61A Lecture 9

Announcements

Data Abstraction

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

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

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

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

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

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

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

Constructor

Selectors

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

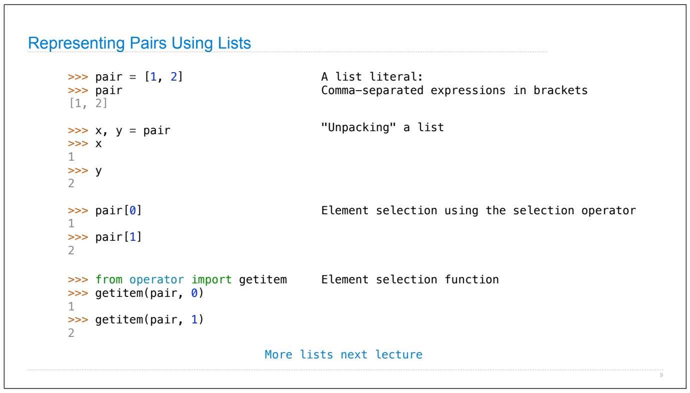
```
def add_rational(x, y):
    nx, dx = numer(x), denom(x)
    ny, dy = numer(y), denom(y)
    return rational(nx * dy + ny * dx, dx * dy)

def print_rational(x):
    print(numer(x), '/', denom(x))

def rationals_are_equal(x, y):
    return numer(x) * denom(y) == numer(y) * denom(x)
```

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

- 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 representation for rational numbers

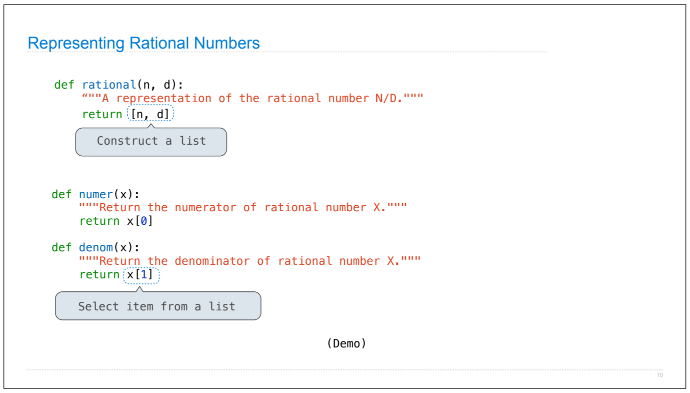
Pairs



Representing Pairs Using Lists

```
>>> pair = [1, 2]                  A list literal:
>>> pair                           Comma-separated expressions in brackets
[1, 2]

>>> x, y = pair                    "Unpacking" a list
>>> x
1
>>> y
2

>>> pair[0]                        Element selection using the selection operator
1
>>> pair[1]
2

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

More lists next lecture



Representing Rational Numbers

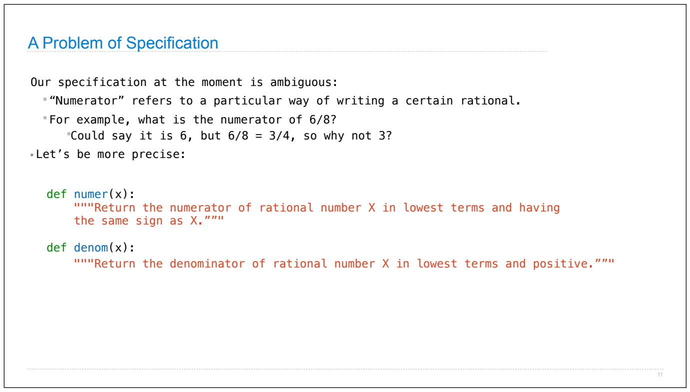
```
def rational(n, d):
    """A representation of the rational number N/D."""
    return [n, d]
```

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)

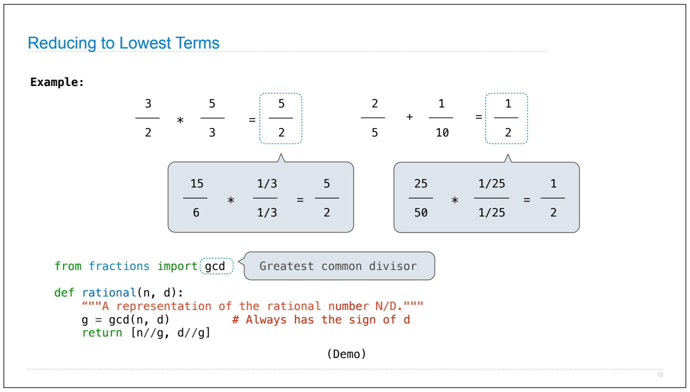


A Problem of Specification

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:

```
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:

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

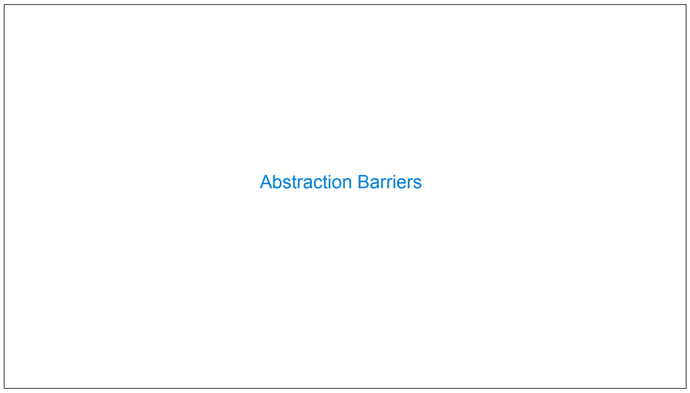
$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2} \qquad \frac{25}{50} * \frac{1/25}{1/25} = \frac{1}{2}$$

```
from fractions import gcd
```

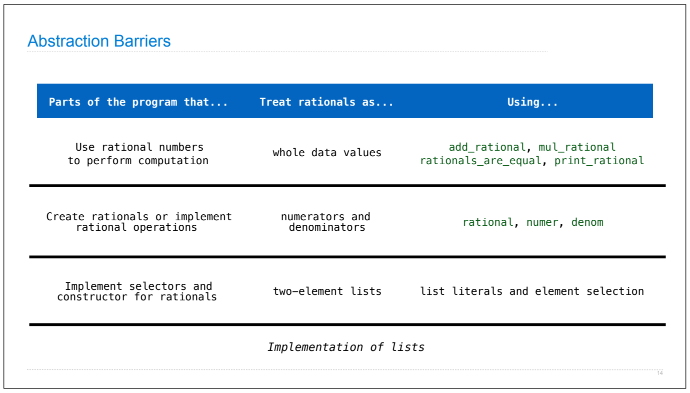
Greatest common divisor

```
def rational(n, d):
    """A representation of the rational number N/D."""
    g = gcd(n, d)         # Always has the sign of d
    return [n//g, d//g]
```

(Demo)



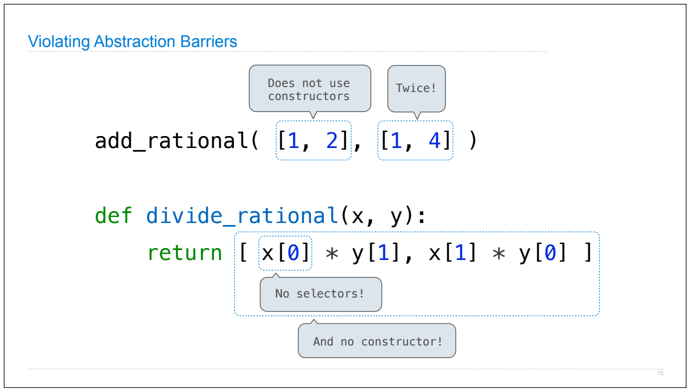
Abstraction Barriers



Abstraction Barriers

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

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!

Data Representations

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

(Demo)

Rationals Implemented as Functions

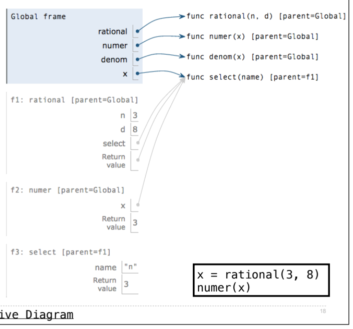
```
def rational(n, d):
    def select(name):
        if name == 'n':
            return n
        elif name == 'd':
            return d
    return select
```

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



Global frame
rational
numer
denom
x
func rational(n, d) [parent=Global]
func numer(x) [parent=Global]
func denom(x) [parent=Global]
func select(name) [parent=f1]
f1: rational [parent=Global]
n 3
d 8
select
Return value
f2: numer [parent=Global]
x
Return value 3
f3: select [parent=f1]
name "n"
Return value 3

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

Interactive Diagram