CS61A Lecture 11
Amir Kamil
UC Berkeley
February 15, 2013

Announcements
- HW4 due Wednesday at 11:59pm
- Hog contest deadline next week
  - Completely optional, opportunity for extra credit
  - See website for details

Fibonacci Sequence

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)
```

Two recursive calls!

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

```
# Fib(5)
<table>
<thead>
<tr>
<th>Fib(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib(3)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fib(2)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fib(1)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fib(0)</td>
</tr>
</tbody>
</table>
```

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
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

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

Counting Change

- $1 = 0.50 + 0.25 + 0.10 + 0.10 + 0.05$
- $1 = 1$ half dollar, 1 quarter, 2 dimes, 1 nickel
- $1 = 2$ quarters, 2 dimes, 30 pennies
- $1 = 100$ pennies

How many ways are there to change a dollar?

<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>1 2 5 10 25</td>
<td>1 5 10 25 50</td>
<td>1 10 25 50 100</td>
<td>1 10 25 50 100</td>
</tr>
<tr>
<td>Ways to make 6 cents using no dimes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Counting Change Recursively

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))
```

Functional abstraction to get next coin

Counting Change Recursively

How many ways are there to change $0.11$?

<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>1 2 5 10 25</td>
<td>1 5 10 25 50</td>
<td>1 10 25 50 100</td>
<td>1 10 25 50 100</td>
</tr>
<tr>
<td>Ways to make 6 cents using no dimes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>