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

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

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

• \(\text{rational}(n, d)\) returns a rational number \(x\)
• \(\text{numer}(x)\) returns the numerator of \(x\)
• \(\text{denom}(x)\) returns the denominator of \(x\)
Rational Numbers

Rational numbers are exact representations of fractions, typically in the form of a pair of integers. However, as soon as division occurs, the exact representation may be lost! (Demo)

Assume we can compose and decompose rational numbers:

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

- \text{numerator} and \text{denominator} are integers.
- \text{numerator} is the top number.
- \text{denominator} is the bottom number.

**Constructor**

\text{rational}(n, d)_1 \text{ returns a rational number x}

- \text{numer}(x) \text{ returns the numerator of } x
- \text{denom}(x) \text{ 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**: `rational(n, d)` returns a rational number `x`
  - **Selectors**:
    - `numer(x)` returns the numerator of `x`
    - `denom(x)` returns the denominator of `x`
Rational Number Arithmetic

Example

General Form
Rational Number Arithmetic

Example

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

General Form
Rational Number Arithmetic

Example

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

Example

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

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

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

Example

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

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

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

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

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

Example

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

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

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 \( x \)

\[
\begin{align*}
\frac{nx}{dx} \times \frac{ny}{dy} &= \frac{nx \times ny}{dx \times dy} \\
\frac{nx}{dx} + \frac{ny}{dy} &= \frac{nx \times dy + ny \times dx}{dx \times dy}
\end{align*}
\]
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 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
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.
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.
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.
Pairs
Representing Pairs Using Lists
Representing Pairs Using Lists

```python
>>> pair = [1, 2]
```
Representing Pairs Using Lists

```python
>>> pair = [1, 2]
>>> pair
[1, 2]
```
Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets
Representing Pairs Using Lists

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

A list literal:

Comma-separated expressions in brackets
Representing Pairs Using Lists

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

>>> x, y = pair
>>> x
1
```

A list literal:
Comma-separated expressions in brackets
Representing Pairs Using Lists

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

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

A list literal:
Comma-separated expressions in brackets
Representing Pairs Using Lists

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

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

A list literal:
Comma-separated expressions in brackets

"Unpacking" a list
Representing Pairs Using Lists

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

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

>>> pair[0]
1
```

A list literal:
Comma-separated expressions in brackets

"Unpacking" a list
Representing Pairs Using Lists

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

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

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

A list literal:
Comma-separated expressions in brackets

"Unpacking" a list
Representing Pairs Using Lists

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

>>> x, y = pair
>>> x
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 Pairs Using Lists

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

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

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

>>> from operator import getitem
Representing Pairs Using Lists

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

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

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

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

A list literal:
Comma-separated expressions in brackets

"Unpacking" a list

Element selection using the selection operator
Representing Pairs Using Lists

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

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

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

>>> from operator importgetitem
>>> 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
**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
Representing Pairs Using Lists

```python
>>> pair = [1, 2]
[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
def rational(n, d):
    """A representation of the rational number N/D."""
    return [n, d]
Representing Rational Numbers

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

Construct a list
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]
```
Representing Rational Numbers

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]
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]
```

Construct a list

Select item from a list
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:
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."""
```
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:
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:

\[
\begin{array}{c}
\frac{3}{2} \times \frac{5}{3} = \boxed{\frac{5}{2}} \\
\frac{15}{6} \times \frac{1}{3} = \frac{5}{2}
\end{array}
\]
Reducing to Lowest Terms

Example:

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

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

Example:

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

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

Example:

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

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

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

\[
\frac{25}{50} \times \frac{1/25}{1/25} = \frac{1}{2}
\]
Reducing to Lowest Terms

Example:

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

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

```python
from fractions import gcd
```
import gcd

def rational(n, d):

Example:

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

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

\[
\frac{25}{50} \times \frac{1/25}{1/25} = \frac{1}{2}
\]
Reducing to Lowest Terms

Example:

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

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

```
from fractions import gcd

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

def rational(n, d):
    """A representation of the rational number N/D."""
    g = gcd(n, d)  # Always has the sign of d
Reducing to Lowest Terms

Example:

\[
\begin{align*}
\frac{3}{2} & \times \frac{5}{3} = \frac{5}{2} \\
\frac{2}{5} & + \frac{1}{10} = \frac{1}{2} \\
\frac{15}{6} & \times \frac{1}{3} = \frac{5}{2} \\
\frac{25}{50} & \times \frac{1}{25} = \frac{1}{2}
\end{align*}
\]

from fractions import gcd

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]
from fractions import gcd

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]
Reducing to Lowest Terms

Example:

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

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

```
from fractions import gcd

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>
</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></td>
<td>to perform computation</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></td>
</tr>
<tr>
<td>Parts of the program that...</td>
<td>Treat rationals as...</td>
<td>Using...</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------</td>
<td>----------</td>
</tr>
<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</td>
<td>whole data values</td>
<td>add_rational, mul_rational, rationals_are_equal, print_rational</td>
</tr>
<tr>
<td>to perform computation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create rationals or implement rational operations</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>
<tr>
<td>Create rationals or implement rational operations</td>
<td>numerators and denominators</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>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>
</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>
</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></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>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></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>two-element lists</td>
<td>list literals and element selection</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>
### 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

<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**
Violating Abstraction Barriers

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

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

Does not use constructors

```python
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] )`

```python
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!
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
Data Representations
What is Data?
What is Data?

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

- Data abstraction uses selectors and constructors to define behavior.
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 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
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 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.
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 \( \text{numer}(x)/\text{denom}(x) \) must equal \( n/d \)

• Data abstraction uses selectors and constructors to define behavior

• If behavior conditions are met, then the representation is valid

You can recognize an abstract data representation by its behavior

(Demo)
Rationals Implemented as Functions
Rationals Implemented as Functions

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

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

18
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`
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

```python
x = rational(3, 8)
numer(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.

Interactive Diagram

```
x = rational(3, 8)
n = 3
d = 8
```

```python
numer(x)
```

Return value

3