

61A Lecture 7

Monday, September 15

Announcements

Announcements

- Homework 2 due Monday 9/15 at 11:59pm

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>
- Midterm 1 is next Monday 9/23 from 7pm to 9pm in various locations across campus

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>
- Midterm 1 is next Monday 9/23 from 7pm to 9pm in various locations across campus
 - Closed book, paper-based exam

2

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>
- Midterm 1 is next Monday 9/23 from 7pm to 9pm in various locations across campus
 - Closed book, paper-based exam
 - You may bring one hand-written page of notes that you created (front & back)

2

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>
- Midterm 1 is next Monday 9/23 from 7pm to 9pm in various locations across campus
 - Closed book, paper-based exam
 - You may bring one hand-written page of notes that you created (front & back)
- Review session on Saturday 9/20 3pm–6pm in 2050 VLSB

2

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>
- Midterm 1 is next Monday 9/23 from 7pm to 9pm in various locations across campus
 - Closed book, paper-based exam
 - You may bring one hand-written page of notes that you created (front & back)
- Review session on Saturday 9/20 3pm–6pm in 2050 VLSB
- Office hours on Friday & Monday will review various topics

2

Announcements

- Homework 2 due Monday 9/15 at 11:59pm
- Project 1 **deadline extended**, due Thursday 9/18 at 11:59pm
 - Extra credit point if you submit by Wednesday 9/17 at 11:59pm
- Project/homework party Monday 9/15: 3pm–4pm in Wozniak Lounge & 6pm–8pm in 2050 VLSB
 - These optional events appear on <http://cs61a.org/weekly.html>
- Midterm 1 is next Monday 9/23 from 7pm to 9pm in various locations across campus
 - Closed book, paper-based exam
 - You may bring one hand-written page of notes that you created (front & back)
- Review session on Saturday 9/20 3pm–6pm in 2050 VLSB
- Office hours on Friday & Monday will review various topics
- No lab or office hours on Tuesday 9/23 and Wednesday 9/24 (staff will be grading exams)

2

Recursive Functions

Recursive Functions

Recursive Functions

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

4

4

Recursive Functions

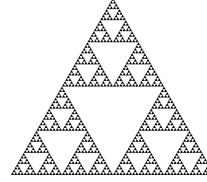
Definition: A function is called recursive if the body of that function calls itself, either directly or indirectly.

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

Recursive Functions

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

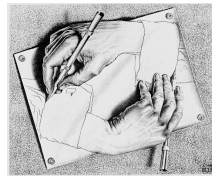
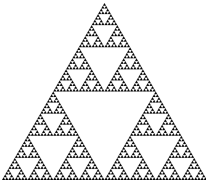
Implication: Executing the body of a recursive function may require applying that function.



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

$$2+0+1+4 = 7$$

Digit Sums

$$2+0+1+4 = 7$$

·If a number a is divisible by 9, then `sum_digits(a)` is also divisible by 9.

Digit Sums

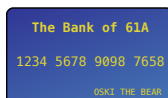
$$2+0+1+4 = 7$$

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

Digit Sums

$$2+0+1+4 = 7$$

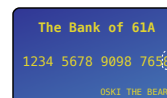
·If a number a is divisible by 9, then `sum_digits(a)` is also divisible by 9.
·Useful for typo detection!



Digit Sums

$$2+0+1+4 = 7$$

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

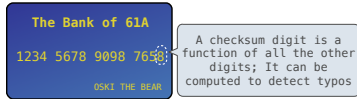


A checksum digit is a function of all the other digits; It can be computed to detect typos

Digit Sums

$$2+0+1+4 = 7$$

- 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

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
```

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

```
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

```
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

```
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

```
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`

```
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
```

7

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

```
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
```

7

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

```
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
```

7

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
```

7

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
```

7

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)

7

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)
```

Interactive Diagram

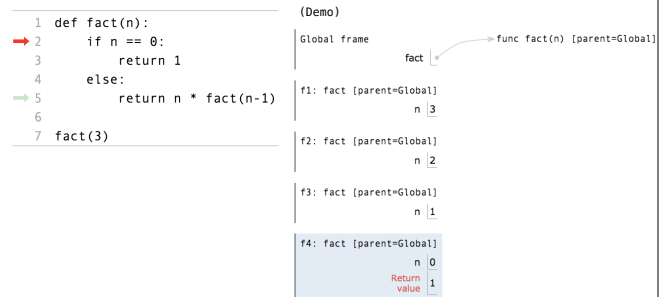
7

Recursion in Environment Diagrams

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

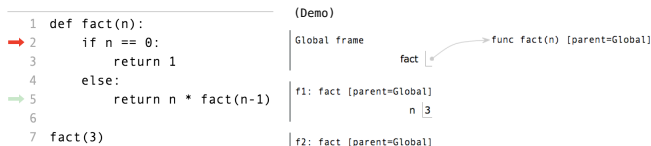
Interactive Diagram

Recursion in Environment Diagrams



Interactive Diagram

Recursion in Environment Diagrams



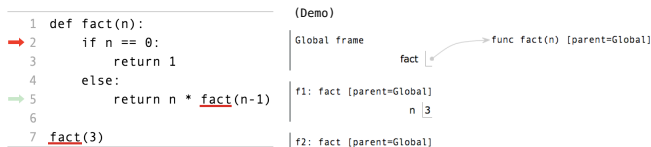
Interactive Diagram

Recursion in Environment Diagrams



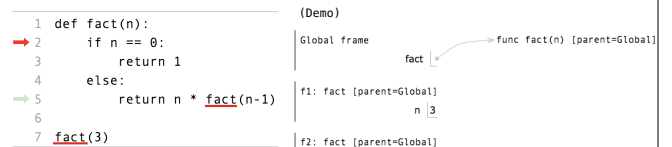
Interactive Diagram

Recursion in Environment Diagrams



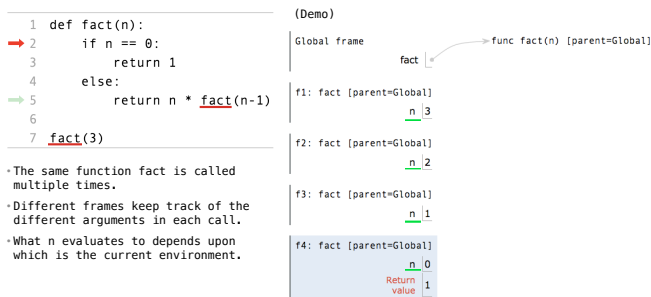
Interactive Diagram

Recursion in Environment Diagrams



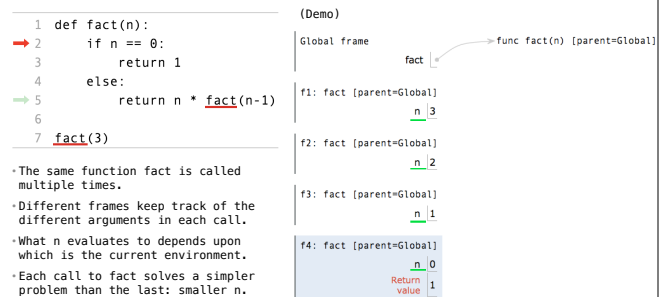
Interactive Diagram

Recursion in Environment Diagrams



Interactive Diagram

Recursion in Environment Diagrams



Interactive Diagram

Iteration vs Recursion

Iteration vs Recursion

Iteration is a special case of recursion

Iteration is a special case of recursion

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

Iteration is a special case of recursion

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

Using while:

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
```

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:

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)
```

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:

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

10

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

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

10

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:

10

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

10

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

10

Verifying Recursive Functions

The Recursive Leap of Faith

The Recursive Leap of Faith

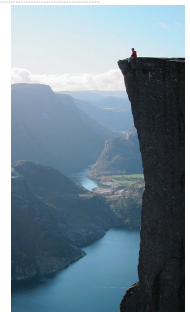


Photo by Kevin Lee, Preikestolen, Norway

10

10

The Recursive Leap of Faith

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



Photo by Kevin Lee, Preikestolen, Norway

12

The Recursive Leap of Faith

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

Is fact implemented correctly?

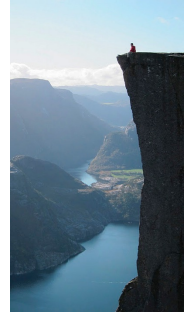


Photo by Kevin Lee, Preikestolen, Norway

12

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.



Photo by Kevin Lee, Preikestolen, Norway

12

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!

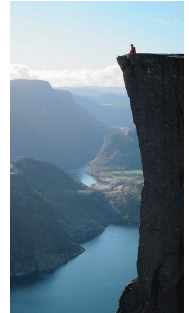


Photo by Kevin Lee, Preikestolen, Norway

12

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.

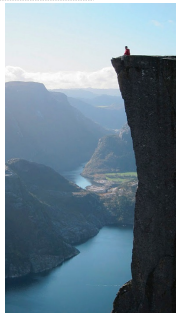


Photo by Kevin Lee, Preikestolen, Norway

12

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, assuming that fact(n-1) correct.

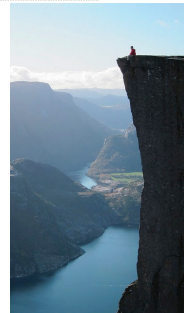


Photo by Kevin Lee, Preikestolen, Norway

12

Mutual Recursion

The Luhn Algorithm

14

The Luhn Algorithm

Used to verify credit card numbers

14

The Luhn Algorithm

Used to verify credit card numbers

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

14

The Luhn Algorithm

Used to verify credit card numbers

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

- 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: $1 + 4 = 5$).

14

The Luhn Algorithm

Used to verify credit card numbers

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

- 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.

14

The Luhn Algorithm

Used to verify credit card numbers

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

- 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.

1	3	8	7	4	3
---	---	---	---	---	---

14

The Luhn Algorithm

Used to verify credit card numbers

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

- 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.

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

14

The Luhn Algorithm

Used to verify credit card numbers

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

- 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.

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

= 30

14

The Luhn Algorithm

Used to verify credit card numbers

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

- 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.

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

= 30

14

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

The Luhn Algorithm

Used to verify credit card numbers

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

- 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.

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

Converting Recursion to Iteration

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

Converting Recursion to Iteration

Converting Recursion to Iteration

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

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

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

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
```

Converting Recursion to Iteration

Converting Recursion to Iteration

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

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

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

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

```
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

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)

16

Converting Iteration to Recursion

Converting Iteration to Recursion

More formulaic: Iteration is a special case of recursion.

Converting Iteration to Recursion

More formulaic: Iteration is a special case of recursion.

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

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
```

17

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)
```

17

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)
```

17

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

17