OBJECT-ORIENTED ETUDES

Curt Clifton
Rose-Hulman Institute of Technology

RECALL: ITERATORS

- Can make our own iterable classes by:
    - Adding *__iter__(self)*
    - Making it return an object with a *next()* method
    - *next()* raises *StopIteration* at end

GENERATORS

- A wicked cool tool for creating iterators
- Imagine writing a function to **print** all of the items that should be iterated over
  - But instead of printing, we use *yield*
- A function that *yields* instead of *returning*, is a generator—it returns an iterator whose *next()* method returns the yielded value, but remembers where it left off

GENERATOR EXAMPLES

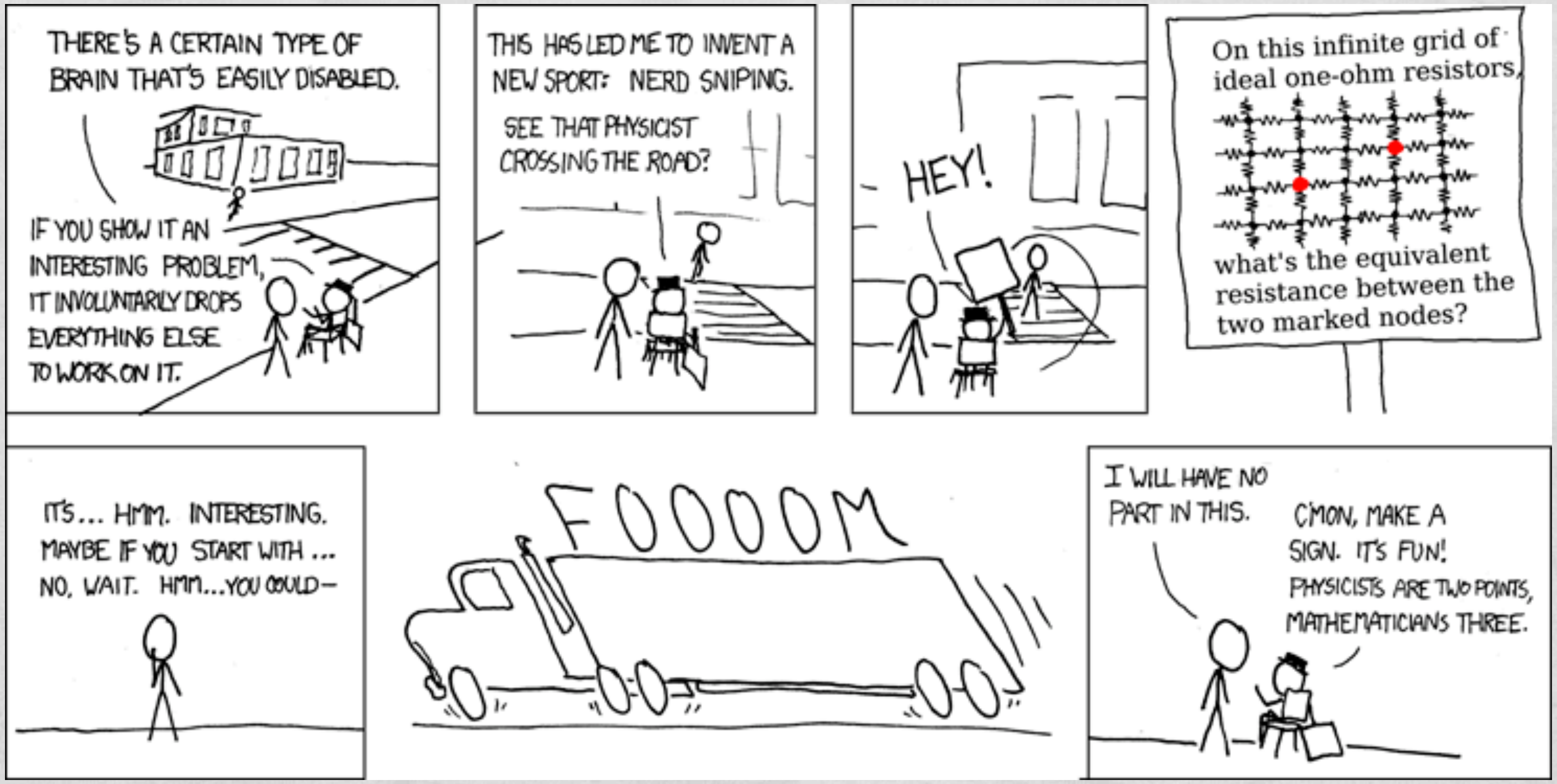
```python
class ShuffleIterator:
    def __init__(self, data):
        self.data = data
        self.order = range(len(data))
        random.shuffle(self.order)
        self.index = len(data)
    def __iter__(self):
        return self
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index -= 1
        itemIndex = self.order[self.index]
        return self.data[itemIndex]

s = 'Ni!'
for c in ShuffleIterator(s):
    print c
```

```python
def shuffle(data):
    order = range(len(data))
    random.shuffle(order)
    for itemIndex in order:
        yield data[itemIndex]

for c in shuffle(s):
    print c
```

NERD SNIPING


THERE'S A CERTAIN TYPE OF BRAIN THAT'S EASILY DISABLED.
IF YOU SHOW IT AN INTERESTING PROBLEM, IT INVOLUNTARILY DROPS EVERYTHING ELSE TO WORK ON IT.
THIS HAS LED ME TO INVENT A NEW SPORT: NERD SNIPING.
SEE THAT PHYSICIST CROSSING THE ROAD?
HEY!
On this infinite grid of ideal one-ohm resistors, what's the equivalent resistance between the two marked nodes?
IT'S... HMM. INTERESTING. MAYBE IF YOU START WITH ... NO, WAIT. HMM...YOU COULD—
FOOOOM
I WILL HAVE NO PART IN THIS.
C'MON, MAKE A SIGN. IT'S FUN! PHYSICISTS ARE TWO POINTS, MATHEMATICIANS THREE.

I first saw this problem on the Google Labs Aptitude Test.
A professor and I filled a blackboard without getting anywhere.
Have fun!

OBJECT-ORIENTED ETUDES

- These aren’t intended to show you good design
- They’re intended to sharpen your skills
- Focus in the object-oriented etudes will be on:
  - Polymorphism
  - Method dispatch

A WARM-UP: BOOLEANS SANS BOOLEANS

- Implement a set of classes to model booleans
- The classes must support:
  - *and*, *or*, and *not* with short-circuit evaluation
  - branching
- The implementation must not use any conditional expressions or statements!

NATURALLY

- Implement a set of classes to model natural numbers
- The classes must support:
  - addition
  - comparisons (returning Boolean instances)
- The implementation must not use any existing numeric types!

LINKED LIST

- Implement a linked list with iterator
- Use polymorphic dispatch for all branching/decisions
- Don’t use Python lists