Announcements

- HW4 due Wednesday at 11:59pm

- Hog contest deadline next week
  - Completely optional, opportunity for extra credit
  - See website for details
The Fibonacci sequence is defined as

\[ \text{fib}(n) = \begin{cases} 
  0, & n = 0 \\
  1, & n = 1 \\
  \text{fib}(n - 1) + \text{fib}(n - 2), & n > 1 
\end{cases} \]

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

Example: [http://goo.gl/DZbRG](http://goo.gl/DZbRG)
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
We can use a higher-order function to see the order in which calls are made and complete

```python
def trace1(fn):
    """Return a function equivalent to fn that also prints trace output.""
    def traced(x):
        print('Calling', fn, '(', x, ')')
        res = fn(x)
        print('Got', res, 'from', fn, '(', x, ')')
        return res
    return traced

# Rebind the name fib to a traced version of fib
fib = trace1(fib)
```
Function Decorators

@trace1
```python
def triple(x):
    return 3 * x
```

is identical to
```python
def triple(x):
    return 3 * x
triple = trace1(triple)
```

Why not just use this?
The Recursive Leap of Faith

```python
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.
2. Treat `factorial(n-1)` as a functional abstraction.
3. Assume that `factorial(n-1)` is correct.
4. Verify that `factorial(n)` is correct, assuming that `factorial(n-1)` is correct.
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

```
smart  →  artsmay
```

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

```
smart  →  marts  →  artsm  →  artsmay
```

2 consonants to be moved
1 consonant to be moved
Base case
How many ways are there to change a dollar?

How many ways to change $0.11?

<table>
<thead>
<tr>
<th>Use a</th>
<th>Use a nickel</th>
<th>No dimes</th>
<th>No nickles</th>
</tr>
</thead>
<tbody>
<tr>
<td>dime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 1 1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ways to make 6 cents using no dimes

$1 = 100$ pennies
How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:

1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin
2. The number of ways to change $a$ using all but the first kind of coin

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>Use a nickel</th>
<th>No dimes</th>
<th>No nickles</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td></td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>5</td>
<td>5 1</td>
<td></td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>5</td>
<td>1 1 1 1 1 1 1</td>
<td></td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Ways to make 6 cents using no dimes
How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:

1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin
2. The number of ways to change $a$ using all but the first kind

```python
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```

- One way to make no amount
- Can’t make negative amount, or any amount with no coins
- Functional abstraction to get next coin