**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
- All strategies must be deterministic, pure functions of the players' scores
- All winning entries will receive extra credit
- The real prizes: honor and glory

**Fall 2011 Winners**
- Kaylee Mann
- Yan Duan & Ziming Li
- Brian Prisk & Zhengao Qian
- Parker Schuh & Robert Chatham
- Chenyang Yuan
- Joseph Hui

**Fall 2012 Winners**
- Paul Brasen
- San Kumar & Kangsik Lee
- Kevin Chen

**Fall 2013 Winners**
- Alan Tong & Elaine Zhao
- Zhenyang Zhang
- Adam Robert Villaflo & Joony Gao
- Zichen Tai & Yile Li

**Spring 2013 Winners**
- Sinho Choi & Alexander Nguyen Tran
- Zhen Qin & Dian Chen
- Kaylee Mann
- Yan Duan & Ziming Li
- Brian Prisk & Zhengao Qian
- Parker Schuh & Robert Chatham
- Chenyang Yuan
- Joseph Hui

**Fall 2015 Winners**
- Paul Brasen
- San Kumar & Kangsik Lee
- Kevin Chen

**Fall 2016 Winners**
- Micah Carroll & Vasilis Oikonomou
- Matthew Wu
- Anthony Yeung and Alexander Dai

**Two Definitions of Cascade**

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

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

- 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

**Inverse Cascade**

Write a function that prints an inverse cascade:

```python
def inverse_cascade(n):
    if n:
        f(n)
        g(n)
```

```python
def inverse_cascade(n):
    grow = lambda n: f_then_g(grow, n)
    shrink = lambda n: f_then_g(shrink, n)
```
Tree Recursion

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

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

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 \texttt{fib} evolves into a tree structure.

Repetition in Tree-Recursive Computation

This process is highly repetitive; \texttt{fib} is called on the same argument multiple times.

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

\[
\begin{align*}
\text{count_partitions}(6, 4) &: 2 + 4 = 6 \\
&= 1 + 1 + 4 = 6 \\
&= 1 + 3 = 6 \\
&= 1 + 2 + 3 = 6 \\
&= 1 + 1 + 1 + 3 = 6 \\
&= 2 + 2 + 2 = 6 \\
&= 1 + 1 + 2 + 2 = 6 \\
&= 1 + 1 + 1 + 1 + 2 = 6 \\
&= 1 + 1 + 1 + 1 + 1 + 1 = 6
\end{align*}
\]