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

All Programmers

- 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***

All Programmers

Great Programmers

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

$$\frac{\text{numerator}}{\text{denominator}}$$

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`

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:

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

Constructor → `rational(n, d)` returns a rational number x

- `numer(x)` returns the numerator of x
- `denom(x)` returns the denominator 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:

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

$$\frac{3}{2} * \frac{3}{5}$$

Example

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

Example

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

Example

$$\frac{nx}{dx} * \frac{ny}{dy}$$

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

Example

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

$$\frac{3}{2} + \frac{3}{5}$$

Example

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

$$\frac{3}{2} + \frac{3}{5} = \frac{21}{10}$$

Example

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

$$\frac{3}{2} + \frac{3}{5} = \frac{21}{10}$$

Example

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy}$$

General Form

Rational Number Arithmetic

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$

$$\frac{3}{2} + \frac{3}{5} = \frac{21}{10}$$

Example

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

General Form

Rational Number Arithmetic Implementation

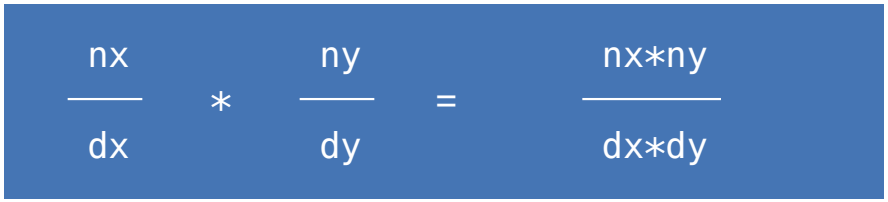
$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

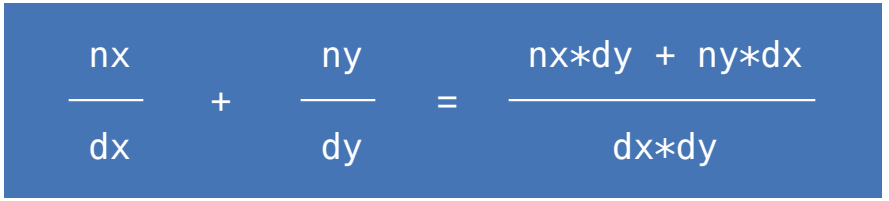
$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

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

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

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

Constructor

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

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

Rational Number Arithmetic Implementation

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

Constructor

Selectors

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

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

Rational Number Arithmetic Implementation

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

Constructor

Selectors

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

- 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 data type for rational numbers

Rational Number Arithmetic Implementation

```
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)
```

Constructor

Selectors

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

- 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 data type for rational numbers

Rational Number Arithmetic Implementation

```
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))
```

Constructor

Selectors

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

- 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 data type for rational numbers

Rational Number Arithmetic Implementation

```
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)
```

Constructor

Selectors

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx*ny}{dx*dy}$$

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx*dy + ny*dx}{dx*dy}$$

- 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 data type for rational numbers

Pairs

Representing Pairs Using Lists

Representing Pairs Using Lists

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

Representing Pairs Using Lists

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

Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets

Representing Pairs Using Lists

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

>>> x, y = pair
```

A list literal:
Comma-separated expressions in brackets

Representing Pairs Using Lists

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

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

A list literal:
Comma-separated expressions in brackets

Representing Pairs Using Lists

```
>>> 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

```
>>> 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

```
>>> 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

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

A list literal:
Comma-separated expressions in brackets

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

"Unpacking" a list

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

Element selection using the selection operator

Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets

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

"Unpacking" a list

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

Element selection using the selection operator

```
>>> from operator import getitem
```

Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets

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

"Unpacking" a list

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

Element selection using the selection operator

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

Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets

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

"Unpacking" a list

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

Element selection using the selection operator

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

Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets

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

"Unpacking" a list

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

Element selection using the selection operator

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

Element selection function

Representing Pairs Using Lists

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

A list literal:
Comma-separated expressions in brackets

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

"Unpacking" a list

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

Element selection using the selection operator

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

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]
```

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]
```

Construct a list

```
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]
```

Construct a list

```
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]
```

Select item from 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]

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

Select item from a list

(Demo)

Reducing to Lowest Terms

Example:

Reducing to Lowest Terms

Example:

$$\frac{3}{2} * \frac{5}{3}$$

Reducing to Lowest Terms

Example:

$$\frac{3}{2} * \frac{5}{3} = \frac{5}{2}$$

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:

$$\frac{3}{2} * \frac{5}{3} = \frac{5}{2} \qquad \frac{2}{5} + \frac{1}{10}$$

$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2}$$

Reducing to Lowest Terms

Example:

$$\frac{3}{2} * \frac{5}{3} = \frac{5}{2} \qquad \frac{2}{5} + \frac{1}{10} = \frac{1}{2}$$

$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2}$$

Reducing to Lowest Terms

Example:

$$\frac{3}{2} * \frac{5}{3} = \frac{5}{2} \qquad \frac{2}{5} + \frac{1}{10} = \frac{1}{2}$$

$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2} \qquad \frac{25}{50} * \frac{1/25}{1/25} = \frac{1}{2}$$

Reducing to Lowest Terms

Example:

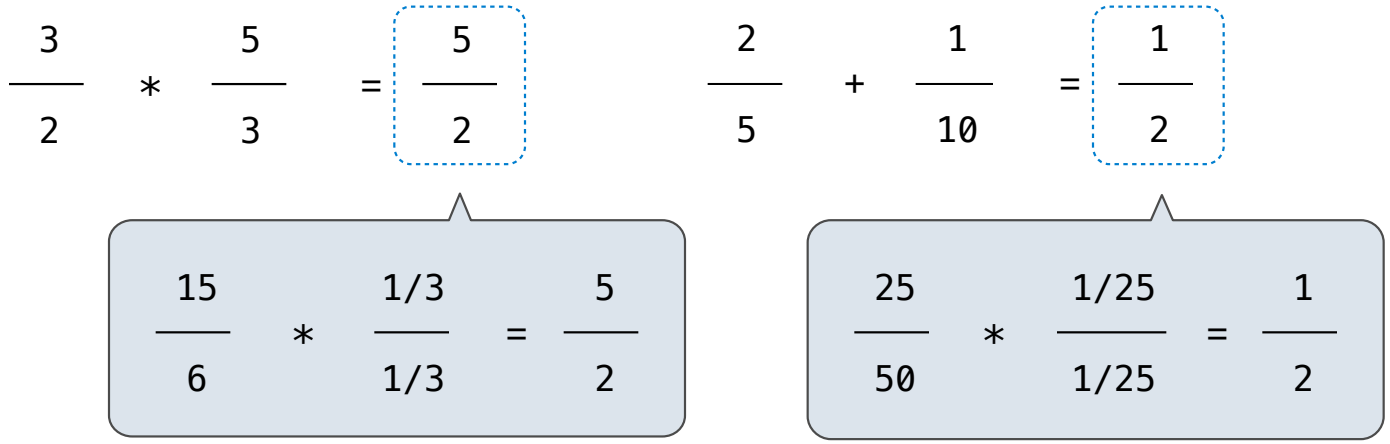
$$\frac{3}{2} * \frac{5}{3} = \frac{5}{2} \qquad \frac{2}{5} + \frac{1}{10} = \frac{1}{2}$$

$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2} \qquad \frac{25}{50} * \frac{1/25}{1/25} = \frac{1}{2}$$

```
from fractions import gcd
```

Reducing to Lowest Terms

Example:

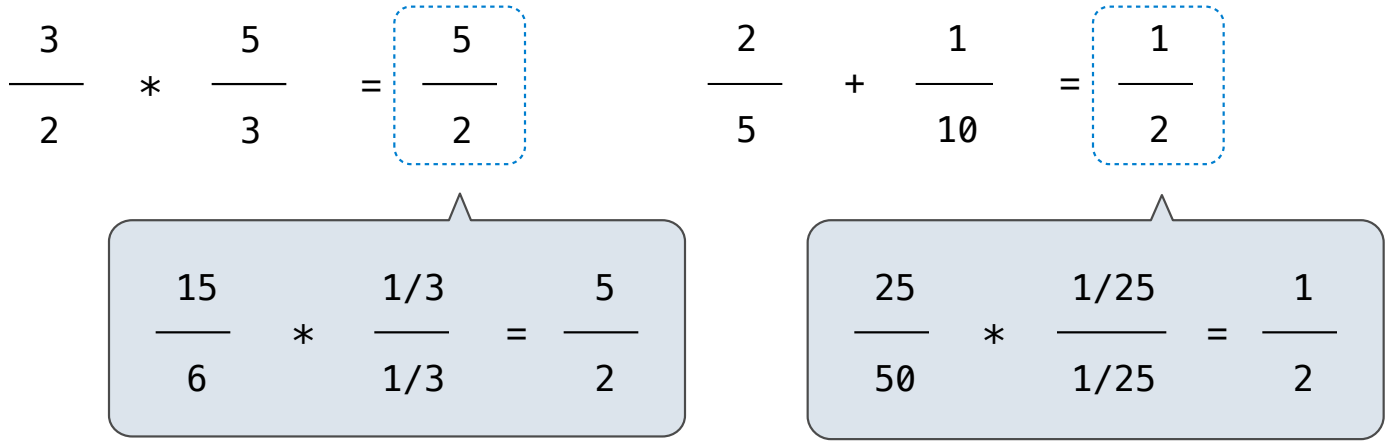



```
from fractions import gcd

def rational(n, d):
```

Reducing to Lowest Terms

Example:

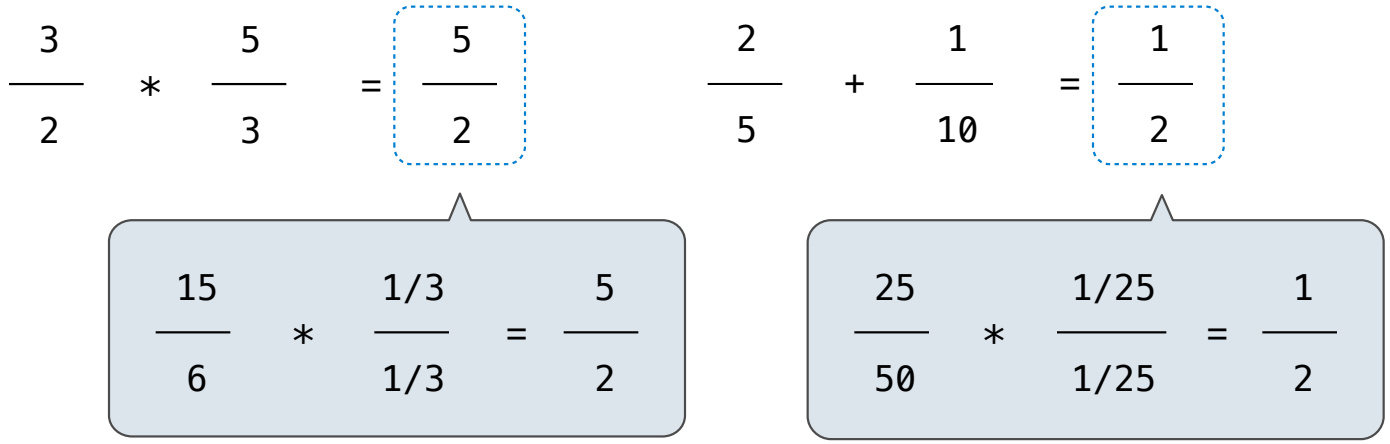



```
from fractions import gcd

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

Reducing to Lowest Terms

Example:

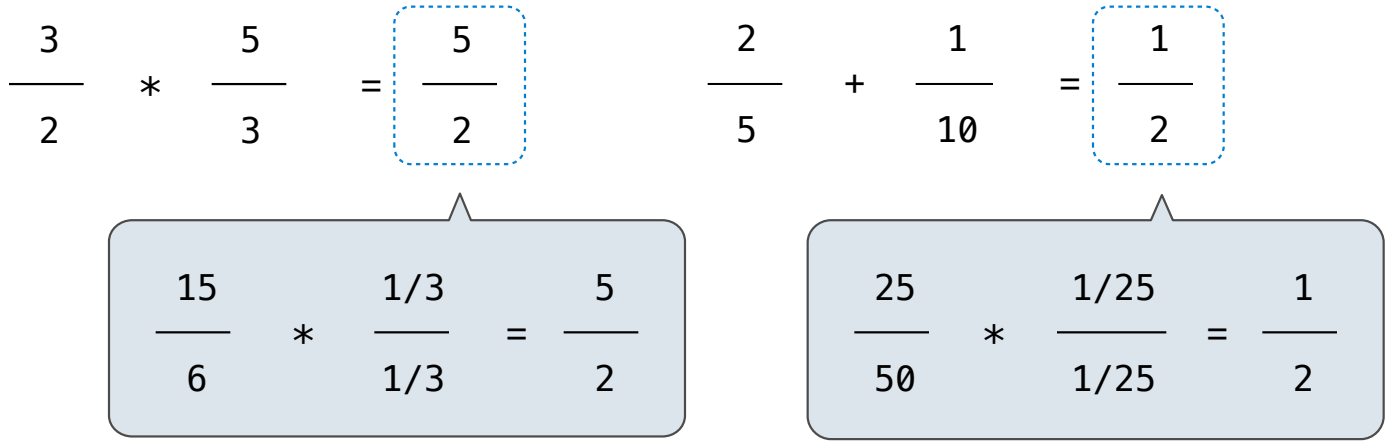



```
from fractions import gcd

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

Reducing to Lowest Terms

Example:




```
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:

$$\frac{3}{2} * \frac{5}{3} = \boxed{\frac{5}{2}} \qquad \frac{2}{5} + \frac{1}{10} = \boxed{\frac{1}{2}}$$

$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2} \qquad \frac{25}{50} * \frac{1/25}{1/25} = \frac{1}{2}$$

```
from fractions import gcd
```

Greatest common divisor

```
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

Parts of the program that...	Treat rationals as...	Using...

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation		

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`
Create rationals or implement rational operations		

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`
Create rationals or implement rational operations	numerators and denominators	

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`
Create rationals or implement rational operations	numerators and denominators	`rational`, `numer`, `denom`

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`
Create rationals or implement rational operations	numerators and denominators	`rational`, `numer`, `denom`

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`
Create rationals or implement rational operations	numerators and denominators	`rational`, `numer`, `denom`
Implement selectors and constructor for rationals		

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_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	

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_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

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	add_rational, mul_rational rationals_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

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_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

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_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

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

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

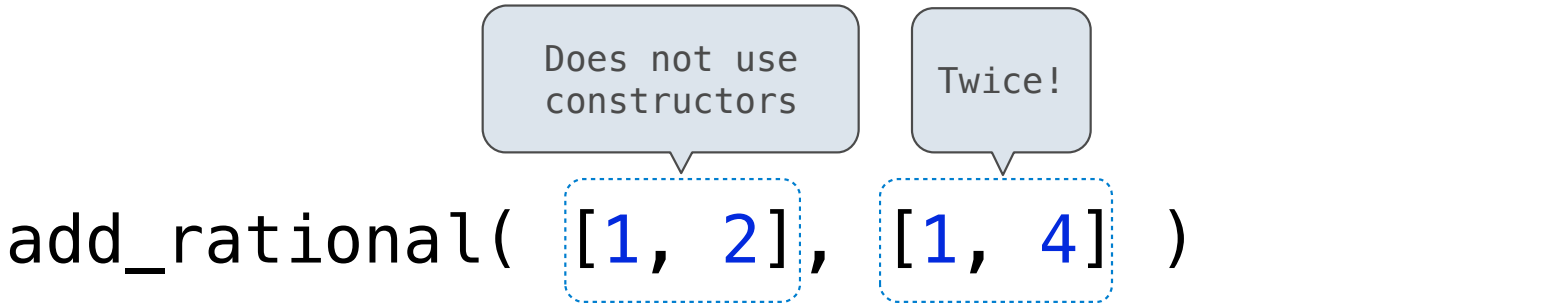
Violating Abstraction Barriers

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




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

Violating Abstraction Barriers

Does not use constructors

Twice!

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

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

No selectors!

Violating Abstraction Barriers

Does not use constructors

Twice!

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

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

No selectors!

And no constructor!

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 numer(x)/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 numer(x)/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 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 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 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
```

This function represents a rational number

```
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
```

This function represents a rational number

Constructor is a higher-order function

```
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
```

This function represents a rational number

Constructor is a higher-order function

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

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

Selector calls x

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
```

This function represents a rational number

Constructor is a higher-order function

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

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

Selector calls x

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

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
```

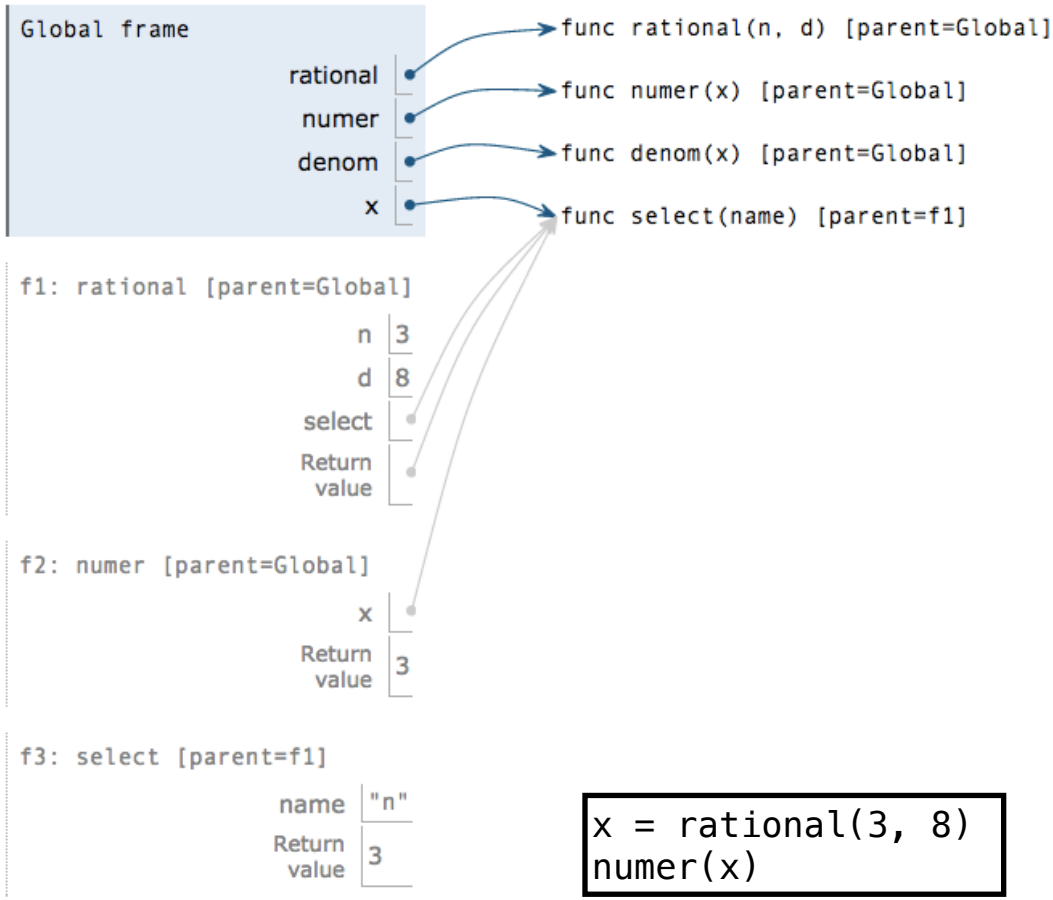
This function represents a rational number

Constructor is a higher-order function

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

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

Selector calls x




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

Interactive Diagram