# 面向对象

## 课程目标

* 能够定义Person类，具有name和age两个变量/属性
* 能够创建Person类的对象，并对其name和age变量/属性赋值
* 能够定义实例方法，用于打印实例属性name和age的值
* 能够说出 \_\_init\_\_ 方法的作用
* 能够说出 \_\_str\_\_ 方法的作用
* 能够完成手机基础版案例
* 能够完成类属性的定义
* 能够完成类方法的定义与调用
* 能够完成静态方法的定义与调用

## 课程内容

* 面向对象
* 对象的成员
* 类的成员
* 封装

### 面向对象

#### 1.1面向对象概念

面向对象是一种编程思想，是对现实世界中的事物进行抽象的方式。应用到代码编程设计中，是一种建立现实世界事物模型的方式。

#### 1.2 面向对象和面向过程区别

面向过程关注的是完成工作的步骤，面向对象关注的是谁能完成工作。面向对象是在完成工作的时候关注哪些个体能够完成对应的工作，找到对应的个体即可完成对应任务。

### 类

#### 2.1类的概念

类是一系列事物的统称，同类事物必定具有相同的特征。日常事物的特征根据特性分为状态和行为，对应到类中是变量和方法。

特征分类：

状态 🡪 变量（成员变量）

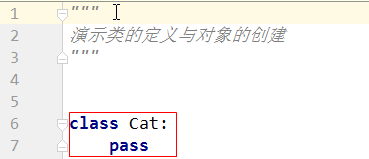
行为 🡪 方法（成员方法）

统称：类的成员

#### 2.2类定义语法格式

**class** 类名：

**pass**



说明：pass是空语句，不具有任何意义，仅为了保持程序结构完整性

#### 2.3类名的命名规则

类名命名规范：满足大驼峰命名规则

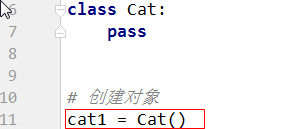
### 对象

#### 3.1对象的概念

对象是类的具体表现形式，是实际存在的个体。（类是一系列事物的统称）

#### 3.2创建对象语法格式

对象名 = 类名()



#### 3.3 注意事项

对象的创建不限制数量，一个类可以创建任意个数的对象

### 成员变量

#### 4.1成员变量的概念

成员变量用于描述对象的固有状态或属性。

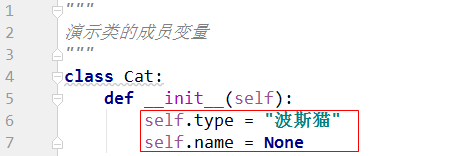
#### 4.2定义成员变量语法格式（公有属性/公有变量）

class 类名：

def \_\_init\_\_(self):

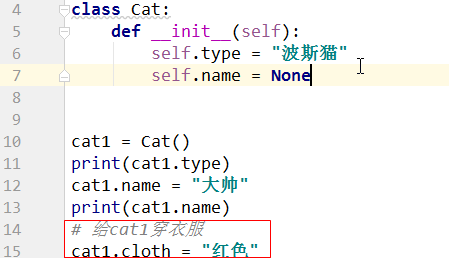
self.变量名1 = 值1

self.变量名2 = None



#### 4.3成员变量定义语法格式（独有属性/独有变量）

对象名.变量名 = 值



#### 4.4公有变量与独有变量的区别

* 公有变量在\_\_init\_\_方法中声明，每个对象都具有该变量
* 独有变量在创建对象后声明，只有当前对象具有此变量
* 定义对象的独有变量时，如果独有变量名与公有变量名相同，视为修改公有变量的值；如果独有变量名与公有变量名不相同，视为定义新的独有属性
* None含义是为空，表示没有具体的数据

#### 4.5变量的调用格式

取值：对象名.变量名

赋值：对象名.变量名 = 值

### 成员方法

#### 5.1成员方法概念

成员方法用于描述对象的固有行为

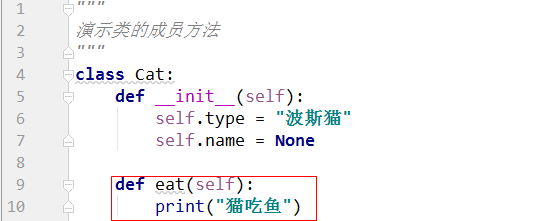
#### 5.2定义成员方法语法格式

格式一（无参方法）：

class 类名：

def 方法名(self):

方法体

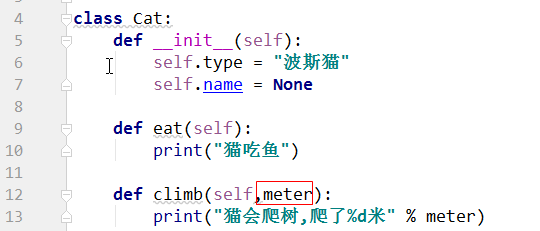


格式二（有参方法）

class 类名：

def 方法名(self,形参1，形参2，…):

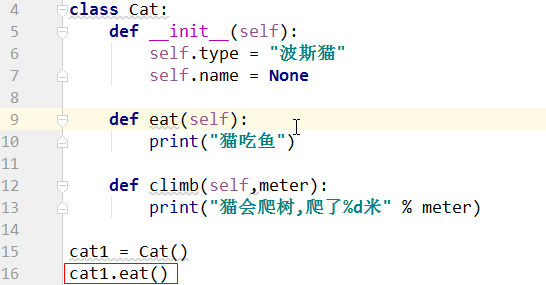
方法体



#### 5.3调用成员方法语法格式

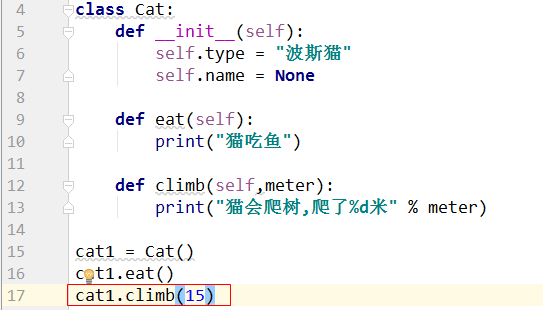
格式一（调用无参方法）：

对象名.方法名（）



格式二（调用有参方法）：

对象名.方法名（实参1，实参2，…）



### \_\_init\_\_方法

#### 6.1\_\_init\_\_方法的作用

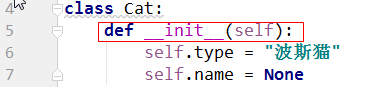
\_\_init\_\_方法，也称为构造方法，创建对象时为对象初始化成员变量

#### 6.2定义\_\_init\_\_方法语法格式

**格式一：**

def \_\_init\_\_(self):

self.变量名1 = 值1

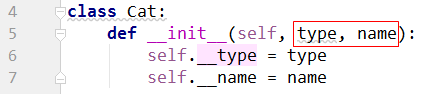


**格式二：**

def \_\_init\_\_(self,形参1，形参2，…):

self.\_\_属性名1 = 形参1

self.\_\_属性名2 = 形参2



#### 6.3\_\_init\_\_方法调用和执行

* \_\_init\_\_方法在创建对象时被运行，无需手工调用即可执行
* \_\_init\_\_方法中如果有形参，在创建对象时需要传递对应的实参

#### 6.4魔术方法

**魔术方法**是在类中声明、定义的方法，执行时无需手工调用，自动执行。

### \_\_str\_\_方法

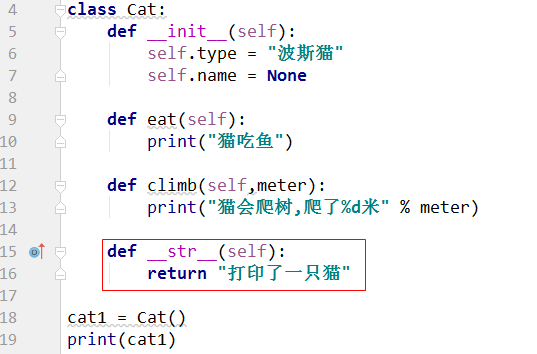
#### 7.1\_\_str\_\_方法的作用

\_\_str\_\_方法是在使用print函数打印输出对象到屏幕时被调用的方法，用于返回对象的规范化字符串表现形式

#### 7.2定义\_\_str\_\_方法语法格式

def \_\_str\_\_(self):

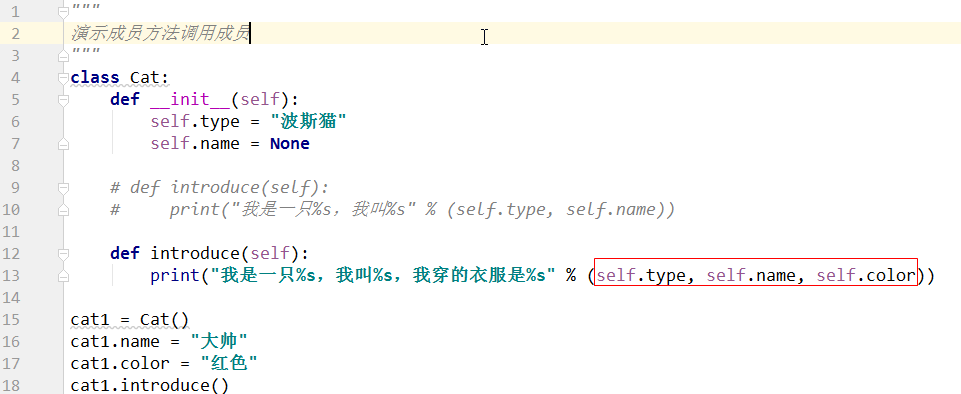
return 打印对象是显示的信息



### 成员方法调用类成员

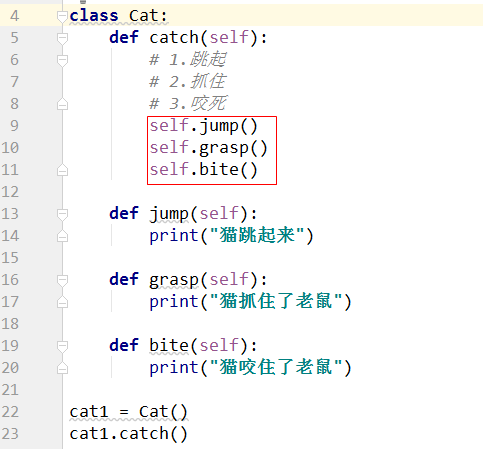
#### 8.1成员方法中调用成员变量语法格式

self.变量名



#### 8.2成员方法中调用成员方法语法格式

self.方法名(实参1，实参2，…)



#### 8.3 self

self仅出现在成员方法中，指代执行该方法的对象。

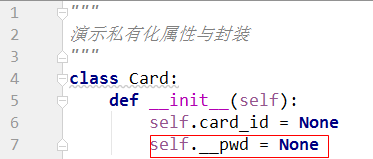
### 9.封装

#### 9.1封装的作用

封装操作可以对受访问保护的成员进行功能开放的控制，达到保护数据不被非法访问的目的

#### 9.2定义私有变量语法格式

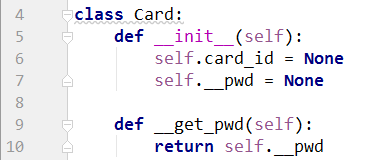
self.\_\_属性名 = 值



#### 9.3定义私有方法语法格式

def \_\_方法名(self，形参1，形参2，…):

方法体



#### 9.4 标准封装语法格式

变量私有化格式：

self.\_\_属性名 = 值

提供公共的访问器格式：

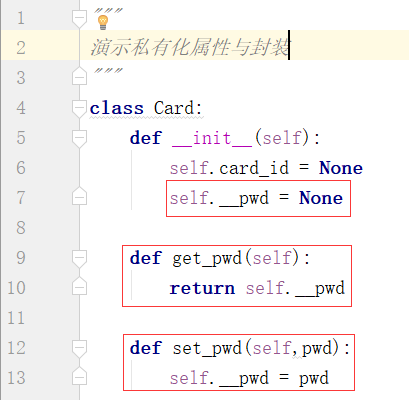
def get\_属性名(self):

return self.\_\_属性名

提供公共的修改器格式：

def set\_属性名(self,属性名):

self.\_\_属性名 = 属性名



### 10.类变量

#### 10.1类变量的概念

类变量是定义在类中，不属于某个具体对象的特征，被所有对象共同使用的特征

#### 10.2定义类变量语法格式

class 类名：

变量名 = 值

#### 10.3调用类变量语法格式

赋值格式：

类名.类变量名 = 值

取值格式：

类名.类变量名（推荐）

对象名.类变量名（不推荐）

#### 10.4注意事项

* + - 类变量推荐使用类名取值访问，也可以使用对象名取值访问
    - 类变量只能通过类名修改；使用对象名调用类变量修改类变量的值时会创建新的对象独有变量；如果此时对象具有同名对象变量，则修改对应的变量值
    - 类变量可以私有化

### 11.类方法

#### 11.1类方法的概念

类方法是定义在类中，不属于某个具体对象的行为，被所有对象共同使用的行为

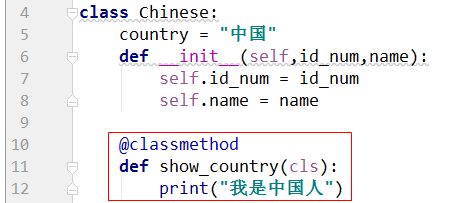
#### 11.2定义类方法基本语法

class 类名：

@classmethod

def 方法名(cls, 形参1，形参2，…):

方法体



#### 11.3调用类方法语法格式

类名.类方法名(实参1，实参2，…) （推荐）

对象名.类方法名(实参1，实参2，…) （不推荐）

#### 11.4注意事项

* + - 类方法中不允许使用实例变量和实例方法
    - 类方法中允许使用类变量和类方法，通过第一个参数cls进行
    - 实例方法中允许使用类变量和类方法

### 12.静态方法

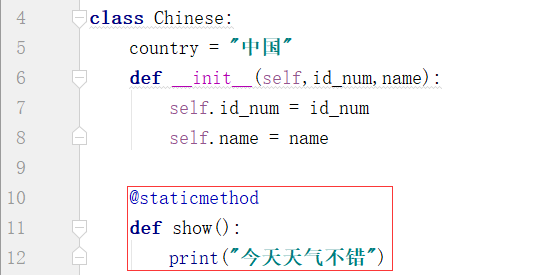
#### 12.1定义静态方法语法格式

class 类名：

@staticmethod

def方法名**(**形参1,形参2，…**):**

方法体



#### 12.2调用静态方法语法格式

类名.静态方法名(实参1，实参2，…) （推荐）

对象名.静态方法名(实参1，实参2，…) （不推荐）

#### 12.3各种方法定义规则

* + - 只访问成员变量的方法，定义成实例方法
    - 只访问类变量的方法，定义类方法
    - 既访问成员变量，也访问类变量的方法，定义成实例方法
    - 既不访问成员变量，也不访问类变量，定义成静态方法

### 案例：手机

"""  
演示手机案例  
要求：  
 手机电量默认是100  
 打游戏每次消耗电量10  
 听歌每次消耗电量5  
 打电话每次消耗电量4  
 接电话每次消耗电量3  
 充电可以为手机补充电量  
"""  
# 分析  
# 1. 定义类Phone  
# 2. 定义变量用于描述电量值  
# 3. 定义4个方法用于描述耗电操作  
# 4. 定义1个方法用于描述充电操作  
# 5. 运行程序，执行上述操作，观察结果  
class Phone:  
 def \_\_init\_\_(self):  
 self.power = 100  
  
 def game(self):  
 """打游戏操作，耗电10"""  
 print("正在打游戏，耗电10")  
 self.power = self.power - 10  
  
 def music(self):  
 print("正在听歌，耗电5")  
 self.power -= 5  
  
 def call(self):  
 print("正在打电话，耗电4")  
 self.power -= 4  
  
 def answer(self):  
 print("正在接电话，耗电3")  
 self.power -= 3  
  
 def charge(self,num):  
 print("正在充电，冲电量是%d" % num)  
 self.power += num  
  
 def \_\_str\_\_(self):  
 return "当前手机电量为:%d" % self.power  
  
# 创建一部电话，当前电量是100  
p = Phone()  
# 执行耗电操作  
p.game()  
print(p)  
p.music()  
print(p)  
p.charge(8)  
print(p)

### 案例：手机加强版

"""  
演示手机加强版案例  
要求：  
 手机耗电操作前要先判定电量是否足够完成此项任务  
 手机充电操作不能充电超过最大值  
"""  
# 分析  
# 1. 耗电操作之前必须进行判定（if系列),根据结果不同做对应的事情  
# 2. 充电操作必须进行判定，手机电量不能超过最大值100  
  
class Phone:  
 def \_\_init\_\_(self):  
 self.power = 100  
  
 def game(self):  
 """打游戏操作，耗电10"""  
 # 判定当前电量是否能够完成打游戏的要求  
 if self.power >= 10:  
 # 正常运行，减少电量  
 print("正在打游戏，耗电10")  
 self.power = self.power - 10  
 else:  
 # 不能正常运行，不减少电量  
 print("当前电量不足以完成打游戏的操作，打游戏操作被终止")  
  
 def music(self):  
 if self.power >= 5:  
 # 正常运行，减少电量  
 print("正在听歌，耗电5")  
 self.power -= 5  
 else:  
 # 不能正常运行，不减少电量  
 print("当前电量不足以完成听歌的操作，听歌操作被终止")  
  
 def call(self):  
 if self.power >= 4:  
 # 正常运行，减少电量  
 print("正在打电话，耗电4")  
 self.power -= 4  
 else:  
 # 不能正常运行，不减少电量  
 print("当前电量不足以完成打电话的操作，打电话操作被终止")  
  
 def answer(self):  
 if self.power >= 3:  
 # 正常运行，减少电量  
 print("正在接电话，耗电3")  
 self.power -= 3  
 else:  
 # 不能正常运行，不减少电量  
 print("当前电量不足以完成接电话的操作，接电话操作被终止")  
  
 def charge(self,num):  
 # 判断当前电量，再与充电量做比较，根据情况执行不同的内容  
 if self.power + num > 100:  
 # 过冲  
 print("充电操作提前结束，当前电量为：100")  
 self.power = 100  
 else:  
 # 正常冲  
 print("正在充电，冲电量是%d" % num)  
 self.power += num  
  
 def \_\_str\_\_(self):  
 return "当前手机电量为:%d" % self.power  
  
# 创建一部电话，当前电量是100  
p = Phone()  
# 执行耗电操作  
p.game()  
p.game()  
p.game()  
p.game()  
p.game()  
p.game()  
p.game()  
p.game()  
p.game()  
p.game()  
print(p)  
p.game()  
p.charge(40)  
print(p)  
p.game()  
print(p)  
p.charge(80)  
print(p)