61A Lecture 7

Announcements

Hog Contest Rules

- Up to two people submit one entry; Max of one entry per person

- Your score is the number of entries against which you win more than 50.00001% of the time

- One more rule! *Pork Chop*

- All strategies must be deterministic, pure functions of the current player scores

- All winning entries will receive 2 points of extra credit

- The real prize: honor and glory

Fall 2011 Winners

Kaylee Mann
Yan Duan & Ziming Li
Brian Prike & Zhenghao Qian
Parker Schuh & Robert Chatham

Your name could be here FOREVER!

Fall 2012 Winners

Chenyang Yuan
Joseph Hui

Fall 2013 Winners

Paul Bramsen
Sam Kumar & Kangsik Lee
Kevin Chen

Fall 2014 Winners

Alan Tong & Elaine Zhao
Zhenyang Zhang
Adam Robert Villaflor & Joany Gao
Zhen Qin & Dian Chen
Zizheng Tai & Yihe Li

Spring 2015 Winners

Sinho Chewi & Alexander Nguyen Tran
Zhaoxi Li
Stella Tao and Yao Ge

Fall 2015 Winners...

cs61a.org/proj/hog_contest

Order of Recursive Calls
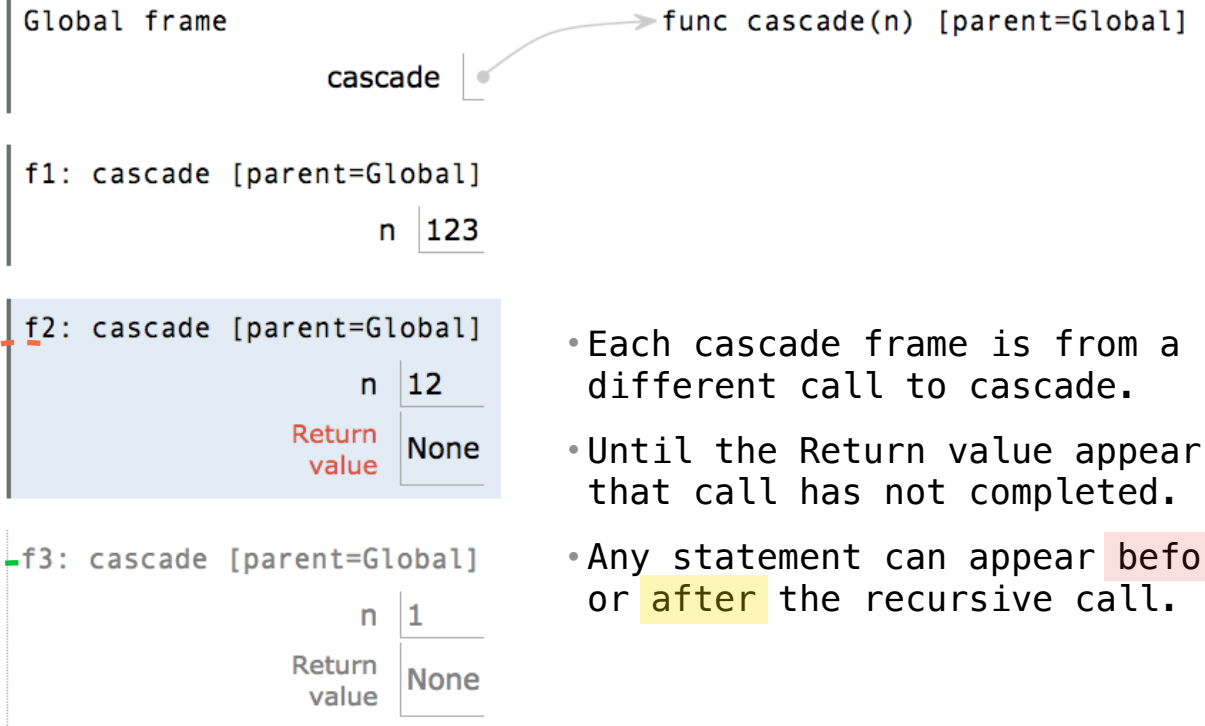
The Cascade Function

(Demo)

```
1  def cascade(n):
2      if n < 10:
3          print(n)
4      else:
5          print(n)
6          cascade(n//10)
7          print(n)
8
9  cascade(123)
```

Program output:
```
123
12
1
12
```

Global frame
                                        func cascade(n) [parent=Global]
        cascade

f1: cascade [parent=Global]
                        n 123

f2: cascade [parent=Global]
                        n 12
        Return   None
        value

f3: cascade [parent=Global]
                        n 1
        Return   None
        value

- Each cascade frame is from a different call to cascade.

- Until the Return value appears, that call has not completed.

- Any statement can appear before or after the recursive call.

Interactive Diagram

Two Definitions of Cascade

(Demo)

```python
def cascade(n):
    if n < 10:
        print(n)
    else:
        print(n)
        cascade(n//10)
        print(n)
```

```python
def cascade(n):
    print(n)
    if n >= 10:
        cascade(n//10)
        print(n)
```

- If two implementations are equally clear, then shorter is usually better
- In this case, the longer implementation is more clear (at least to me)
- When learning to write recursive functions, put the base cases first
- Both are recursive functions, even though only the first has typical structure

Example: Inverse Cascade

Inverse Cascade

Write a function that prints an inverse cascade:

```
1
12
123
1234
123
12
1
```

```python
def inverse_cascade(n):
    grow(n)
    print(n)
    shrink(n)


def f_then_g(f, g, n):
    if n:
        f(n)
        g(n)


grow =   lambda n: f_then_g(                    )
shrink = lambda n: f_then_g(                    )
```

Tree Recursion

Tree Recursion

Tree-shaped processes arise whenever executing the body of a recursive function makes more than one recursive call

```
    n:   0, 1, 2, 3, 4, 5, 6,  7,  8,       ... ,             35

fib(n):  0, 1, 1, 2, 3, 5, 8, 13, 21,       ... ,   9,227,465
```

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```
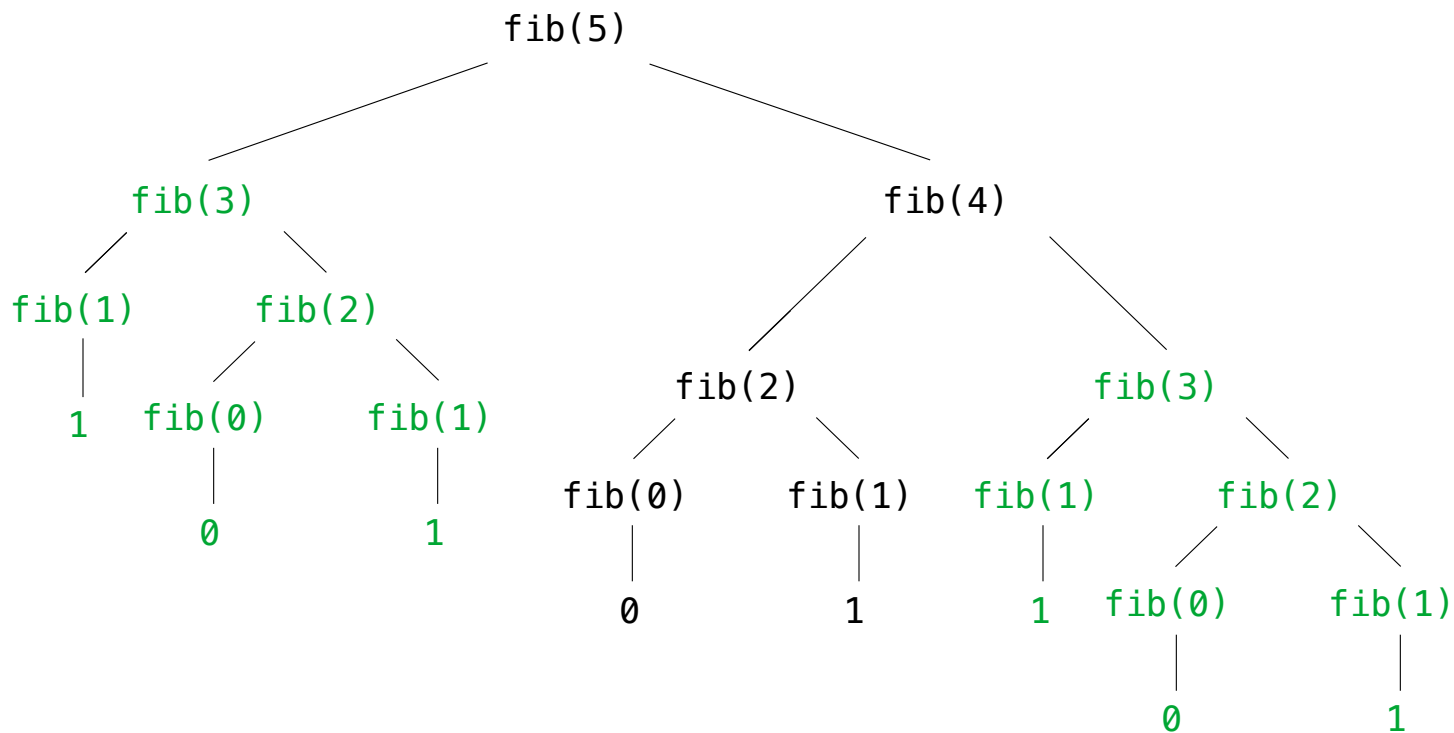
http://en.wikipedia.org/wiki/File:Fibonacci.jpg

A Tree-Recursive Process

The computational process of fib evolves into a tree structure
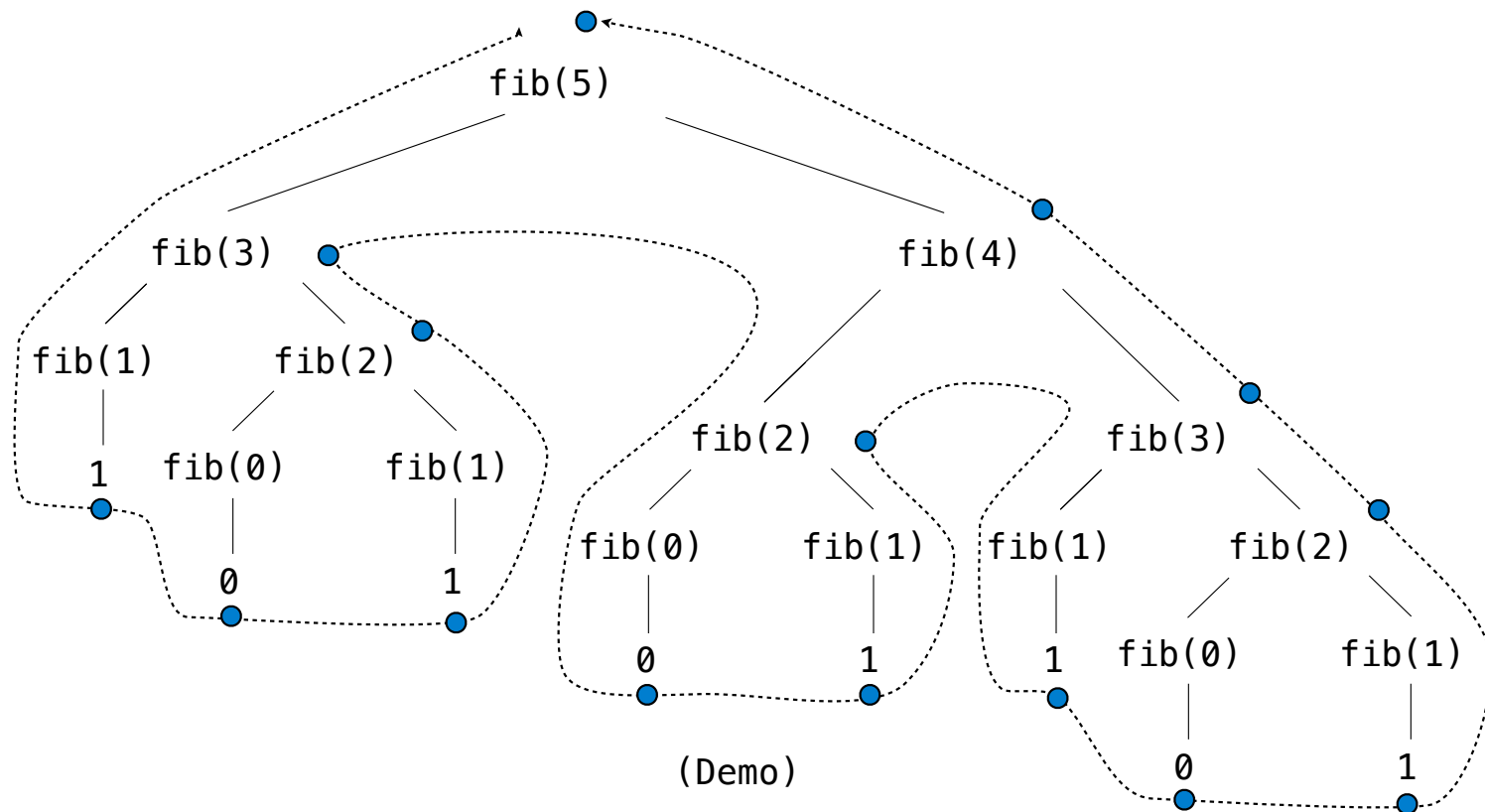
Repetition in Tree-Recursive Computation

This process is highly repetitive; fib is called on the same argument multiple times

(We will speed up this computation dramatically in a few weeks by remembering results)

Example: Counting Partitions

Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in increasing order.

count_partitions(6, 4)

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

1 + 1 + 1 + 3 = 6
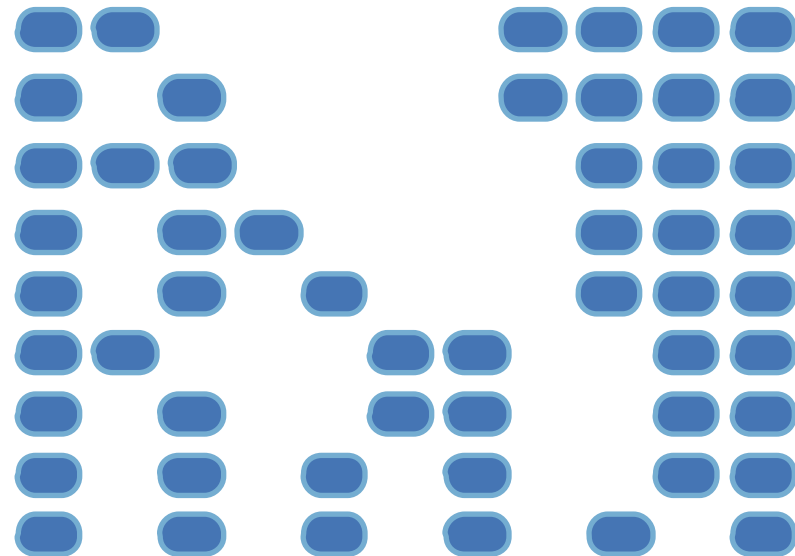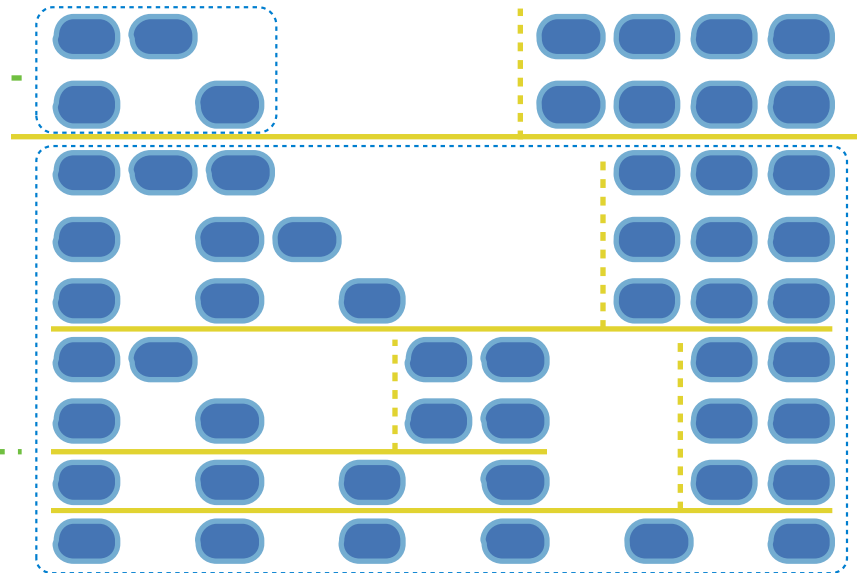
2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

1 + 1 + 1 + 1 + 1 + 1 = 6

Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in increasing order.

count_partitions(6, 4)

- Recursive decomposition: finding simpler instances of the problem.

- Explore two possibilities:

  - Use at least one 4

  - Don't use any 4

- Solve two simpler problems:

  - count_partitions(2, 4)

  - count_partitions(6, 3)

- Tree recursion often involves exploring different choices.

Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in increasing order.

- Recursive decomposition: finding simpler instances of the problem.

- Explore two possibilities:

  - Use at least one 4

  - Don't use any 4

- Solve two simpler problems:

  - count_partitions(2, 4)

  - count_partitions(6, 3)

- Tree recursion often involves exploring different choices.

```python
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
```

(Demo)

Interactive Diagram