

## 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

---

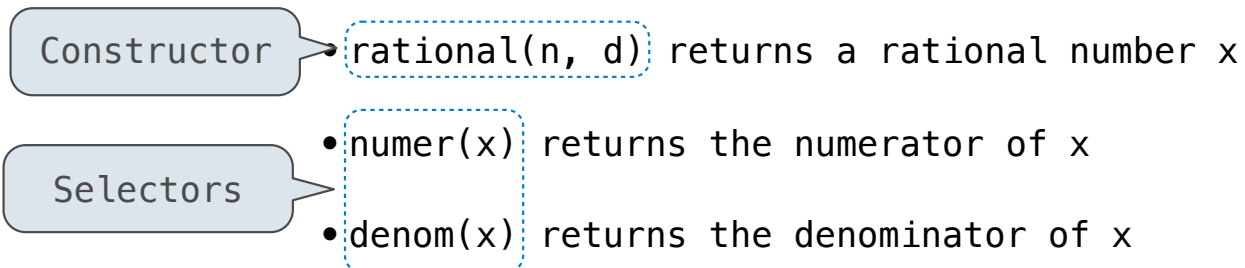
$$\frac{\text{numerator}}{\text{denominator}}$$

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(number(x) * number(y),  
                    denom(x) * denom(y))
```

Constructor

Selectors

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

```
def print_rational(x):  
    print(number(x), '/', denom(x))
```

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

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

These functions implement an abstract representation for rational numbers

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

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

Pairs



## 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 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 fractions 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	<code>add_rational, mul_rational</code> <code>rationals_are_equal, print_rational</code>
Create rationals or implement rational operations	numerators and denominators	<code>rational, numer, denom</code>
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)



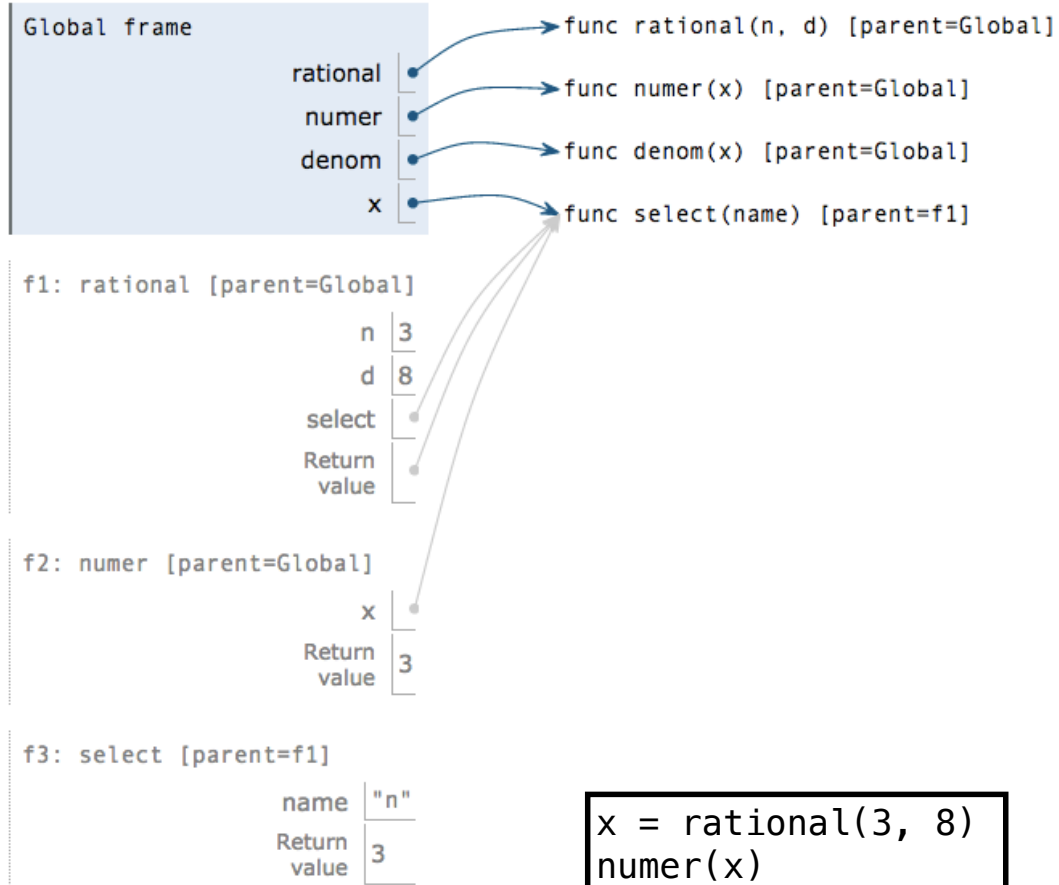
## Rationals Implemented as Functions

This  
function  
represents  
a rational  
number

Constructor is a higher-order function

Selector calls x

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



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

## Dictionaries

```
{'Dem': 0}
```

## Limitations on Dictionaries

---

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)
- Two **keys cannot be equal**; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value