Today: Object Oriented Programs (OOPs)

Oops?

CS 5 Green

Learning Goals

- Explain when classes are useful
- Implement classes
- Define key terms
  - constructor
  - `self`
  - attributes
  - methods

Surgeon General's Warning

This lecture contains some syntactic details. Don't memorize them!

Concentrate on the big ideas and refer to these slides later for syntax details.

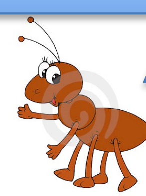
Rocket Science!

The CS 5 Gold/Black "textbook" is now linked from the HW 10 entry on the course website. Read Chapter 6: "OOPs! Object-Oriented Programming"

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

True? False? Maybe?

What's the right *ants*-er?

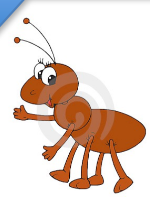


Demo

Wishful Thinking…

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

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



The `Rational` factory!

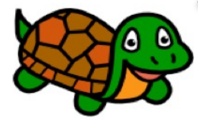
Thinking Rationally

In a file called `Rational.py`

The "constructor"

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

Why is this code so **self**ish?

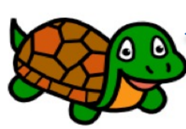


Notice that nothing is **return**ed!

```
>>> from Rational import *
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(2, 6)
>>> my_num1.numerator
?
>>> my_num1.denominator
?
>>> my_num2.numerator
?
```

my_num1 → numerator = 1, denominator = 3

my_num2 → numerator = 2, denominator = 6



This "dot" notation is vaguely familiar!

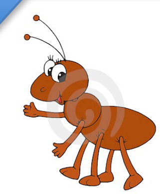
Thinking Rationally

In a file called `Rational.py`

The "constructor"

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

Rational numbers go back to the days of ant-iquity!



```
>>> from Rational import *
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(1, 3)
>>> my_num1 == my_num2
```

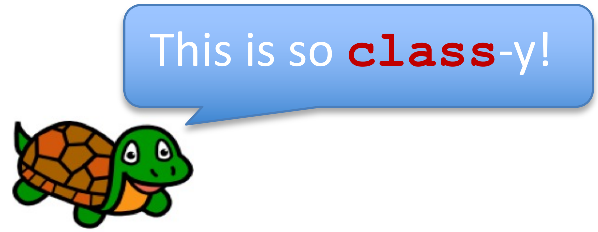
my_num1 → numerator = 1, denominator = 3

my_num2 → numerator = 1, denominator = 3

Demo

Thinking Rationally

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

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




```
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(0, 6)
>>> my_num1.is_zero()
False
>>> my_num2.is_zero()
True
```

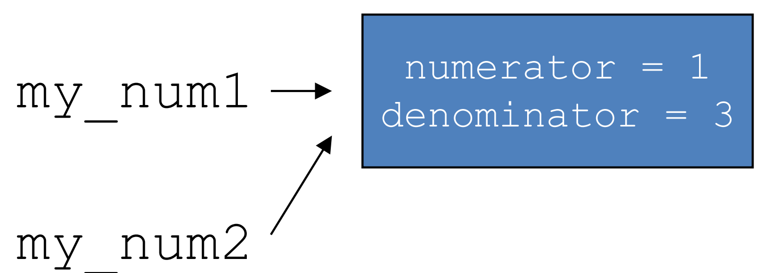
my_num1 → numerator = 1, denominator = 3

my_num2 → numerator = 0, denominator = 6

Thinking Rationally

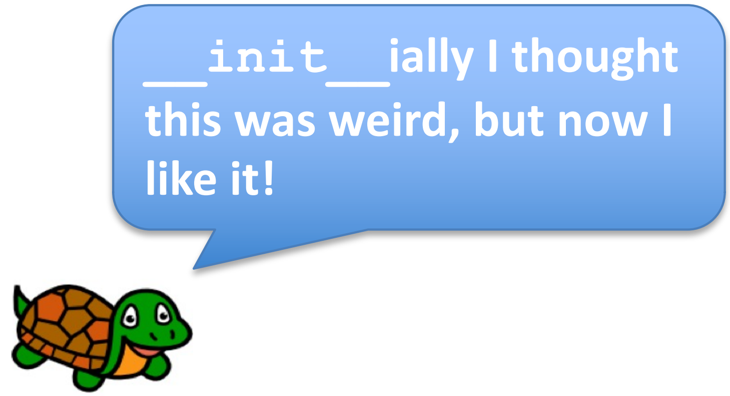
```
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

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

```
>>> my_num1 = Rational(1, 3)
>>> my_num2 = my_num1
>>> my_num2.numerator = 0
>>> my_num2.is_zero()
True
>>> my_num1.is_zero()
???
```

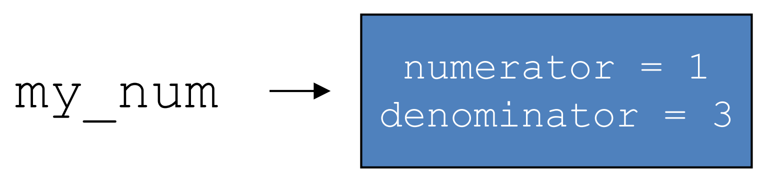


Thinking Rationally

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

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



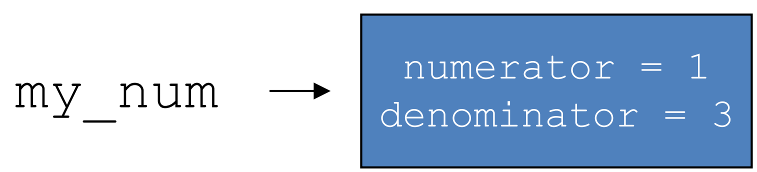

```
>>> my_num = Rational(1, 3)
>>> my_num
<Rational instance at 0xdb3918>
```



Thinking Rationally

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

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

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

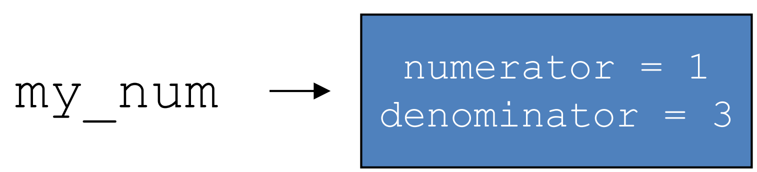
```
>>> my_num = Rational(1, 3)
>>> my_num.__repr__()
'1/3'
>>> my_num
1/3
```



Thinking Rationally

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

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

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

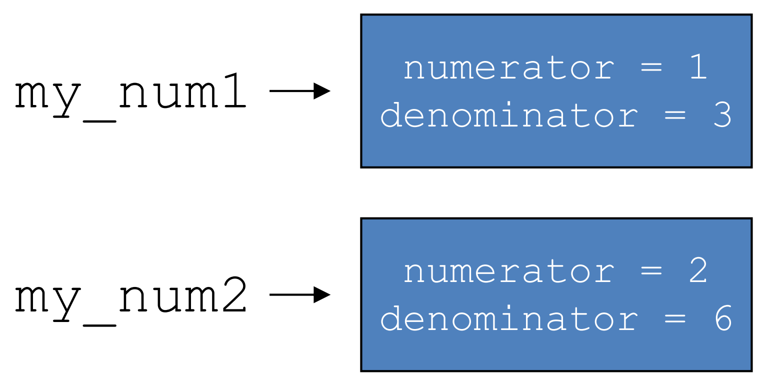
```
>>> my_num = Rational(1, 3)
>>> my_num
Numerator 1 and Denominator 3
```



Thinking Rationally

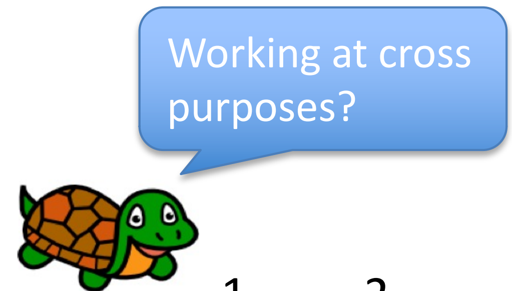
```
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

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

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

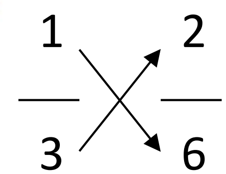
```
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(2, 6)
>>> my_num1 == my_num2
False
```



Thinking Rationally

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

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

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

    def equals(self, other):
```






```python
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(2, 6)
>>> my_num1.equals(my_num2)
True
>>> my_num2.equals(my_num1)
True
```

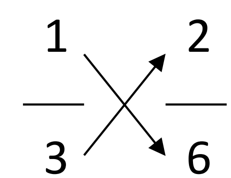
my_num1 → numerator = 1, denominator = 3

my_num2 → numerator = 2, denominator = 6

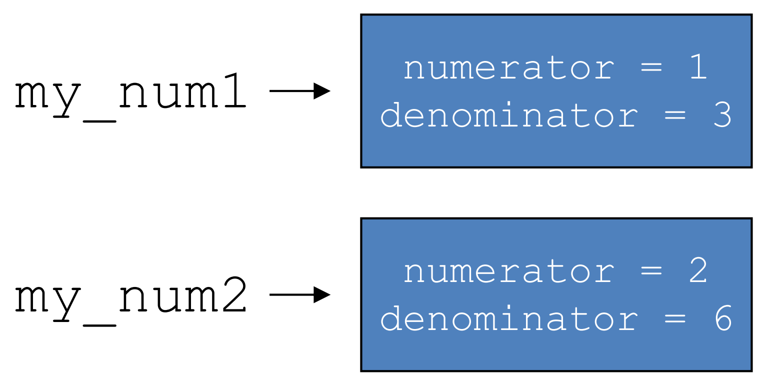
Thinking Rationally

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

    def is_zero(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
```

This just means the line is wrapping…

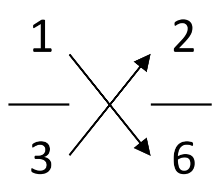



```
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(2, 6)
>>> my_num1.equals(my_num2)
True
>>> my_num2.equals(my_num1)
True
```

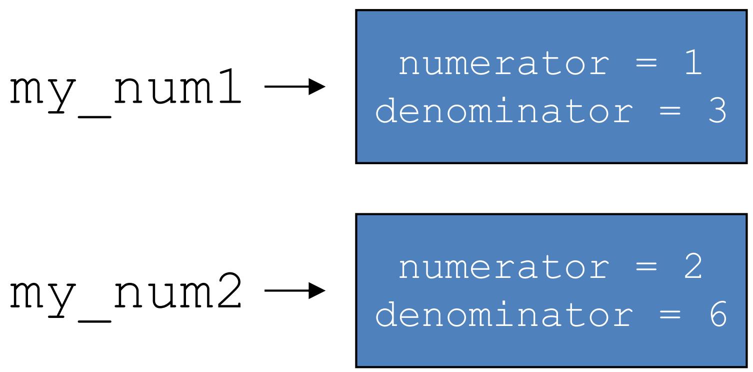


Thinking Rationally

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

    def is_zero(self):
        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
```




```
>>> my_num1 = Rational(1, 3)
>>> my_num2 = Rational(2, 6)
>>> my_num1.__eq__(my_num2)
True
>>> my_num1 == my_num2
True
>>> my_num2 == my_num1
True
```

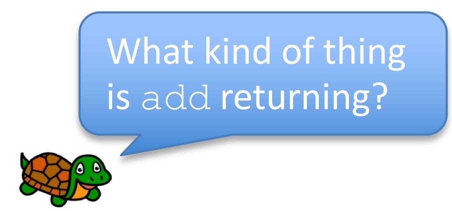


Thinking Rationally

Worksheet Q

```
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!



```
>>> my_num1 = Rational(36, 1000)
>>> my_num2 = Rational(6, 1000)
>>> my_num3 = my_num1.__add__(my_num2)
>>> my_num3
42/1000
>>> my_num1 + my_num2
```

my_num1 → [numerator = 36, denominator = 1000]

my_num2 → [numerator = 6, denominator = 1000]

Thinking Rationally

Worksheet

S

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

    def __add__(self, other):
        num = self.numerator + other.numerator
        den = self.denominator  # assuming same denominators!
        sum = Rational(num, den)
        return sum
```

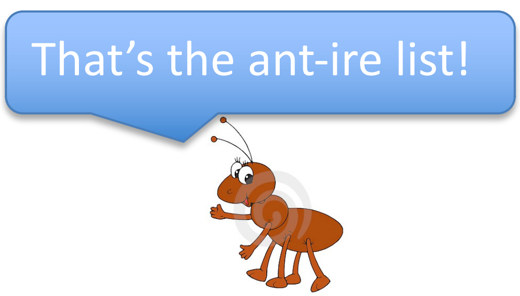
```
>>> my_num1 = Rational(36, 1000)
>>> my_num2 = Rational(6, 1000)
>>> my_num3 = my_num1.__add__(my_num2)
>>> my_num3
42/1000
>>> my_num1 + my_num2
```

my_num1 → numerator = 36, denominator = 1000

my_num2 → 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__				



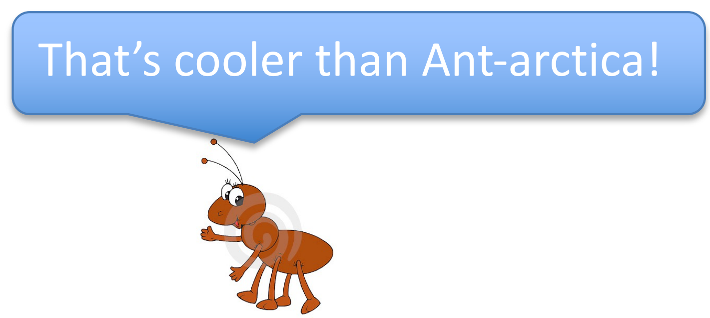
Putting It All Together

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

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

    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 __add__(self, other):
        num = self.numerator*other.denominator + self.denominator*other.numerator
        den = self.denominator*other.denominator
        return Rational(num, den)
```

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

Rationals are now “first class” citizens

```
from Rational import *

def initely():
    r1 = Rational(1, 2)
    r2 = Rational(21, 42)
    r3 = Rational(1,42)
    my_list = [r1, r2, r3]
    r4 = Rational(0, 1)
    for r in my_list:
        r4 = r4 + r
    return r
```



What's the Point?

Q

```
>>> p1 = Point(1.0, 2.0)
>>> p2 = Point(1.0, 2.0)
>>> p1
(1.0,2.0)
>>> p1 == p2
True
```

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

    def __eq__(self, other):
```

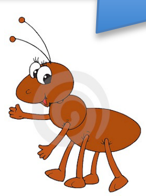
Without this example, the lecture would be Point-less!

What's the Point?

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y
```

```
>>> p1 = Point(1.0, 2.0)
>>> p2 = Point(1.0, 2.0)
>>> p1
(1.0,2.0)
>>> p1 == p2
True
```

Without this example, the lecture would be Point-less!



Thinking Linearly

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y

    def __eq__(self, other):
        return self.x == other.x and self.y == othe



class Line:
    def __init__(self, point1, point2):
        self.slope = (point2.y - point1.y) / (point2.x - point1.x)
        self.yint = point1.y - point1.x * self.slope

    def __repr__(self):

    def __eq__(self, other):

```

```
>>> p1 = Point(1.0, 2.0)
>>> p2 = Point(2.0, 3.0)
>>> l1 = Line(p1,p2)
>>> l1
y = 1.0 x + 1.0
>>> p3 = Point(3.0, 4.0)
>>> p4 = Point(42.0, 43.0)
>>> l2 = Line(p3, p4)
>>> l1 == l2
True
```

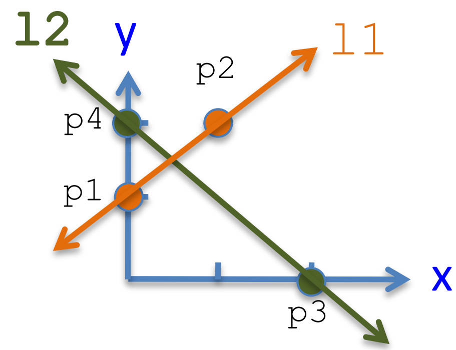
Thinking Linearly

S

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y

    def __eq__(self, other):
        return self.x == other.x and self.y == othe


class Line:
    def __init__(self, point1, point2):
        self.slope = (point2.y - point1.y) / (point2.x - point1.x)
        self.yint = point1.y - point1.x * self.slope

    def __repr__(self):
        return "y = " + str(self.slope) + " x + " + str(self.yint)

    def __eq__(self, other):
        return self.slope == other.slope and self.yint == other.yint
```

```
>>> p1 = Point(1.0, 2.0)
>>> p2 = Point(2.0, 3.0)
>>> l1 = Line(p1,p2)
>>> l1
y = 1.0 x + 1.0
>>> p3 = Point(3.0, 4.0)
>>> p4 = Point(42.0, 43.0)
>>> l2 = Line(p3, p4)
>>> l1 == l2
True
```

```
>>> from Geometry import *
>>> p1 = Point(0, 1)
>>> p2 = Point(1, 2)
>>> l1 = Line(p1, p2)
>>> p3 = Point(2, 0)
>>> p4 = Point(0, 2)
>>> l2 = Line(p3, p4)
>>> l1.parallel(l2)
False
>>> l1.intersection(l2)
(0.5,1.5)
```

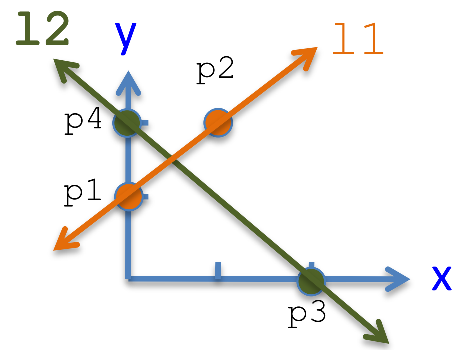


Q

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y
```

```
class Line:
    def __init__(self, point1, point2):
        self.slope = (point2.y - point1.y) / (point2.x - point1.x)
        self.yint = point1.y - point1.x * self.slope

    def __repr__(self):
        ...

    def __eq__(self, other):
        ...

    def parallel(self, other):

    def intersection(self, other):

```

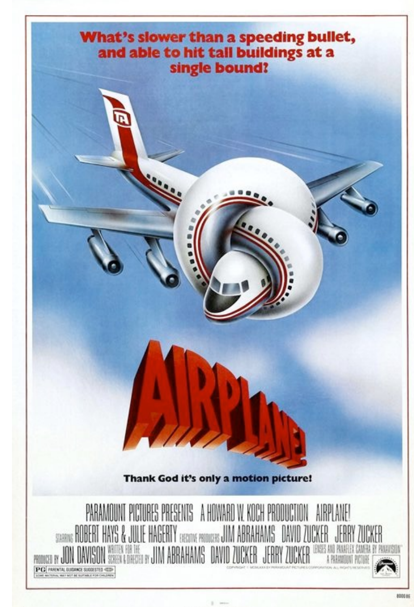
```
>>> from Geometry import *
>>> p1 = Point(0, 1)
>>> p2 = Point(1, 2)
>>> l1 = Line(p1, p2)
>>> p3 = Point(2, 0)
>>> p4 = Point(0, 2)
>>> l2 = Line(p3, p4)
>>> l1.parallel(l2)
False
>>> l1.intersection(l2)
(0.5,1.5)
```




```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y
```

```
class Line:
    def __init__(self, point1, point2):
        self.slope = (point2.y - point1.y) / (point2.x - point1.x)
        self.yint = point1.y - point1.x * self.slope

    def __repr__(self):
        return "y = " + str(self.slope) + " x + " + str(self.yint)

    def __eq__(self, other):
        return self.slope == other.slope and self.yint == other.yint

    def parallel(self, other):
        return self.slope == other.slope

    def intersection(self, other):
        if self.parallel(other): return None
        else:
            x = (self.yint - other.yint) / (other.slope - self.slope)
            y = self.slope * x + self.yint
            return Point(x, y)
```

Vector, Victor!



In a file called `Vector.py`

```
class Vector:
    def __init__(self, x, y):
    def __repr__(self):
    def __add__(self, other):
    def __sub__(self, other):
    def magnitude(self):
    def normalize(self):
```

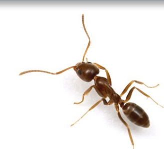
```
>>> victor = Vector(1, 1)
>>> victor
(1, 1)
>>> roger = Vector(0, 2)
>>> roger
(0, 2)
```

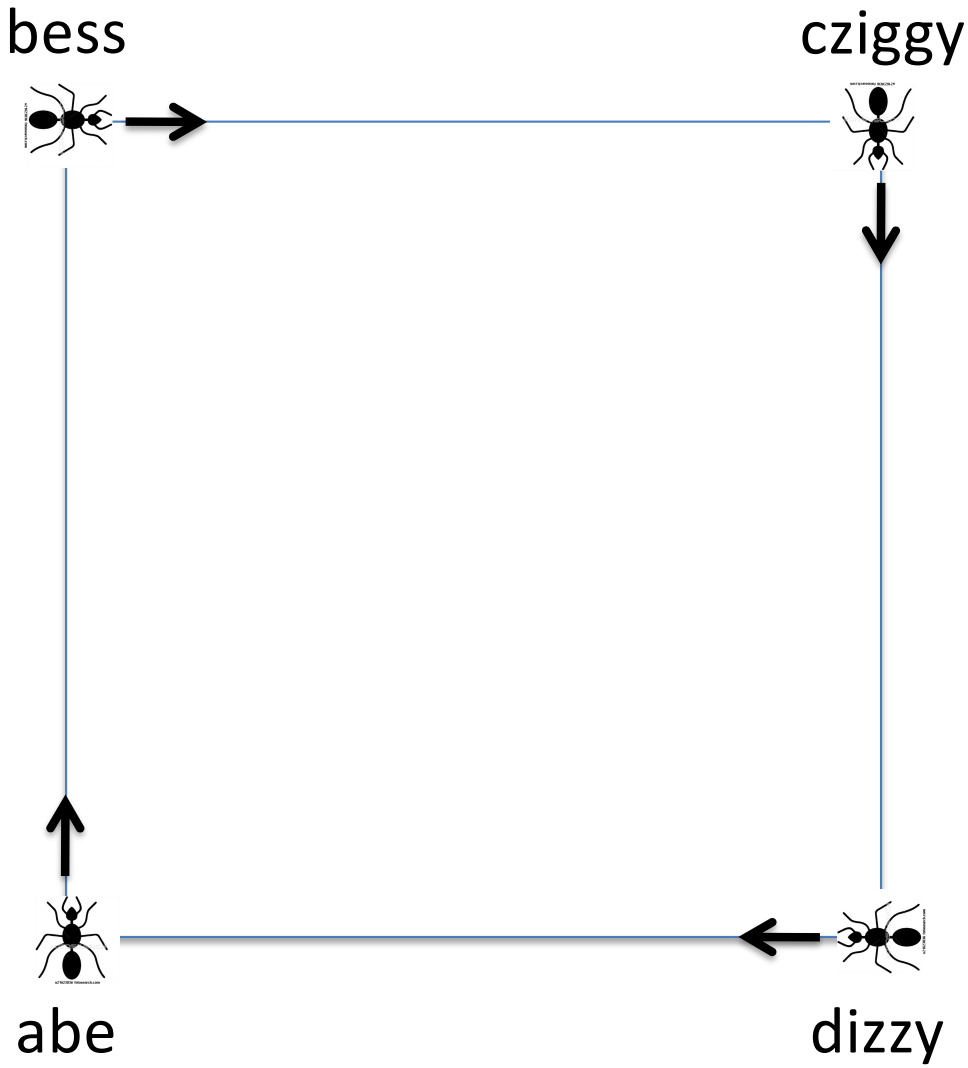
```
>>> A = victor + roger
>>> A
(1, 3)
>>> victor.magnitude()
1.4142135
```

An Ant Class

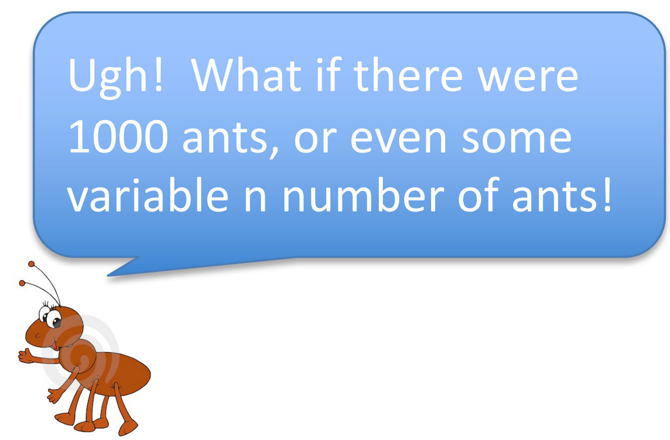
```
from Vector import *

class Ant:
    def __init__(self, pos):
        self.position = pos

    def moveTowards(self, other):
```


```
>>> abes_position = Vector(0, 0)
>>> abe = Ant(abes_position)
>>> bess = Ant(Vector(100, 0))
>>> abe.moveTowards(bess)
```



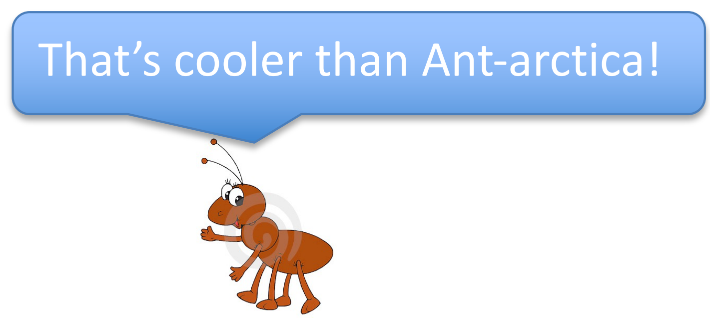


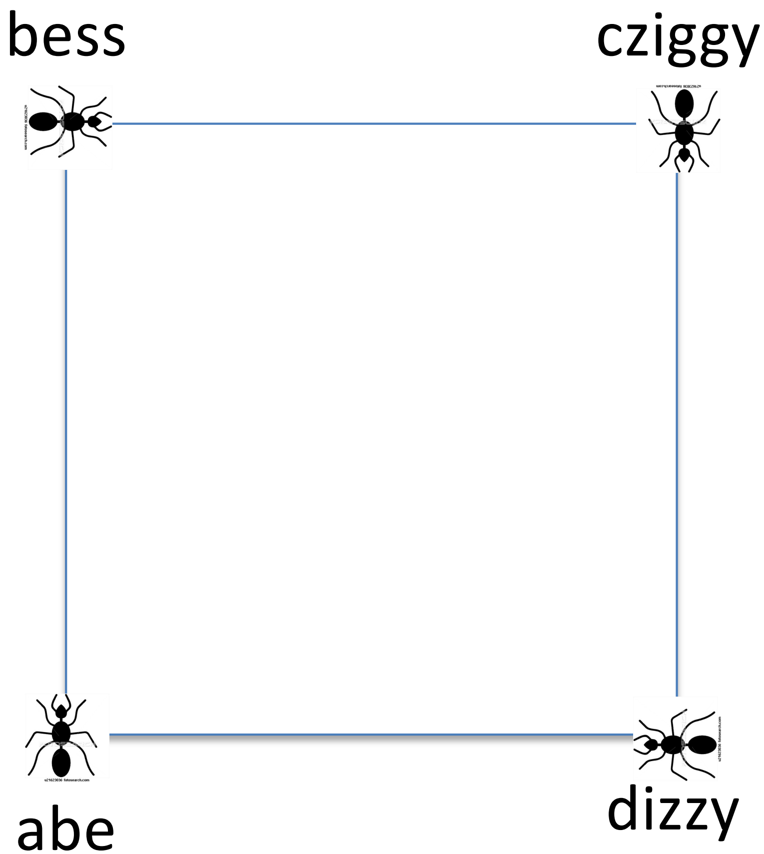

```
abe = Ant(Vector(0, 0))
bess = Ant(Vector(0, 100))
cziggy = Ant(Vector(100, 100))
dizzy = Ant(Vector(100, 0))

while True:
    abe.moveTowards(bess)
    bess.moveTowards(cziggy)
    cziggy.moveTowards(dizzy)
    dizzy.moveTowards(abe)
```



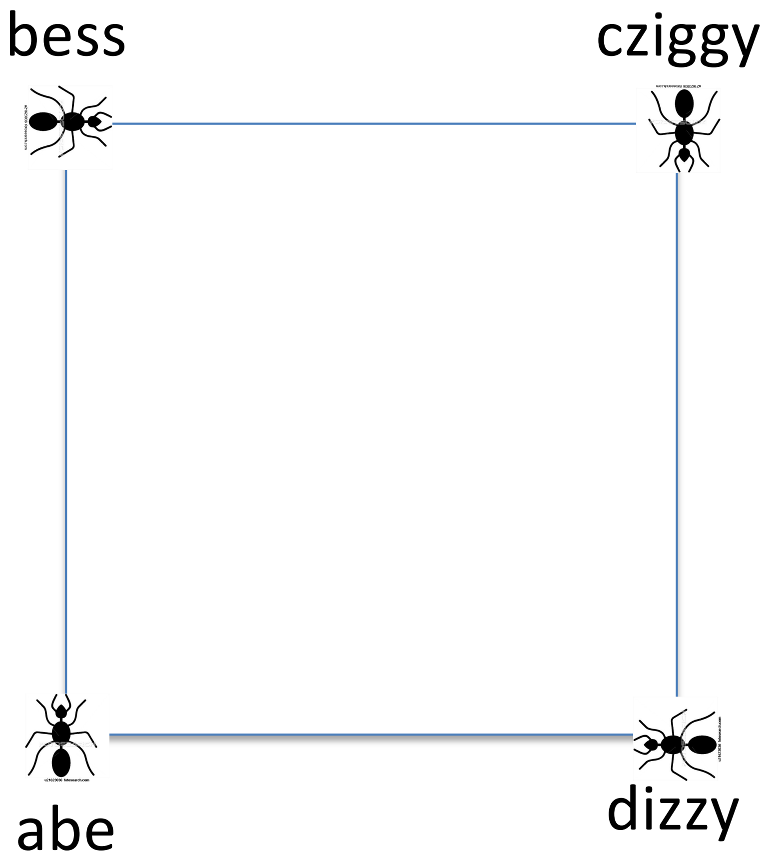
Remember this…

```
from Rational import *

def initely():
    r1 = Rational(1, 2)
    r2 = Rational(21, 42)
    r3 = Rational(1,42)
    my_list = [r1, r2, r3]
    r4 = Rational(0, 1)
    for r in my_list:
        r4 = r4 + r
    return r
```






```
abe = Ant(Vector(0, 0))
bess = Ant(Vector(0, 100))
cziggy = Ant(Vector(100, 100))
dizzy = Ant(Vector(100, 0))
ants = [abe, bess, cziggy, dizzy]

while True:
    for i in range(len(ants)):
```




```
abe = Ant(Vector(0, 0))
bess = Ant(Vector(0, 100))
cziggy = Ant(Vector(100, 100))
dizzy = Ant(Vector(100, 0))
ants = [abe, bess, cziggy, dizzy]

while True:
    for i in range(len(ants)):
        ants[i].moveTowards(ants[(i+1) % len(ants)])
```

In Python, everything is a class!

```
>>>  x = 5
>>>  y = 37
>>>  x + y
42
```

```
>>>  x = int("5")
>>>  y = int("37")
>>>  x.__add__(37)
42
>>>  x + y
42
```

```
class int:
    def __init__(self, str):
```

In Python, everything is a class!

```
class list:
    def __init__(self):
    def append(self, item):
    def __repr__(self):
    def __getitem__(self, index):
    def __setitem__(self, index, value):
```

```
>>>  x = []
>>>  x.append(42)
>>>  x
[42]
>>> x[0]
42
>>> x[0] = 67
```

```
>>>  x = list()
>>>  x.append(42)
>>>  x.__repr__()
[42]
>>> x.__getitem__(0)
42
>>> x.__setitem__(0, 67)
```