

61A Lecture 6

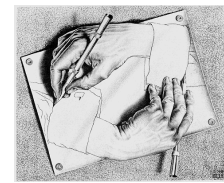
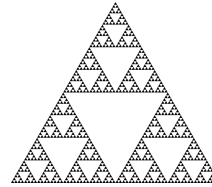
Announcements

Recursive Functions

Recursive Functions

Definition: A function is called recursive if the body of that function calls itself, either directly or indirectly

Implication: Executing the body of a recursive function may require applying that function

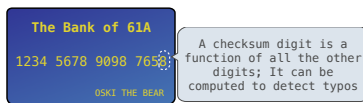


Drawing Hands, by M. C. Escher (Lithograph, 1948)

Digit Sums

$$2+0+1+5 = 8$$

- If a number a is divisible by 9, then $\text{sum_digits}(a)$ is also divisible by 9
- Useful for typo detection!



- Credit cards actually use the Luhn algorithm, which we'll implement after `digit_sum`

Sum Digits Without a While Statement

```
def split(n):
    """Split positive n into all but its last digit and its last digit."""
    return n // 10, n % 10

def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

The Anatomy of a Recursive Function

- The `def` statement header is similar to other functions
- Conditional statements check for **base cases**
- Base cases are evaluated **without recursive calls**
- Recursive cases are evaluated **with recursive calls**

```
def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

(Demo)

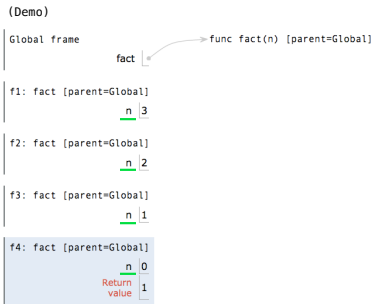
Recursion in Environment Diagrams

Recursion in Environment Diagrams

```

1 def fact(n):
2   if n == 0:
3     return 1
4   else:
5     return n * fact(n-1)
6
7 fact(3)

```



- The same function `fact` is called multiple times
- Different frames keep track of the different arguments in each call
- What `n` evaluates to depends upon the current environment
- Each call to `fact` solves a simpler problem than the last: smaller `n`

Interactive Diagram

Iteration vs Recursion

Iteration is a special case of recursion

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

Using while:

```

def fact_iter(n):
  total, k = 1, 1
  while k <= n:
    total, k = total*k, k+1
  return total

```

Using recursion:

```

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

```

Math: $n! = \prod_{k=1}^n k$

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Names: `n`, `total`, `k`, `fact_iter`

`n`, `fact`

Verifying Recursive Functions

The Recursive Leap of Faith

```

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

```

Is `fact` implemented correctly?

1. Verify the base case
2. Treat `fact` as a functional abstraction!
3. Assume that `fact(n-1)` is correct
4. Verify that `fact(n)` is correct

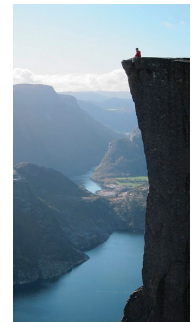


Photo by Kevin Lee, Preikestølen, Norway

Verifying Digit Sum

The `sum_digits` function computes the sum of positive `n` correctly because:

The sum of the digits of [redacted] (base case)

Assuming [redacted] (assume correct)

for all [redacted], (simpler case)

`sum_digits(n)` will [redacted] (conclusion)

```

def sum_digits(n):
  """Return the sum of the digits of positive integer n."""
  if n < 10:
    return n
  else:
    all_but_last, last = split(n)
    return sum_digits(all_but_last) + last

```

Mutual Recursion

The Luhn Algorithm

Used to verify credit card numbers

From Wikipedia: http://en.wikipedia.org/wiki/Luhn_algorithm

- **First:** From the rightmost digit, which is the check digit, moving left, double the value of every second digit; if product of this doubling operation is greater than 9 (e.g., $7 * 2 = 14$), then sum the digits of the products (e.g., $10: 1 + 0 = 1$, $14: 1 + 4 = 5$)
- **Second:** Take the sum of all the digits

1	3	8	7	4	3
2	3	1+6=7	7	8	3

= 30

The Luhn sum of a valid credit card number is a multiple of 10

(Demo)

Recursion and Iteration

Converting Recursion to Iteration

Can be tricky: Iteration is a special case of recursion.

Idea: Figure out what state must be maintained by the iterative function.

```
def sum_digits(n):  
    """Return the sum of the digits of positive integer n."""  
    if n < 10:  
        return n  
    else:  
        all_but_last, last = split(n)  
        return sum_digits(all_but_last) + last
```

What's left to sum

A partial sum

(Demo)

Converting Iteration to Recursion

More formulaic: Iteration is a special case of recursion.

Idea: The state of an iteration can be passed as arguments.

```
def sum_digits_iter(n):  
    digit_sum = 0  
    while n > 0:  
        n, last = split(n)  
        digit_sum = digit_sum + last  
    return digit_sum  
  
def sum_digits_rec(n, digit_sum):  
    if n == 0:  
        return digit_sum  
    else:  
        n, last = split(n)  
        return sum_digits_rec(n, digit_sum + last)
```

Updates via assignment become...

...arguments to a recursive call