CS61A Lecture 9
Amir Kamil
UC Berkeley
February 11, 2013

Announcements

- HW3 due Tuesday at 7pm
- Hog due today!
  - Hog contest due later; see announcement tonight
- Midterm Wednesday at 7pm
  - See course website for assigned locations, more info
- Midterm review in lab this week

Factorial

The factorial of a non-negative integer $n$ is

$$n! = \begin{cases} 
1, & n = 0 \text{ or } n = 1 \\
n \cdot (n-1) \cdot \cdots \cdot 1, & n > 1
\end{cases}$$

This is called a recurrence relation;
Factorial is defined in terms of itself
Can we write code to compute factorial using the same pattern?

Computing Factorial

We can compute factorial using the direct definition

$$n! = \begin{cases} 
1, & n = 0 \text{ or } n = 1 \\
n \cdot (n-1) \cdot \cdots \cdot 1, & n > 1
\end{cases}$$

```python
def factorial_iter(n):
    if n == 0 or n == 1:
        return 1
    total = 1
    while n > 1:
        total, n = total * n, n - 1
    return total
```

Computing Factorial

Can we compute it using the recurrence relation?

$$n! = \begin{cases} 
1, & n = 0 \text{ or } n = 1 \\
n \cdot (n-1)! & n > 1
\end{cases}$$

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

This is much shorter! But can a function call itself?
Let's see what happens!

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

Recursive functions have two important components:

1. **Base case(s)**, where the function directly computes an answer without calling itself
2. **Recursive case(s)**, where the function calls itself as part of the computation

```python
def factorial2(n):
    return factorial_helper(n, 1)

def factorial_helper(n, k):
    if k >= n:
        return k
    return k * factorial_helper(n, k + 1)
```

A function is recursive if the body calls the function itself, either directly or indirectly.

**Recursion Example: Heavy Box**

```python
def lift_box(box):
    if too_heavy(box):
        book = remove_book(box)
        lift_box(box)
        add_book(box, book)
    else:
        move_box(box)
```

**Recursion Example: Duplication**

```python
def duplicate(size):
    return (duplicate(0.6 * size) +
            duplicate(0.6 * size))
```

**Reversing the Order of Recursive Calls**

Some recursive computations may be done more easily by reversing the order of recursive calls.

A helper function helps us to do this.

\[
!n = \begin{cases} 
1, & n = 0 \text{ or } n = 1 \\
 n \cdot (n - 1)!, & n > 1 
\end{cases}
\]

```python
def factorial2(n):
    return factorial_helper(n, 1)

def factorial_helper(n, k):
    if k >= n:
        return k
    return k * factorial_helper(n, k + 1)
```
Here is how the reversed computation evolves

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

Reverse Environment Diagram

Fibonacci Sequence

The Fibonacci sequence is defined as

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

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

Fibonacci Sequence

Example: http://goo.gl/6zz0z

```
Compute 3!
Compute 2*3
Compute 3
```

Example: http://goo.gl/9UJxG

```
def fib_iter(n):
    fib_n, fib_n_1 = 1, 0
    for k in range(1, n):
        fib_n, fib_n_1 = fib_n_1 + fib_n, fib_n
    return fib_n
```

Example: http://goo.gl/WBU6

Tree recursion

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

This is called tree recursion

Example: http://goo.gl/9UJxG