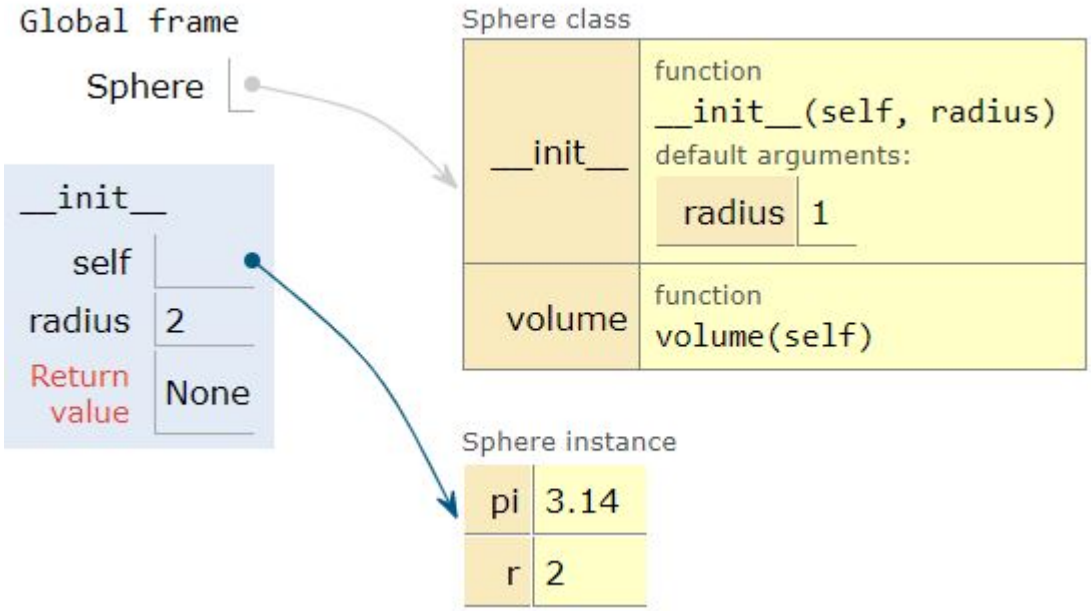
CLASSES:

STATIC
vs.
INSTANCE VARIABLES

Overview:

- distinguish static vs. instance variables
- learn to write "magic" functions for overloading
- define new classes with inheritance
- learn multiple inheritance and abstract classes

self Parameter



- each instance is passed *self* parameter
- similar to *this* in Java/C++
- allows object to keep its own data

Static vs. Instance Variables

- r, pi: instance variables
- each instance has its own copy
- pi is the same across instances
- need to make pi static
- static: single copy per class
- how: define before methods
- access as Class.Name

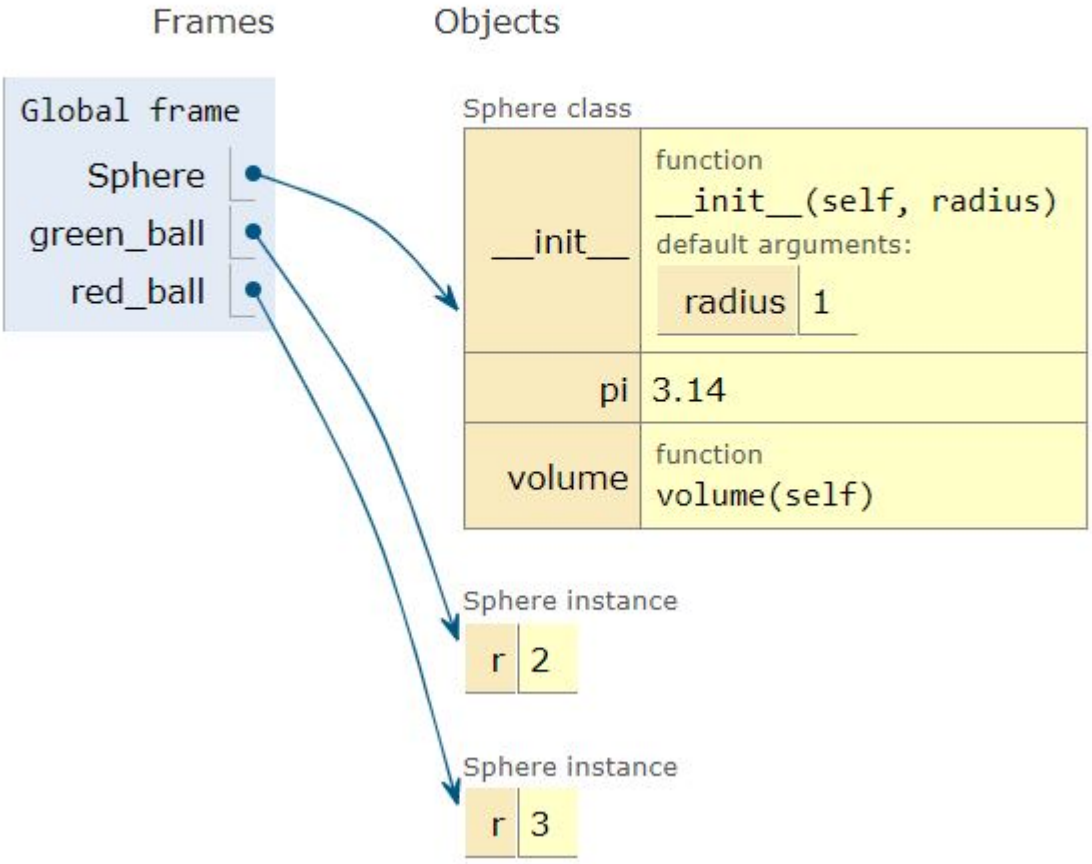
Static Variables

```
class Sphere():
    pi = 3.14                # static

    def __init__(self, radius = 1):
        self.r = radius  # instance

    def volume(self):
        return 4 * Sphere.pi * self.r**3 / 3

green_ball  = Sphere(2)
red_ball    = Sphere(3)
print('green_ball volume:', green_ball.volume())
print('red_ball   volume:', red_ball.volume())
```

- a single copy of *pi* is shared

Example

```
green_ball volume: 33.49
red_ball   volume: 113.04
```



Static **vs.** *Non-Static*

- non-static

```
class Sphere():
    def __init__(self, radius = 1):
        self.pi = 3.14     # instance
        self.r  = radius   # instance

    def volume(self):
        return 4 * self.pi * self.r**3 / 3
```

- static

```
class Sphere():
    pi = 3.14              # static

    def __init__(self, radius = 1):
        self.r = radius    # instance

    def volume(self):
        return 4 * Sphere.pi * self.r**3 / 3
```

Describing Objects

```
class Sphere():
    pi = 3.14

    def __init__(self, radius = 1):
        self.r = radius

    def volume(self):
        return 4 * Sphere.pi * self.r**3 / 3

green_ball = Sphere(2)
print(green_ball)
```

Print output (drag lower right corner to resize)

```
<__main__.Sphere object at 0x7fd1e4b9fb70>
```

- want to give "human" description

__str__() Method

- user-defined description

```
class Sphere():
    pi = 3.14

    def __init__(self, radius = 1):
        self.r = radius

    def __str__(self):
        return 'sphere with radius {}'\
                .format(self.r)

    def volume(self):
        return 4 * Sphere.pi * self.r**3 / 3

green_ball = Sphere(2)
print(green_ball)
```

Print output (drag lower right corner to resize)

```
sphere with radius 2
```

__str__() and *__repr__*()

```
green_ball  = Sphere(2)
print(repr(green_ball))
print(green_ball)
```

Print output (drag lower right corner to resize)

```
<__main__.Sphere object at 0x7fb16f911748>
sphere with radius 2
```

- *__repr__*(): "official" object description
- *__str__*(): "human" object description
- *__str__*() uses *__repr__*() as a fall-back

class Template

```
class Sphere():              # class name

    pi = 3.14                # static data field(s)

    def __init__():          # constructor

    def __str__(self):       # representation

    def volume():            # method(s)
```

- classes use "dot" notation

```
green_ball  = Sphere(2)
volume_1 = green_ball.volume()
```

- all variables are public

Exercise(s):

- make *pi* to be static in your class
- write *__str__()* method for the *Circle* class