

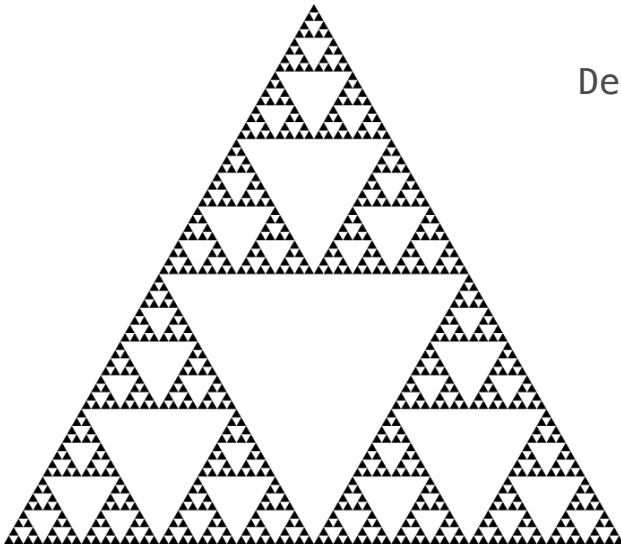
Announcements

Recursive Functions

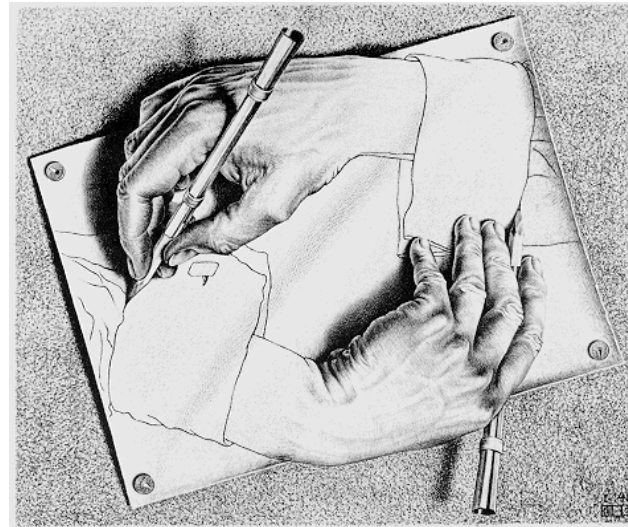
Recursive Functions

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



Demo

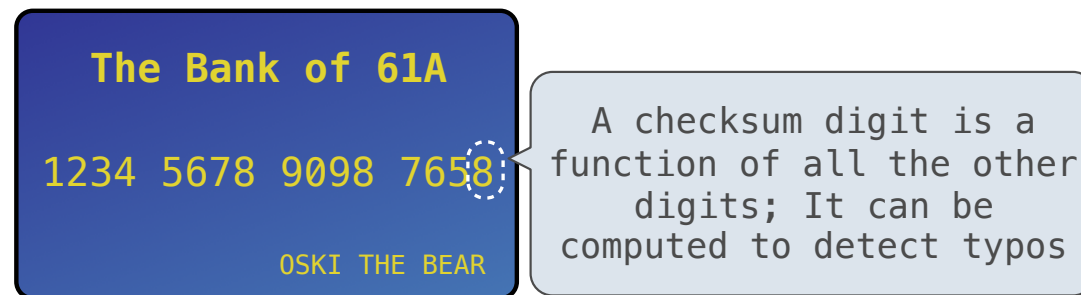


Drawing Hands, by M. C. Escher (lithograph, 1948)

Digit Sums

$$2+0+1+6 = 9$$

- If a number a is divisible by 9, then `sum_digits(a)` is also divisible by 9
- Useful for typo detection!



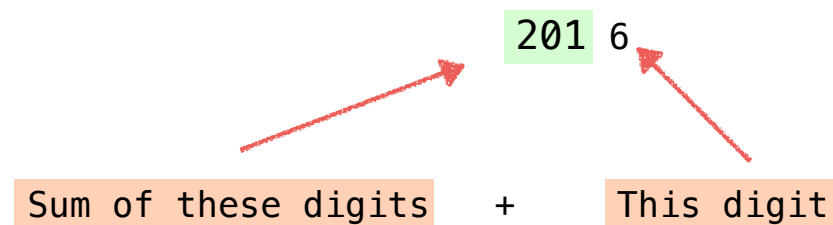
- Credit cards actually use the Luhn algorithm, which we'll implement after `sum_digits`

The Problem Within the Problem

The sum of the digits of 6 is 6.

Likewise for any one-digit (non-negative) number (i.e., < 10).

The sum of the digits of 2016 is



That is, we can break the problem of summing the digits of 2016 into a **smaller instance of the same problem**, plus some extra stuff.

We call this **recursion**

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**

(Demo)

Global frame

fact

func fact(n) [parent=Global]

f1: fact [parent=Global]

n | 3

f2: fact [parent=Global]

n | 2

f3: fact [parent=Global]

n | 1

f4: fact [parent=Global]

n | 0

Return
value | 1

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)
```

recursion simplifies things, esp for reader

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!
just think of what it is going to do
3. Assume that **fact(n-1)** is correct
4. Verify that **fact(n)** is correct

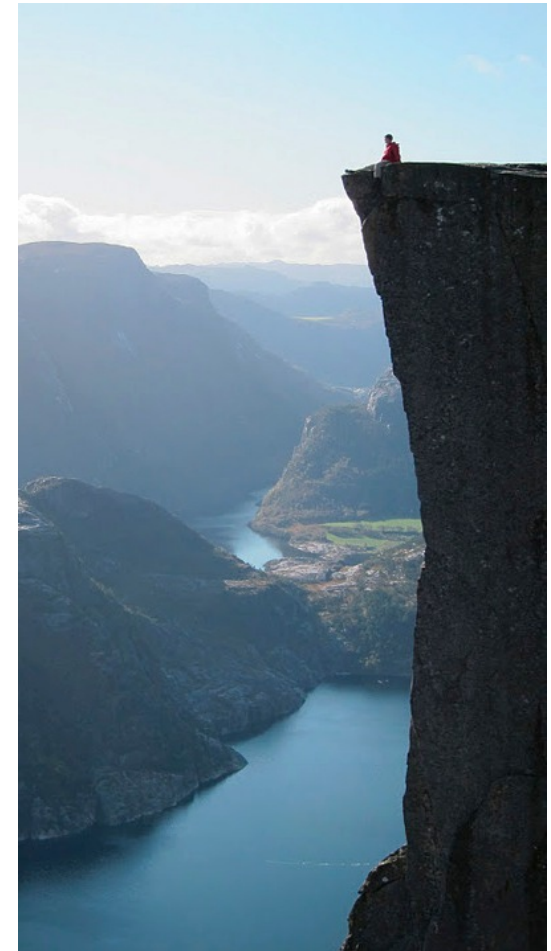


Photo by Kevin Lee, Preikestolen, Norway

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)

The Luhn Algorithm

```
Python
~/lec$ python3 -i ex.py
>>> luhn_sum(2)
2
>>> luhn_sum(32)
8
>>> luhn_sum(5105105105105100)
20
>>> █
```

you can have base cases for
either one or both of
the mutual recursion functions,
here both of them

```
1 def split(n):
2     return n // 10, n % 10
3
4 def sum_digits(n):
5     if n < 10:
6         return n
7     else:
8         all_but_last, last = split(n)
9         return sum_digits(all_but_last) + last
10
11 def luhn_sum(n):
12     if n < 10:
13         return n
14     else:
15         all_but_last, last = split(n)
16         return luhn_sum_double(all_but_last) + last
17
18 def luhn_sum_double(n):
19     all_but_last, last = split(n)
20     luhn_digit = sum_digits(2 * last)
21     if n < 10:
22         return luhn_digit
23     else:
24         return luhn_sum(all_but_last) + luhn_digit
```



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
```

Updates via assignment become...

```
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)
```

...arguments to a recursive call