# Python 简介

1. 程序员：

程序设计人员。

1. 程序：

一组计算机能识别和执行的指令，是实现某种需求的软件。

1. 操作系统：

管理和控制计算机软件与硬件资源的程序；

隔离不同硬件的差异，使开发程序简单化。

例如，Windows，Linux，Unix。

1. 硬件：

主板--计算机的主要电路系统。

CPU --主要负责执行程序指令，处理数据。

硬盘--持久化存储数据的记忆设备，容量大，速度慢。

内存--临时存储数据的记忆设备，容量小，速度快。

IO设备--键盘、鼠标、显示器。

## Python 定义



是一个免费、开源、跨平台、动态、面向对象的编程语言。

## Python程序的执行方式

### 交互式

在命令行输入指令，回车即可得到结果。

1. 打开终端
2. 进入交互式：python3
3. 编写代码：print(“hello world”)
4. 离开交互式：exit()

### 文件式

将指令编写到.py文件，可以重复运行程序。

1. 编写文件。
2. 打开终端
3. 进入程序所在目录：cd 目录
4. 执行程序： python3 文件名

## Linux常用命令

1. pwd：查看当前工作目录的路径
2. ls：查看指定目录的内容或文件信息
3. cd：改变工作目录（进入到某个目录）

练习：

1. 在指定目录创建python文件.

--目录：/home/tarena/1905/month01

--文件名：exercise01.py

2. 在文件中写入：print(“你好，世界!”)

3. 运行python程序

## 执行过程



计算机只能识别机器码(1010)，不能识别源代码(python)。

1. 由源代码转变成机器码的过程分成两类：编译和解释。
2. 编译：在程序运行之前，通过编译器将源代码变成机器码，例如：C语言。

-- 优点：运行速度快

-- 缺点：开发效率低，不能跨平台。

1. 解释：在程序运行之时，通过解释器对程序逐行翻译，然后执行。例如Javascript

-- 优点：开发效率高，可以跨平台；

-- 缺点：运行速度慢。

1. python是解释型语言，但为了提高运行速度，使用了一种编译的方法。编译之后得到pyc文件，存储了字节码（特定于Python的表现形式，不是机器码）。

源代码 -- 编译 --> 字节码 -- 解释 --> 机器码

|————1次———|

## 解释器类型

1. CPython（C语言开发)
2. Jython (java开发)
3. IronPython (.net开发)

# 小结

1. python：免费，开源，跨平台，动态，面向对象的编程语言

2. 执行方式：交互式

　　　　　　　文件式

3. 执行过程：源代码-编译->字节码-解释->机器码

|-----1次-----|-----每次---|

# 数据基本运算

## pycharm常用快捷键

1. 移动到本行开头：home键
2. 移动到本行末尾：end键盘
3. 注释代码：ctrl + /
4. 复制行：ctrl +d
5. 选择列：鼠标左键 + alt
6. 移动行：shift + alt + 上下箭头
7. 智能提示：Ctrl + Space

## 注释

给人看的，通常是对代码的描述信息。

1. 单行注释：以#号开头。
2. 多行注释：三引号开头，三引号结尾。

## 函数

表示一个功能，函数定义者是提供功能的人，函数调用者是使用功能的人。

例如：

1. print(数据) 作用：将括号中的内容显示在控制台中
2. 变量 = input(“需要显示的内容”) 作用：将用户输入的内容赋值给变量

## 变量

1. 定义：关联一个对象的标识符。
2. 命名：必须是字母或下划线开头，后跟字母、数字、下划线。

不能使用关键字(蓝色)，否则发生语法错误：SyntaxError: invalid syntax。

1. 建议命名：字母小写，多个单词以下划线隔开。

class\_name

1. 赋值：创建一个变量或改变一个变量关联的数据。
2. 语法：变量名 = 数据

变量名1 = 变量名2 = 数据

变量名1, 变量名2, = 数据1, 数据2

## del 语句

1. 语法:

del 变量名1, 变量名2

1. 作用：

用于删除变量,同时解除与对象的关联.如果可能则释放对象。

1. 自动化内存管理的引用计数：

每个对象记录被变量绑定(引用)的数量,当为0时被销毁。

## 核心数据类型

1. 在python中变量没有类型，但关联的对象有类型。
2. 通过type函数可查看。

### 空值对象 None

1. 表示不存在的特殊对象。
2. 作用：占位和解除与对象的关联。

### 整形int

1. 表示整数，包含正数、负数、0。

如： -5, 100, 0

1. 字面值：

十进制：5

二进制：0b开头，后跟1或者1

八进制：0o开头，后跟0~7

十六进制：0x开头，后跟0~9,A~F,a~f

1. 小整数对象池：CPython 中整数 -5 至 256,永远存在小整数对象池中,不会被释放并可重复使用。

### 浮点型float

1. 表示小数，包含正数、负数，0.0)。
2. 字面值：

小数：1.0 2.5

科学计数法：e/E (正负号) 指数

1.23e-2 (等同于0.0123)

1.23456e5(等同于123456.0)

### 字符串str

是用来记录文本信息(文字信息)。

字面值：双引号

### 复数complex

由实部和虚部组成的数字。

虚部是以j或J结尾。

字面值： 1j 1+1j 1-1j

### 布尔bool

用来表示真和假的类型

True 表示真(条件满足或成立)，本质是1

False 表示假(条件不满足或不成立)，本质是0

## 数据类型转换

1. 转换为整形: int(数据)
2. 转换为浮点型:float(数据)
3. 转换为字符串:str(数据)
4. 转换为布尔:bool(数据)

结果为False：bool(0) bool(0.0) bool(None)

1. 混合类型自动升级：

1 + 2.14 返回的结果是 3.14

1 + 3.0 返回结果是: 4.0

## 运算符

### 算术运算符

+ 加法

- 减法

\* 乘法

/ 除法：结果为浮点数

// 地板除：除的结果去掉小数部分

% 求余

\*\* 幂运算

优先级从高到低： ()

\*\*

\* / % //

+ -

### 增强运算符

y += x 等同于 y = y + x

y -= x 等同于 y = y - x

y \*= x 等同于 y = y \* x

y /= x 等同于 y = y / x

y //= x 等同于 y = y // x

y %= x 等同于 y = y % x

y \*\*= x 等同于 y = y \*\* x

### 比较运算符

< 小于

<= 小于等于

> 大于

>= 大于等于

== 等于

!= 不等于

返回布尔类型的值

比较运算的数学表示方式:0 <= x <= 100

### 逻辑运算符

#### 与and

表示并且的关系，一假俱假。

示例:

True and True # True

True and False # False

False and True # False

False and False # False

#### 或or

表示或者的关系，一真俱真

示例:

True or True # True

True or False # True

False or True # True

False or False # False

#### 非 not

表示取反

例如：

not True # 返回False

not False # 返回True

#### 短路运算

一但结果确定，后面的语句将不再执行。

### 身份运算符

语法:

x is y

x is not y

作用：

is 用于判断两个对象是否是同一个对象,是时返回True,否则返回False。

is not 的作用与is相反

### 优先级

高到低：

算数运算符

比较运算符

快捷(增强?)运算符

身份运算符

逻辑运算符

# 小结

数据基本运算

变量：关联一个对象的标识符

变量名　＝　？

　变量没有类型

数据类型:

None

int 1 2

float　　　1.0 2.5

str "" "字符"

bool　　　True False

复数　complex

类型转换

int(数据)　　float(数据)

str(数据) bool(数据)

如果数据的格式不正确，会错误。

例如：int("100+")

如果数据表示"没有",转换结果为Ｆａｌｓｅ

bool(1) --> True

bool("") -->False

运算符

算数运算符：＋　－　＊　　／　／／　％　\*\*

增强运算符:＋=　－=　＊=　　／=　／／=　％=　\*\*=

a = 10

a = a + 5

a += 5

比较运算符:> < >= <= == !=

逻辑运算符: 1 > 2 "a" == "b"

False or False

与　and : 一假俱假

或　or :一真俱真

"""

a = 1

a = **"ａ"**

a = **True**

# 问题：控制台中会出现什么

# 短路逻辑：逻辑运算时，尽量将复杂(耗时)的判断放在后边。

num = 1

# and 发现Ｆａｌｓｅ，就有了结论,后续条件不再判断。

# re = num > 1 and input("请输入：") == "a"

# or 发现Ｔｒｕｅ，就有了结论,后续条件不再判断。

re = num + 1 > 1 **or** input(**"请输入："**) == **"a"**

# 语句

## 行

1. 物理行：程序员编写代码的行。
2. 逻辑行：python解释器需要执行的指令。
3. 建议一个逻辑行在一个物理行上。
4. 如果一个物理行中使用多个逻辑行，需要使用分号；隔开。

5. 如果逻辑行过长，可以使用隐式换行或显式换行。

隐式换行：所有括号的内容换行,称为隐式换行

括号包括: () [] {} 三种

显式换行：通过折行符 \ (反斜杠)换行，必须放在一行的末尾，目的是告诉解释器,下一行也是本行的语句。

## pass 语句

通常用来填充语法空白。

## 选择语句

### If elif else 语句

1. 作用:

让程序根据条件选择性的执行语句。

1. 语法:

if 条件1:

语句块1

elif 条件2:

语句块2

else:

语句块3

1. 说明:

elif 子句可以有0个或多个。

else 子句可以有0个或1个，且只能放在if语句的最后。

### if 语句的真值表达式

if 100:

print("真值")

等同于

if bool(100):

print("真值")

### 条件表达式

语法：变量 = 结果1 if 条件 else 结果2

作用：根据条件(True/False) 来决定返回结果1还是结果2。

## 循环语句

### while语句

1. 作用:

可以让一段代码满足条件，重复执行。

1. 语法:

while 条件:

满足条件执行的语句

else:

不满足条件执行的语句

1. 说明:

else子句可以省略。

在循环体内用break终止循环时,else子句不执行。

### for 语句

1. 作用:

用来遍历可迭代对象的数据元素。

可迭代对象是指能依次获取数据元素的对象，例如：容器类型。

1. 语法:

for 变量列表 in 可迭代对象:

语句块1

else:

语句块2

1. 说明:

else子句可以省略。

在循环体内用break终止循环时,else子句不执行。

### range 函数

1. 作用:

用来创建一个生成一系列整数的可迭代对象(也叫整数序列生成器)。

1. 语法:

range(开始点，结束点，间隔)

1. 说明:

函数返回的可迭代对象可以用for取出其中的元素

返回的数字不包含结束点

开始点默认为0

间隔默认值为1

## 跳转语句

### break 语句

1. 跳出循环体，后面的代码不再执行。
2. 可以让while语句的else部分不执行。

### continue 语句

跳过本次，继续下次循环。

# 小结

*day03 复习*

*语句*

*选择语句*

*if bool类型的条件:*

*满足条件执行的语句*

*else:*

*不满足条件执行的语句*

*if 条件1:*

*满足条件1执行的语句*

*if 条件2:*

*满足条件2执行的语句*

*if 条件3:*

*满足条件3执行的语句*

*if 条件1:*

*满足条件1执行的语句*

*elif 条件2:*

*不满足条件１，满足条件2执行的语句*

*elif 条件3:*

*不满足条件１/2,满足条件3执行的语句*

*else:*

*以上条件都不满足执行的语句*

*循环语句*

*if 条件：*

*满足条件执行一次*

*else:*

*不满足条件执行一次*

*while 条件:*

*满足条件一直执行*

*else:*

*不满足条件执行一次*

*跳转语句*

*break*

*循环语句*

*for　+ range()：固定次数的循环*

*while:根据条件执行的循环*

*range(开始,结束,步长)*

*range(2,6,2)->2 4*

*range(2)->0 1*

*range(2,2)->*

# 容器类型

## 通用操作

### 数学运算符

1. +：用于拼接两个容器
2. +=：用原容器与右侧容器拼接,并重新绑定变量
3. \*：重复生成容器元素
4. \*=：用原容器生成重复元素, 并重新绑定变量
5. < <= > >= == !=：依次比较两个容器中元素,一但不同则返回比较结果。

### 成员运算符

1. 语法：

数据 in 序列

数据 not in 序列

1. 作用：

如果在指定的序列中找到值，返回bool类型。

### 索引index

1. 作用：访问容器元素
2. 语法：容器[整数]
3. 说明：

正向索引从0开始，第二个索引为1，最后一个为len(s)-1。

反向索引从-1开始,-1代表最后一个,-2代表倒数第二个,以此类推,第一个是-len(s)。

### 切片slice

1. 作用：

从容器中取出相应的元素重新组成一个容器。

1. 语法：

容器[(开始索引):(结束索引)(:(步长))]

1. 说明：

小括号()括起的部分代表可省略

结束索引不包含该位置元素

步长是切片每次获取完当前元素后移动的偏移量

### 内建函数

1. len(x) 返回序列的长度
2. max(x) 返回序列的最大值元素
3. min(x) 返回序列的最小值元素
4. sum(x) 返回序列中所有元素的和(元素必须是数值类型)

## 字符串 str

### 定义

由一系列字符组成的不可变序列容器，存储的是字符的编码值。

### 编码

1. 字节byte：计算机最小存储单位，等于8 位bit.
2. 字符：单个的数字，文字与符号。
3. 字符集(码表)：存储字符与二进制序列的对应关系。
4. 编码：将字符转换为对应的二进制序列的过程。
5. 解码：将二进制序列转换为对应的字符的过程。
6. 编码方式：

--ASCII编码：包含英文、数字等字符，每个字符1个字节。

--GBK编码：兼容ASCII编码，包含21003个中文；英文1个字节，汉字2个字节。

--Unicode字符集：国际统一编码，旧字符集每个字符2字节，新字符集4字节。

-- UTF-8编码：Unicode的存储与传输方式，英文1字节，中文3字节。

#### 相关函数

1. ord(字符串):返回该字符串的Unicode码。
2. chr(整数):返回该整数对应的字符串。

### 字面值

#### 单引和双引号的区别

1. 单引号内的双引号不算结束符
2. 双引号内的单引号不算结束符

#### 三引号作用

1. 换行会自动转换为换行符\n
2. 三引号内可以包含单引号和双引号
3. 作为文档字符串

#### 转义字符

1. 改变字符的原始含义。

\’ \” \””” \n \\ \t \0 空字符

1. 原始字符串：取消转义。

a = r”C:\newfile\test.py”

#### 字符串格式化

1. 定义：

生成一定格式的字符串。

1. 语法：

字符串%(变量)

"我的名字是%s,年龄是%s" % (name, age)

1. 类型码：

%s 字符串 %d整数 %f 浮点数

## 列表 list

### 定义

由一系列变量组成的可变序列容器。

### 基础操作

1. 创建列表：

列表名 = []

列表名 = list(可迭代对象)

1. 添加元素：

列表名.append(元素)

列表.insert(索引，元素)

1. 定位元素：

索引、切片

1. 遍历列表：

正向：

for 变量名 in 列表名:

变量名就是元素

反向：

for 索引名 in range(len(列表名)-1,-1,-1):

列表名[索引名]就是元素

1. 删除元素：

列表名.remove(元素)

del 列表名[索引或切片]

### 深拷贝和浅拷贝

前提:二维列表

浅拷贝：复制过程中,只复制一层变量,不会复制深层变量绑定的对象的复制过程。

深拷贝：复制整个依懒的变量。

简单来说,就是”内层列表”复制文件夹和创建快捷方式的区别

### 列表VS字符串

1. 列表和字符串都是序列,元素之间有先后顺序关系。
2. 字符串是不可变的序列,列表是可变的序列。
3. 字符串中每个元素只能存储字符,而列表可以存储任意类型。
4. 列表和字符串都是可迭代对象。
5. 函数：

将多个字符串拼接为一个。

result = "连接符".join(列表)

将一个字符串拆分为多个。

列表 = “a-b-c-d”.split(“分隔符”)

### 列表推导式

1. 定义：

使用简易方法，将可迭代对象转换为列表。

1. 语法：

变量 = [表达式 for 变量 in 可迭代对象]

变量 = [表达式 for 变量 in 可迭代对象 if 条件]

1. 说明:

如果if真值表达式的布尔值为False,则可迭代对象生成的数据将被丢弃。

### 列表推导式嵌套

1. 语法：

变量 = [表达式 for 变量1 in 可迭代对象1 for 变量2 in可迭代对象2]

1. 传统写法：

result = []

for r in ["a", "b", "c"]:

for c in ["A", "B", "C"]:

result.append(r + c)

1. 推导式写法：

result = [r + c for r in list01 for c in list02]

## 元组 tuple

### 定义

1. 由一系列变量组成的不可变序列容器。
2. 不可变是指一但创建，不可以再添加/删除/修改元素。

### 基础操作

1. 创建空元组：

元组名 = ()

元组名 = tuple()

1. 创建非空元组：

元组名 = (20,)

元组名 = (1, 2, 3)

元组名 = 100,200,300

元组名 = tuple(可迭代对象)

1. 获取元素：

索引、切片

1. 遍历元组：

正向：

for 变量名 in 列表名:

变量名就是元素

反向：

for 索引名 in range(len(列表名)-1,-1,-1):

元祖名[索引名]就是元素

### 作用

1. 元组与列表都可以存储一系列变量，由于列表会预留内存空间，所以可以增加元素。
2. 元组会按需分配内存，所以如果变量数量固定，建议使用元组，因为占用空间更小。
3. 应用：

变量交换的本质就是创建元组：x, y = y, x

格式化字符串的本质就是创建元祖："姓名:%s, 年龄:%d" % ("tarena", 15)

## 字典 dict

### 定义

1. 由一系列键值对组成的可变映射容器。
2. 映射：一对一的对应关系，且每条记录无序。
3. 键必须惟一且不可变(字符串/数字/元组)，值没有限制。

### 基础操作

1. 创建字典：

字典名 = {键1：值1，键2：值2}

字典名 = dict (可迭代对象)

1. 添加/修改元素：

语法:

字典名[键] = 数据

说明:

键不存在，创建记录。

键存在，修改映射关系。

1. 获取元素：

变量 = 字典名[键] # 没有键则错误

1. 遍历字典：

for 键名 in 字典名:

字典名[键名]

for 键名,值名 in 字典名.items():

语句

1. 删除元素：

del 字典名[键]

### 字典推导式

1. 定义：

使用简易方法，将可迭代对象转换为字典。

1. 语法:

{键:值 for 变量 in 可迭代对象}

{键:值 for 变量 in 可迭代对象 if 条件}

### 字典 VS 列表

1. 都是可变容器。
2. 获取元素方式不同,列表用索引,字典用键。
3. 字典的插入,删除,修改的速度快于列表。
4. 列表的存储是有序的,字典的存储是无序的。

## 集合 set

### 定义

1. 由一系列不重复的不可变类型变量组成的可变映射容器。
2. 相当于只有键没有值的字典(键则是集合的数据)。

### 基础操作

1. 创建空集合：

集合名 = set()

集合名 = set(可迭代对象)

1. 创建具有默认值集合：

集合名 = {1, 2, 3}

集合名 = set(可迭代对象)

1. 添加元素：

集合名.add(元素)

1. 删除元素：

集合名.discard(元素)

### 运算

1. 交集&：返回共同元素。

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 & s2 # {2, 3}

1. 并集：返回不重复元素

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 | s2 # {1, 2, 3, 4}

1. 补集-：返回只属于其中之一的元素

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s1 - s2 # {1} 属于s1但不属于s2

补集^：返回不同的的元素

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 ^ s2 # {1, 4} 等同于(s1-s2 | s2-s1)

1. 子集<：判断一个集合的所有元素是否完全在另一个集合中
2. 超集>：判断一个集合是否具有另一个集合的所有元素

s1 = {1, 2, 3}

s2 = {2, 3}

s2 < s1 # True

s1 > s2 # True

1. 相同或不同== !=：判断集合中的所有元素是否和另一个集合相同。

s1 = {1, 2, 3}

s2 = {3, 2, 1}

s1 == s2 # True

s1 != s2 # False

子集或相同,超集或相同 <= >=

### 集合推导式

1. 定义：

使用简易方法，将可迭代对象转换为集合。

1. 语法:

{表达式 for 变量 in 可迭代对象}

{表达式 for 变量 in 可迭代对象 if 条件}

## 固定集合 frozenset

### 定义

不可变的集合。

#### 作用

固定集合可以作为字典的键,还可以作为集合的值。

#### 基础操作

创建固定集合：frozenset(可迭代对象)

#### 运算

等同于set

# 小结

*字符串str:不可变　　编码值　utf－８*

*字面值*

*单引号　双引号　三引号（所见即所得）*

*转义符* *\字符*

*字符串格式化*

*"...”+变量１+“.."+变量２+".."*

*"...%s..%f.."%(变量１,变量２)*

*通用操作*

*数学运算符* *+ \**

*成员运算符 元素* *in 容器*

*索引:定位单个元素*

*切片：定位多个元素*

*函数:len(容器) 长度*

*字符串:不可变　　存储编码值　　序列*

*列表:可变　　存储变量　　　　　序列*

*基础操作*

*1.创建:[数据] list(容器)*

*2.定位：索引　　切片*

*# 从列表中获取一片元素组成新列表*

*变量* *= 列表名[切片]*

*# 修改一片元素*

*列表名[切片] = 变量*

*3.删除:*

*del 列表名[索引/切片]*

*列表名.remove(元素)*

*从列表中删除多个元素,建议倒序删除.*

*4.增加:*

*列表名.append(元素)*

*列表名.insert(索引,元素)*

*5. 遍历所有元素*

*字符串:不可变 存储编码值 序列*

*列表:可变　存储变量 序列*

*预留空间*

*扩容：开辟更大的空间*

*拷贝原有数据*

*替换引用*

*元组:不可变　存储变量　序列*

*按需分配*

*字典:可变 存储键值对 散列*

*集合:可变 存储键 散列*

*固定集合:不可变 存储键 散列*

*能力提升for for*

*# 结论：外层循环执行一次，内层循环执行多次。*

*外层控制行，内层控制列.*

*for r in range(2):# 0 1*

*for c in range(3):#012 012*

*pass*

# 函数 function

## pycharm相关设置

1. “代码自动完成”时间延时设置

File -> Settings -> Editor -> General -> Code Completion -> Autopopup in (ms):0

1. 快捷键：

Ctrl + P 参数信息（在方法中调用参数）

Ctrl + Q 快速查看文档

Ctrl + Alt + M 提取方法

## 定义

1. 用于封装一个特定的功能，表示一个功能或者行为。
2. 函数是可以重复执行的语句块, 可以重复调用。

## 作用

提高代码的可重用性和可维护性（代码层次结构更清晰）。

## 定义函数

1. 语法：

def 函数名(形式参数):

  函数体

1. 说明：

def 关键字：全称是define，意为”定义”。

函数名：对函数体中语句的描述，规则与变量名相同。

形式参数：方法定义者要求调用者提供的信息。

函数体：完成该功能的语句。

1. 函数的第一行语句建议使用文档字符串描述函数的功能与参数。

## 调用函数

1. 语法：函数名(实际参数)
2. 说明：根据形参传递内容。

## 返回值

1. 定义：

方法定义者告诉调用者的结果。

1. 语法：

return 数据

1. 说明：

return后没有语句，相当于返回 None。

函数体没有return，相当于返回None。

## 可变／不可变类型在传参时的区别

1. 不可变类型参数有:

数值型(整数，浮点数,复数)

布尔值bool

None 空值

字符串str

元组tuple

固定集合frozenset

1. 可变类型参数有:

列表 list

字典 dict

集合 set

1. 传参说明：

不可变类型的数据传参时，函数内部不会改变原数据的值。

可变类型的数据传参时，函数内部可以改变原数据。

## 函数参数

### 实参传递方式argument

#### 位置传参

定义：实参与形参的位置依次对应。

##### 序列传参

定义：实参用\*将序列拆解后与形参的位置依次对应。

#### 关键字传参

定义：实参根据形参的名字进行对应。

##### 字典关键字传参

1. 定义：实参用\*\*将字典拆解后与形参的名字进行对应。
2. 作用：配合形参的缺省参数，可以使调用者随意传参。

### 形参定义方式parameter

#### 缺省参数

1. 语法：

def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):

函数体

1. 说明：

缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则其右侧的所有参数都必须有缺省参数。

缺省参数可以有0个或多个，甚至全部都有缺省参数。

#### 位置形参

语法：

def 函数名(形参名1, 形参名2, ...):

函数体

##### 星号元组形参

1. 语法：

def 函数名(\*元组形参名):

  函数体

1. 作用：

收集多余的位置传参。

1. 说明：

一般命名为'args'

形参列表中最多只能有一个

#### 命名关键字形参

1. 语法：

def 函数名(\*, 命名关键字形参1, 命名关键字形参2, ...):

函数体

def 函数名(\*args, 命名关键字形参1, 命名关键字形参2, ...):  
  函数体

1. 作用：

强制实参使用关键字传参

##### 双星号字典形参

1. 语法：

def 函数名(\*\*字典形参名):

函数体

1. 作用：

收集多余的关键字传参

1. 说明:

一般命名为'kwargs'

形参列表中最多只能有一个

#### 参数自左至右的顺序

位置形参 --> 星号元组形参 --> 命名关键字形参 --> 双星号字典形参

# 小结

*定义:功能，使用一个名称，包装多个语句。*

*语法:*

*做*

*def 名字(形参):*

*函数体*

*用*

*名字(实参)*

*基础语法*

*定义函数*

*def 函数名称(形参):*

*函数体*

*调用函数*

*函数名称(实参)*

*基础概念*

*参数:调用者传递给定义者的信息.*

*定义者要求调用者必须提供的信息.*

*返回值:定义者传递给调用者的结果*

*参数*

*实际参数*

*位置实参:实参与形参按位置对应*

*序列实参：参数过多，可以将实参存储在序列中.*

*用星号拆分后与形参对应.*

*关键字实参：实参与形参按名字对应*

*字典实参：参数过多，可以将实参存储在字典中.*

*用双星号拆分后与形参对应.*

*形式参数*

*默认形参：给形参提供一个默认值，实参可以不提供.*

*位置形参*

*星号元组形参：让位置实参个数无限*

*命名关键字形参：要求实参必须是关键字实参*

*双星号元组形参:让关键字实参个数无限*

# 作用域LEGB

1. 作用域：变量起作用的范围。
2. Local局部作用域：函数内部。
3. Enclosing 外部嵌套作用域 ：函数嵌套。
4. Global全局作用域：模块(.py文件)内部。
5. Builtin内置模块作用域：builtins.py文件。

## 变量名的查找规则

1. 由内到外：L -> E -> G -> B
2. 在访问变量时，先查找本地变量，然后是包裹此函数外部的函数内部的变量，之后是全局变量，最后是内置变量。

## 局部变量

1. 定义在函数内部的变量(形参也是局部变量)
2. 只能在函数内部使用
3. 调用函数时才被创建，函数结束后自动销毁

## 全局变量

1. 定义在函数外部,模块内部的变量。
2. 在整个模块(py文件)范围内访问（但函数内不能将其直接赋值）。

## global 语句

1. 作用：

在函数内部修改全局变量。

在函数内部定义全局变量(全局声明)。

1. 语法：

global 变量1, 变量2, …

1. 说明

在函数内直接为全局变量赋值，视为创建新的局部变量。

不能先声明局部的变量，再用global声明为全局变量。

## nonlocal 语句

1. 作用：

在内层函数修改外层嵌套函数内的变量

1. 语法

nonlocal 变量名1,变量名2, ...

1. 说明

在被嵌套的内函数中进行使用

# 面向对象 Object Oriented

## 概述

### 面向过程

1. 分析出解决问题的步骤，然后逐步实现。

例如：婚礼筹办

-- 发请柬（选照片、措词、制作）

-- 宴席（场地、找厨师、准备桌椅餐具、计划菜品、购买食材）

-- 婚礼仪式（定婚礼仪式流程、请主持人）

1. 公式：程序 = 算法 + 数据结构
2. 优点：所有环节、细节自己掌控。
3. 缺点：考虑所有细节，工作量大。

### 面向对象

1. 找出解决问题的人，然后分配职责。

例如：婚礼筹办

-- 发请柬：找摄影公司（拍照片、制作请柬）

-- 宴席：找酒店（告诉对方标准、数量、挑选菜品）

-- 婚礼仪式：找婚庆公司（对方提供司仪、制定流程、提供设备、帮助执行）

1. 公式：程序 = 对象 + 交互
2. 优点
3. 思想层面：

-- 可模拟现实情景，更接近于人类思维。

-- 有利于梳理归纳、分析解决问题。

(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：增加新的功能，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。

## 类和对象

1. 类：一个抽象的概念，即生活中的”类别”。
2. 对象：类的具体实例，即归属于某个类别的”个体”。
3. 类是创建对象的”模板”。

-- 数据成员：名词类型的状态。

-- 方法成员：动词类型的行为。

1. 类与类行为不同，对象与对象数据不同。

### 语法

#### 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\_ 也叫构造函数，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，名称可以随意。

#### 创建对象(实例化)

变量 = 构造函数 (参数列表)

### 实例成员

#### 实例变量

1. 语法
2. 定义：对象.变量名
3. 调用：对象.变量名
4. 说明
5. 首次通过对象赋值为创建，再次赋值为修改.

w01 = Wife()

w01.name = “丽丽”

w01.name = “莉莉”

1. 通常在构造函数(\_init\_)中创建。

w01 = Wife(“丽丽”,24)

print(w01.name)

1. 每个对象存储一份，通过对象地址访问。
2. 作用：描述实例对象的数据。
3. \_\_dict\_\_：对象的属性，用于存储自身实例变量的字典。

#### 实例方法

1. 语法

(1) 定义： def 方法名称(self, 参数列表):

方法体

(2) 调用： 对象地址.实例方法名(参数列表)

不建议通过类名访问实例方法

2. 说明

(1) 至少有一个形参，第一个参数绑定调用这个方法的对象,一般命名为"self"。

(2) 无论创建多少对象，方法只有一份，并且被所有对象共享。

3. 作用：表示对象行为。

### 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:

变量名 = 表达式

1. 调用：类名.变量名

不建议通过对象访问类变量

1. 说明

-- 存储在类中。

-- 只有一份，被所有对象共享。

1. 作用：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,参数列表):

方法体

1. 调用：类名.方法名(参数列表)

不建议通过对象访问类方法

1. 说明

-- 至少有一个形参，第一个形参用于绑定类，一般命名为'cls'

-- 使用@classmethod修饰的目的是调用类方法时可以隐式传递类。

-- 类方法中不能访问实例成员，实例方法中可以访问类成员。

1. 作用：操作类变量。

### 静态方法

1. 语法
2. 定义：

@staticmethod

def 方法名称(参数列表):

方法体

1. 调用：类名.方法名(参数列表)

不建议通过对象访问静态方法

1. 说明

-- 使用@ staticmethod修饰的目的是该方法不需要隐式传参数。

-- 静态方法不能访问实例成员和类成员

1. 作用：定义常用的工具函数。

## 封装

### 定义

1. 数据角度讲，将一些基本数据类型复合成一个自定义类型。
2. 行为角度讲，向类外提供必要的功能，隐藏实现的细节。
3. **设计角度讲**：

（1）分而治之

-- 将一个大的需求分解为许多类，每个类处理一个独立的功能。

-- 拆分好处：便于分工，便于复用，可扩展性强。

(2) 变则疏之

-- 变化的地方独立封装，避免影响其他类。

(3) 高 内 聚

-- 类中各个方法都在完成一项任务(单一职责的类)。

(4) 低 耦 合

-- 类与类的关联性与依赖度要低(每个类独立)，让一个类的改变，尽少影响其他类。

[例如：硬件高度集成化，又要可插拔]

最高的内聚莫过于类中仅包含1个方法，将会导致高内聚高耦合。

最低的耦合莫过于类中包含所有方法，将会导致低耦合低内聚。

### 作用

1. 简化编程，使用者不必了解具体的实现细节，只需要调用对外提供的功能。
2. 松散耦合，降低了程序各部分之间的依赖性。
3. 数据和操作相关联，方法操作的是自己的数据。

### 私有成员

1. 作用：无需向类外提供的成员，可以通过私有化进行屏蔽。
2. 做法：命名使用双下划线开头。
3. 本质：障眼法，实际也可以访问。

私有成员的名称被修改为：\_类名\_\_成员名，可以通过\_dict\_属性或dir函数查看。

### \_\_slots\_\_

1. 作用：限定一个类创建的实例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：含有\_\_slots\_\_属性的类所创建的对象没有\_\_dict\_\_属性, 即此实例不用字典来存储对象的实例属性。
2. 优点：访止用户因错写属性的名称而发生程序错误。
3. 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。

### 属性@property

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，又使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。

使用:

1.验证实例变量的区间

2.变量私有化 property与\_\_var

1. 定义：

@property

def name(self):

return self.\_\_name

@name.setter

def name(self, name):

self.\_\_name = name

1. 调用：

对象.属性名 = 数据

变量 = 对象.属性名

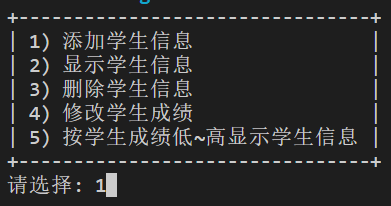
1. 说明：

-- 通常两个公开的属性，保护一个私有的变量。

-- @property 负责读取，@属性名.setter 负责写入

-- 只写：属性名= property(None, 写入方法名)

### 案例:信息管理系统



#### 需求

实现对学生信息的增加、删除、修改和查询。

#### 分析

界面可能使用控制台，也可能使用Web等等。

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，获取输入，显示结果等。

逻辑控制类：负责存储学生信息，处理业务逻辑。比如添加、删除等

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：StudentModel

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 \_\_stu\_list

行为：获取列表 stu\_list,添加学生 add\_student，删除学生remove\_student，修改学生update\_student，根据成绩排序order\_by\_score。

界面视图类：StudentManagerView

数据：逻辑控制对象\_\_manager

行为：显示菜单\_\_display\_menu，选择菜单项\_\_select\_menu\_item，入口逻辑main，

输入学生\_\_input\_students，输出学生\_\_output\_students，删除学生\_\_delete\_student，修改学生信息\_\_modify\_student，按成绩输出学生\_\_output\_student\_by\_score

## 封装小结

*封装*

*数据角度：将多个变量封装到一个自定义类中。*

*优势：*

*符合人类的思考方式*

*可以将数据与对数据的操作封装到一起*

*功能角度：对外提供必要的功能,隐藏实现的细节.*

*DoubleListHelper.get\_elements()*

*-- 私有化：将名称命名为以双下划线开头.*

*内部修改成员名称*

*-- 属性：对实例变量的保护(拦截读/写操作)*

*-- \_\_slots\_\_:限定类创建的对象只能有固定的实例变量.*

*设计角度：*

*分而治之:将大的需求分解为多个类，每个类负责一个职责。*

*变则疏之：遇到变化点单独封装为一个类*

*------------------*

*高内聚：一个类有且只有一个发生变化的原因*

*低耦合：类与类的关系松散*

*View Model Controller*

*界面 数据 业务逻辑*

*变化 载体 变化*

## 继承

### 语法

1. 代码

class 子类(父类):

def \_\_init\_\_(self,参数列表):

super().\_\_init\_\_(参数列表)

self.自身实例变量 = 参数

1. 说明

-- 子类拥有父类的所有成员。

-- 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，以确保父类属性被正常创建。

### 多继承

一个子类继承两个或两个以上的基类，父类中的属性和方法同时被子类继承下来。

同名方法的解析顺序（MRO， Method Resolution Order）:

类自身 --> 父类继承列表（由左至右）--> 再上层父类

A

/ \

/ \

B C

\ /

\ /

D

### 定义

1. 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（共性），还可以扩展新功能。

-- 事物具有一定的层次、渊源，继承可以统一概念。

例如：公司组织架构

老板

行政中心 营销中心 技术中心

人力资源 行政部 销售部 策划部 研发部 产品部

### 优点

1. 一种代码复用的方式。
2. 以层次化的方式管理类。

### 缺点

耦合度高

### 作用

隔离客户端代码与功能的实现方式。

### 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，范围更宽泛；子类相对于父类更具体，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C++，Python）。

Object类：任何类都直接或间接继承自 object 类。

### 内置函数

isinstance(obj, class\_or\_tuple)

返回这个对象obj 是否是某个类的对象,或者某些类中的一个类的对象。

## 多态

### 定义

父类的同一种动作或者行为，在不同的子类上有不同的实现。

### 作用

1. 继承将相关概念的共性进行抽象，多态在共性的基础上，体现类型的个性化（一个行为有不同的实现）。
2. 增强程序扩展性，体现开闭原则。

### 重写

子类实现了父类中相同的方法（方法名、参数），在调用该方法时，实际调用的是子类的方法。

#### 内置可重写函数

Python中，以双下划线开头、双下划线结尾的是系统定义的成员。我们可以在自定义类中进行重写，从而改变其行为。

\_\_str\_\_函数：将对象转换为字符串(对人友好的)

\_\_repr\_\_函数：将对象转换为字符串(解释器可识别的)

#### 运算符重载

定义：让自定义的类生成的对象(实例)能够使用运算符进行操作。

##### 算数运算符



##### 反向算数运算符重载



##### 复合运算符重载



##### 比较运算重载



## 设计原则

### 开-闭原则（目标、总的指导思想）

**O**pen **C**losed **P**rinciple

对扩展开放，对修改关闭。

增加新功能，不改变原有代码。

### 类的单一职责（一个类的定义）

**S**ingle **R**esponsibility **P**rinciple

一个类有且只有一个改变它的原因。

### 依赖倒置（依赖抽象）

**D**ependency **I**nversion **P**rinciple

客户端代码(调用的类)尽量依赖(使用)抽象的组件。

抽象的是稳定的。实现是多变的。

### 组合复用原则（复用的最佳实践）

Composite Reuse Principle

如果仅仅为了代码复用优先选择组合复用，而非继承复用。

组合的耦合性相对继承低。

### 里氏替换（继承后的重写，指导继承的设计）

**L**iskov **S**ubstitution **P**rinciple

父类出现的地方可以被子类替换，在替换后依然保持原功能。

子类要拥有父类的所有功能。

子类在重写父类方法时，尽量选择扩展重写，防止改变了功能。

### 迪米特法则（类与类交互的原则）

Law of Demeter

不要和陌生人说话。

类与类交互时，在满足功能要求的基础上，传递的数据量越少越好。因为这样可能降低耦合度。

## 类与类的关系

**泛化：**子类与父类的关系，概念的复用，耦合度最高；

B类泛化A类，意味B类是A类的一种；

做法：B类继承A类

**关联(聚合/组合)：**部分与整体的关系，功能的复用，变化影响一个类；

A与B关联，意味着B是A的一部分；

做法：在A类中包含B类型成员。

**依赖：**合作关系，一种相对松散的协作，变化影响一个方法；

A类依赖B类，意味A类的某些功能靠B类实现；

做法：B类型作为A类中方法的参数，并不是A的成员。

# 面向对象的小结

## 面向对象基础

*概述：*

*面向过程：关注过程(细节) “干*”

*例如：购物车*

*选择菜单-->购买-->打印商品信息-->创建订单--> ...*

*|->结算-->...*

*面向对象：关心解决问题的人* “*找*”

*类与对象:*

*类：类别*

*对象：个体*

*类与类区别：行为(函数/方法)不同*

*对象与对象区别:数据不同*

*语法：*

*创建类：*

*class 类名:*

*def \_\_init\_\_(self,参数):*

*#数据成员*

*self.数据1　= 参数*

*#行为成员*

*def 方法名称():*

*方法体*

*创建对象：*

*变量名　＝　类名(参数)*

*day10 复习*

*类和对象*

*类:抽象　　　向量* *class Vector2 str int list*

*对象:具体　　1,2 Vector2(1,2) "a" 1 [1,2]*

*之间的区别：类与类行为不同*

*对象与对象数据不同*

*Vector2(1,2)　Vector2(3,4)*

*同一个类型的多个对象,数据不同(1,2/3,4),行为(求方向，求大小)相同.*

*类成员：*

*实例：对象的数据(变量)，对象的行为(方法).*

*类:类的数据(变量),类的行为(方法).*

*可以被所有对象共同操作的数据*

*静态方法:*

*实例方法操作实例变量，表示"个体"行为.*

*类方法操作类变量，表示"大家"行为.*

*静态方法不能操作数据，表示为函数都可以.*

*"""*

*# ------------实例---------------*

**class** MyClass:

**def** \_\_init\_\_(self, a):

*# 实例变量*

self.a = a

*# 实例方法*

**def** print\_self(self):

*# 可以操作实例变量*

print(self.a)

*# 通过对象访问*

m01 = MyClass(100)

m01.b = 1

m02 = MyClass(100)

print(m02.b)

print(m02.a)

*# ------------类---------------*

**class** MyClass02:

*# 类变量*

a = 0

*# 类方法*

@classmethod *# 自动传入当前方法的参数是类，而不是对象.*

**def** print\_self(cls):

*# 可以操作类变量*

print(cls.a)

*# 通过类名访问*

print(MyClass02.a)

MyClass02.print\_self()

*# ------------不常用的访问方式---------------*

*# 访问实例方法，还可以通过类。*

MyClass.print\_self(m01) *# 也必须传递对象*

*# 访问类成员，还可以通过对象*

m03 = MyClass02()

print(m03.a)

print(m03.print\_self())

## 面向对象的设计流程

*面向对象：考虑问题从对象的角度出发.*

*抽象：从多个事物中，舍弃个别的/非本质的特征(不重要)，*

*抽出共性的本质(重要的)过程。*

*三大特征：*

*封装：将每个变化点单独分解到不同的类中。*

*例如：老张开车去东北*

*做法：定义人类，定义车类。*

*继承：重用现有类的功能和概念，并在此基础上进行扩展。*

*统一概念*

*例如：图形管理器，统计圆形/矩形.....面积。*

*做法：用图形类代表/约束，圆形/矩形..具有计算面积的方法.*

*多态：调用父"抽象的"方法，执行子类"具体的"方法.*

*重写：覆盖父类那个比较抽象的方法。*

*例如：图形管理器调用图形的计算面积方法*

*具体图形必须重写图形的计算面积方法。*

*继承是共性(计算面积)，多态个性(长\*宽* */ pi \*r\*\*2)。*

*设计原则*

*开闭原则：允许增加新功能，不允许修改客户端代码.*

*单一职责：一个有且只有一个改变的原因.*

*依赖倒置：调用抽象(父)，不要调用具体(子);*

*抽象不要依赖具体.*

*组合复用：如果仅仅是代码的复用，优先使用组合.*

*类与类关系*

*泛化[继承](做成爸爸)*

*关联(做成成员变量)*

*依赖(做成方法参数)*

# 模块 Module

## 定义

包含一系列数据、函数、类的文件，通常以.py结尾。

## 作用

让一些相关的数据，函数，类有逻辑的组织在一起，使逻辑结构更加清晰。

有利于多人合作开发。

## 导入

### import

1. 语法：

import 模块名

import 模块名 as 别名

1. 作用：将某模块整体导入到当前模块中
2. 使用：模块名.成员

### from import

1. 语法：

from 模块名 import 成员名[ as 别名1]

作用：将模块内的一个或多个成员导入到当前模块的作用域中。

### from import \*

1. 语法：from 模块名 import \*
2. 作用：将某模块的所有成员导入到当前模块。
3. 模块中以下划线(\_)开头的属性，不会被导入，通常称这些成员为隐藏成员。

**模块变量**

\_\_all\_\_变量：定义可导出成员，仅对from xx import \*语句有效。

\_\_doc\_\_变量：文档字符串。

\_\_file\_\_变量：模块对应的文件路径名。

\_\_name\_\_变量：模块自身名字，可以判断是否为主模块。

当此模块作为主模块(第一个运行的模块)运行时，\_\_name\_\_绑定'\_\_main\_\_'，不是主模块，而是被其它模块导入时,存储模块名。

## 加载过程

在模块导入时，模块的所有语句会执行。

如果一个模块已经导入，则再次导入时不会重新执行模块内的语句。

## 分类

1. 内置模块(builtins)，在解析器的内部可以直接使用。
2. 标准库模块，安装Python时已安装且可直接使用。
3. 第三方模块（通常为开源），需要自己安装。
4. 用户自己编写的模块（可以作为其他人的第三方模块）

## 搜索顺序

搜索内建模块(builtins)

sys.path 提供的路径，通常第一个是程序运行时的路径。

# 小结

*模块*

*1. 定义：.py文件*

*2. 作用：多人合作开发*

*3. 导入：*

*import 模块*

*模块.成员名*

*import 模块* *as 别名*

*别名.成员名*

*-------------------*

*from 模块* *import 成员1*

*直接使用成员*

*-------------------*

*from 模块* *import \**

# 包package

## 定义

将模块以文件夹的形式进行分组管理。

## 作用

让一些相关的模块组织在一起，使逻辑结构更加清晰。

## 导入

import 包名 [as 包别名] 需要设置\_\_all\_\_

import 包名.模块名 [as 模块新名]

import 包名.子包名.模块名 [as 模块新名]

from 包名 import 模块名 [as 模块新名]

from 包名.子包名 import 模块名 [as 模块新名]

from 包名.子包名.模块名 import 成员名 [as 属性新名]

# 导入包内的所有子包和模块

from 包名 import \*

from 包名.模块名 import \*

## 搜索顺序

sys.path 提供的路径

## \_\_init\_\_.py 文件

是包内必须存在的文件

会在包加载时被自动调用

### \_\_all\_\_

记录from 包 import \* 语句需要导入的模块

案例：

my\_ project /

\_\_init\_\_.py

main.py

common/

\_\_init\_\_.py

double\_list\_helper.py

list\_helper.py

skill\_system/

\_\_init\_\_.py

skill\_deployer.py

skill\_manager.py

# 小结

*包*

*定义：包含\_\_init\_\_.py文件的文件夹*

*作用:团队分工,结构清晰.*

*导入: import 包.模块* *as 变量*

*from 包.模块* *import 成员*

*from 包.模块* *import ×*

*原理：import sys*

*sys.path + from 的路径可以正确的定位到文件，导包才成功.*

# 异常处理Error

## 异常

1. 定义：运行时检测到的错误。
2. 现象：当异常发生时，程序不会再向下执行，而转到函数的调用语句。
3. 常见异常类型：

-- 名称异常(NameError)：变量未定义。

-- 类型异常(TypeError)：不同类型数据进行运算。

-- 索引异常(IndexError)：超出索引范围。

-- 属性异常(AttributeError)：对象没有对应名称的属性。

-- 键异常(KeyError)：没有对应名称的键。

-- 为实现异常(NotImplementedError)：尚未实现的方法。

-- 异常基类Exception。

## 处理

1. 语法：

try:

可能触发异常的语句

except 错误类型1 [as 变量1]：

处理语句1

except 错误类型2 [as 变量2]：

处理语句2

except Exception [as 变量3]：

不是以上错误类型的处理语句

else:

未发生异常的语句

finally:

无论是否发生异常的语句

1. 作用：将程序由异常状态转为正常流程。
2. 说明：

as 子句是用于绑定错误对象的变量，可以省略

except子句可以有一个或多个，用来捕获某种类型的错误。

else子句最多只能有一个。

finally子句最多只能有一个，如果没有except子句，必须存在。

如果异常没有被捕获到，会向上层(调用处)继续传递，直到程序终止运行。

## raise 语句

1. 作用：抛出一个错误，让程序进入异常状态。
2. 目的：在程序调用层数较深时，向主调函数传递错误信息要层层return 比较麻烦，所以人为抛出异常，可以直接传递错误信息。。

## 自定义异常

1. 定义：

class 类名Error(Exception):

def \_\_init\_\_(self,参数):

super().\_\_init\_\_(参数)

self.数据 = 参数

1. 调用：

try:

….

raise 自定义异常类名(参数)

….

except 定义异常类 as 变量名:

变量名.数据

1. 作用：封装错误信息

# 小结

*异常处理*

*异常：运行时遇到的错误,后续代码不再执行,返回给调用者.*

*处理：将异常流程(向上翻)转为正常流程(向后走).*

*语法:*

*try:*

*可能出错的语句*

*except 错误类型1:*

*处理逻辑*

*except 错误类型2:*

*....*

*处理逻辑*

*else:*

*不错的逻辑*

*finally:*

*无论对与错，一定执行的代码*

*raise: 人为抛出异常*

# 迭代

每一次对过程的重复称为一次“迭代”，而每一次迭代得到的结果会作为下一次迭代的初始值。例如：循环获取容器中的元素。

## 可迭代对象iterable

1. 定义：具有\_\_iter\_\_函数的对象，可以返回迭代器对象。
2. 语法

-- 创建：

class 可迭代对象名称:

  def \_\_iter\_\_(self):

      return 迭代器

-- 使用：

for 变量名 in 可迭代对象:

语句

1. 原理：

迭代器 = 可迭代对象.\_\_iter\_\_()

while True:

try:

print(迭代器.\_\_next\_\_())

except StopIteration:

break

## 迭代器对象iterator

1. 定义：可以被next()函数调用并返回下一个值的对象。
2. 语法

class 迭代器类名:

def \_\_init\_\_(self, 聚合对象):

self.聚合对象= 聚合对象

def \_\_next\_\_(self):

if 没有元素:

raise StopIteration

return 聚合对象元素

1. 说明：

-- 聚合对象通常是容器对象。

4. 作用：使用者只需通过一种方式，便可简洁明了的获取聚合对象中各个元素，而又无需了解其内部结构。

# 生成器generator

1. 定义：能够动态(循环一次计算一次返回一次)提供数据的可迭代对象。
2. 作用：在循环过程中，按照某种算法推算数据，不必创建容器存储完整的结果，从而节省内存空间。数据量越大，优势越明显。
3. 以上作用也称之为延迟操作或惰性操作，通俗的讲就是在需要的时候才计算结果，而不是一次构建出所有结果。

## 生成器函数

1. 定义：含有yield语句的函数，返回值为生成器对象。
2. 语法

-- 创建：

def 函数名():

…

yield 数据

…

-- 调用：

for 变量名 in 函数名():

语句

1. 说明：

-- 调用生成器函数将返回一个生成器对象，不执行函数体。

-- yield翻译为”产生”或”生成”

1. 执行过程：
2. 调用生成器函数会自动创建迭代器对象。
3. 调用迭代器对象的\_\_next\_\_()方法时才执行生成器函数。
4. 每次执行到yield语句时返回数据，暂时离开。
5. 待下次调用\_\_next\_\_()方法时继续从离开处继续执行。
6. 原理：生成迭代器对象的大致规则如下

-- 将yield关键字以前的代码放在next方法中。

-- 将yield关键字后面的数据作为next方法的返回值。

## 内置生成器

### 枚举函数enumerate

1. 语法：

for 变量 in enumerate(可迭代对象):

语句

for 索引, 元素in enumerate(可迭代对象):

语句

1. 作用：遍历可迭代对象时，可以将索引与元素组合为一个元组。

### zip

1. 语法：

for item in zip(可迭代对象1, 可迭代对象2….):

语句

1. 作用：将多个可迭代对象中对应的元素组合成一个个元组，生成的元组个数由最小的可迭代对象决定。

## 生成器表达式

1. 定义：用推导式形式创建生成器对象。
2. 语法：变量 = ( 表达式 for 变量 in 可迭代对象 [if 真值表达式] )

# 迭代与生成器小结

*迭代*

*可迭代对象*

*迭代器*

*生成器*

*class 可迭代对象:*

*def \_\_iter\_\_():*

*创建迭代器对象*

*class 迭代器:*

*def \_\_next\_\_():*

*返回一个元素*

*如果没有元素,则抛出一个StopIteration异常.*

*for 变量* *in 可迭代对象:*

*变量得到的就是\_\_next\_\_方法返回值*

*原理:*

*iterator = 可迭代对象.\_\_iter\_\_()*

*while True:*

*try:*

*变量* *= iterator.\_\_next\_\_()*

*except:*

*break*

*启发:调用next执行一次,计算一次,返回一次.*

*生成器函数:*

*def 函数名():*

*...*

*yield 数据*

*...*

*# 调用方法不执行*

*生成器* *= 函数名()*

*# for 生成器 才执行函数体*

*for item in 生成器:*

*...*

*优势:延迟/惰性操作*

*生成器源码*

*class 生成器:*

*def \_\_iter\_\_():*

*return self*

*def \_\_next\_\_():*

*定义着yield以前的代码*

*返回yield后面的数据*

# 函数式编程

1. 定义：用一系列函数解决问题。

-- 函数可以赋值给变量，赋值后变量绑定函数。

-- 允许将函数作为参数传入另一个函数。

-- 允许函数返回一个函数。

2. 高阶函数：将函数作为参数或返回值的函数。

## 函数作为参数

将核心逻辑传入方法体，使该方法的适用性更广，体现了面向对象的开闭原则。

### lambda 表达式

1. 定义：是一种匿名方法。
2. 作用：作为参数传递时语法简洁，优雅，代码可读性强。

随时创建和销毁，减少程序耦合度。

1. 语法

-- 定义：

变量 = lambda 形参: 方法体

-- 调用：

变量(实参)

1. 说明：

-- 形参没有可以不填

-- 方法体只能有一条语句，且不支持赋值语句。

### 内置高阶函数

1. map（函数，可迭代对象）：使用可迭代对象中的每个元素调用函数，将返回值作为新可迭代对象元素；返回值为新可迭代对象。
2. filter(函数，可迭代对象)：根据条件筛选可迭代对象中的元素，返回值为新可迭代对象。
3. sorted(可迭代对象，key = 函数,reverse = bool值)：排序，返回值为排序结果。
4. max(可迭代对象，key = 函数)：根据函数获取可迭代对象的最大值。
5. min(可迭代对象，key = 函数)：根据函数获取可迭代对象的最小值。

## 函数作为返回值

逻辑连续，当内部函数被调用时，不脱离当前的逻辑。

### 闭包

1. 三要素：

-- 必须有一个内嵌函数。

-- 内嵌函数必须引用外部函数中变量。

-- 外部函数返回值必须是内嵌函数。

1. 语法

-- 定义：

def 外部函数名(参数):

外部变量

def 内部函数名(参数):

使用外部变量

return 内部函数名

-- 调用：

变量 = 外部函数名(参数)

变量(参数)

1. 定义：在一个函数内部的函数,同时内部函数又引用了外部函数的变量。
2. 本质：闭包是将内部函数和外部函数的执行环境绑定在一起的对象。
3. 优点：内部函数可以使用外部变量。
4. 缺点：外部变量一直存在于内存中，不会在调用结束后释放，占用内存。
5. 作用：实现python装饰器。

### 函数装饰器decorators

1. 定义：在不改变原函数的调用以及内部代码情况下，为其添加新功能的函数。
2. 语法

def 函数装饰器名称(func):

def 内嵌函数(\*args, \*\*kwargs):

需要添加的新功能

return func(\*args, \*\*kwargs)

return wrapper

@ 函数装饰器名称

def 原函数名称(参数):

函数体

原函数(参数)

1. 本质：使用“@函数装饰器名称”修饰原函数，等同于创建与原函数名称相同的变量，关联内嵌函数；故调用原函数时执行内嵌函数。

原函数名称 = 函数装饰器名称（原函数名称）

1. 装饰器链：

一个函数可以被多个装饰器修饰，执行顺序为从近到远。

# 函数式编程小结

*"""*

*复习*

*面向对象编程：考虑问题从对象的角度出发*

*函数式编程：考虑问题从函数的角度出发*

*"封装"：封装变化点*

*"继承":抽象变化／隔离变化*

*"多态":调用抽象的函数变量，执行具体的个性函数.*

*lambda:匿名方法,作为实参.*

*"""*

**"""　思想**

**def 功能1():**

**共性代码**

**个性1代码**

**def 功能2():**

**共性代码**

**个性2代码**

**def 功能3():**

**共性代码**

**个性3代码**

**"封装".....**

**def 个性1():**

**个性1代码**

**def 个性2():**

**个性2代码**

**def 个性3():**

**个性3代码**

**继承．．．．．．**

**# 函数类型变量就是具体共性函数的抽象**

**def 共性(函数类型变量):**

**共性代码**

**# 多态．．．．．．**

**函数类型变量() --> 执行具体个性代码**

**执行．．．．．．**

**共性(个性1)**

**共性(个性2)**

**"""**

**""" 项目中的使用**

**将共性代码提取到单独的模块中.**

**在某个代码中导入模块，定义个性代码，调用静态方法(共性代码)．**

**from common.list\_helper import \***

**def 个性代码():**

**...**

**结果** **= ListHelper.静态方法(要操作的数据,个性代码)**

**"""**

**"""lambda**

**def 方法名(参数):**

**函数体**

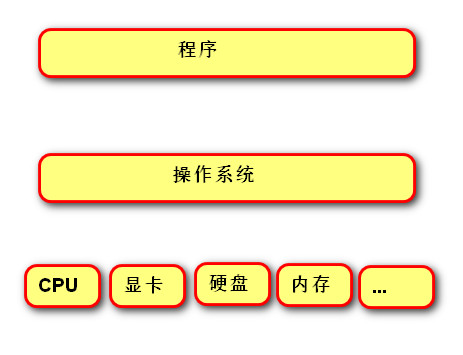
**lambda 参数: 函数体**

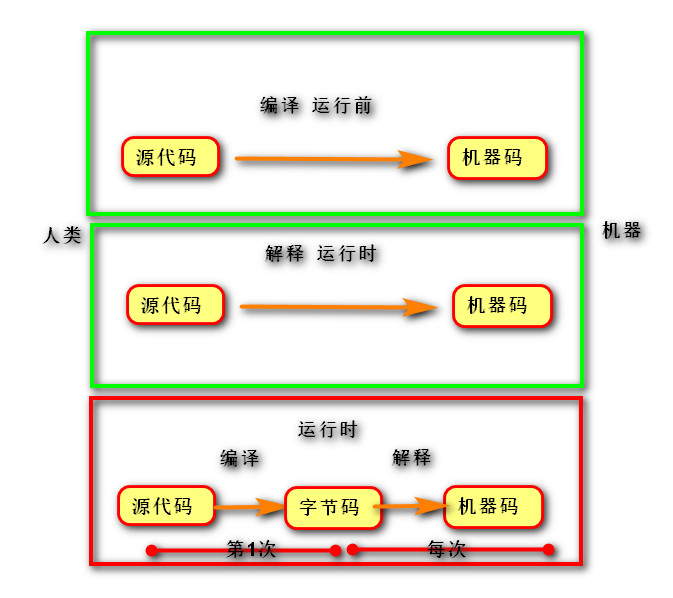
**结果** **= ListHelper.静态方法(要操作的数据,lambda)**

**"""**

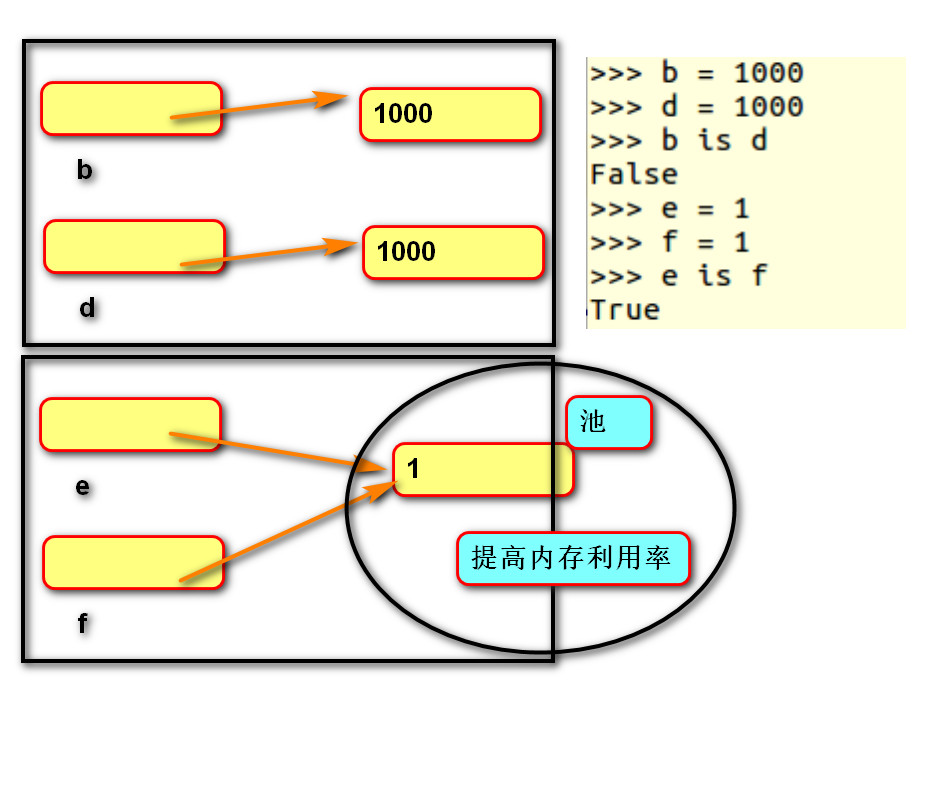
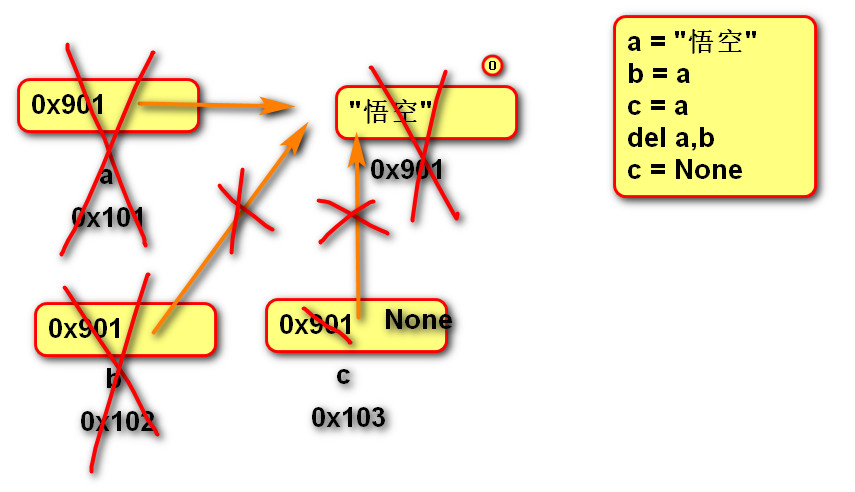
# 附录 : 示意图

## 计算机执行过程

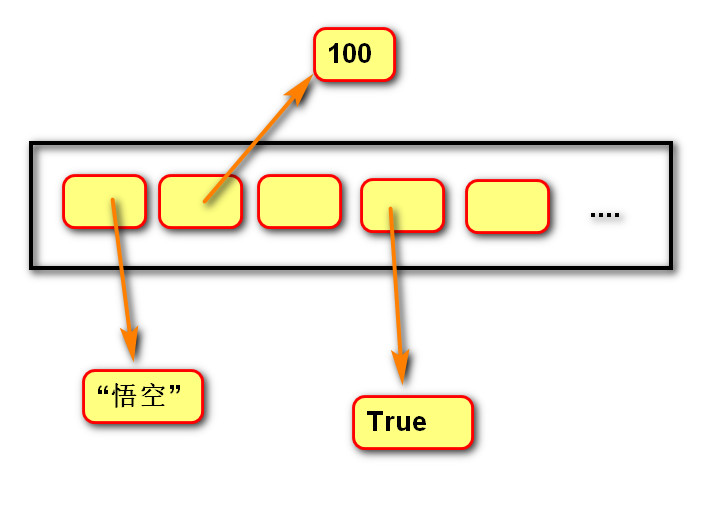
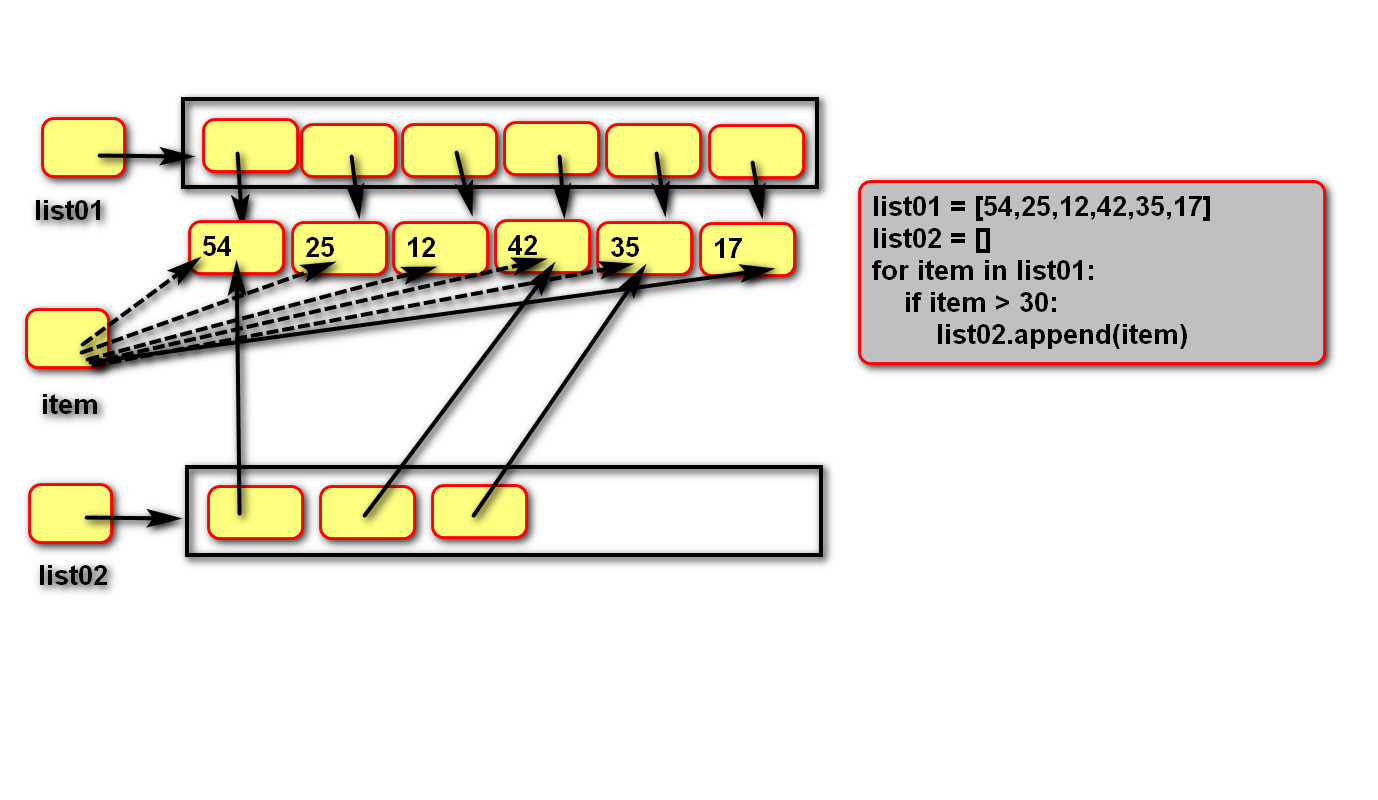


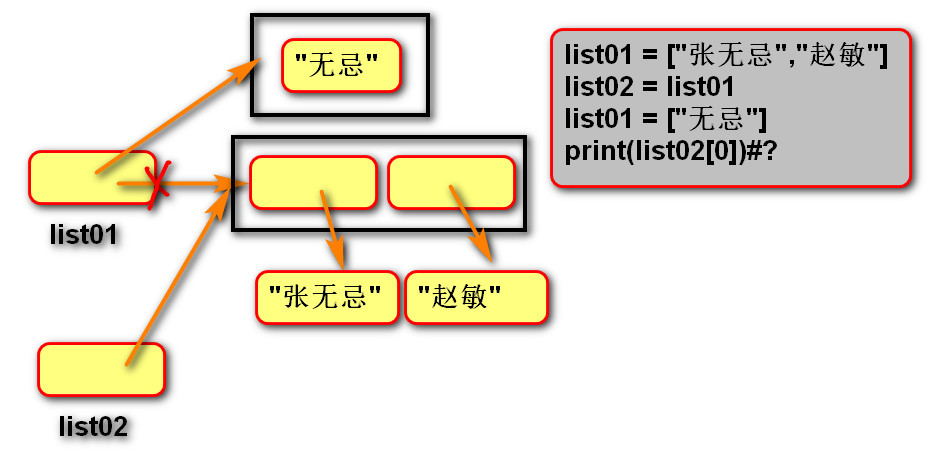
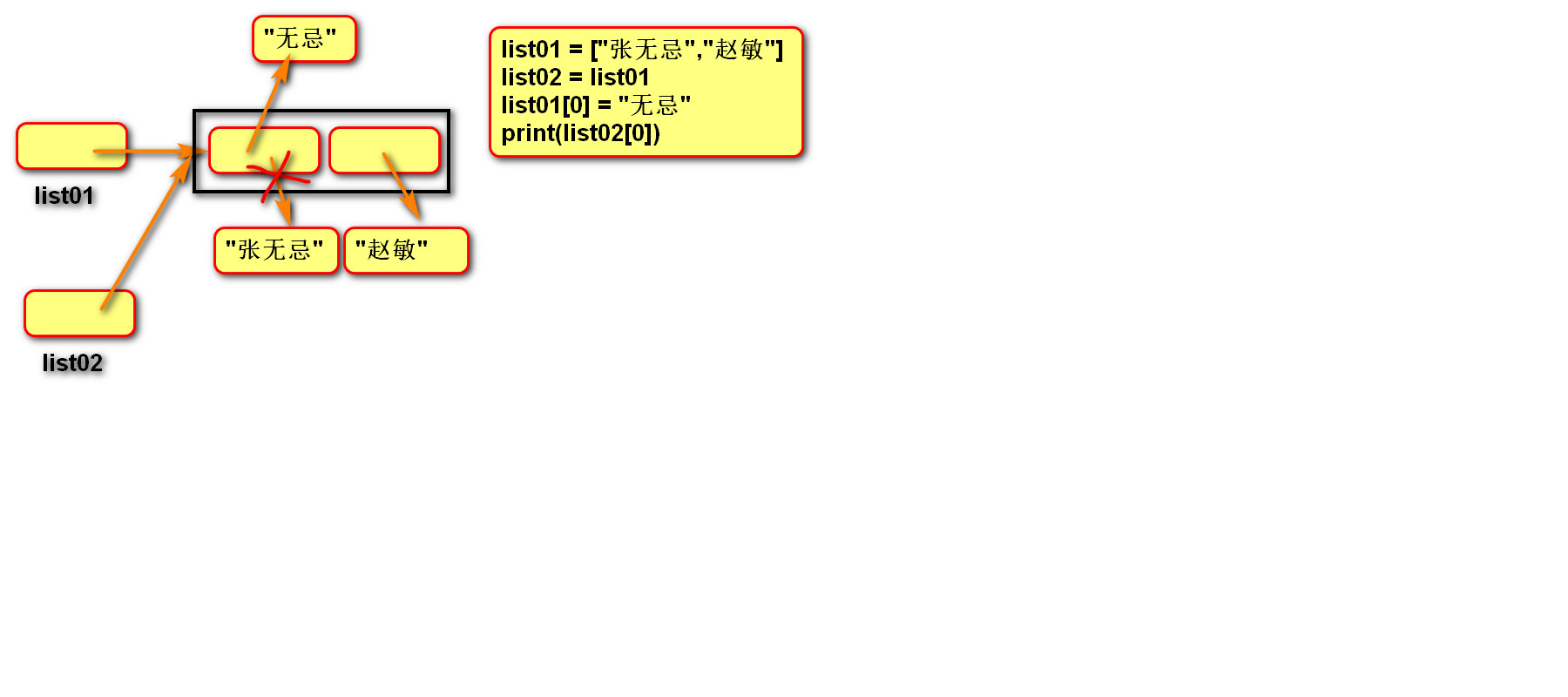


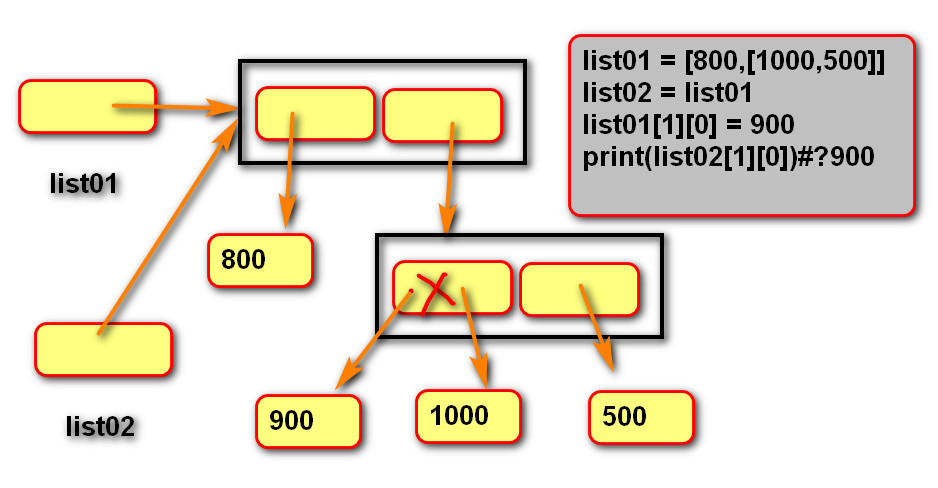
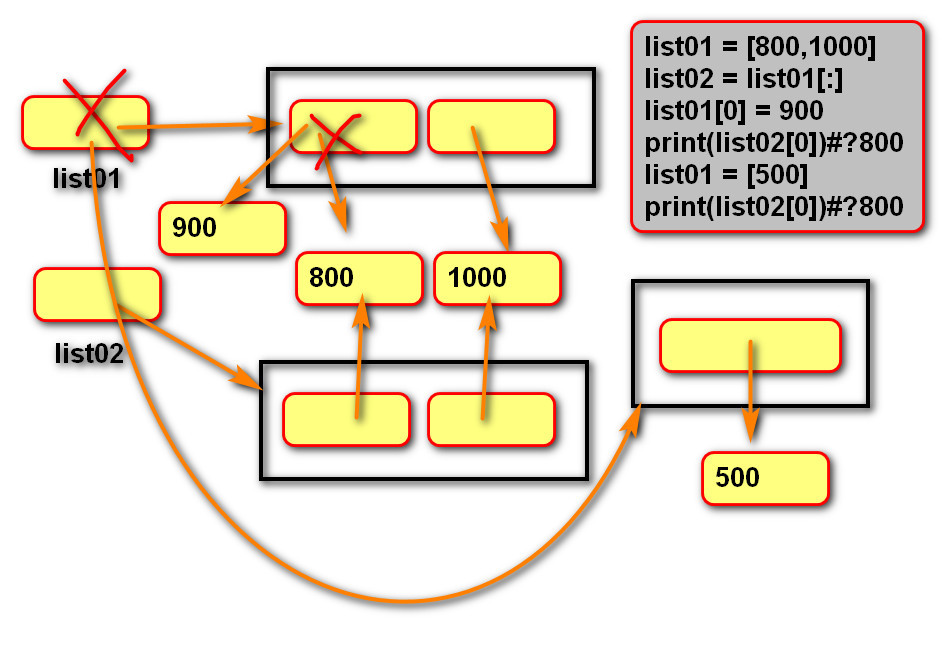
## 变量

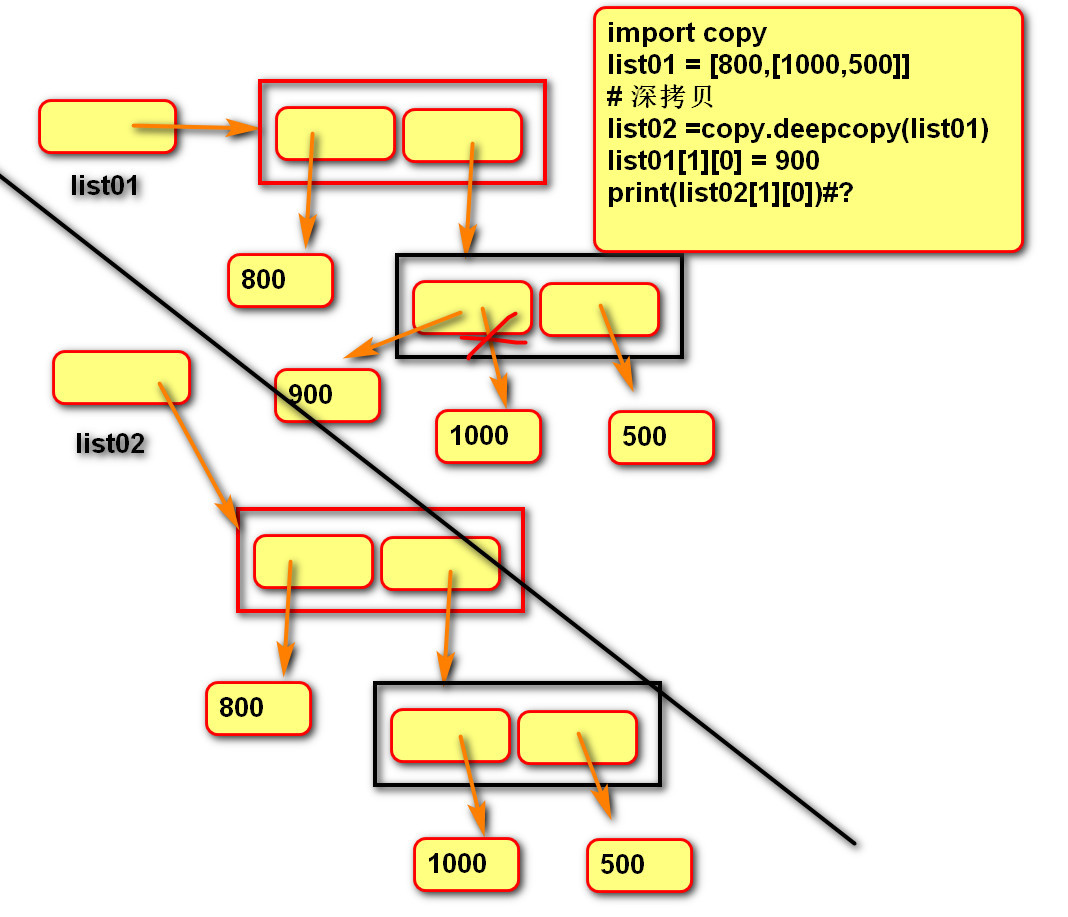
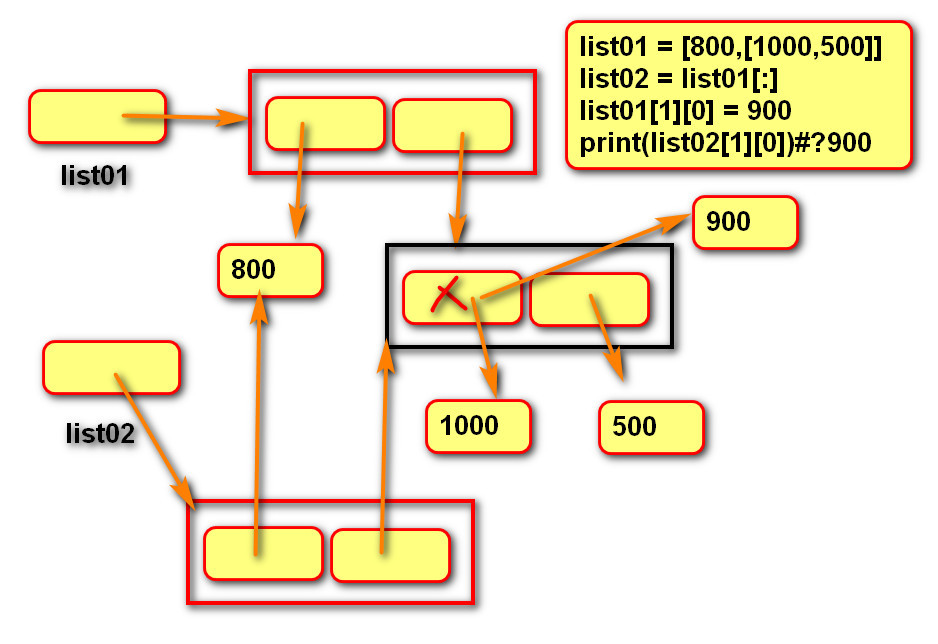


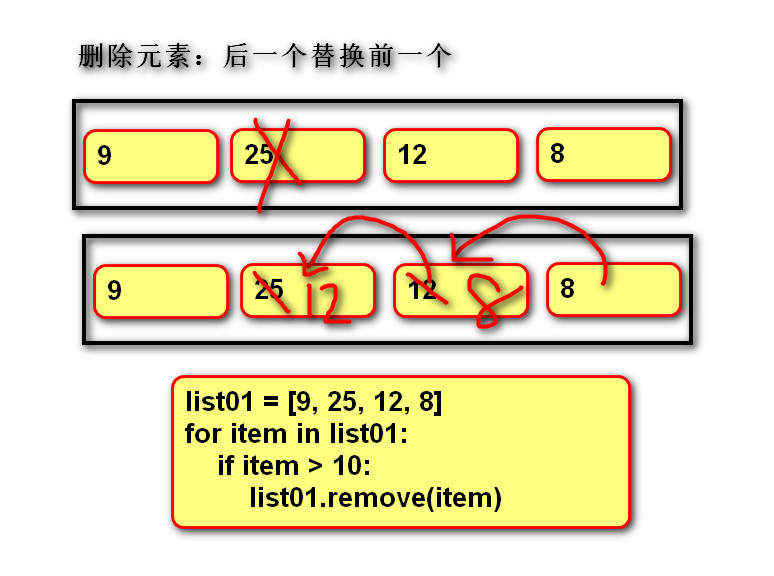
## 列表





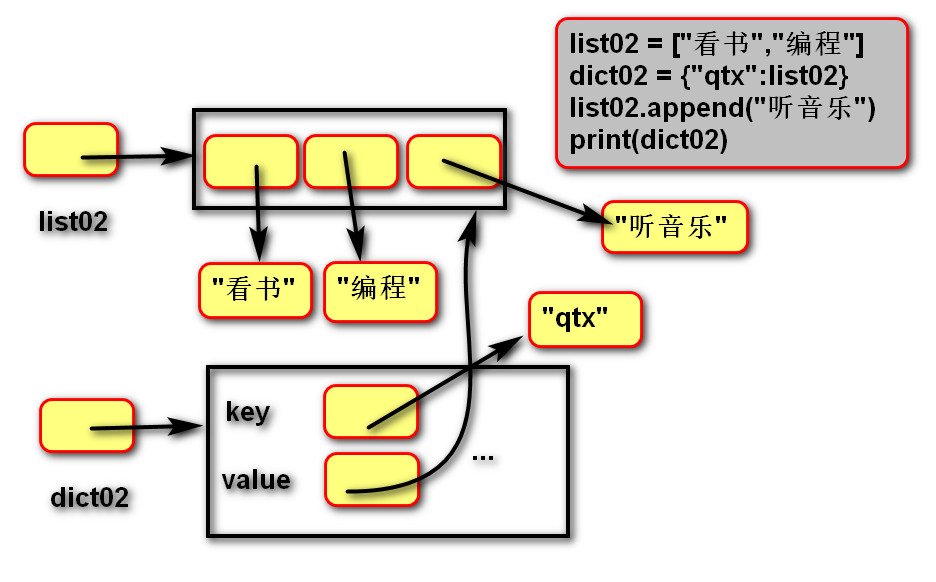
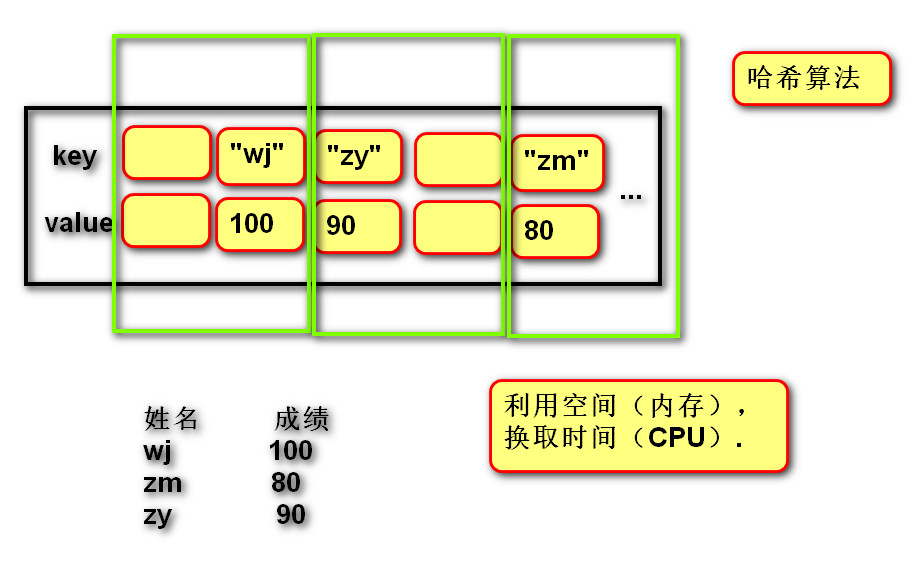




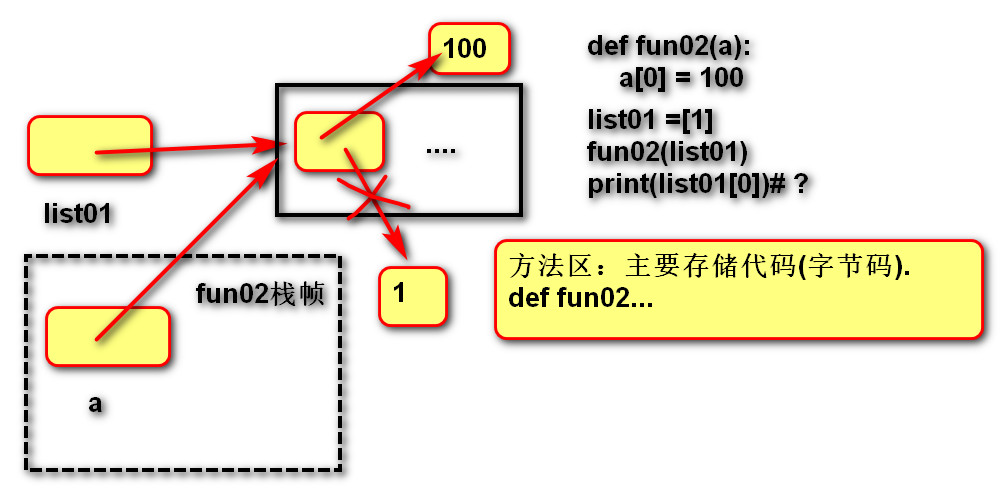


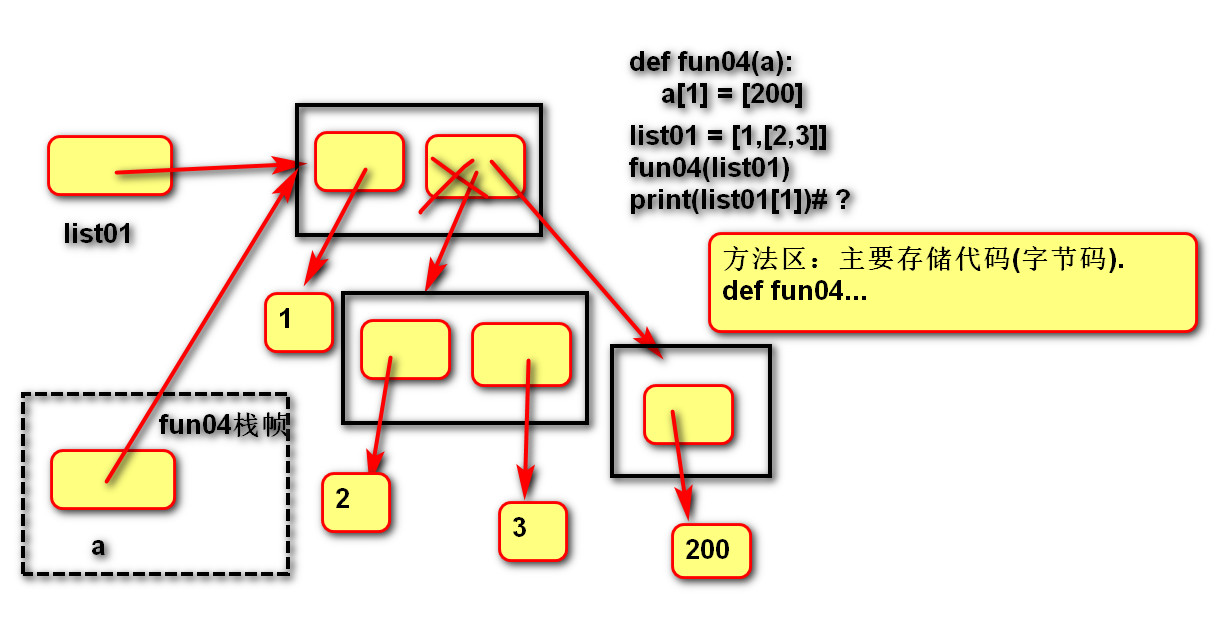
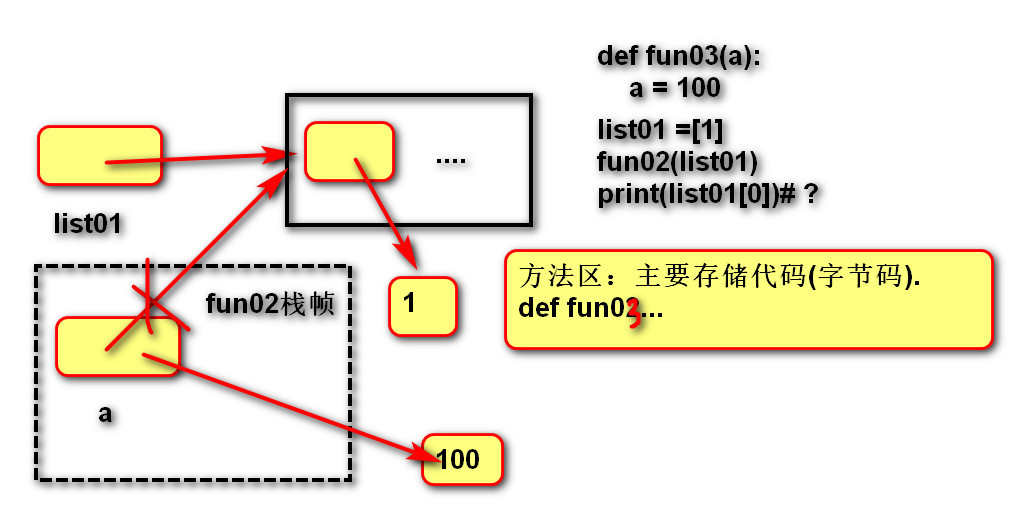
## 元组

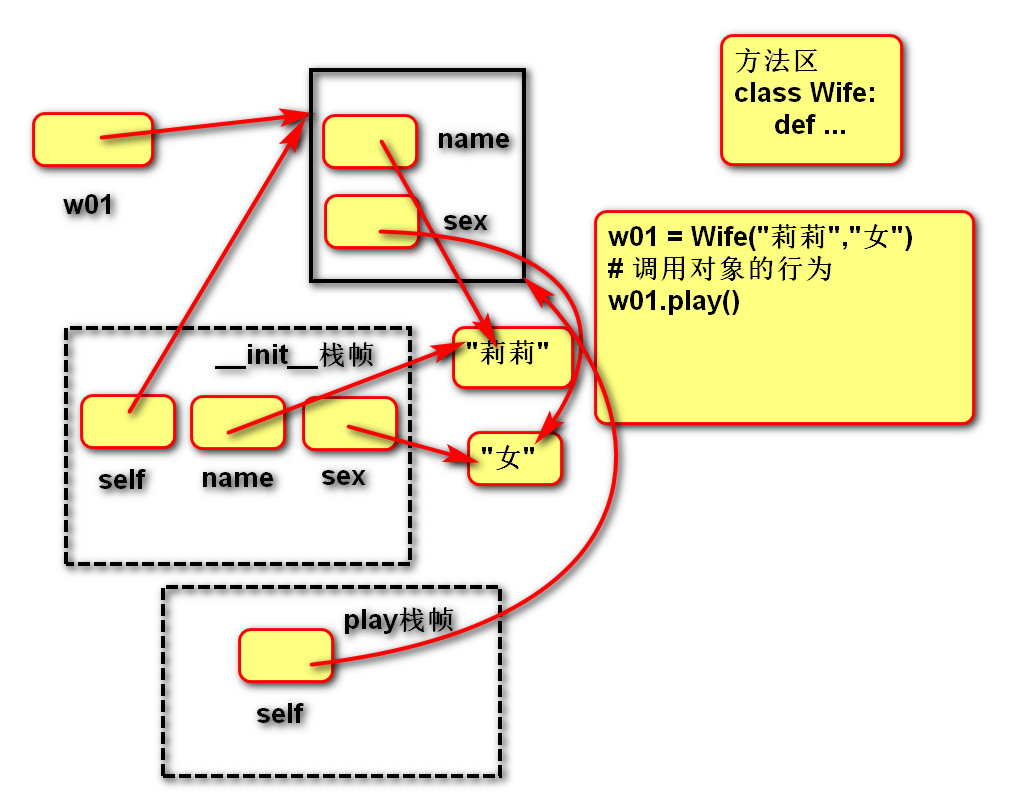
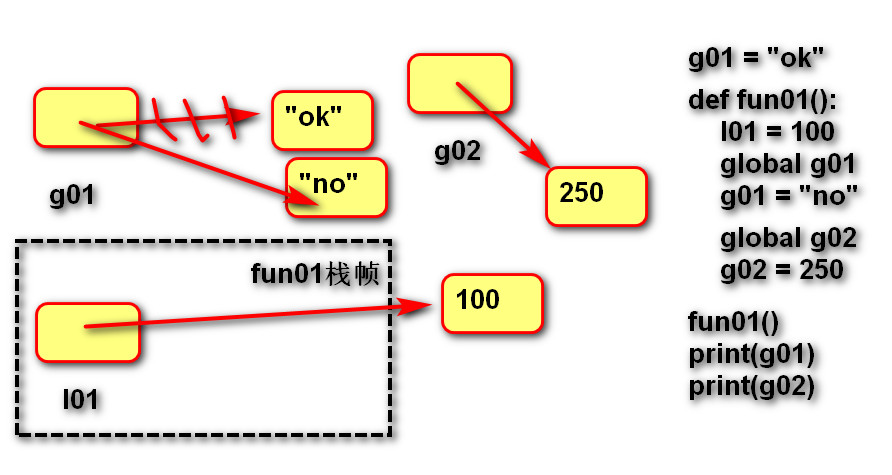
## 字典



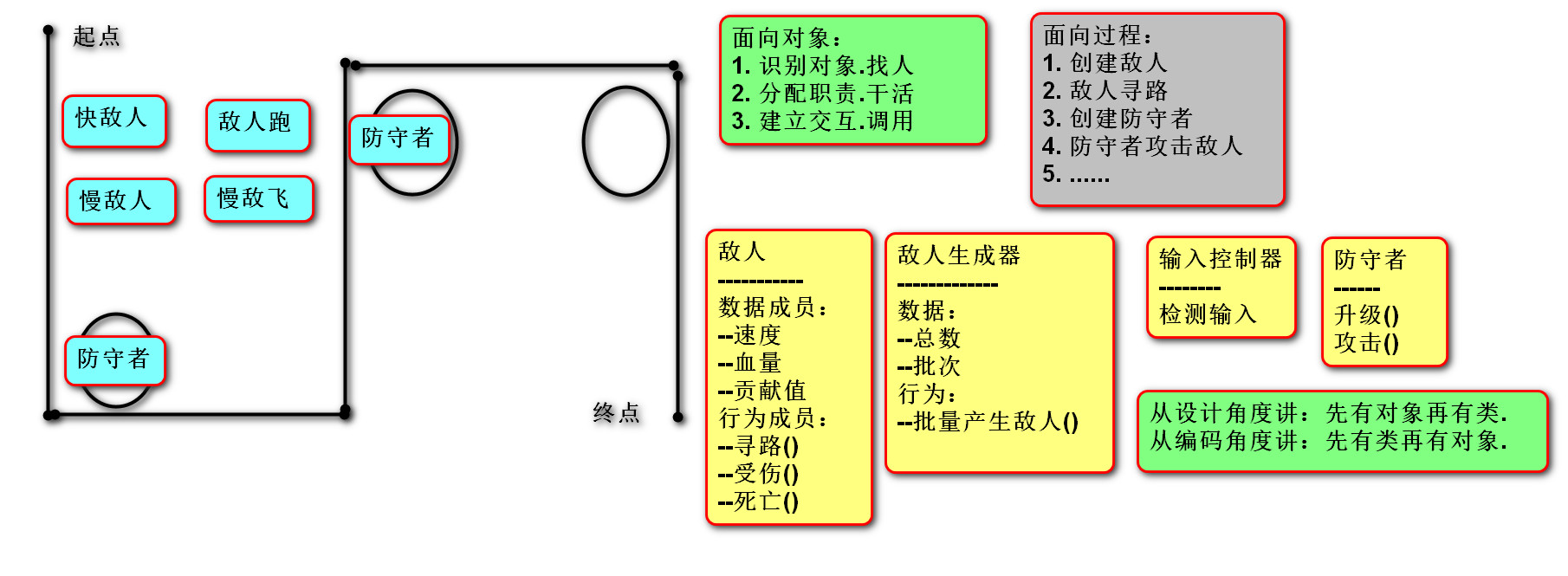
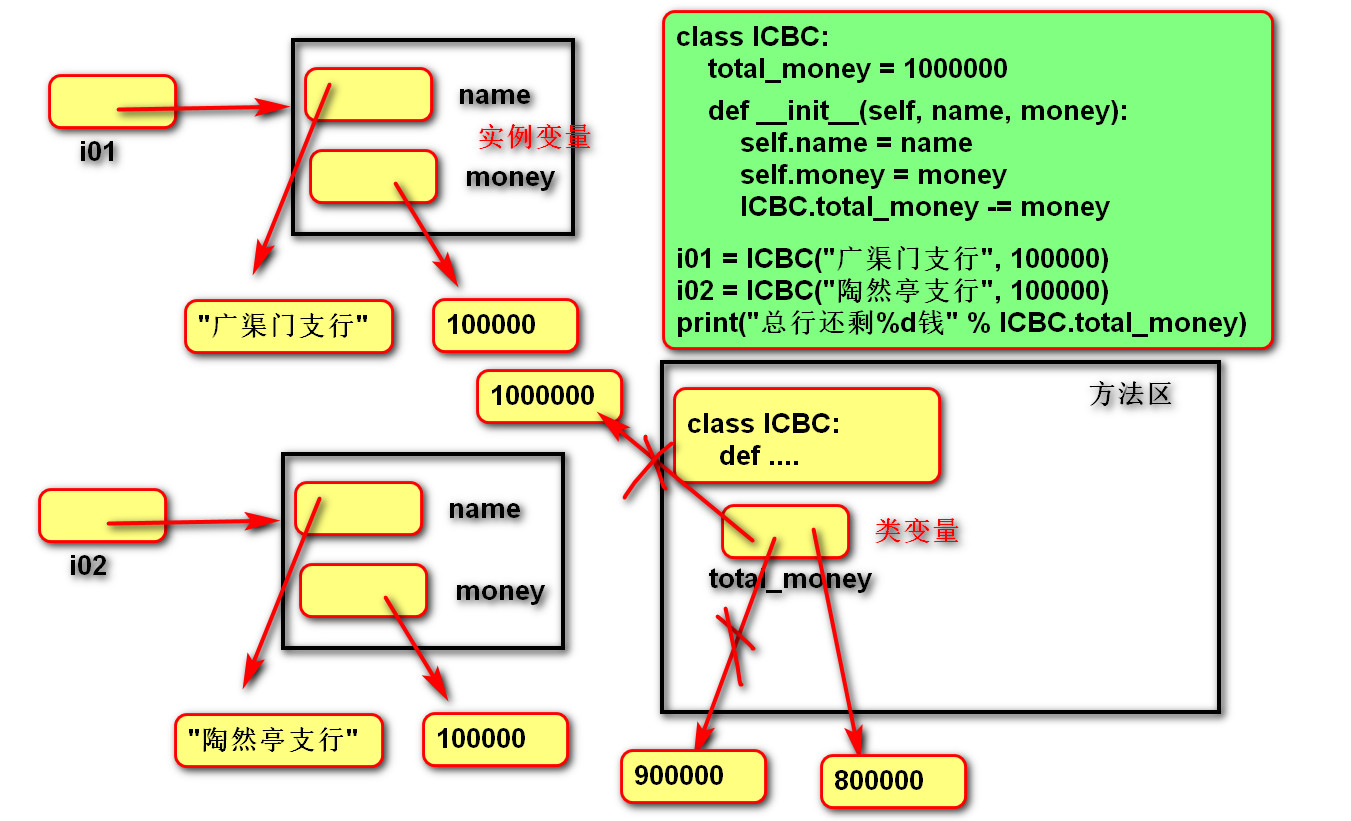
## 函数

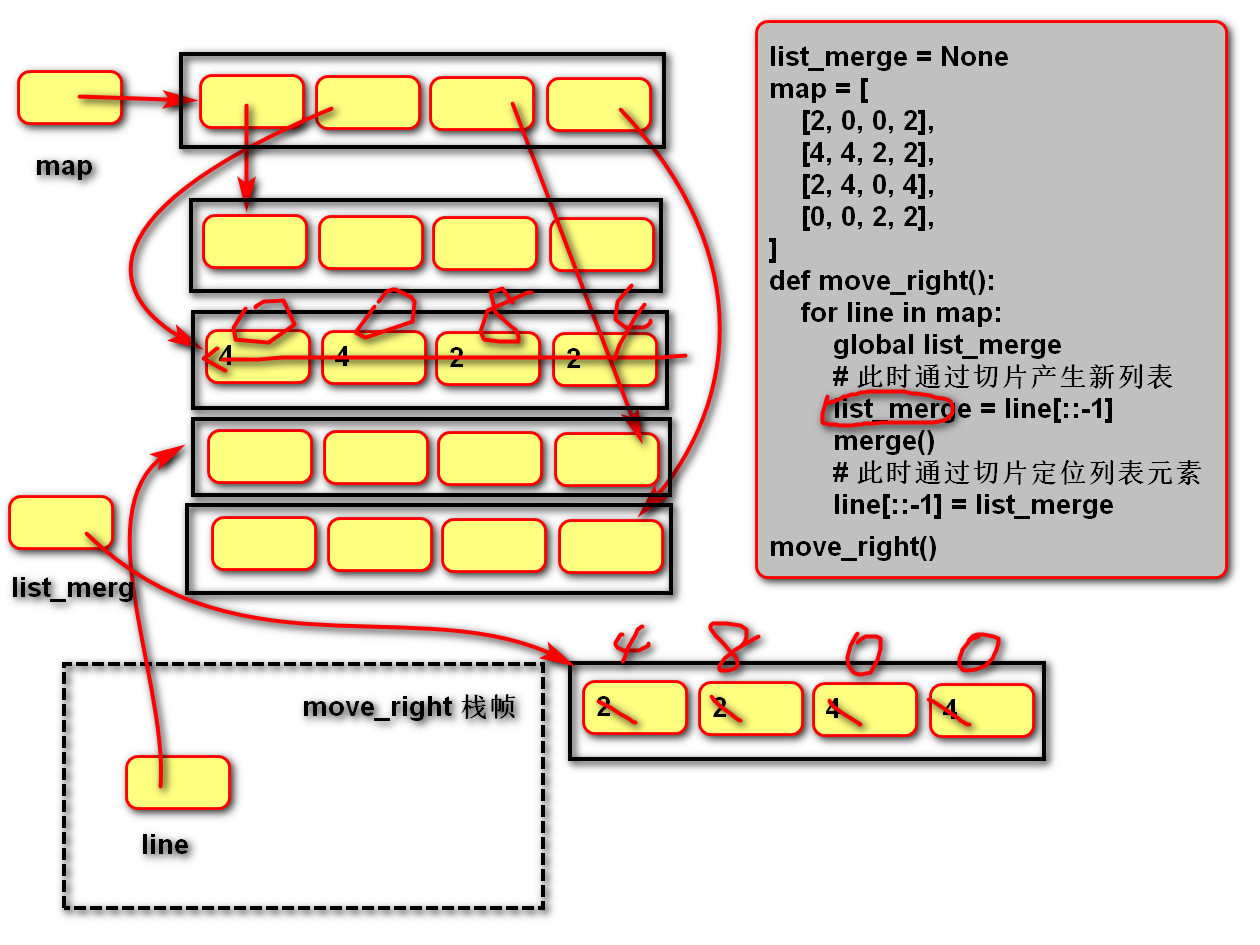
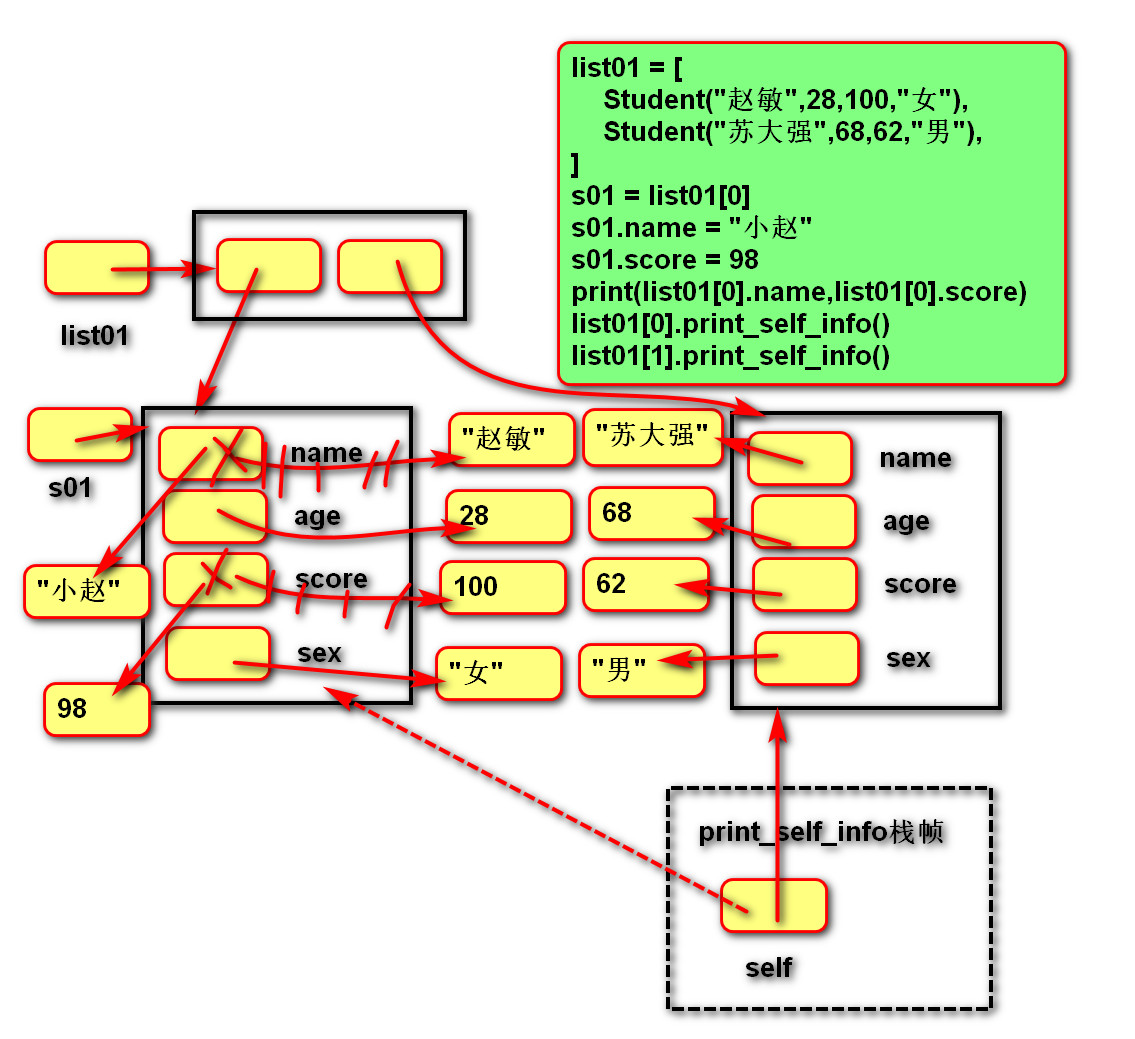


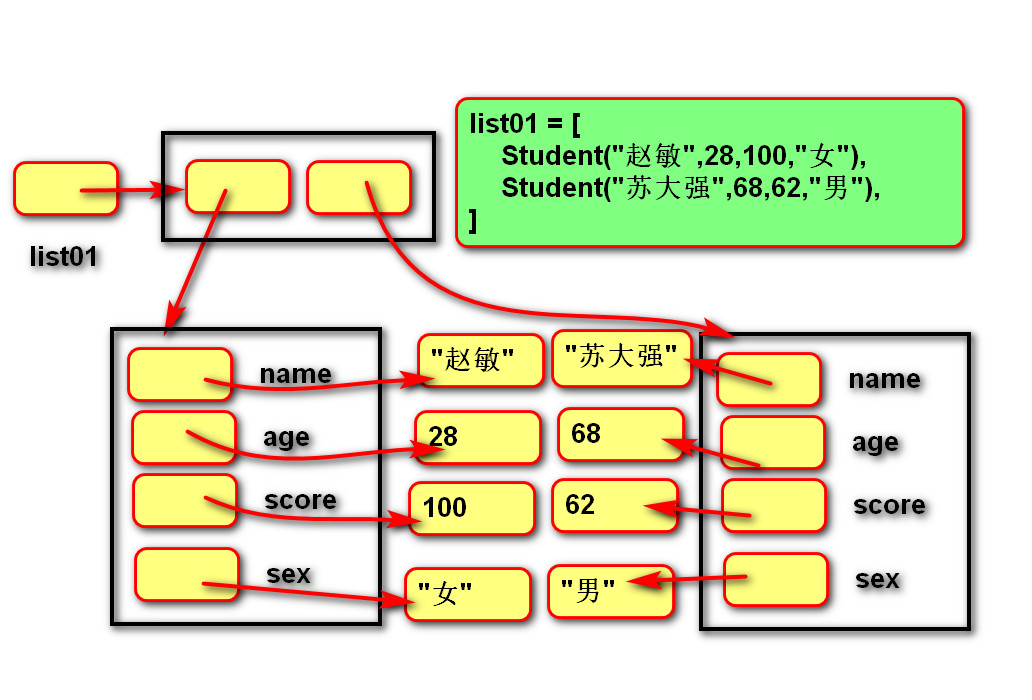




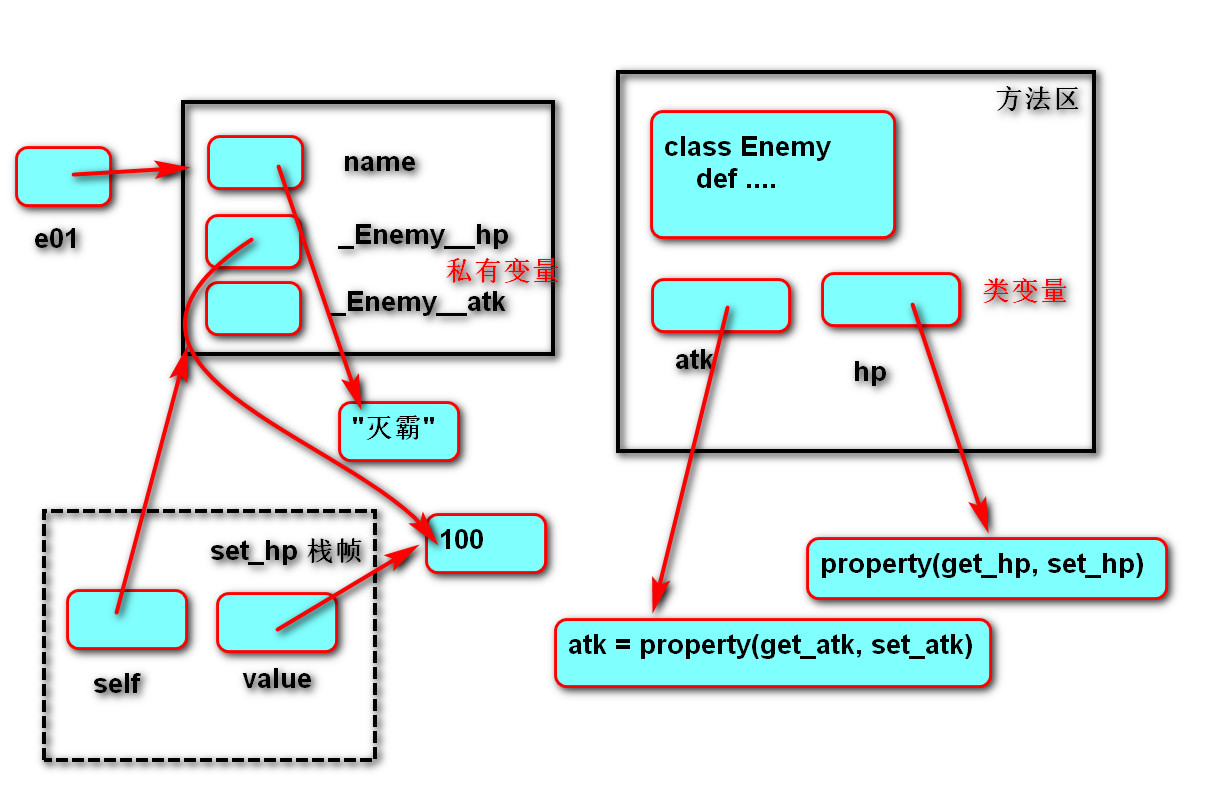
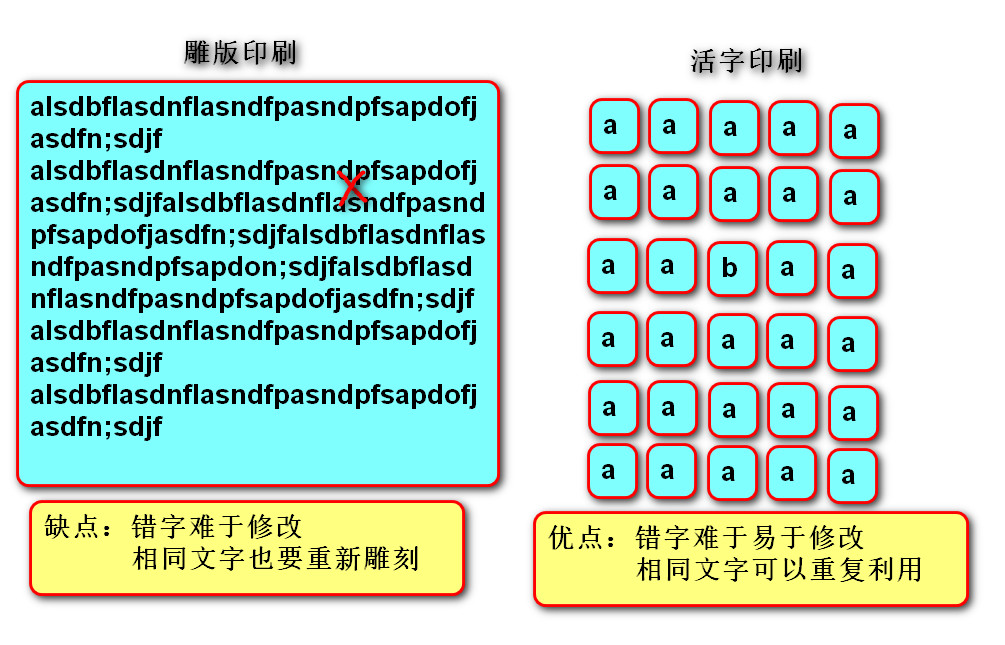
## 面向对象基础



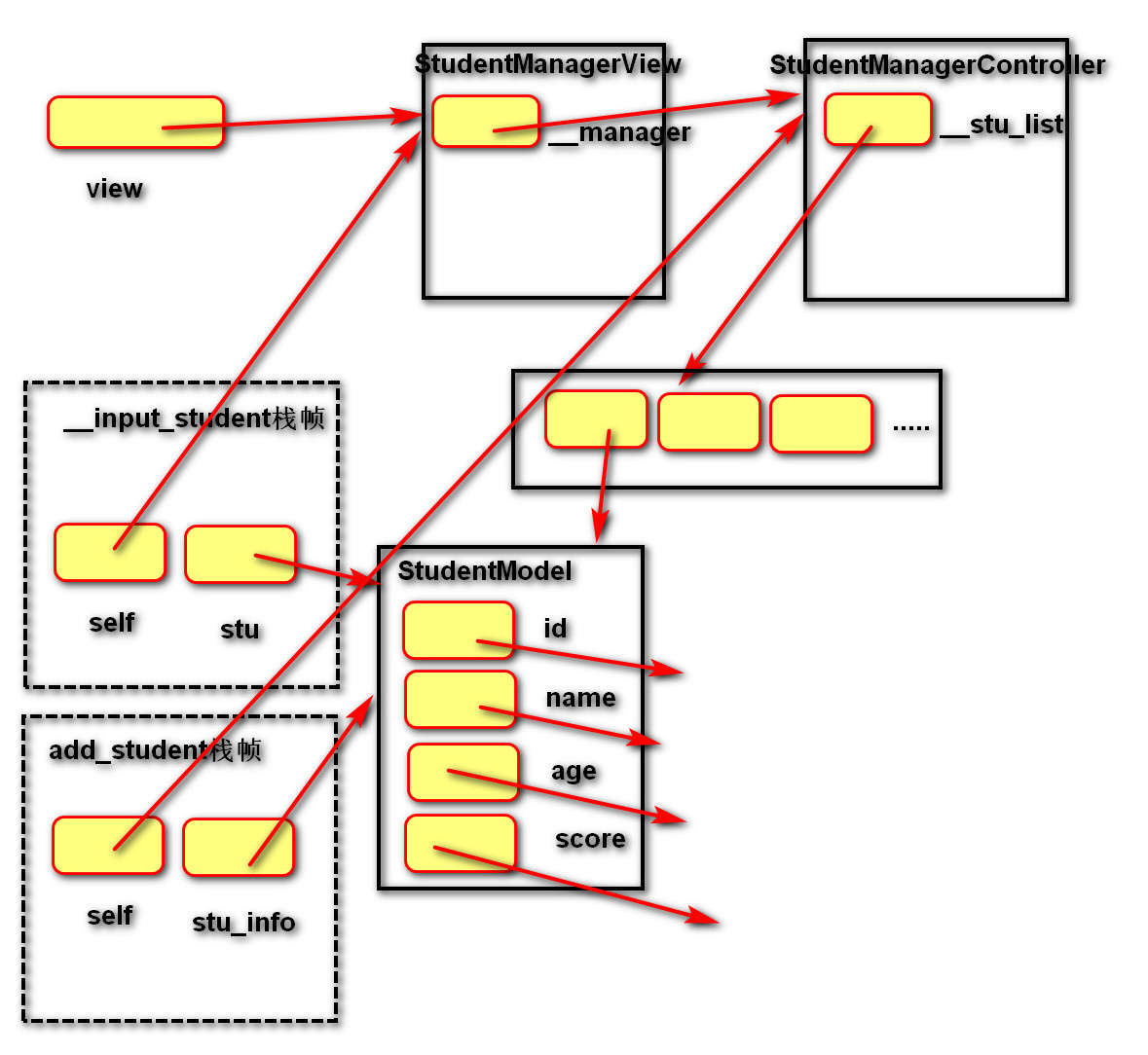
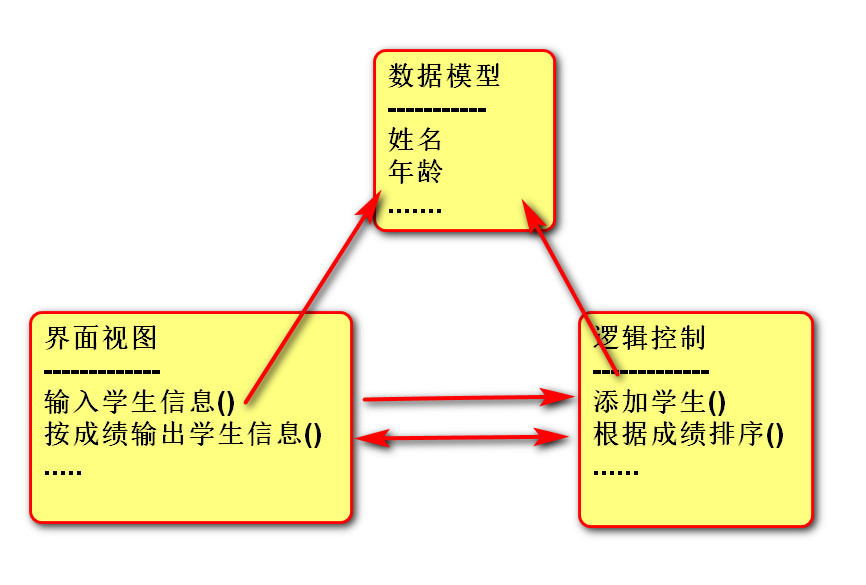




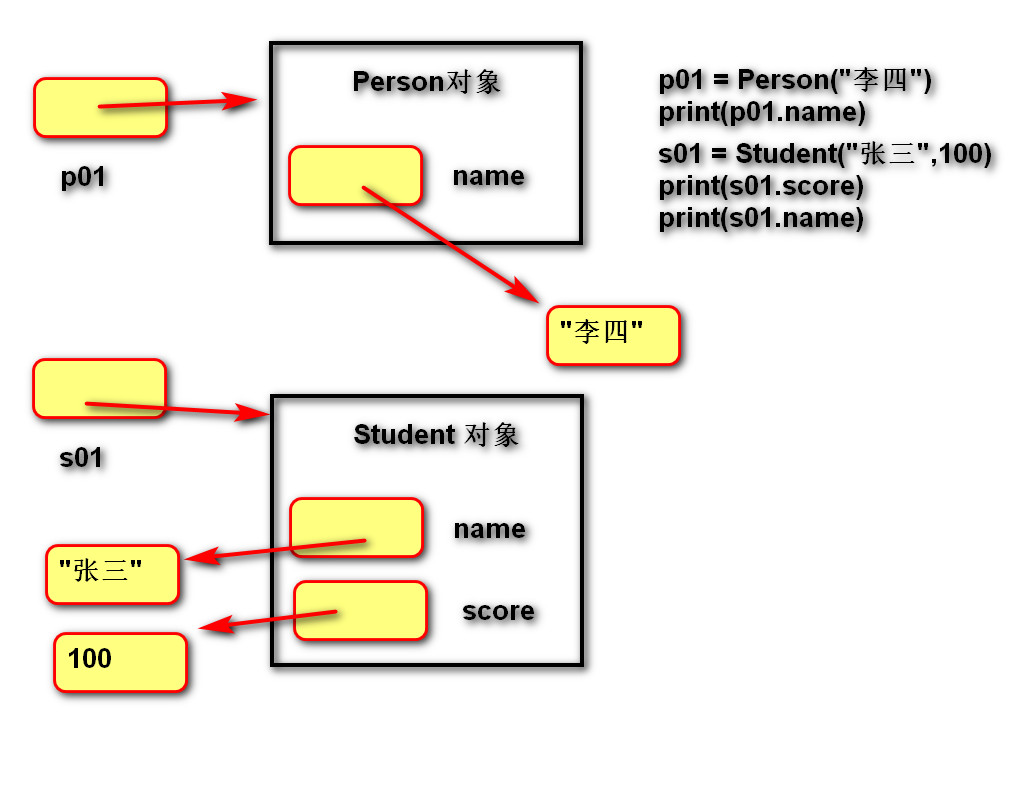
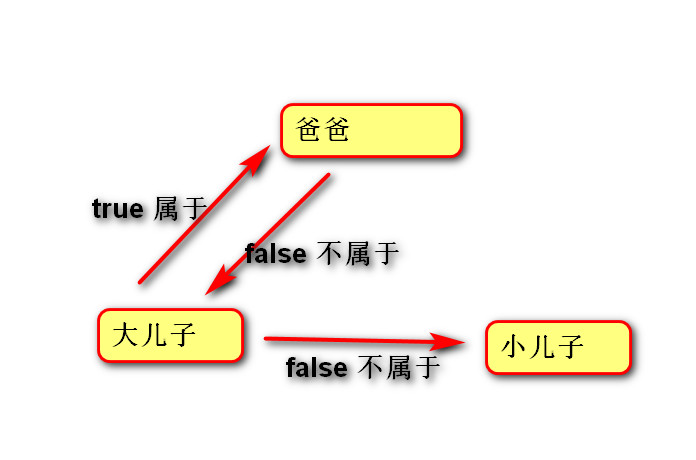
## 封装

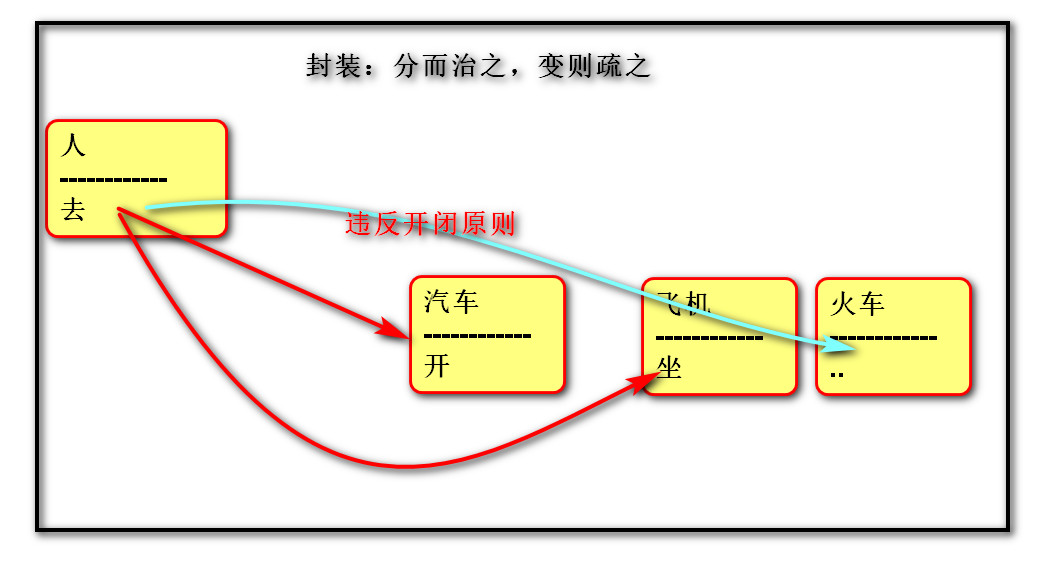


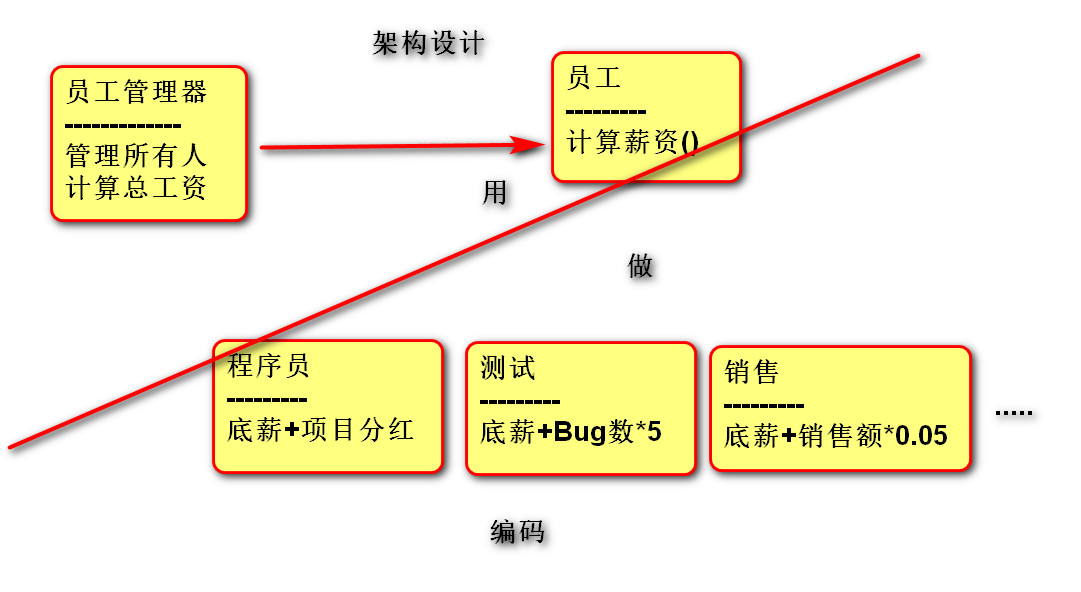
## 面向对象设计流程

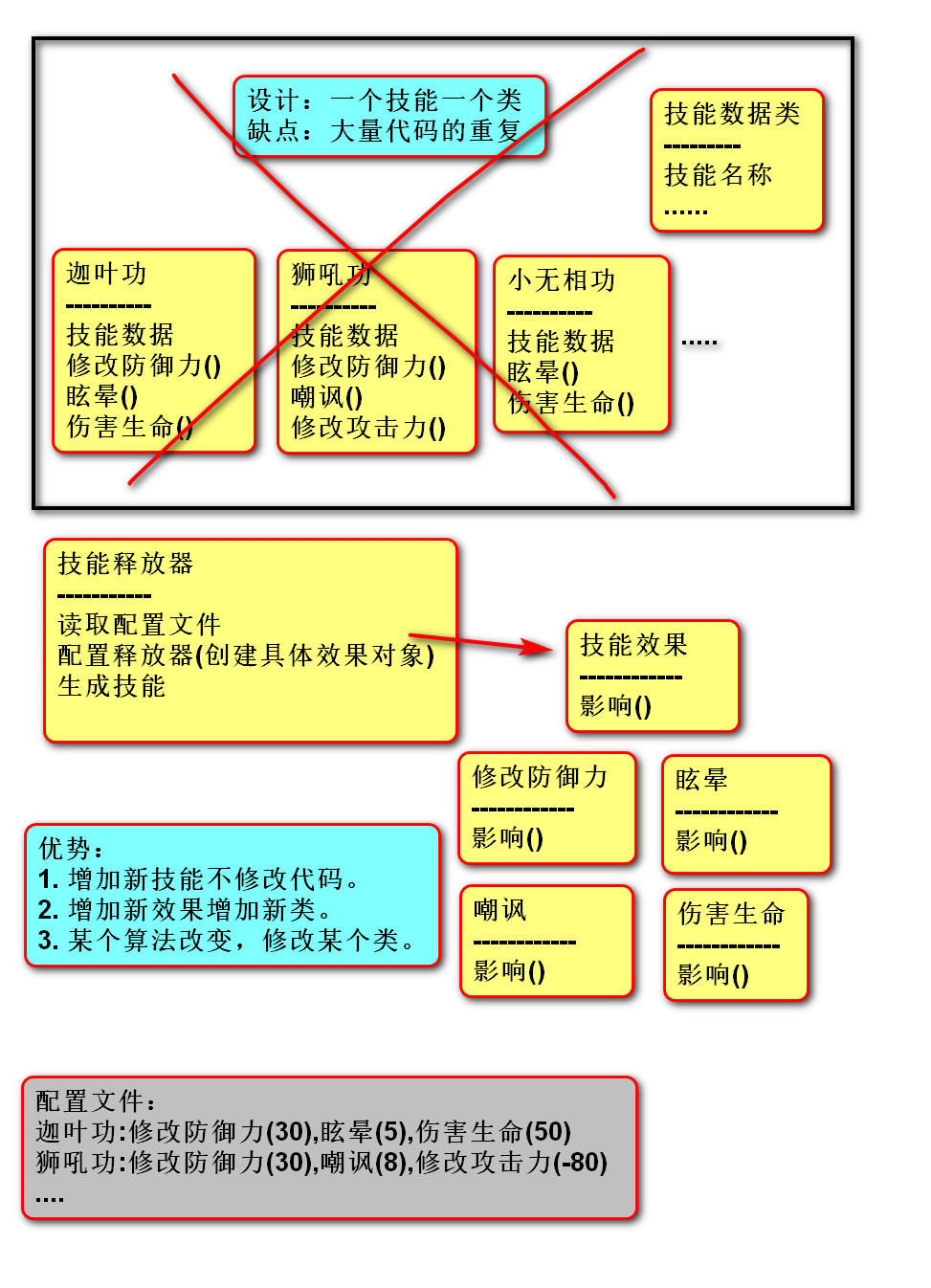


## 继承









## 迭代器

