Announcements

• Homework 2 due Monday 2/2 @ 11:59pm

• Project 1 due Thursday 2/5 @ 11:59pm
  ▪ Project party on Tuesday 2/3 5pm–6:30pm in 2050 VLSB
  ▪ Partner party on Wednesday 2/4 3pm–4pm in Wozniak Lounge, Soda Hall
  ▪ Earn 1 bonus point if you finish by Wednesday 2/4 @ 11:59pm
  ▪ Composition: Programs should be concise, well-named, understandable, and easy to follow

• Extra lecture 2 on Thursday 2/5 5pm–6:30pm in 2050 VLSB
  ▪ Hog strategies & church numerals

• Midterm 1 on Monday 2/9 7pm–9pm
  ▪ Conflict? Fill out the conflict form today! http://goo.gl/2P5fKq
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

If a number $a$ is divisible by 9, then $\text{sum_digits}(a)$ is also divisible by 9.

Useful for typo detection!

$2+0+1+5 = 8$

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

```python
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

• 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 which is the current environment.
• Each call to fact solves a simpler problem than the last: smaller n.

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

fact(3)
```
**Iteration vs Recursion**

**Iteration is a special case of recursion**

\[ 4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24 \]

Using while:

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

Using recursion:

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

**Math:**

\[ n! = \prod_{k=1}^{n} k \]

**Names:**

n, total, k, fact_iter

n, fact
Verifying Recursive Functions
The Recursive Leap of Faith

```python
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, assuming that `fact(n-1)` correct.
Verifying Digit Sum

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

The sum of the digits of \( n < 10 \) is \( n \).  \hspace{1cm} \text{(base case)}

Assuming \( \text{sum_digits}(k) \) correctly sums the digits of \( k \) for all \( k \) with fewer digits than \( n \), \hspace{1cm} \text{(abstraction)}

for all \( \text{sum_digits}(n) \) will be \( \text{sum_digits}(n//10) + \text{last} \) \hspace{1cm} \text{(simpler case)}

sum_digits(n) will be \( \text{sum_digits}(n//10) + \text{last} \).  \hspace{1cm} \text{(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 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).

- Take the sum of all the digits.

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1+6=7</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ = 30 \]

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

(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.

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

```python
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)
```