Announcements

- Guerrilla section this Saturday 2/14 on recursion (Please RSVP on Piazza)
- Composition scores for Project 1 will mostly be assigned this week
- 8/2: Make changes suggested by the TA/tutor in order to earn back the 2 lost points
- 2/2: No need to make changes, but keep their comments in mind for future projects
- Homework 3 due Wednesday 2/18 @ 11:59pm
- Homework party on Tuesday 2/17 @ 5pm in 2050 VLSB
- Optional Hog Contest entries due Wednesday 2/18 @ 11:59pm
- Midterm 1 solutions are posted; grades will be released soon

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

Rational Numbers

- 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:

  Constructor: rational(n, d) returns a rational number x
  Selectors: *num(x) returns the numerator of x
             *denom(x) returns the denominator of x

Rational Number Arithmetic

- Constructor: rational(n, d) returns a rational number x
- Selectors: *num(x) returns the numerator of x
             *denom(x) returns the denominator of x
- General Form

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

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

Pairs
Representing Pairs Using Lists

```python
>>> pair = [1, 2]
>>> x, y = pair  # "Unpacking" a list
>>> x
1
>>> y
2
>>> pair[0]  # Element selection using the selection operator
1
>>> pair[1]
2
>>> from operator import getitem
>>> getitem(pair, 0)
1
>>> getitem(pair, 1)
2
More lists next lecture
```

Representing Rational Numbers

```python
def rational(n, d):
    #"Construct a rational number that represents N/D."
    return [n, d]
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]
```

Reducing to Lowest Terms

Example:

```
3 * 5 = 6
2 * 2 = 2
25
15 * 1/3 = 5
18
15
15
```

```
from fractions import gcd

def rational(n, d):
    #"Construct a rational number x that represents n/d."
    g = gcd(n, d)
    return [n//g, d//g]
```

Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat rationals as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use rational numbers to perform computation</td>
<td>whole data values</td>
<td><code>add_rational</code>, <code>mul_rational</code>, <code>rational</code>, <code>num</code>, <code>denom</code></td>
</tr>
<tr>
<td>Create rational or implement rational operations</td>
<td>numerators and denominators</td>
<td><code>rational</code>, <code>num</code>, <code>denom</code></td>
</tr>
<tr>
<td>Implement selectors and constructor for rationals</td>
<td>two-element lists</td>
<td>list literals and element selection</td>
</tr>
</tbody>
</table>

Implementation of Lists

Does not use constructors Twice!

Does not use selectors!

Violating Abstraction Barriers

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

What is 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 `num(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 data by behavior

(Demo)
Rational Data Abstraction Implemented as Functions

```python
def rational(n, d):
    def select(name):
        if name == 'n':
            return n
        elif name == 'd':
            return d
        return select
    return select

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

This function represents a rational number and is implemented using higher-order functions. The constructor is a higher-order function that takes two arguments, the numerator and the denominator. The selector function is a higher-order function that returns either the numerator or the denominator based on the input parameter. The `numer` and `denom` functions are higher-order functions that select the numerator and denominator, respectively, from the rational data structure.