CS61A Lecture 12

Wednesday, September 25th, 2019

Compound values are values that combine other values together. For example, a date with a year, month and day, or position via latitude and longitude.

Data abstraction lets us manipulate compound values are units, by isolating two parts of any program that uses data:

- How data are represented in parts.
- How data are represented as units.

Data abstraction is a methodology by which functions enforce an abstraction barrier between representation and use. It is one of the most important shifts in the history of computer science, because it led to the beginning of object-oriented programming.

Rational Numbers

An example would be a fraction. Programming languages have a strange quirk in representing decimals (floats) in that they do not store the decimal to a high precision. For example,

```
>>> 1/3 == 0.33333333333333333333333333333333333333795
True
```

They only store to about 15 or so decimal points! So to be precise, we must use rational numbers, $\frac{numerator}{denominator}$. Rational numbers are a pair of integers that can be represented by a function, `constructor`:

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

We can define a function to multiply:

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

and add:

Notice we got this far without ever having to define `rational`, `numer` and `denom`, because we don't care how they work as long as they return what we expect to do.

Representing Pairs

A list literal is a comma-separated expression in brackets. We can unpack them by assigning them to variables:

```
>>> x, y = [1, 2]
>>> x
1
```

```
>>> y
2
```

We can represent rational numbers using lists:

```
def rational(n,d):
    return [n,d]

def numer(x):
    return x[0]

def numer(y):
    return x[1]
```

We can also define a function to print rationals:

```
def print_rational(x):
    print(numer(x),"/",denom(x))
```

Let's try:

```
>>> print_rational(add_rational(rational(2,5), rational(6,10))
50 / 50
```

We can simplify this by adding a function:

```
from math import gcd

def rational(n,d):
    g = gcd(n,d)
    return [n//g,d//g]
```

Because we've already defined *all* of the other functions, we don't need to worry about the other functions and add these two lines to add the ability to simplify fractions.

Abstraction Barriers

When writing longer programs, it is important to isolate the parts of the program that represent data. This is so that if you want to change how you show your data later on, it doesn't break the program as a whole, or require you to rewrite the entire program.

By doing so, we make our program **modular**.

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

The lines above represent the abstraction barriers that we must keep in order to isolate our program properly.

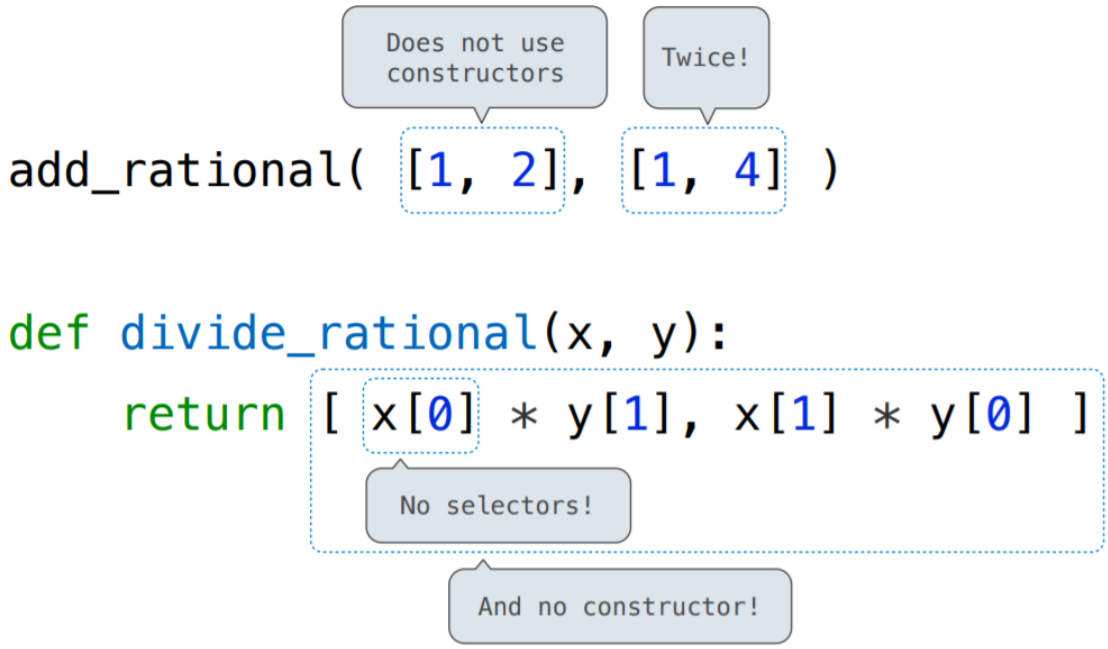
What's wrong with this data?

Take a look at the below code:

```
add_rational([1,2],[1,4])

def divide_rational(x,y):
    return [ x[0] * y[1], y[0] * x[1] ]
```

The above code **violates** abstraction barriers, and doing so only makes someone else do more work down the line.



What is Data?

- We need to guarantee the 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

You can now change the definition of `rational` to return a function instead of a list, or whatever definition of `rational` you can think of. And as long as you change `numer` and `denom` too, the rest of your program **doesn't need to be changed**.

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

```
>>> {1:'I', 5:'V', 10: 'X'}
{1:'I', 5:'V', 10: 'X'}
>>> d = {1:'I', 5:'V', 10: 'X'}
>>> d[5]
'V'
```

This first restriction is tied to Python's underlying implementation of dictionaries, while the second restriction is part of the dictionary abstraction.

You can look up 'V' using its key, although it doesn't work the other way around.

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

Every frame is basically a dictionary in Python: under the hood, it uses the same implementation, because each frame has a unique name and its values.