Strings in Python

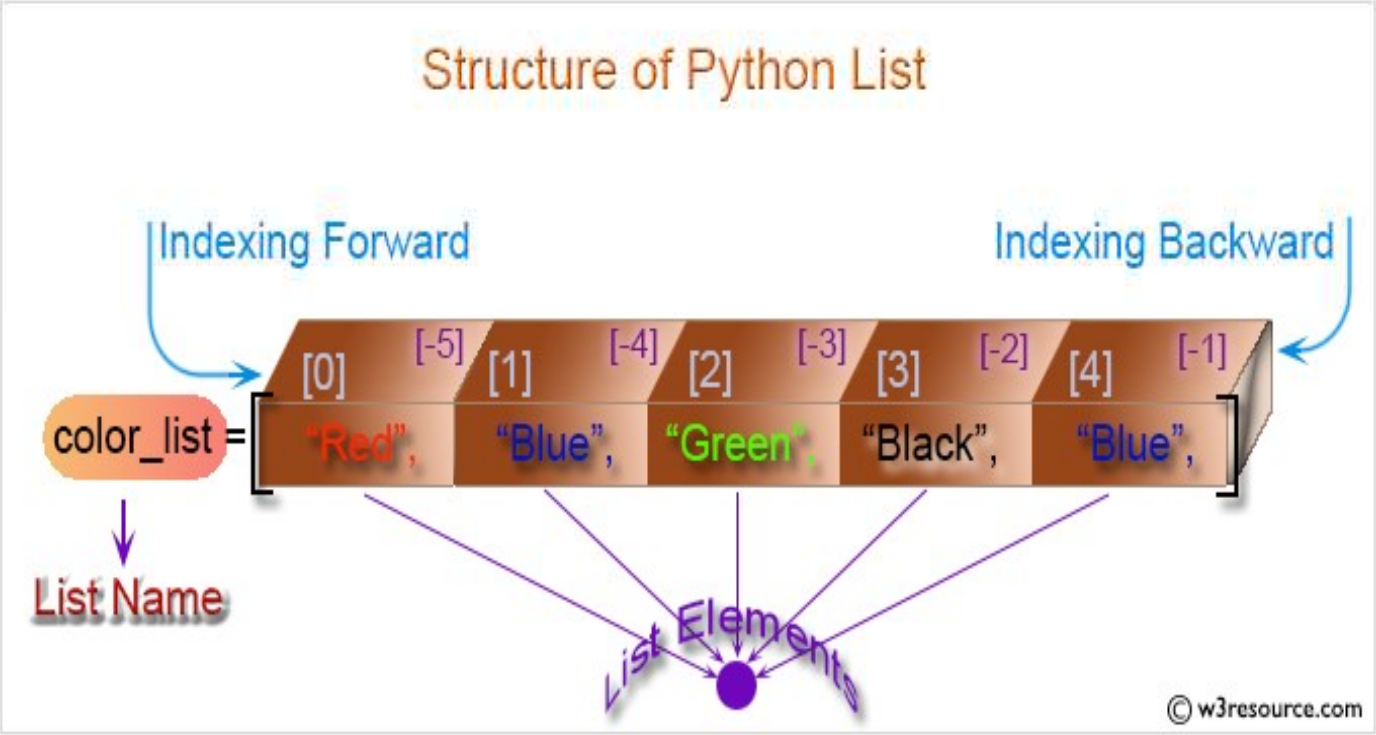
str="BANGALORE"

B	A	N	G	A	L	O	R	E
0	1	2	3	4	5	6	7	8

Accessing string elements
Str[0] returns B
Str[1] returns A
---

count()
format()
index()
islower()
join()
lower()
upper()
strip()
replace()
split()

Lists in Python



Dict in Python

Dict_object.method()

Dict.clear()

Dict.copy():

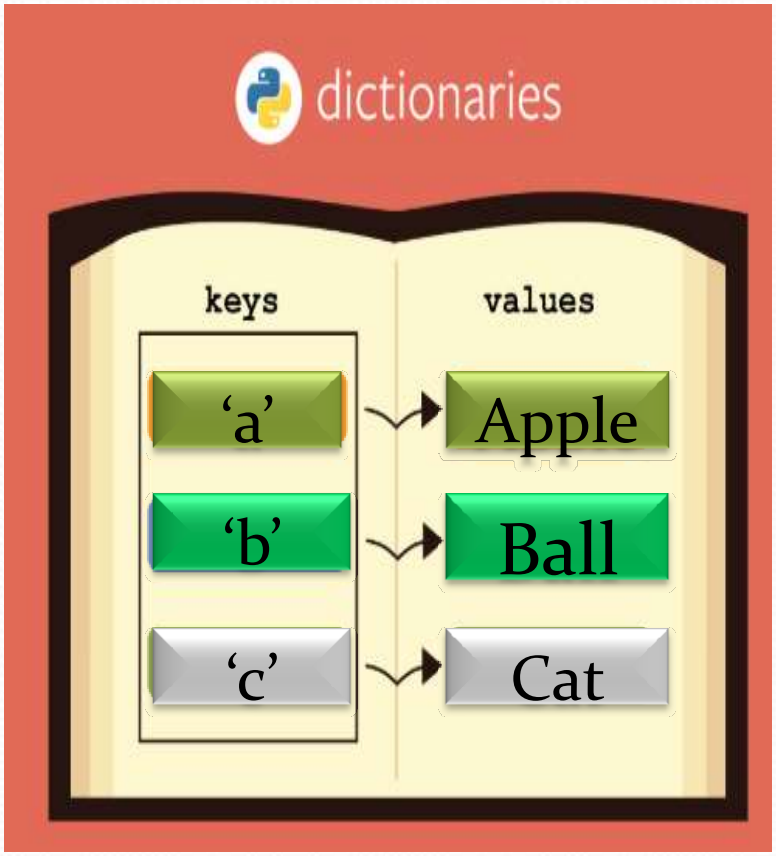
Dict.get(key)

Dict.items():

Dict.keys():

Dict.update();

Dict.values():



Classes in Python: User defined data types

```
class class_name :
        data members
        methods
```

```
Example:
class Circle
        radius
        findArea()
        findPerimeter()
```

- Classes provide a means of bundling data and functions together.
- it is a collection of variables and functions
- variables are called as data members and
- Function are called as member functions or **methods**
- Classes are used to represent real world entities

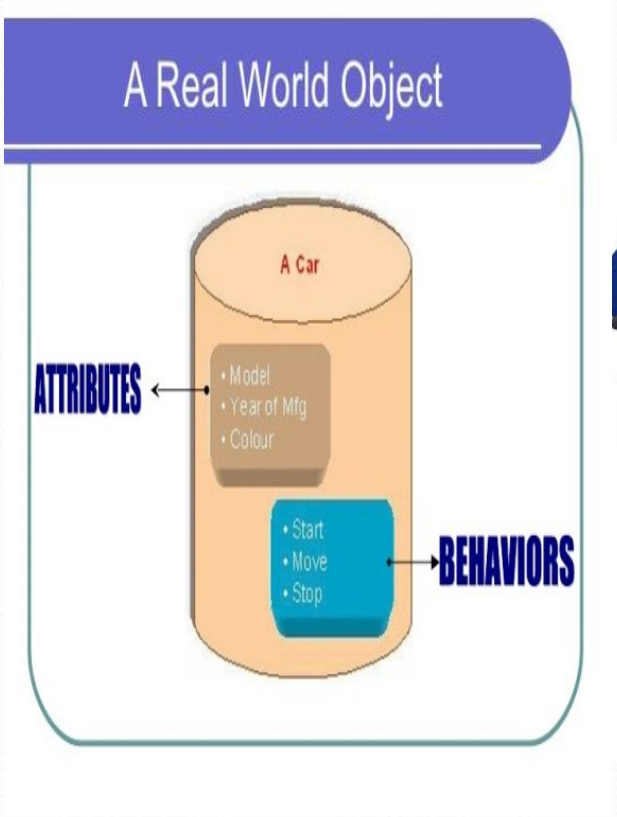
Real world objects/entities have two major things

1)state/attributes(what it is)

2)Behaviour/Actions(what it does)

Python classes can be used to simulate real world entities

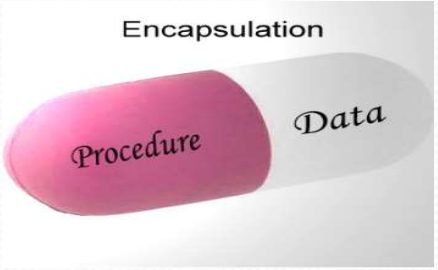
```
class car :
        year; make;
        speed;

        start()
        accelerate();
        brake();


class flower :
        name;
        color;

        makegarlend()
```



Object-Oriented Programming

Python is an object-oriented programming language

- **Encapsulation**
  (security to data)



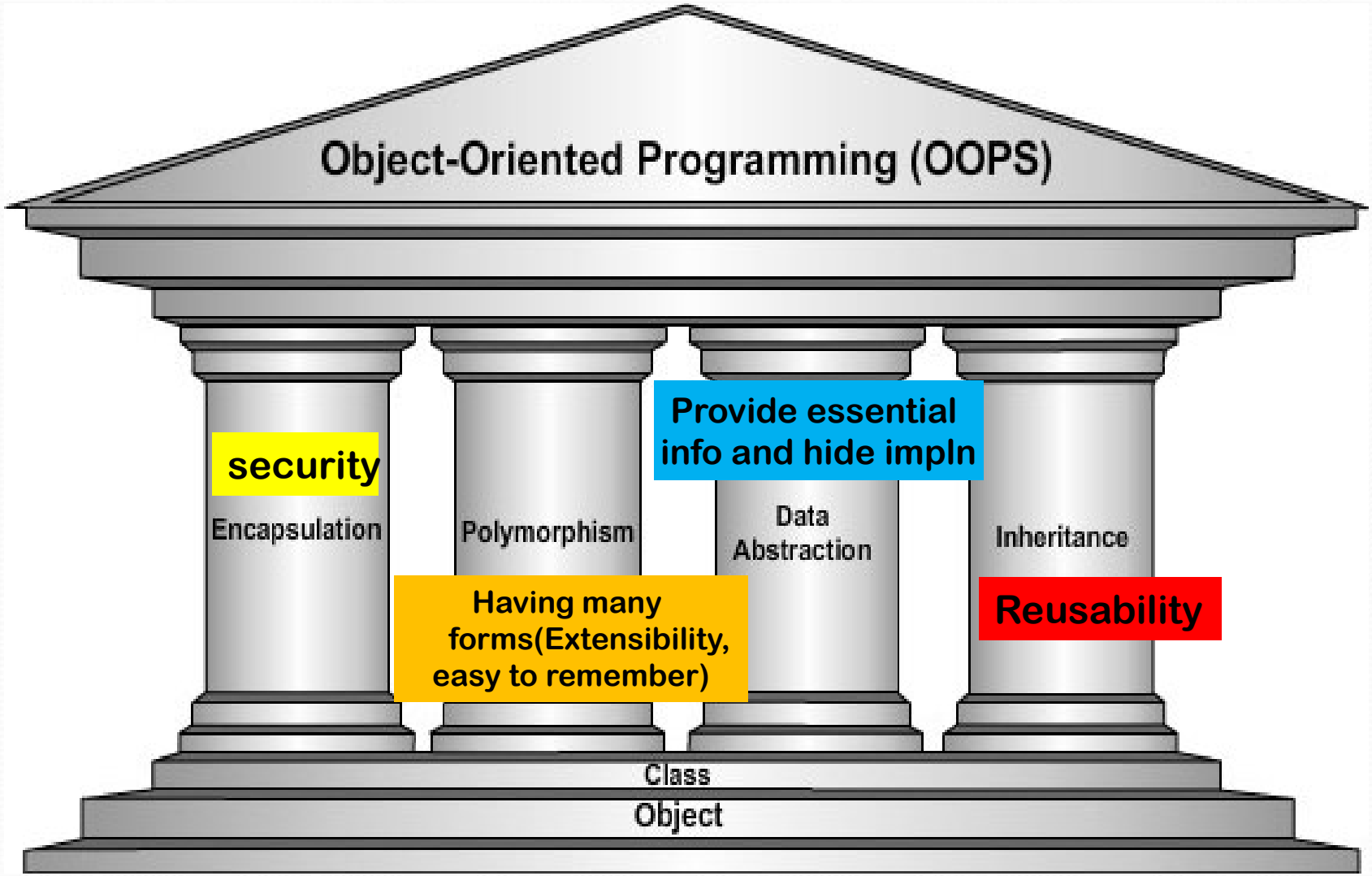
- **Inheritance**
  (reusability)

- **Polymorphism**
  (Having many forms)

These are also called as

pillars of object-oriented development

Pillars of object-oriented development



Classes: user defined data types

```
class person:
    def __init__(self,x,y):
        self.name=x
        self.age=y
    def display(self):
        print(self.name,self.age)

p1=person("john",30)
p2=person("Ram",32)

print(p1.name,p1.age)
print(p2.name,p2.age)

p1.display()
```

Classes in Python

Class is a blueprint of a house

Objects

Person class

```
class person:
    def __init__(self,x,y):
        self.name=x
        self.age=y
    def display(self):
        print(self.name,self.age)
```

p1=person("john",30)

p2=person("Ram",32)

WAP to find area and perimeter of a circle using classes

```
class Circle:
    def __init__(self,r):
        self.radius=r
    def findarea(self):
        print(3.14*self.radius*self.radius)
    def findperimeter(self):
        print(2*3.14*self.radius)


c=Circle(1)
c.findarea()
c.findperimeter()
```

Using classes, write a program to find distance between two points

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def disp(self):
        print ("x : ", self.x)
        print ("y : ", self.y)
    def findDistance(p1,p2):
        res=pow(p2.x-p1.x,2)+pow(p2.y-p1.y,2)
        print(math.sqrt(res))

p1 = Point(3, 4)
p2= Point(4, 3)
p1.findDistance(p2)
```

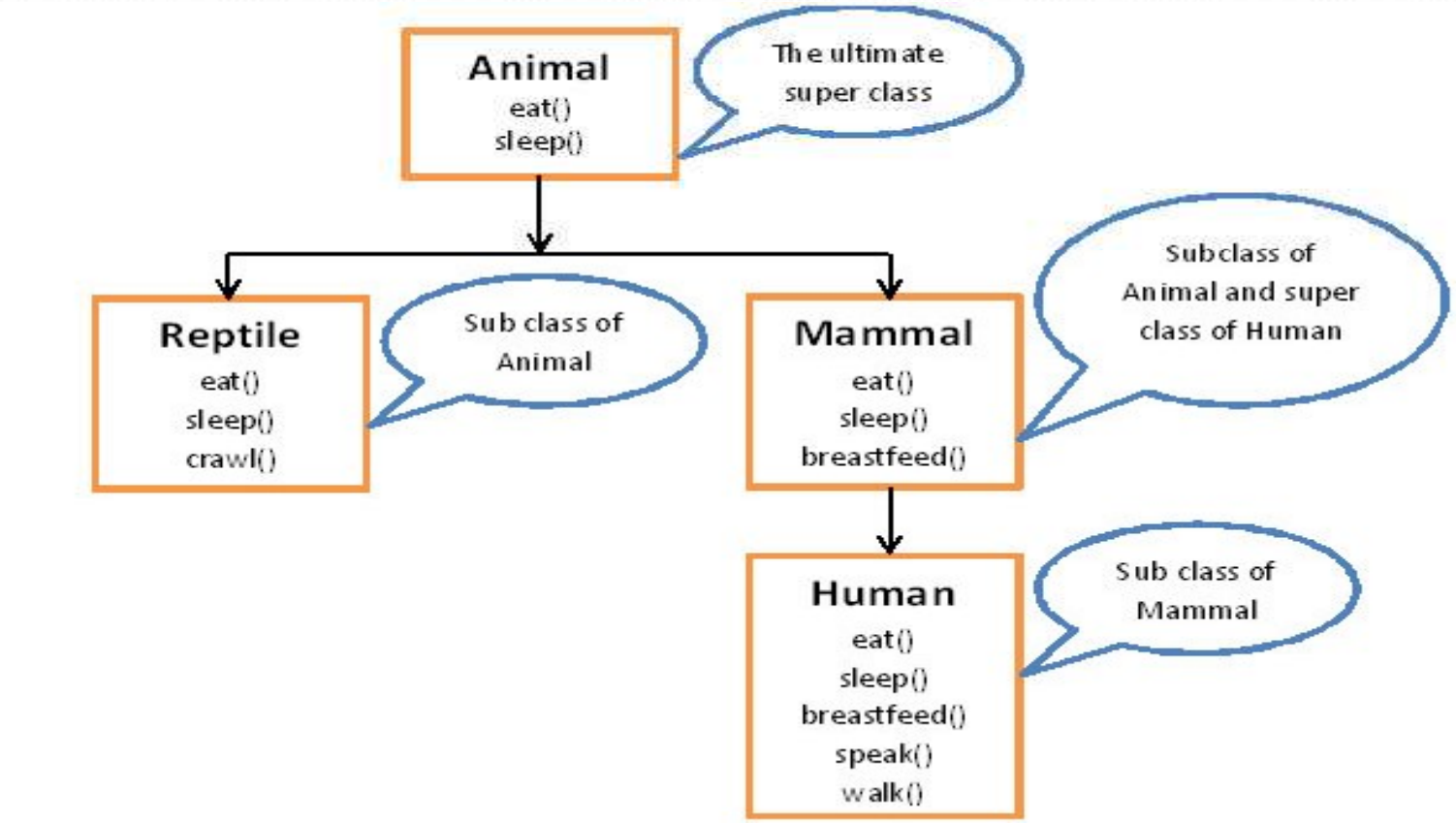
WAP to implement banking operations

```
class Bank:
    def __init__(cust,a,b,c):
        cust.accno=a
        cust.name=b
        cust.bal=c

    def Deposit(self):
        amt=int(input("enter the amount"))
        self.bal=self.bal+amt

    def Wdraw(self):
        amt=int(input("enter the amount"))
        self.bal=self.bal-amt

    def BalEnq(self):
        print("Hello",self.name, "your present bal ",self.bal)

B=Bank(123,"Amar",2000)
B.Deposit()
B.BalEnq()

B.Wdraw()
B.BalEnq()
```

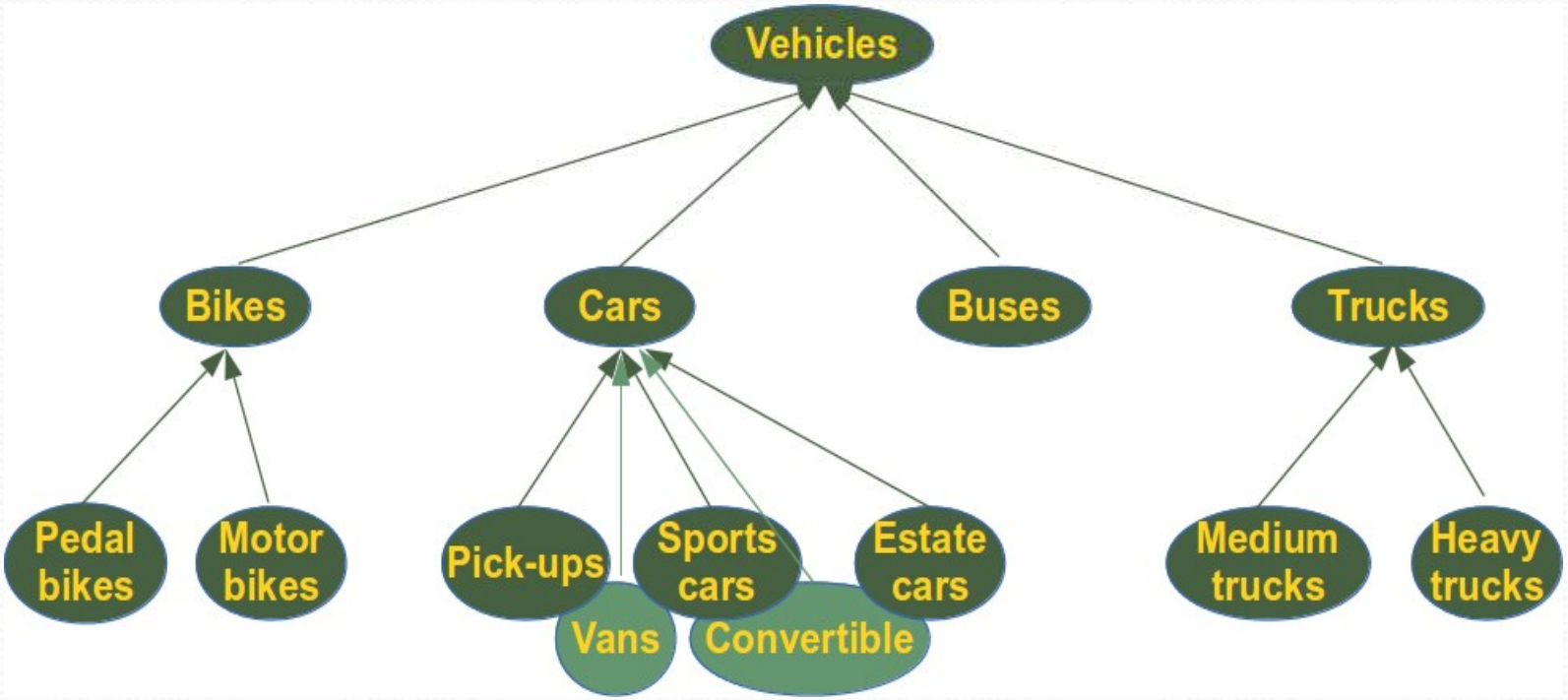
Inheritance

- Inheritance is the process by which one object can acquire the properties of another object.
- Mechanism of deriving a new class from an existing one is called inheritance or derivation.
- The old class is referred to as the base class and the new one is called the derived class.
- It supports the concept of classification

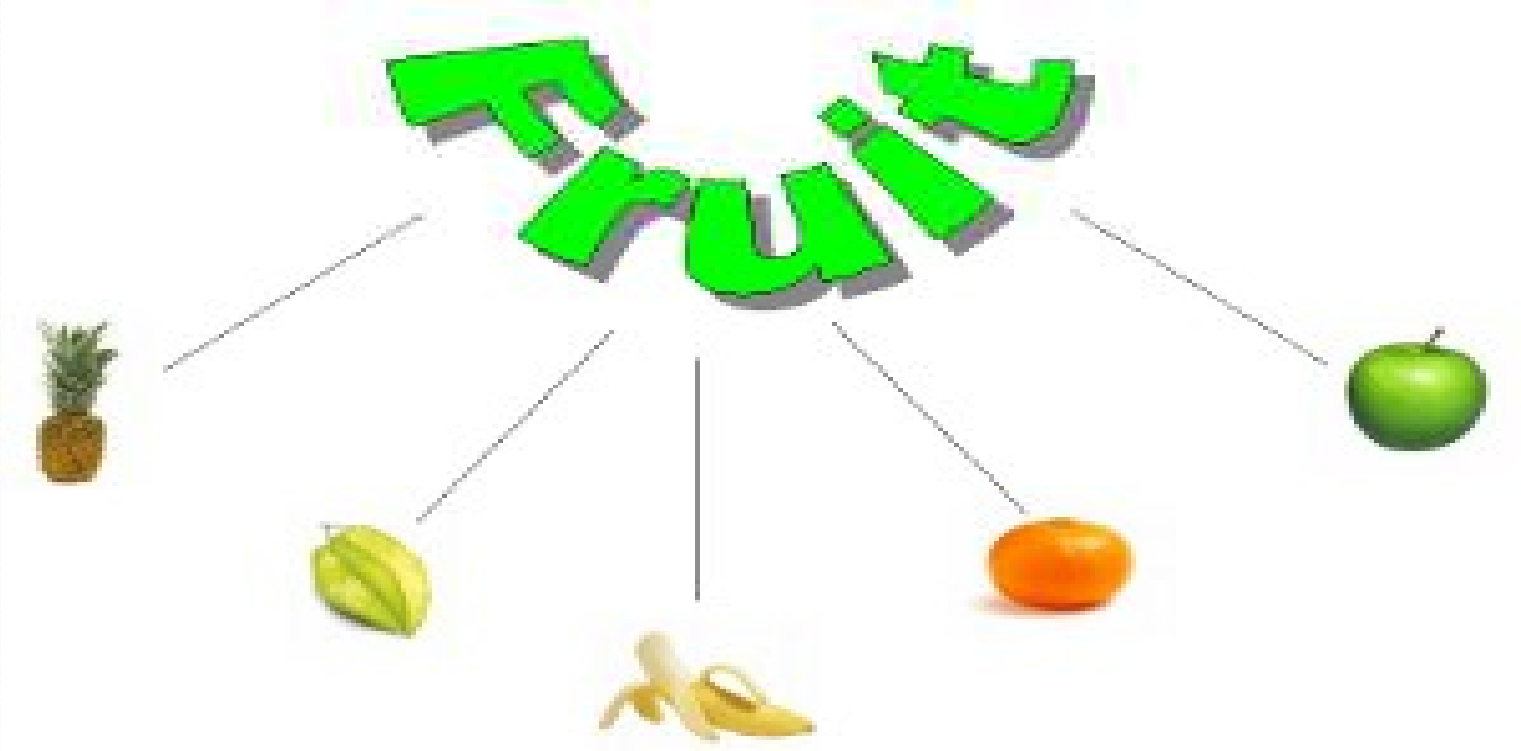
Inheritance in Python



Inheritance in Python



Inheritance in Python



Inheritance

```
class father:
    def __init__(self,a,b,c):
        self.name=a
        self.qual=b
        self.sal=c
    def display(self):
        print(self.name,self.sal)
    def findAnualIncome(self):
        print("income=",12*self.sal)
```

```
class son(father):
    def __init__(self,a,b,c,d):
        father.__init__(self,a,b,c)
        self.sport=d

    def display(self):
        father.display(self)
        print("plays",self.sport)
```

```
f=father("amar","B.E",20000)
f.display()
f.findAnualIncome()
s=son("kumar","B.E",30000,"cricket")
s.display()
s.findAnualIncome()
```

Types of Inheritance:

1) Single Inheritance.

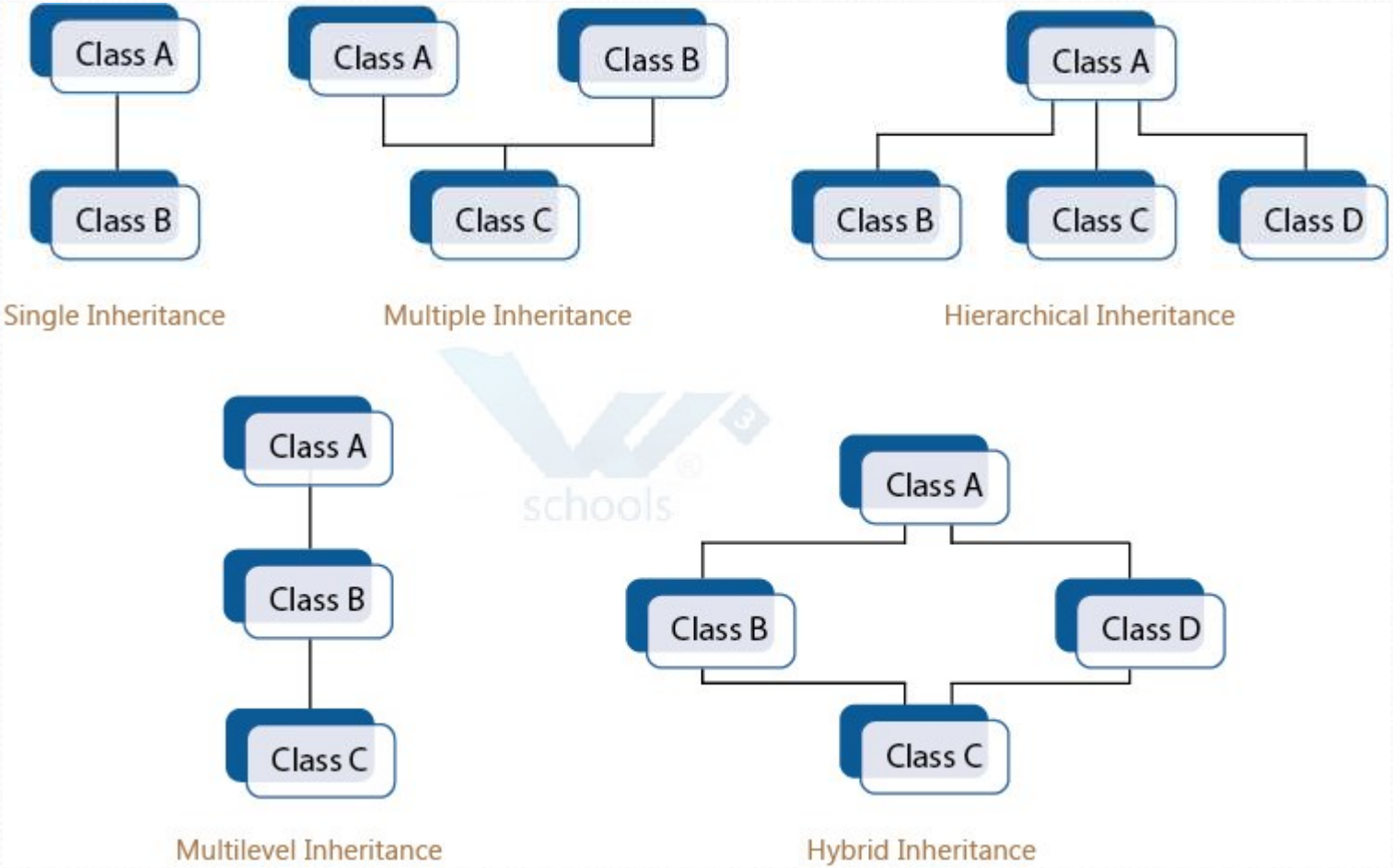
2) Multiple Inheritance.

3) Multilevel Inheritance.

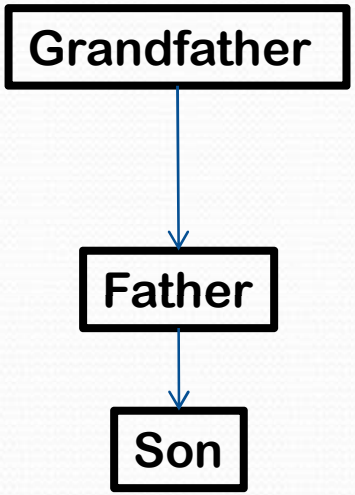
4) Hierarchical Inheritance.

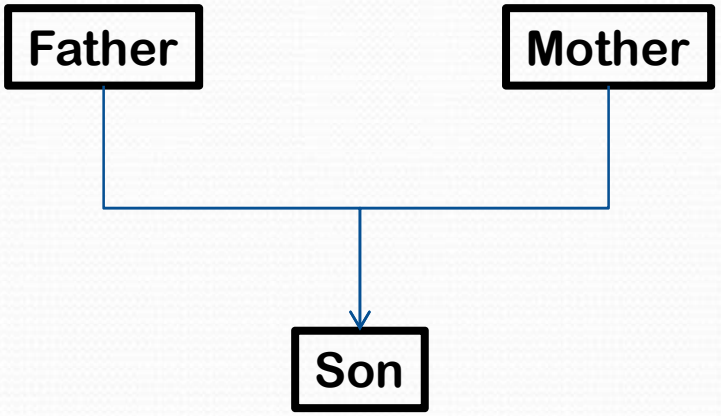
5) Hybrid Inheritance.

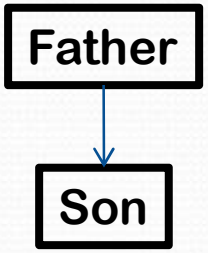
Types of inheritance

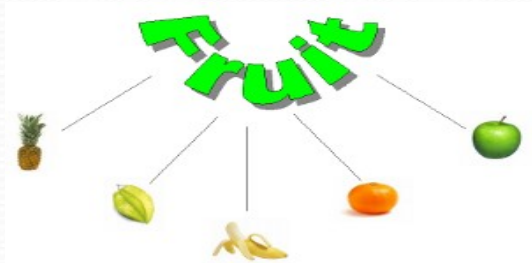


Types of inheritance









Method overriding

```
class A:
    def display(self):
        print("this is class A")
class B(A):
    def display(self):
        print("this is class B")

a=A()
a.display()
b=B()
b.display()
```

Python Class: Polymorphism

```
class Animal:
    def talk(self):
        pass
class Dog(Animal):
    def talk(self):
        print('bow bow!')
class Cat(Animal):
    def talk(self):
        print('MEOW!')
c= Cat()
c.talk()
d=Dog()
d.talk()
```

Polymorphism

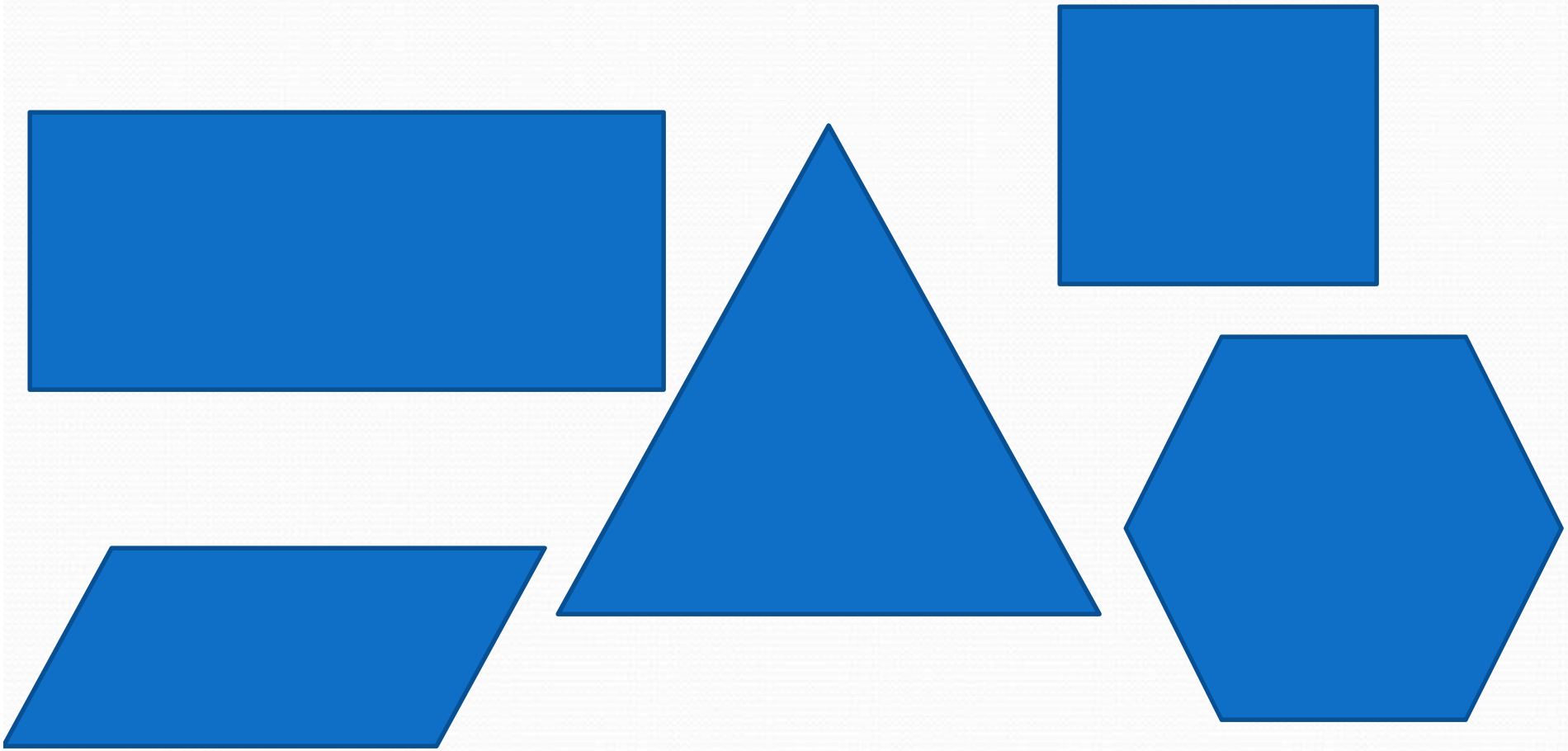
```
class Animal:
    def __init__(self,name):
        self.name=name
    def talk(self):
        pass
class Dog(Animal):
    def talk(self):
        return(self.name,'bow bow!')
class Cat(Animal):
    def talk(self):
        return(self.name,'MEOW!')

animals = [Cat('kitty'),Cat('pinky'),Dog('johny')]

for animal in animals:
   print (animal.name , animal.talk())
```

the "add" operation is defined in many mathematical entities, but in particular cases you "add" according to specific rules:

1+1 = 2, but (1+2i)+(2-9i)=(3-7i

“1”+”1” =11

Polygons (real world example)



Polygons

Area=l*b;

Area=s^2

Area=(b*h)/2

(3*sqrt(3)*s^2)/2

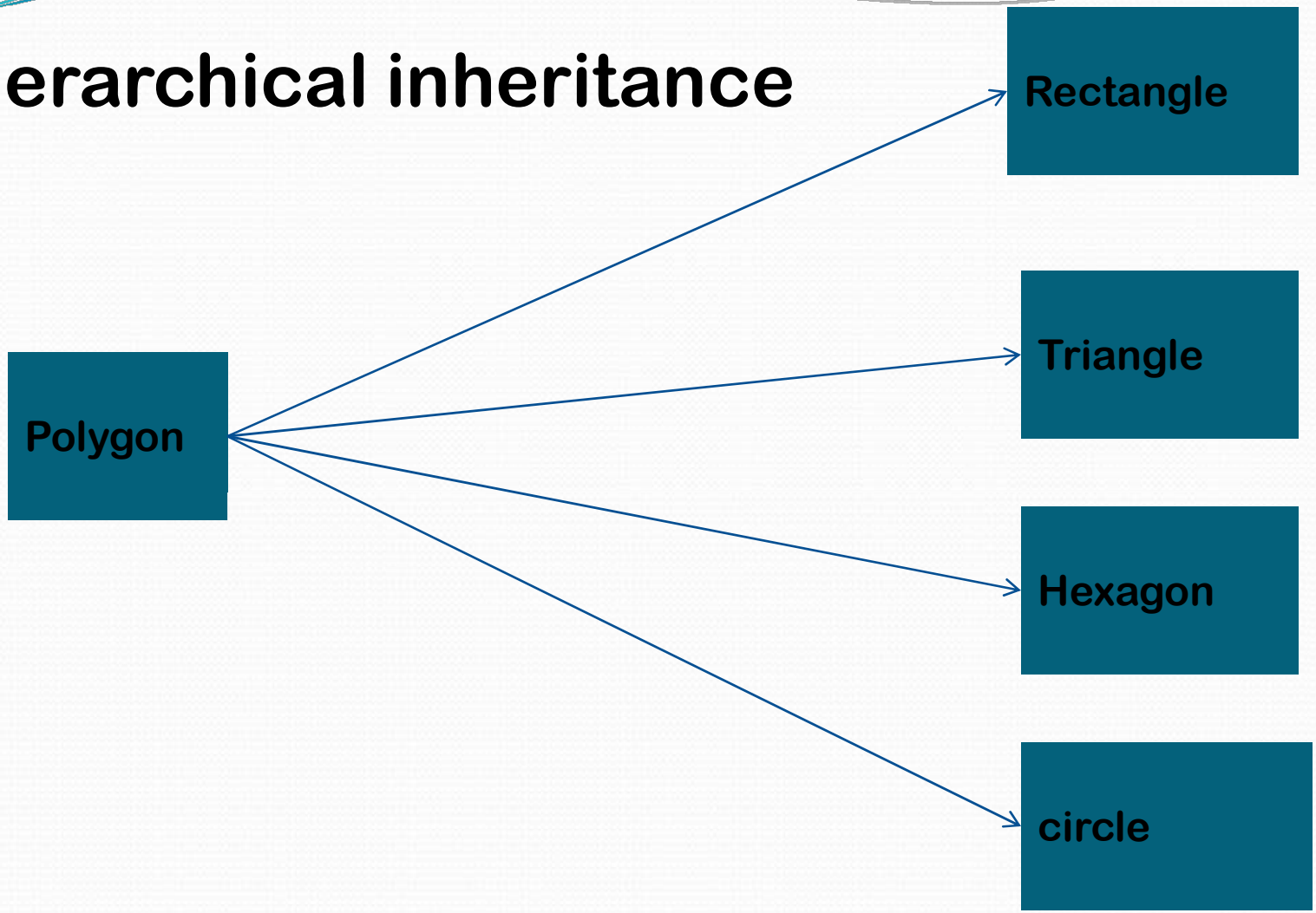
Area=l*b

Polymorphism

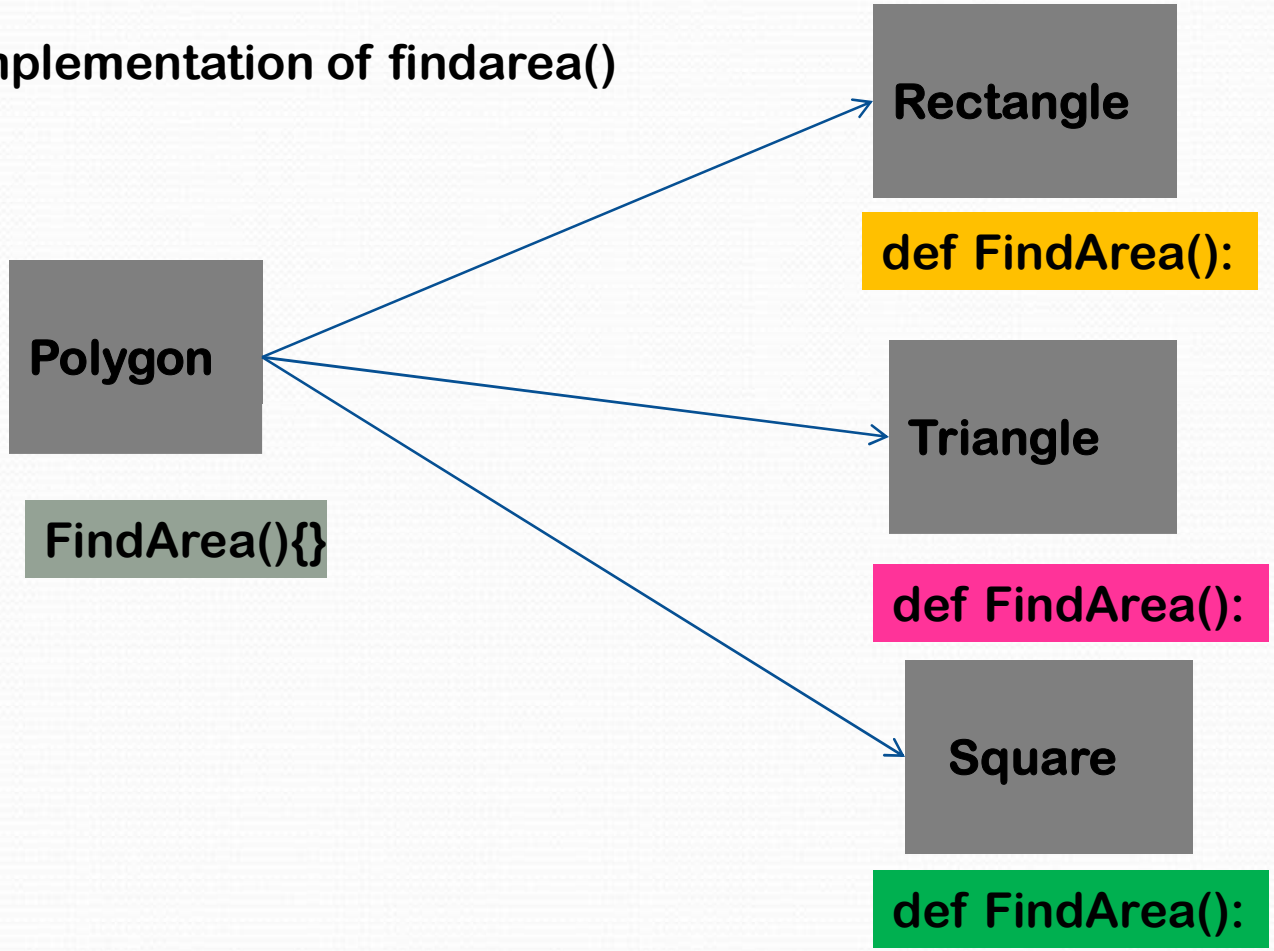
```
class Polygon:
    def findarea(self):
        pass
class Circle(Polygon):
    def __init__(self,r):
        self.radius=r
    def findarea(self):
        print(3.14*self.radius*self.radius)
class Triangle(Polygon):
    def __init__(self,b,h):
        self.base=b
        self.height=h
    def findarea(self):
        print(0.5*self.base*self.height)
```

```
c=Circle(2)
c.findarea()
t=Triangle(2,3)
t.findarea()
```

Hierarchical inheritance

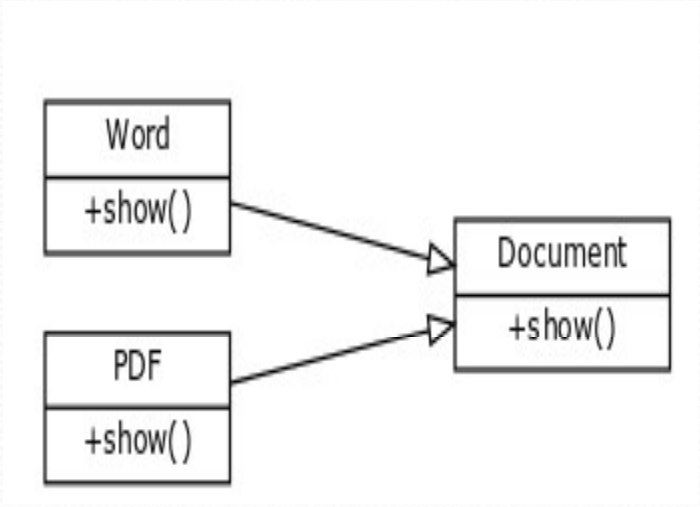


Deferent implementation of findarea()



Polymorphism with abstract class

If you create an editor you may not know in advance what type of documents a user will open (pdf format or word format?).




```
for document in documents:
    print document.name + ': ' + document.show()
```

Polymorphism

```
class Fruit:
    def __init__(self,name):
        self.name=name
    def Eat(self):
        pass
class Apple(Fruit):
    def Eat(self):
        return(self.name,'Cut and eat!')
class Banana(Fruit):
    def Eat(self):
        return(self.name,'Peel and eat!')
class Orange(Fruit):
    def Eat(self):
        return(self.name,'Peel and eat!')

Fruits = [Apple("apple"),Banana("banana"),Orange("orange")]

for Fruit in Fruits:
   print (Fruit.name , Fruit.Eat())
```

Benefits of classes and inheritance

- the class is sharable, so codes can be reused.
- The productivity of programmers increases
- Data is safe and secure with data abstraction.
- Extensibility