The CS 5 Times-Picayune

Claremont Penguin Takes Olympics Gold

London: In a stunning upset, a local penguin has been retroactively awarded the gold medal in the women's 10-meter platform diving event at the Antarctic Olympics.

"All of the judges agreed that her dives were flawless," stated an unidentified official. "But our computerized scoring system calculated her totals incorrectly. A careful audit has revealed that the computer system we purchased from a Saudi Arabian manufacturer could malfunction in the presence of water. Worse, it would break down completely if exposed to fish oil. We're terribly sorry, but it could have happened to anybody."

Other observers were less forgiving. "How anyone could use a desert computer at a water-based event is beyond me," said a U.S. coach. I have no clue what they were thinking—and it's clear that they have no clue, period."

The penguin herself was apparently too excited to comment, limiting herself to a few loud squawks.

Reading: Sections 6.1–6.6

Rocket Science!

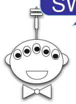
```
>>> fuelNeeded = 42/1000
>>> tank1 = 36/1000
>>> tank2 = 6/1000
>>> tank1 + tank2 >= fuelNeeded
```

True? False? Maybe? **DEMO!**

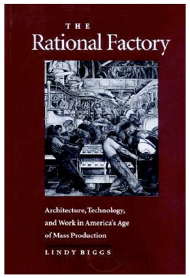
Wishful Thinking…

```
>>> from Rational import *
>>> fuelNeeded = Rational(42, 1000)
>>> tank1 = Rational(36, 1000)
>>> tank2 = Rational(6, 1000)
>>> tank1 + tank2 >= fuelNeeded
True
```

That would be so SWEET!



The `Rational` factory!

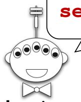


Thinking Rationally

The "constructor"

```
class Rational(object):
    def __init__(self, n, d):
        if d == 0:
            print("Invalid denominator!")
            sys.exit(1)   # import sys for this to work (ugly!)
        else:
            self.numerator = n
            self.denominator = d
```

Why is this code so **selfish**?



Nothing is returned here!

In a file called `Rational.py`

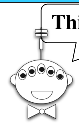
```
>>> from Rational import *
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)
>>> myNum1.numerator
? 1
>>> myNum1.denominator
? 3
>>> myNum2.numerator
? 2
```

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 2, denominator = 6

Thinking Rationally

```
from exceptions import ValueError
class Rational(object):
    """Support rational numbers."""
    def __init__(self, n, d):
        if d == 0:
            raise ValueError("Invalid denominator!")
        else:
            self.numerator = n
            self.denominator = d

    def isZero(self):
        return self.numerator == 0
```




```
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(0, 6)
>>> myNum1.isZero()
?
>>> myNum2.isZero()
?
```

F

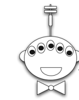
isZero(myNum1)

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 0, denominator = 6

Thinking Rationally

```
class Rational(object):
    def __init__(self, n, d):
        if d == 0:
            raise ValueError("Invalid denominator!")
        else:
            self.numerator = n
            self.denominator = d

    def isZero(self):
        return self.numerator == 0
```




```
>>> myNum1 = Rational(1, 3)
>>> myNum2 = myNum1
>>> myNum2.numerator = 42  # CHEATING!
>> myNum1
<Rational instance at 0x14ba68e87438>
```

myNum1 → numerator = 42, denominator = 3

Thinking Rationally

```
class Rational(object):
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

    def isZero(self):
        return self.numerator == 0

    def __str__(self):
        return str(self.numerator) + "/" + str(self.denominator)
```

```
>>> myNum = Rational(1, 3)
>>> myNum.__str__()
'1/3'
>>> myNum
<__main__.Rational object at 0x2b513566b7d0>
>>> print(myNum)
1/3
```

myNum → numerator = 1, denominator = 3

Thinking Rationally

```
class Rational(object):
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

    def isZero(self):
        return self.numerator == 0

    def __repr__(self):
        return "Rational(" + str(self.numerator) + \
            ", " + str(self.denominator) + ")"
```

continue line

```
>>> myNum = Rational(1, 3)
>>> myNum.__repr__()
Rational(1, 3)
>>> myNum
Rational(1, 3)
```

myNum → numerator = 1, denominator = 3

Thinking Rationally

```
class Rational(object):
   def __init__(self, n, d):
      self.numerator = n
      self.denominator = d

   def isZero(self):
      return self.numerator == 0

   # The lazy way to do both str and repr
   def __repr__ (self):
      return str(self.numerator) + "/" + str(self.denominator)
```

0x—0

```
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)
>>> print(myNum2)
2/6
>>> myNum1 == myNum2
False
```

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 2, denominator = 6

0x—8

Thinking Rationally

```
class Rational(object):
   """Support rational numbers."""
   def __init__(self, n, d):
      """Construct a Rational: no error checking, not reduced."""
      self.numerator = n
      self.denominator = d

   def isZero(self):
      return self.numerator == 0

   def __repr__(self):
      return str(self.numerator) + "/" + str(self.denominator)

   def equals(self, other):
      return self.numerator * other.denominator ==
             self.denominator * other.numerator
```

Working at cross purposes?

$\frac{1}{3} \times \frac{2}{6}$

```
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)
>>> myNum1.equals(myNum2)
True
>>> myNum2.equals(myNum2)
True
```

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 2, denominator = 6

Thinking Rationally

```
class Rational(object):
   def __init__(self, n, d):
      self.numerator = n
      self.denominator = d

   def isZero(self):
      return self.numerator == 0

   def __repr__(self):
      return str(self.numerator) + "/" + str(self.denominator)

   def __eq__(self, other):
      return self.numerator * other.denominator ==
             self.denominator * other.numerator
```

myNum1.__eq__(myNum2)

```
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)
>>> myNum1 == myNum2
True
>>> myNum2 == myNum1
True
```

This is what I would *really* like!

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 2, denominator = 6

Thinking Rational-ly

```
class Rational(object):
   def __init__(self, n, d):
      self.numerator = n
      self.denominator = d

   def add(self, other):
```

Start by assuming that the denominators are the same, but then try to do the case that they may be different!

What kind of thing is add returning?

```
>>> myNum1 = Rational(36, 1000)
>>> myNum2 = Rational(6, 1000)
>>> myNum3 = myNum1.add(myNum2)
>>> myNum3
42000/1000000
```

myNum1 → numerator = 36, denominator = 1000

myNum2 → numerator = 6, denominator = 1000

Overloaded Operator Naming

+	__add__	+	__pos__	==	__eq__
-	__sub__	-	__neg__	!=	__ne__
*	__mul__		__abs__	<=	__le__
/	__div__		__int__	>=	__ge__
//	__floordiv__		__float__	<	__lt__
%	__mod__		__complex__	>	__gt__
**	__pow__				

__radd__

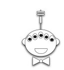
```
def _ _int_ _(self):
  return self.numerator//self.denominator
```


```
>>> myNum = Rational(9, 2)
>>> myNum.int()
Barf!
>>> int(myNum)
4
```

Putting It All Together

```
class Rational(object):
    def __init__(self, n, d):
      self.numerator = n
      self.denominator = d

    def __add__(self, other):
        newNumerator =
        newDenominator =
        return Rational(newNumerator, newDenominator)

    def __eq__(self, other):
        return ???

    def __ge__(self, other):
        return ???

    def __repr__(self):
        return str(self.numerator) + "/" + str(self.denominator)
```

```
>>> from Rational import *
>>> fuelNeeded = Rational(42, 1000)
>>> tank1 = Rational(36, 1000)
>>> tank2 = Rational(6, 1000)
>>> tank1 + tank2 >= fuelNeeded
True
```

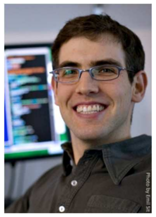


Rationals Are Now "First Class" Citizens!

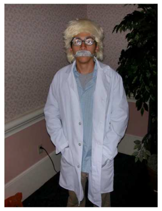
```
>>> r1 = Rational(1, 2)
>>> r2 = Rational(1, 4)
>>> r3 = Rational(1, 8)
>>> L = [r1, r2, r3]
```

True Story

Dan Aguayo



Max Krohn



Jeremy Stribling

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-tauted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in $\Omega((n + \log n))$ time [22]. In the end, we conclude.

II. ARCHITECTURE

---

k^{th} Order Markov Processes

Andrey Markov
1856-1922

Training File: "I like spam. I like toast and spam. I eat ben and jerry's ice cream too."

First order Markov Dictionary:

```
I : like, like, eat
like : spam., toast
spam. : I, I
and : spam, jerry's
MORE ENTRIES…
```

Generating "random" text:

"I like spam. I like spam."

"I eat ben and spam. I like toast and jerry's ice cream too."

---

k^{th} Order Markov Processes

Andrey Markov
1856-1922

Training File: Wikipedia essay on Huffman Compression

First order Markov sentences generated…

"Huffman was a source symbol."

"Huffman became a known as a character in a particular symbol frequencies agree with those used for each possible value of Engineering."

---

k^{th} Order Markov Processes

Andrey Markov
1856-1922

Training File: "I like spam. I like toast and spam. I eat ben and jerry's ice cream too."

First order Markov Dictionary:

```
I : like, like, eat
like : spam, toast
spam. : I, I
and : spam, jerry's
MORE ENTRIES…
```

Second order Markov Dictionary:

```
I like : spam., toast
like spam. : I
spam. I : like, eat
```

k^{th} Order Markov Processes

Training File: **Wikipedia essay on Huffman Compression**

Second order Markov sentences generated…

```
"Huffman coding is such a code
 is not produced by Huffman's algorithm."

"Huffman was able to design the most
common characters using shorter strings
of bits than are used for lossless
data compression."
```