Data Representations

### Defining Functions with Shared Local State

```python
def box(contents):
def get():
    return contents

def put(value):
    nonlocal contents
    contents = value

get, put = box('Hello')
before = get()
put('Goodbye')
after = get()
```

### Pairs Implemented as Functions

```python
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

```python
def pair(x, y):
def dispatch(m):
    if m == 'first':
        return x
    elif m == 'second':
        return y
    return dispatch

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

### 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

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

### An Inefficient Dictionary Implementation

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

```python
def dict_dispatch():
    >>> d = dict_dispatch()
    >>> d['I'] = 1
    >>> d['V'] = 5
    >>> d('set')('X', 10)
    >>> print(d)
    {'I': 1, 'V': 5, 'X': 10}
```

### Dispatch Dictionaries

---

Interactive Diagram

---

Functions with Shared Local State

---

Announcements
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):
                nonlocal contents
                contents = value
            return put
        return dispatch

def box_dict(contents):
    def put(value):
        d['contents'] = contents
        d['put'] = put
        return d
```

Constraint Networks

Solving for Variables in an Equation

Algebraic equations are declarative: They describe a relation among different quantities.

Python functions are 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

Challenge: We want a general means of combination.

A Constraint Network for Temperature Conversion

Combination idea: All intermediate quantities have values too.

- Both sides of the equation are equal: they must be the same quantity.
- This quantity relates directly to celsius.
- This quantity relates directly to fahrenheit.

(Demo)