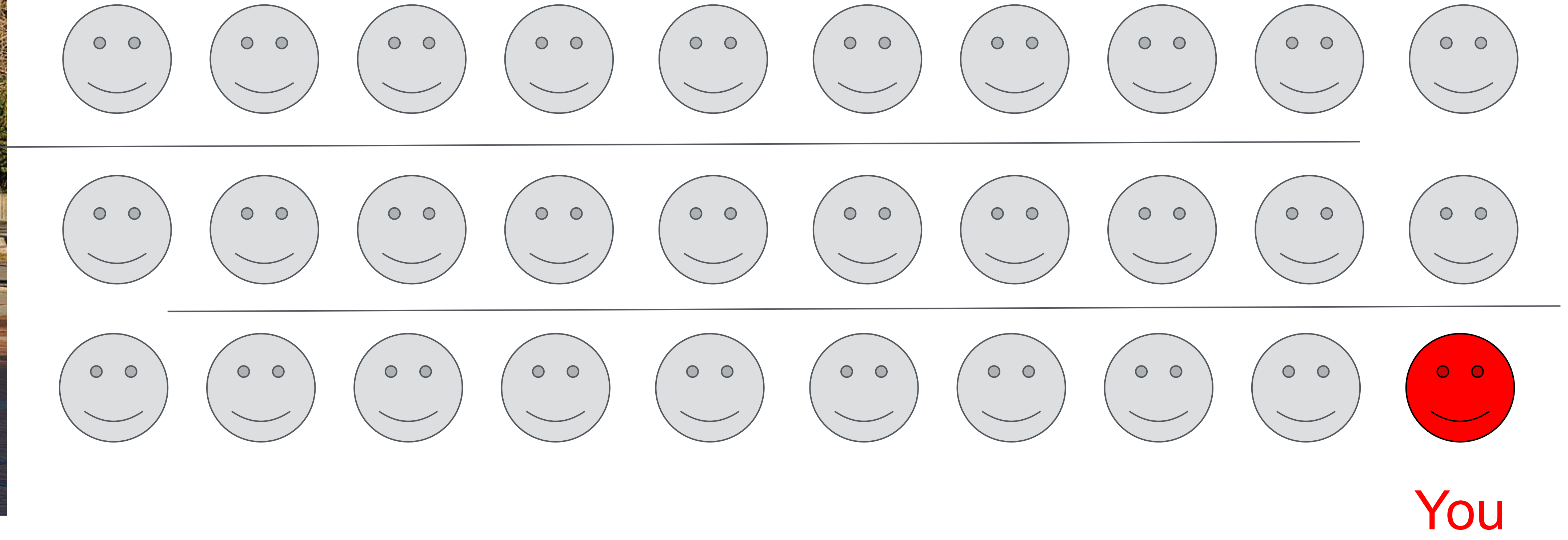


Photo by Chris Rycroft, https://creativecommons.org/licenses/by/2.0/deed.en

Recursive Analogy

Iterative approach

- Ask a friend to go to the front of the line
- Have them count each person till they get to you
- They'll tell you the answer

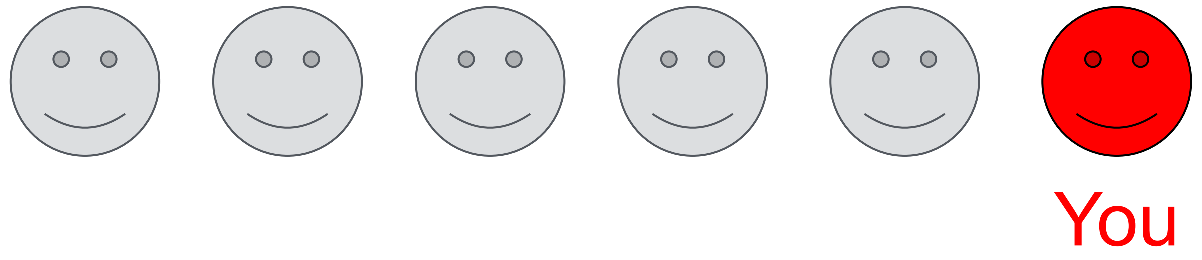
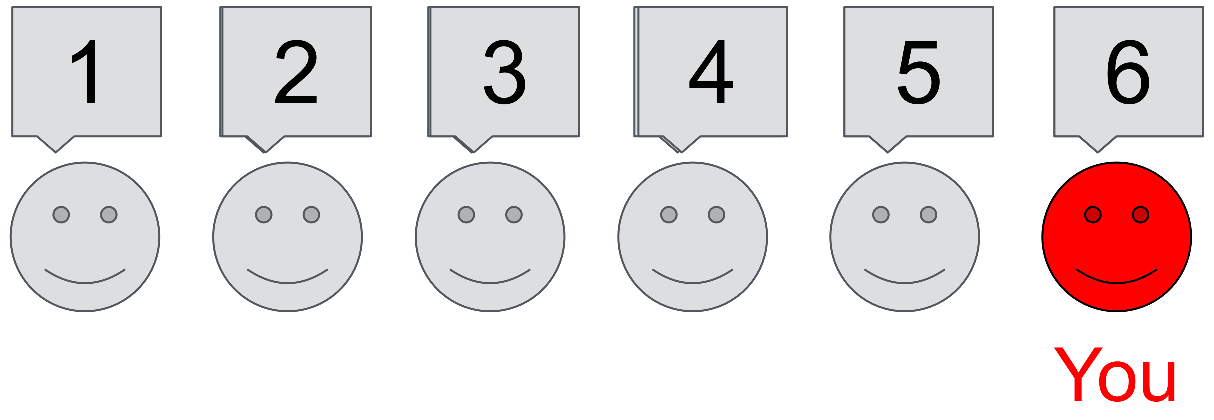


Photo by Chris Rycroft, https://creativecommons.org/licenses/by/2.0/deed.en

Recursive Analogy

Recursive approach

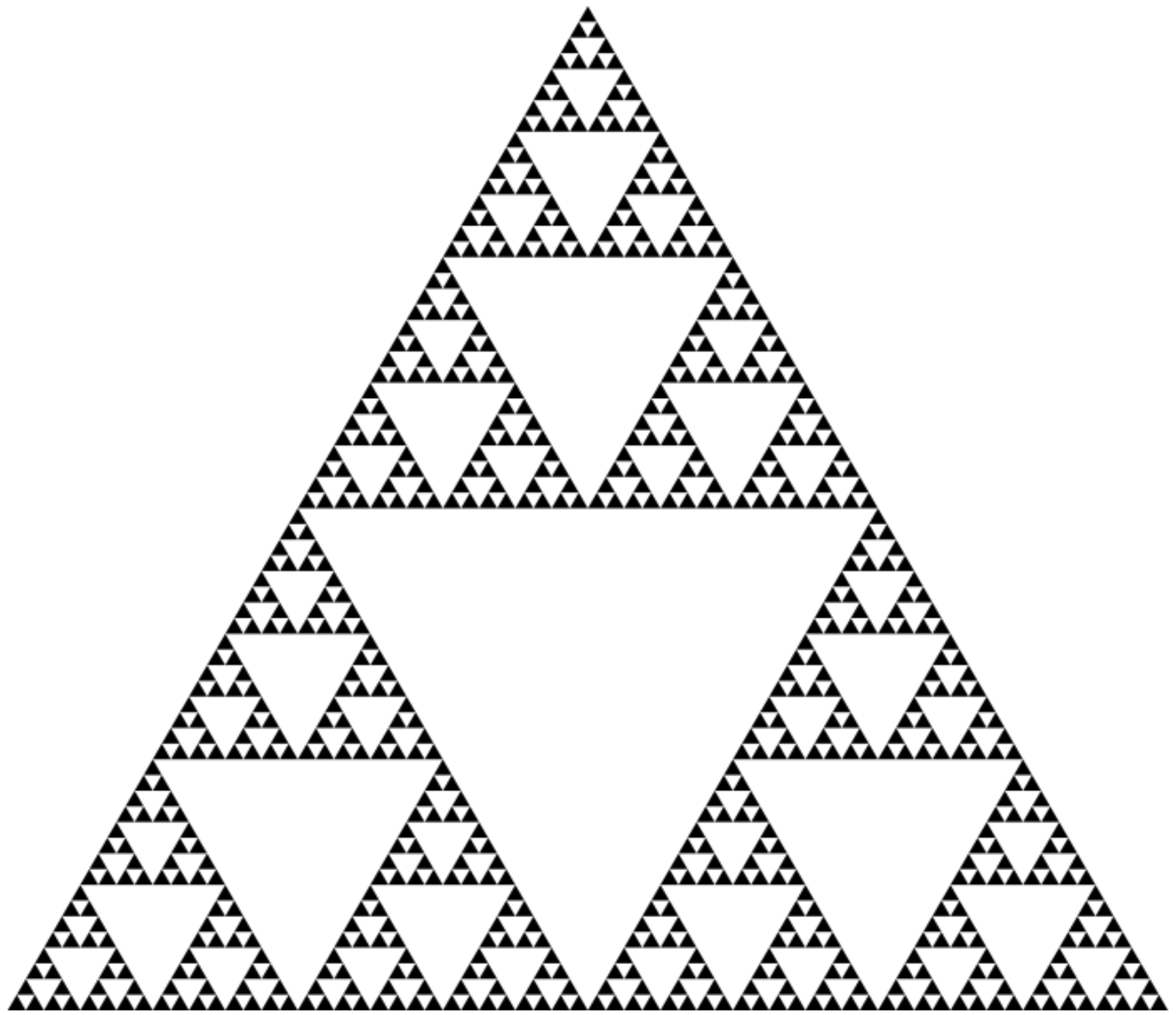
- You know the very first person in line can see that they are first
- For any other person, ask the person in front of them, “How many people are in front of you?”
    - The following person repeats this process
    - Once the person in front of you responds, add 1 to their answer

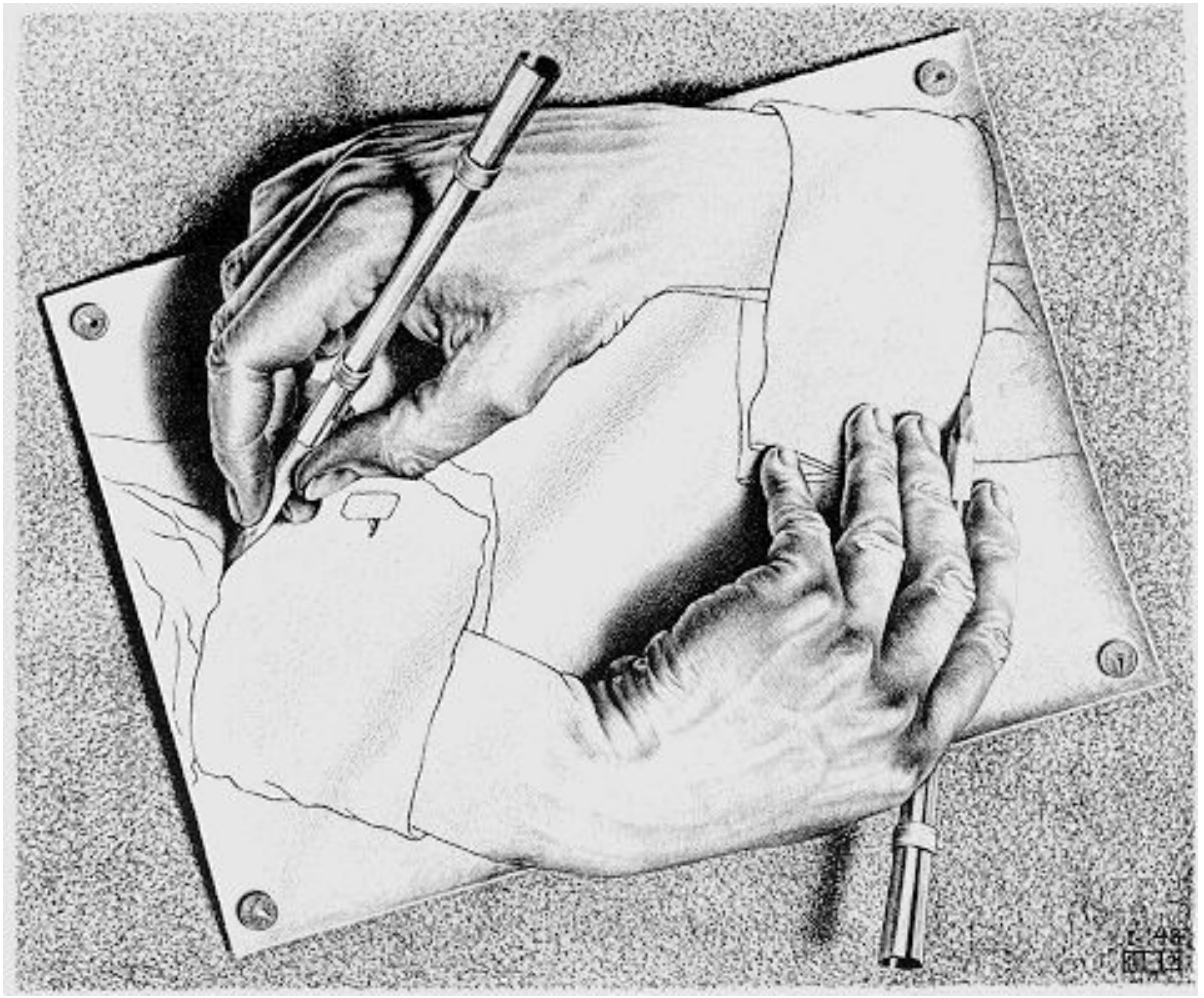


Recursive Functions

Definition: A function is called recursive if the body of that function calls itself, either directly or indirectly

Implication: Executing the body of a recursive function may require applying that function





Recursive Call Structure

Base case(s): the simplest instance of the problem that can be solved without much work

- 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, "How many people are in front of you?"

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.

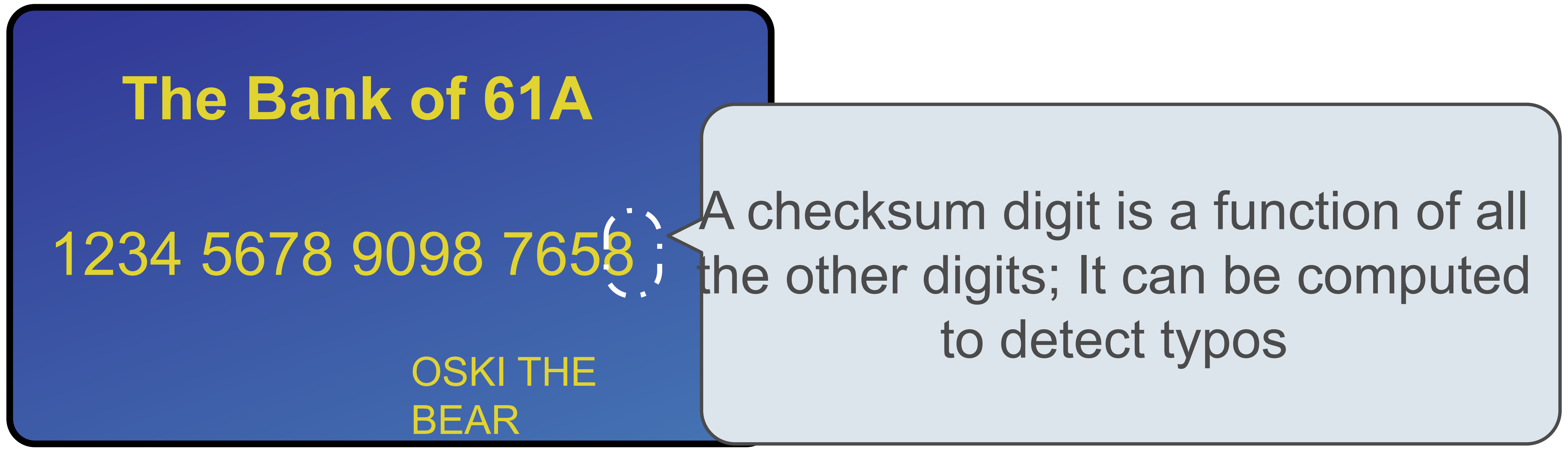
Example: Factorial

(Demo)

Sum Digits

2+0+2+3 = 7

- If a number a is divisible by 9, then sum_digits(a) is also divisible by 9
- Useful for typo detection!



- Credit cards actually use the Luhn algorithm, which we'll implement after sum_digits

The Problem Within the Problem

The sum of the digits of 6 is 6.

Likewise for any one-digit (non-negative) number (i.e., < 10).

The sum of the digits of 2023 is

202 3

Sum of these digits + This digit

That is, we can break the problem of summing the digits of 2023 into a smaller instance of the same problem, plus some extra stuff.

We call this recursion.

Sum Digits Without a While Statement

```
def split(n):
    """Split positive n into all but its last digit and its last digit."""
    return n // 10, n % 10


def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

The Anatomy of a Recursive Function

- The def statement header is similar to other functions
- Conditional statements check for base cases
- Base cases are evaluated without recursive calls
- Recursive cases are evaluated with recursive calls

```
def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

(Demo)

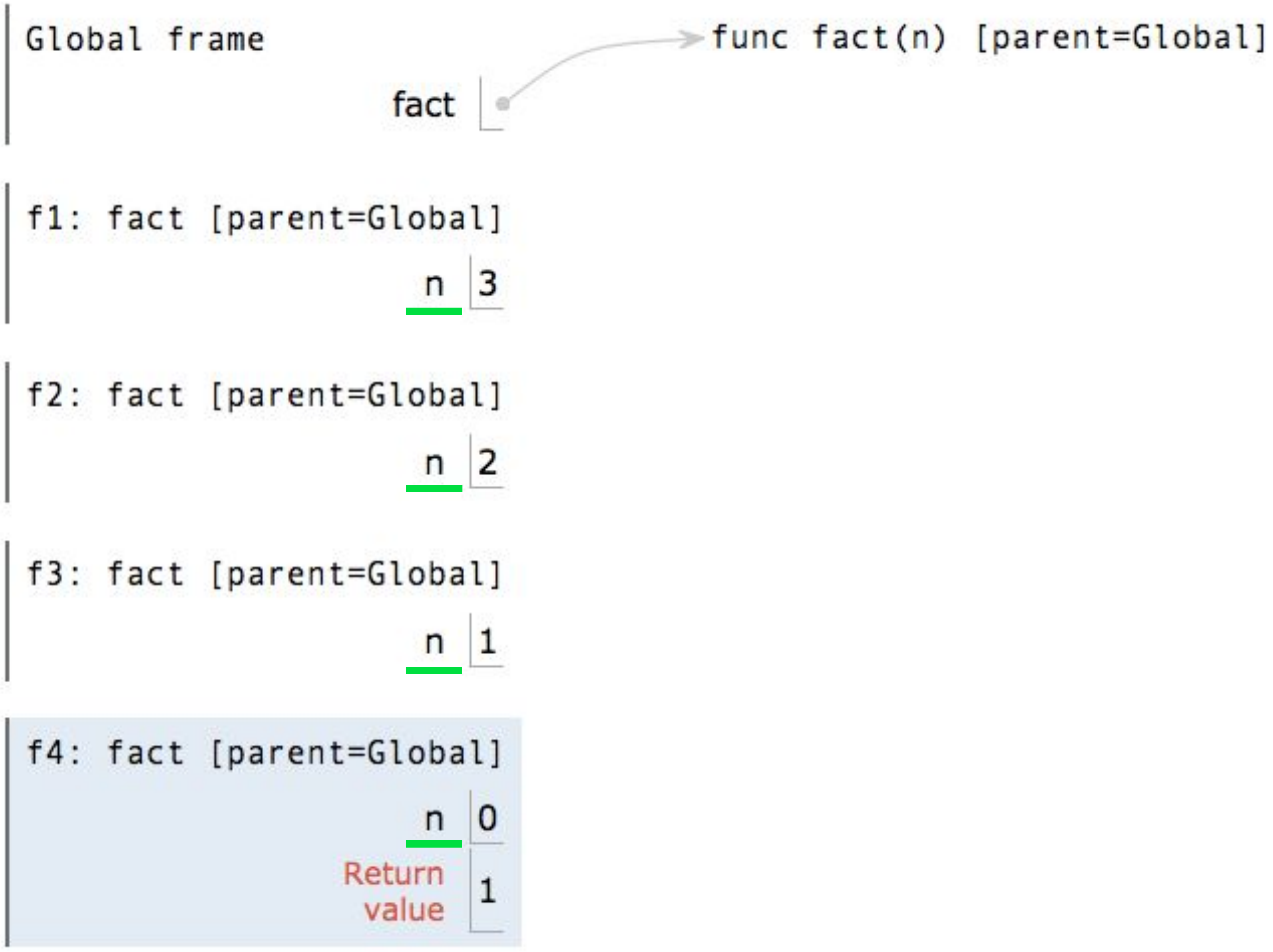
Recursion in Environment Diagrams

Recursion in Environment Diagrams

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)

fact(3)
```

(Demo)



- The same function **fact** is called multiple times
- Different frames keep track of the different arguments in each call
- What **n** evaluates to depends upon the current environment
- Each call to **fact** solves a simpler problem than the last: smaller **n**

Iteration vs Recursion

Iteration is a special case of recursion

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

Using while:

```
def fact_iter(n):
    total, k = 1, 1
    while k <= n:
        total, k = total*k, k+1
    return total
```

Using recursion:

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

Math:

$$n! = \prod_{k=1}^{n} k$$

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Names: ***n, total, k, fact_iter*** ***n, fact***

Verifying Recursive Functions

The Recursive Leap of Faith

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

Is fact implemented correctly?

1. Verify the base case
2. Treat **fact** as a functional abstraction!
3. Assume that **fact(n-1)** is correct
4. Verify that **fact(n)** is correct

Photo by Kevin Lee, Preikestolen, Norway

Mutual Recursion

The Luhn Algorithm

Used to verify credit card numbers

From Wikipedia: http://en.wikipedia.org/wiki/Luhn_algorithm

- **First:** From the rightmost digit, which is the check digit, moving left, double the value of every second digit; if product of this doubling operation is greater than 9 (e.g., 7 * 2 = 14), then sum the digits of the products (e.g., 10: 1 + 0 = 1, 14: 1 + 4 = 5)

- **Second:** Take the sum of all the digits

1	3	8	7	4	3	
2	3	1+6=7	7	8	3	= 30

The Luhn sum of a valid credit card number is a multiple of 10

(Demo)

Break

More Examples

Implementing a Function (Again!)

```
def remove(n, digit):
    """Return all digits of non-negative N
      that are not DIGIT, for some
      non-negative DIGIT less than 10.

    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    >>> remove(24313, 2)
    4313
    >>> remove(2431, 2)
    431

    """
    if ____base case____:
        return _base case return value_
    else:
        all_but_last, last = ____digit logic?____
        if __digit logic?__
            return __recursion?__
        return _recursion?_
```

Read the description

Verify the examples & pick a simple one

Read the template

Implement without the template, then change your implementation to match the template.
OR
If the template is helpful, use it.

Annotate names with values from your chosen example

Write code to compute the result

Did you really return the right thing?

Check your solution with the other examples

(Demo)

Implementing a Function (Again!)

```
def remove(n, digit):
    """Return all digits of non-negative N
      that are not DIGIT, for some
      non-negative DIGIT less than 10.

    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    >>> remove(24313, 2)
    4313
    >>> remove(2431, 2)
    431

    """
   if ________________:
       return __________
   else:
       all_but_last, last = ______________
       if ______________
           return __________
       return __________
```

4313 + 0

431 * 10 + 3

43 * 10 + 1

Read the description

Verify the examples & pick a simple one

Read the template

Implement without the template, then change your implementation to match the template.
OR
If the template is helpful, use it.

Annotate names with values from your chosen example

Write code to compute the result

Did you really return the right thing?

Check your solution with the other examples

Recursion and Iteration

Converting Recursion to Iteration

Idea: Figure out what state must be maintained by the iterative function.

```
def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

A partial sum

What's left to sum

(Demo)

Converting Iteration to Recursion

Idea: The state of an iteration are passed as arguments.

```
def sum_digits_iter(n):
    digit_sum = 0
    while n > 0:
        n, last = split(n)
        digit_sum = digit_sum + last
    return digit_sum
def sum_digits_rec(n, digit_sum):
    if n > 0:
        n, last = split(n)
        return sum_digits_rec(n, digit_sum + last)
    else:
        return digit_sum
```

Updates via assignment become...

...arguments to a recursive call

Summary

- Recursive functions
- Anatomy of recursive functions
  - Base case
  - Recursive case
  - Recombination
- Mutual recursion
- Relationship between iteration and recursion
- Implementing recursive functions