61A Lecture 9

Wednesday, February 11
Announcements
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
  ▪ 0/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
  ▪ 0/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
  ▪ 2/2: No need to make changes, but keep their comments in mind for future projects
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
  ▪ 0/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
  ▪ 2/2: No need to make changes, but keep their comments in mind for future projects

• Homework 3 due Wednesday 2/18 @ 11:59pm
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
  ▪ 0/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
  ▪ 2/2: No need to make changes, but keep their comments in mind for future projects

• Homework 3 due Wednesday 2/18 @ 11:59pm
  ▪ Homework party on Tuesday 2/17 @ 5pm in 2050 VLSB
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
  ▪ 0/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
  ▪ 2/2: No need to make changes, but keep their comments in mind for future projects

• Homework 3 due Wednesday 2/18 @ 11:59pm
  ▪ Homework party on Tuesday 2/17 @ 5pm in 2050 VLSB

• Optional Hog Contest entries due Wednesday 2/18 @ 11:59pm
Announcements

• Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza!)

• Composition scores for Project 1 will mostly be assigned this week
  ▪ 0/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
  ▪ 2/2: No need to make changes, but keep their comments in mind for future projects

• Homework 3 due Wednesday 2/18 @ 11:59pm
  ▪ Homework party on Tuesday 2/17 @ 5pm in 2050 VLSB

• Optional Hog Contest entries due Wednesday 2/18 @ 11:59pm

• Midterm 1 solutions are posted; grades will be released soon
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

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

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

\[
\begin{align*}
\text{numerator} \\
\hline
\text{denominator}
\end{align*}
\]

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

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

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

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

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

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

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

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

Example

General Form

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

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

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

Example

General Form

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

Example

\[
\begin{array}{c}
\frac{3}{2} \times \frac{3}{5} = \frac{9}{10} \\
\frac{3}{2} + \frac{3}{5} = \frac{21}{10}
\end{array}
\]

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

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

General Form
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

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

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

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

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 \( x \)
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 \( \frac{nx}{dx} \)
- `numer(x)` returns the numerator of \( x \)
- `denom(x)` returns the denominator of \( x \)

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

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

These functions implement an abstract data type 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 $\frac{n}{d}$
- `numer(x)` returns the numerator of $x$
- `denom(x)` returns the denominator of $x$

These functions implement an abstract data type 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 data type 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 x
- numer(x) returns the numerator of x
- denom(x) returns the denominator of x

These functions implement an abstract data type for rational numbers
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
>>> pair = [1, 2]
>>> pair
[1, 2]

>>> 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]
[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

>>> 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

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

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

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

>>> from operator importgetitem
```

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 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

>>> 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
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
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]
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]
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]
```
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:

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

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

Example:

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

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

Example:

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

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

Example:

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

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

Example:

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

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

from fractions import gcd
Reducing to Lowest Terms

Example:

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

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

from fractions import gcd

def rational(n, d):

Reducing to Lowest Terms

Example:

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

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

```python
from fractions import gcd

def rational(n, d):
    """Construct a rational number x that represents n/d."""
```
from fractions import gcd

def rational(n, d):
    """Construct a rational number x that represents n/d."""
    g = gcd(n, d)

Reducing to Lowest Terms

Example:

\[
\frac{3}{2} \times \frac{5}{3} = \frac{5}{2} \quad \text{and} \quad \frac{2}{5} + \frac{1}{10} = \frac{1}{2}
\]
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):
    """Construct a rational number x that represents n/d."""
    g = gcd(n, d)
    return [n//g, d//g]
Reducing to Lowest Terms

Example:

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

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

```
from fractions import gcd

def rational(n, d):
    """Construct a rational number x that represents n/d."""
    g = gcd(n, d)
    return [n//g, d//g]
```
Abstraction Barriers
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>


<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><code>add_rational</code>, <code>mul_rational</code>, <code>rationals_are_equal</code>, <code>print_rational</code></td>
</tr>
</tbody>
</table>

Create rationals or implement rational operations
### 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</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>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><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>
</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, 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></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><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>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</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>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

```
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

```
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] ]
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 \( \text{numerator}(x)/\text{denominator}(x) \) 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 \( \text{numer}(x)/\text{denom}(x) \) must equal \( n/d \)

• Data abstraction uses selectors and constructors to define behavior

• If behavior conditions are met, then the representation is valid
What 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 $\text{num}(x)/\text{denom}(x)$ 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 data by 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 data by behavior

(Demo)
Rational Data Abstraction Implemented as Functions
Rational Data Abstraction 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')
Rational Data Abstraction 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')
Rational Data Abstraction 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.
Rational Data Abstraction 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`
Rational Data Abstraction 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

```python
x = rational(3, 8)
numer(x)
```
Rational Data Abstraction 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)
numer(x)
```