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

$$n! = \begin{cases} 1, & n = 0 \text{ or } n = 1 \\ n \times (n - 1) \times \cdots \times 1, & n > 1 \end{cases}$$
The factorial of a non-negative integer $n$ is

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

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

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

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
Can we compute it using the recurrence relation?

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

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
1  def factorial(n):
2     if n == 0 or n == 1:
3         return 1
4     return n * factorial(n - 1)
5
6  factorial(4)
```

Example: [http://goo.gl/NjCKG](http://goo.gl/NjCKG)
A function is *recursive* if the body calls the function itself, either directly or indirectly.

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 factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)
```
Recursion Example: Heavy Box

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))
```

No base case!
Recursion Example: Dreaming

def dream(level):
    if level == 3:
        return inception()
    else:
        return dream(level + 1)
Some recursive computations may be done more easily by reversing the order of recursive calls.

A helper function helps us to do this.

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

```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

```python
1 def factorial2(n):
2     return factorial_helper(n, 1)
3 def factorial_helper(n, k):
4     if k >= n:
5         return k
6     return k * factorial_helper(n, k + 1)
8 factorial2(3)
```

Example: [http://goo.gl/6zz0z](http://goo.gl/6zz0z)
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_iter(n):
    if n == 0:
        return 0
    fib_n, fib_n_1 = 1, 0
    k = 1
    while k < n:
        fib_n, fib_n_1 = fib_n_1 + fib_n, fib_n
        k += 1
    return fib_n
```

Example: [http://goo.gl/9UJxG](http://goo.gl/9UJxG)
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