### 61A Lecture 8

Wednesday, September 14

### **Data Abstraction**

- Compound objects combine primitive objects together
- A date: a year, a month, and a day
- A geographic position: latitude and longitude
- An abstract data type lets us manipulate compound objects 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  $\it representation$  and  $\it use$

rogrammers Great

ogrammers All

**Rational Numbers** 

numerator

denominator

Exact representation of fractions

A pair of integers

Wishful

thinking

As soon as division occurs, the exact representation is lost!

Assume we can compose and decompose rational numbers:

Constructor make\_rat(n, d) returns a rational number x numer(x) returns the numerator of x Selectors denom(x) returns the denominator of x

# **Rational Number Arithmetic**

### Example:

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

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

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

### **Rational Number Arithmetic Implementation**

```
def mul_rat(x, y):
    """Multiply rational numbers x and y."""
    return (make_rat(numer(x) * numer(y), denom(x) * denom(y)))
                     Constructor
                                                                                Selectors
def add_rat(x, y):
    """Add rational numbers x and y."""
    nx, dx = numer(x), denom(x)
    ny, dy = numer(y), denom(y)
    return make_rat(nx * dy + ny * dx, dx * dy)
def eq_rat(x, y):
    """Return whether rational numbers x and y are equal."""
    return numer(x) * denom(y) == numer(y) * denom(x)
```

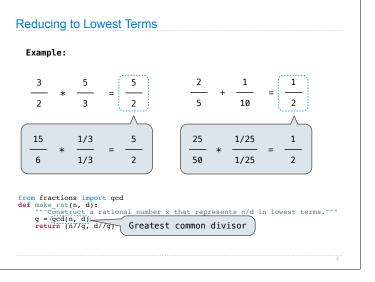
- make\_rat(n, d) returns a rational number x
- numer(x) returns the numerator of x
- denom(x) returns the denominator of x

**Tuples** 

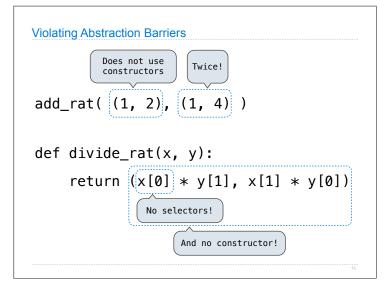
```
>>> pair = (1, 2)
                                           A tuple literal:
                                           Comma-separated expressions
>>> x, y = pair
>>> x
                                           "Unpacking" a tuple
>>> y
>>> from operator import getitem
>>> getitem(pair, 0)
                                           Element selection
>>> getitem(pair, 1)
```

More tuples next lecture

# def make\_rat(n, d): """Construct a rational number x that represents n/d.""" return(n, d)) Construct a tuple from operator import getitem def numer(x): """Return the numerator of rational number x.""" return getitem(x, 0) def denom(x): """Return the denominator of rational number x.""" return (getitem(x, 1)) Select from a tuple



# Abstraction Barriers Rational numbers in the problem domain add\_rat mul\_rat eq\_rat Rational numbers as numerators & denominators make\_rat numer denom Rational numbers as tuples tuple getitem However tuples are implemented in Python



# What is Data?

- We need to guarantee that constructor and selector functions together specify the right behavior.
- Rational numbers: If we construct x from n and d, then numer(x)/denom(x) must equal n/d.
- $^{\circ}$  An abstract data type is some collection of selectors and constructors, together with some behavior conditions.
- If behavior conditions are met, the representation is valid.

You can recognize data types by behavior, not by bits

## Behavior Conditions of a Pair

To implement our rational number abstract data type, we used a two-element tuple (also known as a pair).

What is a pair?

Constructors, selectors, and behavior conditions:

If a pair p was constructed from values x and y, then

- $\bullet$  getitem\_pair(p, 0) returns x, and
- getitem\_pair(p, 1) returns y.

Together, selectors are the inverse of the constructor

Generally true of container types. < Not true for

Not true for rational numbers

```
Using a Functionally Implemented Pair
```

```
>>> p = make_pair(1, 2)
>>> getitem_pair(p, 0)
1
>>> getitem_pair(p, 1)
2
```

As long as we do not violate the abstraction barrier, we don't need to know that pairs are just functions

If a pair p was constructed from values  $\boldsymbol{x}$  and  $\boldsymbol{y}\text{, then}$ 

- getitem\_pair(p, 0) returns x, and
- getitem\_pair(p, 1) returns y.

This pair representation is valid!

-----