Object-oriented programming in Python

Python is a powerful object-oriented programming language. Therefore, creating and using objects is very easy. This article will introduce some basic concepts in object-oriented programming, as well as how to create and use them.

Python is a powerful object-oriented programming language. Therefore, creating and using objects is very easy. This article will introduce some basic concepts in object-oriented programming, as well as how to create and use them. Invites you to read the track.

Introducing OOP in Python

Object-oriented programming (English: Object-oriented programming) (OOP) is a support technique that allows programmers to directly work with objects they define. The effectiveness of this technique increases productivity, simplifies maintenance complexity as well as software expansion. Currently there are many object-oriented programming languages ??such as C ++, Java, PHP, . and also Python.

Python's OOP concept focuses on creating reusable code. This concept is also called DRY (Don't Repeat Yourself).

Principles

In Python, the concept of OOP follows some basic principles: *encapsulation, inheritance and polymorphism* .

Inheritance : allows a class (class) to inherit properties and methods from other defined classes.

Encapsulation : is a rule that requires the internal state of an object to be protected and accessible from outside code (ie the external code cannot directly see and change the state of the object). that object).

Polymorphism: A concept in which two or more classes have similar methods but can be implemented in different ways.

Class (Class) and Object (Object)

Class and Object are two basic concepts in object-oriented programming.

Objects (Object) are existent entities with behaviors.

For example, the object is a car with its name, color, type of material, behavior going on, stopping, parking, exploding .

Class (Class) is a special type of data defined by the user, gathering many attributes specific to all objects created from that class.

Attributes are the values ??of the class. Later when objects are created from the class, the class properties now become the features of that object.

Distinguish between Objects (Object) and Class (Class):

Object (Object): has state and behavior.

Class (Class): can be defined as a template that describes the state and behavior that the object type of the class supports. An object is an instance of a class

Examples of Class and Object:

```
class Car:

 # thu?c tính l?p
 loaixe = "Ô tô"

 # thu?c tính ??i t??ng
 def __init__(self, tenxe, mausac, nguyenlieu):
 self.tenxe = tenxe
 self.mausac = mausac
 self.nguyenlieu = nguyenlieu

# instantiate the Car class
toyota = Car("Toyota", "??", "?i?n")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# access the class attributes
print("Porsche là {}.".format(porsche.__class__.loaixe))
print("Toyota là {}.".format(toyota.__class__.loaixe))
print("Lamborghini c?ng là {}.".format(lamborghini.__class__.loaixe))

# access the instance attributes
print("Xe {} có màu {}. {} là nguyên li?u v?
```

```
n hành.".format( toyota.tenxe, toyota.mausac, toyota.nguyenlieu))
 print("Xe {} có màu {}. {} là nguyên li?u v?
n hành.".format( lamborghini.tenxe, lamborghini.mausac,lamborghini.nguyenlieu))

 print("Xe {} có màu {}. {} là nguyên li?u v?
n hành.".format( porsche.tenxe, porsche.mausac, porsche.nguyenlieu))
```

The returned result will be:

```
 Porsche là Ô tô.
 Toyota là Ô tô.
 Lamborghini c?ng là Ô tô.
 Xe Toyota có màu ??. ?i?n là nguyên li?u v?n hành.
 Xe Lamborghini có màu Vàng. Deisel là nguyên li?u v?n hành.
 Xe Porsche có màu Xanh. Gas là nguyên li?u v?n hành.
```

The above program creates a *Car* class , then determines the properties and characteristics of the object

We access class properties using *__class __. Loaixe.* Class properties are shared for all instances of the class.

Similarly, we access the instance properties using *toyota.tenxe, toyota.mausac* and *toyota.nguyenlieu.*

However, instance attributes are different for each instance of a class.

Method

Methods (Methods) are functions defined inside the body of a class. They are used to identify the behavior of an object.

Example of Class and Method

```
 class Car:

  # thu?c tính ??i t??ng
  def __init__(self, tenxe, mausac, nguyenlieu):
  self.tenxe = tenxe
  self.mausac = mausac
  self.nguyenlieu = nguyenlieu

  # ph??ng th?c
  def dungxe(self, mucdich):
  return "{} ?ang d?ng xe ?? {}".format(self.tenxe,mucdich)

 def chayxe(self):
  return "{} ?ang ch?y trên ???ng".format(self.tenxe)

  def nomay(self):
 return "{} ?ang n? máy".format(self.tenxe)

 # instantiate the Car class
 toyota = Car("Toyota", "??", "?i?n")
 lamborghini = Car("Lamborghini", "Vàng", "Deisel")
```

```
porsche = Car("Porsche", "Xanh", "Gas")

# call our instance methods
print(toyota.dungxe("n?p ?i?n"))
print(lamborghini.chayxe())
print(porsche.nomay())
```

Run the program, the screen will return results:

```
Toyota ?ang d?ng xe ?? n?p ?i?n
Lamborghini ?ang ch?y trên ???ng
Porsche ?ang n? máy
```

In this example, there are three methods: *dungxe (), chayxe ()* and *nomay ()*. They are called instance methods because they are called on an instance object *(toyota, lamborghini, porsche).*

Inheritance

Inheritance allows a class (class) to inherit properties and methods from other defined classes. The existing class is called the parent class, the newly generated class is called the subclass. The subclass inherits all elements of the superclass, which can extend inheritance elements and add new components.

```
# L?p cha
class Car:

 # Constructor
 def __init__(self, hangxe, tenxe, mausac):
 # L?p Car có 3 thu?c tính: tenxe, mausac, hang xe
 self.hangxe = hangxe
 self.tenxe = tenxe
 self.mausac = mausac

 # ph??ng th?c
 def chayxe(self):
 print ("{} ?ang ch?y trên ???ng".format(self.tenxe))

 def dungxe(self, mucdich):
 print ("{} ?ang d?ng xe ?? {}".format(self.tenxe, mucdich))

# L?p Toyota m? r?ng t? l?p Car.
class Toyota(Car):

def __init__(self, hangxe, tenxe, mausac, nguyenlieu):
 # G?i t?i constructor c?a l?p cha (Car)
# ?? gán giá tr? vào thu?c tính c?a l?p cha.
 super().__init__(hangxe, tenxe, mausac)

 self.nguyenlieu = nguyenlieu

 # K? th?a ph??ng th?c c?
 def chayxe(self):
 print ("{} ?ang ch?y trên ???ng".format(self.tenxe))
```

```
 # Ghi ?è (override) ph??ng th?c cùng tên c?a l?p cha.
 def dungxe(self, mucdich):
 print ("{} ?ang d?ng xe ?? {}".format(self.tenxe, mucdich))
 print ("{} ch?y b?ng {}".format(self.tenxe, self.nguyenlieu))

 # B? sung thêm thành ph?n m?i
def nomay(self):
 print ("{} ?ang n? máy".format(self.tenxe))

toyota1 = Toyota("Toyota", "Toyota Hilux", "??", "?i?n")
toyota2 = Toyota("Toyota", "Toyota Yaris", "Vàng", "Deisel")
toyota3 = Toyota("Toyota", "Toyota Vios", "Xanh", "Gas")

toyota1.dungxe("n?p ?i?n")
toyota2.chayxe()
toyota3.nomay()
```

Results returned:

```
Toyota Hilux ?ang d?ng xe ?? n?p ?i?n
Toyota Hilux ch?y b?ng ?i?n
Toyota Yaris ?ang ch?y trên ???ng
Toyota Vios ?ang n? máy
```

This program creates two inheritance classes: the *Car* parent class and the *Toyota* subclass .

Declare the new constructor to assign values ??to the properties of the parent class. The *super ()* function precedes *__init __* to call *__init __* content of Car.

Toyota Class inherits the function *chayxe ()* and *dungxe ()* of the Car class and modifies a behavior expressed in *dungxe () method.* Then the subclass adds new element *nomay ()* to extend inheritance.

Encapsulation

Using OOP in Python, we can restrict access to the inner state of the object. This prevents data from being modified directly, called **packaging** . In Python, we denote this private property by using an underscore as a prefix: '_' or '__'.

```
class Computer:

 def __init__(self):
# Thu?c tính private ng?n ch?n s?a ??i tr?c ti?p
 self.__maxprice = 900

def sell(self):
print("Giá bán s?n ph?m: {}".format(self.__maxprice))

 def setMaxPrice(self, price):
 self.__maxprice = price

c = Computer()c.sell()

# thay ??i giá.
```

```
c.__maxprice = 1000
c.sell()

# s? d?ng hàm setter ?? thay ??i giá.
c.setMaxPrice(1000)
c.sell()
```

Screen showing results:

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

In this example, you initialize the Computer class, use *__init __ ()* to store the maximum selling price of the computer. But after use, you need to modify the price, but you can't change it in the normal way because Python has considered *__maxprice* as a private property. So to change the value, we use the setter function setMaxPrice ().

Polymorphism

Polymorphism is a concept where two or more classes have the same methods but can be implemented in different ways.

Suppose, we need to color a shape, there are many options for your shapes like rectangles, squares, circles. However, you can use the same method to color any shape.

```
class Toyota:

 def dungxe(self):
 print("Toyota d?ng xe ?? n?p ?i?n")

 def nomay(self):
 print("Toyota n? máy b?ng h?p s? t? ??ng")

class Porsche:

 def dungxe(self):
 print("Porsche d?ng xe ?? b?m x?ng")

 def nomay(self):
 print("Porsche n? máy b?ng h?p s? c?")

# common interface
def  # common interface
def kiemtra_dungxe(car): car.dungxe()

# instantiate objects
toyota = Toyota()
porsche = Porsche()

# passing the object
kiemtra_dungxe(toyota)
kiemtra_dungxe(porsche)
```

In this example, you have just created two layers of *Toyota* and *Porsche*, both of which have a *dungeon () method*. Of course their functions are different. We use polymorphism to create common functions for two classes, that is *kiemtra_dungxe ()*. Next, you pass the *toyota* and *porsche* object to the newly created function, and we get this result:

```
Toyota d?ng xe ?? n?p ?i?n
Porsche d?ng xe ?? b?m x?ng
```

So Quantrimang has just introduced you to the highlights of OOP. Through the article, some comments like this can be drawn:

1. Programming becomes easier and more efficient.
2. Class can be shared so the code is easily reused.
3. Productivity of the program increased
4. Data is safe and secure with data abstraction.

See more:

1. Steve Jobs defines object-oriented programming to make the world admire
2. Object-oriented programming in PHP

Previous article: Exception handling - Exception Handling in Python

Next lesson: Learn Class and Object in Python