61A Lecture 9
Data Abstraction
Data Abstraction

• Compound values combine other values together
  ▪ A date: a year, a month, and a day
  ▪ A geographic position: latitude and longitude

• Data abstraction lets us manipulate compound values as units

• Isolate two parts of any program that uses data:
  ▪ How data are represented (as parts)
  ▪ How data are manipulated (as units)

• Data abstraction: A methodology by which functions enforce an abstraction barrier between \textit{representation} and \textit{use}
Rational Numbers

Exact representation of fractions

A pair of integers

As soon as division occurs, the exact representation may be lost! (Demo)

Assume we can compose and decompose rational numbers:

- `rational(n, d)` returns a rational number \( x \)
- `numer(x)` returns the numerator of \( x \)
- `denom(x)` returns the denominator of \( x \)
Rational Number Arithmetic

\[
\frac{3}{2} \times \frac{3}{5} = \frac{9}{10}
\]

\[
\frac{3}{2} + \frac{3}{5} = \frac{21}{10}
\]

Example

General Form

\[
\frac{nx}{dx} \times \frac{ny}{dy} = \frac{nx \times ny}{dx \times dy}
\]

\[
\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx \times dy + ny \times dx}{dx \times dy}
\]
Rational Number Arithmetic Implementation

```python
def mul_rational(x, y):
    return rational(numer(x) * numer(y), denom(x) * denom(y))

def add_rational(x, y):
    nx, dx = numer(x), denom(x)
    ny, dy = numer(y), denom(y)
    return rational(nx * dy + ny * dx, dx * dy)

def print_rational(x):
    print(numer(x), '/', denom(x))

def rationals_are_equal(x, y):
    return numer(x) * denom(y) == numer(y) * denom(x)
```

- `rational(n, d)` returns a rational number \( \frac{n}{d} \)
- `numer(x)` returns the numerator of \( x \)
- `denom(x)` returns the denominator of \( x \)

These functions implement an abstract representation for rational numbers.
Pairs
Representing Pairs Using Lists

```python
>>> pair = [1, 2]
>>> pair
[1, 2]

>>> x, y = pair
>>> x
1
>>> y
2

>>> pair[0]
1
>>> pair[1]
2

>>> from operator import getitem
>>> getitem(pair, 0)
1
>>> getitem(pair, 1)
2
```

A list literal: Comma-separated expressions in brackets

"Unpacking" a list

Element selection using the selection operator

Element selection function

More lists next lecture
Representing Rational Numbers

```python
def rational(n, d):
    """A representation of the rational number N/D."""
    return [n, d]

def numer(x):
    """Return the numerator of rational number X."""
    return x[0]

def denom(x):
    """Return the denominator of rational number X."""
    return x[1]
```

(Demo)
A Problem of Specification

Our specification at the moment is ambiguous:

- “Numerator” refers to a particular way of writing a certain rational.
  - For example, what is the numerator of 6/8?
    - Could say it is 6, but 6/8 = 3/4, so why not 3?

Let’s be more precise:

```python
def numer(x):
    """Return the numerator of rational number X in lowest terms and having the same sign as X.""

def denom(x):
    """Return the denominator of rational number X in lowest terms and positive.""
```
Reducing to Lowest Terms

Example:

\[
\frac{3}{2} \times \frac{5}{3} = \frac{5}{2} \quad \frac{2}{5} + \frac{1}{10} = \frac{1}{2}
\]

\[
\frac{15}{6} \times \frac{1/3}{1/3} = \frac{5}{2} \quad \frac{25}{50} \times \frac{1/25}{1/25} = \frac{1}{2}
\]

from fractions import gcd  

```python
def rational(n, d):
    """A representation of the rational number N/D."""
    g = gcd(n, d)  # Always has the sign of d
    return [n//g, d//g]
```

(Demo)
Abstraction Barriers
### Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat rationals as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use rational numbers to perform computation</td>
<td>whole data values</td>
<td><code>add_rational, mul_rational, rationals_are_equal, print_rational</code></td>
</tr>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</td>
<td><code>rational, numer, denom</code></td>
</tr>
<tr>
<td>Implement selectors and constructor for rationals</td>
<td>two-element lists</td>
<td>list literals and element selection</td>
</tr>
</tbody>
</table>

*Implementation of lists*
Violating Abstraction Barriers

add_rational( [1, 2], [1, 4] )

def divide_rational(x, y):
    return [ x[0] * y[1], x[1] * y[0] ]

Does not use constructors
Twice!
No selectors!
And no constructor!
Data Representations
What is Data?

- We need to guarantee that constructor and selector functions work together to specify the right behavior.

- Behavior condition: If we construct rational number \( x \) from numerator \( n \) and denominator \( d \), then \( \text{numerator}(x)/\text{denominator}(x) \) must equal \( n/d \).

- Data abstraction uses selectors and constructors to define behavior.

- If behavior conditions are met, then the representation is valid.

You can recognize an abstract data representation by its behavior.

(Demo)
def rational(n, d):
    def select(name):
        if name == 'n':
            return n
        elif name == 'd':
            return d
    return select

def numer(x):
    return x('n')

def denom(x):
    return x('d')

This function represents a rational number

Constructor is a higher-order function

Selector calls x

Interactive Diagram