Review

- Lists
- Functional programming - map()
- List comprehensions
- Sets
- Dictionaries
- Nested Data Structures

Assignment for Today

- Write a program with three classes
  - Rectangle
  - Triangle
  - Ellipse
- Requirements
  - Give each class the appropriate draw method and its own fill color
  - Default stroke is dark gray with a weight of 1
  - When the mouse moves over an object, the stroke color should change to white
  - If an object is clicked, it becomes “selected” which is indicated by a stroke weight of 5

Start with a standard setup

```
# shapes1.py
from Processing import *
import math

window(500, 500)

…

shapes = []
def draw(o, e):
  background(200)
  for s in shapes:
      s.draw()

frameRate(20)
onLoop += draw
loop()
```

shapes1.py

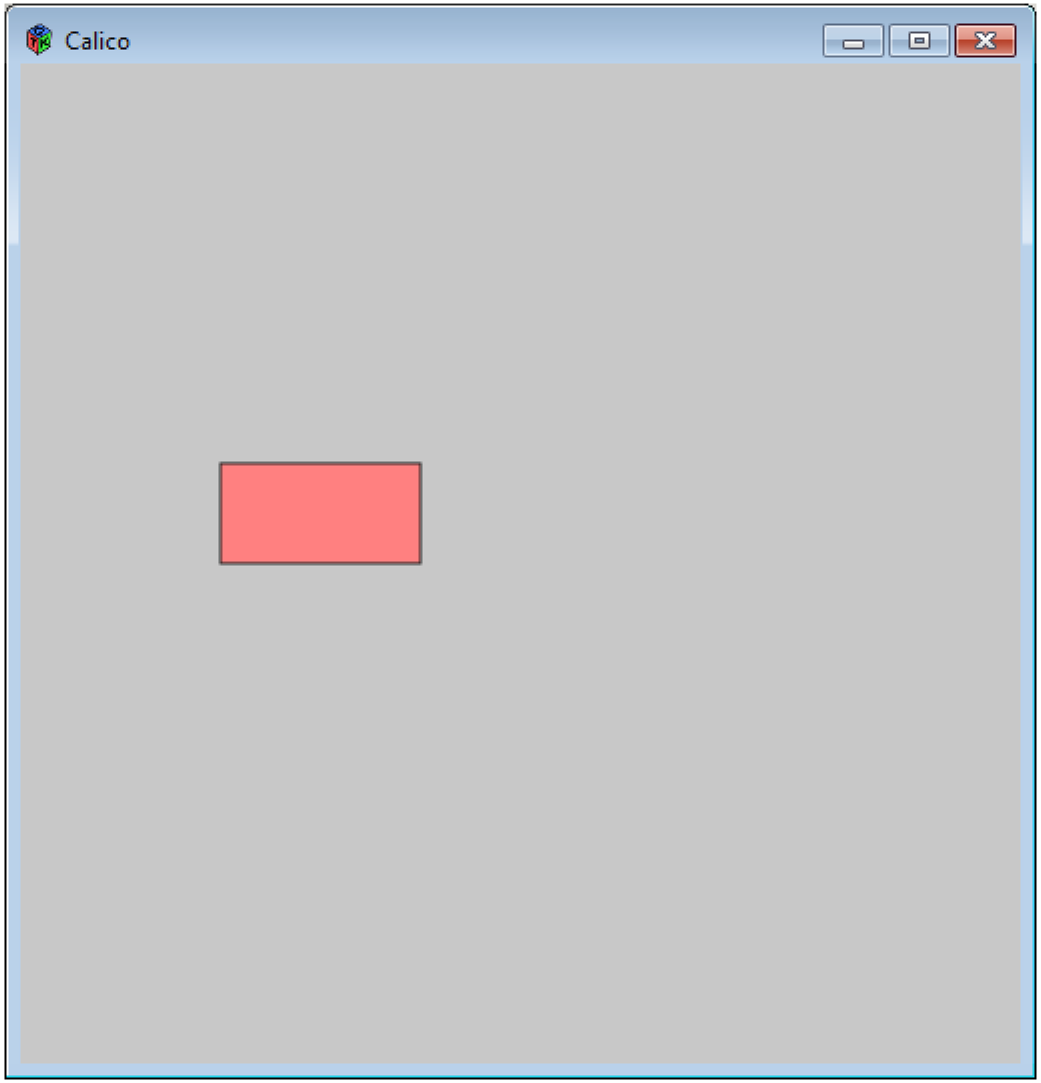
The Rectangle Class

```
# Rectangle Class
class Rectangle:
    def __init__(self, pts):
        self.pts = pts
        self.strokeColor = color(32)
        self.fillColor = color(255, 128, 128)

    def draw(self):
        rectMode(CORNER)
        fill( self.fillColor )
        stroke( self.strokeColor )
        w = self.pts[1][0] - self.pts[0][0]
        h = self.pts[1][1] - self.pts[0][1]
        rect(self.pts[0][0], self.pts[0][1], w, h)


…

shapes.append( Rectangle ( [[100, 200], [200, 250]] ) )
```

A list of points

A point is a list with two members, and a list of points is a list of lists.

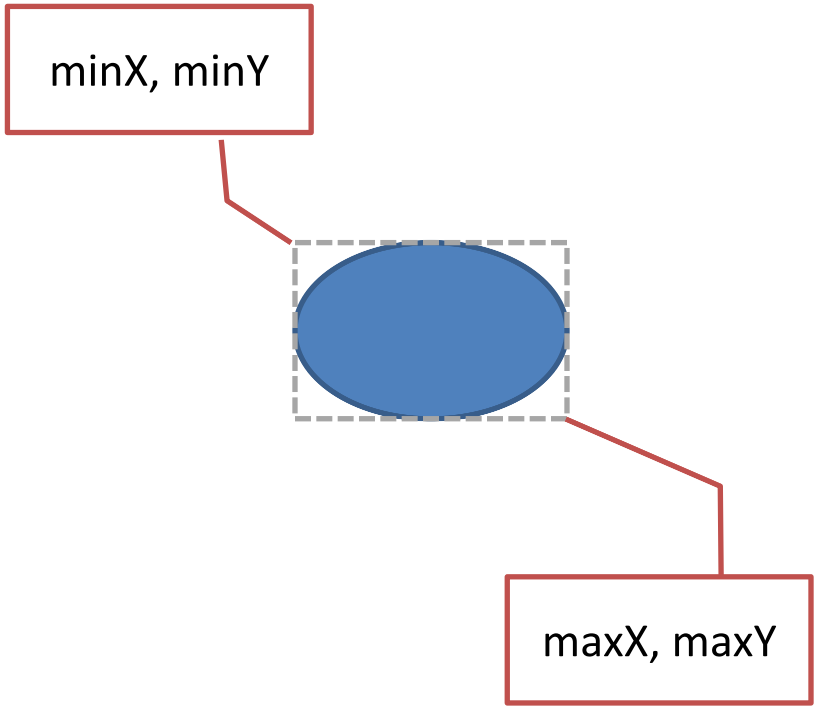
shapes1.py



shapes1.py

Bounding Box Function

```
# Given shape points, compute bounding box
def boundingBox( pts ):
    minX, maxX = pts[0][0], pts[0][0]
    minY, maxY = pts[0][1], pts[0][1]
    for p in pts:
        if p[0] < minX:
            minX = p[0]
        elif p[0] > maxX:
            maxX = p[0]
        if p[1] < minY:
            minY = p[1]
        elif p[1] > maxY:
            maxY = p[1]
    return [minX, minY, maxX, maxY]
```



A reusable function to calculate the bounding box given a list of points

shapes2.py

Rectangle w/ Bounding Box and other Parameters

```
# Rectangle Class
class Rectangle:
    def __init__(self, pts):
        self.pts = pts
        self.strokeColor = color(32)
        self.fillColor = color(255, 128, 128)
        self.bbox = boundingBox(self.pts)
        self.width = self.bbox[2] - self.bbox[0]
        self.height = self.bbox[3] - self.bbox[1]
        self.centerX = 0.5 * (self.bbox[2] + self.bbox[0])
        self.centerY = 0.5 * (self.bbox[3] + self.bbox[1])

    def draw(self):
        rectMode(CORNER)
        fill( self.fillColor )
        stroke( self.strokeColor )
        rect(self.bbox[0], self.bbox[1], self.width, self.height)
```

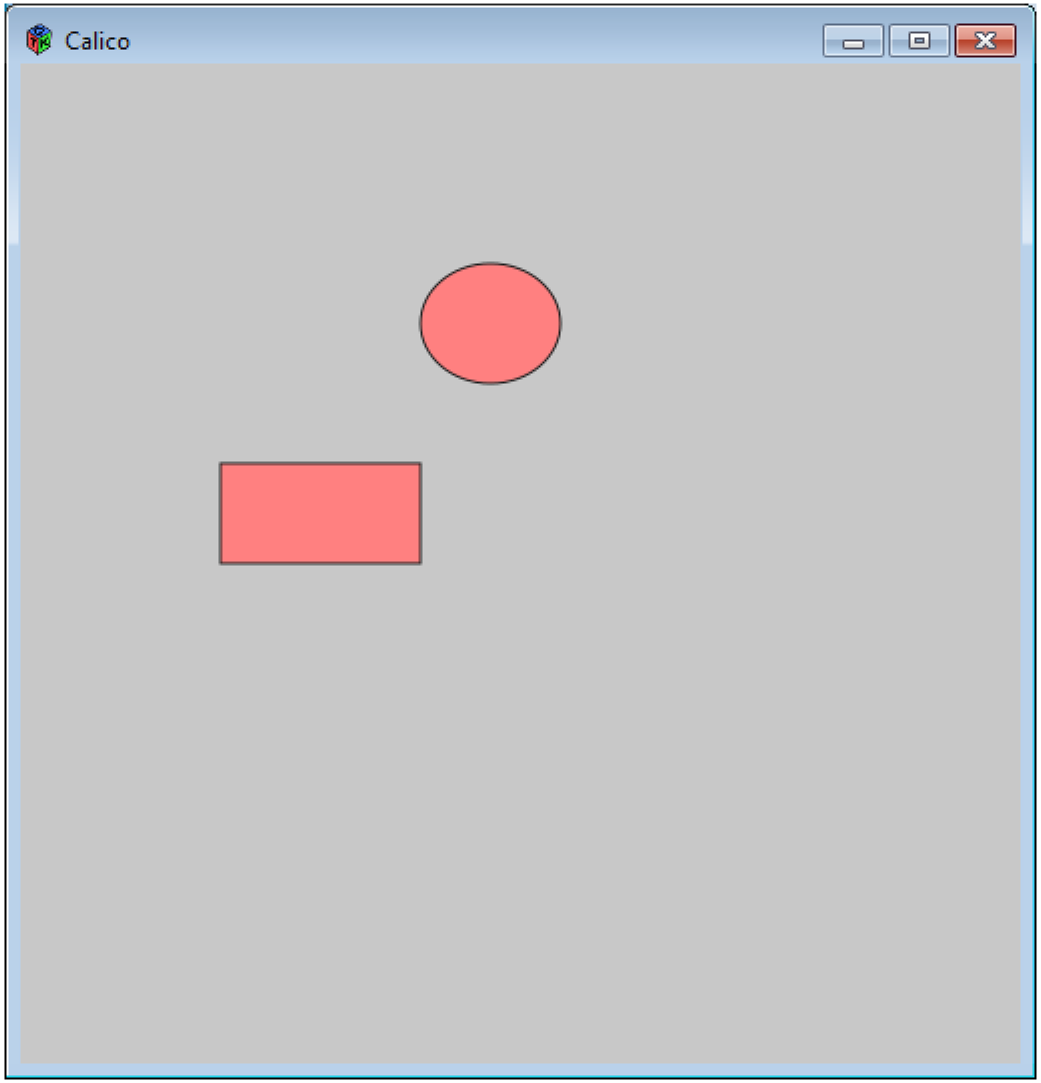
The Rectangle class is expanded to calculate several parameters, including bounding box width, height and center point

shapes2.py

The Ellipse Class – Very Similar to Rectangle

```
# Ellipse Class
class Ellipse:
    def __init__(self, pts):
        self.pts = pts
        self.strokeColor = color(32)
        self.fillColor = color(255, 128, 128)
        self.bbox = boundingBox(self.pts)
        self.width = self.bbox[2] - self.bbox[0]
        self.height = self.bbox[3] - self.bbox[1]
        self.centerX = 0.5 * (self.bbox[2] + self.bbox[0])
        self.centerY = 0.5 * (self.bbox[3] + self.bbox[1])

    def draw(self):
        ellipseMode(CORNER)
        fill( self.fillColor )
        stroke( self.strokeColor )
        ellipse(self.bbox[0], self.bbox[1], self.width, self.height)
```

shapes3.py



shapes3.py

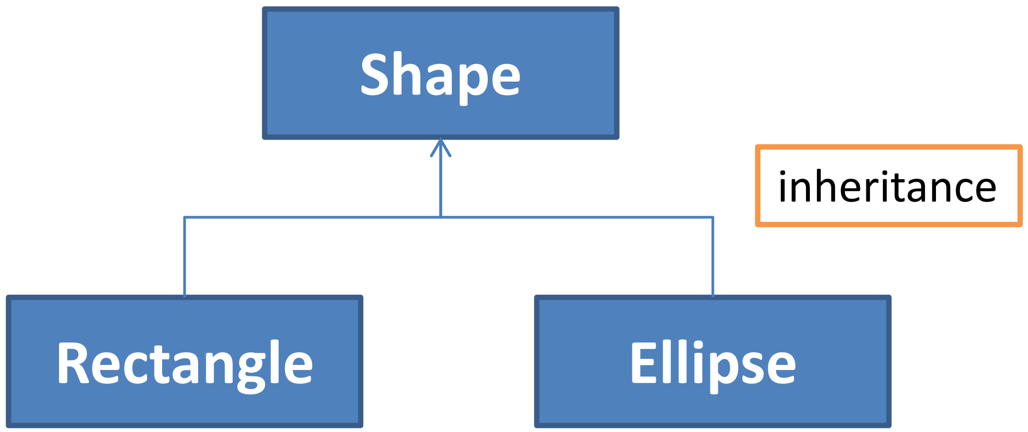
?

- We don’t want to repeat same code over and over again for each new shape class created.
- How can we share methods and instance variables among related classes?

Inheritance

We can set up an explicit relationship between Rectangle and a new class called Shape, and between Ellipse and Shape, called ***Inheritance.***

*This will automatically cause Shape **variables** and **methods** to be automatically accessible by Rectangle and Ellipse.*



Base Class :	Shape
Child Class:	Rectangle, Ellipse

Inheritance - Terminology

- A new class (base class) can be declared to extend the behavior of an existing class (child class)
  - A child class is aka: *derived class, subclass*, …
  - A base class is aka: *parent class*, *superclass*
- A child class automatically gets (i.e. inherits) all members of the base class
  - *Members* include both instance vars and methods
- A child class can override the members of a base class by declaring new members with the same name

The Shape Class

```
# Shared Shape class
class Shape:
    def __init__(self, pts):
        self.pts = pts
        self.strokeColor = color(32)
        self.fillColor = color(255, 128, 128)
        self.bbox = boundingBox(self.pts)
        self.width = self.bbox[2] - self.bbox[0]
        self.height = self.bbox[3] - self.bbox[1]
        self.centerX = 0.5 * (self.bbox[2] + self.bbox[0])
        self.centerY = 0.5 * (self.bbox[3] + self.bbox[1])

    def draw(self):
        fill( self.fillColor )
        stroke( self.strokeColor )
        self.drawShape()

    # Override to perform class-specific behavior
    def drawShape(self):
        pass
```

- All common methods and instance variables moved to the Shape Class
- Subclass specific drawing commands moved to a new drawShape() method

```
# Rectangle Class - Before
class Rectangle:
    def __init__(self, pts):
        self.pts = pts
        self.strokeColor = color(32)
        self.fillColor = color(255, 128, 128)
        self.bbox = boundingBox(self.pts)
        self.width = self.bbox[2] - self.bbox[0]
        self.height = self.bbox[3] - self.bbox[1]
        self.centerX = 0.5 * (self.bbox[2] + self.bbox[0])
        self.centerY = 0.5 * (self.bbox[3] + self.bbox[1])

    def draw(self):
        rectMode(CORNER)
        fill( self.fillColor )
        stroke( self.strokeColor )
        rect(self.bbox[0], self.bbox[1], self.width, self.height)
```

Before

After

```
# Rectangle Class - After
class Rectangle(Shape):
    def __init__(self, pts):
        Shape.__init__(self, pts)

    def drawShape(self):
        rectMode(CORNER)
        rect(self.bbox[0], self.bbox[1], self.width, self.height)
```

Rectangle class before and after consolidating common behavior into Shape subclass

Before

```
# Ellipse Class - Before
class Ellipse:
    def __init__(self, pts):
        self.pts = pts
        self.strokeColor = color(32)
        self.fillColor = color(255, 128, 128)
        self.bbox = boundingBox(self.pts)
        self.width = self.bbox[2] - self.bbox[0]
        self.height = self.bbox[3] - self.bbox[1]
        self.centerX = 0.5 * (self.bbox[2] + self.bbox[0])
        self.centerY = 0.5 * (self.bbox[3] + self.bbox[1])

    def draw(self):
        rectMode(CORNER)
        fill( self.fillColor )
        stroke( self.strokeColor )
        rect(self.bbox[0], self.bbox[1], self.width, self.height)
```

After

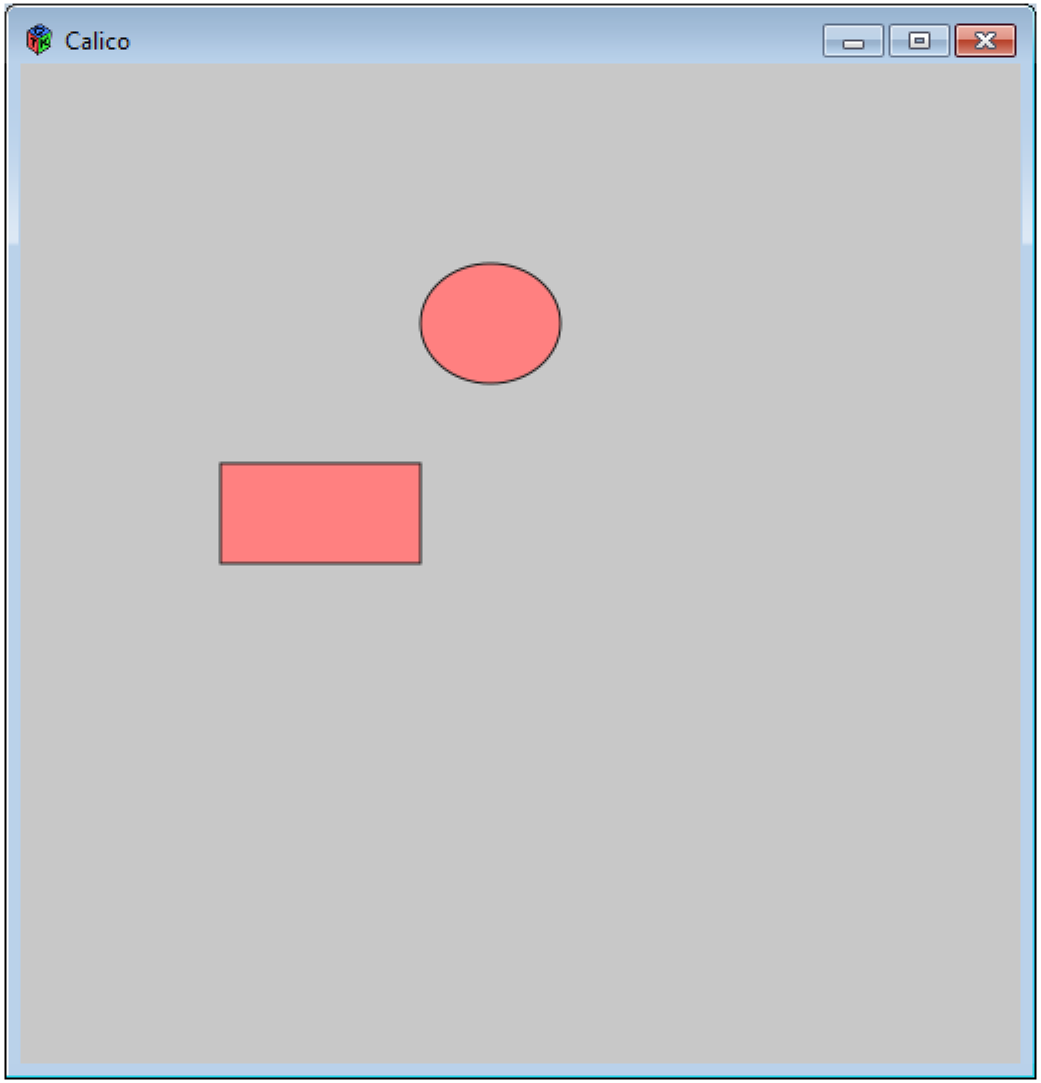
```
# Ellipse Class - After
class Ellipse(Shape):
    def __init__(self, pts):
        Shape.__init__(self, pts)

    def drawShape(self):
        ellipseMode(CORNER)
        ellipse(self.bbox[0], self.bbox[1], self.width, self.height)
```

Ellipse class before and after consolidating common behavior into Shape subclass

Inheritance Relationship

- Set up between classes by adding the base class name in parentheses after child class name
- Optionally, invoke base class constructor with self parameter if child class

```
# Rectangle Class - After
class Rectangle(Shape):
    def __init__(self, pts):
        Shape.__init__(self, pts)
```



Consolidating shared members into the common Shape base class results in identical behavior

shapes4.py

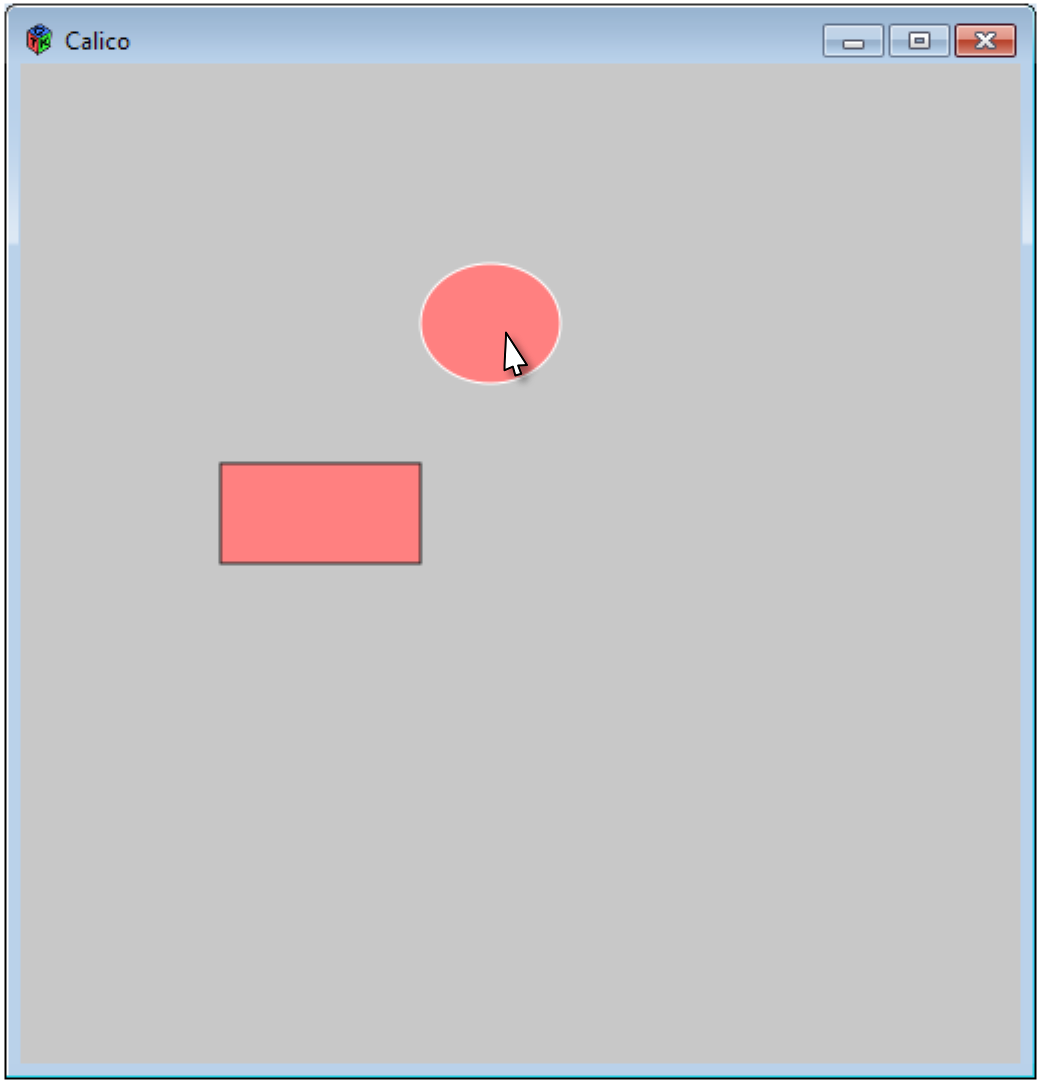
The Power of Inheritance

- A new behavior can be added easily to all child classes by <u>adding once to a common base class</u>
- A common behavior of all child classes can be modified easily by <u>making changes to a base class</u>
- Entirely new child classes can be created by <u>declaring only how it differs</u> wrt the base class

```
# Shared Shape class
class Shape:
    def __init__(self, pts):
        …

    # Default implementation of containsPoint checks bounding box
    def containsPoint(self, x, y):
        if x < self.bbox[0]: return False
        if x > self.bbox[2]: return False
        if y < self.bbox[1]: return False
        if y > self.bbox[3]: return False
        return True

    def mouseMoved(self):
        x, y = mouseX(), mouseY()
        if self.containsPoint(x, y):
            self.strokeColor = color(255)
        else:
            self.strokeColor = color(32)

def mouseMoved(o, e):                # Relay event to all instances
    for s in shapes:
        s.mouseMoved()

onMouseMoved += mouseMoved           # Handle onMouseMoved event
```

Add new methods to Shape that changes stroke color to white when mouse is over the shape

shapes5.py



We added mouseMoved() to Shape only, but all child classes also get the method through inheritance.

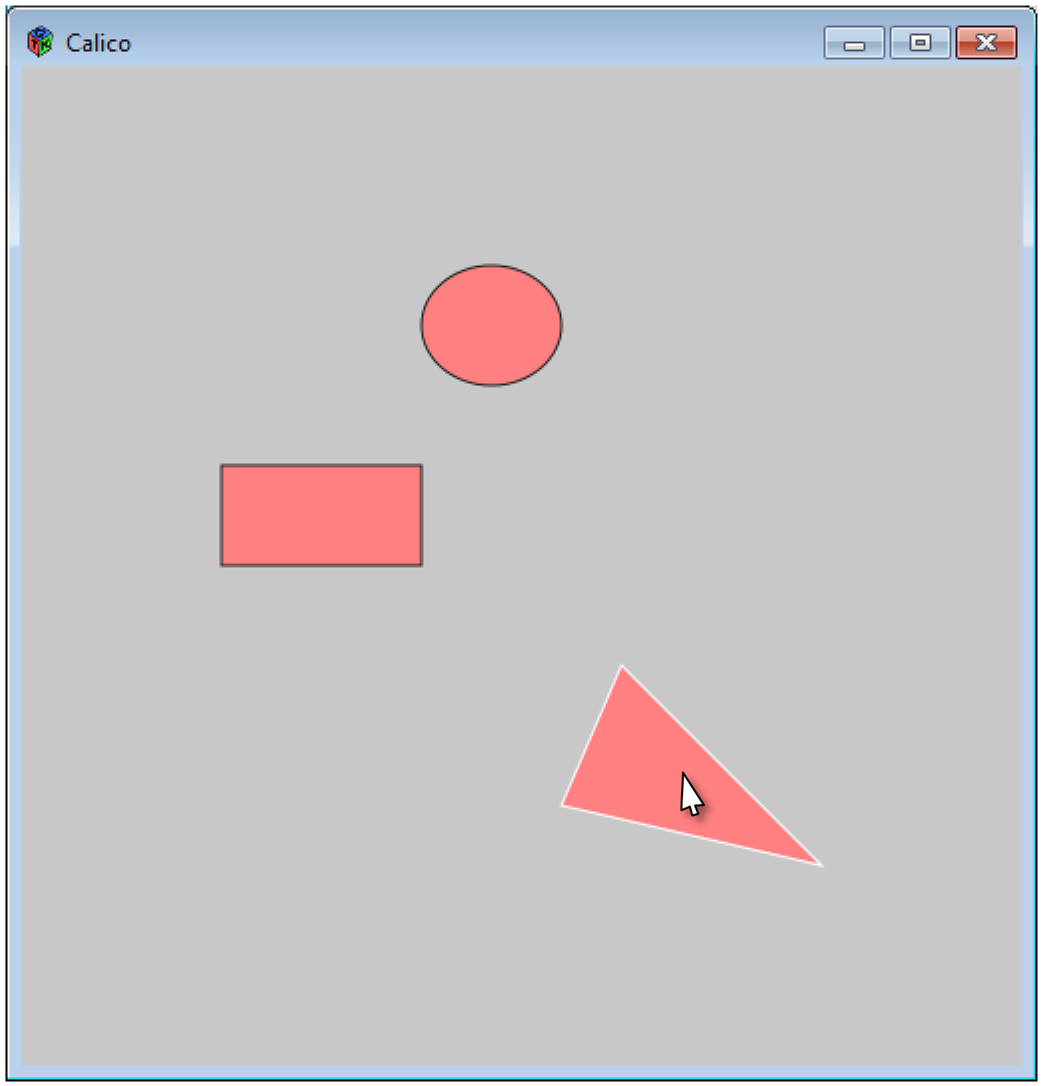
shapes5.py

Adding a Triangle Class

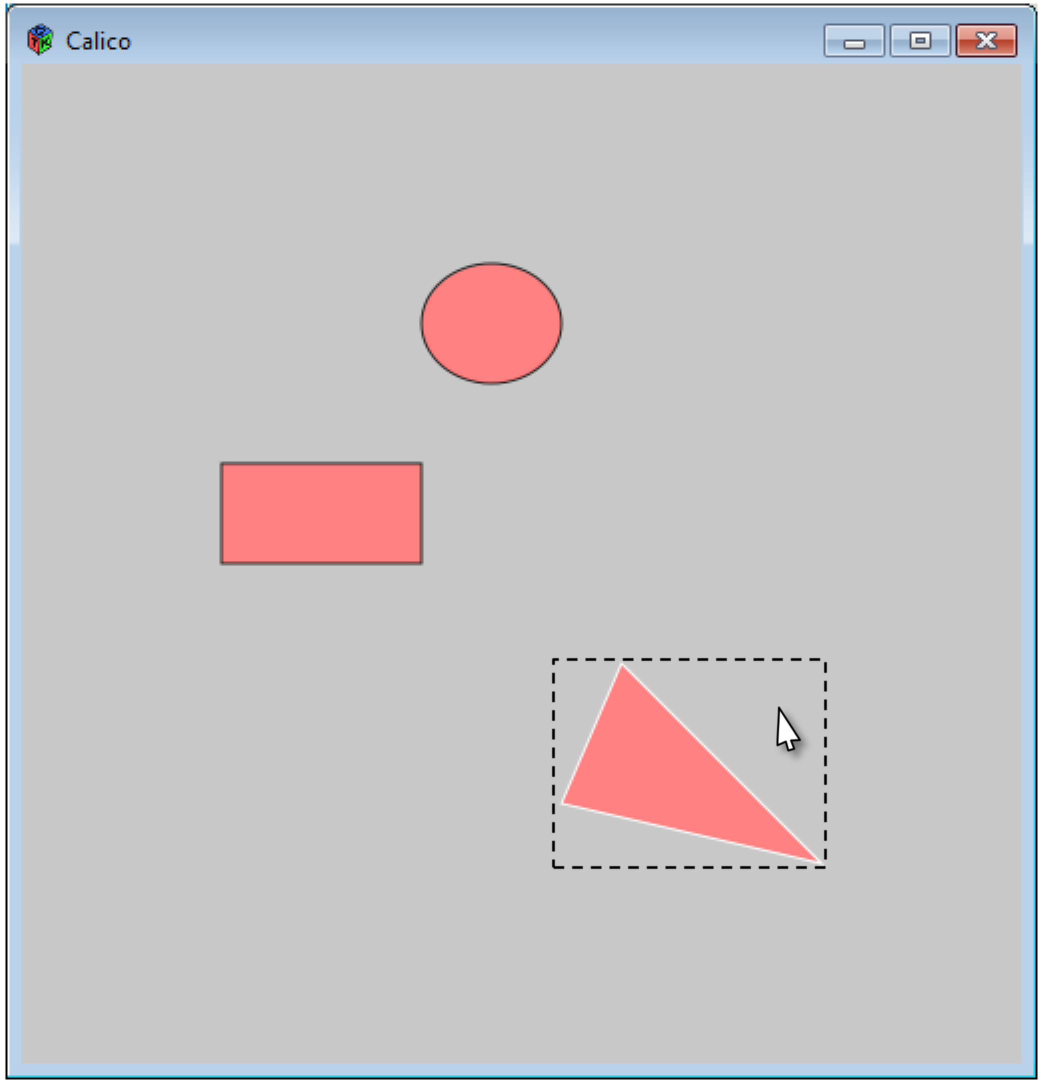
```
class Triangle(Shape):
    def __init__(self, pts):
        Shape.__init__(self, pts)

    # Draw the triangle
    def drawShape(self):
        triangle(self.pts[0][0], self.pts[0][1],
                 self.pts[1][0], self.pts[1][1],
                 self.pts[2][0], self.pts[2][1])
```

- *Adding a new shape means subclassing Shape and adding a drawShape() method*
- *All behavior - like mouse over changing stroke color - is inherited automatically*

shapes6.py



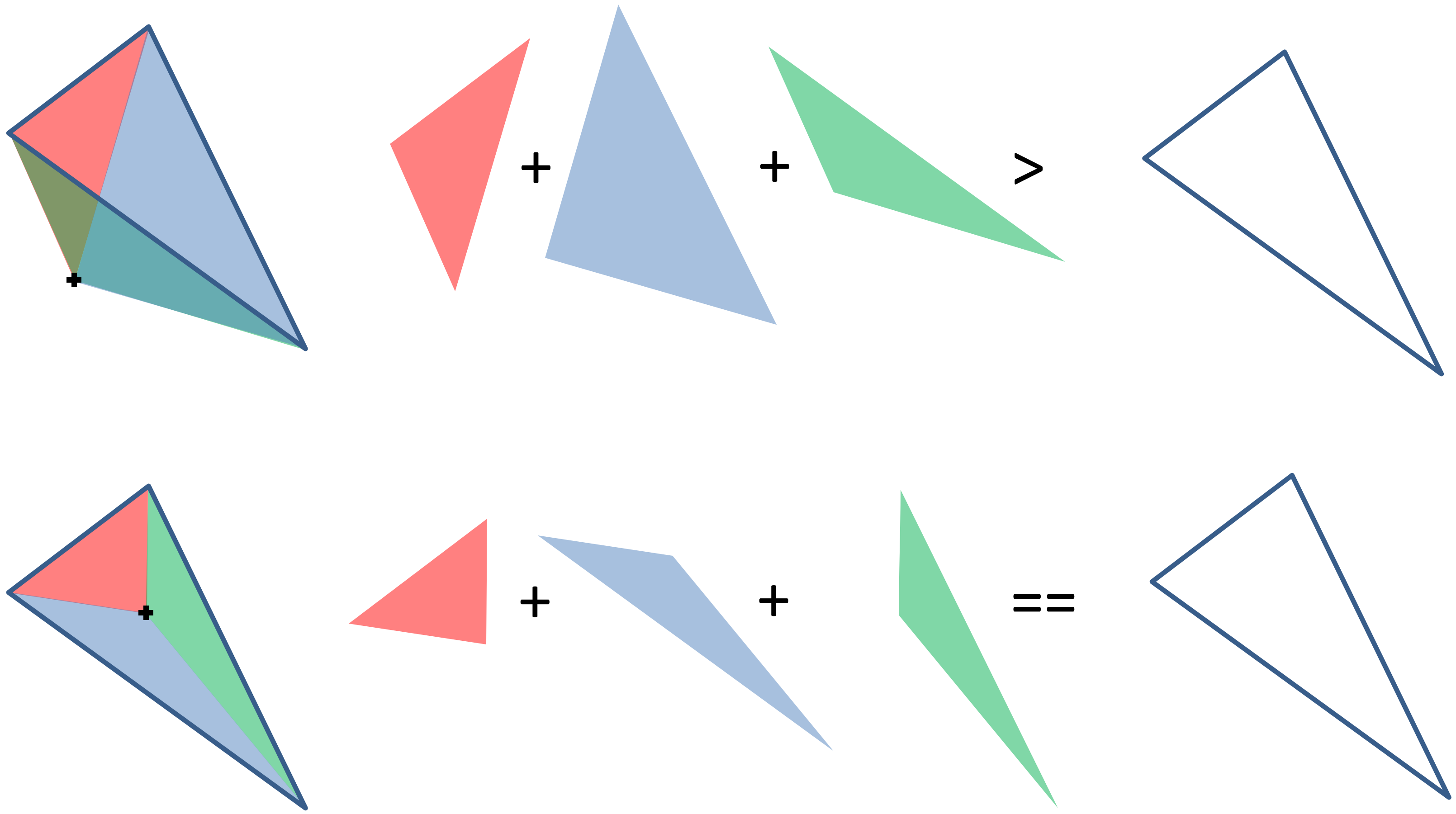
shapes6.py



- *Shape's containsPoint() method is too crude.*
- *We need one that is specific to Triangle.*

shapes6.py

containsPoint() for a Triangle



```
# Helper function
def triangleArea(x1, y1, x2, y2, x3, y3):
    return 0.5*math.fabs((x2-x1)*(y3-y1)-(y2-y1)*(x3-x1))


class Triangle(Shape):
    def __init__(self, pts):
        Shape.__init__(self, pts)
…
    # A point is in a triangle if the sum of the areas of all
    # sub-triangles made with the point <= the area of the
    # triangle itself
    def containsPoint(self, x, y):
        a1 = triangleArea(self.pts[0][0], self.pts[0][1],
                          self.pts[1][0], self.pts[1][1], x, y)
        a2 = triangleArea(self.pts[0][0], self.pts[0][1], x, y,
                          self.pts[2][0], self.pts[2][1])
        a3 = triangleArea(x, y, self.pts[1][0], self.pts[1][1],
                          self.pts[2][0], self.pts[2][1])
        a  = triangleArea(self.pts[0][0], self.pts[0][1],
                          self.pts[1][0], self.pts[1][1],
                          self.pts[2][0], self.pts[2][1])
        return (a1 + a2 + a3) <= a
```

Improve the Triangle class by overriding Shape's containsPoint() method with a better version.

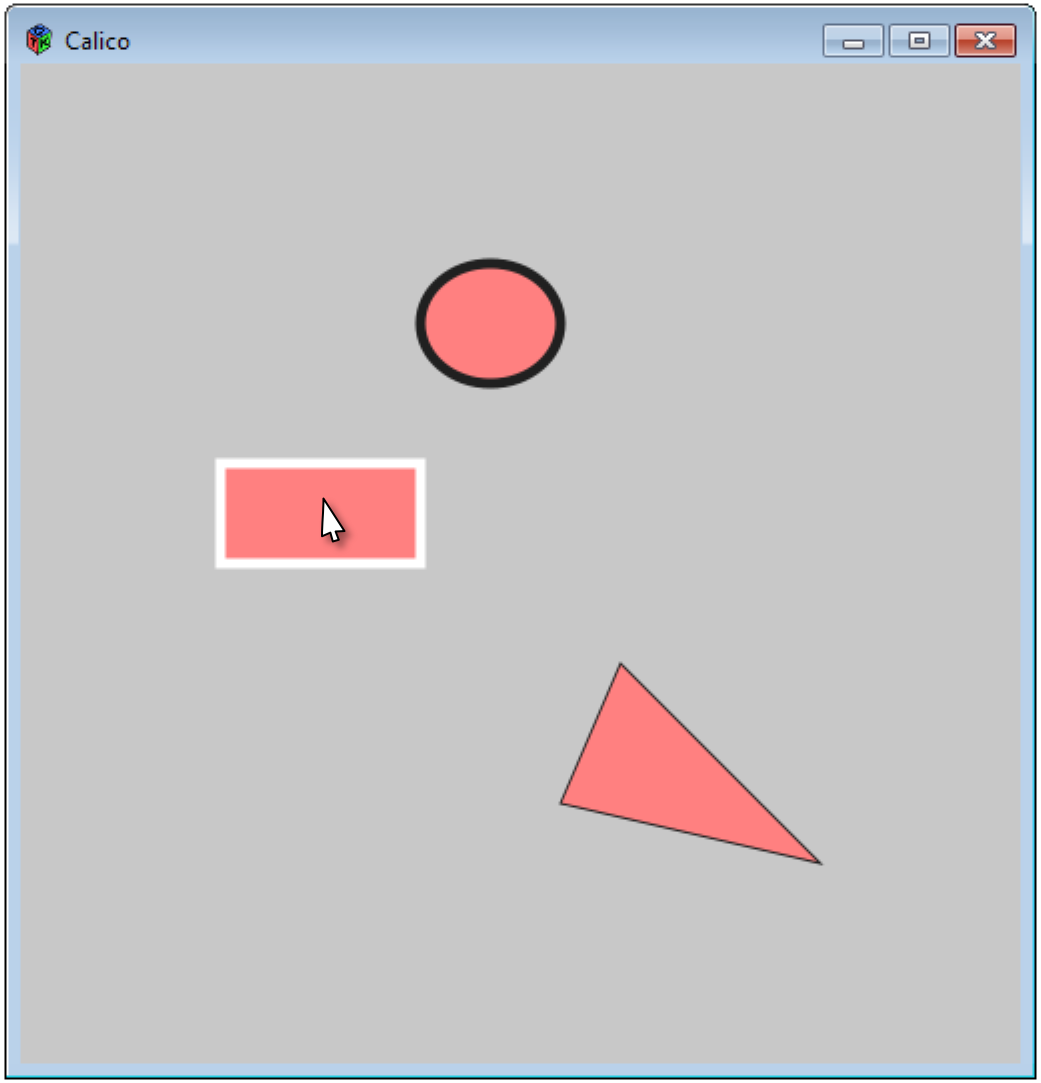
shapes7.py

```
# Shared Shape class
class Shape:
    def __init__(self, pts):
        …
        self.selected = False


    def draw(self):
        fill( self.fillColor )
        stroke( self.strokeColor )
        if self.selected == True:
            strokeWeight(5)
        else:
            strokeWeight(1)
        self.drawShape()
    …

    def mousePressed(self):
        x, y = mouseX(), mouseY()
        # Modify selection state
        if self.containsPoint(x, y):
            self.selected = True
        elif keyCode() != 65505:
            self.selected = False

# Event handlers
def mousePressed(o, e):
    for s in shapes:
        s.mousePressed()

onMousePressed += mousePressed
```

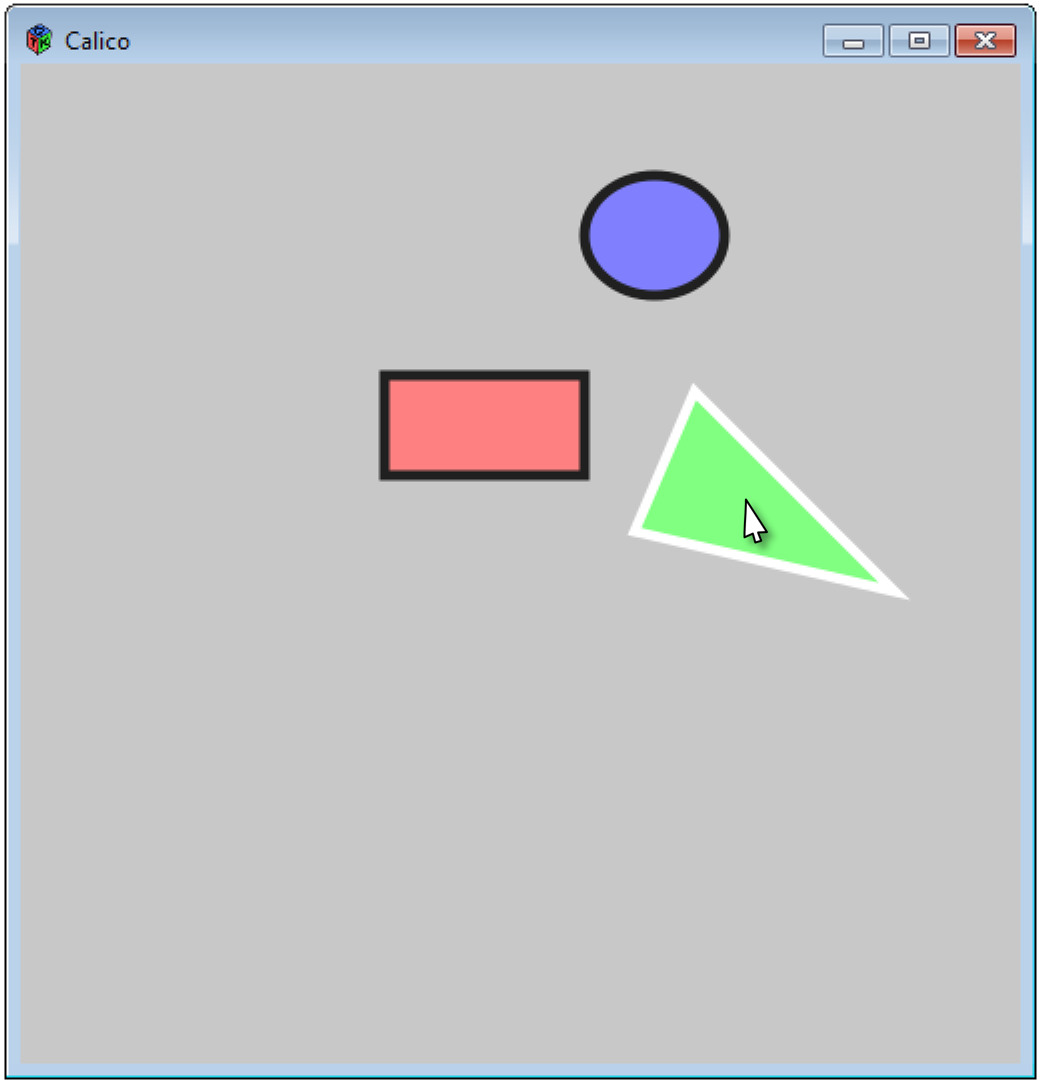
Selection behavior can be added to the Shape Class

All child classes inherit behavior

shapes8.py



shapes8.py



- Final version adds dragging behavior
- And overrides fillColor in child classes

shapes.py

Polymorphism

poly = many, *morph* = form

In Biology, when there is more than one form in a single population

Light-morph jaguar (typical)

Dark-morph or melanistic jaguar (about 6% of the South American population)

In Computing, we have two common types of Polymorphism

1. Signature Polymorphism
2. Subtype Polymorphism

http://en.wikipedia.org/wiki/Polymorphism_%28biology%29

Signature Polymorphism

- It is possible to define multiple functions with the <u>same name</u>, but <u>different signatures</u>.
  - A *function signature* is defined as
    - The function name, and
    - The order and type of its parameters

- Consider the built-in color() function …

```
color(gray)
color(gray, alpha)
color(value1, value2, value3)
color(value1, value2, value3, alpha)
…
```

Subtype Polymorphism

- Inheritance implements Subtype Polymorphism
  - A Rectangle is a type of Shape
  - An Ellipse is a type of Shape
  - A Triangle is a type of Shape
- Implication:
  - A Rectangle can be used in place of a Shape

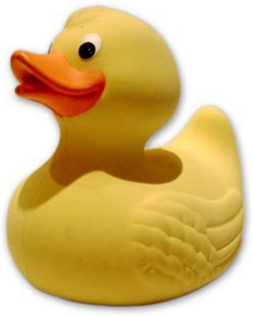
Duck Typing

Python employs so-called "Duck Typing"

If it walks like a duck and quacks like a duck, it's a duck!

Stated more formally…

- "An object's methods and properties determine the valid semantics, rather than its inheritance from a particular class"



Testing Inheritance and Instance Relationships

- isinstance(*object, class*)
  - Returns True if object is an instance of class

- issubclass(*class1, class2*)
  - Returns True if class1 is a child class (direct or indirect) of class2

```
r = Rectangle ( [[100, 200], [200, 250]] )

print( isinstance( r, Rectangle ) )
>>> True

print( issubclass( Rectangle, Shape ) )
>>> True
```

Inheritance Summary

- A relationship established between two classes, established by following child class name with base class in parentheses
- Members (instance vars and methods) of the base class become part of all child classes, automatically
- Child classes can replace (override) base class members by declaring new members with same name
- Inheritance implements the concept of subtype polymorphism
  - Objects of a child class type are also considered to be of a base class type – use issubclass() to test
- Python follows the principle of Duck Typing
  - If it walks like a duck and quacks like a duck, it is a duck