CLASSES: DATA ENCAPSULATION

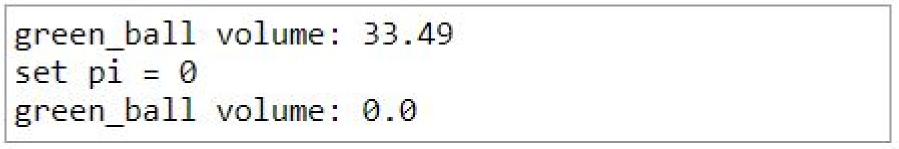
Overview:

- learn name mangling for class variables

Data Privacy

- no mechanism for privacy

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

Sphere.pi = 0
print('set pi = 0 ')
print('green_ball volume:', green_ball.volume())
```



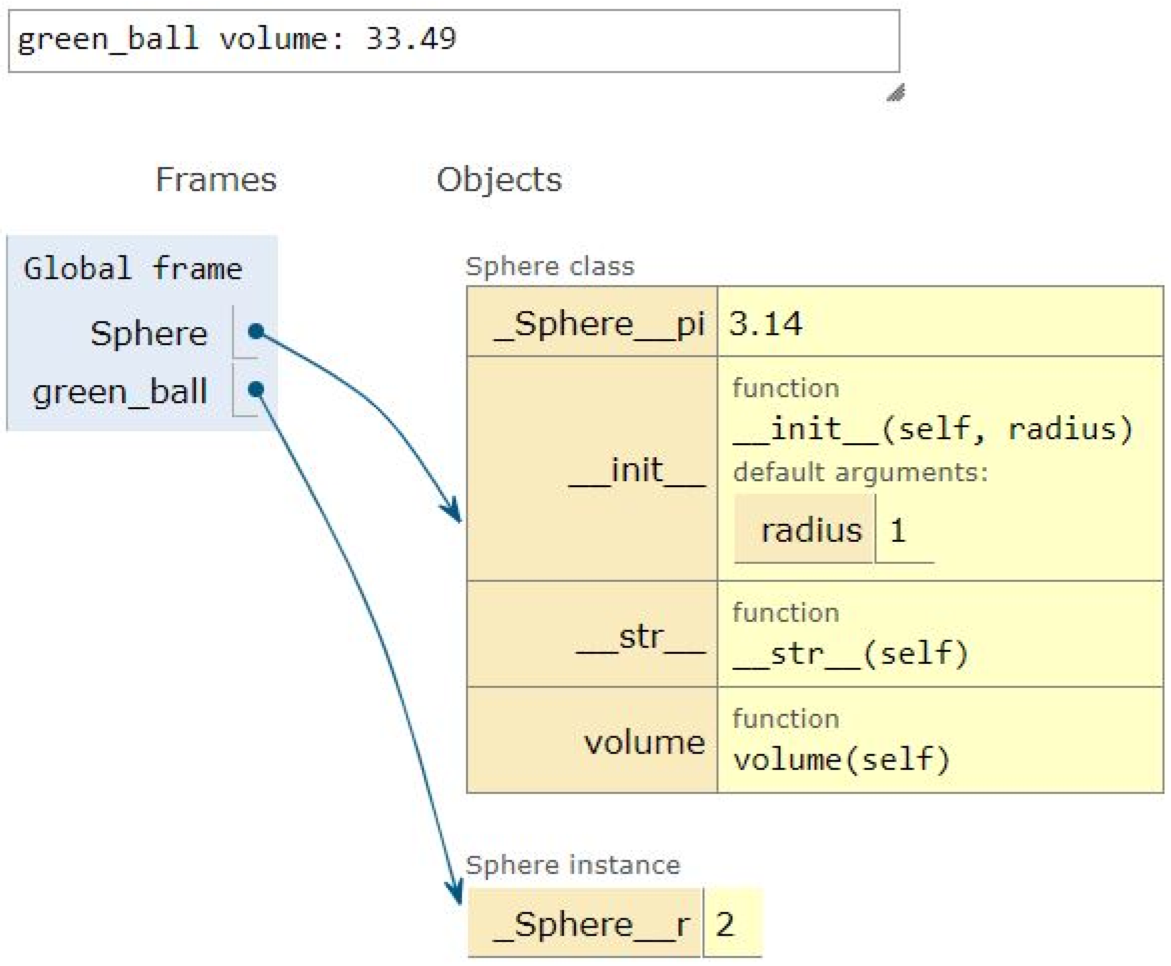
```
green_ball volume: 33.49
set pi = 0
green_ball volume: 0.0
```

- solution: ”name mangling”
- how? *pi* $\mapsto$ *__pi* & *r* $\mapsto$ *__r*
- Python creates new names: *_Sphere__pi* & *_Sphere__r*

Name Mangling

```
class Sphere():
    __pi = 3.14                 # name mangling

    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 volume:', round(green_ball.volu

# code below will now generate an error
Sphere.pi = 0

# in theory, can still set it to 0
Sphere._Sphere__pi = 0
```

Name Mangling



```
green_ball volume: 33.49
```

- prevents accidental change

Accessing and Setting Class variables

- data encapsulation
- expose instance variables by methods

1. *accessors:* return values
2. *mutators:* set or change variables

Modified Class Example

```
class Sphere():
    __pi = 3.14                  # name mangling

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

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

    def set_radius(self, r):     # mutator
        self.__r = r             # (setter)

    def get_radius(self):        # accessor
        return self.__r          # (getter)

    def volume(self):
        return 4 * Sphere.__pi*self.__r**3 / 3
```

Exercise(s):

- apply "name mangling" to class variables in *Circle* class