OOPs! (Object-Ori*ant*ed Programs)

```
>>> assigned = Date(11, 13, 2013)
>>> due = Date(11, 21, 2013)
>>> due – assigned
8
>> if due > assigned:
       print “Go watch a movie!”
```

One implem*ant*ation

```
class Date:
  def __init__(self, m, d, y):
     '''constructor for instances of the class'''
    self.month = m
    self.day = d
    self.year = y
```

```
>>> d = Date(1, 21, 1969)
```

At a distant university, simulated ant dating explains the pun of the day.
At Stevens, ants are here as reminder of the TEST in the dist*ant* future.

Another implem*ant*ation…

```
class Date:
  def __init__(self, m, d, y):
    self.daysSince1900 = …
```

```
>>> d = Date(1, 21, 1969)
```

Why would any sane person w***ant*** to store the date as the number of days since January 1, 1900?

Getters and Setters

```
class Date:
  def __init__(self, m, d, y):
    self._daysSince1900 = …

  def setDay(self, d):
    if d <= 0 or d > 31:
       …
    else:
       self._daysSince1900 = …
```

Expert tip: Python has a kind of 'decorator' called '@property' for making getters and setters. You are not responsible to know about this.

```
>>> d = Date(1, 21, 1969)
>>> d.setDay(28)     # SETTER
>>> x = d.getDay()   # GETTER
```

Date “Abstraction”

```
Date
  __init__(self, month, day, year)
 setDay(self, day)
 setMonth(self, month)
 setYear(self, year)
 getDay(self)
 getMonth(self)
 getYear(self)
 ==, >, <, >=, <=, +, -
```

An Import  Point

```
import turtle
import math
import Date

turtle.forward(100)
print math.cos(math.pi)
today = Date.Date(11, 9, 2011)
```

```
from turtle import *
from math import *
from Date import *

forward(100)
print cos(pi)
today = Date(11, 9, 2011)
```

Advantages? Disadvantages?

Another Point...

```
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
???
```

What's the Point?

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
```

```
>>> P1 = Point(1.0, 2.0)
>>> P2 = Point(2.0, 3.0)
>>> L1 = Line(P1,P2)
>>> str(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
```

```
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):

    def __eq__(self, other):
```

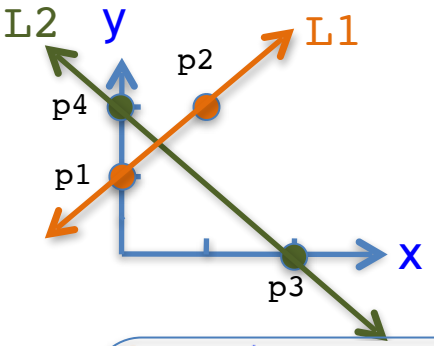
Thinking Linearly

```
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
```

```
>>> P1 = Point(1.0, 2.0)
>>> P2 = Point(2.0, 3.0)
>>> L1 = Line(P1,P2)
>>> str(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
```

```
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
```

```
>>> 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)
```

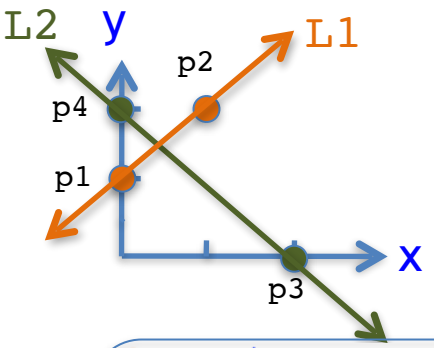



```
class Point:
    def __init__(self, InputX, InputY):
        self.x = 1.0*InputX
        self.y = 1.0*InputY

    def __repr__(self):
        return "(" + 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

    def parallel(self, other):


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

Can you think of another way of writing this line?

```
>>> 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 "(" + 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 __str__(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 Line.parallel(self, other): return None
        else:
            x = (self.yintercept - other.yintercept)/(other.slope - self.slope)
            y = self.slope * x + self.yintercept
            return Point(x, y)
```

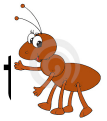
```
if self.parallel(other): return None
```

Default arguments

A minor point

```
class Student:

   def __init__(self,
      firstName, lastName, school="Stevens", major =
"undeclared")
```

```
>>> nick = Student("Nick", "Carter")
>>> joe = Student(firstName = "Joe", "Shmo", "HMC")
>>> anna = Student("Anna", "Litik", major="Physics")

NOT:
>>> elmo = Student("Elmo")
>>> bigBird = Student("Big", "Bird", firstName = "Tweety")
>>> bart = Student(school="PIT", "Bart", "Simpson")
```

Inheritance (inherit  s!)

A BIG DEAL!
On the next slide.

```
class Person:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __str__(self):
        return self.firstName + " " + self.lastName
```

```
>>> dave = Person("Dave", "Naumann")
>>> dave
Dave Naumann
>>> dave.asleep(2)
True
```

```
class Person:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __str__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        Person.__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11
    def __str__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"
```

Sleeping until 11 AM!?

```
>>> s = Student("Sue", "Persmart", 18)
>>> str(s)
Sue Persmart, 18 years old
>>> s.asleep(2)
False
```

```
class Person:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __str__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        Person.__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11

    def __str__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"

class Mudder(Student):
    def __init__(self, first, last, age, dorm):
        Student.__init__(self, first, last, age)
        self.dorm = dorm

    def asleep(self, time):
        return False
```

```
>>> wally = Mudder("wally", "wart",
                   42, "west")
>>> str(wally)
?
>>> wally.asleep(2)
?
```

PRIVATE VARIABLES – DON'T EXIST IN PYTHON

But by convention, names of certain form are treated specially.

```
class PrivatePerson:
    def __init__(self, first, last):
        self.firstName = first
        self.__lastName = last # note two underscores

    def asleep(self, time):
        return 0 <= time <= 7

    def __str__(self):
        return self.firstName + " " + self.__lastName

Try this in the shell:

a = Person("Ada", "Lovelace")
b = Person("Maria", "Klawe")
a.firstName # returns its value
a.__lastName # error, no such attribute
a.firstName = "ada!"
a.__lastName = "Lovelace!" # ok
print(a) # ada! Lovelace -- last name unchanged!
a._Person__lastName = ”changed”
print(a) # ada! Changed
```

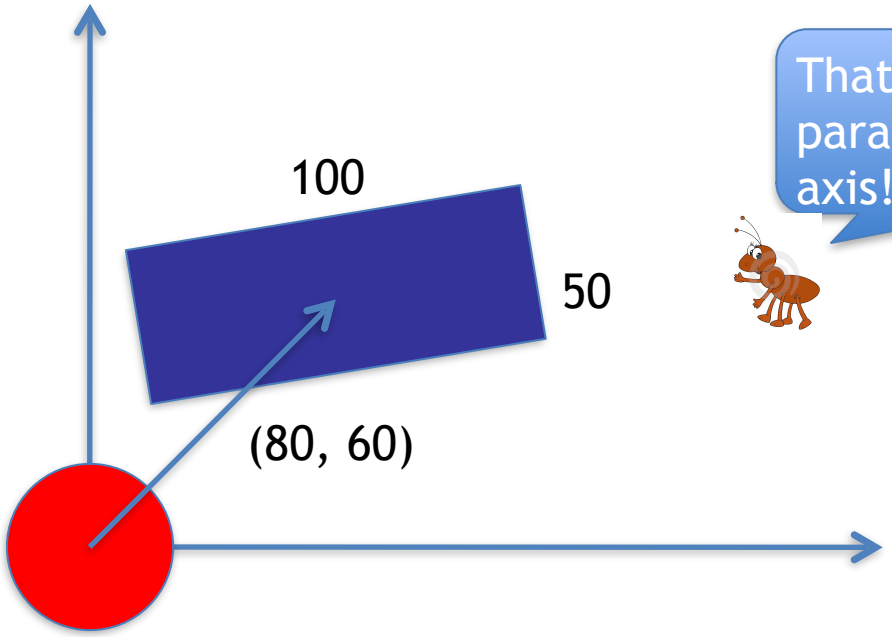
Private variables are for **encapsulation**, an important topic we don't have much time for.
One ingredient: use 'private' variable and provide getters/ setters for attributes.
An attribute without a setter is *immutable*.

This shows that names like __lastName get renamed by Python to _Person__lastNam

The Dangers of Inheritance

Millisoft “Shapes”

```
>>> r = Rectangle(100, 50, center=Vector(80, 60), color=“blue”)
>>> c = Circle(radius=30, color = “red”) # default center (0,0)
>>> r.rotate(15) # 15 degree counter-clockwise rotation
>>> r.render()
>>> c.render()
```



That rectangle an’t parallel to the x-axis!

Demo `ShapesDemo.py`

```
import math  # Now we have math.cos(angle), math.sin(angle), etc.  Angles are in radians
import turtle
from Matrix import *
from Vector import *

class Shape:
    def __init__(self):
        self.points = []   # List of Vectors!

    def render(self):
        turtle.penup()
        turtle.setposition(self.points[0].x, self.points[0].y)
        turtle.pendown()
         …

    def rotate(self, theta):
        """ Rotate shape by theta degrees """
        theta = math.radians(theta) # Python thinks in radians
         …

    def translate(self, vect):
        """ Move  """
        theta = math.radians(theta) # Python thinks in radians
         …

class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):
     …


class Square …  (constructor takes width, optional center, optional color)
```

Time to get our Object-Oriented muscles in Shape!

Transformations, Matrices, and all that Jazz…

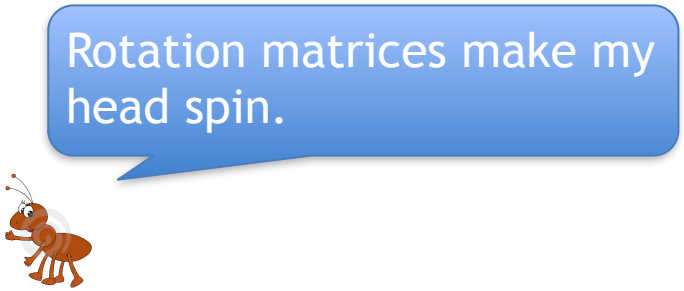
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} = \begin{pmatrix} aa' + bc' & ba' + dc' \\ ca' + dc' & cb' + dd' \end{pmatrix}$$

Represent point (x,y) in vector notation.
Represent transformation as matrix. Transform = multiply (fast!)
(Optional: compose transformations by matrix multiplication.)

Transformation Matrices

Rotation
Scaling
Translation



A Matrix Class

```
>>> m1 = Matrix(0, -1, 1, 0)
>>> m2 = Matrix(1, 2, 3, 4)
>>> m1
0 -1
1 0
>>> m2
1 2
3 4
>>> m1+m2
1 1
4 4
>>> m1*m2
-3 -4
1 2
>>> m1.get(0, 1)
-1
>>> m1.set(1, 0, 42)
```

I thought that Linear Algebra was the Matrix Class!

Matrix Class

```
def Matrix:
    """2x2 matrix class"""

    def __init__(self, a11=0, a12=0, a21=0, a22=0)
          self.array = [[a11, a12], [a21, a22]]

    def __repr__(self)

    def set(self, row, column, value):

    def get(self, row, column):

    def __mul__(self, other):
        """other may be a matrix OR a vector and returns
        the product of self and other."""
         Blah, blah, blah
         if other.__class__.__name__ == "Matrix":
           blah, blah, blah
         else:  # it's HOPEFULLY a Vector!
```

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

    def magnitude(self):
        blah, blah, blah
        return …

    def normalize(self):
      mag = self.magnitude()
      self.x = self.x/mag
      self.y = self.y/mag
```

I was trying to *anti*cipate how this would be done!

```
def __mul__(self, other):
    """ if other is a Matrix, returns a Matrix.
    If other is a Vector, returns a Vector."""
    if other.__class__.__name__ == "Matrix":
        result = Matrix()
        for row in range(0, 2):
            for col in range(0, 2):
            # Compute result matrix
                entry = 0
                for i in range(0, 2):
                    entry += _____________________________
                result.set(row, col, entry)
        return result
    elif other.__class__.__name__ == "Vector":

        x = _____________________________________
        y = _____________________________________
        return _____________________________________
    else:
        print "Can't multiply a matrix by a ", other.__class__.name__
```

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

    def magnitude(self):
        blah, blah, blah
        return …

    def normalize(self):
      mag = self.magnitude()
      self.x = self.x/mag
      self.y = self.y/mag
```

Fill this in…

```
def __mul__(self, other):
   """ if other is a Matrix, returns a Matrix.
   If other is a Vector, returns a Vector."""
   if other.__class__.__name__ == "Matrix":
      result = Matrix()
      for row in range(0, 2):
         for col in range(0, 2):
         # Compute result matrix in the given row and col
            entry = 0
            for i in range(0, 2):
               entry += self.get(row, i) * other.get(i, col)
            result.set(row, col, entry)
      return result
   elif other.__class__.__name__ == "Vector":

      x = self.get(0, 0) * other.x + self.get(0, 1) * other.y
      y = self.get(1, 0) * other.x + self.get(1, 1) * other.y
      return Vector(x, y)
   else:
      print "Can't multiply a matrix by a ", other.__class__.name__
```

```
import math  # Now we have math.cos(angle), math.sin(angle), etc.  Angles are in radians
import turtle
from Matrix import *
from Vector import *

class Shape:
    def __init__(self):
        self.points = []  # List of Vectors!

    def render(self):
        turtle.penup()
        turtle.setposition(self.points[0].x, self.points[0].y)
        turtle.pendown()
        turtle.fillcolor(self.color)
        turtle.pencolor(self.color)
        turtle.begin_fill()
        for vector in self.points[1:]:
            turtle.setposition(vector.x, vector.y)
        turtle.setposition(self.points[0].x, self.points[0].y)
        turtle.end_fill()

    def erase(self):
        temp = self.color
        self.color = "white"
        self.render()
        self.color = temp

    def rotate(self, theta):
        """ Rotate shape by theta degrees """
        theta = math.radians(theta) # Python thinks in radians

class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):

class Square…  (constructor takes width, optional center, optional color)
```

Time to get our Object-Oriented muscles in Shape!

```
class Shape:

  def __init__(self):
     self.points = []

  def render(self):

  def rotate(self, theta):
     """ Rotate shape by theta degrees """
     theta = math.radians(theta) # Python thinks in radians
```

Do this one last

```
class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):
        SW = Vector(center.x - width/2.0, center.y - height/2.0)
        NW = Vector(center.x - width/2.0, center.y + height/2.0)
        NE = Vector(center.x + width/2.0, center.y + height/2.0)
        SE = Vector(center.x + width/2.0, center.y - height/2.0)
        self.points = [SW, NW, NE, SE]
        self.color = color
```

```
class Square
    def __init__(self, width, center=Vector(0, 0), color = "black"):
```

Do this one first

```
class Shape:

  def __init__(self):
     self.points = []

  def render(self):

  def rotate(self, theta):
     """ Rotate shape by theta degrees """
     theta = math.radians(theta)
```

Do this one last

```
class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):
        SW = Vector(center.x - width/2.0, center.y - height/2.0)
        NW = Vector(center.x - width/2.0, center.y + height/2.0)
        NE = Vector(center.x + width/2.0, center.y + height/2.0)
        SE = Vector(center.x + width/2.0, center.y - height/2.0)
        self.points = [SW, NW, NE, SE]
        self.color = color
```

```
class Square(Rectangle):
    def __init__(self, width, center=Vector(0, 0), color = "black"):
        Rectangle.__init__(self, width, width, center, color)
```

```
class Shape:

  def __init__(self):
     self.points = []

  def render(self):
```

```
def rotate(self, theta):
    """Rotate shape by theta degrees """
    theta = math.radians(theta)  # Python thinks in radians!
    RotationMatrix = Matrix(math.cos(theta), -1*math.sin(theta), math.sin(theta), math.cos(theta))
    NewPoints = []
    for vector in self.points:
        newvector = RotationMatrix * vector
        NewPoints.append(newvector)
    self.points = NewPoints
```

```
class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):
        SW = Vector(center.x - width/2.0, center.y - height/2.0)
        NW = Vector(center.x - width/2.0, center.y + height/2.0)
        NE = Vector(center.x + width/2.0, center.y + height/2.0)
        SE = Vector(center.x + width/2.0, center.y - height/2.0)
        self.points = [SW, NW, NE, SE]
        self.color = color
```

```
class Square(Rectangle):
    def __init__(self, width, center=Vector(0, 0), color = "black"):
        Rectangle.__init__(self, width, width, center, color)
```

```
class Shape:

  def __init__(self):
     self.points = []

  def render(self):

  def rotate(self, theta):
```

```
class Circle (Shape):
    def __init__(self, center=Vector(0,0), radius=10, color="black")
```

```
turtle.circle(50)
```

Inherit render and rotate from Shape?

More “Draw” Tricks

Rotation about an arbitrary point

Homogenous coordinates and translatio

August Mobius