Testing for Identity

Implementing Dice

Random numbers are useful for experimentation. They also appear in lots of algorithms, e.g.,
- Primality tests
- Machine learning techniques

```python
def make_dice(sides=6):
    seed = 1
    multiplier = pow(7, 5) - 1
    big_prime = pow(2, 31) - 1

def dice():
    nonlocal seed
    seed = (multiplier * seed) % big_prime
    return (sides * seed) // big_prime + 1
```


Implementing a Mutable Container Object

Dispatch Functions

A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair."""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```

Message Passing

An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other
- What is your fourth element?
- Change your third element to this new value. (please?)

Encapsulates the behavior of all operations on a piece of data

Important historical role: The message passing approach strongly influenced object-oriented programming (next lecture)
A Mutable Container That Uses Message Passing

```python
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents
        if message == 'put':
            contents = value
        return dispatch
    return dispatch
```

Implementing Dictionaries

```python
def dictionary():
    """Return a functional implementation of a dictionary."""
    records = []
    def getitem(key):
        for k, v in records:
            if k == key:
                return v
    def setitem(key, value):
        for item in records:
            if item[0] == key:
                item[1] = value
        records.append([key, value])
    def dispatch(message, key=None, value=None):
        if message == 'getitem':
            return getitem(key)
        elif message == 'setitem':
            setitem(key, value)
        elif message == 'keys':
            return tuple(k for k, _ in records)
        elif message == 'values':
            return tuple(v for _, v in records)
    return dispatch
```

Example: Constraint Programming

```plaintext
a + b = c
a = c - b
b = c - a
9 * c = 5 * (f - 32)
```

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.

Implementing Mutable Recursive Lists

```python
def mutable_rlist():
    contents = empty_rlist
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'len':
            return len_rlist(contents)
        elif message == 'getitem':
            return getitem_rlist(contents, value)
        elif message == 'push_first':
            contents = make_rlist(value, contents)
        elif message == 'pop_first':
            f = first(contents)
            contents = rest(contents)
            return f
        elif message == 'str':
            return str(contents)
        return dispatch
```

Dispatch Dictionaries

Enumerating different messages in a conditional statement isn't very convenient:
- Equality tests are repetitive
- We can't add new messages without writing new code

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

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

In Javascript, all objects are just dictionaries

A Constraint Network for Temperature Conversion

Combination idea: All intermediate quantities have values too.

Combination idea: All intermediate quantities have values too.

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