# 前言



## 获取资源

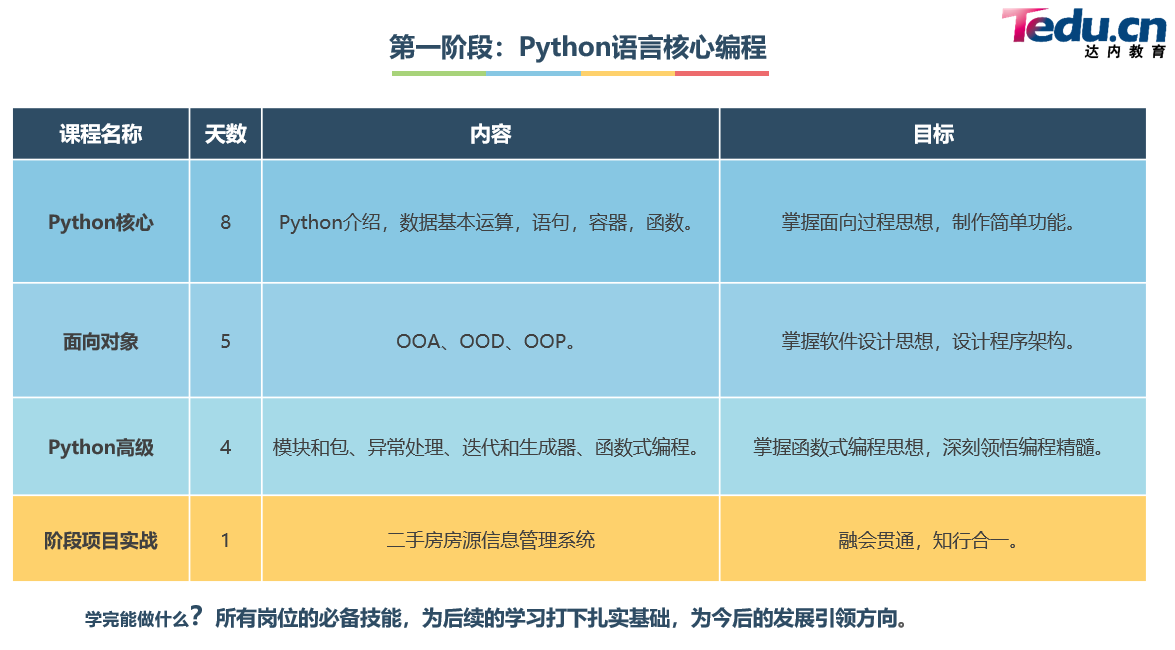
浏览器输入：code.tarena.com.cn

账号：tarenacode

密码：code\_2013

地址： AIDCode/aid2004/01\_month01/to\_student\_for\_month01.zip

## 课程介绍



## 教学理念

**理念一：弱语法，重本质**

是一种弱化语言规则，注重程序原理的学习过程。

语法是表象，只有了解深层机理，才能灵活运用。

学习编程要深入内存，剖析原理，才能看透语法。

就像太极“用意不用力，先在心后在身”的道理。

**理念二：重思想，重设计**

注重解决问题的思维方式，注重编写程序的架构。

通过面向对象三大特征，六大原则学习设计思想。

通过Model View Controller体会程序框架结构。

通过"全国面向对象课程答辩峰会"总结设计思想。

**理念三：是技术，更艺术**

编程是一门技术，但更是一种艺术。

写出高质量的代码（功能性、维护性，灵活性），享受编程所带来的乐趣。

**理念四：项目化，实战化**

2048核心算法贯穿Python核心。

疫情信息管理系统贯穿面向对象。

集成操作框架贯穿Python高级。

二手房信息管理系统贯穿全阶段。

## 学习方法

**第一步：当天笔记必须整理**

梳理当天所学内容的过程。

防止日后因为遗忘和混淆造成的麻烦。

将理论，代码，图示三合一。

**第二步：当天知识必须理解**

What 是什么，即理解知识点的定义。

Why 为什么，即理解知识点的作用。

Where 在哪里，即理解知识点的适用性。

How 如何使用，即理解知识点的语法。

**第三步：当天练习必须会做**

课堂演示用眼看 demo01、demo02 …

课堂练习动手敲 exercise01、exercise02 …

独立完成才算会做。

# Python 简介

## 计算机基础结构

### 硬件

1944年，美籍匈牙利数学家冯·诺依曼提出计算机基本结构。

五大组成部分：运算器、控制器、存储器、输入设备、输出设备。

-- 运算器：按照程序中的指令，对数据进行加工处理。

-- 控制器：根据程序需求，指挥计算机的各个部件协调工作。

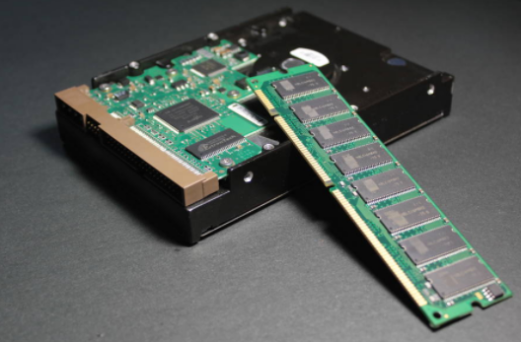
通常将运算器和控制器集成在中央处理器（CPU）中。



-- 存储器：保存各类程序的数据信息。

内存RAM -- 容量小，速度快，临时存储数据

硬盘HDD -- 容量大，速度慢，永久存储数据



输入设备：外界向计算机传送信息的装置。

例如：鼠标、键盘、扫描仪…

输出设备：计算机向外界传送信息的装置。

例如：显示器、音响、打印机…



### 软件



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，使软件开发简单化。

-- Windows，Linux，Unix。

应用软件：为了某种特定的用途而被开发的软件。

软件：程序 + 文档。

-- 程序是一组计算机能识别和执行的指令集合。

-- 文档是为了便于了解程序所需的说明性资料。

## 基础知识

### 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：ipython
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/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———|

#*Pycharm : Python 集成开发工具  
 创建工程:文件 File --> 创建项目 New Project --> 设置路径Location/解释器Interpreter  
 打开工程:Open --> 选择路径Location*

### 解释器类型

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

# 数据基本运算

## 基础知识

### 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. 删除行：shift + delete
6. 选择列：shift + alt +鼠标左键
7. 移动行：shift + alt + 上下箭头
8. 智能提示：ctrl + space
9. 代码格式化：ctrl+alt+l

### 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

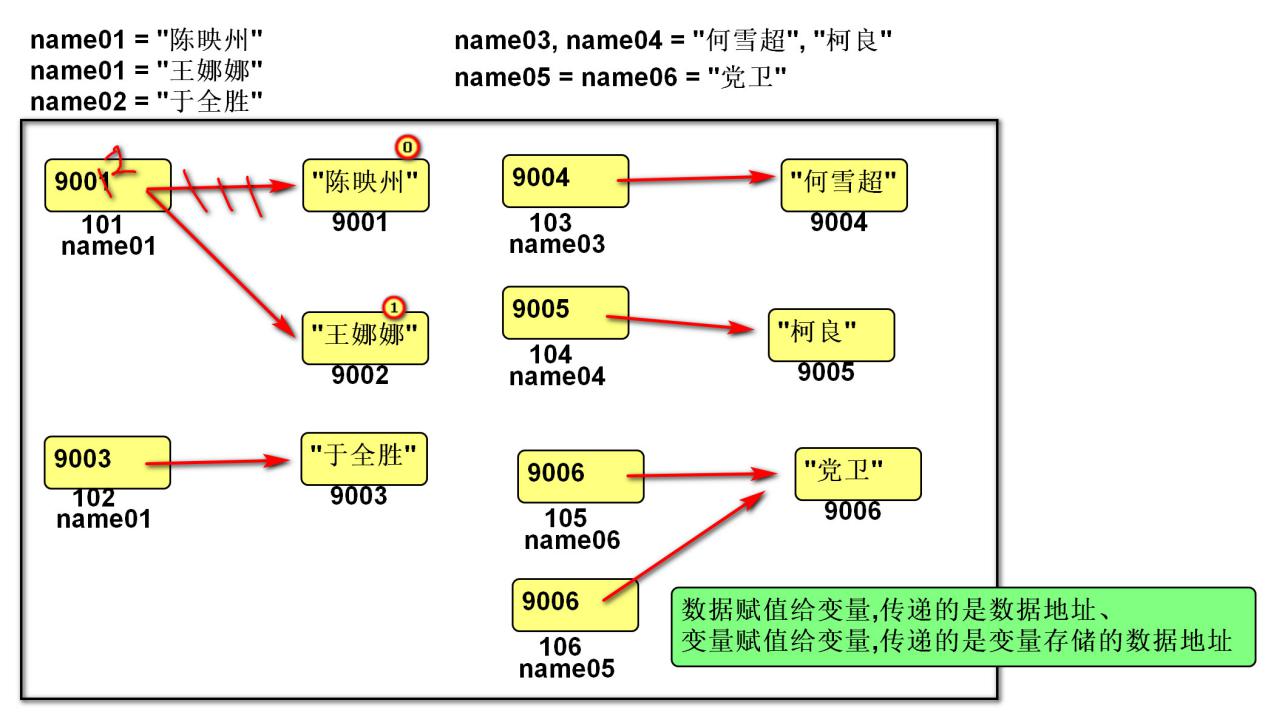
*# 1. 字面意思: 打印(需要打印的内容)  
# 2. 作用:向终端中输出/显示内容  
# 3. 适用性:显示结果*

1. 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

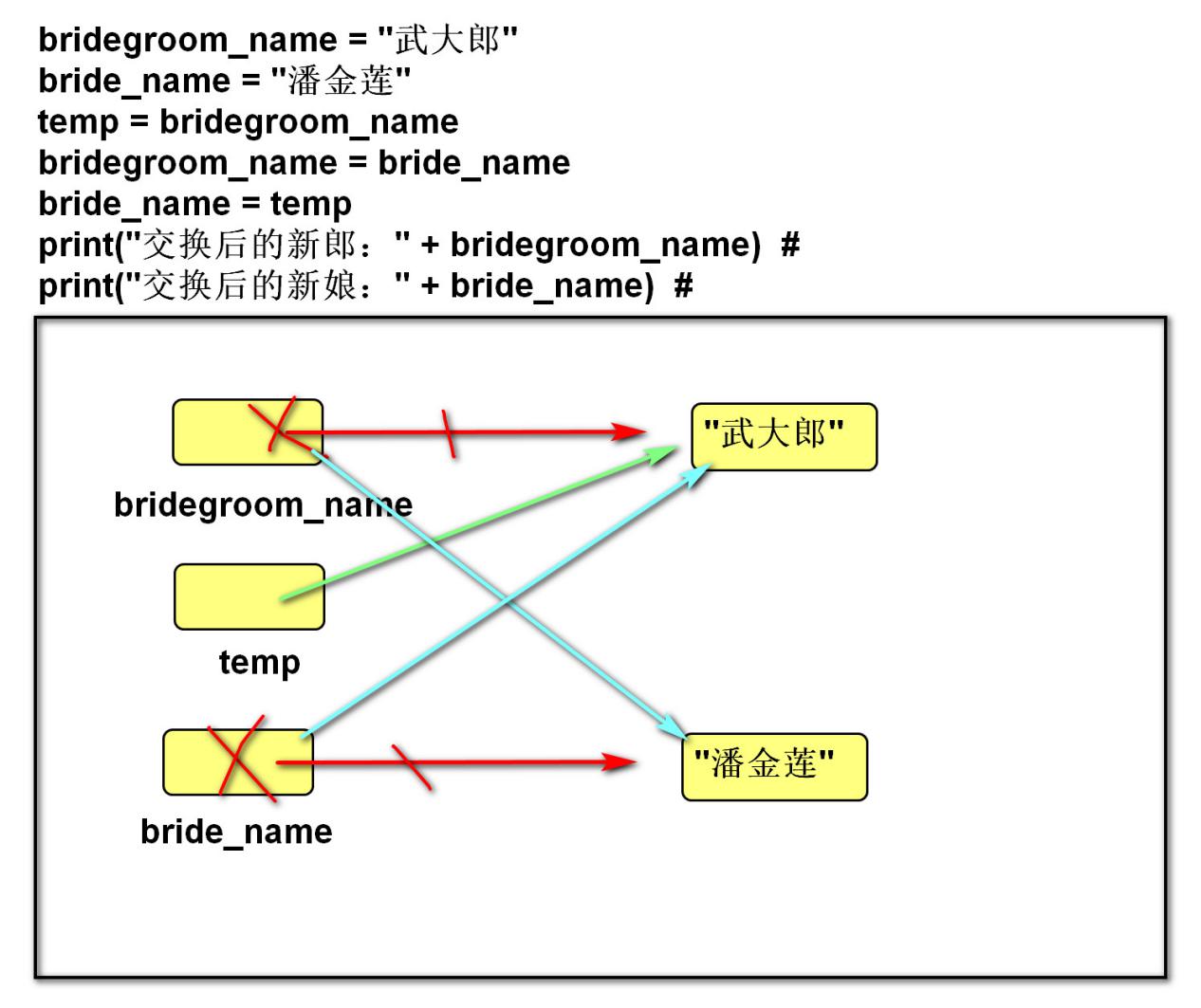
*# 赋值号 = : 将右边复制一份给左边  
# 1. 字面意思:结果 = 输入(提示信息)  
# 2. 作用: 从终端中获取用户输入的信息  
# 3. 适用性:获取数据*

## 变量

*问题一:程序运行在哪里? --- 内存中  
 问题二:程序在处理什么? --- 数据  
 作用:在内存中操作数据  
 语法1:变量名 = 数据  
 语法2:变量名1,变量名2 = 数据1,数据2  
 语法3:变量名1= 变量名2 = 数据1  
  
# 写法1:  
name01 =* ***"陈映州"*** *# 创建变量关联数据  
name01 =* ***"王娜娜"*** *# 修改变量关联的数据  
name02 =* ***"于全胜"*** *# 创建变量关联数据  
print(****"name01"****) # 打印出固定文字  
print(name01) # 打印变量(可以变化的数据) 王娜娜  
# 写法2:  
name03, name04 =* ***"何雪超"****,* ***"柯良"****# 写法3:  
# 数据赋值给变量,传递的是数据地址  
# 变量赋值给变量,传递的是变量存储的数据地址  
name05 = name06 =* ***"党卫"***

******

*两个变量交换思想：  
 左手 = "武大郎"  
 右手 = "潘金莲"  
 步骤：  
 桌子 = 左手  
 左手 = 右手  
 右手 = 桌子  
 结果  
 print(左手) # "潘金莲"  
 print(右手) # "武大郎"*

**

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 = “xxx”

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 语句语法:

*#name01 =* ***"许文杰"  
#****name02 = name01  
  
#****del*** *name01*

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。

数字类型包括（整形(int),浮点型(float)，复数(complex)）

### 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：每位用十种状态计数，逢十进一，写法是0~9。

二进制：每位用二种状态计数，逢二进一，写法是0b开头，后跟0或者1。

八进制：每位用八种状态计数，逢八进一，写法是0o开头，后跟0~7。

十六进制：每位用十六种状态计数，逢十六进一，

写法是0x开头，后跟0~9,A~F,a~f

### 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

#ea=10的a次方

1.23e-2 (等同于0.0123)

1.23456e5(等同于123456.0)

3.**复数:**由实数部分和虚数部分构成，可以用a + bj表示,实部和虚部都是浮点数,Python的复数可以用complex(a,b)表示,

### 字符串str

是用来记录文本信息(文字信息)。

字面值：双引号

### 布尔bool

*#命题:带有判断性的陈述句,返回ture或者false*

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

// 整除：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 相当于 y = y + x

y -= x 相当于 y = y - x

y \*= x 相当于 y = y \* x

y /= x 相当于 y = y / x

y //= x 相当于 y = y // x

y %= x 相当于 y = y % x

y \*\*= x 相当于 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

*#逻辑运算符  
 判断多个命题关系  
 与 and 并且 --> 都满足  
 或 or 或者 --> 一个就行  
 非 not 反过来*

#### 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

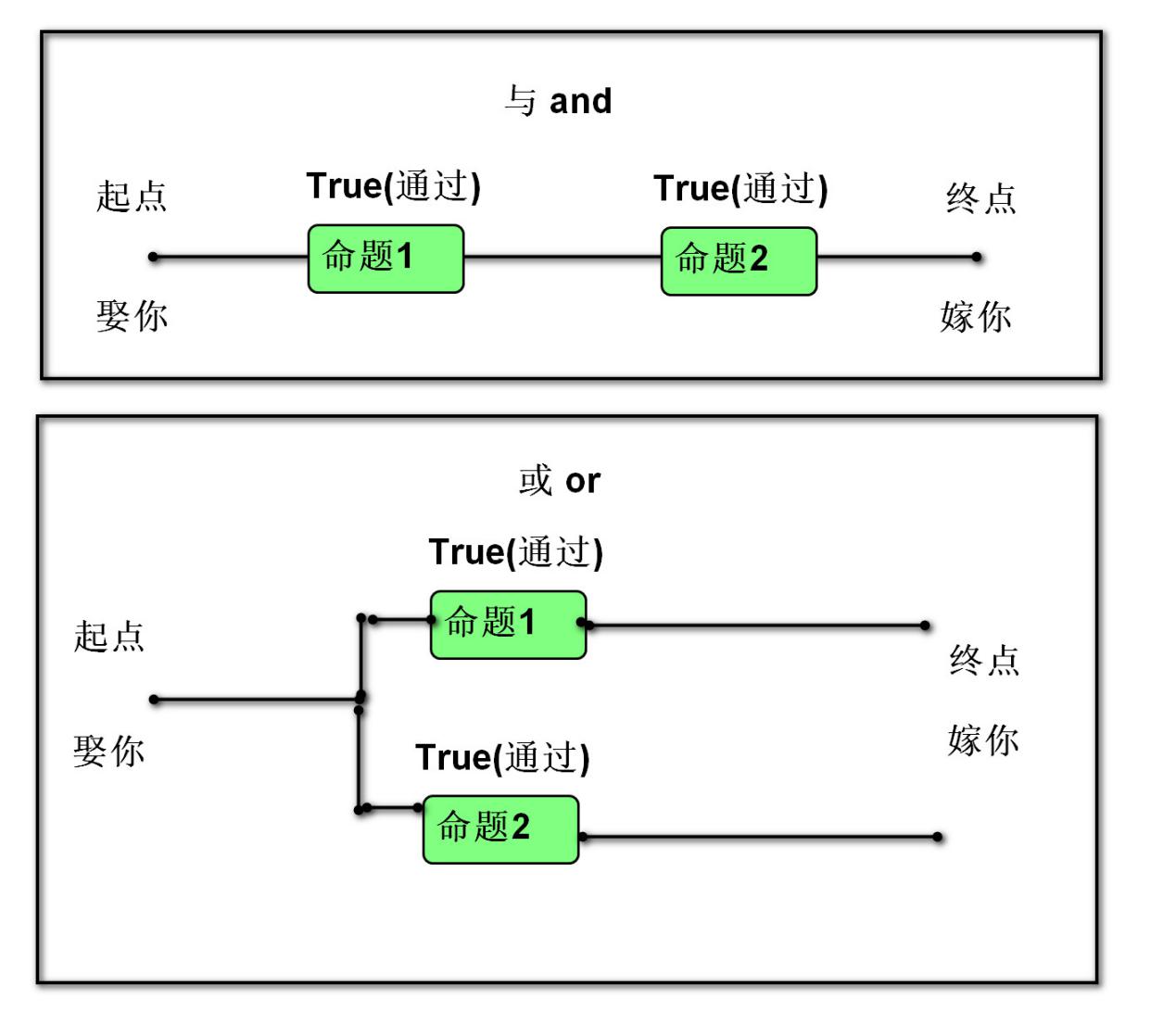
示例:

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

增强运算符

身份运算符

逻辑运算符

# 语句

*插入重点：*

*Debug 调试  
 目的:让程序中断,审查变量取值(Debugger面板的Variables),  
 查看执行流程  
 步骤:  
 1. 加入断点 -- 在可能出错的地方  
 2. 开始调试 -- Debug  
 3. 执行一步 -- F8  
 .....  
 4. 关闭终端 ctrl + Shift + F4*

*插入练习：*

*"""  
 while 的 else  
 猜数字2.0  
 最多猜三次  
 正确提示:恭喜你才对了  
 超过次数提示:游戏失败  
  
"""****import*** *random  
  
random\_number = random.randint(1, 100)  
print(random\_number)  
count = 1****while*** *count <= 3:  
 get\_number = int(input(****"请输入要猜的数字（1-100）："****))* ***if*** *get\_number == random\_number:  
 print(****"恭喜你才对了,总共猜了"*** *+ str(count) +* ***"次"****)* ***break  
 elif*** *get\_number > random\_number:  
 print(****"大了"****)* ***else****:  
 print(****"小了"****)  
  
 count += 1****else****: # if count > 3  
 # while 条件不满足才执行else语句  
 # while 循环从break结束,不执行else语句  
 print(****"游戏失败"****)*

## 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

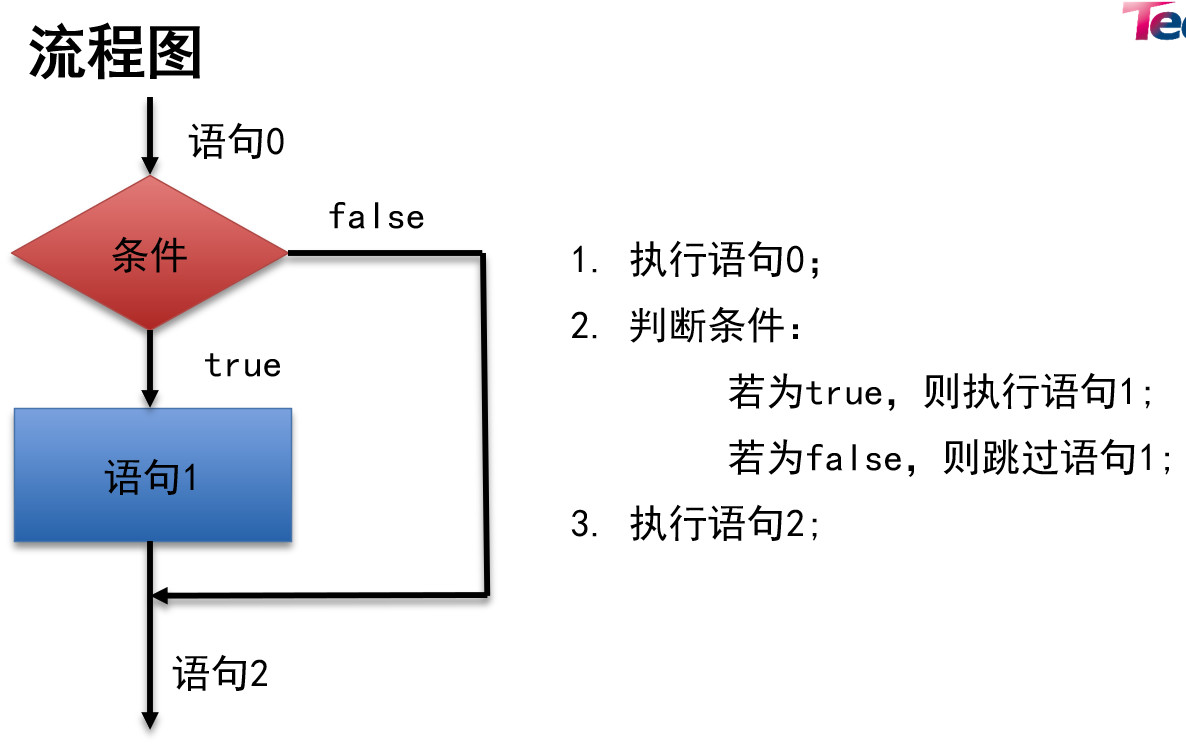
通常用来填充语法空白。

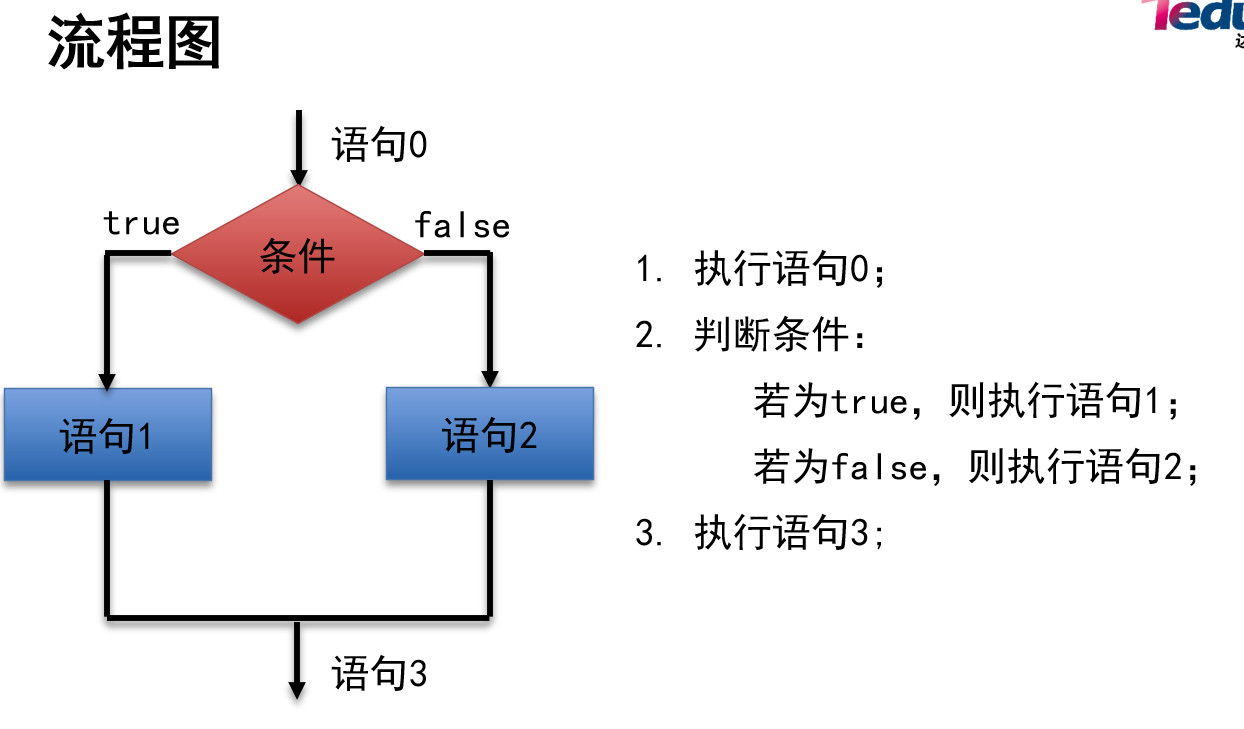
*# if 条件:  
# pass # 没想好，先占位  
# else:  
# 逻辑*

## 选择语句

### If elif else 语句

*#选择语句 -- if else  
 价值:  
 在一定条件下,执行某些语句.  
 语法:  
 if 条件:  
 满足条件执行的代码*

*  
 else:  
 不满足条件执行的代码*

**

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:

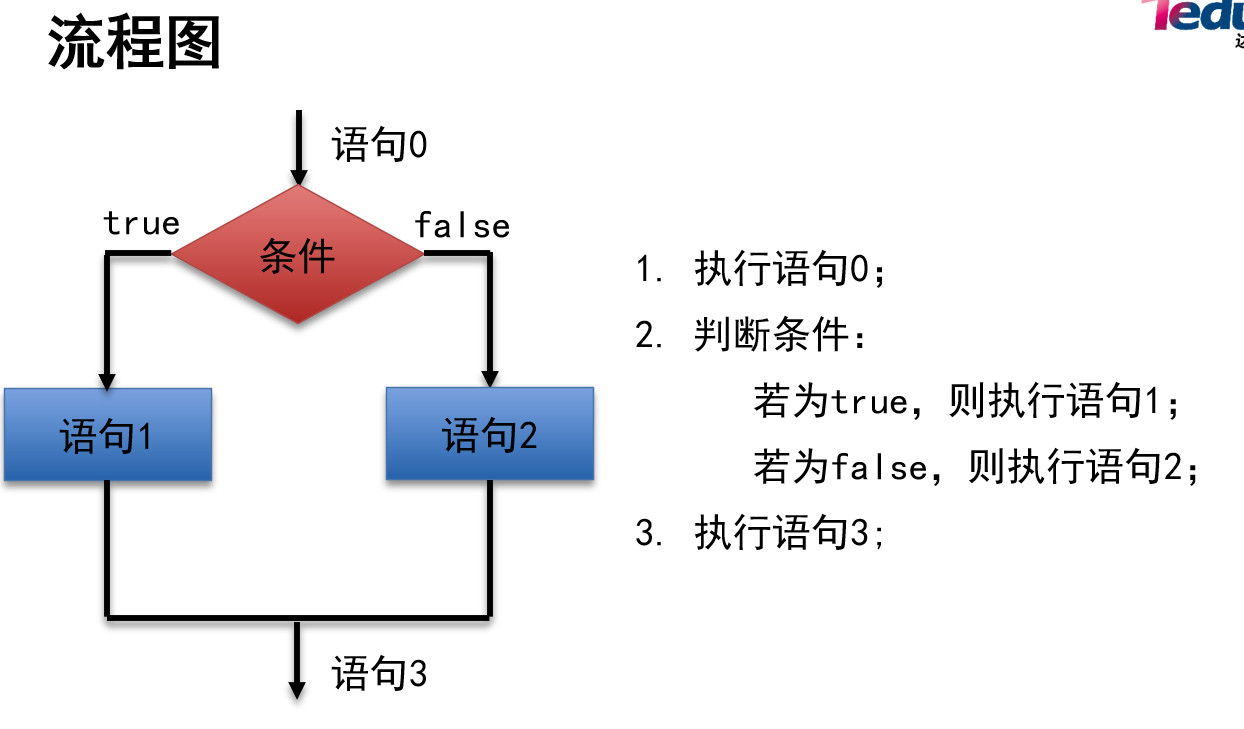
语句块1

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

*#循环以前 创建计数器*

*while 循环计数 循环条件 判断计数器是否满足条件  
 循环以内 累加*

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

*#循环，历遍目标容器*

***#for循环嵌套：***

***for*** *r* ***in*** *range(2): # 行数 0 1* ***for*** *c* ***in*** *range(5): # 列数 01234 01234  
 print(****"老王"****, end=****" "****)  
 print() # 换行*

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

*#"""  
 for + range  
 依次获取一个范围内的数  
"""  
# 写法1:  
# range(开始,结束,间隔)  
# 包含开始,不包含结束****for*** *item* ***in*** *range(2, 6, 1):  
 print(item) # 2~5  
  
# 写法2:  
# range(开始,结束)  
# 间隔默认1****for*** *item* ***in*** *range(2, 6):  
 print(item) # 2~5  
  
# 写法3:  
# range(结束)  
# 开始默认0****for*** *item* ***in*** *range(6):  
 print(item) # 0~5*

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（跳出一个循环体，若在内循环中，不跳出外循环），后面的代码不再执行。
2. 可以让while语句的else部分不执行。#else也是while的一部分。

### continue 语句

跳过本次，继续下次循环。#不跳出该循环体

# 容器类型

*"""  
 容器总结  
 字符串:存储字符编码值,不可变,序列  
 价值:表达文字信息  
 列表:存储变量,可变,序列  
 价值:定位灵活(索引/切片)/存储单一维度信息  
 元组:存储变量,不可变,序列  
 价值:省内存  
 字典:存储键值对,可变,散列  
 价值:定位快/存储多维度信息 hlj = {"city": "黑龙江", "cofirmed": 138, "total": 944}  
 集合:存储键,可变,散列  
 价值:去重复/数学运算  
  
 不可变:所有数据的本质(在内存中开辟空间后,空间不能改变大小),内存会按需分配  
 可变:在不可变基础上,赋予了扩容机制,内存会预留空间  
  
 序列:有顺序,省内存,定位灵活(可以使用索引切片)  
 散列:无顺序,费内存,定位快  
"""*

*插入重点：  
 变量交换算法:  
 a,b = b,a  
 获取极值算法:  
 假设第一个就是最大  
 使用假设的与后面的进行比较,如果发现更大,则替换.  
"""  
# 获取最大值  
list01 = [43, 45, 8, 6, 4, 8, 55, 33]  
max\_value = list01[0]****for*** *i* ***in*** *range(1, len(list01)):* ***if*** *max\_value < list01[i]:  
 max\_value = list01[i]  
print(max\_value)*

*"""  
 排序算法  
 [取]出前几个数据(不要最后一个)  
 与后面元素进行[比]较  
"""  
list01 = [4, 54, 5, 6, 67, 788]  
# list01[0]  
# for i in range(1, len(list01)):  
# if list01[0] < list01[i]:  
# list01[0], list01[i] = list01[i], list01[0]  
  
# list01[1]  
# for i in range(2, len(list01)):  
# if list01[1] < list01[i]:  
# list01[1], list01[i] = list01[i], list01[1]  
# print(list01)  
# 获取数据****for*** *r* ***in*** *range(len(list01) - 1):  
 # 与后面元素比较* ***for*** *c* ***in*** *range(r + 1, len(list01)):* ***if*** *list01[r] < list01[c]:  
 list01[r], list01[c] = list01[c], list01[r]*

*#交换赋值  
  
print(list01)*

## 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

*name =* ***"悟空"****name +=* ***"八戒"  
print****(name) # 悟空八戒*

1. \*：重复生成容器元素
2. \*=：用原容器生成重复元素, 并重新绑定变量

*name =* ***"悟空"****name \*= 72****print****(name) #“悟空悟空悟空悟空。。。。。。”*

1. < <= > >= == !=：依次比较两个容器中元素,一但不同则返回比较结果。 *返回比较结果（ture/false）***print**(**"孙悟空"** > **"猪八戒"**)

# 比较两容器开头字符串编码大小

### 成员运算符

1. 语法：

数据 in 序列

*in 成员运算 如果在指定的序列中找到值，返回bool类型。****print****(****"是齐" in "我是齐天大圣孙悟空"****) # ture*

*注意该字符串是“有序”的*

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。

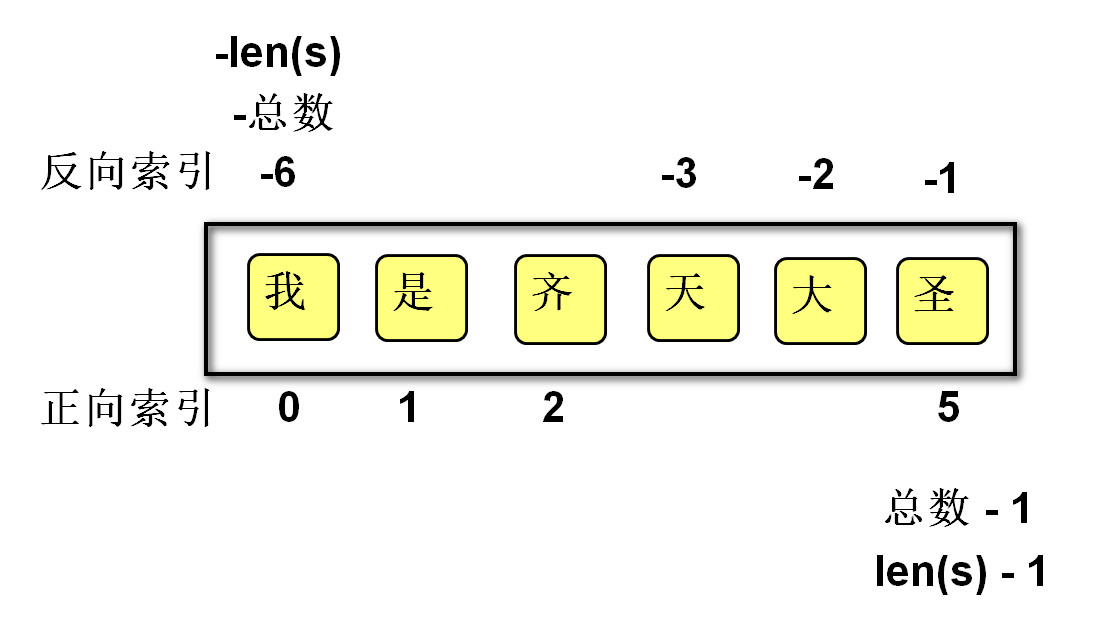
### 索引index # 取出为字符串

1. 作用：定位单个容器元素
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。 # len内建函数有解释

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)。

*# message =* ***"我是齐天大圣"  
print****(message[0]) # 获取第一个元素  
# print(message[-6]) # 获取第一个元素****print****(message[-1]) # 获取最后一个元素  
# print(message[5]) # 获取最后一个元素*

*  
  
# 注意: 索引不能越界  
# print(message[99999]) # IndexError  
# print(message[-99999]) # IndexError****print****(message[-0]) # 相当于print(message[0])*

### 切片slice #取出为列表

1. 作用：

定位多个容器元素。

1. 语法：

容器[开始索引:结束索引:步长]

1. 说明：

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

开始、结束和步长都可以省略 #默认为0、len(s)、1

*#*

*message =* ***"我是齐天大圣"  
print****(len(message))  
# 不包含结束****print****(message[0:5:1]) # 我是齐天大  
# 间隔默认1****print****(message[0:5]) # 我是齐天大  
# 开始默认为"开头"****print****(message[:5]) # 我是齐天大  
# 结束默认为"尾巴"****print****(message[:]) # 我是齐天大圣  
# 反转  
# 开始是"尾巴" 结束是"开头"****print****(message[::-1]) # 圣大天齐是我  
# 是齐天大****print****(message[1:-1])  
# 大天齐是****print****(message[-2:0:-1])  
  
# 注意:  
# 切片越界不报错****print****(message[:99999]) # 应该写　print(message[:5])*

*print(message[::-1]) 圣大天齐是我****print****(message[1:1]) # 空****print****(message[1:4:-1]) # 空  
# print(message[1:4]) # 从索引为1到索引为3元素*

### 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

*#"""  
 字符串 - 不可变  
"""  
name =* ***"悟空"****name =* ***"孙悟空"****# 不是将字符串"悟空",改为字符串"孙悟空"  
# 而是创建新字符串"孙悟空",替换变量name存储的地址.  
print(name) # "孙悟空"*

### 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

*# usd = 10  
# rmb = 70.812  
# print(str(usd) + "美元 = " + str(rmb) + " 人民币")  
  
usd = 10  
rmb = 70.892  
print(****"%.2f 美元 = %.1f 人民币"*** *% (usd, rmb))  
  
number\_jin = 3  
number\_liang = 6  
print(****"结果为：%d斤%d两"*** *% (number\_jin, number\_liang))*

*#print(f****"结果为：{****number\_jin****}斤{****number\_liang****}两" ）***

1. 定义：

生成一定格式的字符串。

1. 语法：

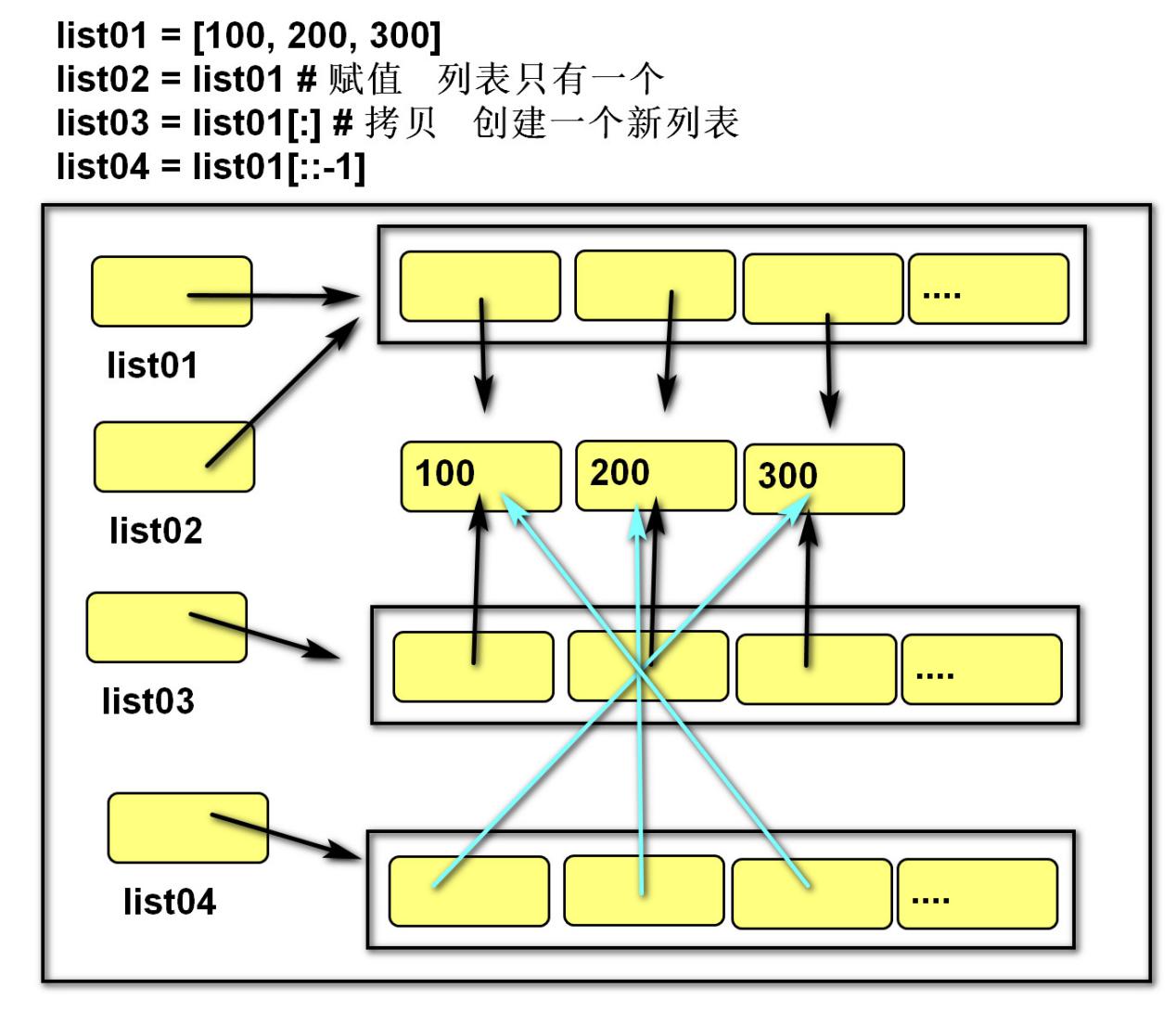
字符串%(变量)

"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

赋值：并没有创建新列表。 拷贝：创建新列表。

1. 创建列表：

列表名 = []

列表名 = list(可迭代对象)

1. 添加元素：

语法：列表名.append(元素)

*# 应用:根据条件拼接一个字符串方法：“+”以及append  
# range(10) --> "0123456789"  
# result = ""  
# for item in range(10):  
# result = result + str(item)  
# print(result)  
# “+”缺点: 每次拼接+都会产生新字符串对象,旧的成为垃圾# 解决: 频繁修改不可变对象时,使用可变对象代替.  
  
# 每次append,都追加到原有列表  
result = []****for*** *item* ***in*** *range(10):  
 result.append(str(item))  
result =* ***""****.join(result)****print****(result)*

*list\_message = [****"我"****,* ***"爱"****,* ***"你"****,* ***"p"****,* ***"y"****,* ***"t"****,* ***"h"****,* ***"o"****,* ***"n"****, 666]  
  
# "".join(list\_message) # 666 是整数，不能拼接为str  
  
str\_message =* ***""****.join([str(item)* ***for*** *item* ***in*** *list\_message])  
print(****"转化后的字符串为:"*** *+ str\_message)*

列表.insert(索引，元素)

1. 定位元素：

列表名[索引] = 元素

变量 = 列表名[索引]

变量 = 列表名[切片] # 赋值给变量的是切片所创建的新列表

列表名[切片] = 容器 # 右侧必须是可迭代对象，左侧切片没有创建新列表。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]

*#*

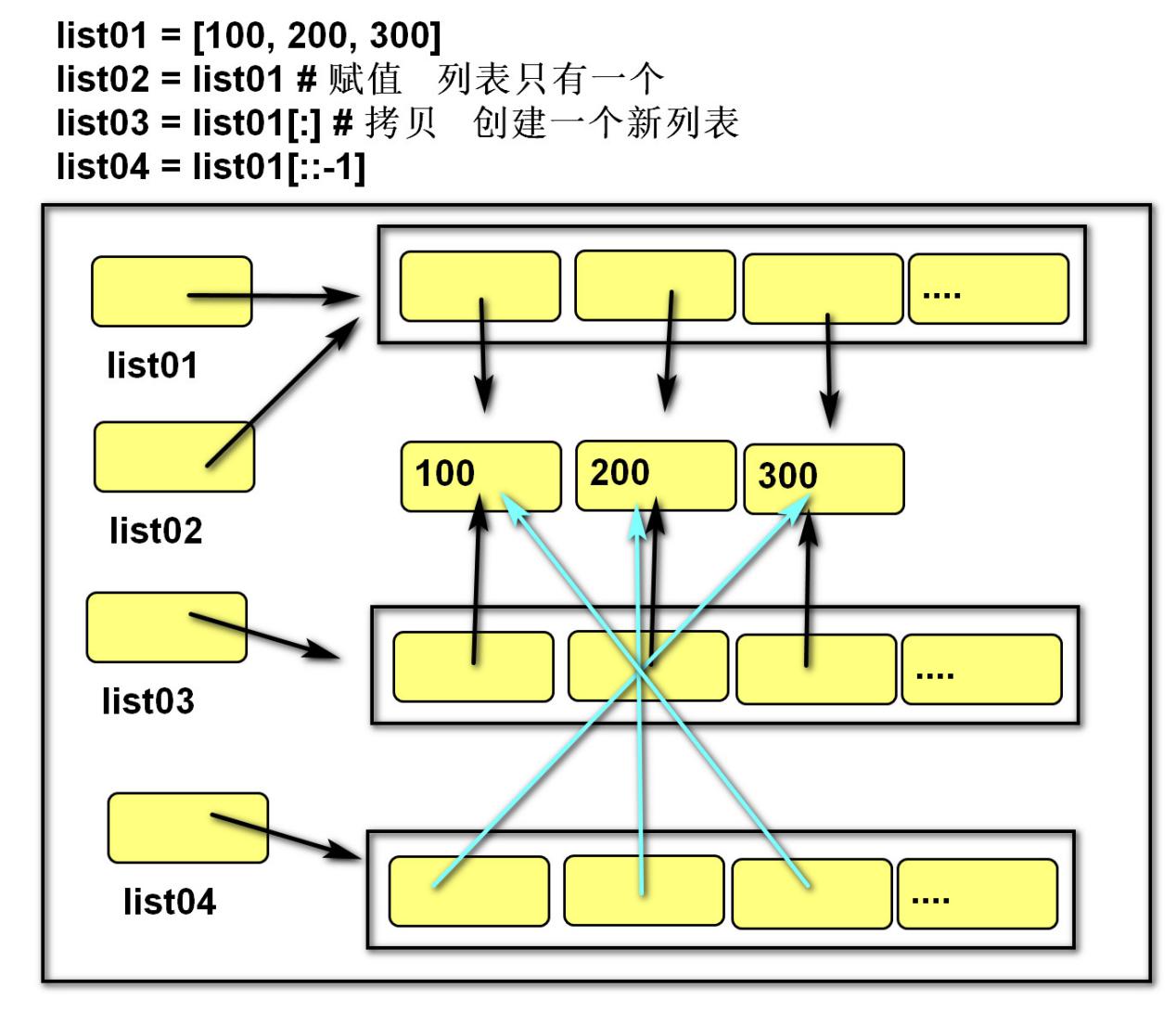
1. *创建列表  
   # -- 写法1 列表名 = [元素1, 元素2]  
   list\_names = [****"林恺"****,* ***"柯良"****,* ***"王琳雅"****]  
   # -- 写法2 列表名 = list(容器)  
   list\_numbers = list(range(6)) # [0, 1, 2, 3, 4, 5]  
   list01 = list(****"我是齐天大圣孙悟空"****) # ['我', '是', '齐', '天', '大', '圣', '孙', '悟', '空']*

*# 2. 添加元素  
# -- 写法1 列表名.append(元素)*

*#-- 只能加在最后  
list\_names.append(****"刘硕"****)  
# -- 写法2 列表名.insert(插入的索引,元素)  
#-- 可以插入任何位置*

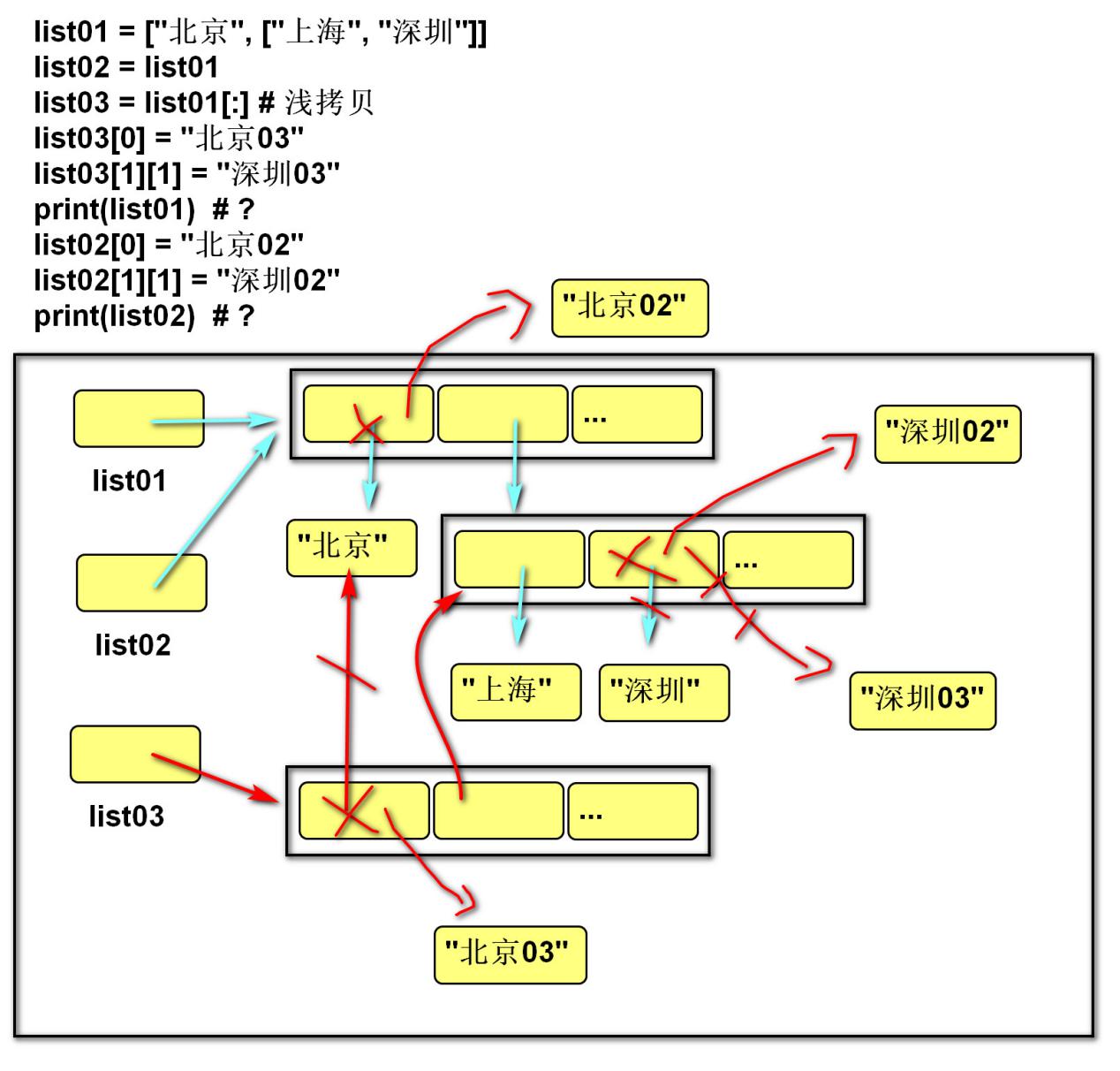
*list\_names.insert(2,* ***"李自强"****)  
  
# 3. 定位元素****print****(list\_names) # ['林恺', '柯良', '李自强', '王琳雅', '刘硕']  
# -- 索引:单个元素****print****(list\_names[2]) # 读取第三个元素  
list\_names[2] =* ***"强强"*** *# 修改第三个元素  
  
# -- 切片:多个元素  
# 通过切片获取元素,会产生新容器****print****(list\_names[-3:]) # 读取后三个元素['李自强', '王琳雅', '刘硕']  
# $遍历右侧容器,依次为左侧定位的元素赋值  
list\_names[-3:] = [****"老李"****,* ***"老王"****,* ***"老刘"****] # 修改后三个元素****print****(list\_names)  
  
# 4. 删除元素  
# -- 根据定位删除  
# del 列表名称[索引]  
# del 列表名称[切片]****del*** *list01[2] # 删除list01中第三个元素****del*** *list01[-3:] # 删除list01中最后三个元素  
# -- 根据元素删除  
# remove 内部会从头到尾进行查找,删除第一个匹配项（历遍）  
list01.remove(****"天"****)  
# 如果不知道元素是否存在,先通过in判断****if "你" in*** *list01:  
 list01.remove(****"你"****)  
  
# 5. 遍历列表  
# -- 从头到尾依次获取全部元素****for*** *name* ***in*** *list\_names:* ***print****(name)*

*# -- 索引  
# 从前向后跳着获取元素****for*** *i* ***in*** *range(0, len(list\_names), 2): # i : 0 2 4* ***print****(list\_names[i])  
  
# 从后向前获取所有元素 4 3 2 1 0  
# 开始: len(list\_names) - 1  
# 结束:-1, 不会包含-1,会获取上一个(0)  
# 间隔:-1, 倒序****for*** *i* ***in*** *range(len(list\_names) - 1, -1, -1): # 4 3 2 1 0 # 注意：range中没有反索引，所以-1的上一个为0.若是索引或者切片，-1=len(s)-1  
 # print(i) # 不是打印索引* ***print****(list\_names[i])*



*list\_devices01 = [****"存储器"****,* ***"控制器"****,* ***"运算器"****]  
# 将右侧列表地址赋值给第一个元素  
list\_devices01[0] = [****"内存"****,* ***"硬盘"****]  
  
list\_devices02 = [****"控制器"****,* ***"运算器"****,* ***"存储器"****]  
# 切片修改,左右容器长度可以不一致  
# 遍历右侧列表元素（２个） 依次赋值给左侧定位的区域(１个)  
list\_devices02[0:1] = [****"内存"****,* ***"硬盘"****] #[ '内存', '硬盘', '运算器', '存储器']  
# 遍历右侧列表元素（1个） 依次赋值给左侧定位的区域(3个)  
list\_devices02[1:] = [****"cpu"****]# ['内存', 'cpu']  
  
print(list\_devices01)# [['内存', '硬盘'], '控制器', '运算器']  
print(list\_devices02)# ['内存', 'cpu']*

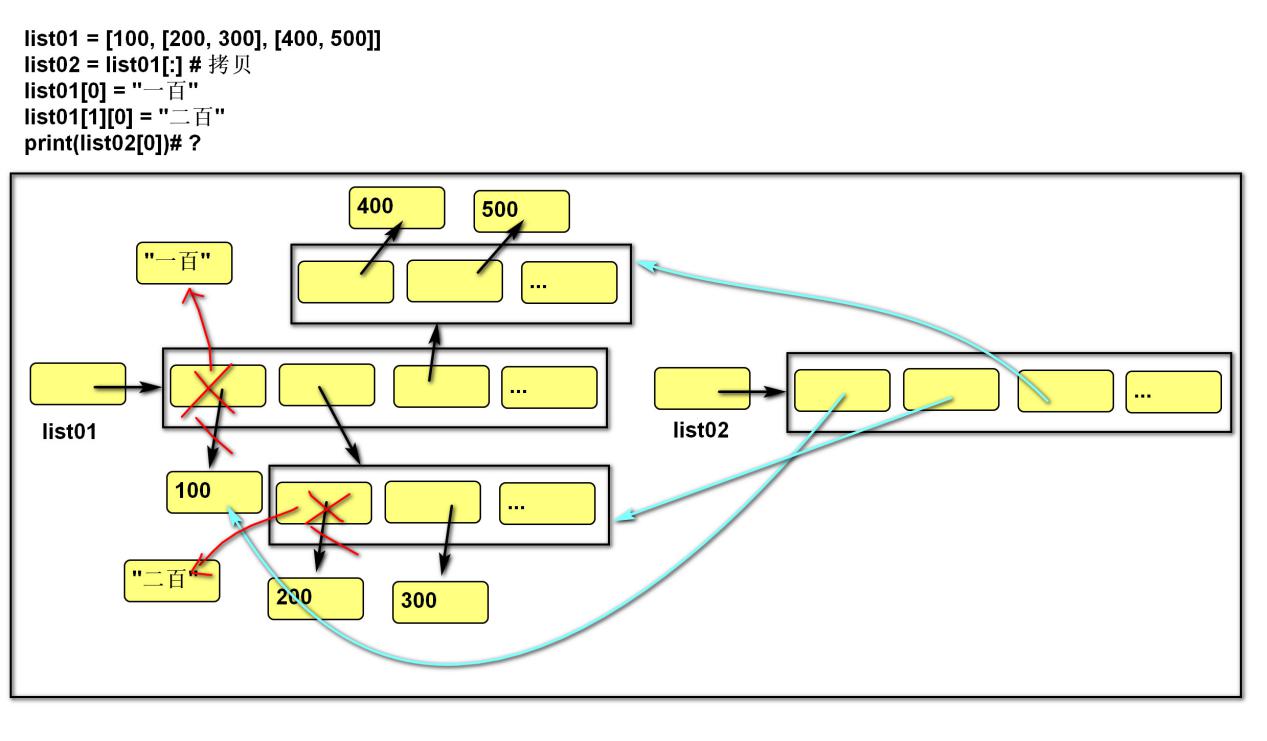
### 深拷贝和浅拷贝



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。

# 若列表中存在嵌套时切片，原列表嵌套内容改变，切片列表会改变

List\_1 = [0,1,[2,3],4,5] # list\_1[2][0] = 2

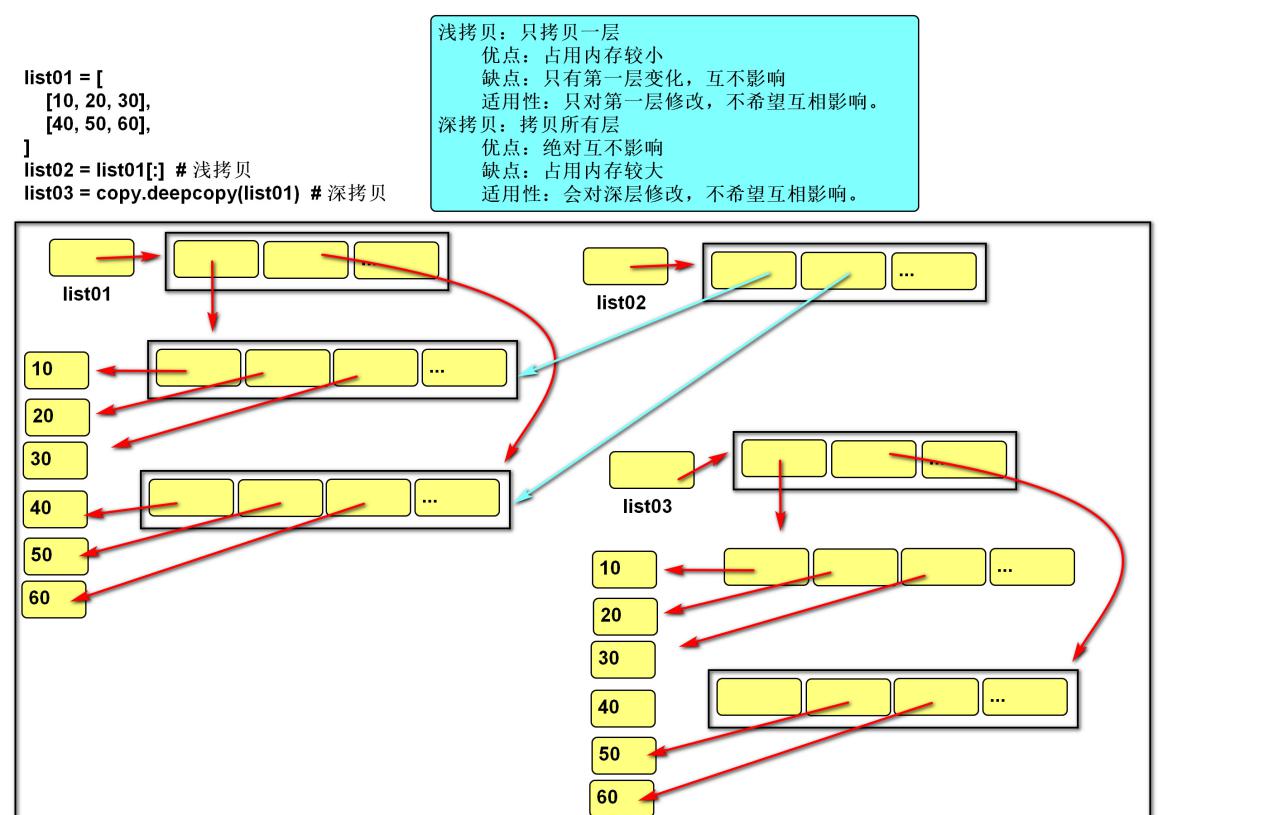


————————————————————————

*深拷贝：复制整个依懒的变量。*

*list01 = [  
 [10, 20, 30],  
 [40, 50, 60],  
]  
list02 = list01[:] # 浅拷贝  
list03 = copy.deepcopy(list01) # 深拷贝*

*# 验证:  
# 浅拷贝修改深层,互相影响  
# list02[0][0] = "十"  
# print(list01) = [[十, 20, 30], [40, 50, 60]]  
  
# 浅拷贝修改第一层,互不影响  
# list02[0] = "列表"  
# print(list01)# [[10, 20, 30], [40, 50, 60]]  
# print(list02)#　['列表', [40, 50, 60]]  
  
# 深拷贝修改,互不影响  
list03[0] =* ***"列表"****list03[1][0] =* ***"四十"  
print****(list01)# [[10, 20, 30], [40, 50, 60]]****print****(list03)# ['列表', ['四十', 50, 60]]*



### 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)

*# 语法  
list01 = [****"a"****,* ***"b"****,* ***"c"****]  
result =* ***"-"****.join([****"a"****,* ***"b"****,* ***"c"****])****print****(result) #“a-b-c”*

将一个字符串拆分为多个。

列表 = “a-b-c-d”.split(“分隔符”)

*# 语法  
result =* ***"a-b-c-d"****.split(****"-"****)****print****(result) # ['a', 'b', 'c', 'd']  
  
# 应用:一个字符串表达多个信息  
str\_names =* ***"悟空\_唐三藏\_八戒"****list\_names = str\_names.split(****"\_"****)****print****(list\_names) #****【“悟空”，“唐三藏”，“八戒”】***

### 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

*"""  
 列表推导式嵌套  
"""  
list01 = [****"香蕉"****,* ***"苹果"****,* ***"哈密瓜"****]  
list02 = [****"牛奶"****,* ***"咖啡"****,* ***"雪碧"****,* ***"可乐"****]  
# result = []  
# for r in list01:  
# for c in list02:  
# result.append(r + c)  
result = [r + c* ***for*** *r* ***in*** *list01* ***for*** *c* ***in*** *list02]  
print(result)#“香蕉牛奶”，“香蕉咖啡”、、、*

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)

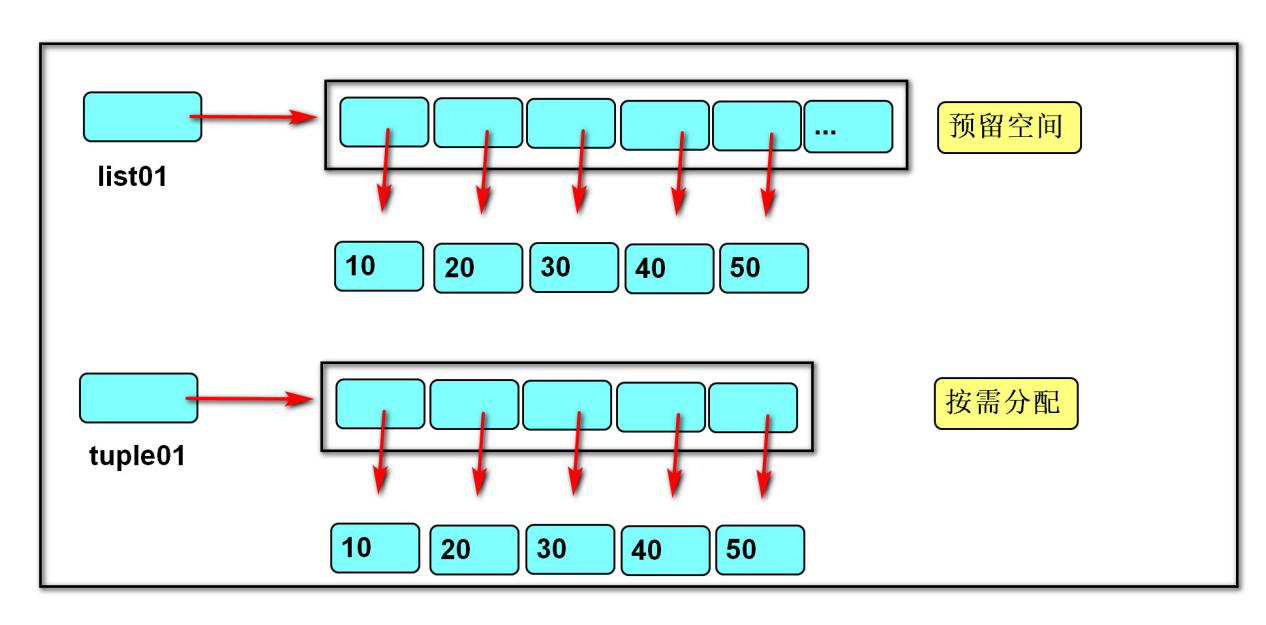
1. 推导式写法：

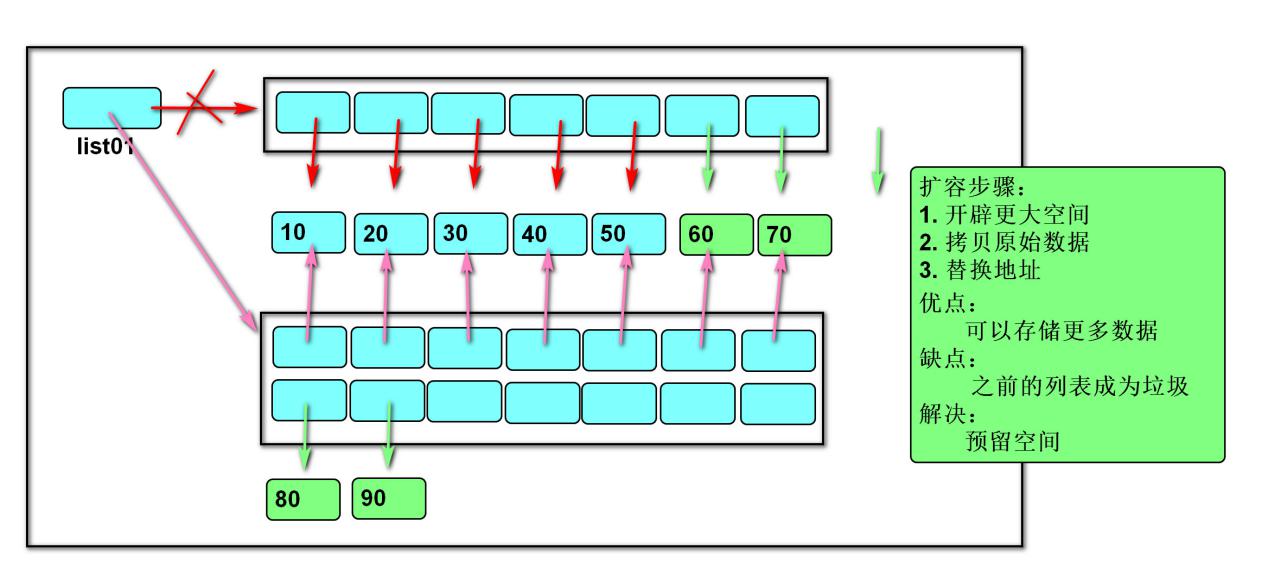
result = [r + c for r in list01 for c in list02]

*# result = []  
# for item in range(10):  
# result.append(str(item))  
result = [str(item)* ***for*** *item* ***in*** *range(10)]****print****(result) # ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']  
  
# result = []  
# for number in range(20):  
# if number % 2 == 0:  
# result.append(number)  
result = [number* ***for*** *number* ***in*** *range(20)* ***if*** *number % 2 == 0]****print****(result) #0,2,4,6,8,10,12,14,16,18*

*# list\_year = []  
# for item in range(1970, 2051):  
# if item % 4 == 0 and item % 100 != 0 or item % 400 == 0:  
# list\_year.append(item)  
  
list\_year = [item* ***for*** *item* ***in*** *range(1970, 2051)* ***if*** *item % 4 == 0* ***and*** *item % 100 != 0* ***or*** *item % 400 == 0]  
print(list\_year)*

## 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。

*#不可变:不能扩容,内存采用按需分配的存储机制.  
 如果增加新元素,会创建新对象.  
 name = "悟空"  
 name = "孙悟空" # 创建新字符串,"悟空"成为垃圾  
 因为如果在原有基础上修改,会破坏其他对象内存空间.(损人利己)  
 优势是节省内存  
可变:具有扩容功能,内存采用预留空间的存储机制.  
 list01 = [10]  
 list01.append(20)  
 因为预留了多余空间,所以可以在原有基础上修改.  
 优势是使用方便(存储更多信息)  
面试题:  
 Python 数据类型有哪些?  
 可变类型 不可变类型  
 (list...) (int/float/bool/str...)*

### 基础操作

1. 创建空元组：

元组名 = ()

元组名 = tuple()

*# -- 写法1 元组名称 = (元素1,元素2)  
tuple01 = (10, 20, 30, 40, 50)  
list01 = [****"悟空"****,* ***"八戒"****]  
# 预留空间 --> 按需分配  
# -- 写法2 元组名称 = tuple(其他容器)  
tuple02 = tuple(list01)*

*#如果元组只有一个元素,必须写逗号  
# tuple03 = ("汉堡") 为字符串  
tuple03 = (****"汉堡"****,)****print****(type(tuple03)) #为元组*

*#创建元组的小括号,可以省略  
tuple04 =* ***"汉堡"****,* ***"薯条"****,* ***"鸡翅"  
print****(tuple04)  
  
# 拆包  
# a, b, c = ("汉堡", "薯条", "鸡翅")  
# a, b, c = ["汉堡", "薯条", "鸡翅"]  
a, b, c =* ***"大薯条"  
print****(a)****print****(b)****print****(c)*

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：

变量 = 元组名[索引]

变量 = 元组名[切片] # 赋值给变量的是切片所创建的新列表

*# -- 索引****print****(tuple01[1]) # 打印元组的第二个元素  
# -- 切片****print****(tuple01[:3]) # 打印元组的第三个元素 (10, 20, 30)*

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):

元组名[索引名]就是元素

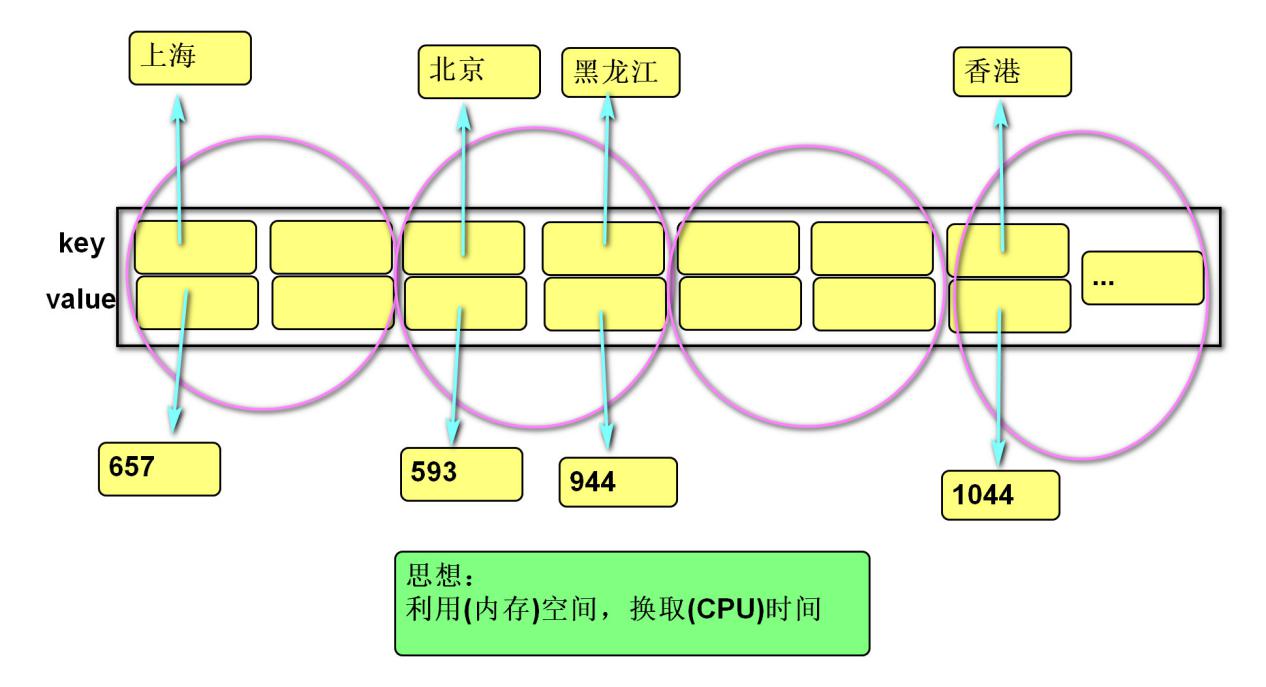
### 作用

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 #根据什么找什么

1. 由一系列键值对组成的可变散列容器。
2. 散列：对键进行哈希运算，确定在内存中的存储位置，每条数据存储无先后顺序。
3. 键必须惟一且不可变(字符串/数字/元组)，值没有限制。

*dict\_color\_info = {****"R"****:* ***"红色"****,* ***"G"****:* ***"绿色"****,* ***"B"****:* ***"蓝色"****,* ***"A"****:* ***"透明度"****}  
color = input(****"请输入颜色(RGBA):"****)  
# print(dict\_color\_info[color]) # 如果字典不存在当前key,会报错.****if*** *color* ***in*** *dict\_color\_info:  
 print(dict\_color\_info[color])****else****:  
 print(****"您输入的颜色不存在"****)*

### 基础操作

*#列表擅长存储一个维度信息  
 例如:疫情地区名称  
 list\_city = ["黑龙江", "香港", "台湾"]  
 确诊人数  
 list\_cofirmed = [138, 104, 95]  
 累计人数  
 list\_total = [944, 1040, 440]  
#字典擅长存储多个维度信息  
 例如:黑龙江疫情信息  
 hlj = {"city": "黑龙江", "cofirmed": 138, "total": 944}*

*# 列表存储多个维度信息  
# hlj = ["黑龙江", 138, 944]  
# print("地区名称:" + hlj[0])  
# print("确诊人数:" + str(hlj[1]))  
# print("累计人数:" + str(hlj[2]))  
# 代码可读性差: 0 --> 地区名称 1 --> 确诊人数 2 --> 累计人数  
  
hlj = {****"city"****:* ***"黑龙江"****,* ***"cofirmed"****: 138,* ***"total"****: 944}****print****(****"地区名称:"*** *+ hlj[****"city"****])****print****(****"确诊人数:"*** *+ str(hlj[****"cofirmed"****]))****print****(****"累计人数:"*** *+ str(hlj[****"total"****]))  
# 代码可读性强: city --> 地区名称 cofirmed --> 确诊人数 total--> 累计人数*

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)

*dict01 = {****"北京"****: 593,* ***"上海"****: 657}  
# -- 写法2 字典名称 = dict(格式) 格式 -- > [(k,v),(k,v)]  
# 注意:格式的核心思想是能够拆分为键和值  
dict02 = dict([(****"ts"****,* ***"唐僧"****), (****"bj"****,* ***"猪八戒"****), (****"ss"****,* ***"沙和尚"****)])  
# dict02 = dict(["唐僧", ("bj", "猪八戒"), ["ss", "沙和尚"]])****print****(dict02) # {'ts': '唐僧', 'bj': '猪八戒', 'ss': '沙和尚'}*

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，修改值。

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

*# -- 获取字典中元素****for x******in*** *dict01():* ***print****(x) # -- 获取所有键*

***for*** *value* ***in*** *dict01.values():* ***print****(value) # -- 获取所有值*  
  
*# -- 获取所有键,值  
item 是元组(键,值)  
for item in dict01.items():  
 print(item[0])  
 print(item[1])  
  
# k,v = 元组(键,值)****for*** *k, v* ***in*** *dict01.items():* ***print****(k)* ***print****(v)*

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}

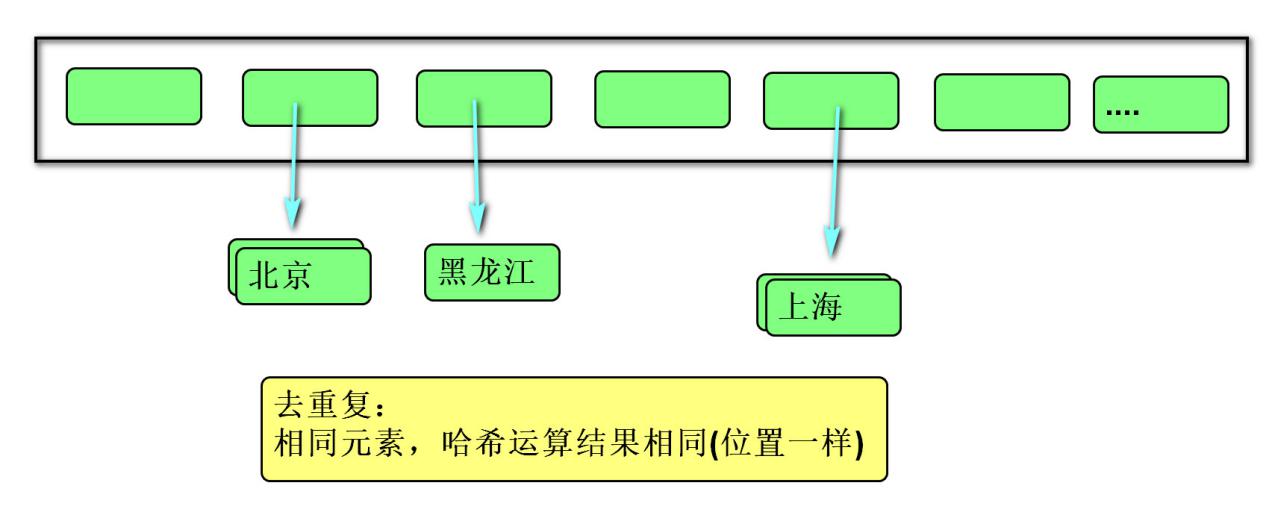
{键:值 for 变量 in 可迭代对象 if 条件}

*"""  
  
 列表推导式 [对变量的处理 for 变量 in 其他容器 if 条件]  
 字典推导式 {对变量的处理 : 对变量的处理 for 变量 in 其他容器 if 条件}  
"""  
# 需求: key:0~9 value: key的平方  
# dict01 = {}  
# for number in range(10):  
# dict01[number] = number \*\* 2  
  
dict01 = {number: number \*\* 2* ***for*** *number* ***in*** *range(10)}  
print(dict01) # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}  
  
# 需求: key:0~9 能被3整除的 value: key的平方  
# dict01 = {}  
# for number in range(10):  
# if number % 3 == 0:  
# dict01[number] = number \*\* 2  
  
dict01 = {number: number \*\* 2* ***for*** *number* ***in*** *range(10)* ***if*** *number % 3 == 0}  
print(dict01) # {0: 0, 3: 9, 6: 36, 9: 81}*

*——————————————————*

*names = [****"张无忌"****,* ***"赵敏"****,* ***"周芷若"****]  
rooms = [101, 102, 103]  
dict01 = {names[i]: rooms[i]* ***for*** *i* ***in*** *range(len(names))}  
  
dict02 = {value: key* ***for*** *key, value* ***in*** *dict01.items()}  
print(dict02)  
  
# 需求: 根据值查找键  
# 解决方案1: 键值对颠倒  
# 解决方案2:****for*** *key, value* ***in*** *dict01.items():* ***if*** *value == 103:  
 print(key)*

## 集合 set



### 定义 #去重复

1. 由一系列不重复的不可变类型变量(元组/数/字符串)组成的可变散列容器。
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)

**————————————————————————————**

*# set01 = {} # 是字典 不是集合  
set01 = {****"北京"****,* ***"上海"****,* ***"黑龙江"****}  
print(set01) # {'上海', '黑龙江', '北京'}  
  
list01 = [****"北京"****,* ***"北京"****,* ***"北京"****]  
set02 = set(list01)  
print(set02) # {'北京'}*

*————————————————*

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)

*# set01.add(****"重庆"****)  
 set01.add(****"上海"****) # 如果添加已经存在的元素,集合 中仍然只有一份  
 print(set01)*

1. 删除元素：

集合名.discard(元素)

*# set01.remove(****"北京"****)删除set01中的“北京”*

### 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：

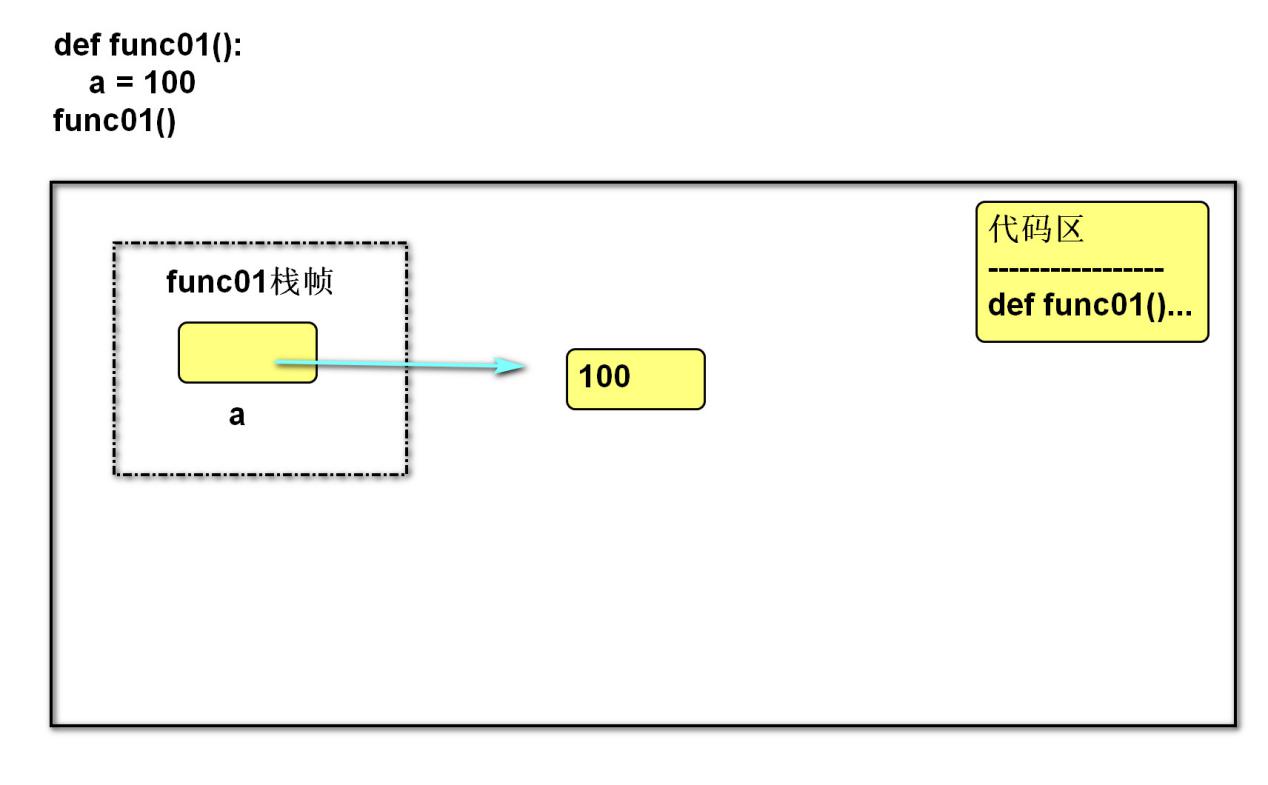
使用简易方法，将可迭代对象转换为集合。

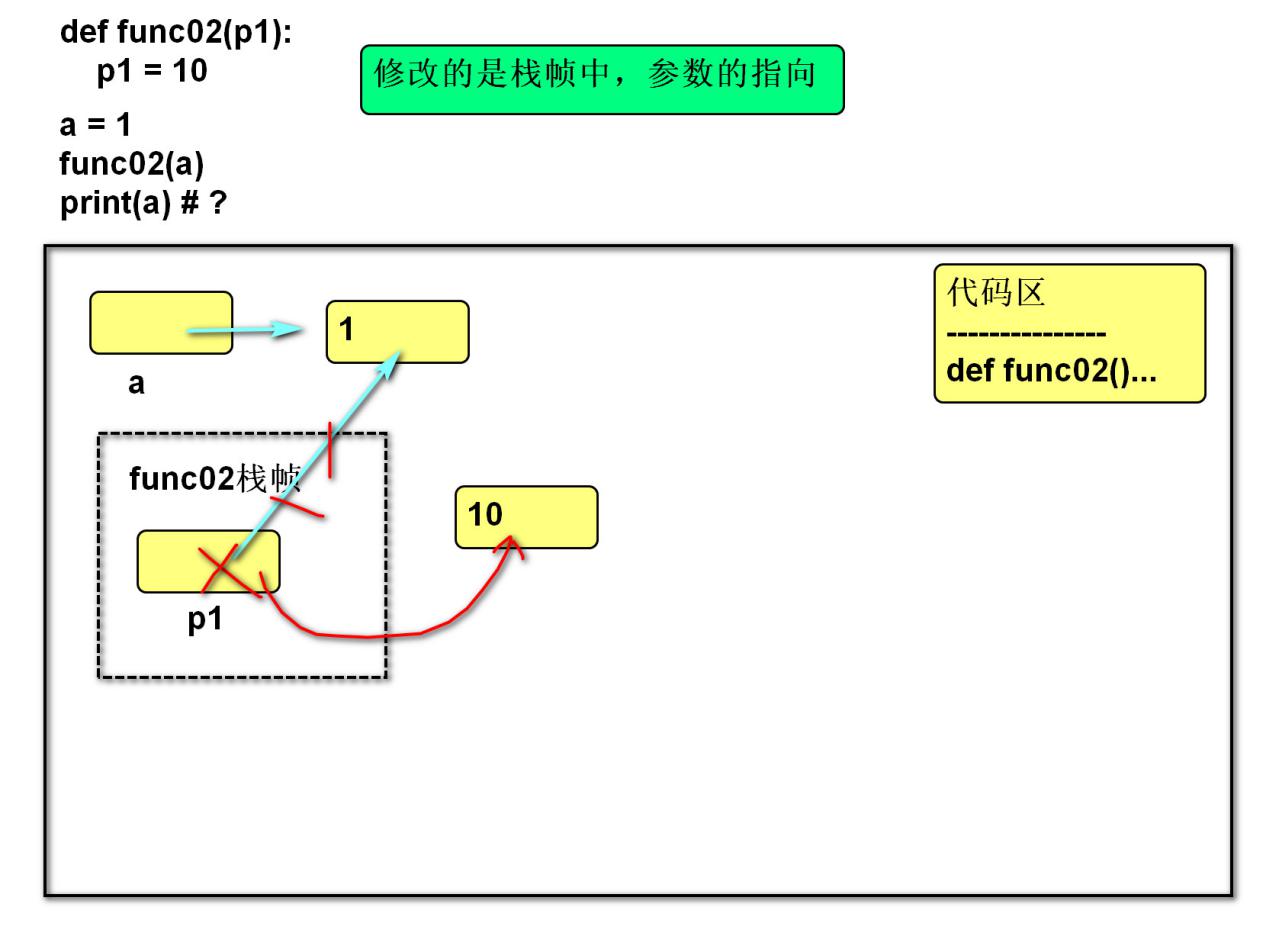
1. 语法:

{表达式 for 变量 in 可迭代对象}

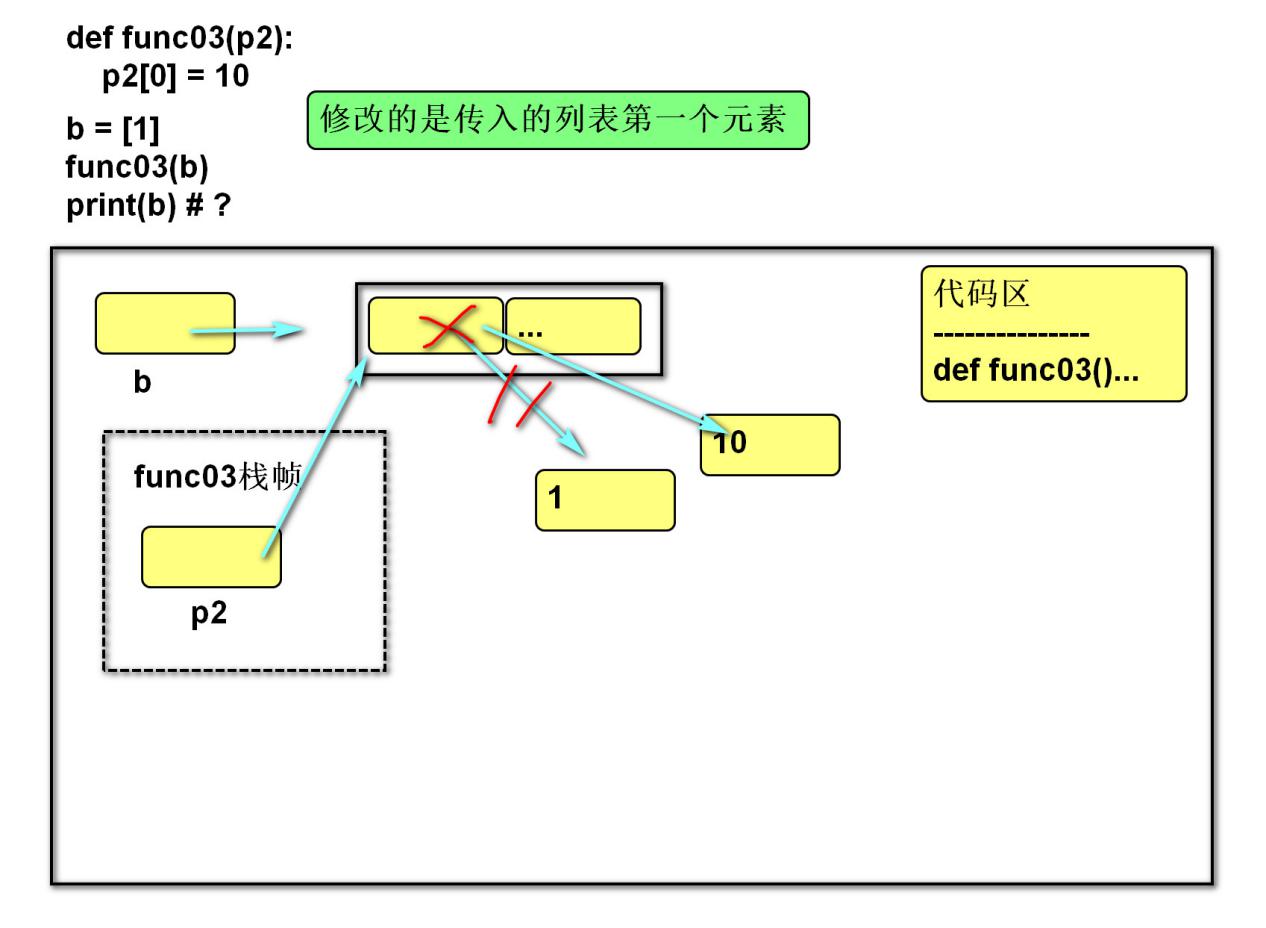
{表达式 for 变量 in 可迭代对象 if 条件}

# 函数 function





#print(a)=1



*#print(b)=[10]*

*#'''  
 函数 -- 功能  
 价值1:减少代码的重复  
 \*制作  
 def 函数名称(参数): #函数名用动词  
 """  
 文档字符串  
 """  
 函数体  
 调用  
'''  
  
# 调用(使用)函数  
print(****"你好,世界!"****)  
result = list(****"你好,世界!"****)  
print(result) # ['你', '好', ',', '世', '界', '!']  
  
# 为什么要制作功能?  
# 因为之前都是做法+使用,当做法变化时,会修改多次.****"""  
# 做法 + 使用  
print("直拳")  
print("摆拳")  
print("勾拳")  
  
# ...... 此处省略100行代码  
  
# 做法 + 使用  
print("直拳")  
print("摆拳")  
print("勾拳")  
"""****# 解决:做法1次,使用多次.当变化时只修改1次  
# 重命名快捷键:shift+f6  
# 在函数调用语句上:ctrl + 鼠标左键,可以调到函数定义的语句****def*** *single\_attack():  
 """  
 单次攻击的做法 x 1次* ***:return****:  
 """  
 print(****"直拳"****)  
 print(****"摆拳"****)  
 print(****"勾拳"****)  
  
  
# 使用 x 2次  
single\_attack()  
single\_attack()  
  
  
# 参数:调用函数时 给 制作函数时 传递的信息  
# 形式参数:表面的/虚拟的/抽象的 变量  
# 价值:让数据灵活****def*** *repeated\_attacks(count):  
 """  
 重复攻击的做法* ***:param*** *count: int类型,重复的次数* ***:return****:  
 """* ***for*** *number* ***in*** *range(count):  
 print(****"直拳"****)  
 print(****"摆拳"****)  
 print(****"勾拳"****)  
  
  
# 实际参数:真实的/确定的/具体的 数据  
repeated\_attacks(3)  
repeated\_attacks(5)  
repeated\_attacks(10)  
  
# 调试:  
# F8:逐过程,不进入函数体内部  
# F7:逐语句,进入函数体内部*

*'''  
 函数 -- 调用过程  
'''  
# ------------------定义函数------------------****def*** *repeated\_attacks(count): # -----------(2)  
 """  
 重复攻击的做法* ***:param*** *count: int类型,重复的次数* ***:return****:  
 """* ***for*** *number* ***in*** *range(count):  
 single\_attack()****def*** *single\_attack(): # -----------(3)  
 """  
 单次攻击的做法 x 1次* ***:return****:  
 """  
 print(****"临门一脚"****)  
 print(****"直拳"****)  
 print(****"摆拳"****)  
 print(****"勾拳"****)  
  
# ------------------调用函数------------------  
repeated\_attacks(3) # -----------(1)*

*"""  
 将下列代码,定义为函数.  
 for r in range(2): # 行数 0 1  
 for c in range(5): # 列数 01234 01234  
 print("老王", end=" ")  
 print() # 换行  
  
"""****def*** *print\_table(data, r\_count, c\_count):* ***for*** *r* ***in*** *range(r\_count): # 行数* ***for*** *c* ***in*** *range(c\_count): # 列数  
 print(data, end=****" "****)  
 print() # 换行  
  
print\_table(****"老崔"****, 6, 3)*

*"""  
 函数内存分布  
"""  
  
  
# 1. 将函数代码存储内存的代码区(函数体不执行).****def*** *func01():  
 a = 100  
  
  
# 2. 调用函数时,在内存中开辟一块空间(栈帧),用于存储函数内部变量.  
func01()  
  
  
# 3. 函数执行后,栈帧立即清空.****def*** *func02(p1):  
 p1 = 10  
  
  
a = 1  
func02(a)  
print(a) # ?****def*** *func03(p2):  
 p2[0] = 10  
  
  
b = [1]  
func03(b)  
print(b) # ?*

## 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。

*#"""  
 参数与返回值对比 -- 本质  
 参数：　用　　--> 做　　传递信息  
 返回值：做　　--> 　用 　传递信息  
"""  
# 1. 参数与返回值的本质就是数据给变量赋值  
# 即真实具体的数据,赋值给虚拟抽象的变量  
变量 =* ***"数据"****# 虚拟/抽象 = 真实具体  
  
# 2.****def*** *函数名(虚拟):* ***return "真实"****虚拟 = 函数名(****"真实"****)  
  
# 3.****def*** *函数名(参数):* ***return "数据"****数据 =* ***"实参"****结果 = 函数名(数据)  
  
# 4. 例如****def*** *func01(a):* ***pass****# b = "A"  
# func01(b) # 形参与实参名称可以不同,因为本质是数据A传递给变量a  
func01(****"A"****)*

## *返回值*

*#"""  
 返回值 -- 语法  
"""  
  
  
# 价值1: return 返回数据****def*** *func01():* ***return*** *100  
  
  
# 使用变量接收返回值  
result = func01()  
  
  
# 价值2: return 可以退出函数****def*** *func02():  
 print(****"func2执行喽"****)* ***return*** *100 # 退出函数  
 print(****"func2又执行喽"****)  
  
  
result = func02() # result 变量 接受到的是100  
print(result)  
  
  
# 特点:return 默认返回None  
# 函数体如果没有return,相当于返回None****def*** *func03():  
 print(****"func3执行喽"****)  
 # return # return None  
  
  
result = func03()  
print(result)*

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)

布尔值bool

None 空值

字符串str

元组tuple

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　-- 设计思想：崇尚小而精,拒绝大而全  
 函数　-- 返回值  
 def 函数名():  
 函数体  
 return 数据  
  
 结果　= 函数名()  
  
 定义函数,美元转换为人民币.  
"""  
# usd = float(input("请输入美元:"))  
# rmb = usd \* 7.0812  
# print(str(usd) + "美元 = " + str(rmb) + " 人民币")  
  
# 转换****def*** *usd\_to\_rmb(usd):  
 rmb = usd \* 7.0812  
 # 定义函数时　给　调用函数传递　结果* ***return*** *rmb  
  
usd = usd\_to\_rmb(10)  
print(usd)*

## *函数参数*

*——————————————————————————————————*

*#"""  
 函数参数  
 实际参数  
 位置实参  
 关键字实参  
 形式参数  
 默认实参  
"""****def*** *func01(p1, p2, p3):  
 print(p1)  
 print(p2)  
 print(p3)  
  
  
# 1. 位置实参: 与形参一一对应  
func01(1, 2, 3)  
# func01(1,2,3,4) # TypeError: func01() takes 3 positional arguments but 4 were given  
# func01(1, 2)# TypeError: func01() missing 1 required positional argument: 'p3'  
  
# 2. 关键字实参: 与形参按名字对应  
func01(p1=1, p2=2, p3=3)  
func01(p2=2, p1=1, p3=3) # 顺序无所谓  
  
  
# func01(p2 = 2) # 报错,缺少p1/p3  
  
# 3. 默认形参:实参可选****def*** *func02(p1=****""****, p2=0.0, p3=0):  
 print(p1)  
 print(p2)  
 print(p3)  
  
func02(p2=2)  
func02()  
func02(1, 2, 3)*

*"""  
 函数参数  
 实际参数  
 位置实参:实参与形参按照顺序对应  
 序列实参:拆后按照顺序对应  
 关键字实参:实参与形参按照名称对应  
 字典实参:拆后按照名称对应  
 形式参数  
 默认形参:实参可选  
"""****def*** *func01(p1, p2, p3):  
 print(p1)  
 print(p2)  
 print(p3)  
  
  
list01 = [1, 2, 3]  
func01(\*list01)# 序列实参:拆 后按顺序对应  
  
dict02 = {****"p1"****: 1,* ***"p2"****: 2,* ***"p3"****: 3}  
func01(\*\*dict02)# 字典实参:拆 后按名称对应*

*——————————————————————*

*"""  
 函数参数  
 实际参数:与 形参 对应  
 1.位置实参:按顺序  
 2.序列实参:拆  
 3.关键字实参:按名称  
 4.字典实参:拆  
 形式参数：约束实参传递方式  
 5.默认形参:实参可选  
 6.位置形参:实参必填  
 7.命名关键字形参:必须关键字实参  
 不定长参数  
 8.星号元组形参:位置实参数量无限　　　合  
 9.双星号字典形参:关键字实参数量无限　合  
"""  
  
  
# 1. 默认形参注意:必须从右向左依次存在****def*** *func01(p1, p2=0, p3=0):  
 print(p1)  
 print(p2)  
 print(p3)  
  
func01(1)  
  
# 2.位置形参:必填****def*** *func02(p1, p2, p3):  
 print(p1)  
 print(p2)  
 print(p3)  
  
  
# 3.星号元组形参：将实参合并为一个元组（只适用于位置实参）  
# 价值：位置实参数量可以无限****def*** *func03(\*args):  
 print(args)  
  
  
func03(1, 2, 3) # 函数内部得到的是一个元组 (1,2,3)  
func03() # 函数内部得到的是一个空元组 ()  
  
  
# func03(a=1,b=2)#TypeError: func03() got an unexpected keyword argument 'a'  
  
# 4. 命名关键字形参:强调必须是关键字实参  
# -- 星号元组形参后面的参数 a, b, c****def*** *func04(\*args, a, b, c):  
 print(args)  
 print(a)  
 print(b)  
 print(c)  
  
  
func04(1, 2, 3, 4, a=5, b=6, c=7)  
  
  
# -- 星号后面的参数 a, b, c  
# 第一个参数是必填(重要),后面两个参数是可选(不重要)****def*** *func05(a, \*, b=0, c=0):  
 print(a)  
 print(b)  
 print(c)  
  
  
func05(5)  
# func05(5,6,7)　　  
func05(5, b=6)  
func05(5, c=7)  
func05(5, b=6, c=7)  
  
  
# 5.　双星号字典形参****def*** *func06(\*\*kwargs):  
 print(kwargs)  
  
  
func06() # 函数内部得到的是空字典  
func06(a=1, b=2) # 函数内部得到的是{"a":1,"b":2}  
# func06(1,2,3,4)# TypeError: func06() takes 0 positional arguments but 4 were given*

### 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

# 作用域LEGB

*#"""  
 作用域：变量起作用的范围。  
 适用性:  
 全局变量:在大范围(多个函数)内操作数据  
 局部变量:在小范围(一个函数)内操作数据  
"""  
# -----------------全局变量:数据------------------  
# 2. 全局变量:文件内部(函数外部)创建,适用于整个文件.  
g01 =* ***"a"****# -----------------函数:行为------------------****def*** *func01():  
 # 1. 局部变量: 函数内部创建的变量,只适用于函数内部.  
 a = 10  
 print(g01) # 在局部作用域内,读取全局变量  
# -----------------入口:调用------------------  
func01()*

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

*#"""  
 global  
"""  
g01 =* ***"a"  
  
  
def*** *func01():  
 # g01 = "b" # 创建局部变量,没有修改全局变量  
 # 局部作用域 修改 全局变量* ***global*** *g01 # 必须先声明全局变量  
 g01 =* ***"b"****func01()  
  
print(g01) # ?*

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