The end is near!
Gauging your Workload

On your worksheet…
   On the front, bottom-right corner…

How many hours, outside of lecture, did you spend on this course this past week?
(Include lab time in your reported hours)
Learning Goals
• Describe inheritance of shapes
• Describe matrix operations on shapes
>>> r = Rectangle(100, 50, center=Vector(200, 0), color="blue")
>>> c = Circle(radius=30, color="red") # default center (0,0)
>>> r.rotate(45) # 45 degree ccw rotation about origin
>>> r.render() # draw it!
>>> c.render() # draw it!

This material is shared with CS 5 “Black”
class Shape:
    def __init__(self):
        self.points = []  # list of Vectors!
    def render(self):  # code here...
    def erase(self):  # code here...
    def rotate(self, theta):  # code here...
    def scale(self, stretch):  # code here...
    def translate(self, shift):  # code here..

class Rectangle(Shape):

class Circle(Shape):

Inheriting from Shape means we don’t have to reinvent the wheel!

I don’t plan to ever call the Shape factory to make a Shape. But I will call the Rectangle and Circle factories!

an “abstract” class

>>> r = Rectangle(100, 50, center=Vector(200, 0), color="blue")
>>> c = Circle(radius=30, color="red")  # default center (0,0)
>>> r.rotate(45)  # 45 degree ccw rotation about origin
>>> r.render()  # draw it!
>>> c.render()  # draw it!
Matrix Class

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __repr__(self):
        return ...
    def magnitude(self):
        blah, blah, blah
        return ...
    def normalize(self):
        blah, blah, blah
    def __neg__(self):
        return Vector(blah, blah)
    def __add__(self, other):
        blah, blah, blah
        return Vector(blah, blah)
    def __sub__(self, other):
        blah, blah, blah
        return Vector(blah, blah)

from vector import *

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):
        return ...
    def get(self, row, column):
        return self.array[row][col]
    def set(self, row, column, value):
        self.array[row][col] = value
    def __add__(self, other):
        return ...
    def __mul__(self, other):
        return ...

>>> M = Matrix(1, 2, 3, 4)
>>> N = Matrix(1, 1, 2, 2)
>>> v = Vector(1, 3)
>>> P = M * N  # M.__mul__(N)
>>> q = M * v  # M.__mul__(v)
```
class Shape:
    def __init__(self):
        self.points = []  # list of Vectors!

    def render(self):
    def erase(self):
        def rotate(self, theta):
        def scale(self, stretch):
        def translate(self, shift):

First, let us rotate a single point about the origin. We will extend in homework...

\[
R = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]
import math  # has math.cos(angle) and math.sin(angle)

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

    def render(self):
        blah blah blah

    def rotate(self, theta):
        """Rotate shape by theta degrees""

        # Python's trig functions expect input in radians,
        # so this function converts from degrees into radians.
        theta = math.radians(theta)

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)
import math  # has math.cos(angle) and math.sin(angle)

class Shape:
    def __init__(self):
        self.points = []
    def render(self):
        blah blah blah
    def rotate(self, theta):
        """Rotate shape by theta degrees""
        theta = math.radians(theta)
        rotation_matrix = Matrix(math.cos(theta), -math.sin(theta),
                                  math.sin(theta), math.cos(theta))
        new_points = []
        for vector in self.points:
            new_vector = rotation_matrix * vector
            new_points.append(new_vector)
        self.points = new_points

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)
What does this do?

def demo1():
    r = Square(50, Vector(0, 100), color="blue")
    for i in range(6):
        for j in range(6):
            r.render()
            r.rotate(60, r.center)
            r.rotate(60, Vector(0, 0))

Demo shapes_demo.py!
class Shape:
    def __init__(self):
        self.points = []  # list of Vectors!

    def render(self):  code...
    def erase(self):  code...
    def rotate(self, theta):  code...
    def scale(self, stretch):  code...
    def translate(self, shift):  code...
Circles…

class Shape:

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

    def render(self):
        code here...

    def rotate(self, theta):
        code here...

class Circle(Shape):

    def __init__(self, center=Vector(0,0), radius=10, color="black"):
        self.center = center
        self.radius = radius
        self.color = color

The turtle draws circles using
turtle.circle(radius)

Should we still use the
Shape class render, rotate, translate, etc?
def demo2():
    r = Square(40, Vector(0, 100), color="blue")
    for i in range(6):
        for j in range(6):
            r.render()
            r.rotate(60, r.center)
            r.scale(1.1)
            r.rotate(60)
    c = Circle(Vector(0, 250), 50, color="red")
    for i in range(6):
        c.render()
        c.rotate(60)
        c.scale(0.8)

Demo shapes_demo.py!
Getting in Shape

• Translate
• Rotate about an arbitrary point
• Scale
• Add another shape of your choice (inheritance!)
• Compound shapes!
• Flip about an arbitrary line (defined by two Vectors) [EXTRA CREDIT!]
def demo3():
    radius = 30
    side_length = 10
    group = Compound()
    for angle in range(0, 360, 30):
        angle = math.radians(angle)  # convert from degrees to radians
        s = Square(side_length, Vector(radius * math.cos(angle), \
                   radius * math.sin(angle)))
        group.append(s)
    group.translate(Vector(-200, -200))
    for do_it in range(5):
        group.render()
        group.translate(Vector(30, 30))
class Compound(Shape):
    def __init__(self, center=Vector(0, 0), shape_list=[]):
        blah blah blah
        self.shapes = shape_list
    def render(self):
    def translate(self, shift):
    def rotate(self, theta, about=Vector(0, 0)):
    def scale(self, stretch):
    def append(self, shape):
    def __add__(self, other):

Compound “is a” Shape (inheritance) and “has” Shapes (composition).
class Person:
    def __init__(self, ...):
        blah, blah, blah

class Infantryman(Person):
    def __init__(self, ...):
        blah, blah, blah

class Kangaroo(Infantryman):
    def __init__(self, ...):
        blah, blah, blah
Learning Goals
• Practice inheritance
• Have fun simulating critters!
Critters!

hares

foxes
import turtle
import math
import random

#========================================
# globals
RADIUS = 300
NUMFOXES = 6
NUMHARES = 10

#========================================
# functions

def random_position():
    """Sample uniformly in circle""
    r = RADIUS * math.sqrt(random.uniform(0,1))
    theta = random.uniform(0, 2*math.pi)
    x = r * math.cos(theta)
    y = r * math.sin(theta)
    return x, y

def random_heading():
    """Sample uniformly""
    return random.uniform(0, 360) # degrees

#========================================
# classes

class Critter(turtle.Turtle):
    """A Critter""
    code here...

class Fox(Critter):
    """A Fox (fast darkorange square)""
    code here...

class Hare(Critter):
    """A Hare (slow tan circle)""
    code here...

#========================================
# main

def main():
    # initialize field
    turtle.dot(2*RADIUS, "darkgreen")
    # initialize critters
    critters = []
    for i in range(NUMFOXES):
        critters.append(Fox("Fox " + str(i)))
    for j in range(NUMHARES):
        critters.append(Hare("Hare " + str(j)))
    # move critters indefinitely
    while True:
        for critter in critters:
            critter.move()}
class Critter(turtle.Turtle):
    """A Critter""
    
def __init__(self, name):
        turtle.Turtle.__init__(self)  # set name
        # set random position and heading
        self.penup()

def move(self):
    """Move the critter""
    # move forward one step
    if self.distance(0,0) >= RADIUS:
        # invalid, go back, pick new random heading

class Fox(Critter):
    """A Fox (fast dark orange square)""
    
def __init__(self, name):
        

class Hare(Critter):
    """A Hare (slow tan circle)""
    
def __init__(self, name):
        

Turtle Methods you may find useful
- def speed(speed) – set speed
- def shape(name) – set shape
- def color(name) – set color
- def setposition(position) – set position
- def setheading(heading) – set heading
- def forward(distance) – move forward by distance

Complete according to specifications.
Demo critters.py and critters_chase.py!

Critter Specifications
- A Critter has a name.
- On a move, a Critter moves forward by speed. If the resulting location is invalid, a Critter turns around, moves back, and picks a new random heading.
- A Fox moves fast (speed = 5), is "darkorange" and "square"
- A Hare moves slow (speed = 1), is "tan" and "circle"
class Critter(turtle.Turtle):
    """A Critter""

    def __init__(self, name):
        turtle.Turtle.__init__(self)
        self.name = name  # set name

        # set random position and heading
        self.penup()
        self.setposition(random_position())
        self.setheading(random_heading())

    def move(self):
        """Move the critter""
        # move forward one step
        self.forward(self.speed())

        if self.distance(0,0) >= RADIUS:
            # invalid, go back, pick new random heading
            self.right(180)
            self.forward(self.speed())
            self.setheading(random_heading())

class Fox(Critter):
    """A Fox (fast dark orange square)""

    def __init__(self, name):
        Critter.__init__(self, name)
        self.speed(5)
        self.shape("square")
        self.color("darkorange")

class Hare(Critter):
    """A Hare (slow tan circle)""

    def __init__(self, name):
        Critter.__init__(self, name)
        self.speed(1)
        self.shape("circle")
        self.color("tan")

Turtle Methods you may find useful
• def speed(speed) – set speed
• def shape(name) – set shape
• def color(name) – set color
• def setposition(position) – set position
• def setheading(heading) – set heading
• def forward(distance) – move forward by distance

Complete according to specifications.
Demo critters.py and critters_chase.py!

Critter Specifications
• A Critter has a name.
• On a move, a Critter moves forward by speed. If the resulting location is invalid, a Critter turns around, moves back, and picks a new random heading.
• A Fox moves fast (speed = 5), is “darkorange” and “square”
• A Hare moves slow (speed = 1), is “tan” and “circle”
What’s coming next…

• Tuesday: Projects showcase!
• After break
  – Work on your project (milestone + final project)
  – Labs are just for working and getting help on projects
• Class material: The limits of computation!