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.
Digit Sums

2+0+1+5 = 8

• If a number $a$ is divisible by 9, then $\text{sum} \_ \text{digits}(a)$ is also divisible by 9
• Useful for typo detection!

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

Each call to `fact` solves a simpler problem than the last: smaller `n`.

---

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

(Demo)

Global frame

```
func fact(n) [parent=Global]
```

```
f1: fact [parent=Global]
```

```
f2: fact [parent=Global]
```

```
f3: fact [parent=Global]
```

```
f4: fact [parent=Global]
```

Interactive Diagram
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

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

Names: 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
Mutual Recursion
The Luhn Algorithm

Used to verify credit card numbers


• **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., $7 \times 2 = 14$), 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

<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.

```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)
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

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