Emulating Lazy Evaluation

Henry Chen
Hackbright Academy

functions are lazy

```
>>> def fib(n):
...     if n < 2:
...         return 1
...     return n * fib(n - 1)
...
>>> fib(10) # stuff happens
3628800
```

A function does not **do anything** until called.

can i haz moar lazy?

```
>>> def fib(n):
...     if n() < 2:
...         return lambda: 1
...     return lambda: n() * fib(lambda: n() - 1)()
...
>>> fib(lambda: 10)
<function <lambda> at 0x102c22ed8>
>>> fib(lambda: 10)()
3628800
>>> fib(lambda: 1000) # no stack overflow
<function <lambda> at 0x102c70c80>
>>> fib(lambda: 1000)()
RuntimeError: maximum recursion depth exceeded
```

Nothing happens until result is called!

lazy expressions

idea: create class with lazy versions of __add__, __mul__, etc.

```
>>> from lazy import LazyBase
>>> class LazyInteger(LazyBase):
...     _type = int
...     _operators = '__add__', '__mul__'
...
>>> two = LazyInteger.lazify(2)
>>> three = LazyInteger.lazify(3)
>>> five = two + three
>>> five
<__main__.LazyInteger object at 0x102b12850>
>>> five.value # stuff happens
5
```

iteration becomes recursion

```
>>> for i in range(500):
...     two += five
...
>>> two.value # ok
2502
>>> for i in range(500):
...     two += five
...
>>> two.value
RuntimeError: maximum recursion depth exceeded
```

What is lazy good for?

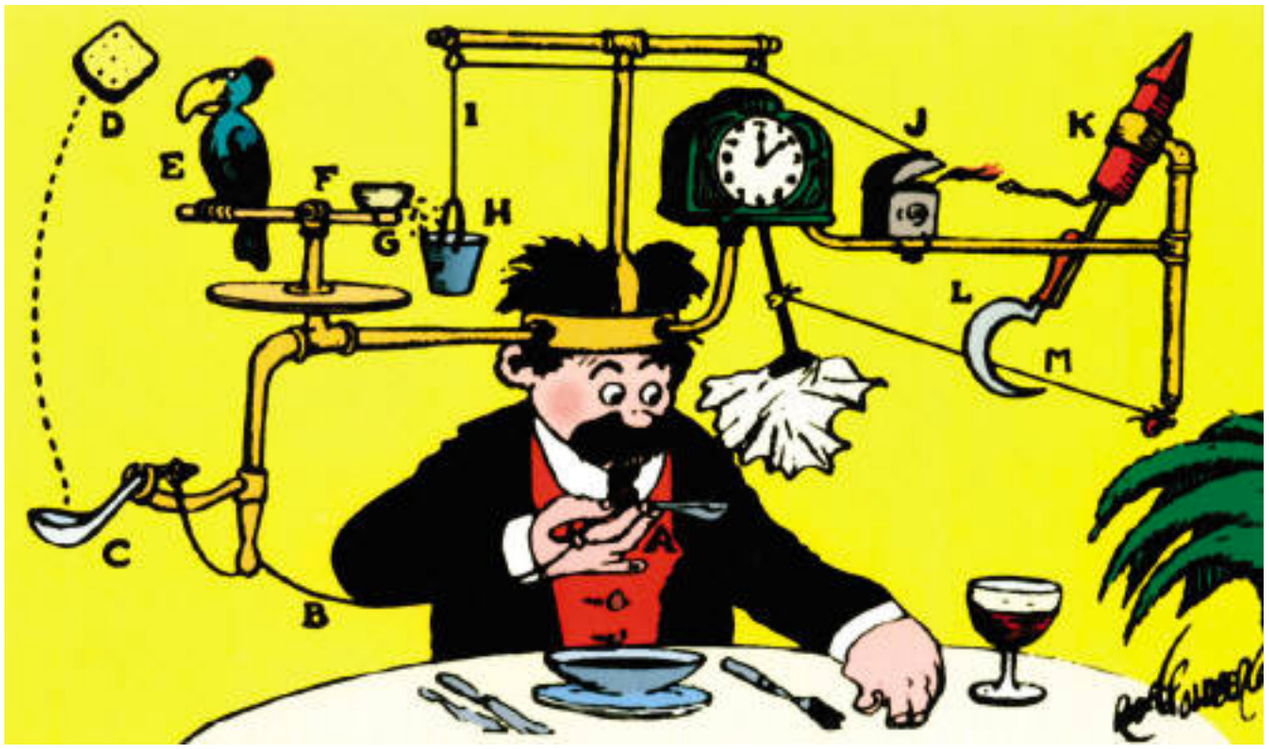
Wikipedia:

- The ability to define control flow (structures) as abstractions instead of primitives.
- The ability to define potentially infinite data structures. This allows for more straightforward implementation of some algorithms.
- Performance increases by avoiding needless calculations, and error conditions in evaluating compound expressions.

(Usually laziness is done in Python with generators.)

LazyBase seems… too easy

(Basically, a metaclass takes each method name and creates a descriptor object that wraps the corresponding eager method inside a lazy inner function….)



github.com/scotchka/lazy_arithmetic