CLASSES

MULTIPLE INHERITANCE & ABSTRACT CLASSES

Overview:

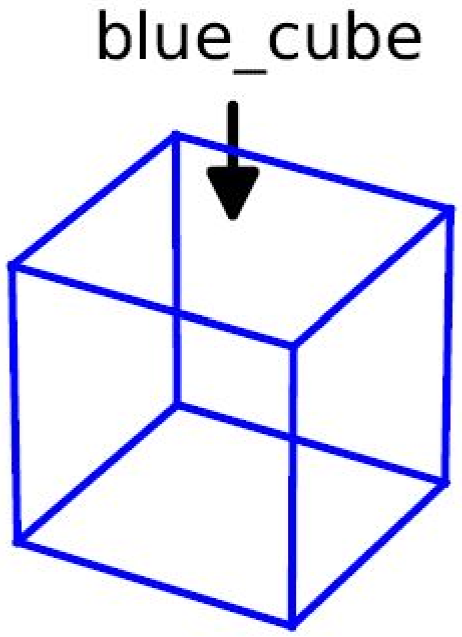
- learn multiple inheritance and abstract classes

Cube **Class**

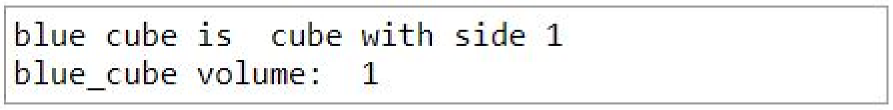
```
class Cube:
    def __init__(self, side = 1):
        self.__side = side

    def __str__(self):
        return 'cube with side {}'\
        .format(self.__side)

    def volume(self):
        return self.__side ** 3
```

- similar to *Sphere* class
- methods: *volume*() and *__str__*()

Cube **Class**



```
blue_cube = Cube(1)
red_ball = Sphere(1)
print('blue cube is ', blue_cube)
print('red ball  is ', red_ball)
print('blue_cube volume: ', blue_cube.volume())
print('red ball  volume: ', red_ball.volume())
```

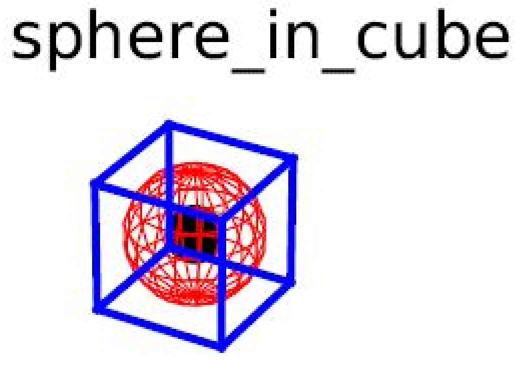


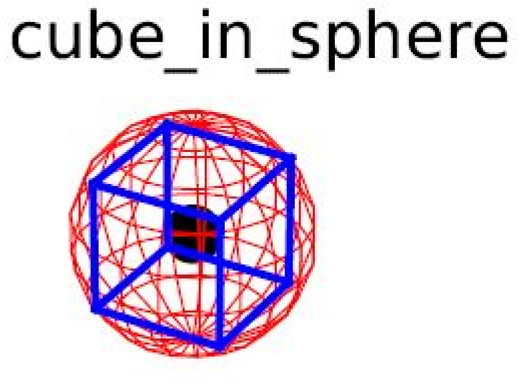
```
blue cube is  cube with side 1
blue_cube volume:  1
```

Multiple Inheritance

- allowed in Python (as in C++)
- not allowed in Java/C
- subclass inherits methods from all parents

```
class Sphere_In_Cube(Cube, Sphere)
          -----
class Cube_in_Sphere(Sphere, Cube)
          -----
```





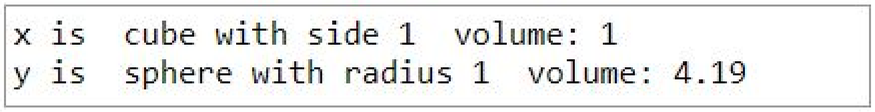
Class Details

```
import math

class Cube_in_Sphere(Cube, Sphere):
    def __init__(self, side =1):
        radius = side * math.sqrt(3.0)
        Cube.__init__(self, side)
        Sphere.__init__(self, radius)

class Sphere_in_Cube(Sphere, Cube):
    def __init__(self, side = 1):
        Cube.__init__(self, side)
        Sphere.__init__(self, side)
```

- inherit all methods from all parents classes
- how to resolve name conflicts?

Which Method is Used?

```
x = Cube_in_Sphere(1)
y = Sphere_in_Cube(1)
print('x is ',x, ' volume:', x.volume())
print('y is ',y, ' volume:', y.volume())
```



```
x is  cube with side 1  volume: 1
y is  sphere with radius 1  volume: 4.19
```

- determined by order of parent classes

  ```
  class Cube_in_Sphere(Cube, Sphere):
  class Sphere_in_Cube(Sphere, Cube)
  ```

- first found method is used

Abstract Class

- new class *Shape*
- contains *Sphere* and *Sphere*
- may add other classes like *Cylinder* or *Pyramid*
- can define it as abstract class

```
class Shape:
    def __init__(self, r):
            self.r = r

    def volume(self):
        raise NotImplementedError

class Cube(Shape):
class Sphere(Shape):
class Cylinder(Shape):
class Pyramid(shape):
```

Exercise(s):

- define a class *Cylinder* derived from both *Cube* and *Circle*
- base has radius r from circle
- height h is *side* from cube
- write code for its volume and (surface) area

$$\begin{aligned} \text{Volume} &= \pi r^2 h \\ \text{Area} &= 2\pi r(r+h) \end{aligned}$$

Summary:

- classes are user-defined types
- classes have methods
- details of implementation are hidden
- can *overload* standard operators (”magic” functions)
- new classes can be constructed through inheritance
- same functionality via *polymorphism*