# 面向对象Object Oriented

## 概述

### 面向过程

1. 例如：婚礼筹办

-- 发请柬(选照片，措辞，制作)

-- 宴席(找厨师，准备桌椅餐具，计划菜品，购买食材)

-- 仪式(司仪，确定流程)

2. 分析解决问题的步骤，然后逐步实现。

3. 优点：所有环节、细节自己掌控。

4. 缺点：费心，考虑所有细节，工作量大。

### 面向对象

1. 例如：婚礼筹办

-- 发请柬：找摄影公司。

-- 宴席：找酒店（告诉对方标准，数量）。

-- 仪式：找婚庆公司（对方提供司仪，制定流程，提供设备）。

2. 找出解决问题的人，然后分配职责。

3. 优点：

(1) 思想层面：

-- 更接近人的思维方式。

-- 有利于梳理归纳问题。

(2)技术层面：

-- 高复用：对重复的代码进行封装，提高开发效率。

-- 高扩展：增加新功能，不修改以前的代码。

-- 高维护：代码可读性好，逻辑清晰，结构规整。

4. 缺点：

初学者通常不能将功能拆分开。

## 类和对象

1. 类：一个抽象的概念，即生活中的”类别”

例如：学生、水果。

1. 对象：类的具体实例，即归属于某个类别的”个体”

例如：张三同学、苹果

1. 类是创建对象的”模板”.

-- 数据成员：名词性的状态。例如：姓名

-- 方法成员：动词性的行为。例如：学习

1. 类与类的行为不同，对象与对象的数据不同。

举例子：

1. 狗是一个类，具有姓名，性别，体重，品种数据成员，

具有咬人，叫，拉臭臭方法成员。

对象：拉不拉多，数据：米咻，母，70，大型犬。

金毛，数据：赵金多，母，60，大型犬。

1. 电子产品是一个类，具有类型，档次，品牌等数据成员，

具有打开，关闭等方法成员。

对象：电脑，数据：笔记本，高端，外星人

手机，数据：翻盖，中端，诺基亚。

(3)字符串是一个类，”abc”对象。

### 定义类

1. 代码：

class 类名:

“””文档说明”””

def \_\_init\_\_(self,参数列表):

self.实例变量 = 参数

方法成员

1. 说明：

-- 类名所有单词首字母大写

-- \_\_init\_\_叫做构造函数,创建对象时被调用，也可以省略。

-- self变量绑定的是被创建的对象，名字通常叫做”self”。

### 语法

#### 创建对象(实例化对象)

变量 = 构造函数(参数列表)

备注：构造函数的self参数会自动绑定对象地址，不用传递。

练习：创建汽车类car,具有类型type，速度speed等数据

启动start，停止stop，行驶run等方法。

创建对象：

宝马

比亚迪

作业1：开放性

以“万物皆对象”的思想，审查客观世界中的对象，然后进行抽象化，形成类(数据/行为)。

要求：每个同学自行创 建2个类。

作业2：将2048所有算法中的函数返回值去掉。

扩展作业3：完成向右移动功能。

作业4：菜鸟教程/电子书，学习面向对象。

### 实例成员

#### 实例变量

1. 语法：

-- 定义：对象地址.变量名称

-- 调用：对象地址.变量名称

2. 说明

(1) 首次通过对象赋值为创建变量，再次赋值为修改。

w01 = Wife()

w01.name = “丽丽” # 创建

w01.name = “莉莉” # 修改

(2) 通常在构造函数(\_\_init\_\_)中创建.

w01 = Wife(“丽丽”)

w01.name = “丽丽” # 修改

(3) 每个对象存储一份，通过对象地址访问。

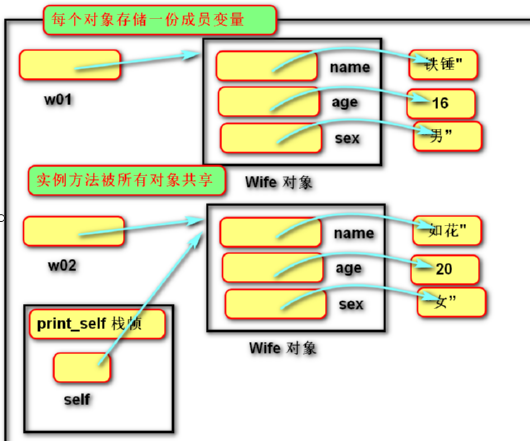
#### 实例方法

1. 语法：
2. 定义： def 方法名称(self,参数):

方法体

1. 调用：对象地址.实例方法名称(参数)
2. 说明：
3. 至少有一个形参，用于绑定调用该方法的对象，一般命名为”self”。
4. 实例方法被所有对象共享.
5. 作用：表示对象的行为。

#### 内存图



练习1：画出内存图

class Student:

def \_\_init\_\_(self,name,score):

self.name = name

self.score = score

s01 = Student("张三",100)

s02 = Student("李四",80)

s03 = s01

# 改变对象

s01.score = 200

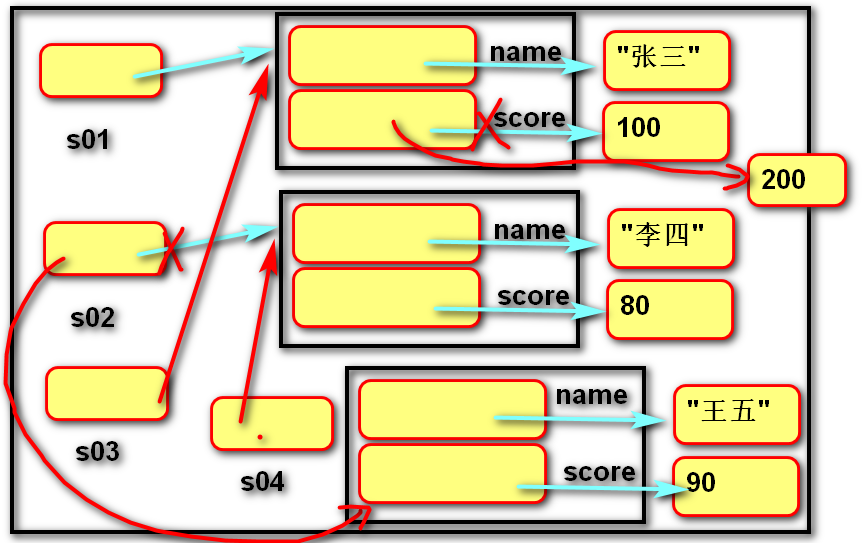
print(s03.score)#

s04 = s02

# 改变变量

s02 = Student("王五",90)

print(s04.score)



练习2：画出内存图

class Student:

def \_\_init\_\_(self,name,score):

self.name = name

self.score = score

def print\_self(self):

print(self.name,self.score)

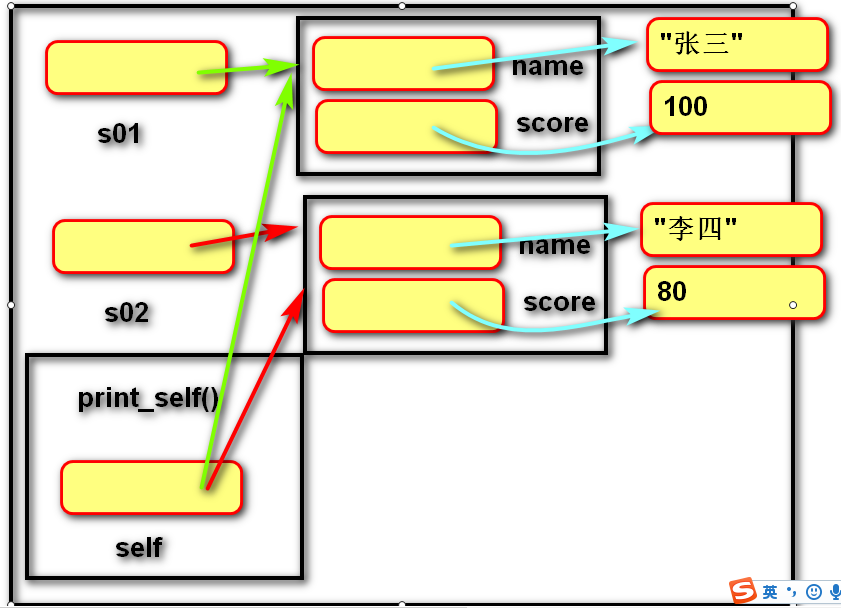
# 画出下列代码内存图：

s01 = Student("张三",100)

s02 = Student("李四",80)

s01.print\_self()

s02.print\_self()



### 类成员

#### 类变量

1. 语法
2. 定义：在类中，方法外定义变量

class 类名:

类变量名 = 数据

1. 调用：

类名.类变量名

1. 说明
2. 存储在类中。
3. 只有一份，被所有对象共享。
4. 描述所有对象的共有数据。

#### 类方法

1. 语法
2. 定义:

@classmethod

def 方法名称(cls,参数):

方法体

1. 调用：类名.方法名(参数)
2. 说明
3. 至少有一个形参，用于绑定调用该方法的类，一般命名为” cls”。
4. 使用@classmethod修饰的目的是调用方法时隐式传递类。
5. 类方法不能访问实例成员，实例方法可以访问类成员。
6. 作用：操作类变量

#### 内存图

class ICBC:

# 类变量：类的,被所有对象共享的数据。

moneys = 10000000

@classmethod

def get\_total\_moneys(cls):

print(cls.moneys)

def \_\_init\_\_(self,name,money):

self.name = name

self.money = money

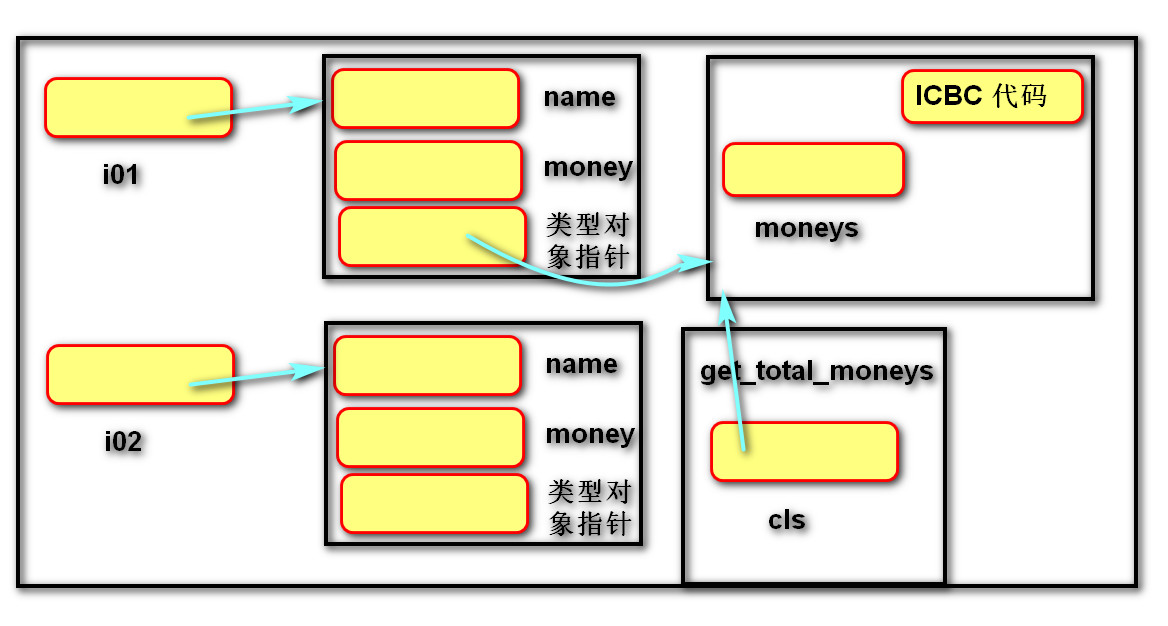
ICBC.moneys -= money

i01 = ICBC("广渠门支行",100000)

ICBC.get\_total\_moneys()

i02 = ICBC("中关村支行",200000)

ICBC.get\_total\_moneys()



练习：对象计数器，统计Student类，总共创建了多少对象。

代码：code10/exercise02

### 静态方法

1. 语法
2. 定义：

@staticmethod

def 方法名称(参数):

方法体

1. 调用：类名.方法名称(参数)
2. 说明
3. 使用@staticmethod修饰的目的是该方法不需要隐式传递参数。
4. 静态方法不能访问实例成员和类成员
5. 作用：统一管理函数(定义在.py文件中的函数)

表达不需要使用实例成员和类成员时，使用静态方法。

练习1：在控制台中录入学生信息（姓名，成绩，性别，年龄）

步骤：1.创建类，定义4个数据成员

2. 在控制台中循环3次，录入学生成绩。

作业1：画出下列代码内存图

class Student:

all\_stu\_count = 0

def \_\_init\_\_(self,name = "",score = 0,sex=0,age=0):

self.name = name

self.score = score

self.sex = sex

self.age = age

def print\_self(self):

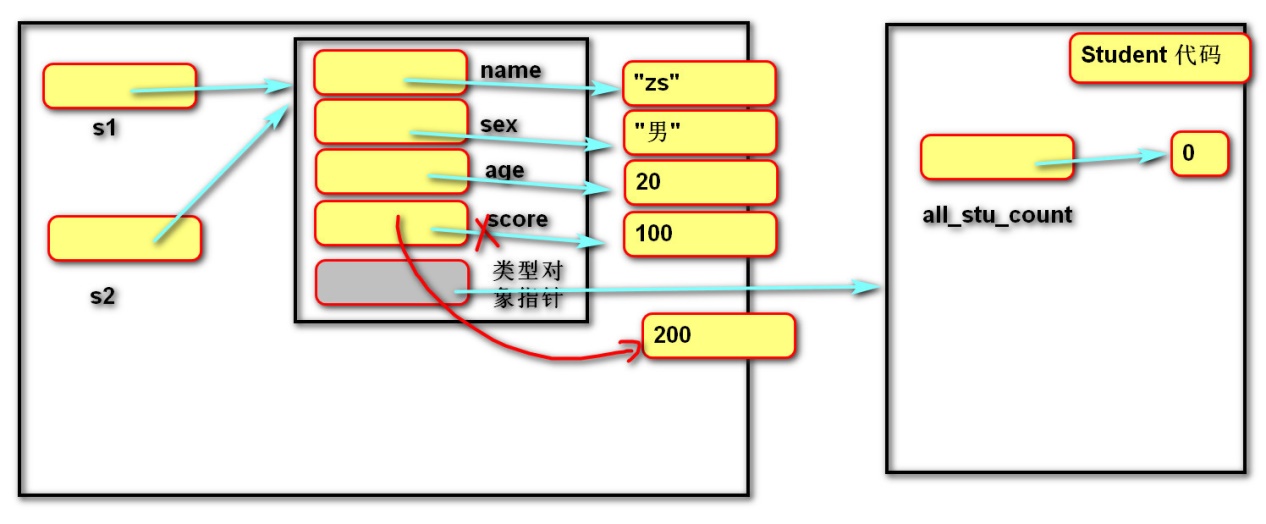
pass

s1 = Student("zs",100,"男",10)

s2 = s1

s2.score = 200

print(s1.score)



作业2：在控制台中录入汽车信息(类型，速度，重量)，按e键退出，最后将每个信息显示在控制台中。

作业3：在控制台中录入狗的信息(姓名，年龄，性别)，按e键退出，最后将每个信息显示在控制台中。

练习4：将昨天开放作业中，定义的类，在控制台中录入信息，按e键退出，最后将每个信息显示在控制台中。

## 封装

### 定义

1. 从数据角度讲：

将一些基础变量复合为一个自定义类型。

比如：向量(x,y,z 模长等操作) 狗(姓名,年龄,拉臭臭等行为)

不但可以准确的描述事物，还可以体现该事物的行为。

1. 从行为角度讲：

向类外提供必要的功能，隐藏实现的细节。

比如：random.randint(1,100)

使用者可以不必操心实现过程。

1. 从设计角度讲：
2. 分而治之

-- 将一个大的需求分解为许多类，让每个类处理一个独立的功能。

-- 优点：便于分工，便于复用，可扩展性强。

1. 封装变化

-- 需求可能会变化的功能要单独封装，避免影响其他类。

1. 高内聚

-- 类中各个方法都在完成一项任务（类中所有成员负责一件事）

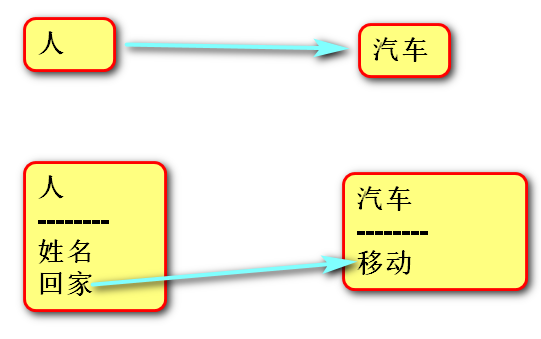
1. 低耦合

-- 类与类的关联性与依赖度要低，让一个类变化，尽少影响其他的类。

例如：活字印刷术

计算机硬件(主板/CPU/内存条/显卡/硬盘….)

案例：张三开车回家



练习：小明在招商银行取钱。

### 私有成员

代码：day12/code01

1. 做法：命名使用双下划线开头
2. 作用：修改变量名，让外界”不能直接访问”
3. 本质：障眼法，也可以访问：

\_类名\_\_成员名

练习：定义学生类(姓名，年龄)

要求：使用方法封装数据。

代码：day12/exercise01

### 属性

代码：day11/code02

公开的实例变量，缺少逻辑验证。私有的实例变量与两个公开的方法相结合，又显得调用者的操作略显复杂。而属性可以将两个方法的使用方式像操作变量一样方便。

1. 定义：

@property # 读取数据时执行

def 属性名(self):

return self.\_\_ 属性名 # 私有的实例变量

@属性名.setter # 写入数据时执行

def 属性名(self,参数):

self.\_\_ 属性名 = 参数

1. 调用：

对象地址.属性名 = 数据

变量 = 对象地址.属性名

1. 说明：
2. 通常两个公开属性，保护一个私有的变量。
3. @property 负责处理读取逻辑
4. @属性名.setter 负责处理写入逻辑

练习：定义学生类(姓名，年龄，成绩，性别)

要求：用属性封装数据。

代码：day11/exercise02

### \_\_slots\_\_

1. 作用：限制一个类创建的对象只能有固定的实例变量，不能再额外添加。
2. 语法：

在类中定义 \_\_slots\_\_ = (“变量名1”,”变量名2”)

1. 说明：
2. 含有\_\_slots\_\_属性的类所创建的对象没有\_\_dict\_\_属性。
3. 优点：防止因写错属性名称而发生的错误。
4. 缺点：丧失了动态语言可以在运行时为对象添加变量的灵活性。

作业1：穷尽一切手段，收集面向对象，第一大特征，封装的设计思想的资料，整理。

最后形成自己的体会心得。

作业2：定义技能数据类(技能编号，技能名称，消耗法力，冷却时间，动画名称)，

使用属性进行封装，使用\_\_slots\_\_。

扩展作业3：使用代码描述一下场景

张三 教 李四 学王者荣耀。

李四 教 张三 学Python

李四 上班赚了 5000元钱

最后输出张三具有的技能，李四具有的技能，以及他们的钱。