
GoCode
We learn by doing, by falling down,
and by picking ourselves back up
HTTP://GOCODENOW.COM

Agenda

1) **Define OOP**

2) **What are key features?**

3) **Examples in Python**

Object Oriented Programming (OOP)

1) OOP was designed to mimic real-world objects

2) Programming paradigm based on coupling code with data in one “object”.

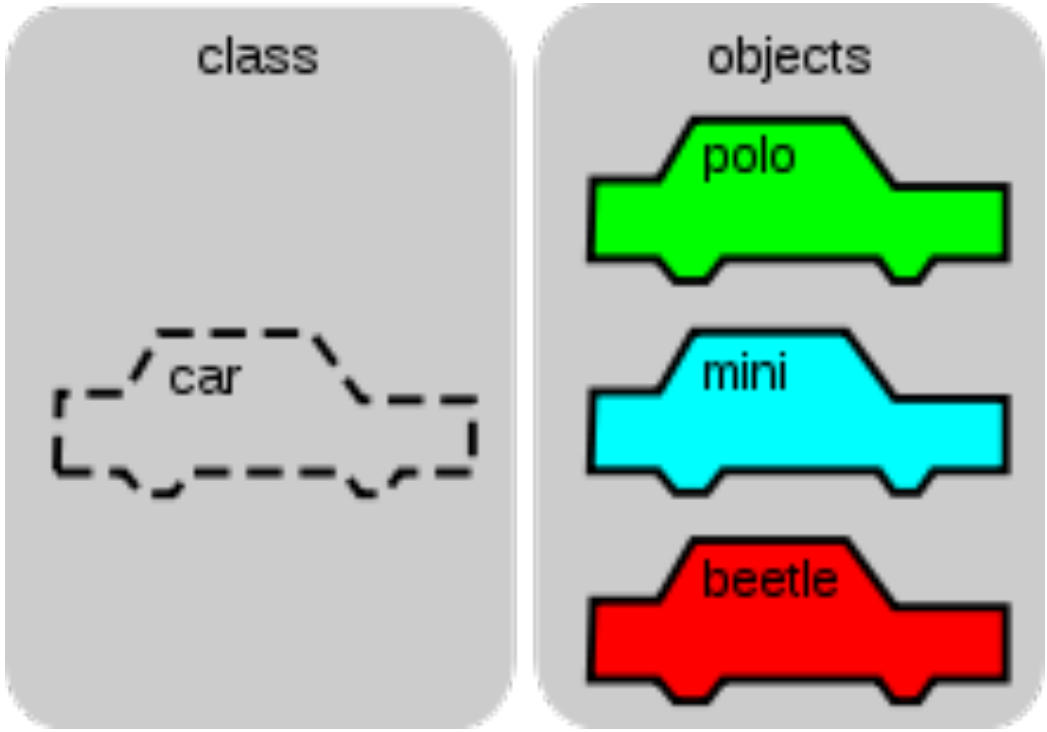
3) In OOP objects have a "class".

4) A "class" define the attributes and the abilities of an object.

5) A class is like a blueprint

Creating an instance

A class is a blueprint for how to build an instance of a class.



Classes in Python

```
class BankAccount(object):

    def __init__(self, name, balance=0.0):
        self.name = name
        self.balance = balance

    def withdraw(self, amount):
        self.balance -= amount
        return self.balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance
```

Using a class

Creating an instance of a class

```
new_customer = BankAccount(“Bob”)

new_customer.deposit(100.00)
```

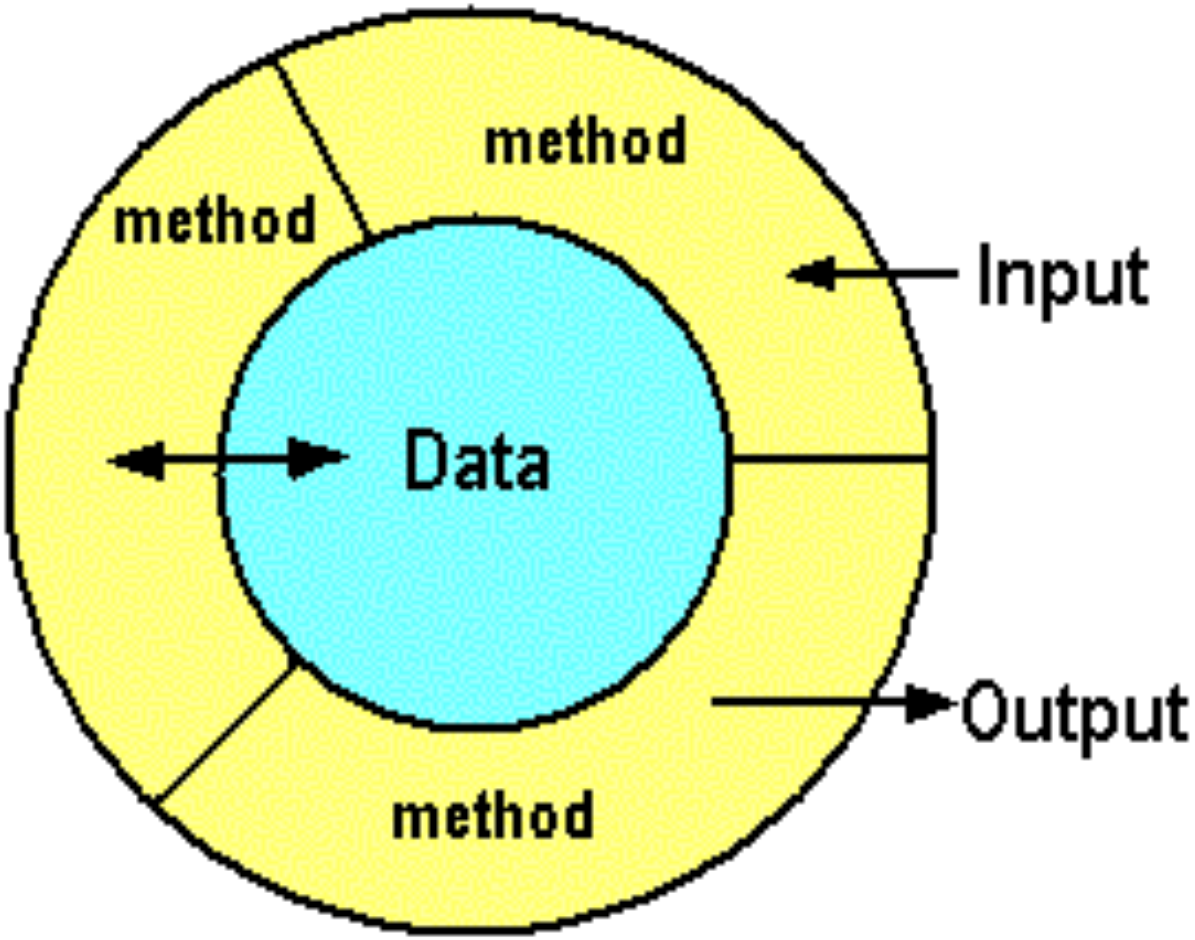
Key ideas in OOP

1) **Encapsulation**

2) **Inheritance**

3) **Polymorphism**

Encapsulation



Encapsulation

1) **To retrieve data, use a "getter"**

2) **To change data, use a "setter"**

Encapsulation

```
class Book:
    title = ''
    pages = 0

    def __init__(self, title='', pages=0):
        self.title = title
        self.pages = pages

    def __str__(self):
        return self.title
```

```
class C:
    def accessible(self):
        print 'you can see me'
    def __inaccessible(self):
        print 'you can not see me'
```

Define public function

Define private function

```
>>> C().accessible()
you can see me
>>> C().inaccessible()
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    C().inaccessible()
AttributeError: C instance has no attribute 'inaccessible'
>>> C()._C__inaccessible()
you can not see me
```

Access public function

Can't access private function

Access private function via changed name

```
class Person:
    def __init__(self):

        self.A = 'Yang Li'
        self.__B = 'Yingying Gu'

    def PrintName(self):
        print self.A
        print self.__B

P = Person()
```

Public variable

Private variable

Invoke private variable in class

```
>>> P.A
'Yang Li'
>>> P.__B
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    P.__B
AttributeError: Person instance has no attribute '__B'
>>> P.PrintName()
Yang Li
Yingying Gu
```

Access public variable out of class, succeed

Access private variable our of class, fail

Access public function but this function access Private variable __B successfully since they are in the same class.

Operator Overloading

```
>>> execfile('book-example.py')
>>> b = Book("Harry Potter")
>>> b
<__main__.Book instance at 0x105b08fc8>
>>> str(b)
'Harry Potter'
>>>
```

Operator Overloading

```
>>> b = Book("Harry Potter",350)
>>> c = Book("POA",450)
>>> b + c
800
```

Key Points For Encapsulation

1) **By encapsulating data, we can control access (using getters and setters)**

2) **OOP with encapsulation we can create new datatypes**

3) **Operator Overloading allows us to standard operators in new ways**

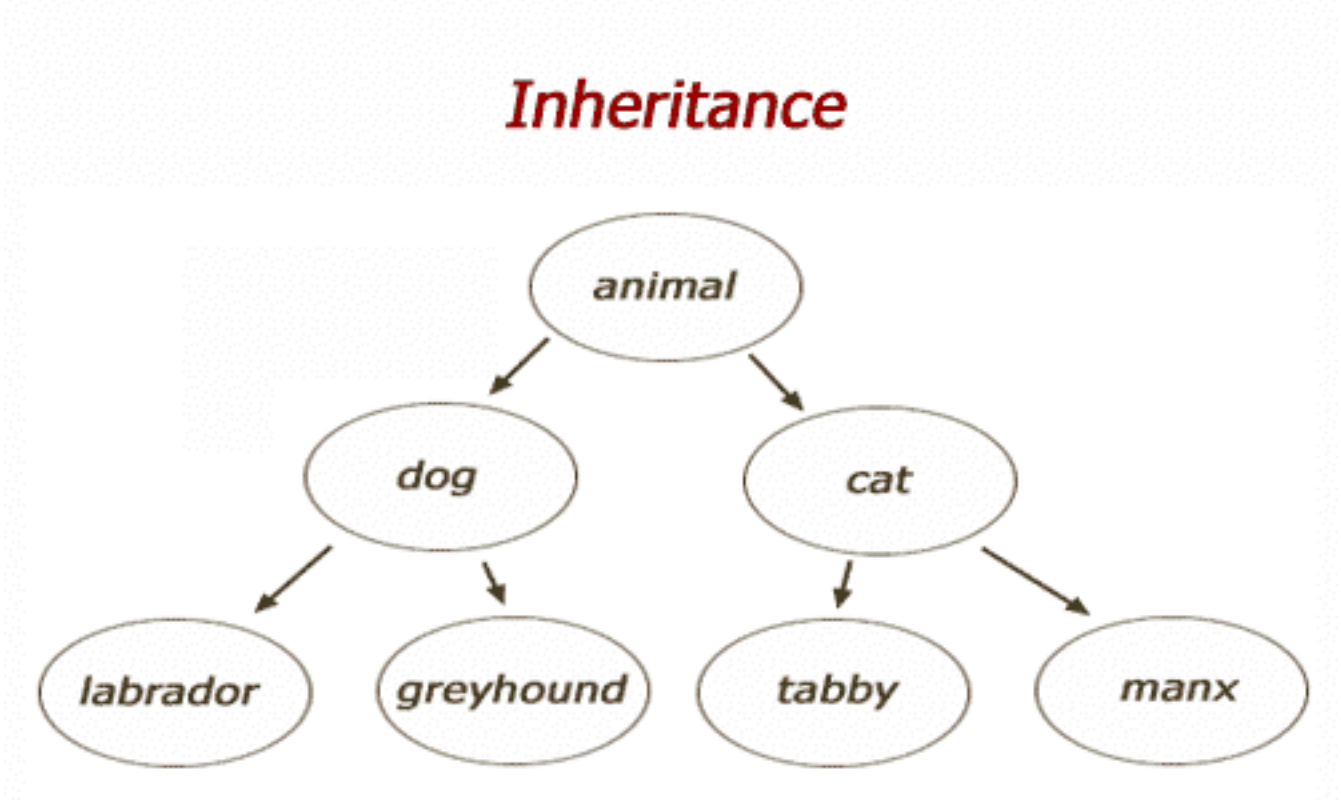
Inheritence

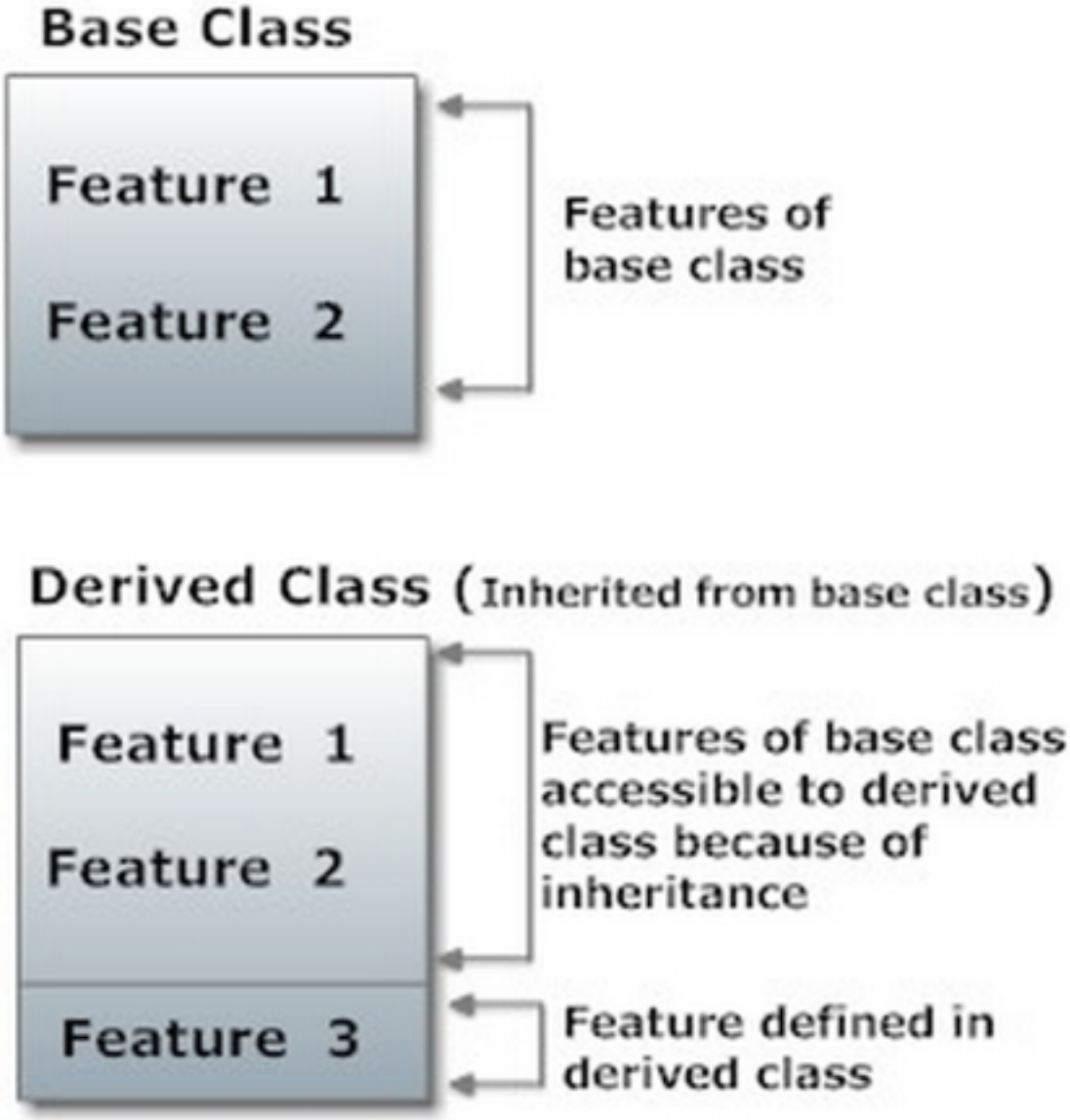
1) A key feature of OOP is inheritance.

2) This allows a class to inherit attributes and methods from a parent

3) The idea behind this is to facilitate code reuse.

Object Oriented Programming (OOP)




Base Class
Feature 1
Feature 2
Features of base class
Derived Class (Inherited from base class)
Feature 1
Feature 2
Features of base class accessible to derived class because of inheritance
Feature 3
Feature defined in derived class

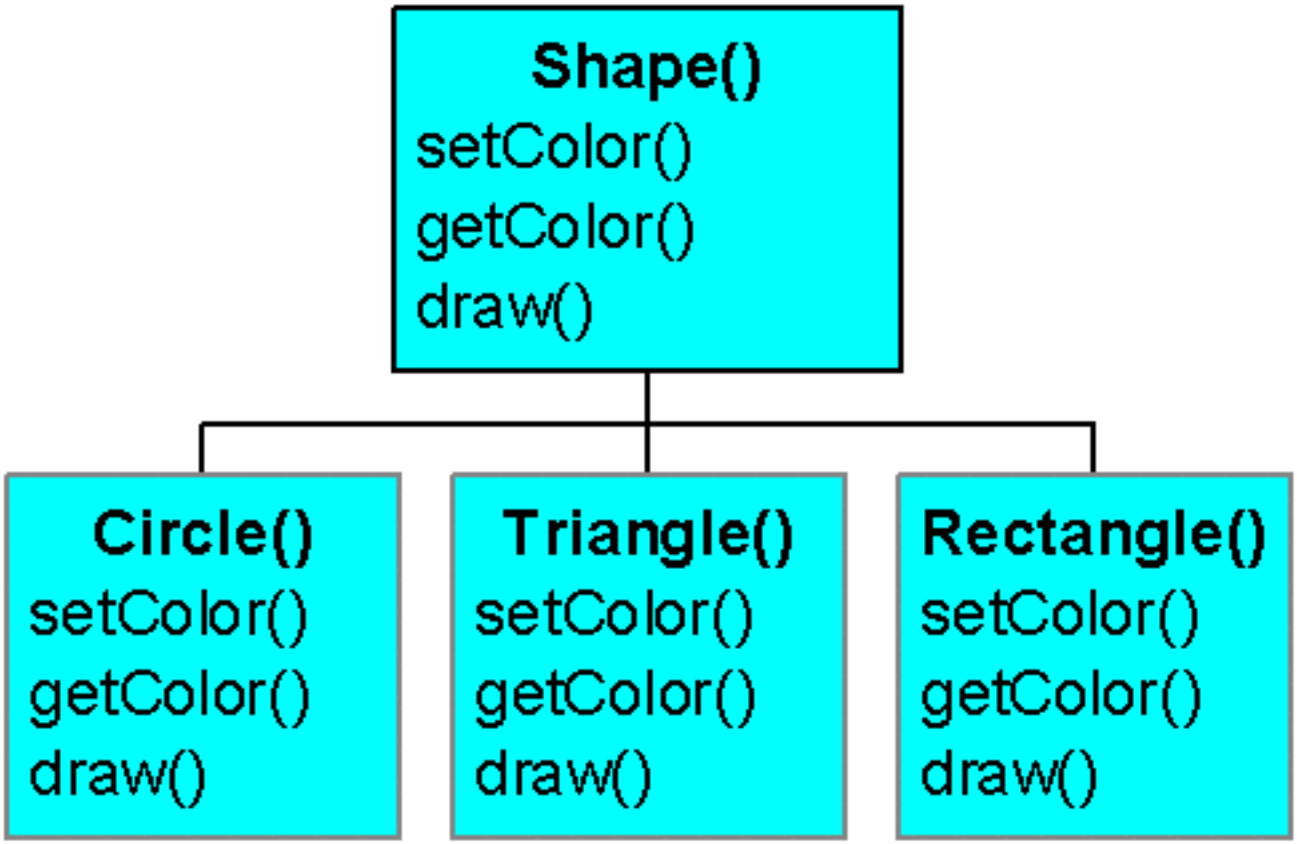
Classes in Python

```
class Animal:
    def __init__(self):
        print "Create an Animal"

class Dog(Animal):
    def __init__(self):
        Animal.__init__()
        print "Create a Dog"

class Cat(Animal):
    def __init__(self):
        print "Create a Cat"
```

Polymorphism



Classes in Python

A drawing app.

```
shapes = [Circle(),Triangle(),Rectangle()]

for shape in shapes:
    shape.setColor("Red")
    shape.draw()
```

```
class Animal:
    def Name(self):
        pass
    def Sleep(self):
        print 'sleep'
    def MakeNoise(self):
        pass

class Dog(Animal):
    def Name(self):
        print 'I am a dog!'
    def MakeNoise(self):
        print 'Woof!'

class Cat(Animal):
    def Name(self):
        print 'I am a cat!'
    def MakeNoise(self):
        print 'Meow'

class Lion(Animal):
    def Name(self):
        print 'I am a lion!'
    def MakeNoise(self):
        print 'Roar'

class TestAnimals:
    def PrintName(self,animal):
        animal.Name()
    def GotoSleep(self,animal):
        animal.Sleep()
    def MakeNoise(self,animal):
        animal.MakeNoise()
```

```
TestAnimals = TestAnimals()
dog = Dog()
cat = Cat()
lion = Lion()

TestAnimals.PrintName(dog)
TestAnimals.GotoSleep(dog)
TestAnimals.MakeNoise(dog)
TestAnimals.PrintName(cat)
TestAnimals.GotoSleep(cat)
TestAnimals.MakeNoise(cat)
TestAnimals.PrintName(lion)
TestAnimals.GotoSleep(lion)
TestAnimals.MakeNoise(lion)
```

```
>>>
I am a dog!
sleep
Woof!
I am a cat!
sleep
Meow
I am a lion!
sleep
Roar
```

Key points

Goal of OOP:

- Encapsulate data, hiding and controlling access to it.

- Inherit attributes and abilities to pass down functionality (multiple!)

- Polymorphism, separate concerns of callee and caller

Key points