Object Oriented Programs (OOPs)

Oops?
Rocket Science!

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

True? False? Maybe?

What’s the right ants-er?
Wishful Thinking...

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

We need an ant-i-dote for this problem!

The Rational factory!
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

>>> from Rational import *
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)

myNum1
myNum2

1
3
2

numerator = 1
denominator = 3
numerator = 2
denominator = 6

Notice that nothing is returned!

In a file called Rational.py

Why is this code so selfish?
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

>>> from Rational import *
>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(1, 3)
>>> myNum1 == myNum2
Rational numbers go back to the days of anti-iquity!

In a file called Rational.py
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

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

>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(0, 6)

>>> myNum1.isZero()
False

>>> myNum2.isZero()
True

This is "self-explanatory!"

This is so class-y!

Hey, turtle, your bad puns are ant-agonizing me!
class Rational:
    def __init__(self, n, d):
        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
import sys

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

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

myNum = Rational(1, 3)
print(myNum)

# Output:
# <Rational instance at 0xdb3918>
class Rational:
    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)

myNum = Rational(1, 3)
myNum
1/3
myNum.
myNum.
myNum
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

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

    def __repr__(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
class Rational:
    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)

>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)
>>> myNum1 == myNum2
False
class Rational:
    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 equals(self, other):
        return self.numerator * other.denominator ==
        other.numerator * self.denominator

numerator = 1
 denominator = 3
myNum1 = Rational(1, 3)
myNum2 = Rational(2, 6)
>>> myNum1.equals(myNum2)  # myNum1 →
True
>>> myNum2.equals(myNum2)  # myNum2 →
True

Working at cross purposes?
class Rational:
    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 ==
               other.numerator * self.denominator

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

In your notes

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

What kind of thing is add returning?
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)

numerator = 36
denominator = 1000
myNum1 = Rational(36, 1000)
myNum2 = Rational(6, 1000)
myNum3 = myNum1.add(myNum2)
myNum3
42/1000
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)

numerator = 36
 denominator = 1000
myNum1 = Rational(36, 1000)
myNum2 = Rational(6, 1000)
myNum3 = myNum1.add(myNum2)
myNum3
42/1000
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__

That’s the ant-ire list!
from Rational import *

def initely():
    r1 = Rational(1, 2)
    r2 = Rational(5, 42)
    r3 = Rational(7, 3)
    myList = [r1, r2, r3]
    r4 = Rational(0, 1)
    for r in myList:
        r4 = r4 + r
    return r