

# 61A Lecture 11

---

Friday, September 21

## Midterm 1 Recap

The exam was more difficult than the Fall 2011 Midterm 1

Typically, more than 75% of students receive A's & B's in 61A

**Problem 4(c):** **through** doesn't rhyme with **cough** , and **20** (*twenty*) doesn't rhyme with **10** (*ten*)

*Sight rhyme:* A pair of words that don't rhyme, but look like they should

*twelve*    *twenty-two*

**1X**    vs    **WX**

```
if first_tens(p)==1:
    return second_tens(p)!=1
else:
    return second_tens(p)==1
```

*twenty-two*    *twelve*

**WX**    vs    **1X**

*zero*    *twenty* ✓  
*twenty*    *zero* ✓

**X0**    vs    **Y0**

```
if first_tens(p)==0:
    return second_tens(p)!=0
else:
    return second_tens(p)==0
```

*"You may not use boolean operator or"*

Demo

# Mapping a Function over a Sequence

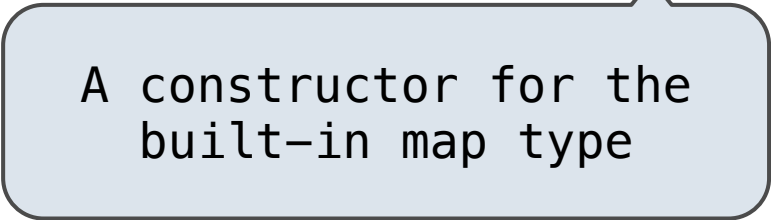
---

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))  
(1, 2, 3, 4, 5)
```

The returned value of **map** is an iterable map object



A constructor for the  
built-in map type

The returned value of **filter** is an iterable filter object

Demo

## Accumulation and Iterable Values

---

Iterable objects give access to some elements in order.

*However, you may only be able to access the elements once!*

Many built-in functions take iterable objects as argument.

<code>tuple</code>	Return a tuple containing the elements
<code>sum</code>	Return the sum of the elements
<code>min</code>	Return the minimum of the elements
<code>max</code>	Return the maximum of the elements

For statements also operate on iterable values.

## Reducing a Sequence

---

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul  
  
>>> from functools import reduce  
  
>>> reduce(mul, (1, 2, 3, 4, 5))  
120
```

First argument:  
A two-argument  
function

Second argument:  
an iterable object

Like accumulate from Homework 2, but with iterable objects

# Generator Expressions

---

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

- Evaluates to an iterable object.
- `<iter exp>` is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed.

Short version: `(<map exp> for <name> in <iter exp>)`

Precise evaluation rule introduced in Chapter 4.

Demo

# Python Lists

---

`['Demo']`

<http://docs.python.org/py3k/library/stdtypes.html#mutable-sequence-types>

---

# List Comprehensions

---

```
[<map exp> for <name> in <iter exp> if <filter exp>]
```

Short version: [`<map exp> for <name> in <iter exp>`]

Unlike generator expressions, the map expression is evaluated when the list comprehension is evaluated.

```
>>> suits = ['heart', 'diamond', 'spade', 'club']
```

```
>>> from unicodedata import lookup
```

```
>>> [lookup('WHITE ' + s.upper() + ' SUIT') for s in suits]
[ '♥', '♦', '♠', '♣' ]
```



# Dictionaries

---

```
{ 'Dem': 0 }
```

## Limitations on Dictionaries

---

Dictionaries are **unordered** collections of key-value pairs.

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** an object of a **mutable built-in** type.
- Two **keys cannot be equal**. There can be at most one value for a given key.

This first restriction is tied to Python's underlying implementation of dictionaries.

The second restriction is an intentional consequence of the dictionary abstraction.