CLASSES:

OVERLOADING

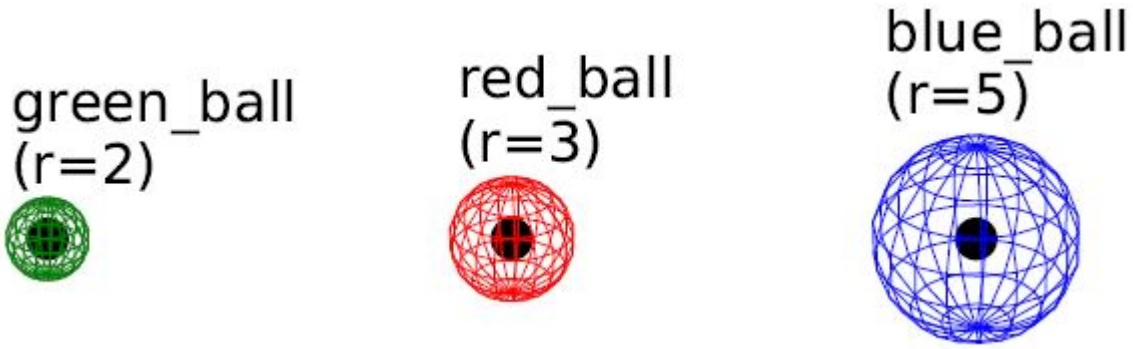
Overview:

- learn to write "magic" functions for overloading

Operator Overloading

- built-in types have common operators (+, <, ==)
- can override built-in methods
- how: special functions ("magic" methods)
- such functions start and end with __ (double underscore)
- example: to use '+', need to define $\_\_add\_\_()$

Overloading +

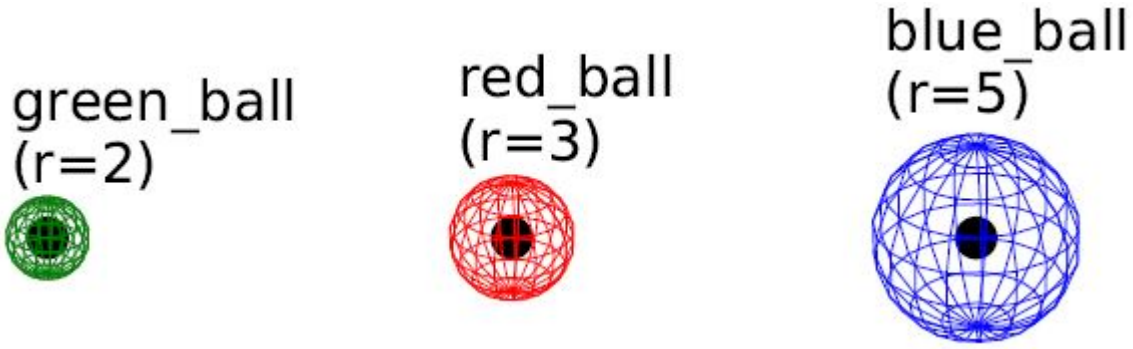


- need to define *__add__()*:

```
def __add__(self, other):
    return Sphere(self.__r + \
                  other.__r)
```

- add or compare objects like any data types

Overloading + (cont'd)




```
green_ball = Sphere(2)
red_ball   = Sphere(3)
blue_ball  = green_ball + red_ball
print(blue_ball)
```

```
sphere with radius 5
```

Overloading $<$, $==$

- need to define $\_\_lt\_\_()$ and $\_\_eq\_\_()$:

```
def __lt__(self, other):
    return (self.__r < other.__r)

def __eq__(self, other):
    return (self.__r == other.__r)
```

- polymorphic behavior for operators

```
green_ball = Sphere(2)
red_ball   = Sphere(3)
blue_ball  = green_ball + red_ball
print(green_ball < blue_ball)
print(red_ball == blue_ball)
```

```
True
False
```

Some Magic Methods

operator	method
+	__add__(self, other)
−	__sub__(self, other)
<	__lt__(self, other)
>	__gt__(self, other)
==	__eq__(self, other)
!=	__neq__(self, other)
&	__and__(self, other)
!	__or__(self, other)

Exercise(s):

- for the class *Circle* define the following:
  1. '+': new circle with $r = \max(r_1, r_2)$
  2. '−': new circle with $r = \min(r_1, r_2)$
  3. '==': True if both radii are the same
  4. '>': True if $r_1 > r_2$
- run the following script

```
circle_1 = circle(1
circle_2 = circle(2)
circle_3 = circle_1 + circle_2
print(circle_2 == circle_3)
print(circle_2 >  circle_1)
```