This Week: Oops!

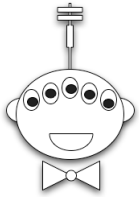
- Object-Oriented Programs (OOPs)

Oops?

Rocket Science!

```
>>> fuelNeeded = 42.0/1000
>>> tank1 = 36.0/1000
>>> tank2 = 6.0/1000
>>> tank1 + tank2 >= fuelNeeded
```

True? False? Maybe?

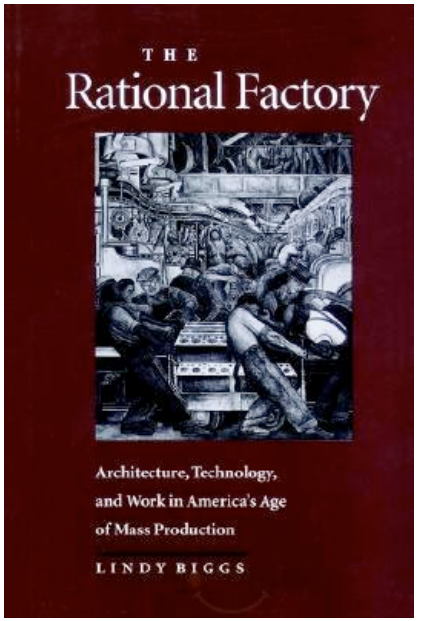


Demo

Wishful Thinking…

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

The `Rational` factory!



Thinking Rationally

```
class Rational:
    def __init__(self, n, d):
        if d == 0:
            raise ZeroDivisionError('Denom. cannot be zero.')
        else:
            self.numerator = n
            self.denominator = d
```

The "constructor"

Why is this code so selfish?

Notice that nothing is returned!

In a file called `Rational.py`

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

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 2, denominator = 6

Thinking Rationally

The "constructor"

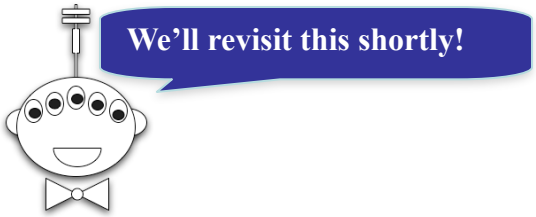
```
class Rational:
   def __init__(self, n, d):
       if d == 0:
          raise ZeroDivisionError('Denom. can't be zero.')
       else:
          self.numerator = n
          self.denominator = d
```

In a file called `Rational.py`

```
>>> from Rational import *
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(1, 3)
>>> myNum1 == myNum2
```

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 1, denominator = 3



Thinking Rationally


```
class Rational:
   def _ _init_ _(self, n, d):
      if d == 0:
         raise ...
      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()
?
```

myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 0, denominator = 6

Thinking Rationally

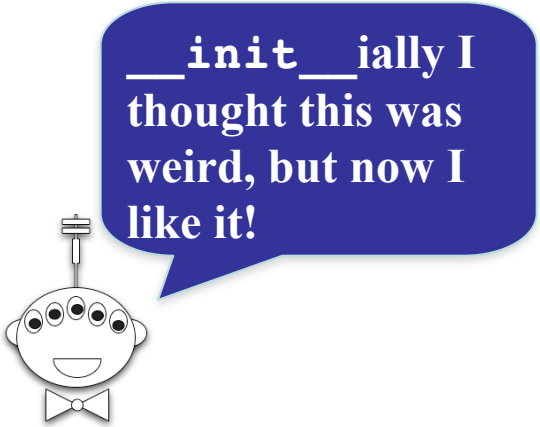
```
class Rational:
    def _ _init_ _(self, n, d):
        if d == 0:
            raise …
        else:
            self.numerator = n
            self.denominator = d

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

```
>>> myNum1 = Rational(1, 3)
>>> myNum2 = myNum1
>>> myNum2.numerator = 42
```

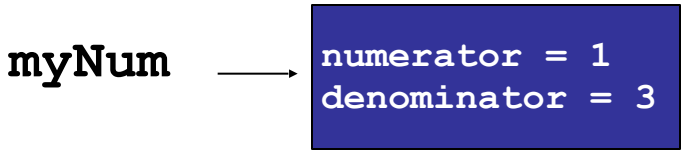
myNum1 ⟶ numerator = 1
denominator = 3

Thinking Rationally

```
class Rational:
   def _ _init_ _(self, n, d):
      self.numerator = n
      self.denominator = d

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




```
>>> myNum = Rational(1, 3)
>>> myNum
<Rational instance at 0xdb3918>
```



Thinking Rationally

```
class Rational:
   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_ _()
?
>>> myNum
?
```

myNum → numerator = 1
denominator = 3

Thinking Rationally

```
class Rational:
   def _ _init_ _(self, n, d):
      self.numerator = n
      self.denominator = d

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

   def _ _str_ _ (self):
      return "Numerator " + str(self.numerator) + \
             " and Denominator " + str(self.denominator)
```

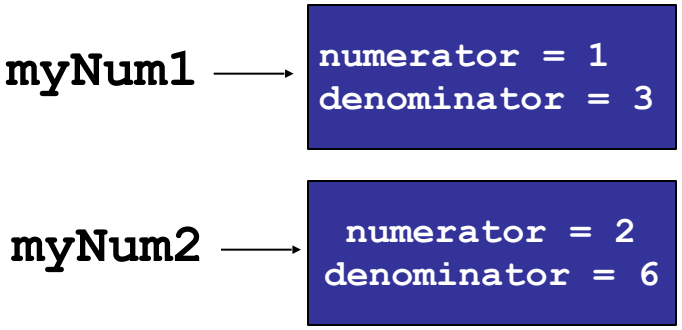
```
>>> myNum = Rational(1, 3)
>>> myNum
Numerator 1 and Denominator 3
```

myNum → numerator = 1
denominator = 3

Thinking Rationally

```
class Rational:
   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)
```

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



Thinking Rationally

```
class Rational:
    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)

    def equals(self, other):
        ???
```

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
```

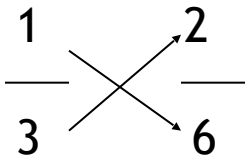
myNum1 → numerator = 1, denominator = 3

myNum2 → numerator = 2, denominator = 6

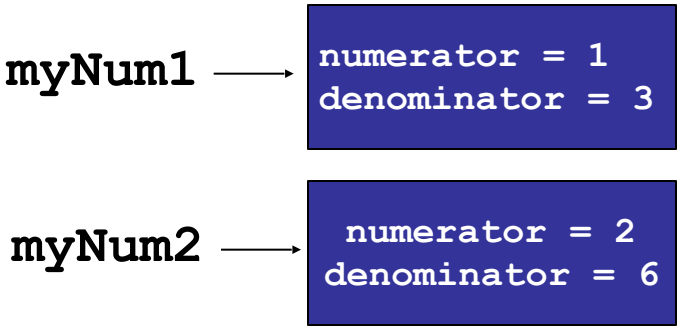
Thinking Rationally

```
class Rational:
    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)
```

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



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



Thinking Rationally

```
class Rational:
    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)

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

```
>>> 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 Rationally

```
class Rational:
    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
42/1000
```

myNum1 → numerator = 36, denominator = 1000

myNum2 → numerator = 6, denominator = 1000

Thinking Rationally

```
class Rational:
    def _ _init_ _(self, n, d):
      self.numerator = n
      self.denominator = d
```

```
  def add(self, other):
    newNumerator = self.numerator + other.numerator
    return Rational(newNumerator, self.Denominator)
```

SAME DENOMINATORS

What kind of thing is add returning?

```
>>> myNum1 = Rational(36, 1000)
>>> myNum2 = Rational(6, 1000)
>>> myNum3 = myNum1.add(myNum2)
>>> myNum3
42/1000
```

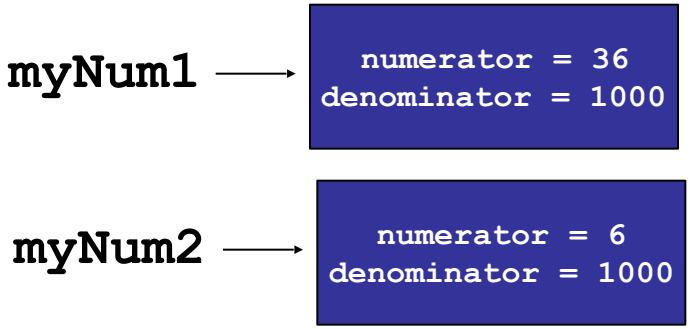
myNum1 → numerator = 36, denominator = 1000

myNum2 → numerator = 6, denominator = 1000

Thinking Rationally

```
class Rational:
    def _ _init_ _(self, n, d):
      self.numerator = n
      self.denominator = d


  def add(self, other):
     newDenominator = self.denominator*other.denominator
     newNumerator = self.numerator*other.denominator + self.denominator*other.numerator
     return Rational(newNumerator, newDenominator)
```

```
>>> myNum1 = Rational(36, 1000)
>>> myNum2 = Rational(6, 1000)
>>> myNum3 = myNum1.add(myNum2)
>>> myNum3
42/1000
```



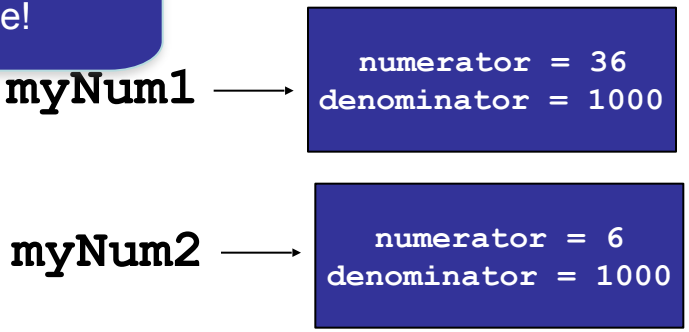
Thinking Rationally

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

```
    def __add__(self, other):
        newDenominator = self.denominator*other.denominator
        newNumerator = self.numerator*other.denominator + self.denominator*other.numerator
        return Rational(newNumerator, newDenominator)
```

```
>>> myNum1 = Rational(36, 1000)
>>> myNum2 = Rational(6, 1000)
>>> myNum3 = myNum1 + myNum2
>>> myNum3
42/1000
```

This is what I would *really, really* like!



Overloaded Operator Naming

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

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

Very _ _int_ _eresting!

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

Putting it all Together

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

    def __add__(self, other):
        newNumerator = self.numerator*other.denominator + self.denominator*other.numerator
        newDenominator = self.denominator*other.denominator
        return Rational(newNumerator, newDenominator)

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

    def __ge__(self, other):
        return self.numerator*other.denominator >= self.denominator*other.numerator

    def __str__(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
```

Mission accomplished!

Rationals are now “first class” citizens!

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