面向对象

封装 定义简单类

一、dir 内置函数

如何查看对象（变量、数据、函数等）的方法?

1、 IDE 中，在 标识符 / 数据 后输入一个 .，然后按下 Tab 键，iPathon 会显示对象能够调用的 方法列表

2、使用内置函数 dir 传入标识符 / 数据，可以查看对象内的所有属性方法，以列表形式显示处理

```
>>> dir(1)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delatt
r__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '_
_floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__g
t__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__in
vert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__ne
g__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdiv
mod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__
', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '_
_rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__',
 '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__'
, 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', '
real', 'to_bytes']
>>> def demo():
        """这是一个测试函数"""
        print("hello world")


>>> demo()
hello world
>>> dir(demo)
['__annotations__', '__call__', '__class__', '__closure__', '__code__', '__defau
lts__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
 '__ge__', '__get__', '__getattribute__', '__globals__', '__gt__', '__hash__', '
__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__', '__module_
_', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex_
_', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
>>> demo.__doc__
'这是一个测试函数'
```

二、定义简单的类

1、定义值包含方法的类：

```
class 类名:
 def 方法1(self,参数列表1)
 pass
 def 方法2(self,参数列表2)
 pass
```

注意： （1）类名命名规则符合 大驼峰命名法

（2）类中方法的定义，类别之前学的方法的定义，参数列表中多了一个 self ，若该方法属于无参方法，则只写一个 self 即可

2、创建对象

对象变量 = 类名()

```
class Cat:

 def eat(self):
 print("小猫爱吃鱼")
 def drink(self):
 print("小猫爱喝水")

tom = Cat()

 tom.eat()
 tom.drink()
```

引用概念的强调：

使用 print 函数输出 对象变量，默认情况下，是能够输出这个变量 引用的对象 是由哪一个类创建的对象，以及在内存中的地址（十六进制）

而之前学的 id 函数，输出的也是内存地址（十进制）

```
print(lazy_cat)
print("%d" % id(lazy_cat))
print("%x" % id(lazy_cat))
```

02_新建两个猫对象

```
小猫爱吃鱼
小猫爱喝水
<__main__.Cat object at 0x0000026531171240>
2633638548032
26531171240
```

三、给对象增加属性

1、不想修改类的代码，可以在类的外部，通过 对象名.属性名 = *** 的赋值语句，给该对象设置属性

```
class Cat:

 def eat(self):
 # 哪一个对象调用的方法，self接收哪一个对象的引用
 print("%s 爱吃鱼" % self.name)

```

```
    def drink(self):
    print("小猫爱喝水")

# 创建猫对象
tom = Cat()

# 在类外边可以使用 .属性名，给对象定义一个属性
tom.name = "Tom"

tom.eat()
tom.drink()

# 在创建一个猫对象
lazy_cat = Cat()

lazy_cat.name = "大懒猫"

lazy_cat.eat()
lazy_cat.drink()

print(tom)
print(lazy_cat)

lazy_cat2 = lazy_cat
print(lazy_cat2)
```

注：不建议使用

2、self

哪一个对象调用的方法，self接收哪一个对象的引用(表示 当前调用方法的对象自己)

四、初始化方法（类的内部定义属性）

1、当使用 类名 () 创建对象时，对自动执行以下操作：

（1）为对象在内存中 分配空间 —— 创建对象

（2）为对象的属性 设置初始值 —— 初始化方法(__init__)

1) __init__ 方法是对象的内置方法

2) __init__ 方法是专门用来定义一个类 具有哪些属性的方法

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

    print("这是一个初始化方法")

```

```
6
7 tom = Cat()
```

这是一个初始化方法

进程已结束,退出代码0

用户未调用 __init__ 方法，但是运行结果显示了结果，说明代码自动调用了该方法

2、在初始化方法内部定义属性

```
1 class Cat:
2  def __init__(self):
3
4  print("这是一个初始化方法")
5
6  # self.属性名 = 属性的初始值
7  self.name = "Tom"
8
9  def eat(self):
10  print("%s 爱吃鱼" % self.name)
11
12 tom = Cat()
13 tom.eat()
14 print(tom.name)
```

3、利用参数设置属性初始值

```
1 class Cat:
2  def __init__(self,new_name):
3
4  print("这是一个初始化方法")
5
6  # self.属性名 = 属性的初始值
7  # self.name = "Tom"
8  self.name = new_name
9
10  def eat(self):
11  print("%s 爱吃鱼" % self.name)
12
13 tom = Cat("Tom")
14 tom.eat()
15 print(tom.name)
16
17 lazy_cat = Cat("大懒猫")
```

```
18 lazy_cat.eat()
```

```
这是一个初始化方法
Tom 爱吃鱼
Tom
这是一个初始化方法
大懒猫 爱吃鱼
```

对 __init__ 方法进行改造：

（1）把希望设置的属性值，定义成 __init__ 方法的参数

（2）在方法内部使用 self.属性 = 形参 接收外部传递的参数

（3）在创建对象时，使用 类名(属性1，属性2……) 调用

4、__del__ 方法

__del__ 当一个对象在内存中被销毁前，会自动调用该方法

```
1  class Cat:
2
3   def __init__(self,new_name):
4
5   self.name = new_name
6
7   print("%s 来了" % self.name)
8
9   def __del__(self):
10
11   print("%s 走啦" % self.name)
12
13  # Tom 是一个全局变量
14  tom = Cat("Tom")
15  print(tom.name)
16
17  # del 关键字可以删除一个对象
18  # del tom
19
20  print("+" * 50)
```

对比，del tom 该句出现与否时 “Tom 走啦” 出现在分割线的位置

无 del

```
Tom 来了
Tom
++++++++++++++++++++++++++++++++++++++++++++++++++
Tom 走啦

进程已结束,退出代码0
```

有

```
Tom 来了
Tom
Tom 走啦
++++++++++++++++++++++++++++++++++++++++++++++++++
```

5、__str__ 方法

若希望在使用 print 方法输出时，输出用户希望的字符串，则可以使用该方法 返回希望输出的字符串

```python
class Cat:

 def __init__(self,new_name):

 self.name = new_name

 print("%s 来了" % self.name)

 def __del__(self):

  print("%s 走啦" % self.name)

  def __str__(self):

  # 必须返回一个字符串
  return "我是小猫[%s]" % self.name

# Tom 是一个全局变量
tom = Cat("Tom")
print(tom)
```

```
Tom 来了
我是小猫[Tom]
Tom 走啦
```

四、私有属性和私有方法

私有属性和私有方法的定义：在属性名或者方法名前**增加 两个下划线**，定义的属性和方法就是私有的

```
class Women:

 def __init__(self,name):

 self.name = name
 self.__age = 18

 def secret(self):

  #在对象的方法内部，是可以访问对象的私有属性的
  print("%s 的年龄是 %d" % (self.name,self.__age))

xiaofang = Women("小芳")

# 私有属性和私有方法在外界不能被直接访问
# print(xiaofang.__age)

xiaofang.secret()
```

python不存在绝对私有属性或者私有方法

之前只是 伪私有

解释器对于私有属性 / 方法的处理方式：

```
print(xiaofang._Women__age)
```