Lecture11 Data Abstraction

Data Abstraction

Data Abstraction

- Compound objects combine 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 ***representation*** and ***use***

7

Rational Numbers

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

Exact representation of fractions

A pair of integers

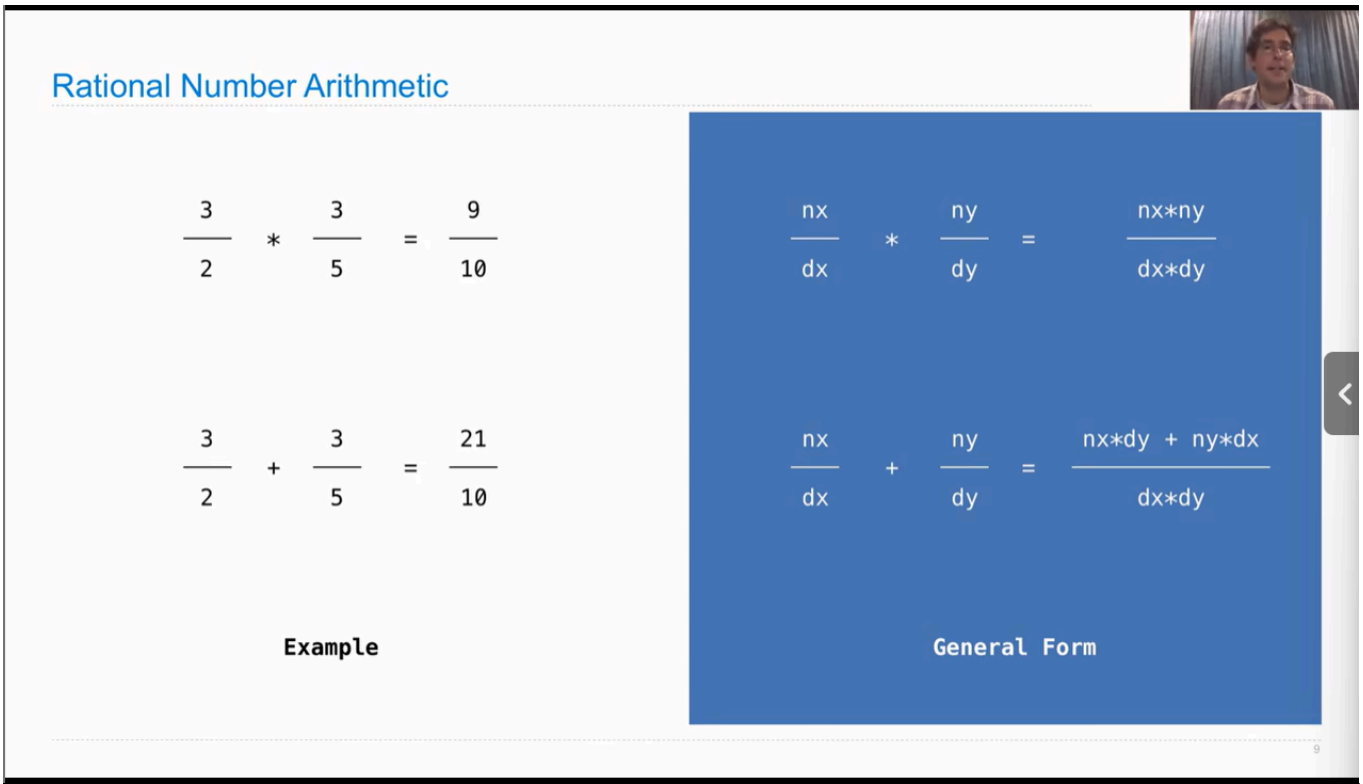
As soon as division occurs, the exact representation may be lost!

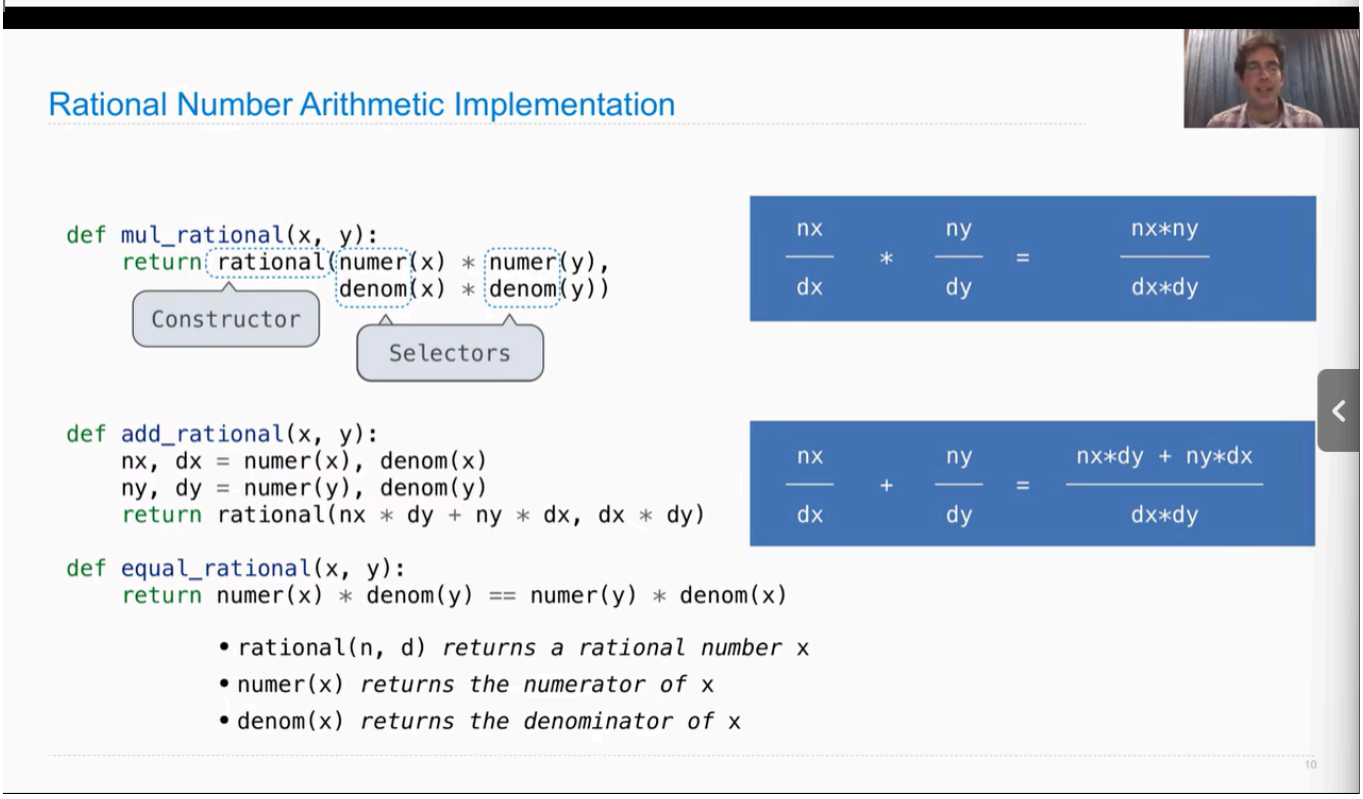
Assume we can compose and decompose rational numbers:

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

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

8

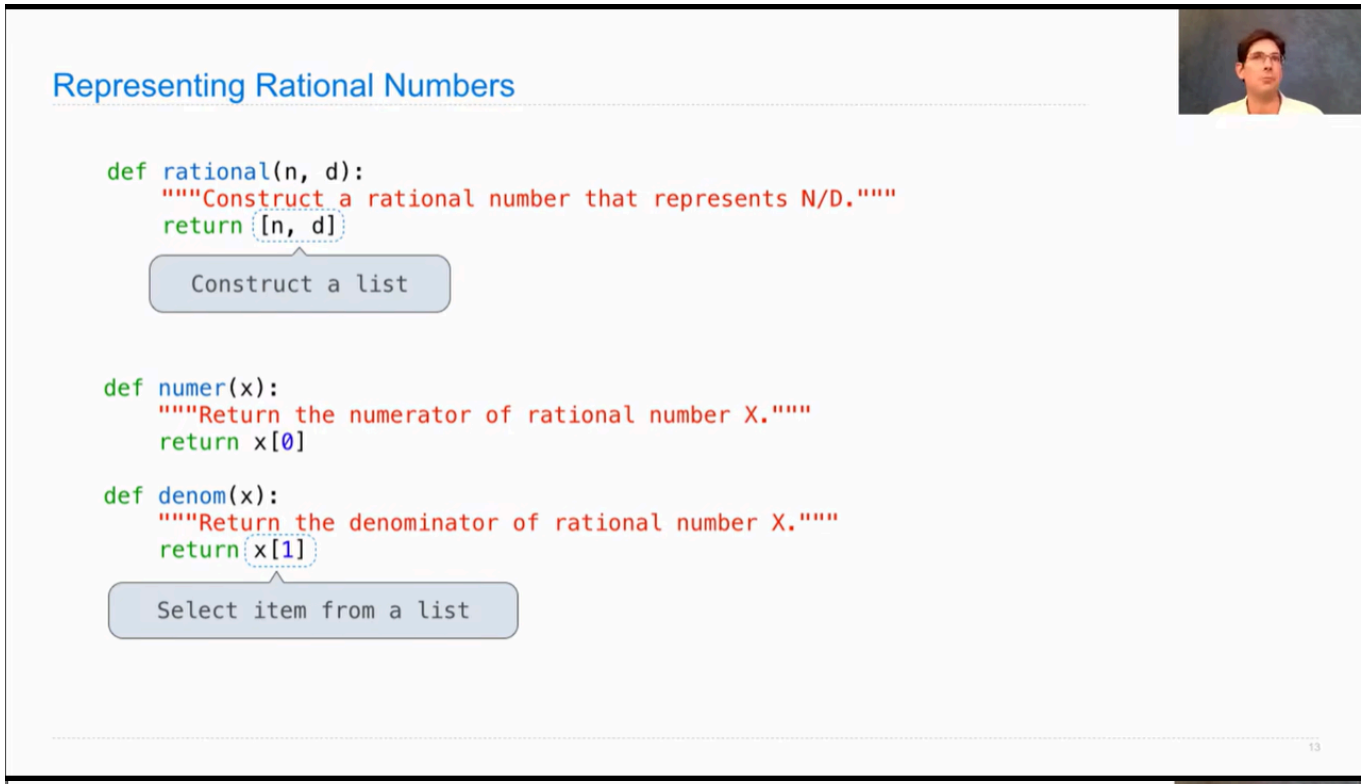


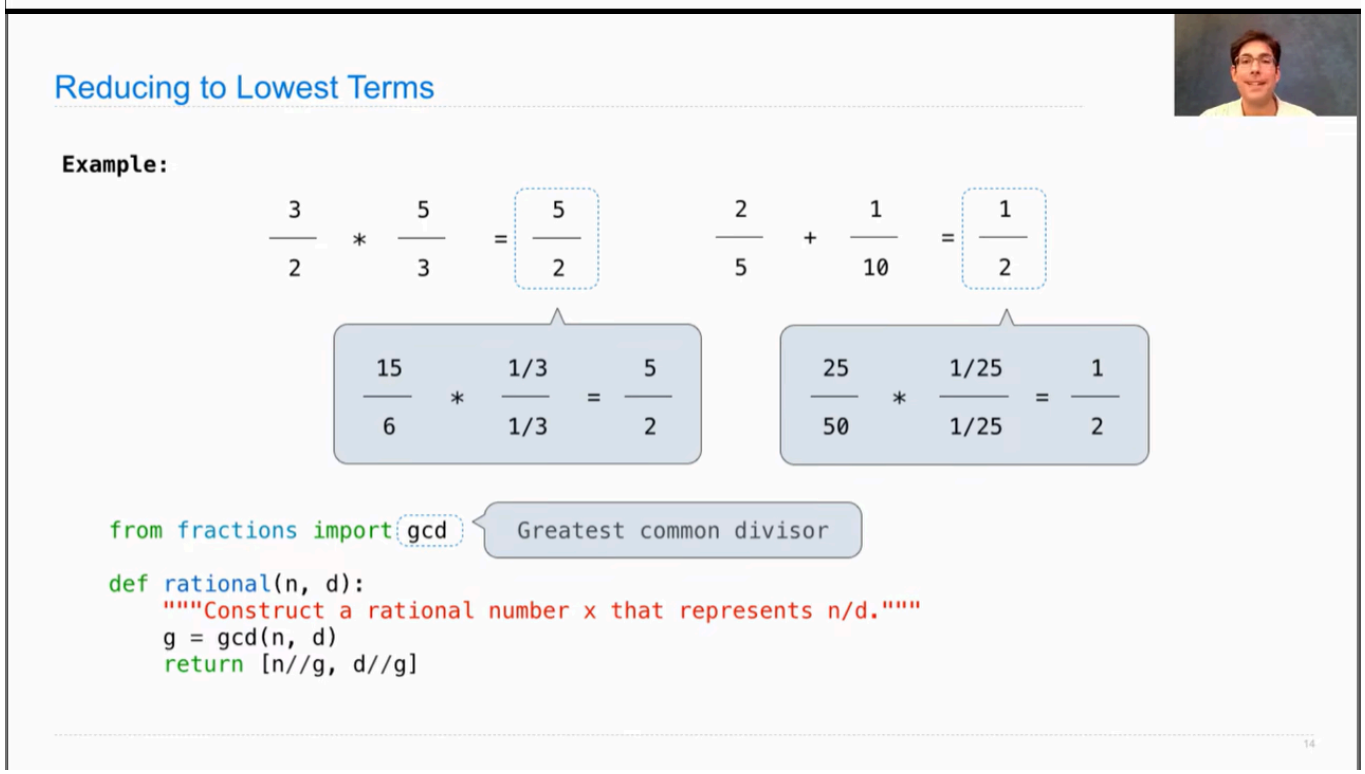


Pairs

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

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

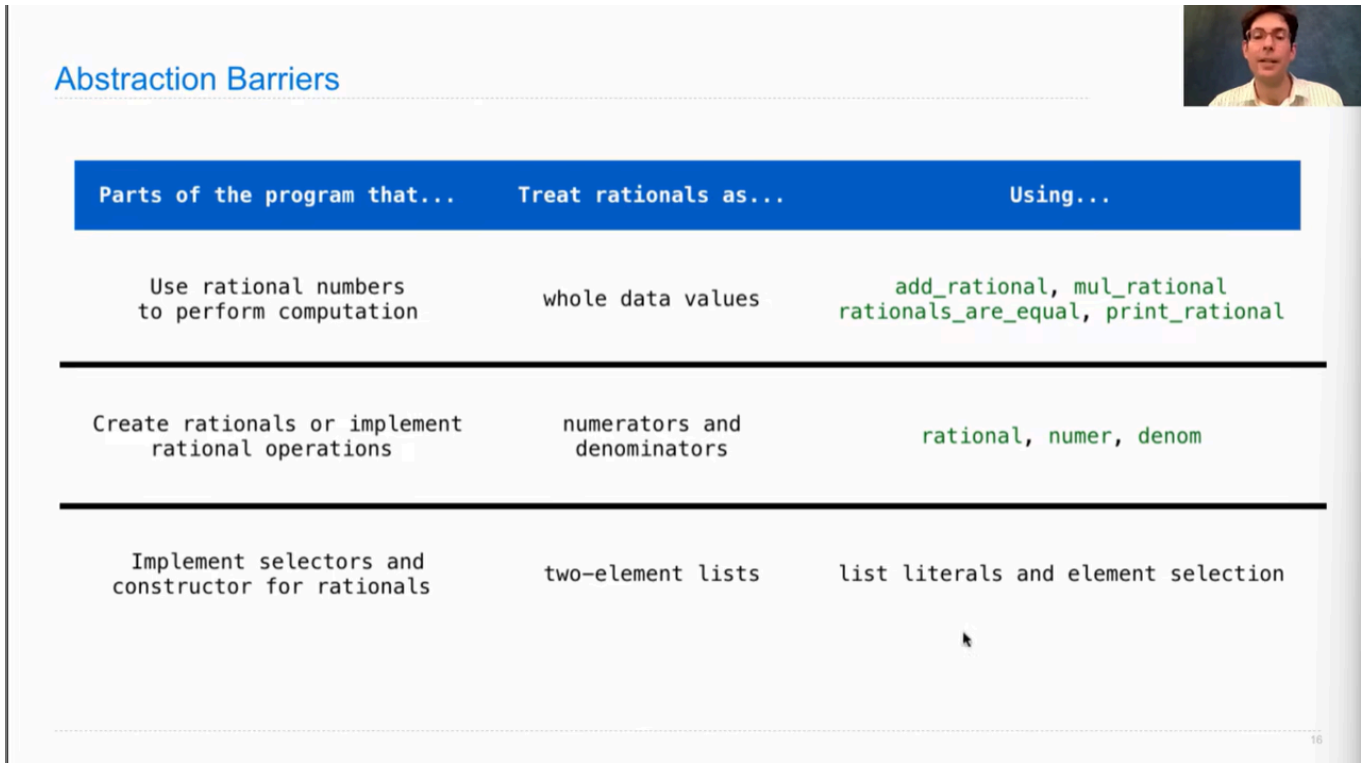
```
1
>>> y
2
#this is called unpacking the list

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

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





Abstraction Barriers

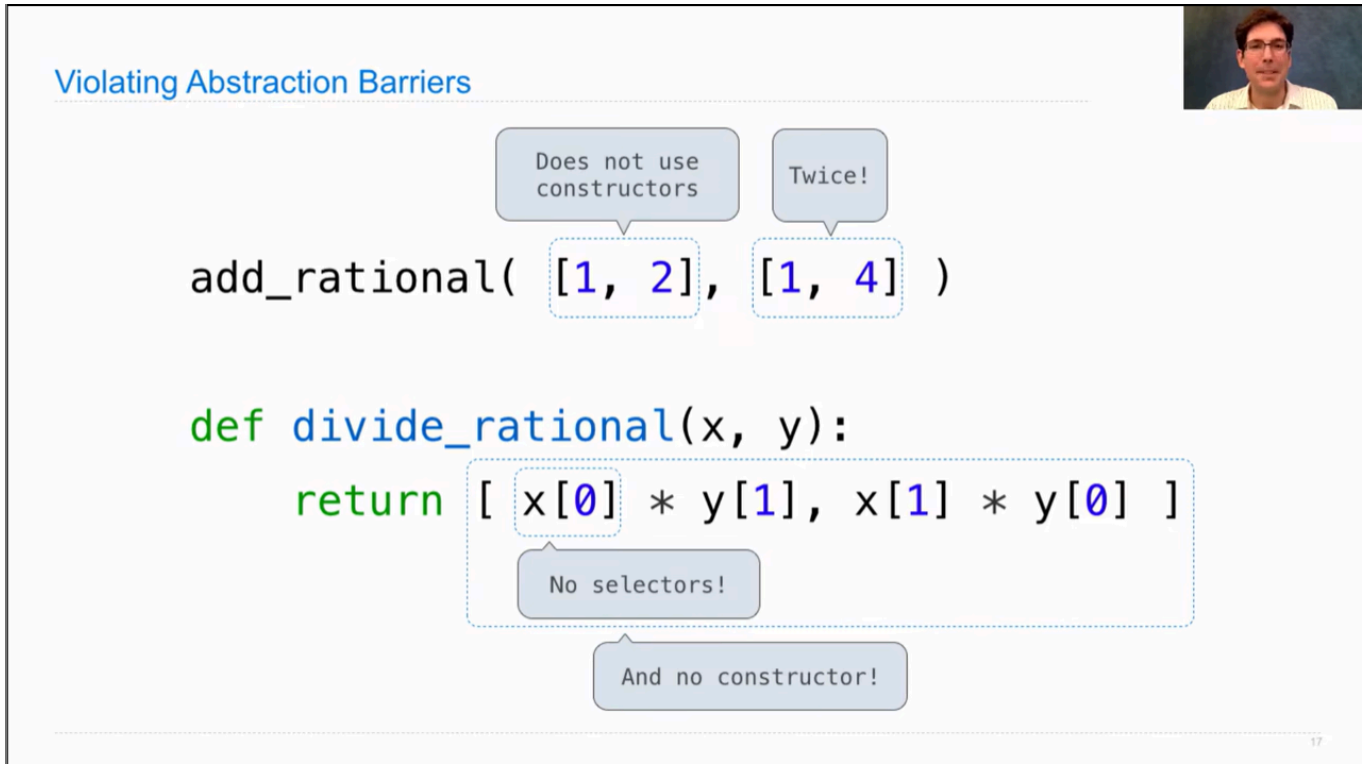
don't need to let the function know every part of the program, but only a part of it
moreover, it allows changes which would be take advantage of of other parts and keep consistency



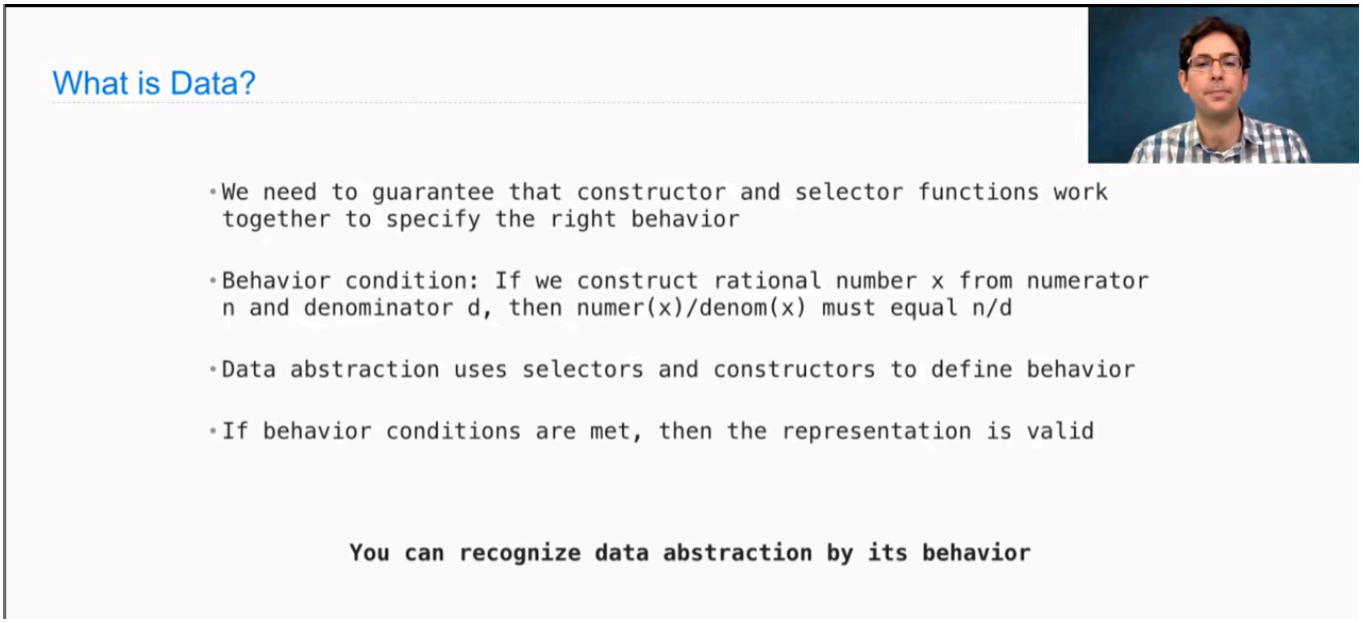
should not be using the function in another different "layer", obey the abstractions

? need a deeper understanding

```
use the level of abstraction appropriately
```

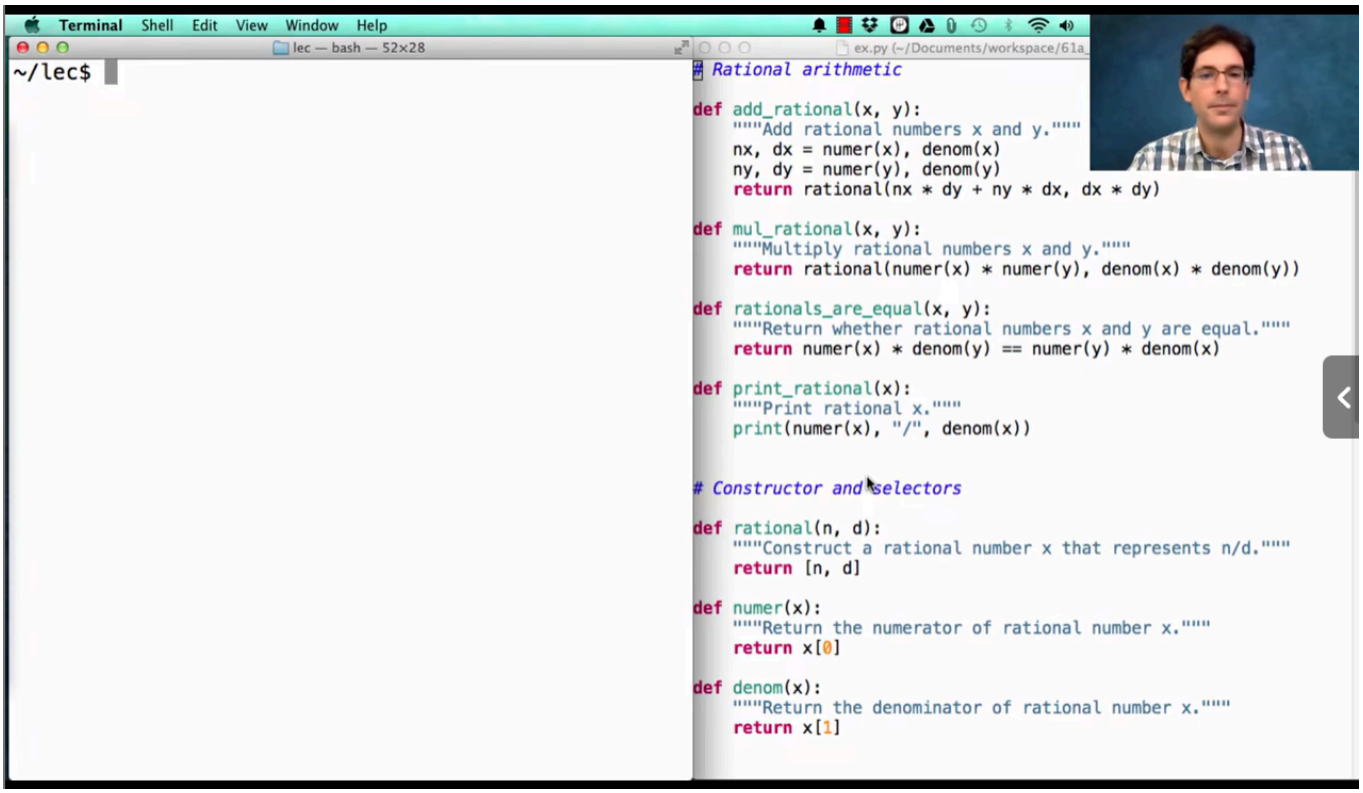


Data Representation



> this is exactly the same to what happens in math! we don't actually care about how we define sth. in fact, we would say that, no matter what that have these characteristics we would call them the same thing! that's abstraction!

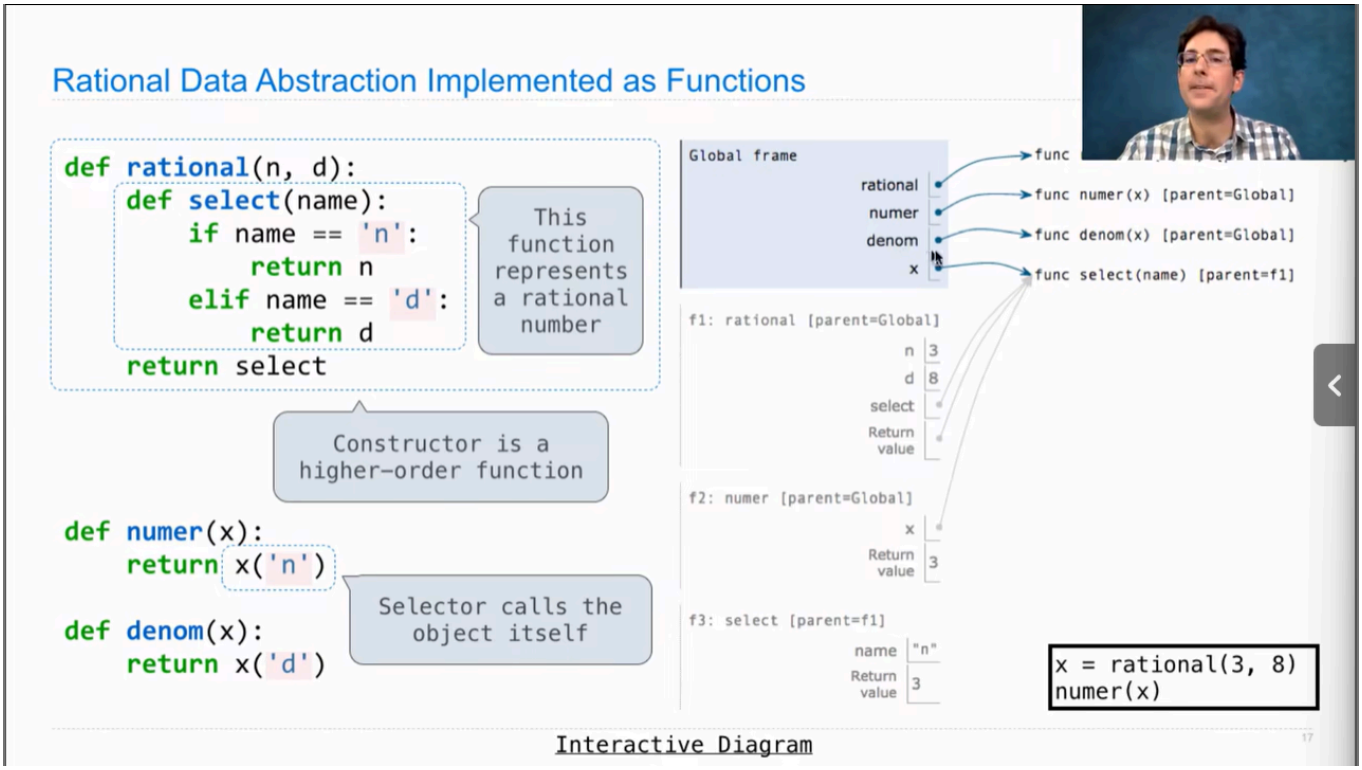
former



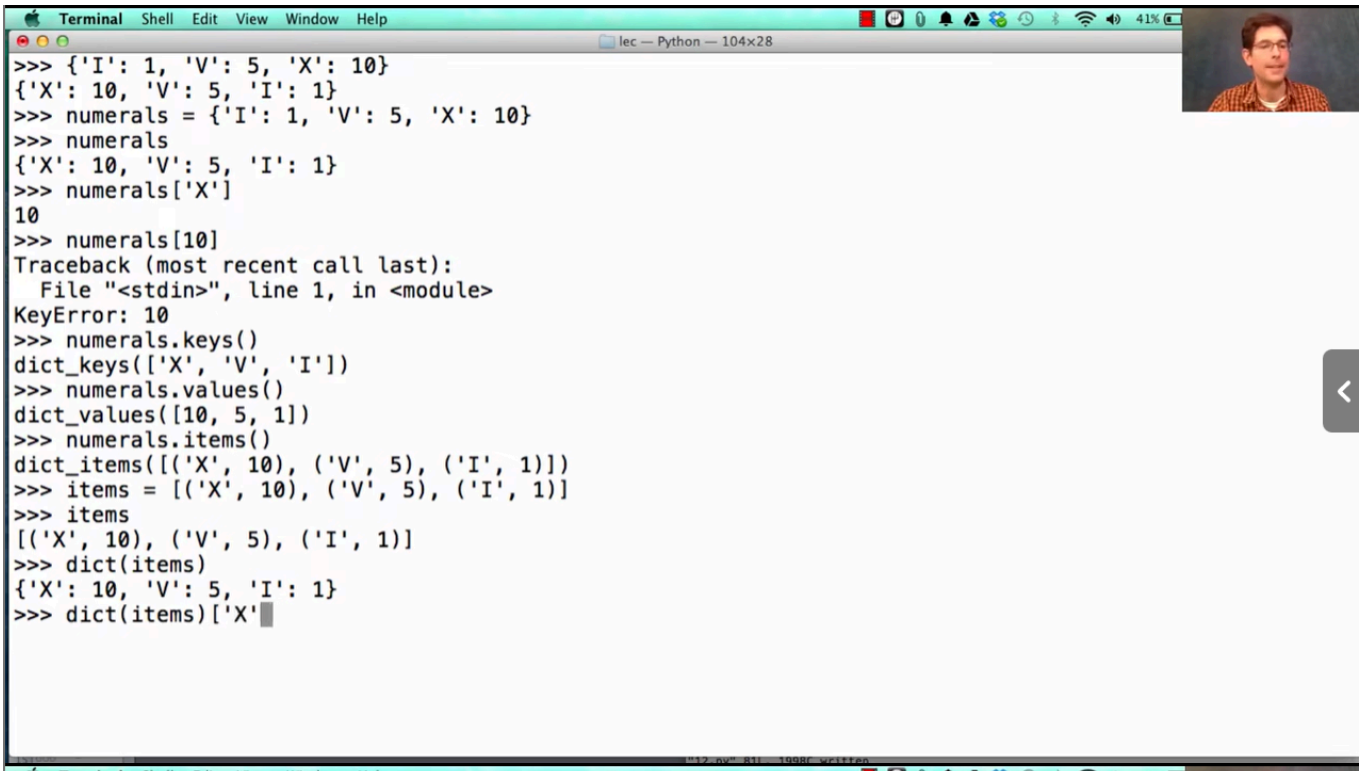
after

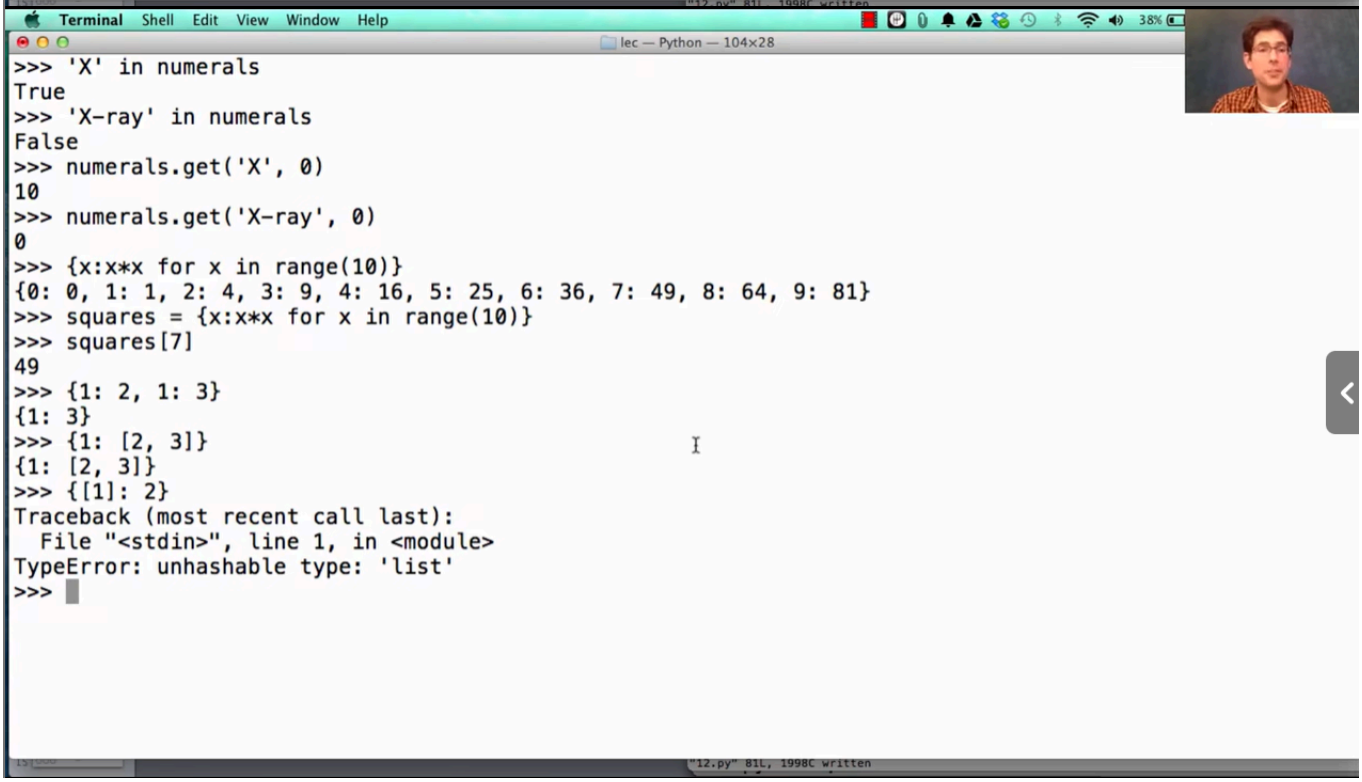
```
# Constructor and selectors

def rational(n, d):
    """Construct a rational number x that represents n/d."""
    def select(name):
        if name == 'n':
            return n
        elif name == 'd':
            return d
    return select

def numer(x):
    """Return the numerator of rational number x."""
    return x('n')

def denom(x):
    """Return the denominator of rational number x."""
    return x('d')

~
-- VISUAL --
```

> we want to show the power of data abstraction here, where we change the representation but functions in different layers worked well.



Dictionaries


Terminal Shell Edit View Window Help
lec — Python — 104×28
>>> {'I': 1, 'V': 5, 'X': 10}
{'X': 10, 'V': 5, 'I': 1}
>>> numerals = {'I': 1, 'V': 5, 'X': 10}
>>> numerals
{'X': 10, 'V': 5, 'I': 1}
>>> numerals['X']
10
>>> numerals[10]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 10
>>> numerals.keys()
dict_keys(['X', 'V', 'I'])
>>> numerals.values()
dict_values([10, 5, 1])
>>> numerals.items()
dict_items([('X', 10), ('V', 5), ('I', 1)])
>>> items = [('X', 10), ('V', 5), ('I', 1)]
>>> items
[('X', 10), ('V', 5), ('I', 1)]
>>> dict(items)
{'X': 10, 'V': 5, 'I': 1}
>>> dict(items)['X']


Terminal Shell Edit View Window Help
lec — Python — 104×28
>>> 'X' in numerals
True
>>> 'X-ray' in numerals
False
>>> numerals.get('X', 0)
10
>>> numerals.get('X-ray', 0)
0
>>> {x:x*x for x in range(10)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
>>> squares = {x:x*x for x in range(10)}
>>> squares[7]
49
>>> {1: 2, 1: 3}
{1: 3}
>>> {1: [2, 3]}
{1: [2, 3]}
>>> {[1]: 2}
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>>
"12.py" 81L, 1998C written

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