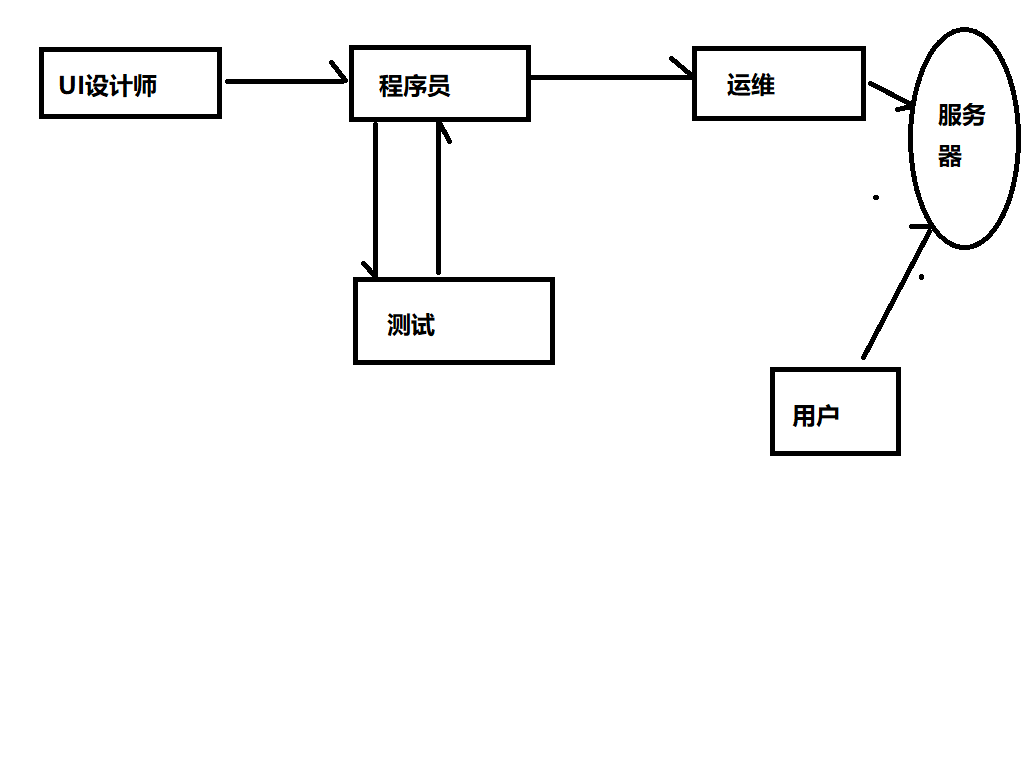
# Python 简介

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：

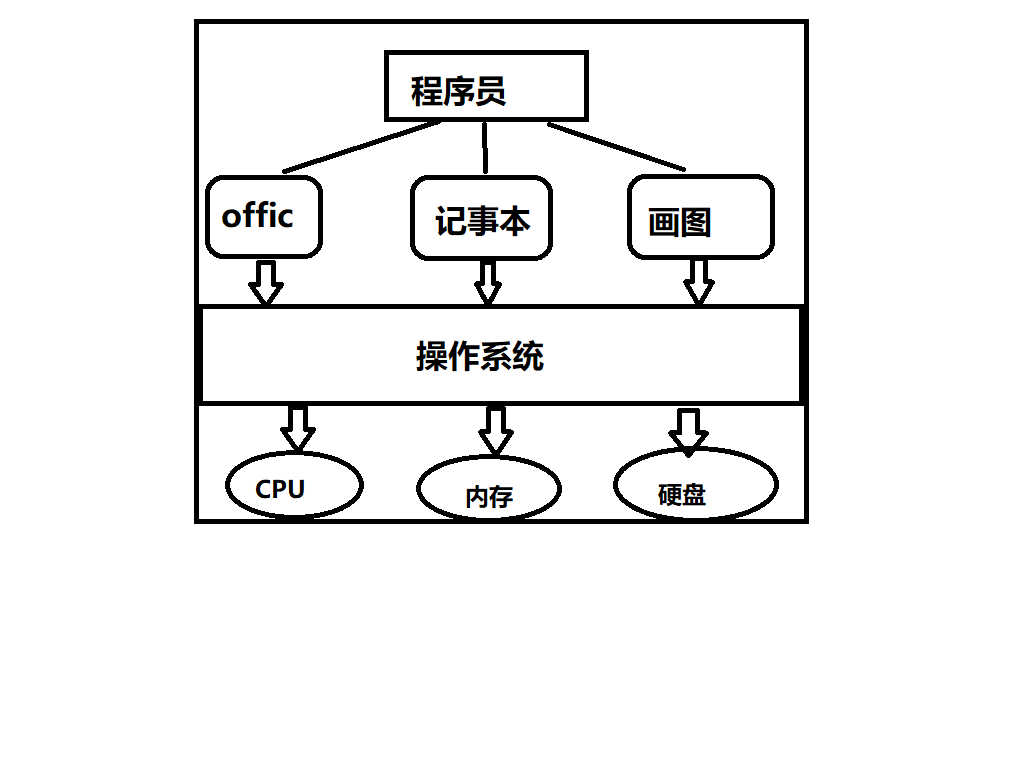
主板--计算机的主要电路系统。

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。

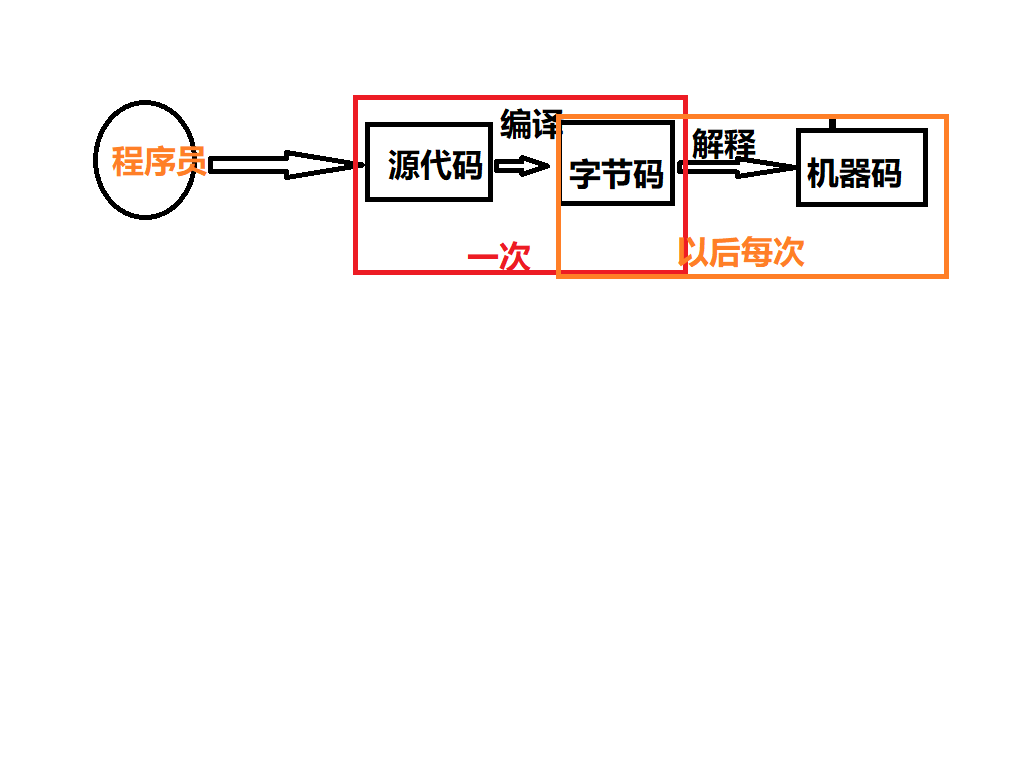
1. 解释：在程序运行之时，通过解释器对程序逐行翻译，然后执行。例如Javascript

-- 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)

# 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input输出格式为str

## 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。

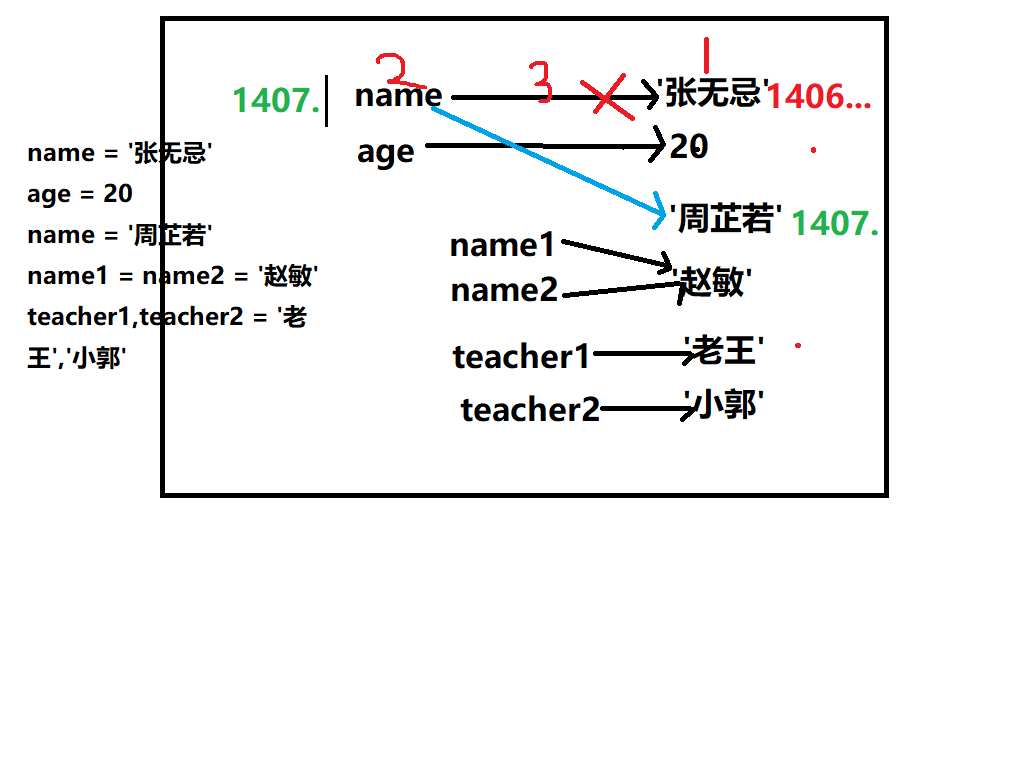
1. 建议命名：字母小写，多个单词以下划线隔开。

class\_name

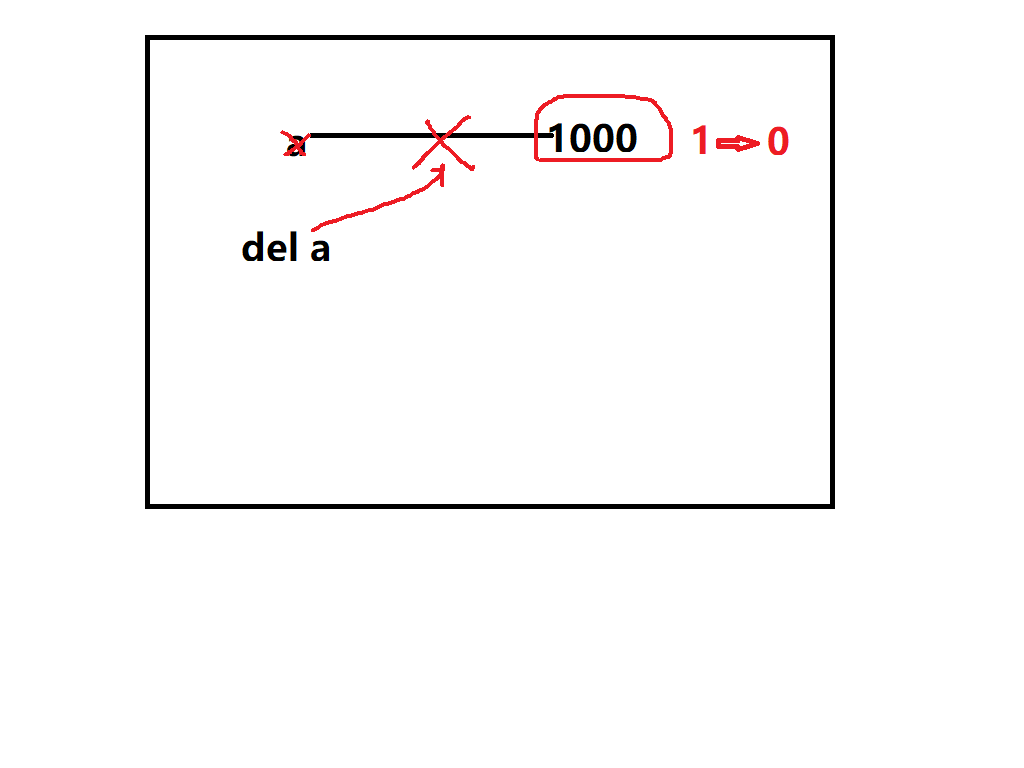
1. 赋值：创建一个变量或改变一个变量关联的数据。
2. 语法：变量名 = 数据

变量名1 = 变量名2 = 数据

变量名1, 变量名2, = 数据1, 数据2 (#对应变量赋值)



## 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 #单引号，三引号

### 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) bool(***''*** )

#数字，数字，单词，一对单引号

1. 混合类型自动升级：（整数和浮点数运算，结果自动转换为浮点数）

1 + 2.14 返回的结果是 3.14

1 + 3.0 返回结果是: 4.0

（任意数据类型转换为布尔类型时，如果有值就为True ,没有值就为False）

## 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

字符串比较(单词的首字母)是给ascii值依次比较,

逐个比较两个串中对应的字符，字符大小按照ASCII码值确定，从左向右比较，如果遇到不同字符，所遇第一对不同字符的大小关系就确定了两个字符串的大小关系，如果未遇到不同字符而某个字符串首先结束

在ascii值大就大如’A’>’B’#True

ascii: 0 (48) A(65) a(97)

### 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

优先级：not > and> or

#### 短路运算

一但结果确定，后面的语句将不再执行。

（#逻辑运算时，如果前面的条件能判断出最终结论，后续条件不在判断）

（#因此，在编写代码时，尽量将复杂的判断放在后面）

### 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

（#id函数 获取变量存储对象的地址；is运算本质是通过id函数进行判断）

### 优先级

高到低：

算数运算符

比较运算符

快捷运算符

身份运算符

逻辑运算符

# 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。 （#占位）

## 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: （#1.选择语句执行过程中，只会选择一个符合条件的代码块执行，其余代码块都不会被解释器执行；2.严格执行缩进规则，4个空格或一个TAB键。）

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。

（#调试：1.让程序中断，逐行执行；2.目的是审查程序运行时的变量和变量取值，审查程序运行流程。

#步骤：1.加断点（调试过程中遇到断点就停止程序）可能会出问题/你想详细了解的地方加断点。

2.运行程序Shift+F9。

3.程序会在断点处停止，按F8执行一行。

4.按Ctrl+F2停止调试。

### 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。 (#字符串是可迭代对象)

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: (#变量列表 存储的是每个字符的地址)

语句块1 （#嵌套循环）

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 （#即range函数默认为递增）

## 跳转语句

### break 语句

1. 跳出循环体，后面的代码不再执行。 （#注意缩进规则）
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。 （#次数未知用while循环

#次数已知/有确切数据用for循环）

# 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： （#切片方式参考range函数）

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

#（print(str[:])*#复制字符串*

print(str[::-1])*#反转字符串*

*str[:] = []#清空字符串）*

*列表切片:步长为:-1,表示从右向左取值*

*str01 =* ***'welcome to beijng' (0-16)(-1---17)***

str02 = str01[-11::-1] +str01[-7:-11:-1] +str01[:-8:-1]

print(str02)

#在str01[-11::-1] 中间项可以为小于-17的数,也可以不填,

在str01[-7:-11:-1] 中也可以填[10:6:-1]

在str01[:-8:-1] 中也可以填[16:10:-1]

str02 = str01[-11::-1] +str01[10:6:-1] +str01[16:10:-1]

print(str02)*#emoclew ot gnjieb*

str01 = str01[::-1]*#gnjieb ot emoclew*

str01 = str01.split()*#['gnjieb', 'ot', 'emoclew']*

str01 = str01[::-1]*#['emoclew', 'ot', 'gnjieb']*

str01 = **' '**.join(str01)*#emoclew ot gnjieb*

print(str01)

### 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) #*字符串里面的字符不能直接相加*

## 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 （#所见即所得）

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 （#打印单引号，双引号，一对双引号，换行符，反斜杠，制表符（tab键），’0’或者为空字符）

1. 原始字符串：取消转义。

a = r”C:\newfile\test.py” （在字符串前加r,取消转义）

#### 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) (#%s转成字符串,数字.字母均可,但%d%f不能表示字符串 )

"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 （#占位符，%.2f保留两位小数）

## 列表 list

### 定义

由一系列变量组成的可变序列容器.

(#预留空间存储机制) 一串连续的数据

### 基础操作

1. 创建列表：

列表名 = []

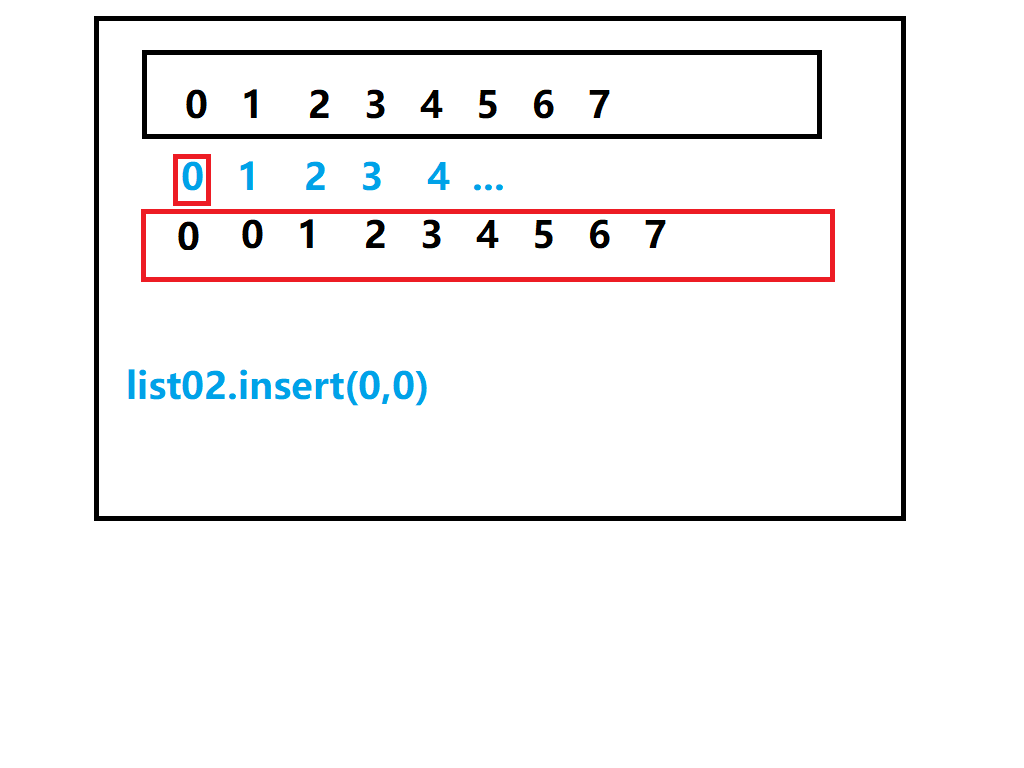
列表名 = list(可迭代对象)

1. 添加元素：

列表名.append(元素) #在列表最后添加一个对象,元素或列表等)

当 list 类型的对象进行 append 操作时，实际上追加的是该对象的引用。

列表.insert(索引，元素) (*#extend()使用可迭代对象扩展列表(向列表末尾添加一系列元素,相当于合并 )*

*(#创建列表或者说是类型转换为列表)*

*list02 = list(****"我是小妖怪"****)*

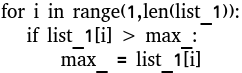
*list02 = list(range(4)))*

>>[0, 1, 2, 3]

['我', '是', '小', '妖', '怪']

1. 定位元素：

索引、切片 (#通过索引变成效率高,尽量不要切片)

1. 遍历列表：

正向：

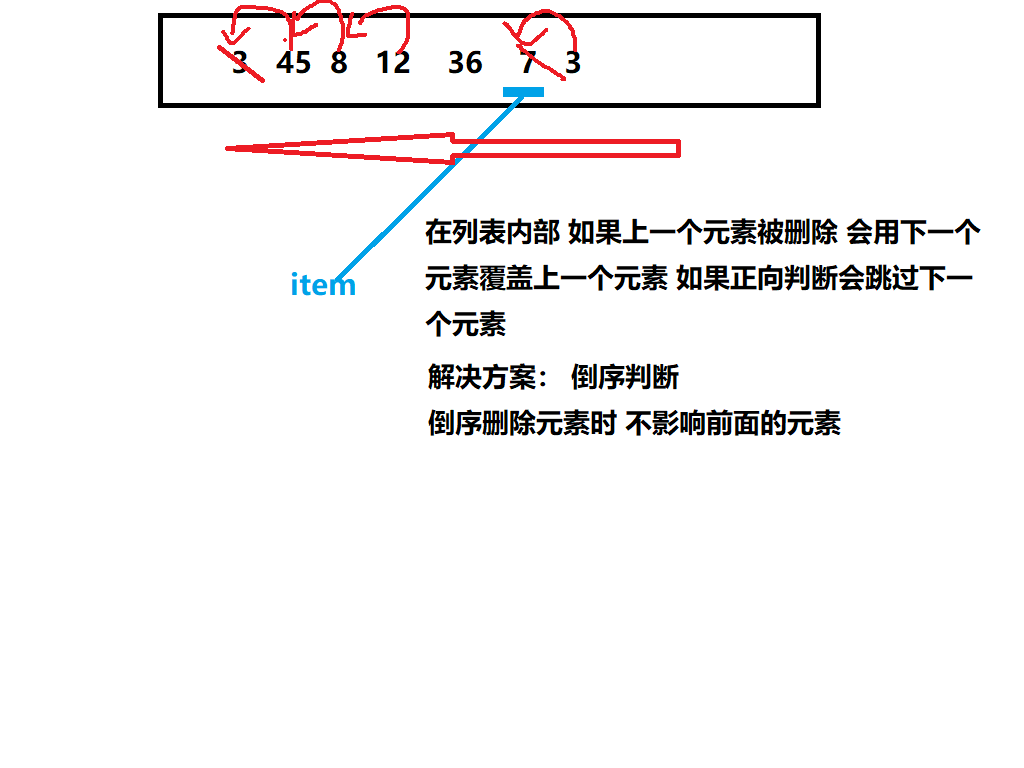
for 变量名 in 列表名:

变量名就是元素

反向：

for 索引名 in range(len(列表名)-1,-1,-1): #最好有这种方法效率高(索引)

列表名[索引名]就是元素 (L.clear() | 清空列表,等同于 L[:] = [])

(列表内存放的是下表,并不是元素本身)

1. 删除元素：

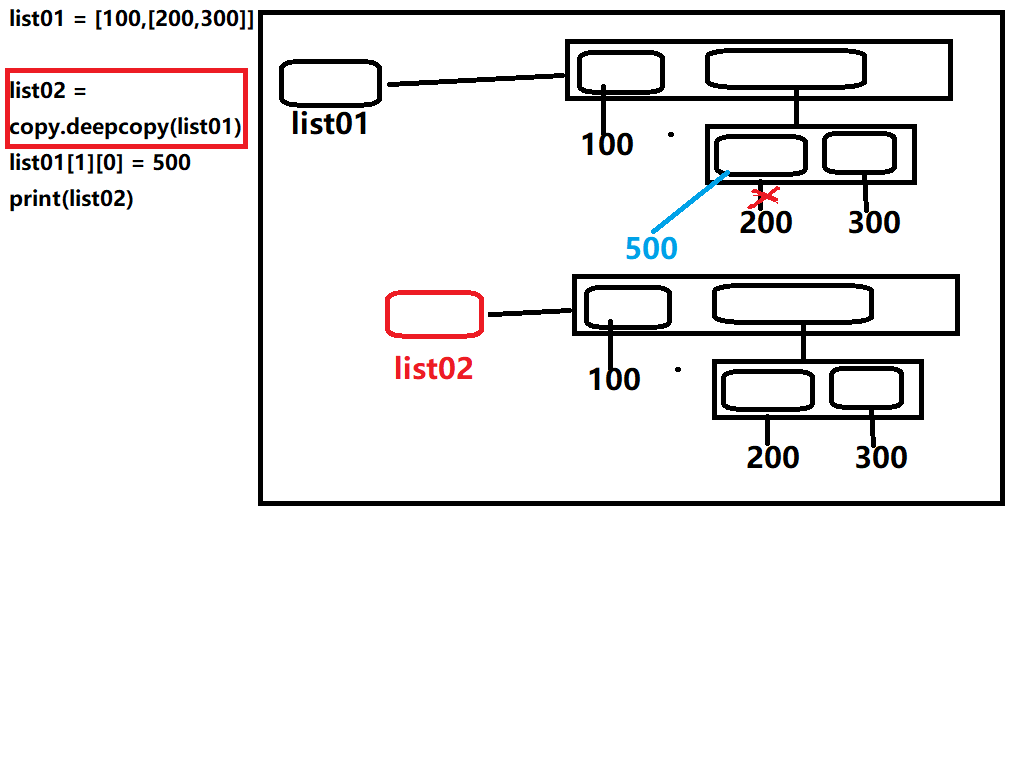
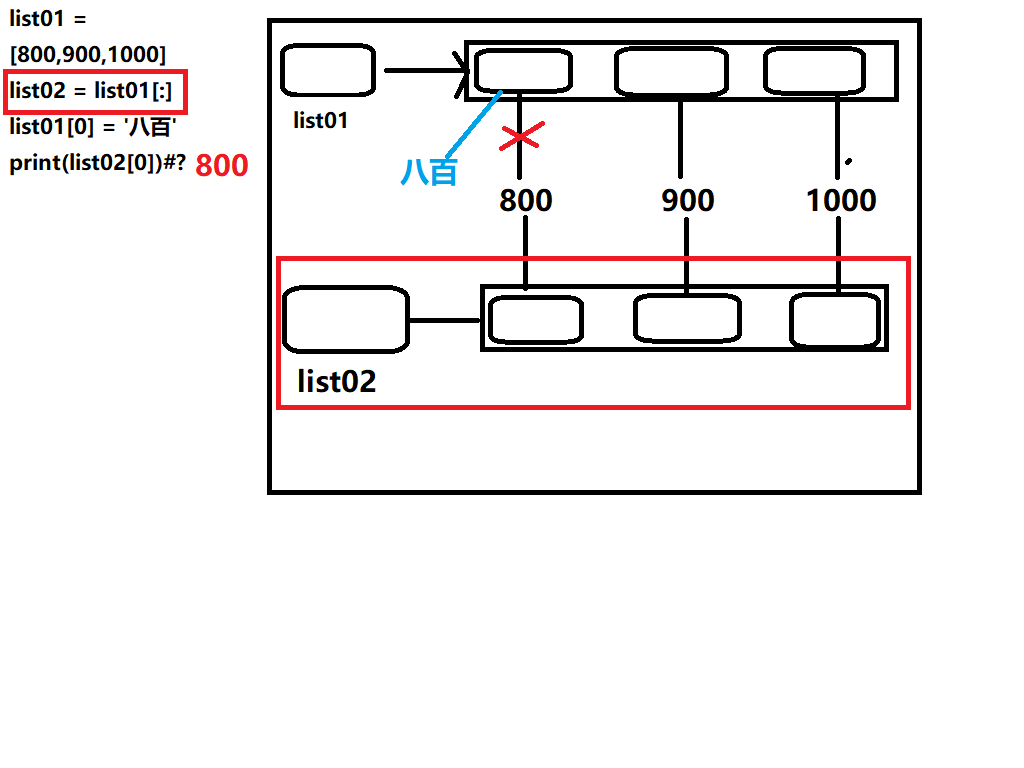
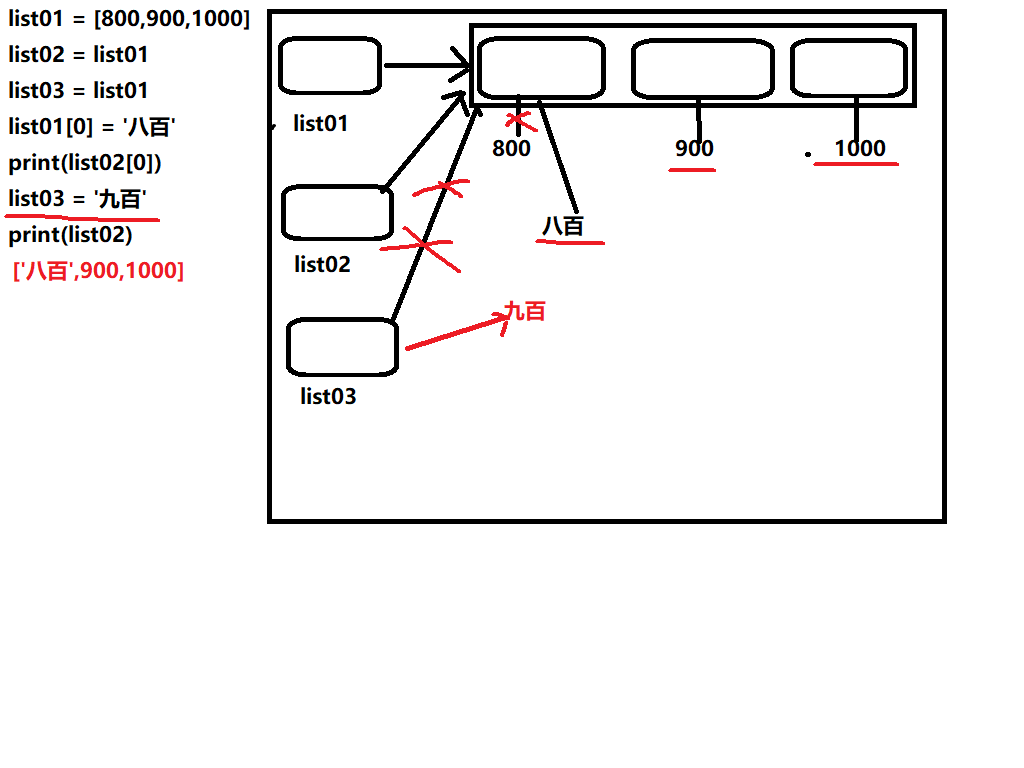
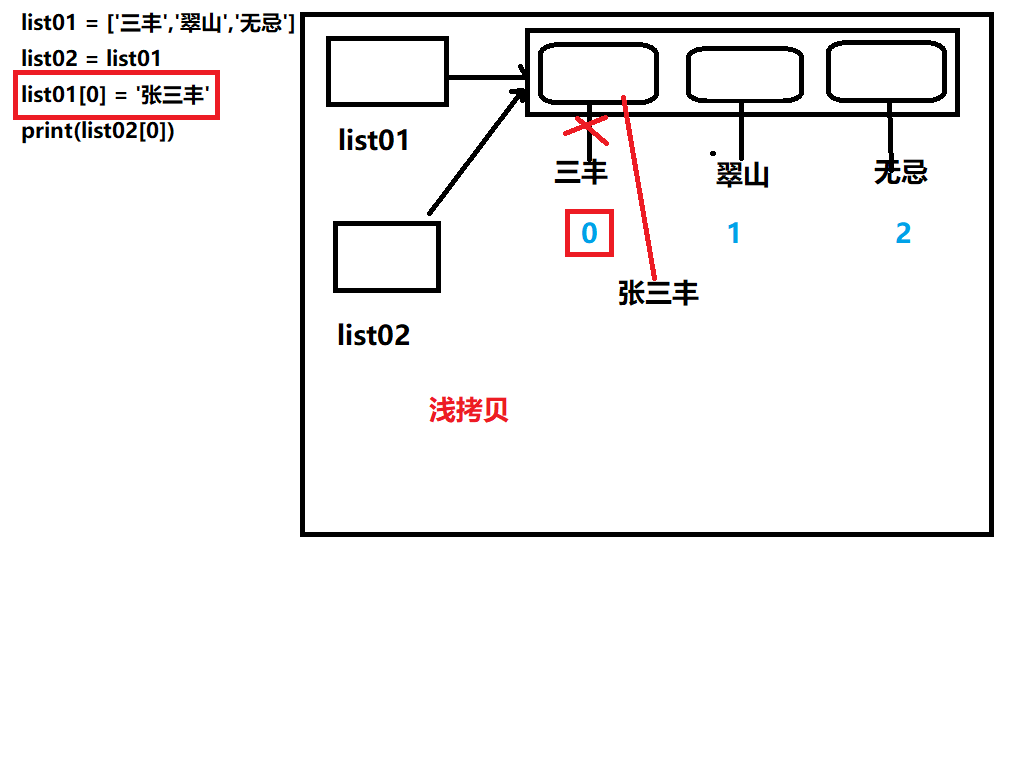
列表名.remove(元素)

del 列表名[索引或**切片**]

### 深拷贝和浅拷贝

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。(共享对象引用)

深拷贝：复制整个依懒的变量。 (#深拷贝可能会占用大量内存,拷贝前后对象不影响)



### 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”) #a-b-c-d就是字符串

(#:通过可变对象list保存所有要拼接的**字符串(必须是字符串)**,最后再将列表中的内容转为字符串)

### 列表推导式

1. 定义： 相当于append()方法

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] (#先判断条件)

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。

>>>vec1=[2,4,6] >>>vec2=[4,3,-9]  
>>>[x\*y for x in vec1 for y in vec2] >>[8,6,-18,16,12,-36,24,18,-54]  
>>>[vec1[i]\*vec2[i] for I in range(len(vec1))] >>[8,12,-54]

### 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]

list1 =[]

**for** i **in** range(2):

list1.append([])

**for** j **in** range(1,4):

list1[i].append(j+i\*3)

print(list1)*#[[1, 2, 3], [4, 5, 6]]*

*#列表嵌套式:*

print([[j+i\*3 **for** j **in** range(1,4)] **for** i **in** range(2)])

#行列转换:list\_1 = []

**for** i **in** range(len(game\_map[0])):

list\_1.append([])

**for** row **in** game\_map:

list\_1[i].append(row[i])

print(list\_1)

方法二:将上述转化为列表嵌套式

list\_1 = []

**for** i **in** range(len(game\_map[0]):

list\_1.append([row[i] **for** row **in** game\_map])

print(list\_1)

转换为列表嵌套式:print([[row[i] **for** row **in** game\_map]**for** i **in** range(len(game\_map[0])])

## 元组 tuple

### 定义

1. 由一系列变量组成的不可变序列容器。 (#按需分配的存储机制,元组与列表可相互转换)
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,) (#只有一个元素时不加逗号,则为int类型)

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： (#元组不可修改元素)

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):

元组名[索引名]就是元素

(#元组中嵌套的列表可以修改)

### 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组："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。 (#内部元素没有顺序)
3. 键必须惟一且不可变(字符串/数字/元组)，值没有限制。

*#可变类型无法通过哈希算法运算#所以字典的键都是不可变类型*

### 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) *dict02 = dict([['name','shibw']]) #必须在容器里面*

#外层列表表示可迭代对象,里层列表创建为字典的键与值*#单独的name 无法创建字典 需要指定键值对*

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]

(#存储多个数据时,使用什么样的数据结构? 根据需求 结合本身优缺点 综合考虑

列表:

优点:添加/获取元素时比较灵活,可以使用索引/切片操作

缺点:数据量大时,查找慢,通过索引获取值时,如果信息多,代码可读差

字典:

优点:根据key获取value 速度快,如果获取数据较多时,通过键获取值代码可读性好)

缺点:不能使用索引和切片操作元素 不灵活,占内存较大

### 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**可变散列**容器。(#内部元素没有顺序)
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} (#不能包含**重复**字符,内部元素输出时不一定按输入顺序)

集合名 = set(可迭代对象)

1. 添加元素：

集合名.add(元素) (#只能添加一个元素)*（字符串、元组、数字）*

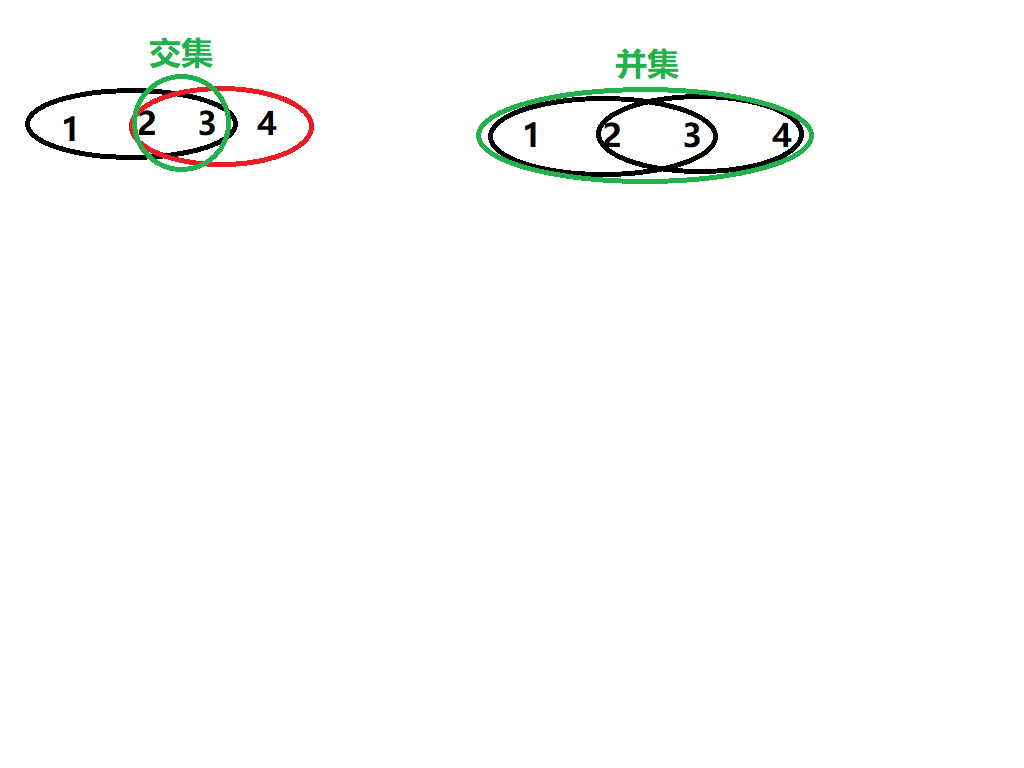
*#列表是可变类型 不能放入集合*

1. 删除元素：

集合名.discard(元素)(#删除不存在元素不会报错

集合名.remove(元素) #删除不存在元素会报错)

### 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 (# | 的快捷键shif+ \)

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。 (#主要作用:与其他容器对象转换)

#### 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

(#固定集合不能添加\修改\删除,其余运算同set)

**总结:**

*容器对象*

*#字符串* *str()*

*#只能装字符 不可变序列*

*#列表* *list()*

*#任意数据类型 可变序列*

*#元组* *tuple()*

*#任意数据类型 不可变序列*

*#序列的通用操作*

*# + 拼接两个序列*

*# \* 生成重复的序列*

*# += \*=*

*# 索引* */ 切片*

*#字典* *dict()*

*#键必须是不可变类型 值可以为任意对象*

*#可变散列对象*

*#集合* *set()*

*#只能包含不可变类型 可变散列对象*

*#固定集合* *frozenset() 不可变散列对象*

*# None*

*# int float bool complex str list tuple dict set*

*#可变* *list dict set*

*#不可变 数字* *str tuple*

*# and 一假俱假*

*# or 一真俱真*

*#短路逻辑*

*# not*

*# 语句*

*# if*

*# while*

*# for*

***# 内建函数所有容器类型都可以使用,***

*# 对于字典来说,只判断键的大小 长度 和值*

# 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。

(#函数名保存的是对应功能的代码地址)

## 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。(#调用函数时,开辟栈帧,调用结束,栈帧销毁)

## 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  (#相当于返回结果,退出函数,是否需要return看用户需求)

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。

*#总结*

*#如果向函数传递一个可变的类型(列表)*

*#如果函数内部修改可变对象*

*#函数执行后，修改原列表的值*

*#可以不用返回值也能拿到结果 列表.insert()*

## 函数参数

### 实参传递方式argument 4种

#### 位置传参

定义：实参与形参的位置依次对应。*#如果实参位置发生改变 会影响函数结果*

##### 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。*#序列 字符串 列表 元组*

*list01 = [10,20,30]*

fun01(\*list01)

#### 关键字传参

定义：实参根据形参的名字进行对应。 *#使用关键字实参 传参的顺序可以不固定*

##### 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。

(#位置传参和关键字传参混合使用时,先写位置参数后写关键字参数)

dict01 = {**'a'**:10,**'b'**:20,**'c'**:30}

fun01(\*\*dict01)

### 形参定义方式parameter 5种

#### 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。

(#默认参数,就是缺省参数,函数的缺省参数,将常见的值设置为参数的缺省值

#### 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。*#接受不定数量的实参 让实参的数量可以无限多*

1. 说明：

一般命名为'args'

形参列表中最多只能有一个

**def** fun02(p1,p2,\*args):

print(p1)

print(p2)

print(\*args)

fun02(1,2,3,4,5,6,7,8,9,10)

→

1

2

3 4 5 6 7 8 9 10

#### 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：*#星号以后的位置形参是命名关键字形参*

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 *双星号字典形参 让关键字实参的数量无限*,*将关键字实参的变量名作为字典的键 值做为字典中对应键的值保存*

1. 说明:

一般命名为'kwargs'

形参列表中最多只能有一个

**def** fun05(\*\*kwargs):

print(kwargs)

fun05(a=10,b=20,c=30)

-->

{'a': 10, 'b': 20, 'c': 30}

#### 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

*# 混合使用*

*#语法 位置形参要在关键字形参前面定义*

*#只要出现关键字形参 后面的参数也要带有关键字*

# 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。*只能在函数内部使用*,*当函数调用时创建局部变量 函数结束后销毁*
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。 *全局变量 在文件内都可以访问到*
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。

*locals()*函数*收集当前的局部变量 保存到字典*

***return*** *locals()*

**def** user\_info():

uesr\_name = **'shibw'**

user\_age = 20

user\_email = **'shibw@tedu.cn'**

gender = **'男'**

address = **'北京市东城区'** *#locals() 收集当前的局部变量 保存到字典*

**return** locals()

*#获取到返回的元组*

print(user\_info())

inofs = user\_info()

*#输出详细信息*

print(**'姓名：%s,年龄：%s,邮箱：%s,性别：%s,地址：%s'** % (inofs[**'uesr\_name'**],inofs[**'user\_age'**],inofs[**'user\_email'**],inofs[**'gender'**],inofs[**'address'**]) )

## 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

**def** f1():

v = 200

**def** f2():

v = 300

**def** f3():

**nonlocal** v #在内层函数修改外层嵌套函数内的变量

v = 400

f3()

print(**'f2.v= '**,v )

f2()

print(**'f1.v='**,v)

f1()

--→

f2.v= 400

f1.v= 200