

Data Abstraction

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

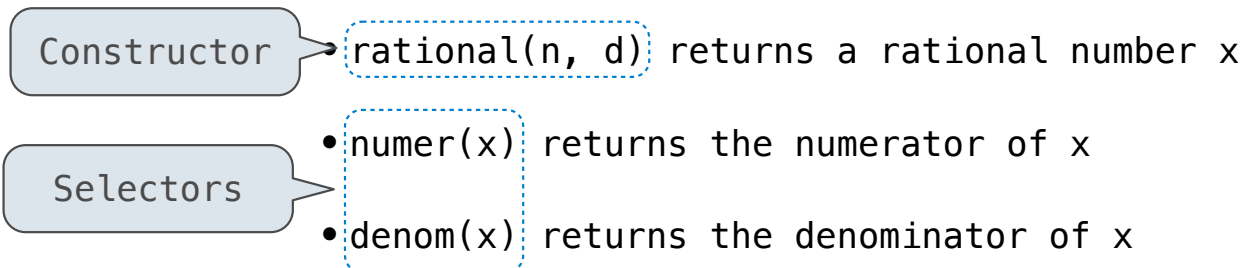
$$\frac{\text{numerator}}{\text{denominator}}$$

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:



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)
```

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

```
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 representation for rational numbers

Representing Rational Numbers

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 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:

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

$$\frac{2}{5} + \frac{1}{10} = \frac{1}{2}$$

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

$$\frac{25}{50} * \frac{1/25}{1/25} = \frac{1}{2}$$

```
from math import gcd
```

Greatest common divisor

```
def rational(n, d):
```

```
    """Construct a rational that represents n/d in lowest terms."""
```

```
    g = gcd(n, 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 are Data?

- We need to guarantee that constructor and selector functions work together to specify the right behavior
- Behavior condition: If we construct rational number x from numerator n and denominator d , then $\text{numer}(x)/\text{denom}(x)$ must equal n/d
- Data abstraction uses selectors and constructors to define behavior
- If behavior conditions are met, then the representation is valid

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')
```

Selector calls x

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

