61A Extra Lecture 5
Announcements
Data Representations
def box(contents):
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
    return get, put

get, put = box('Hello')
before = get()
put('Goodbye')
after = get()
def pair(x, y):
    def dispatch(m):
        if m == 'first':
            return x
        elif m == 'second':
            return y
    return dispatch

This function represents the pair (x, y)

Constructor is a higher-order function

>>> p = pair(3, pair(4, 5))
>>> p('first')
3
>>> p('second')('first')
4
>>> p('second')('second')
5

(Demo)
Linked Lists (Sneak Preview)

• An empty list is called "nil" and represented as None

• A non-empty list is represented as a pair
  • The first element of the pair is the first element of the list
  • The second element of the pair is the rest of the list

```
nil = None
def list_len(s):
    if s is nil:
        return 0
    else:
        return 1 + list_len(s('second'))

def append(s, x):
    if s is nil:
        return pair(x, nil)
    else:
        first, second = s('first'), s('second')
        return pair(first, append(second, x))
```

(Demo)
An Inefficient Dictionary Implementation

• A list of key-value pairs can be used to implement dictionary behavior

```python
>>> d = dict_dispatch()
>>> d('set')('I', 1)
>>> d('set')('V', 5)
>>> d('set')('X', 10)
```

(Demo)
Dispatch Dictionaries
Dispatch Dictionaries

Enumerating different messages in a conditional statement isn't very convenient:

- Equality tests are repetitive
- We can't add new messages without re-writing the dispatch function

A dispatch dictionary has messages as keys and functions (or data objects) as values

Dictionaries handle the message look-up logic; we can concentrate on implementing behavior

```python
def box_dispatch(contents):
    def dispatch(m):
        if m == 'contents':
            return contents
        if m == 'put':
            def put(value):
                contents = value
            return put
    return dispatch

def box_dict(contents):
    def put(value):
        d['contents'] = value
    d = {'contents': contents, 'put': put}
    return d

(Demo)
```
Constraint Networks
Solving for Variables in an Equation

\[ a + b = c \]
\[ a = c - b \]
\[ b = c - a \]

\[ p \times v = n \times k \times t \]
\[ 9 \times c = 5 \times (f - 32) \]

Algebraic equations are \textit{declarative}: They describe a relation among different quantities.

Python functions are \textit{procedural}: They describe how to compute a result from a set of input arguments.

Constraint programming:
- We define the relationship between quantities
- We provide values for the "known" quantities
- The system computes values for the "unknown" quantities

\textbf{Challenge}: We want a general means of combination.
A Constraint Network for Temperature Conversion

Combination idea: All intermediate quantities have values too.

\[
9 \times \text{celsius} = 5 \times (\text{fahrenheit} - 32)
\]

This quantity relates directly to \text{celsius}

This quantity relates directly to \text{fahrenheit}

Both sides of the equation are equal: they must be the same quantity