Recursion
Announcements
Self-Reference

(Demo)
Returning a Function Using Its Own Name

```python
def print_sums(n):
    print(n)
    def next_sum(k):
        return print_sums(n+k)
    return next_sum

print_sums(1)(3)(5)
```
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.
Sum Digits

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

Useful for typo detection!

\[2 + 0 + 2 + 1 = 5\]

Credit cards actually use the Luhn algorithm, which we'll implement after $\text{sum_digits}$
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 2022 is

\[
\begin{align*}
\text{Sum of these digits} & \quad + \quad \text{This digit} \\
202 & \quad 2
\end{align*}
\]

That is, we can break the problem of summing the digits of 2022 into a smaller instance of the same problem, plus some extra stuff.

We call this recursion.
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
```
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`
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:

Using while: \( n, \text{total}, k, \text{fact}_\text{iter} \)

Using recursion: \( n, \text{fact} \)
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 * 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

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

Idea: The state of an iteration are 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:
        n, last = split(n)
        return sum_digits_rec(n, digit_sum + last)
    else:
        return digit_sum
```

Updates via assignment become...

...arguments to a recursive call