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 representation and use
Rational Numbers

- **numerator**
- **denominator**

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

Representing Rational Numbers

- rational(n, d) returns a rational number x
- numer(x) returns the numerator of x
- denom(x) returns the denominator of x

These functions implement an abstract representation for rational numbers

Rational Number Arithmetic Implementation

```python
def mul_rational(x, y):
    nx, dx = numer(x), denom(x)
    ny, dy = numer(y), denom(y)
    return rational(nx*ny, dx*dy)

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)
```
Representing Pairs Using Lists

```python
>>> pair = [1, 2]
>>> pair
[1, 2]
>>> x, y = pair
1
>>> y
2
>>> pair[0]
1
>>> pair[1]
2
```

A list literal: Comma-separated expressions in brackets

"Unpacking" a list

Element selection using the selection operator

Representing Rational Numbers

```python
def rational(n, d):
    """Construct a rational number that represents 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]
```

```
Defining a rational number.

>>> from math import gcd

>>> def rational(n, d):
...     """Construct a rational that represents n/d in lowest terms.""
...     g = gcd(n, d)
...     return [n/g, d/g]
...
>>> rational(15, 6)
[5, 2]
>>> rational(25, 10)
[1/2]
```

Reducing to Lowest Terms

Example:

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

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

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>add_rational, mul_rational, rations_are_equal, print_rational</td>
</tr>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</td>
<td>rational, numer, denom</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] ]
```

Data Representations

What are 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 numer(x)/denom(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)
This function represents a rational number.

Constructor is a higher-order function.

Selector calls x.