Today: Object Oriented Programs (OOPs)
A little reflection: the semester in perspective

- Basic elements of programming:
  - Functions
  - Loops
  - Conditionals
- Solving ‘hard’ problems with recursion
- A look under the hood: how recursion works at the level of the machine
Next week…

• No class on Tuesday
• Office hours held as normal (Mon 3-5, Tues 4:15-5:15)
• Homework due as normal at 11:59 pm Tuesday
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!

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

True?  False?  Maybe?

What’s the right ants-er?

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”
Wishful Thinking...

>>> 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!
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.numerator
...    numerator = 1
...    denominator = 3
>>> myNum1.denominator
...    numerator = 1
...    denominator = 3
>>> myNum2.numerator
...    numerator = 2
...    denominator = 6
>>> myNum2.denominator
...    numerator = 2
...    denominator = 6

This “dot” notation is vaguely familiar!
Thinking Rationally

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

Rational numbers go back to the days of ant-i-quity!

In a file called `Rational.py`

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

Demo
```python
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
```

Hey, turtle, your bad puns are ant-agonizing me!

This is so class-y!
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 = 0
>>> myNum2.isZero()
True
>>> myNum1.isZero()
???
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>
```python
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.__repr__()
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)

myNum = Rational(1, 3)
myNum.__repr__()  # 1/3
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 "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)

numerator = 1
denominator = 3
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):

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

numerator = 1
denominator = 3
myNum1 = Rational(1, 3)
myNum2 = Rational(2, 6)
myNum1.equals(myNum2)  # True
myNum2.equals(myNum2)  # True
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 == self.denominator * other.numerator

>>> myNum1 = Rational(1, 3)
>>> myNum2 = Rational(2, 6)
>>> myNum1.__eq__(myNum2)
True
>>> myNum1 == myNum2
True
>>> myNum2 == myNum1
True
class Rational:
    def __init__(self, n, d):
        self.numerator = n
        self.denominator = d

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

Worksheet

>>> myNum1 = Rational(36, 1000)
>>> myNum2 = Rational(6, 1000)
>>> myNum3 = myNum1.__add__(myNum2)
>>> myNum3
42/1000
>>> myNum1 + myNum2
myNum1 ➞
    numerator = 36
denominator = 1000

myNum2 ➞
    numerator = 6
denominator = 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):
        num = self.numerator + other.numerator
        den = self.denominator  # assuming same denominators!
        sum = Rational(num, den)
        return sum

>>> myNum1 = Rational(36, 1000)
>>> myNum2 = Rational(6, 1000)
>>> myNum3 = myNum1.__add__(myNum2)
>>> myNum3
42/1000
>>> myNum1 + myNum2
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!
        return Rational(num, den)

numerator = 36
denominator = 1000
myNum1 = Rational(36, 1000)
myNum2 = Rational(6, 1000)
myNum3 = myNum1.__add__(myNum2)
myNum3

myNum1 + myNum2
Overloaded Operator Naming

+ __add__
- __sub__
* __mul__
/ __div__
// __floordiv__
% __mod__
** __pow__

+ __pos__
- __neg__
__abs__
__int__
__float__
__complex__

== __eq__
!= __ne__
<= __le__
>= __ge__
< __lt__
> __gt__

That’s the ant-ire list!
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 __repr__(self):
        return str(self.numerator) + "/" + str(self.denominator)

Mission accomplished!

>>> from Rational import * *(necessary?)
>>> fuelNeeded = Rational(42, 1000)
>>> tank1 = Rational(36, 1000)
>>> tank2 = Rational(6, 1000)
>>> tank1 + tank2 >= fuelNeeded
True
Rationals are now “first class” citizens

from Rational import *

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

This is beyond awesome!

That’s cooler than Ant-arctica!
What’s the Point?

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

    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, InputX, InputY):
        self.x = InputX
        self.y = InputY

    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
???
>>> P1 == P2
True
```

Without this example, the lecture would be Point-less!
class Point:
    def __init__(self, InputX, InputY):
        self.x = InputX
        self.y = InputY

    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.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)

    def __repr__(self):
        return "y = \frac{\text{Point1.y} - \text{Point2.y}}{\text{Point1.x} - \text{Point2.x}} \times x + \text{Point1.y} - \frac{\text{Point1.y} - \text{Point2.y}}{\text{Point2.x} - \text{Point1.x}}\text{Point2.x}"

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

>>> 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

class Point:
    def __init__(self, InputX, InputY):
        self.x = InputX
        self.y = InputY
    
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.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)
    
def __repr__(self):
        return "y = " + str(self.slope) + " x + " + str(self.yintercept)
    
def __eq__(self, other):
        return self.slope == other.slope and self.yintercept == other.yintercept

>>> 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
```python
>>> from Point 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)
```

```python
class Point:
    def __init__(self, InputX, InputY):
        self.x = 1.0*InputX
        self.y = 1.0*InputY

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

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

class Line:
    def __init__(self, Point1, Point2):
        self.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)

    def __parallel__(self, other):
        return self.slope == other.slope and self.yintercept == other.yintercept

    def intersection(self, other):
```

```python
>>> from Point 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, InputX, InputY):
        self.x = 1.0*InputX
        self.y = 1.0*InputY
    def __repr__(self):
        return "y = \" + str(self.slope) + ",\" + str(self.yintercept) + ")"
    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

class Line:
    def __init__(self, Point1, Point2):
        self.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)
    def __repr__(self):
        return "y = " + str(self.slope) + " x + " + str(self.yintercept)
    def __eq__(self, other):
        return self.slope == other.slope and self.yintercept == other.yintercept
    def parallel(self, other):
        return self.slope == other.slope
    def intersection(self, other):
        if self.parallel(other):
            return None
        else:
            x = (self.yintercept - other.yintercept)/(other.slope - self.slope)
            y = self.slope * x + self.yintercept
            return Point(x, y)
```
class Vector:

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

In a file called Vector.py

>>> 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

```python
from Vector import *

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

    def moveTowards(self, other):
        AbesPosition = Vector(0, 0)
        Abe = Ant(AbesPosition)
        Bess = Ant(Vector(100, 0))
        Abe.moveTowards(Bess)

>>> AbesPosition = Vector(0, 0)
>>> Abe = Ant(AbesPosition)
>>> Bess = Ant(Vector(100, 0))
>>> Abe.moveTowards(Bess)
```

Hey, draw me a picture of Abe and Bess!

(100, 0)!! That’s practically in the Dutch Ant-illes!

I’m feeling strong ant-ipathy for ant puns!
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)

Ugh! What if there were 1000 ants, or even some variable n number of ants!
from Rational import *

def indefinitely():
    r1 = Rational(1, 2)
    r2 = Rational(21, 42)
    r3 = Rational(1, 4)
    myList = [r1, r2, r3]
    r4 = Rational(0, 1)
    for r in myList:
        r4 = r4 + r
    return r

This is beyond awesome!

That’s cooler than Ant-arctica!
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!

```python
class int:
    def __init__(self, str):
```
In Python, everything is a class!

class list:
    def __init__(self):
    def __repr__(self):
    def __append__(self, item):
    def __add__(self, other):
    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)