Data Abstraction
Announcements
Data Abstraction
Data Abstraction
Data Abstraction

• Compound values combine other values together
Data Abstraction

• Compound values combine other values together
  • A date: a year, a month, and a day
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

• 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
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:
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)
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

- 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
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
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
Rational Numbers

\[
\text{numerator} \quad \frac{\text{denominator}}{}
\]
Rational Numbers

\[
\frac{\text{numerator}}{\text{denominator}}
\]

Exact representation of fractions
Rational Numbers

\[
\frac{\text{numerator}}{\text{denominator}}
\]

Exact representation of fractions

A pair of integers
Rational Numbers

\[
\frac{\text{numerator}}{\text{denominator}}
\]

Exact representation of fractions

A pair of integers

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

\[
\frac{\text{numerator}}{\text{denominator}}
\]

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 Numbers

\[
\begin{array}{c}
\text{numerator} \\
\hline
\text{denominator}
\end{array}
\]

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
Rational Numbers

\[
\begin{array}{c}
\text{numerator} \\
\hline \\
\text{denominator}
\end{array}
\]

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:

- \text{rational}(n, d) \text{ returns a rational number } x
- \text{numer}(x) \text{ returns the numerator of } x
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 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 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:

Constructor
```
\text{rational}(n, d) \text{ returns a rational number } x
```

Selectors

\begin{itemize}
  \item \text{numer}(x) \text{ returns the numerator of } x
  \item \text{denom}(x) \text{ returns the denominator of } x
\end{itemize}
Rational Number Arithmetic

Example

General Form
Rational Number Arithmetic

\[
\frac{3}{2} \times \frac{3}{5}
\]
Rational Number Arithmetic

Example

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

General Form
Rational Number Arithmetic

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

Example

General Form

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

Example

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

General Form

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

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

\[
\frac{3}{2} + \frac{3}{5}
\]

Example

General Form

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

Example

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

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

General Form

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

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

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

\[
\begin{align*}
\frac{3}{2} \times \frac{3}{5} & = \frac{9}{10} \\
\frac{3}{2} + \frac{3}{5} & = \frac{21}{10}
\end{align*}
\]

Example

\[
\begin{align*}
\frac{nx}{dx} \times \frac{ny}{dy} & = \frac{nx \times ny}{dx \times dy} \\
\frac{nx}{dx} + \frac{ny}{dy} & = \frac{nx \times dy + ny \times dx}{dx \times dy}
\end{align*}
\]

General Form
Rational Number Arithmetic Implementation

- 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 Implementation

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

- `rational(n, d)` returns a rational number `x`
- `numer(x)` returns the numerator of `x`
- `denom(x)` returns the denominator of `x`
def mul_rational(x, y):
    return rational(numer(x) * numer(y),
                    denom(x) * denom(y))

• 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 Implementation

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

- 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 Implementation

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

- `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

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

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

- `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.

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

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

- `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):
    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.
Representing Rational Numbers
def rational(n, d):
    """Construct a rational number that represents N/D."""
    return [n, d]
Representing Rational Numbers

def rational(n, d):
    """Construct a rational number that represents N/D."""
    return [n, d]

Construct a list
Representing Rational Numbers

def rational(n, d):
    """Construct a rational number that represents N/D."""
    return [n, d]

    Construct a list

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

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]
Representing Rational Numbers

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]
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]
```

(Demo)
Reducing to Lowest Terms

Example:
Reducing to Lowest Terms

Example:

\[
\frac{3}{2} \times \frac{5}{3}
\]
Reducing to Lowest Terms

Example:

\[
\frac{3}{2} \times \frac{5}{3} = \frac{5}{2}
\]
Reducing to Lowest Terms

Example:

\[
\frac{3}{2} * \frac{5}{3} = \frac{5}{2}
\]

\[
\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2}
\]
Reducing to Lowest Terms

Example:

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

\[
\frac{15}{6} \times \frac{1/3}{1/3} = \frac{5}{2}
\]
Reducing to Lowest Terms

Example:

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

\[
\frac{15}{6} \times \frac{1/3}{1/3} = \frac{5}{2}
\]
Reducing to Lowest Terms

Example:

\[
\frac{3}{2} \times \frac{5}{3} = \frac{5}{2} \\
\frac{2}{5} + \frac{1}{10} = \frac{1}{2}
\]
Reducing to Lowest Terms

Example:

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

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

from math import gcd
Reducing to Lowest Terms

Example:

\[
\begin{array}{ccc}
\frac{3}{2} & \times & \frac{5}{3} & = & \frac{5}{2} \\
\frac{2}{5} & + & \frac{1}{10} & = & \frac{1}{2}
\end{array}
\]

\[
\begin{array}{ccc}
\frac{15}{6} & \times & \frac{1}{3} & = & \frac{5}{2} \\
\frac{25}{50} & \times & \frac{1}{25} & = & \frac{1}{2}
\end{array}
\]

```python
from math import gcd

def rational(n, d):
```
from math import gcd

def rational(n, d):
    """Construct a rational that represents n/d in lowest terms."""

Reducing to Lowest Terms

Example:

\[
\begin{align*}
\frac{3}{2} \times \frac{5}{3} &= \frac{5}{2} \\
\frac{2}{5} + \frac{1}{10} &= \frac{1}{2}
\end{align*}
\]

\[
\begin{align*}
\frac{15}{6} \times \frac{1/3}{1/3} &= \frac{5}{2} \\
\frac{25}{50} \times \frac{1/25}{1/25} &= \frac{1}{2}
\end{align*}
\]
Reducing to Lowest Terms

Example:

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

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

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

\[
\frac{25}{50} \times \frac{1/25}{1/25} = \frac{1}{2}
\]

```python
from math import gcd

def rational(n, d):
    """Construct a rational that represents n/d in lowest terms."""
    g = gcd(n, d)
```
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]

Reducing to Lowest Terms

Example:

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

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

Example:

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

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

```python
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]
```
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]
Abstraction Barriers
Abstraction Barriers
<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat rationals as...</th>
<th>Using...</th>
</tr>
</thead>
</table>

### 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</td>
<td>to perform computation</td>
<td></td>
</tr>
</tbody>
</table>
### 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</td>
<td>whole data values</td>
<td></td>
</tr>
<tr>
<td>to perform computation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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</td>
<td>whole data values</td>
<td>add_rational, mul_rational</td>
</tr>
<tr>
<td>to perform computation</td>
<td></td>
<td>rationals_are_equal, print_rational</td>
</tr>
</tbody>
</table>
### 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></td>
<td></td>
</tr>
</tbody>
</table>

Use rational numbers to perform computation.

Whole data values:
- add_rational
- mul_rational
- rations_are_equal
- print_rational
### 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</td>
</tr>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</td>
<td>rationals_are_equal, print_rational</td>
</tr>
</tbody>
</table>
# 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</td>
</tr>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</td>
<td>rational, numer, denom</td>
</tr>
</tbody>
</table>
## 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>
</tbody>
</table>
## 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></td>
<td></td>
</tr>
</tbody>
</table>
### 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</code>, <code>mul_rational</code>, <code>rations_are_equal</code>, <code>print_rational</code></td>
</tr>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</td>
<td><code>rational</code>, <code>numer</code>, <code>denom</code></td>
</tr>
<tr>
<td>Implement selectors and constructor for rationals</td>
<td>two-element lists</td>
<td></td>
</tr>
</tbody>
</table>
### 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 rations_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><code>list literals and element selection</code></td>
</tr>
</tbody>
</table>
### 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</code>, <code>mul_rational</code>, <code>rationals_are_equal</code>, <code>print_rational</code></td>
</tr>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</td>
<td><code>rational</code>, <code>numer</code>, <code>denom</code></td>
</tr>
<tr>
<td>Implement selectors and constructor for rationals</td>
<td>two-element lists</td>
<td><code>list literals</code> and <code>element selection</code></td>
</tr>
</tbody>
</table>
### 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, rationals_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*
## Abstraction Barriers

### Parts of the program that... | Treat rationals as... | Using...
---|---|---
Use rational numbers to perform computation | whole data values | add_rational, mul_rational, rations_are_equal, print_rational
Create rationals or implement rational operations | numerators and denominators | rational, numer, denom
Implement selectors and constructor for rationals | two-element lists | list literals and element selection

*Implementation of lists*
Violating Abstraction Barriers

\[
\text{add\_rational}(\ [1, 2], \ [1, 4] \ )
\]

\[
def \text{divide\_rational}(x, y):
    \text{return} \ [x[0] * y[1], x[1] * y[0]]
\]
Violating Abstraction Barriers

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

def divide_rational(x, y):
    return [x[0] * y[1], x[1] * y[0]]
```
Violating Abstraction Barriers

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

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

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

def divide_rational(x, y):
    return [ x[0] * y[1], x[1] * y[0] ]
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!
Violating Abstraction Barriers
Data Representations
What are Data?
What are Data?

• We need to guarantee that constructor and selector functions work together to specify the right behavior
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
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
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 \( \text{numer}(x)/\text{denom}(x) \) must equal \( n/d \).

• Data abstraction uses selectors and constructors to define behavior.

• If behavior conditions are met, then the representation is valid.
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 $\text{num}(x) / \text{denom}(x)$ 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
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 $\text{num}(x)/\text{denom}(x)$ 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)
Rationals Implemented as Functions
Rationals Implemented as Functions

```python
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')
```
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')
Rationals Implemented as Functions

```python
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

Rationals Implemented as Functions

```python
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 \( x \)
Rationals Implemented as Functions

```python
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`.

```
x = rational(3, 8)
numer(x)
```
def rational(n, d):
    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

x = rational(3, 8)
numer(x)