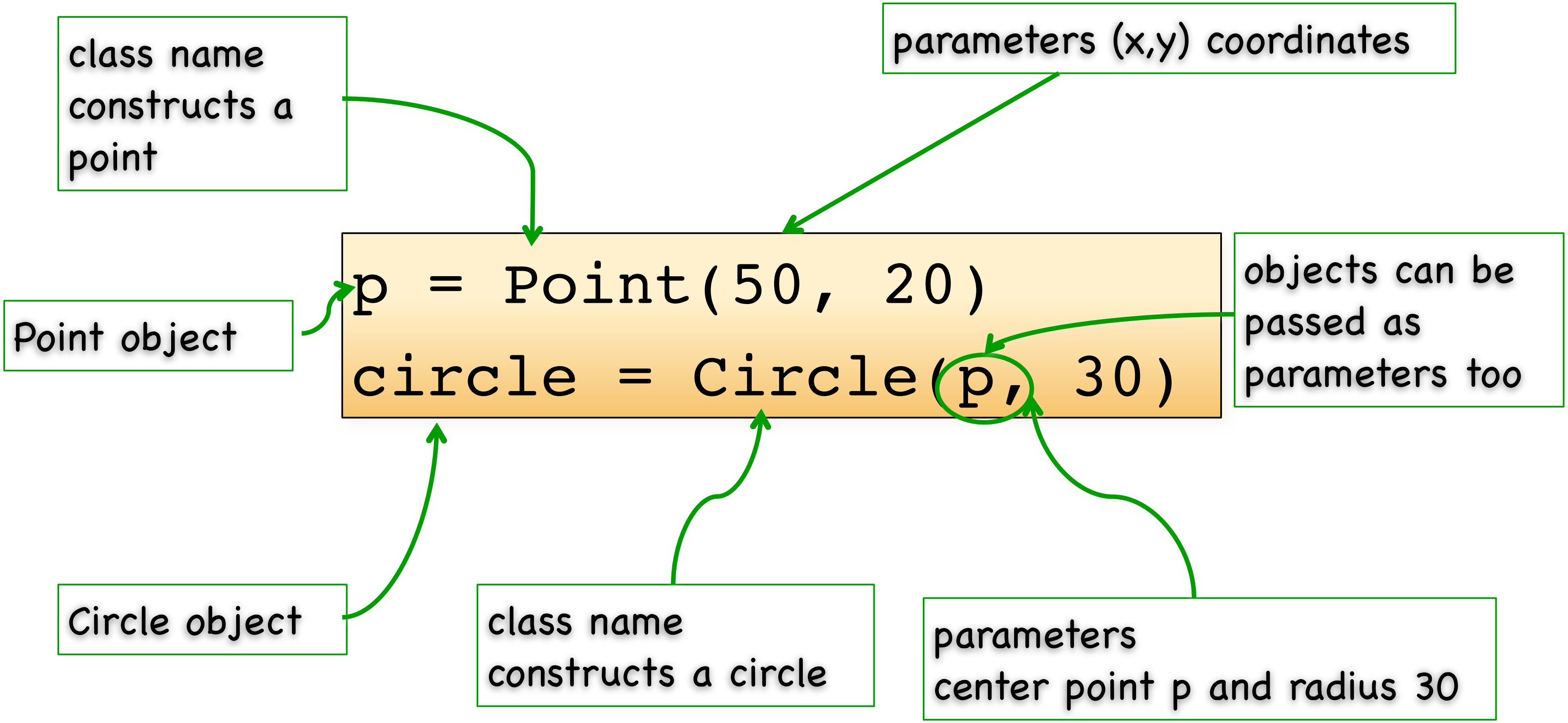
Graphics Objects

- Use graphics.py module
- Graphics objects available:
  - Point
  - Line
  - Circle
  - Oval
  - Rectangle
  - Polygon
  - Text

Creating an object



class name
constructs a
point

parameters (x,y) coordinates

Point object

```
p = Point(50, 20)
circle = Circle(p, 30)
```

objects can be
passed as
parameters too

Circle object

class name
constructs a circle

parameters
center point p and radius 30

Accessing Attributes and Methods

- Using dot (.)

```
p = Point(50, 20)
print p.x, p.y
print p.getX(), p.getY()
```

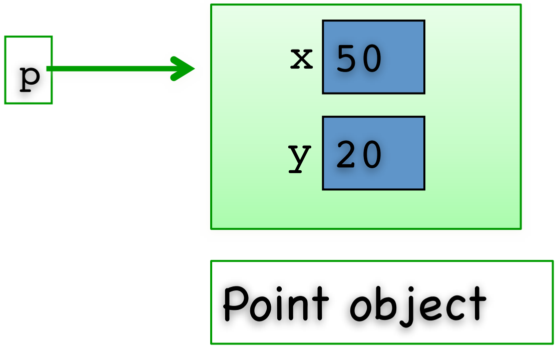
attributes or instance variables

```
50 20
50 20
```

methods to get the values of the entries

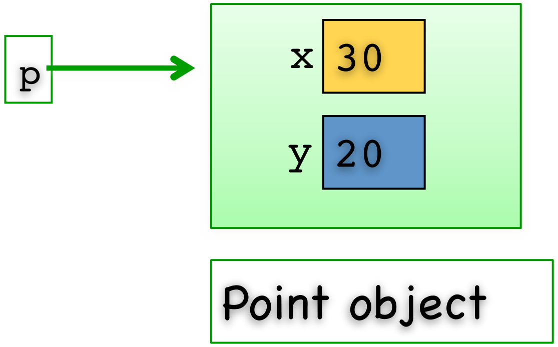
Objects are mutable

```
p = Point(50, 20)
p.x = p.x - 20
p2 = p
p2.x = p2.x + 10
print p.getX(), p.getY()
```



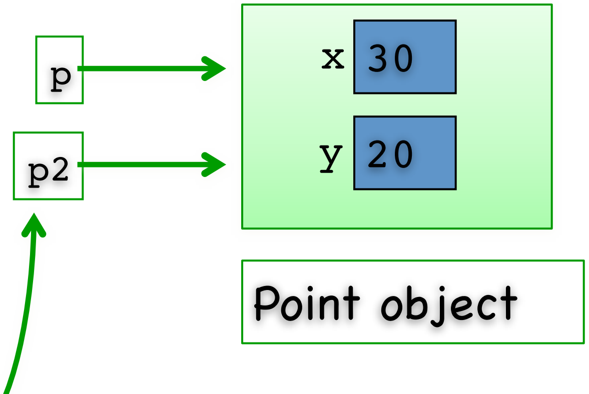
Objects are mutable

```
p = Point(50, 20)
p.x = p.x - 20
p2 = p
p2.x = p2.x + 10
print p.getX(), p.getY()
```



Objects are mutable

```
p = Point(50, 20)
p.x = p.x - 20
p2 = p
p2.x = p2.x + 10
print p.getX(), p.getY()
```

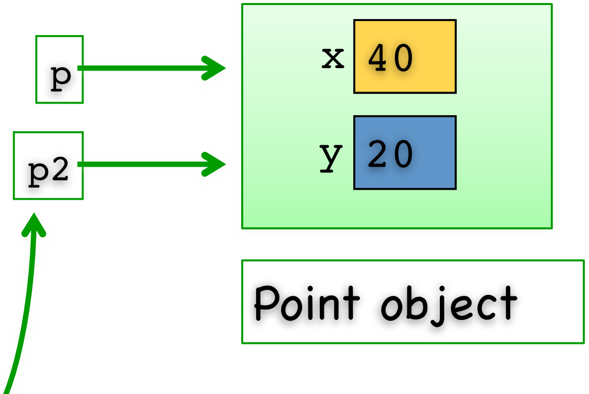


Point object

p2 is an alias of p, i.e. it refers to the same point object

Objects are mutable

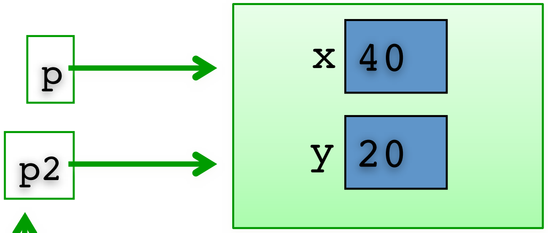
```
p = Point(50, 20)
p.x = p.x - 20
p2 = p
p2.x = p2.x + 10
print p.getX(), p.getY()
```



p2 is an alias of p, i.e. it refers to the same point object

Objects are mutable

```
p = Point(50, 20)
p.x = p.x - 20
p2 = p
p2.x = p2.x + 10
print p.getX(), p.getY()
```



Point object

p2 is an alias of p, i.e. it refers to the same point object

```
40 20
```

Scoping in functions

- Basic types – create a copy of the variable inside the function

```
def move_by_10(x, y):
   x = x + 10
   y = y + 10

x = 10
y = 10
move_by_10(x, y)
print x, y
```

What does this print?

10 10

Scoping in functions

- Objects – create an alias of the variable inside the function

```
def move_by_20(p):
   p.x = p.x + 20
   p.y = p.y + 20

p1 = Point(10, 10)
move_by_20(p1)
print p1.getX(), p1.getY()
```

creates an alias to the object that is passed as a parameter; **not** a copy of the object

What does this print?

30 30

Simple Graphics Program

graphics module
defines the graphics objects
we will use

```
from graphics import *

win = GraphWin('My Circle', 100, 100)
c = Circle(Point(50,50), 10)
c.setFill('red')
c.draw(win)

win.mainloop()
```

Simple Graphics Program

```
from graphics import *

win = GraphWin('My Circle', 150, 150)
c = Circle(Point(50,50), 10)
c.setFill('red')
c.draw(win)

win.mainloop()
```

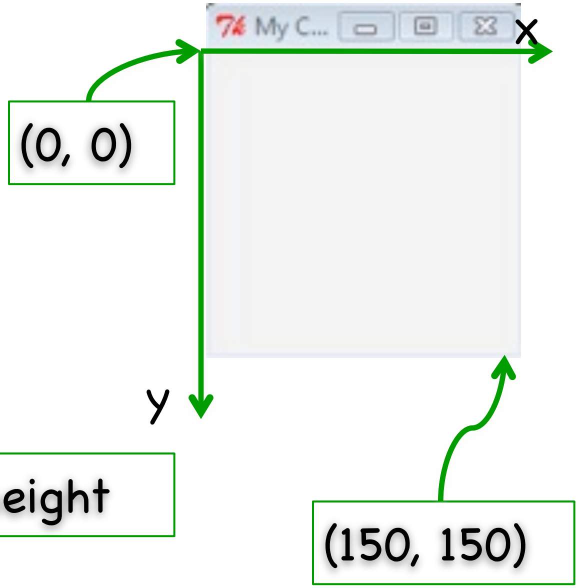
Creates a window with a canvas to draw on

Window title

Canvas width

Canvas height

Inverted coordinate system (units are pixels)



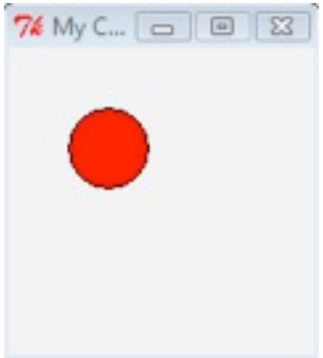
Simple Graphics Program

create a Circle object

```
from graphics import *

win = GraphWin('My Circle', 150, 150)
c = Circle(Point(50,50), 10)
c.setFill('red')
c.draw(win)

win.mainloop()
```

Circle center

Circle radius



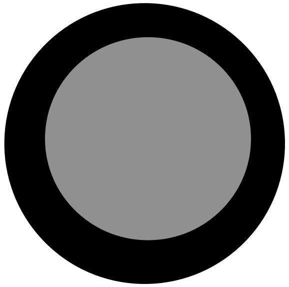
Simple Graphics Program

```
from graphics import *

win = GraphWin('My Circle', 150, 150)
c = Circle(Point(50,50), 10)
c.setFill('red')
c.draw(win)

win.mainloop()
```

every graphics program must end with this line;
it allows the window to process mouse clicks and keyboard input

User-defined types

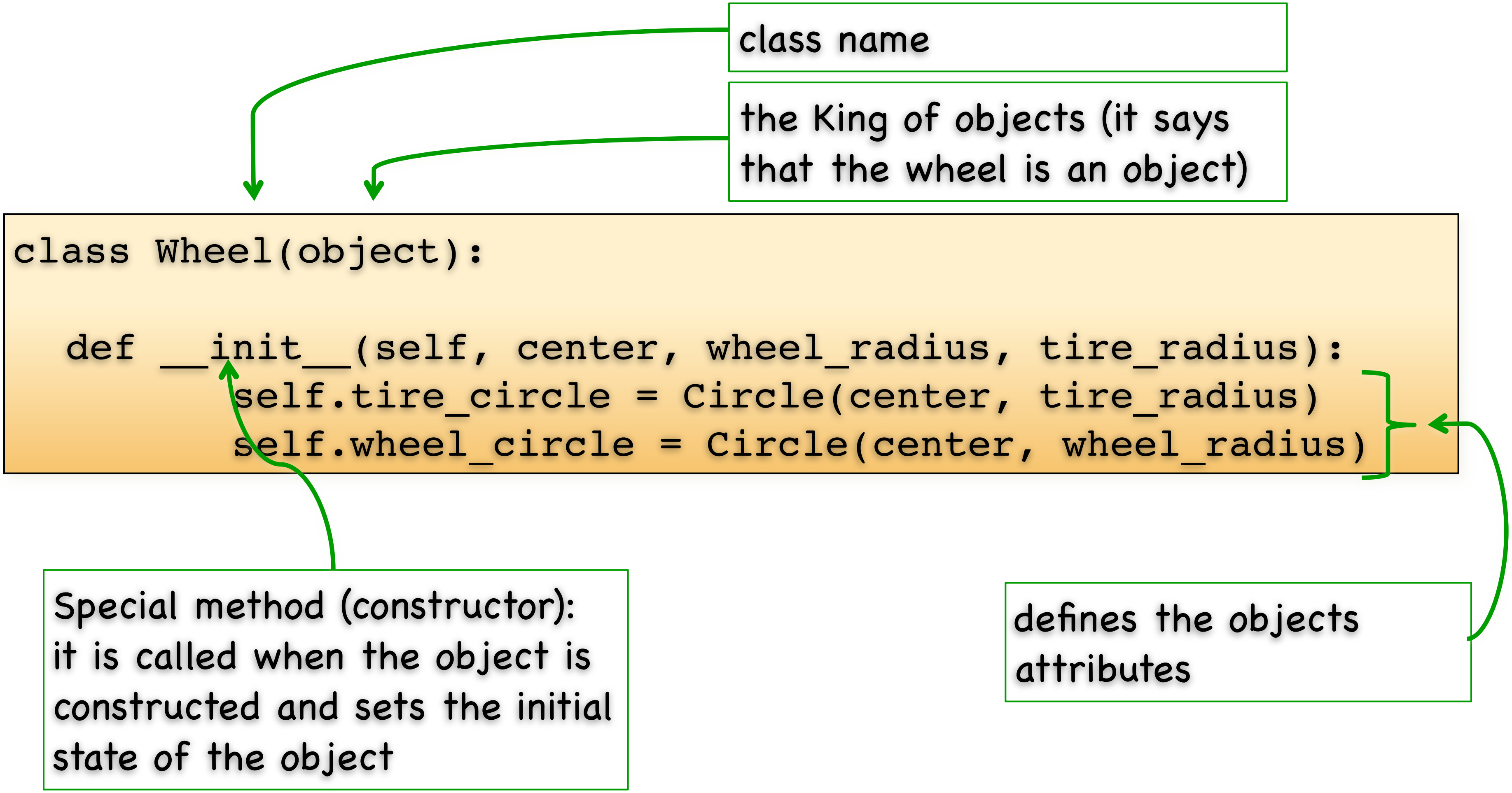
- What if we want to create our own class?
- E.g. let's create a class that draws a car wheel. For simplicity, the wheel will look like this:



Wheel class

- Attributes
  - `tire_circle`
  - `wheel_circle`
- Methods
  - `draw`
  - `move`
  - `get_size`
  - `get_center`
  - `set_color`

Wheel Class Definition



class name

the King of objects (it says that the wheel is an object)

```
class Wheel(object):

  def __init__(self, center, wheel_radius, tire_radius):
         self.tire_circle = Circle(center, tire_radius)
         self.wheel_circle = Circle(center, wheel_radius)
```

Special method (constructor): it is called when the object is constructed and sets the initial state of the object

defines the objects attributes

Wheel Class Definition

```
class Wheel(object):

  def __init__(self, center, wheel_radius, tire_radius):
         self.tire_circle = Circle(center, tire_radius)
         self.wheel_circle = Circle(center, wheel_radius)
```

- What is this `self` parameter?
- `self` is an alias to the object instance
- Must use it to access any of the object's attributes or methods
- it must always be the first parameter in a method signature

Wheel Class Definition

```
class Wheel(object):

  def __init__(self, center, wheel_radius, tire_radius):
        self.tire_circle = Circle(center, tire_radius)
        self.wheel_circle = Circle(center, wheel_radius)
```

Attributes are defined inside the __init__ method using the self parameter.

Attributes vs Local Variables

- Attribute
  - Defined in the __init__ method
  - Belongs to a specific object
  - Exists as long as the containing object exists
- Local variable
  - Declared within a method or a function
  - Exists only during the execution of its containing method or function

Wheel Class Definition

```
class Wheel(object):

  def __init__(self, center, wheel_radius, tire_radius):
         self.tire_circle = Circle(center, tire_radius)
         self.wheel_circle = Circle(center, wheel_radius)

  def draw(self, win):
        self.tire_circle.draw(win)
        self.wheel_circle.draw(win)

   def move(self, dx, dy):
        self.tire_circle.move(dx, dy)
        self.wheel_circle.move(dx, dy)
```

method definitions

Wheel Class Definition

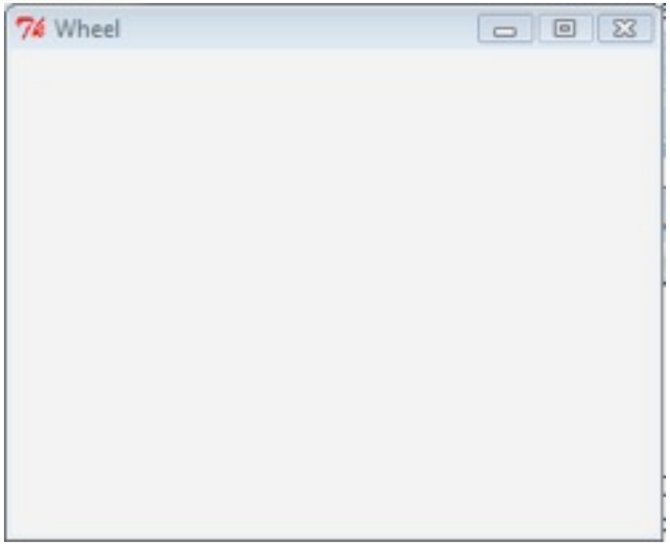
```
class Wheel(object):
   ''' This class defines a wheel template with two circles.
       Attributes: tire_circle, wheel_circle
   '''

   def __init__(self, center, wheel_radius, tire_radius):
         self.tire_circle = Circle(center, tire_radius)
         self.wheel_circle = Circle(center, wheel_radius)

   def draw(self, win):
         self.tire_circle.draw(win)
         self.wheel_circle.draw(win)

   def move(self, dx, dy):
         self.tire_circle.move(dx, dy)
         self.wheel_circle.move(dx, dy)

   def set_color(self, wheel_color, tire_color):
         self.tire_circle.setFill(tire_color)
         self.wheel_circle.setFill(wheel_color)
```

Wheel Class Definition

```
……. . .

def undraw(self):
      self.tire_circle.undraw()
      self.wheel_circle.undraw()

 def get_size(self):
      return self.tire_circle.getRadius()

def get_center(self):
       return tire_circle.getCenter()
```

Using our Wheel class

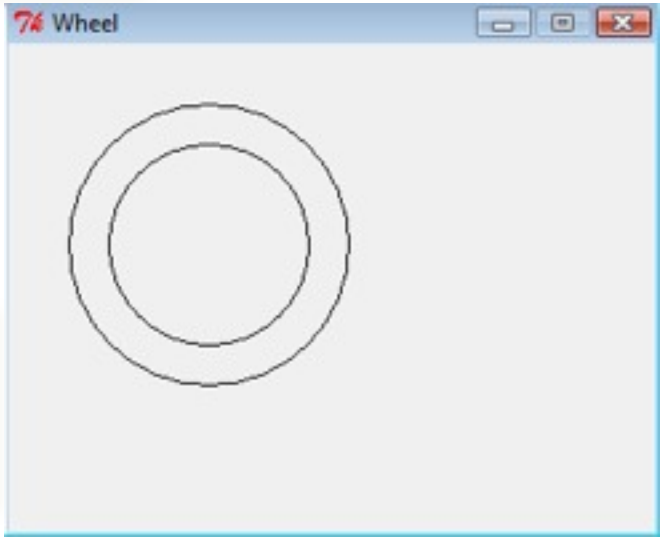
```
win = GraphWin('Wheel', 320, 240)
w = Wheel(Point(100, 100), 50, 70)
w.draw(win)
w.set_color('gray', 'black')
w.undraw()
win.mainloop()
```



Using our Wheel class

```
win = GraphWin('Wheel', 320, 240)
w = Wheel(Point(100, 100), 50, 70)
w.draw(win)
w.set_color('gray', 'black')
w.undraw()
win.mainloop()
```

What happened to the mysterious self parameter?

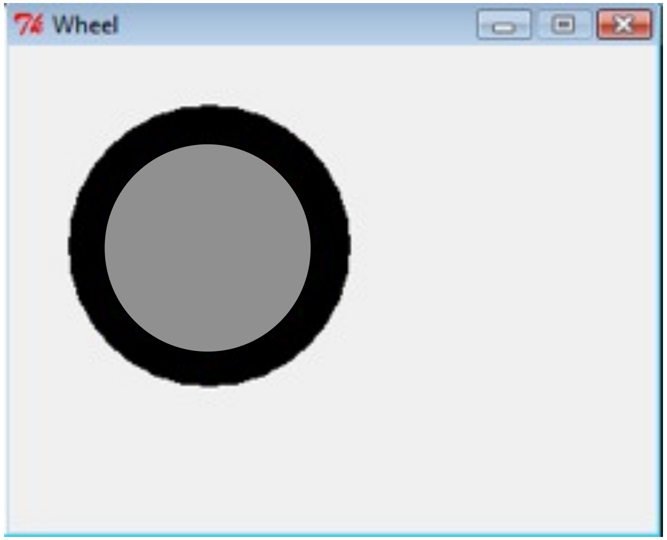


```
self = w
```

```
def draw(self, win):
   self.tire_circle.draw(win)
   self.wheel_circle.draw(win)
```

Using our Wheel class

```
win = GraphWin('Wheel', 320, 240)
w = Wheel(Point(100, 100), 50, 70)
w.draw(win)
w.set_color('gray', 'black')
w.undraw()
win.mainloop()
```



Using our Wheel class

```
win = GraphWin('Wheel', 320, 240)
w = Wheel(Point(100, 100), 50, 70)
w.draw(win)
w.set_color('gray', 'black')
w.undraw()
win.mainloop()
```