Announcements

• Project 1 is due Thursday 9/18 @ 11:59pm; Early bonus point for submitting on Wednesday!
  ▪ Project Party in Stern Main Lounge (Stern Hall in Unit 4) 8pm–10pm on Wednesday 9/17
• Midterm 1 is on Monday 9/22 from 7pm to 9pm
  ▪ 2 review sessions on Saturday 9/20 3pm–4:30pm and 4:30pm–6pm in 1 Pimentel
  ▪ HKN review session moved to Sunday 9/21, time/location TBD
  ▪ Includes topics up to and including this lecture
  ▪ Closed book/note exam, except for one page of hand-written notes and a study guide
  ▪ Cannot attend? Fill out the conflict form by Wednesday 9/17 @ 5pm!
• Optional Hog strategy contest ends Wednesday 10/1 @ 11:59pm
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% of the time
• 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

Fall 2012 Winners
Chenyang Yuan
Joseph Hui

Fall 2013 Winners
Paul Bramsen
Sam Kumar & Kangsik Lee
Kevin Chen

Fall 2014 Winners
YOUR NAME COULD BE HERE... FOREVER!
Order of Recursive Calls
The Cascade Function

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

Program output:

```
123
12
1
12
```

(Demo)

- 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)

    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
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(grow, print, n // 10)
shrink = lambda n: f_then_g(print, shrink, n // 10)
```
Tree Recursion
Tree Recursion

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

\[
\begin{align*}
  n: & \quad 0, 1, 2, 3, 4, 5, 6, 7, 8, \quad \ldots, \quad 35 \\
  \text{fib}(n): & \quad 0, 1, 1, 2, 3, 5, 8, 13, 21, \quad \ldots, \quad 9,227,465 \\
\end{align*}
\]

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

A Tree-Recursive Process

The computational process of fib evolves into a tree structure.

(Demo)
Repetition in Tree-Recursive Computation

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

We can 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.

$$\text{count_partitions}(6, 4)$$

- 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:
  - $\text{count_partitions}(2, 4)$
  - $\text{count_partitions}(6, 3)$
- 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:
  - $\text{count_partitions}(2, 4)$
  - $\text{count_partitions}(6, 3)$
- Tree recursion often involves exploring different choices.

```
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)