CLASSES AND OBJECTS

Outline

- Simple types
- Creating your own data types
  - Classes
  - Objects
  - Constructors
  - Attributes
  - Methods
  - Lists of objects

A Foundation for Programming

any program you might want to write

http://www.flickr.com/photos/vermegrigio/5923415248/

objects

build even bigger programs and reuse code

functions

graphics, sound, and image I/O

lists

conditionals and loops

math | text I/O

primitive data types | assignment statements

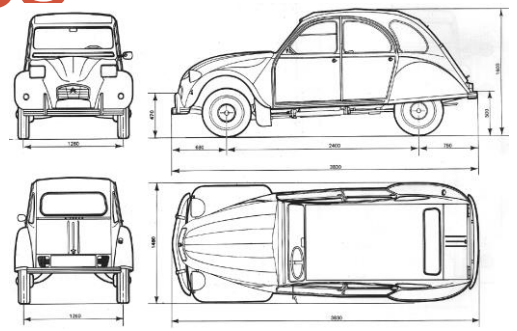
Python Simple Types

Python type	what it stores	examples
`int`	integer values	42 1234
`float`	less precise floating-point values	9.95 3.0e8
`boolean`	truth values	True False
`string`	characters	'a', 'b', ‘Hello!'

Simple Types: Limitations

- Simple types
  - Limited set of operations
    - Example: `int` data type operations: add, subtract, multiply, divide, modulo
  - Can't easily combine related information
    - e.g. MarsLander:
      - two `float`'s to represent your Mars lander's position
      - another two for velocity, etc.

Create Your Own Data Types



- Class
  - ***Blueprint*** for a custom data type
- Object
  - ***Instance*** of a class
  - May be multiple objects for a particular class blueprint

  - Objects have a set of things they know (“state”)
    - Color of different body panels, location, fuel remaining
  - Objects have a set of things they can do (“behavior”)
    - Honk horn
    - Turn on lights
    - Drive forward

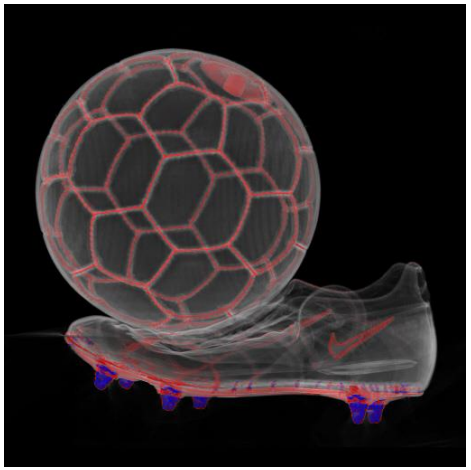
Let's Build a Simple Class

- Goal: represent a circle in 2D
  - What does a circle need to know?
    - x-coordinate
    - y-coordinate
    - radius
  - What can a circle do?
    - Draw itself
    - Print out its position and radius

Setting up the Circle Class

- Create `Circle.py` containing Circle class

```
class Circle:
```



Hello Constructors

- Add a constructor, creates and initializes attributes

```
class Circle:

    # Construct a Circle object at the given (x, y) position with radius r
    def __init__(self, x = 0.0, y = 0.0, r = 0.0):
        self.posX   = x
        self.posY   = y
        self.radius = r
        self.color = color.Color(224, 173, 255)
```

constructor:
Function name _ _init_ _.
This is a requirement!

Adding a Method

- Add methods for what a `Circle` can do

```
class Circle:

…

    # Draw this Circle its location in its current color
    def draw(self):
        StdDraw.setPenColor(self.color)
        StdDraw.filledCircle(self.posX, self.posY, self.radius)

    # Text representation of this Circle
    def toString(self):
        return "(" + str(self.posX) + ", " + str(self.posY) + ") r = " +
str(self.radius)
```

Let's try out our new class!

- Instantiating objects

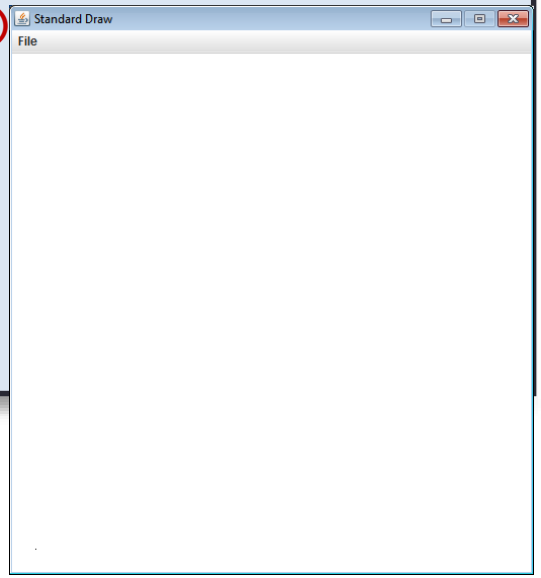
```
import Circle
import random
import StdDraw

# Declare and instantiate the two circle objects
big   = Circle.Circle()
small = Circle.Circle()

# Change both Circles from the default color
big.setColor(random.randint(0, 255), random.randint(                55))
small.setColor(random.randint(0, 255), random.randi                 255))

# Draw them on the screen
big.draw()
small.draw()
StdDraw.show(500)
```

"Build me a Circle object, I'm not sending you any input about how to do it."

Let's try out our New Class!

- Instantiating objects

```
# Declare and instantiate the two circle objects
big   = Circle.Circle()
small = Circle.Circle()

# Change both Circles from the default color
big.setColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
small.setColor(random.randint(0, 255), random.randint(0, 255)
255))

# Draw them on the screen
big.draw()
small.draw()
StdDraw.show(500)
```



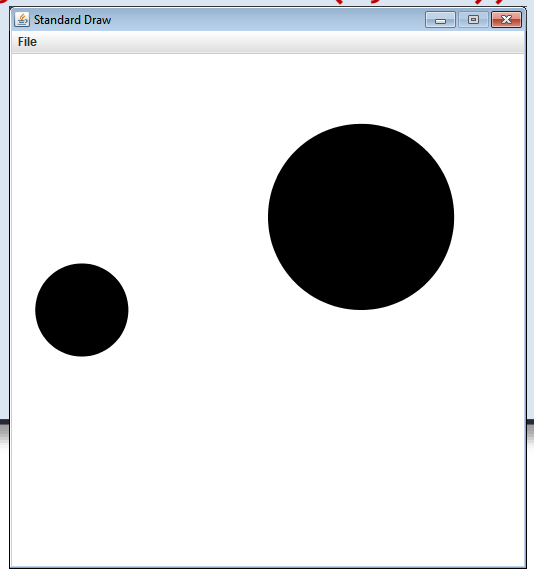

```
% python LimitedCircleClient.py
big: (0.0, 0.0) r = 0.0
small: (0.0, 0.0) r = 0.0
```

CircleClient Take Two

- Constructor called when we create objects

```
# Declare and instantiate the two circle objects
big   = Circle.Circle(0.7, 0.7, 0.2)
small = Circle.Circle(0.1, 0.5, 0.1)

# Change both Circles from the default color
big.setColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
small.setColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))

# Draw them on the screen
big.draw()
small.draw()

print("big: " + big.toString())
print("small: " + small.toString())

StdI
```

```
% python CircleClient.py
big: (0.7, 0.7) r = 0.2
small: (0.1, 0.5) r = 0.1
```

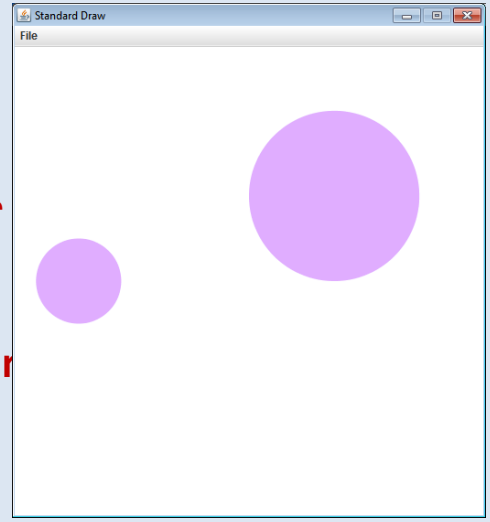


Colored Circles

- Goal: make each Circle object have a color specified by an red-green-blue (RGB) value
- Call `StdDraw.setPenColor()` in `draw()`
  - Create a new `Color` object for a given RGB value
  - `Color` is a class in the color library
  - Default color for our Circle objects: mauve

Circle in Living Color

```
import color

class Circle:

    # Construct a Circle object at the given (x, y) position with radius r
    def __init__(self, x = 0.0, y = 0.0, r = 0.0):
        self.posX   = x
        self.posY   = y
        self.radius = r
        self.color = color.Color(224, 173, 255)

    # Draw this Circle its location in its current color
    def draw(self):
        StdDraw.setPenColor(self.color)
        StdDraw.filledCircle(self.posX, self.posY, self.r
```

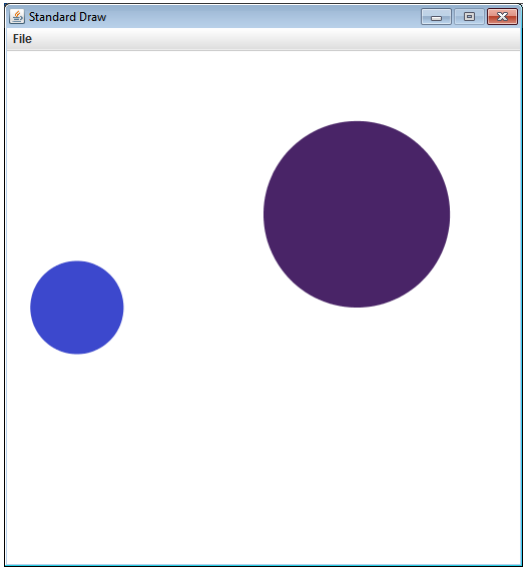


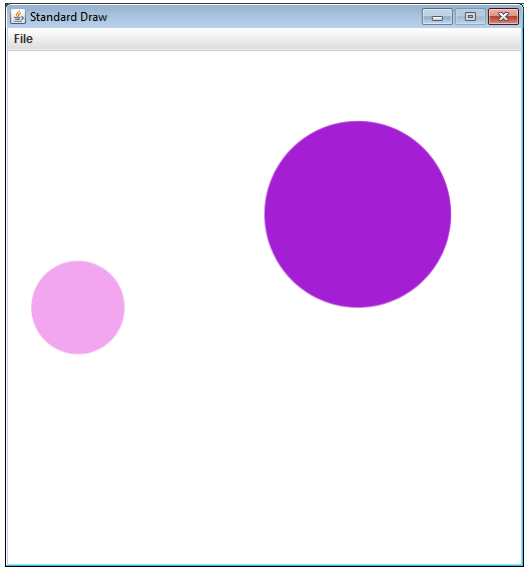
Allowing Clients to Change Color

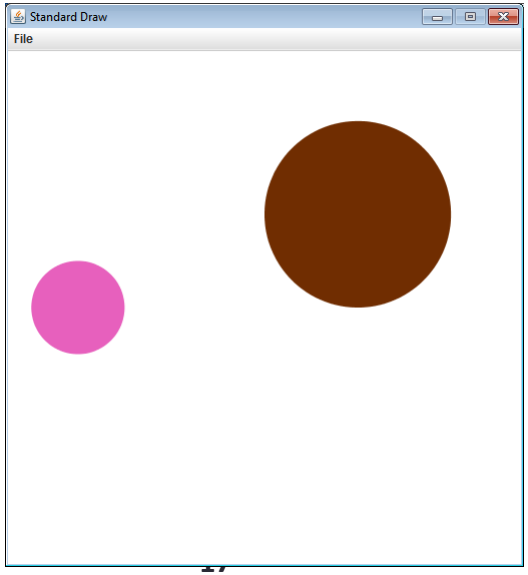
```
import color

class Circle:

    # Construct a Circle object at the given (x, y) position with radius r
    def __init__(self, x = 0.0, y = 0.0, r = 0.0):
        self.posX   = x
        self.posY   = y
        self.radius = r
        self.color = color.Color(224, 173, 255)

    # Draw this Circle its location in its current color
    def draw(self):
        StdDraw.setPenColor(self.color)
        StdDraw.filledCircle(self.posX, self.posY, self.radius)

    # Change the color of this Circle given an RGB pair where r, g, b are
    # in [0, 255]
    def setColor(self, r, g, b):
        self.color = color.Color(r, g, b)
```

Client Setting Random Color

```
import random

…

# Change both Circles from the default color
big.setColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
small.setColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
```



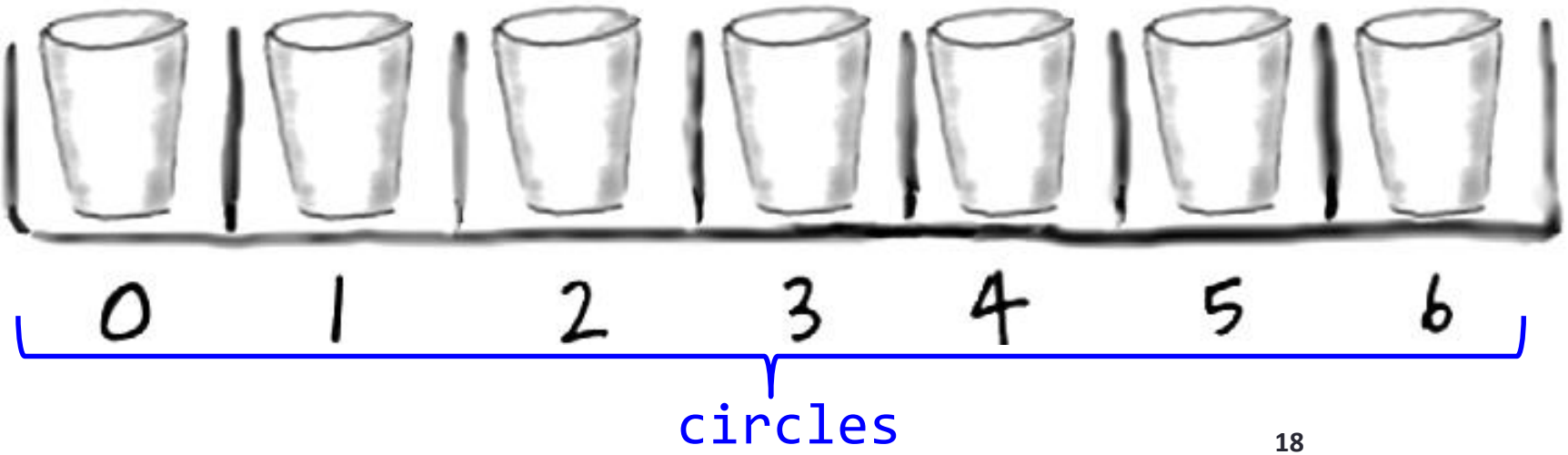




Creating Lots of Circles

- We can have an list of objects
- Step 1: create a list

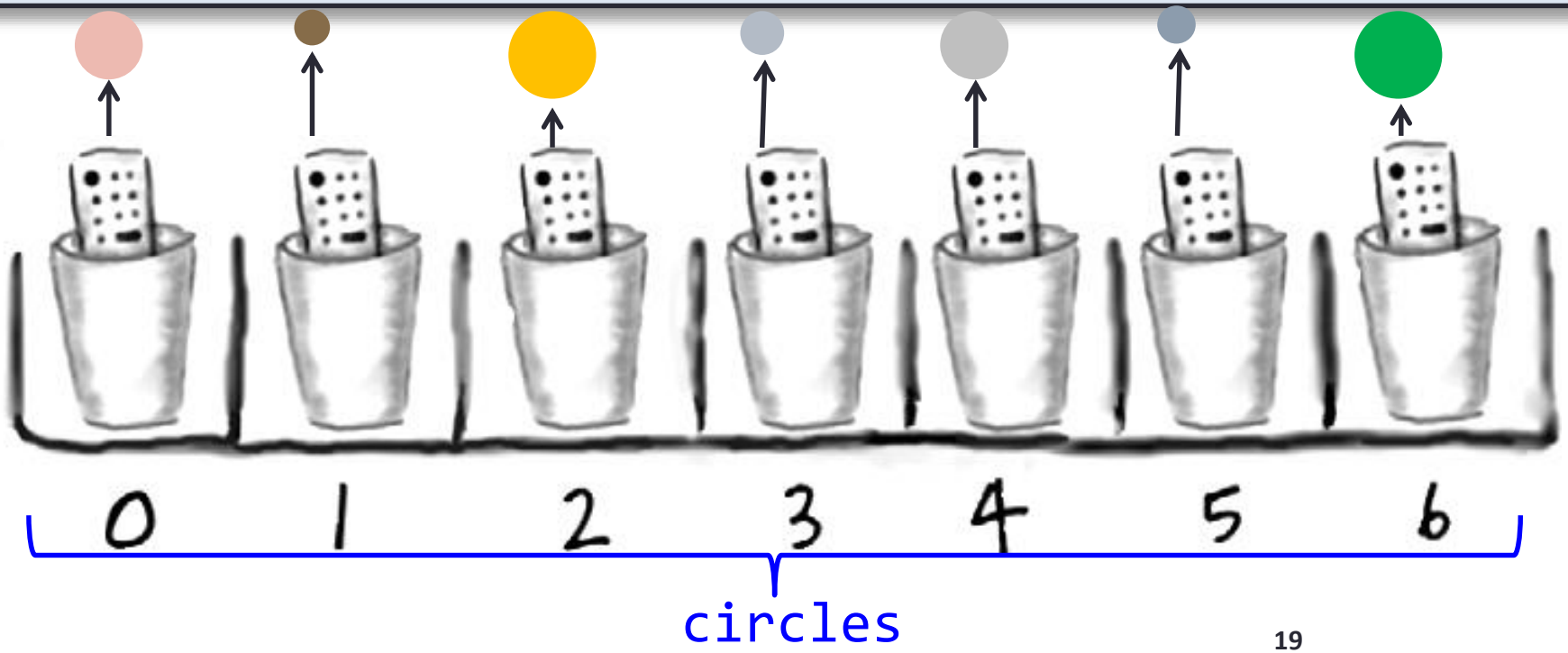
```
circles = []
```



Creating all the `Circle` Objects

- Each array location needs a new object

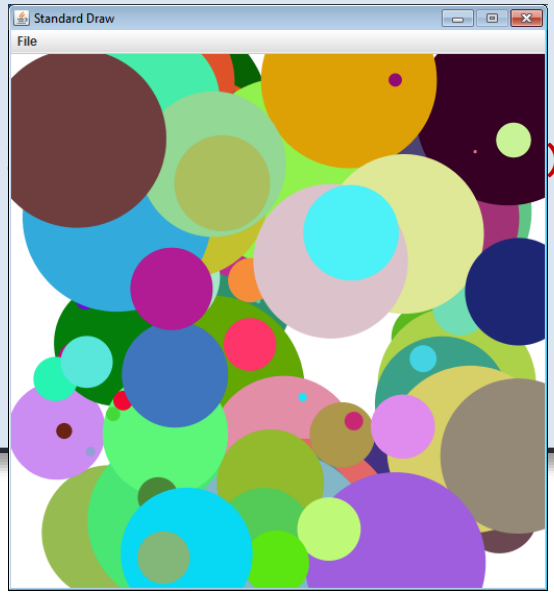
```
circles = []
for i in range(0, int(sys.argv[1])):
    circles.append(Circle.Circle(random.random(), random.random(), random.random()*0.2))
```



Client to Draw Lots of `Circle` Objects

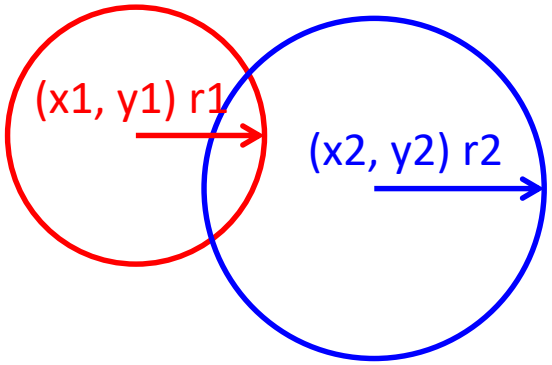
```
import Circle
import sys
import random
import StdDraw

# We create as many Circles as user specifies on command line
circles = []
for i in range(0, int(sys.argv[1])):
    circles.append(Circle.Circle(random.random(), random.random(), random.random()*0.2))

for circle in circles:
    # Set this new Circle to a random color
    circle.setColor(random.randint(0,255), random.randint(0,                    ))

    # Draw it on the screen
    circle.draw()
    StdDraw.show(50)
```

```
% python CircleClientDeluxe.py 100
```



Overlap Detection

- Goal: draw many `Circle` objects without overlap
  - When do two circles overlap?



Euclidean distance between centers:

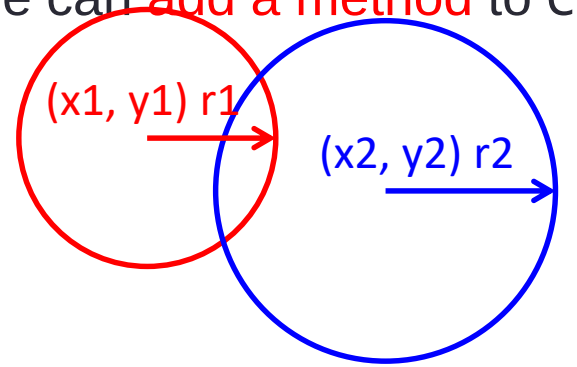
$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

Circles overlap if:

$d < (r1 + r2)$

Implementing Overlap Detection

- Overlap detection is something a `Circle` can do
  - We can add a method to `Circle` class for this!



Euclidean distance between centers:

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

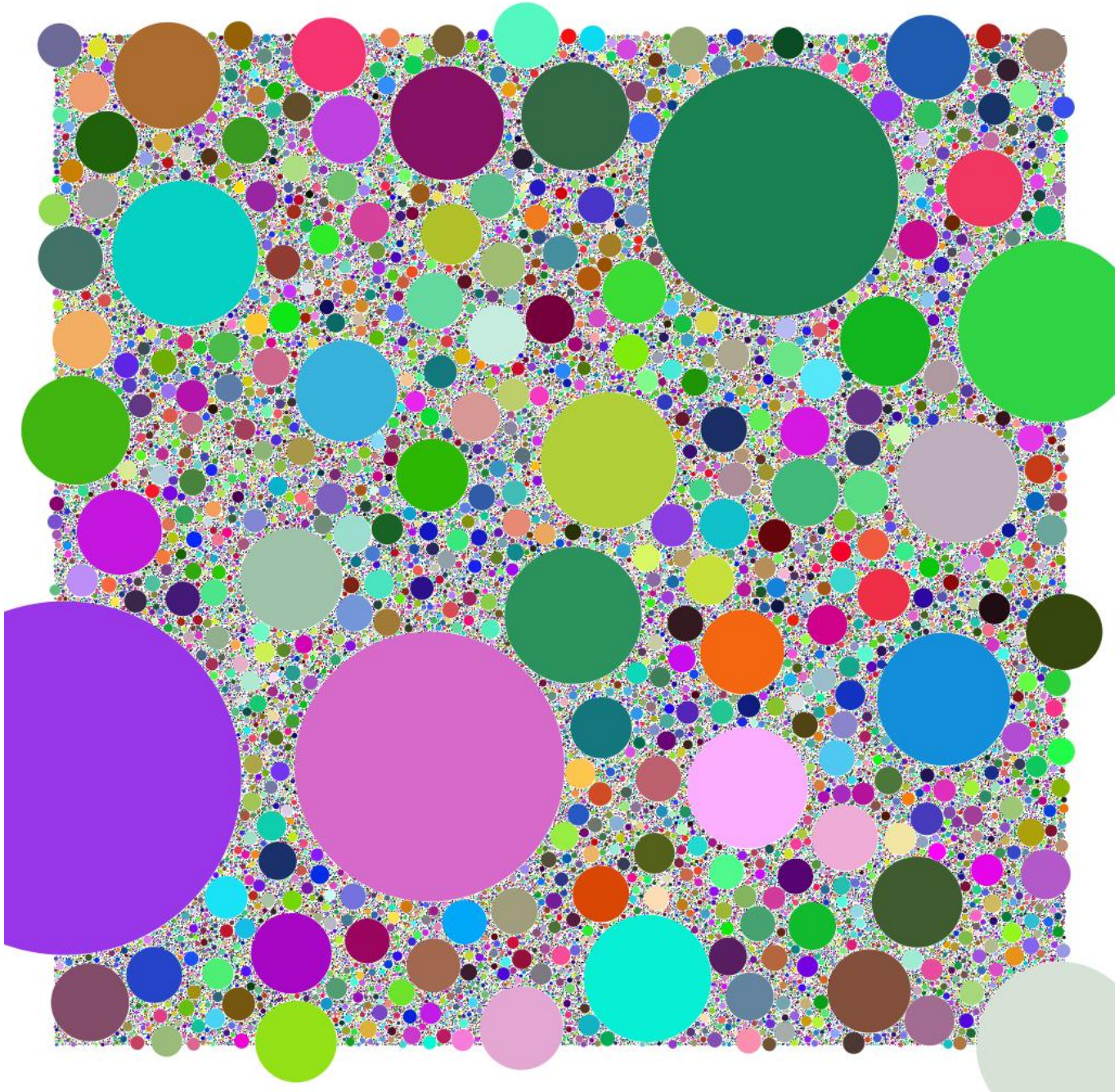
Circles overlap if:

$d < (r1 + r2)$

```
# Figure out if this circle and another circle overlap
   def overlap(self, other):
       deltaX = self.posX - other.posX
       deltaY = self.posY - other.posY
       d = math.sqrt(deltaX**2 + deltaY**2)
       if d < (self.radius + other.radius):
           return True
       return False
```

CircleClientSuperDeluxe

```
import Circle
import sys
import random
import StdDraw

def overlapAny(circles, circle):
    result = False
    for i in range(0, len(circles)):
        if circles[i].overlap(circle):
            return True
    return result

# We create as many Circles as user specifies on command line
circles = []
for i in range(0, int(sys.argv[1])):
    newCircle = Circle.Circle(random.random(), random.random(), random.random()*0.2)
    while overlapAny(circles, newCircle):
        newCircle = Circle.Circle(random.random(), random.random(), random.random()*0.1)
    newCircle.setColor(random.randint(0,255), random.randint(0,255), random.randint(0,255))
    circles.append(newCircle)
    newCircle.draw()
    StdDraw.show(10)
StdDraw.show(2000)
```



Recap

- Creating your own data types
  - Object-oriented programming (OOP)
  - Design classes encapsulating:
    - What objects know (“state”)
    - What objects can do (“behavior”)
  - Prevalent concept in most modern programming languages

Summary

- Simple types
- Creating your own data types
  - Classes
  - Objects
  - Constructors
  - Attributes
  - Methods
  - Lists of objects