# 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)

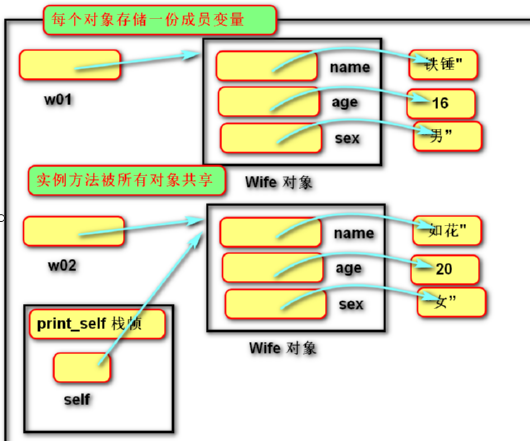
不建议的用法：

可以通过类名访问，并传递对象地址。

类名.实例方法名(对象地址,参数)

1. 说明：
2. 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”。
3. 实例方法被所有对象共享.
4. 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

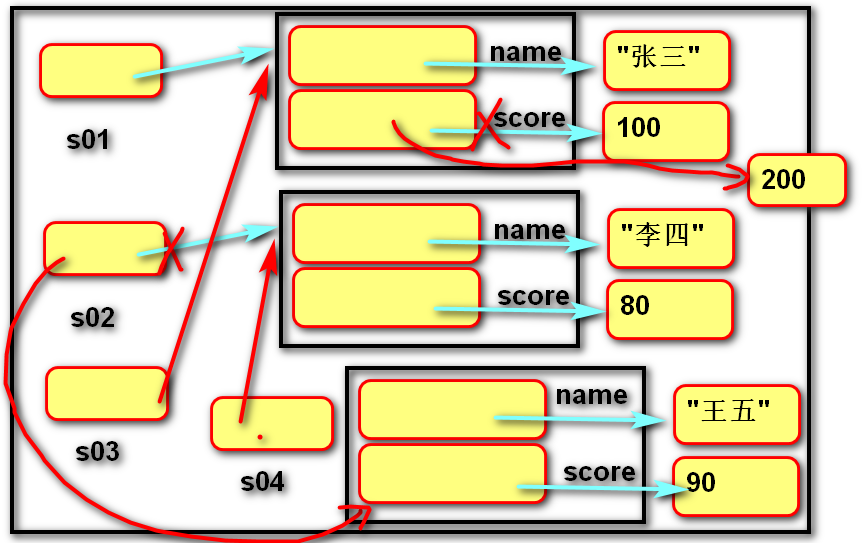
print(s03.score)#

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)

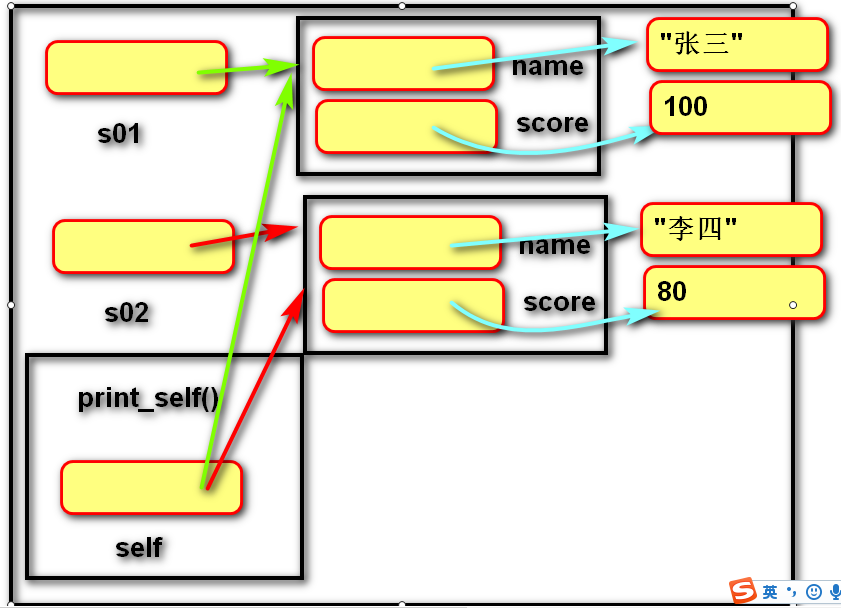
# 画出下列代码内存图：

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

不建议使用对象.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)

不建议使用对象.类方法名.

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

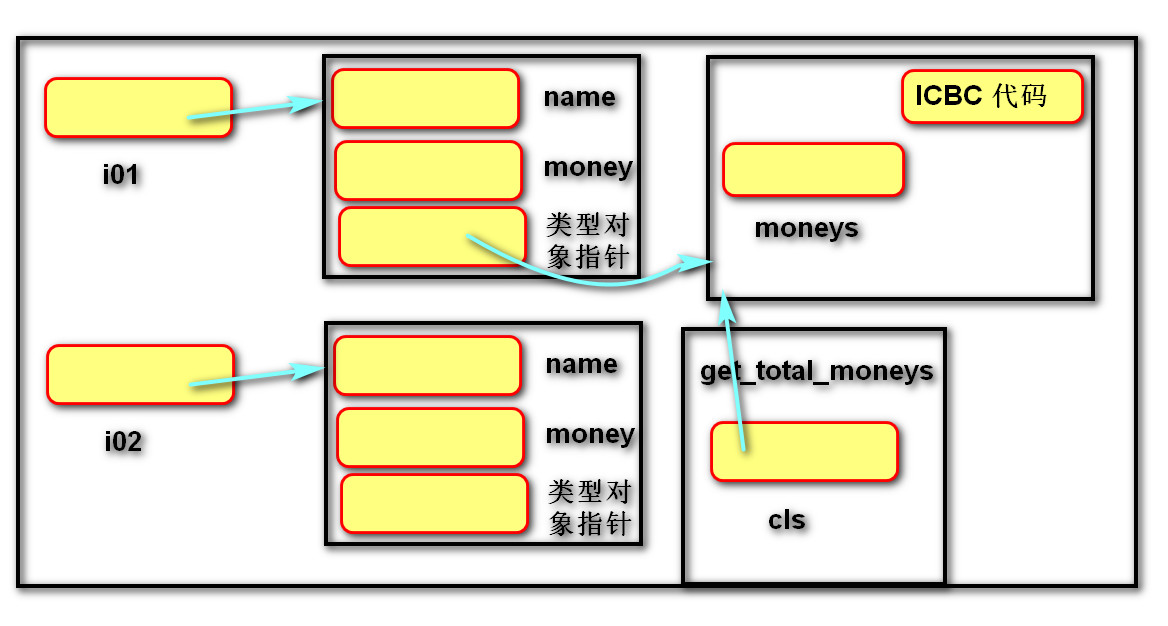
ICBC.moneys -= money

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)

不建议使用对象.静态方法名.

1. 说明
2. 使用@staticmethod修饰的目的是该方法不需要隐式传递参数。
3. 静态方法不能访问实例成员和类成员
4. 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):

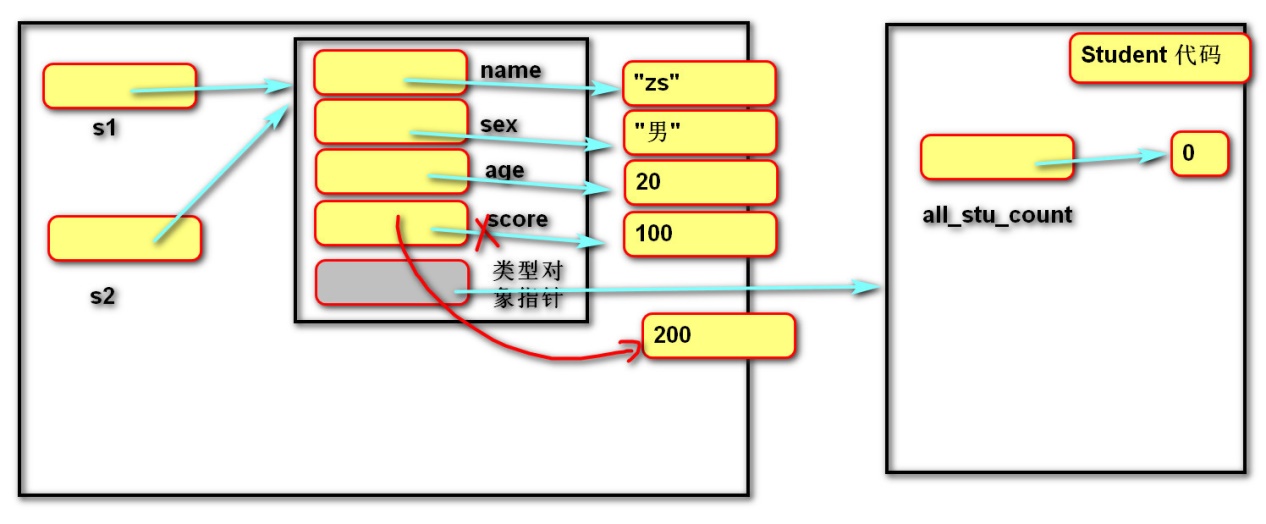
pass

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

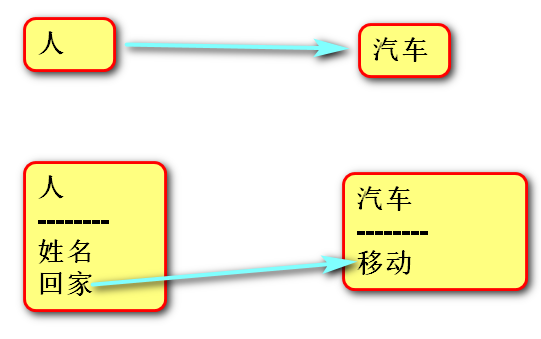
1. 低耦合

-- 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：day11/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：day11/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
5. 只读属性

def \_\_init\_\_(self,参数):

…

self.\_\_属性名 = 100 # 创建私有变量

@property

def 属性名(self):

return self.\_\_属性名

1. 只写属性

def \_\_init\_\_(self,参数):

self.\_\_属性名 = 默认值 # 创建私有变量

def \_\_set\_属性名(self, value):

self.\_\_属性名 = value

属性名 = property(None, \_\_set\_属性名)

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。

day12作业

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。

day13作业：

定义技能数据类(技能编号，技能名称，消耗法力，冷却时间)

作业1：在技能数据列表中，查找指定名称的技能对象。

作业2：查找冷却时间大于5的所有技能对象。

作业3：查找技能数据列表中，消耗法力最小的技能。

作业4：查找技能数据列表中，冷却时间最大的技能。

作业5：根据冷却时间，对技能列表进行升序(小到大)排列。

### 案例：学生管理系统

#### 需求

实现对学生信息的增加、删除、修改、查询。

界面：

1. 添加学生信息
2. 显示学生信息
3. 删除学生信息
4. 修改学生信息
5. 按成绩做升序排列
6. ……

#### 分析

界面可能使用控制台，也可能使用Web….

1. 识别对象：界面视图类、逻辑控制类、数据模型类。
2. 分配职责：

-- 界面视图类:负责处理界面逻辑。

比如：显示菜单，跳转其他界面；获取输入，显示结果。

-- 逻辑控制类:负责存储学生信息，处理业务逻辑。

比如：增加、删除、修改、查询。

-- 数据模型类：定义需要处理的数据类型。

比如：学生信息。

1. 建立交互：

界面视图对象 <---> 数据模型对象 <---> 逻辑控制对象

#### 设计

数据模型类：StudnetModel

数据：编号id,姓名,name，年龄age，成绩score。

逻辑控制类：StudentManagerController

数据：学生列表\_\_list\_stu

行为：学生列表只读属性list\_stu，添加学生add\_stduent,

删除学生remove\_student，修改学生update\_student

根据成绩降序排列order\_by\_score

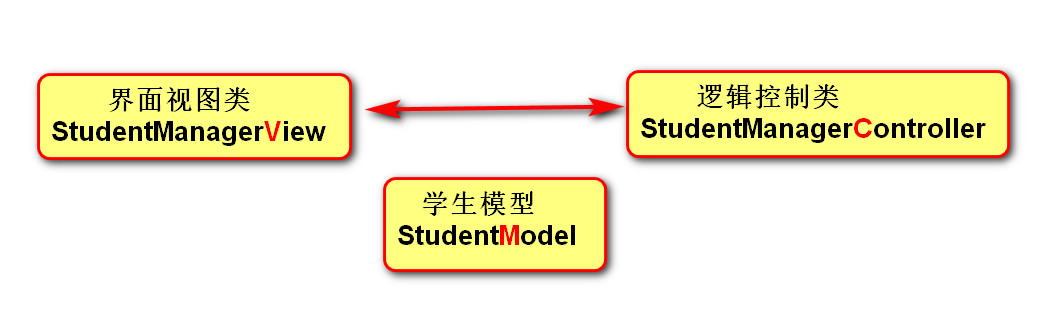
界面视图类：StudentManagerView

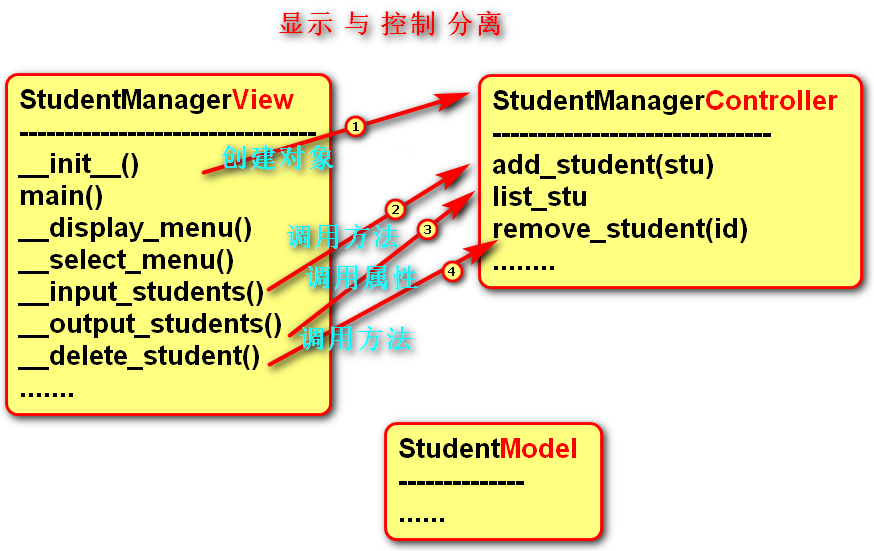
数据：逻辑控制器对象\_\_controller

行为：显示菜单\_\_display\_menu，选择菜单\_\_select\_menu\_item,

入口方法main

#### 架构图





#### 编码

创建文件student\_manager\_system01.py，

（一）完成管理器

定义如下成员：

数据模型类：StudnetModel

数据：编号id,姓名,name，年龄age，成绩score。

逻辑控制类：StudentManagerController

数据：学生列表\_\_list\_stu

行为：学生列表只读属性list\_stu，

添加学生add\_stduent(self,stu)

要求：生成学生编号（根据容器长度确定）

1. 为逻辑控制类添加删除学生方法

remove\_student(self,id):

3．定义修改学生信息的方法 update\_student(self,stu\_info)

4. 按照成绩降序排序order\_by\_score

-----------------------

（二）完成界面视图类

界面视图类：StudentManagerView

数据：逻辑控制器对象\_\_controller

行为：显示菜单\_\_display\_menu，选择菜单\_\_select\_menu\_item,

入口方法main

1. 定义录入学生信息方法input\_students

步骤：

循环录入学生信息，按y继续。

将学生信息封装为学生对象，调用控制器的add\_student方法

#### 时序图（流程）

StudentManagerView. main --> StudentManagerView. \_\_select\_menu

* StudentManagerView. input\_students --> StudentManagerController. add\_student

作业:完成学生管理系统下列功能

添加/删除/显示学生信息

## 继承

### 语法

代码：day15/code01 code02

1. 代码：

class 子类名称(父类名称):

def \_\_init\_\_(self,父类参数,自身参数):

super().\_\_init\_\_(父类参数)

self.自身实例变量 =自身参数

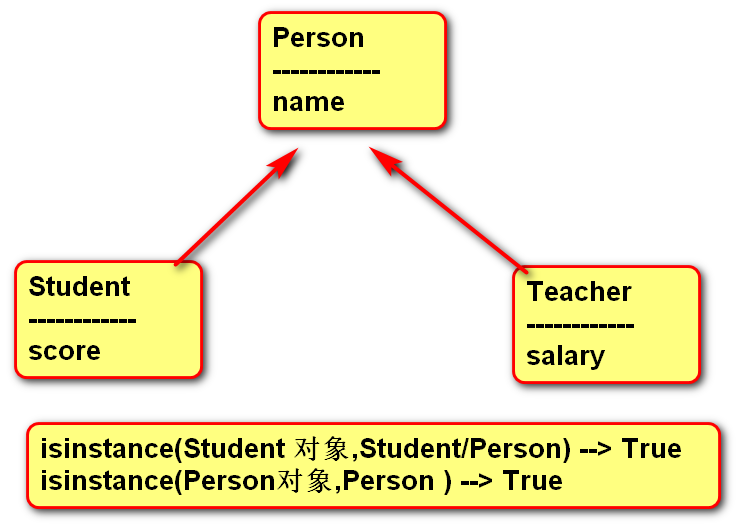
1. 说明：

-- 子类拥有父类所有成员.

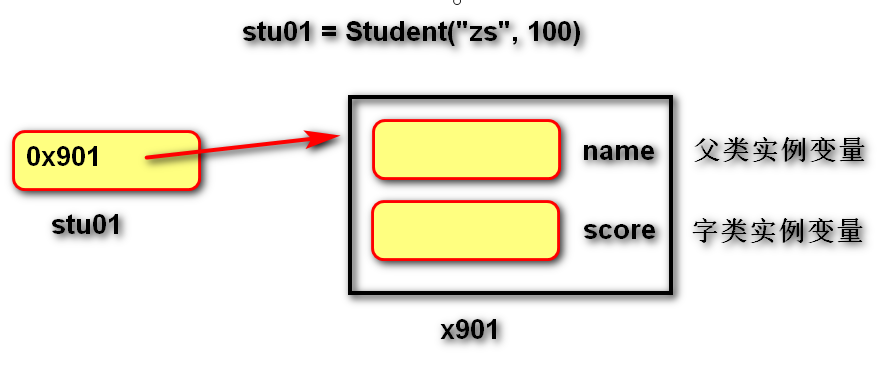
-- 子类如果没有构造函数，将自动执行父类的，但如果有构造函数将覆盖父类的。

此时必须通过super()函数调用父类的构造函数，以确保父类数据成员被正常创建。

1. isisntance（对象，类型）函数:返回对象是否兼容类型。



### 内存图



练习1：

定义宠物类，数据：姓名

定义狗类，数据：工作

定义猫类，行为：抓。

分别创建三个类对象，调用各自成员。

体会：继承的语法现象。

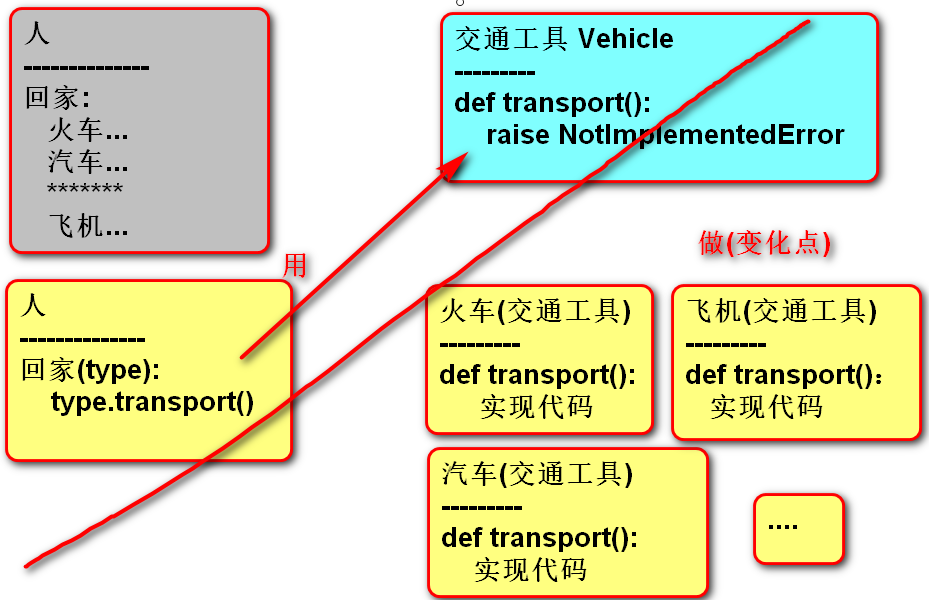
使用isinstance函数，测试各个对象的兼容性。

代码：day15/exercise01

案例： 老张开车回家

需求变化：可能坐飞机/火车/.....

代码：day10/code03



练习：

若干个圆形(面积：半径平方\*pi)，若干个矩形(面积：长\*款)。

定义图形管理器，定义记录所有图形(圆形，矩形)的变量.

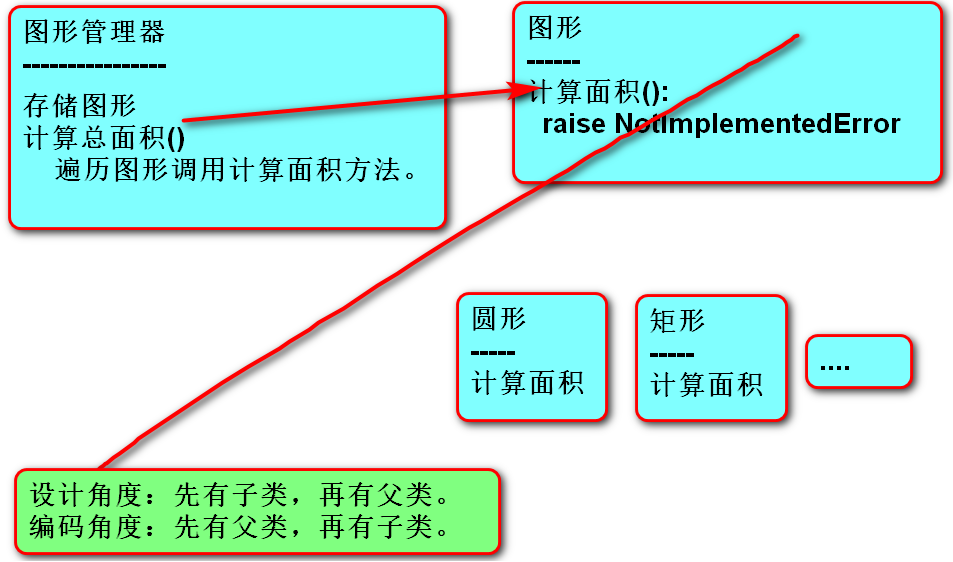
定义计算所有图形面积的方法。

需求变化：可能会增加新的图形(三角形…..)。

要求：代码体现开闭原则，依赖倒置原则。

验收：架构图、说出哪里体现了面向对象设计原则、代码实现。

代码：day15/exercise02



### 定义

1. 定义：重用现有类的功能与概念，并在此基础上进行扩展。
2. 说明：

-- 子类直接具有父类的成员，还可以具有自己的功能。

-- 事物具有一定的层次、渊源，继承可以统一概念。

例如：交通工具统一了火车、汽车、飞机

公司的组织架构

老板

行政中心 营销中心 技术中心

人力资源 行政部 销售部 市场部 研发部 产品部

### 优点

1. 一种代码的复用方式。
2. 以层次化的方式管理类。

-- 人回家，使用交通工具，不关心火车、汽车等

### 缺点

1. 子类与父类耦合度高(父类构造函数/成员的变化，直接影响所有子类)
2. 切换不灵活。

人可以使用手机或者座机或者网络电话。

### 适用性

多个类在概念上是一致的，且需要进行统一的处理。

### 相关概念

父类(基类/超类)、子类(派生类).

父类相对于子类更抽象，使用范围更宽泛；

子类相对于父类更具体，使用范围更狭小。

单继承：父类只有一个(例如：java，c#,……)

多继承：父类有多个(例如：python、c++)

Object类：任何类都直接或者间接继承自Object类。

(万类之祖)

### 多继承

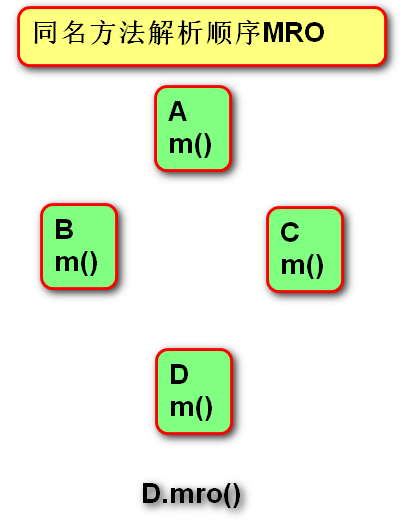
一个子类继承两个或两个以上的基类，父类中的成员同时被子类继承下来。

同名方法的解析顺序（MRO）：

C3算法模拟的广度优先。

类自身 --> 父类继承列表(由左至右) --> 再上层父类

代码：day17/code01



## 多态

### 定义

调用父类同一个方法，但在不同子类间，有不同的表现。

例如：回家方法，调用交通工具的运输方法，实际执行的是火车或者飞机运输方法。

图形管理器……

### 作用

1. 体现子类的个性化。
2. 增强程序扩展性，体现开闭原则。

作业：

1. 穷尽一切手段，收集面向对象三大特征的资料，结合课堂所讲，自述到word文档中。
2. 一家公司有如下几种岗位，

程序员：底薪 + 项目分红

软件测试：底薪 + Bug数 \* 5

销售：底薪 + 销售额\*5%

定义员工管理器，记录所有员工，计算所有员工的总薪资。

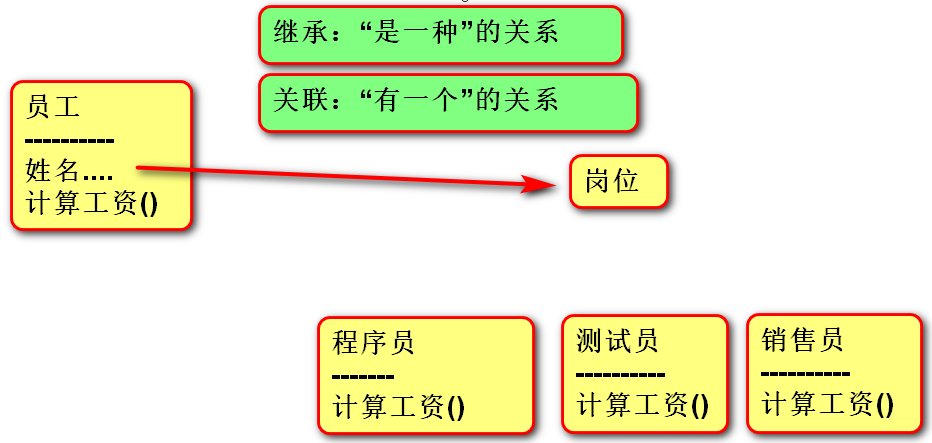
要求：增加新的岗位，员工管理器满足开闭原则。

画出架构图，写出体现依赖倒置原则、开闭原则的点。

代码：day16/day15\_exercise/exercise01

案例：员工转岗

代码：day16/code01



### 重写

子类实现了父类中相同(名称、参数)的方法，在调用该方法时，实际执行的是子类的方法。

#### 内建函数重写

代码：day16/code03 code04

在python中，以双下划线开头和结尾的函数，叫做内建函数。

我们可以在自定义类中对其重写，以改变实现细节。

1. 对象转换为字符串：

-- \_\_str\_\_:重写的时候，返回给人看的友好的支持。

--\_\_repr\_\_:重写的时候，返回解释器可以识别(eval)的字符串。

2. 自定义类的对象可以使用运算符，本质就是调用内建方法。

(详见…..)

## 类与类的关系

代码：day16/code02

### 泛化(继承)

子类与父类的关系，耦合度最高；

B类泛化A类，意味着B类是A类的子类。

做法：B类继承A类。

### 关联(聚合/组合)

部分与这个整体的关系，耦合度要低于泛化。

A类关联B类，意味着B是A的一部分。

做法：在A类中包含B类的成员。

### 依赖

合作关系，耦合度最低。

A类依赖B类，意味着A类的某个功能以依靠B类实现。

做法：在A类的某个方法中，将B类作为参数。

## 设计原则

### 开闭原则(目标)

对扩展开放，对修改关闭。

允许增加新功能，不修改客户端(使用者)代码。

### 类的单一职责(类的定义)

一个类有且只有一个改变的原因。

外界一个需求的变化，内部一个改变的类。

### 依赖倒置(找爹)

客户端代码尽量依赖(使用)抽象(父)的组件。

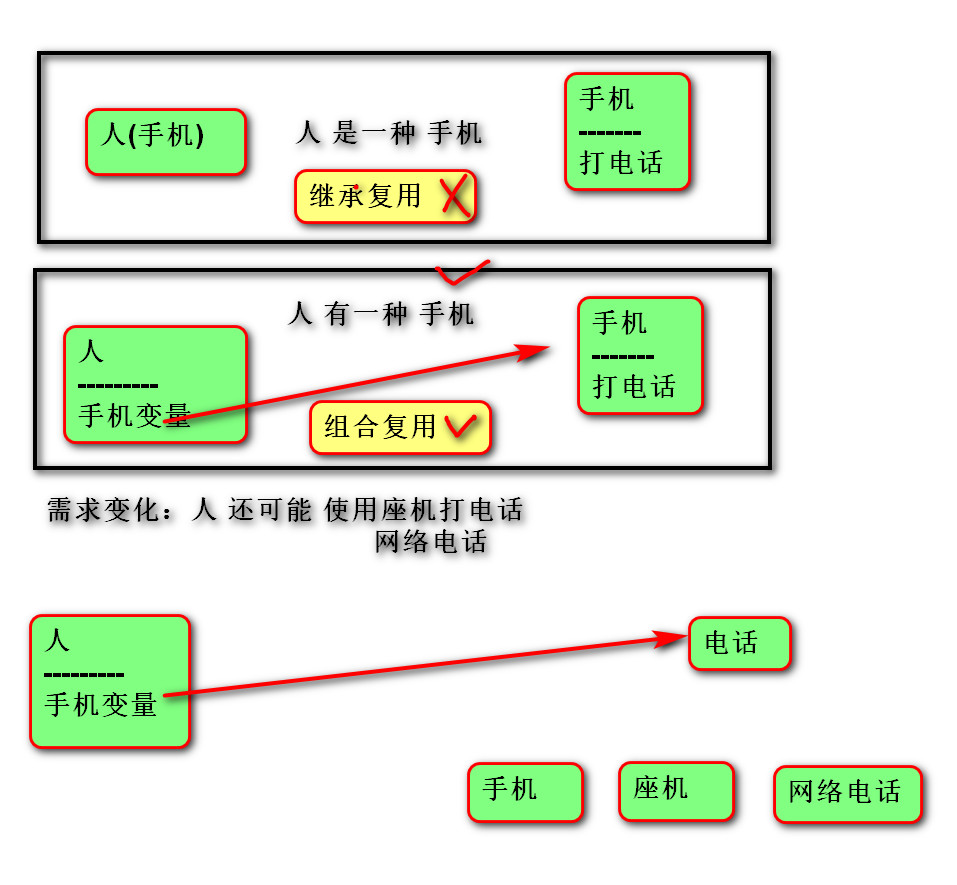
抽象的是稳定的，实现是多变的。

### 组合复用原则(少用继承)

如果仅仅为了代码复用优先选择组合关系，而非继承关系。

组合的耦合度低于继承，灵活度高于继承。

例如：人可以使用手机/座机/网络电话.



### 里氏替换(重写注意事项)

父类出现的地方可以被子类替换，在替换后依然保持原有功能。

-- 从内存角度解释：父类(成员少) 子类(成员多)

-- 子类在重写父类方法时，尽量选择扩展重写(先调父类同名方法)，不要改变原有功能。

### 迪米特法则(低耦合)

原话：不要和陌生人说话。

类与类交互时，在满足功能的基础上，传递的数据量越少越好。

低耦合。

作业：

1. 穷尽一切手段，在互联网中收集”面向对象设计原则”,结合课堂所讲，写一份自述。
2. 根据天龙八部游戏技能策划案，分析设计类(图)。

尝试：一个技能一个类。

## 总结

### 主要思想

封装变化、隔离变化、执行变化。

**三大特征**  
 封装：分而治之、封装变化、高内聚、低耦合。

继承：重用现有类功能和概念，在此基础上进行扩展。

多态：继承体现的是共性，多态体现的是个性。

### 三大关系

泛化：父子关系

关联：成员关系

依赖：协作关系(做成参数)

### 六大原则

开闭原则：对扩展开放，对修改关闭。

单一原则：一个类有且只有一个改变的原因。  
 依赖倒置：使用抽象(父类),不使用具体(子类).

抽象不依赖于具体.

组合复用：使用关联关系，不使用泛化关系。

里氏替换：父类出现的地方，可以被子类替换，替换后可以保持原功能。

迪米特法则：低耦合。

### 优势

高复用：没有重复的代码。

高扩展：开闭原则。

高维护：逻辑清晰，结构规整。

通俗的讲：灵活。

# 技能系统架构设计

代码：day17/day16\_exercise/exercise01

