

# 网上python语法总结，推荐详读

- chap1 输出函数

- 输出函数 print

- 内容

- 可以输出数字 `print(520)`

- 可以输出字符串 `print('helloworld')`

加引号是为了直接输出，因为 `helloworld` 不是表达式

- 可以输出含有运算符的表达式 `print(3+1)`，输出 4

3,1 是操作数，+ 是运算符，进行运算，输出结果

- 输出的目的地

- 显示器

界面下面

- 文件

例将数据输出到文件当中

- `a+` 表示如果没有这个文件就新建，如果有就在文件后面继续追加

```
fp=open('E:/text.txt','a+')
print('helloworld',file=fp)
fp.close()
```

注意，1.所指定的盘符存在 2.使用 `file=**` 的形式，要不然数据进不去 3.不进行换行输出（即输出内容在一行当中）那要写 `print('hello','world','Python')`，即用逗号分隔

- 转义字符

- 概念

- 反斜杠+想要实现的转义功能首字母

为什么需要转义字符？

- 当字符串中包含反斜杠、单引号和双引号等特殊用途的字符时，必须使用反斜杠对这些字符进行转义（转换一个含义）

反斜杠 `\`  
单引号 `'`  
双引号 `"`

- 当字符串中包含换行、回车、水平制表符或退格等无法直接表示的特殊字符时，也可以使用转义字符当字符串中包含换行、回车、水平制表符或退格等无法直接表示的特殊字符时，也可以使用转义字符

换行 `\n`  
回车 `\r`  
水平制表符 `\t`  
退格 `\b`

- chap2 数据与变量

- 二进制与字符编码

- 8bit→1byte，1024byte→1KB，1024KB→1MB，1024MB→1GB，1024GB→1TB

8 个位置→1 个字节，KB 是千字节，MB 是兆字节，GB 是吉字节，

- python 中的标识符和保留字

- 保留字

- 有一些单词被赋予了特定的意义，给对象起名字时不能用
    - 'False', 'None', 'True', '\_\_peg\_parser\_\_', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

- 标识符

- 变量、函数、类、模块和其他对象起的名字就叫标识符
    - 规则
      - 字母、数字、下划线
      - 不能以数字开头
      - 不能是保留字
      - 严格区分大小写

- 变量的定义和使用

- 变量就像是一个带标签的盒子，把需要的数据放进去

- name=马丽亚

name 是变量名，马丽亚是值

- 变量由标识、**类型**、值组成

- **数据类型**

- 常见的数据类型

- 整数类型--int

- 可以表示正数、负数和零
      - 不同进制表达方式
        - 十进制--默认
        - 二进制--0b 开头
        - 八进制--0o 开头
        - 十六进制--0x 开头

- 浮点数类型--float

•

- 布尔类型--bool

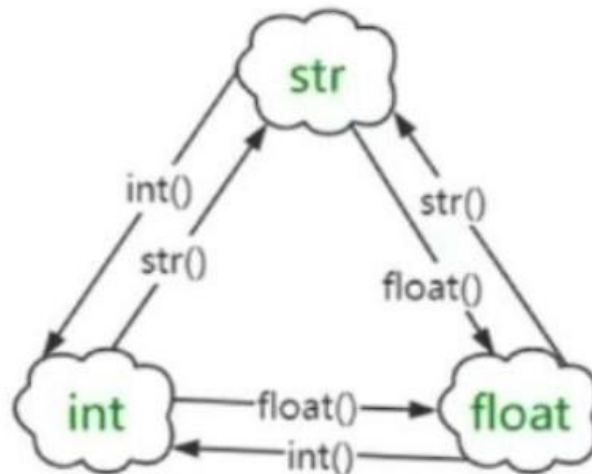
只可以取两个值：true,false

- 字符串类型--str

不管什么，只要加上单引号或双引号或三引号就是字符串类型

- 数据类型转换

- 



- chap3 输入函数

- 输入函数 input

- 作用：接受来自用户的输入
- 返回值是 str 类型
- 使用=对输入的值进行存储，存储到变量中

- python 中常见的运算符

- 算术运算符

- 标准算术运算符

加，减，乘，除，整除

- 取余运算符

%

- 幂运算符

\*\*

- 赋值运算符

- 执行顺序：从右到左
- 支持链式赋值
- 支持参数赋值
- 支持系列解包赋值
- 比较运算符
  - 对变量或表达式的结果进行大小、真假等比较
  - $>, <, >=, <=, !=$
  - $==$   
比较 value
  - $is, is\ not$   
比较 id

- 布尔运算符
  - 对于布尔值之间的运算
  - $and, or, not, in, not\ in$

- 位运算符



超纲，不用看

- 将数据转成二进制进行计算
- 按位于&

0	0	0	0	0	1	0	0	4	4&8
0	0	0	0	1	0	0	0	8	
0	0	0	0	0	0	0	0	结果为0	

二进制中同为 1 时结果为 1，其余为 0

- 按位或|

0	0	0	0	0	1	0	0	4	4 8
0	0	0	0	1	0	0	0	8	
0	0	0	0	1	1	0	0	结果为12	

二进制中同为 0 时结果为 0，其余为 1

- 左移位运算符<<

高位溢出	0	0	0	0	0	1	0	0	4	左移位	
0	0	0	0	0	1	0	0	0	8	一位，相当于乘以2	
高位溢出								低位补0			
0	0	0	0	1	0	0	0	0	16		
								低位补0			

高位溢出舍弃，低位补 0。向左移动一位，相当于乘 2

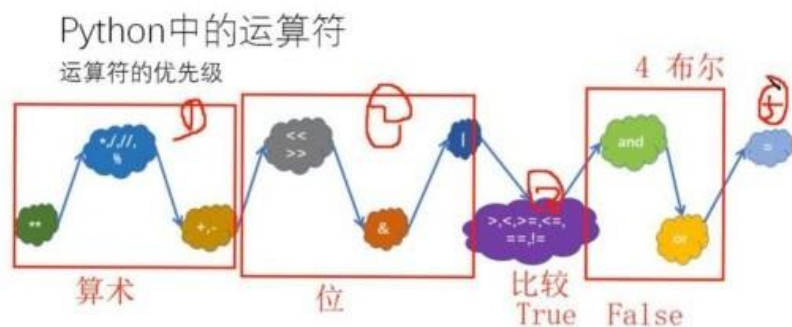
### • 右移位运算符 >>

0	0	0	0	0	1	0	0	4	右移位	
0	0	0	0	0	0	1	0	0	2	
高位补0								低位截断		
0	0	0	0	0	0	0	1	0	1	
高位补0								低位截断		

低位溢出舍弃，高位补 0。向右移动一位，相当于除以 2

### • 运算符中的优先级

#### • 有括号先算括号



### • chap4 程序的组织结构

#### • 程序的组织结构

##### • 顺序结构

- 程序从上到下顺序地执行代码，中间没有任何的判断和跳转，直到程序结束

```

----程序开始----
1. 把冰箱门打开
2. 把大象放进去
3. 把冰箱门关上
----程序结束----

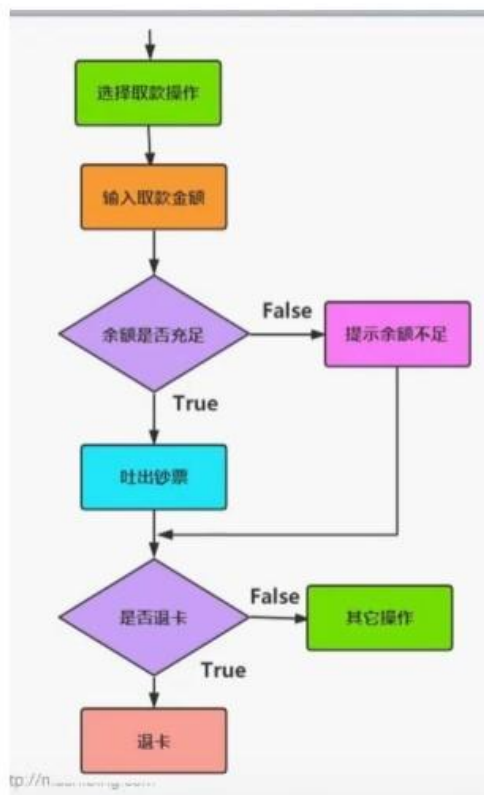
```

##### • 选择结构

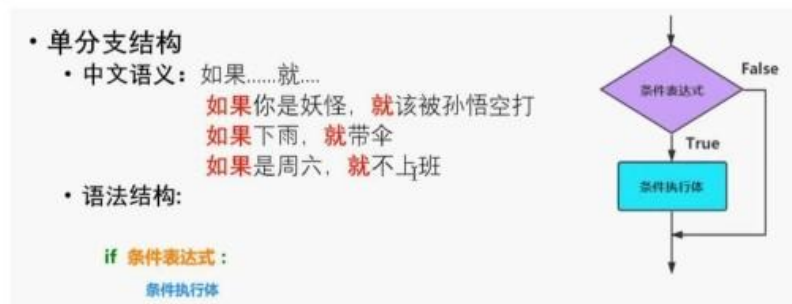
if 语句

- python 中一切皆对象，所有对象都有一个布尔值，可以使用 bool() 来获取对象的布尔值。其中布尔值为 False 的是

- False
- 数值 0
- None
- 空字符串
- 空列表
- 空元祖
- 空字典
- 空集合
- 选择结构--程序根据判断条件的布尔值选择性地执行部分代码



- 明确的让计算机知道在什么条件下该去做什么
- 单分支结构
  -





- 双分支结构

- 

- 双分支结构

- 中文语义：如果.....不满足.....就.....

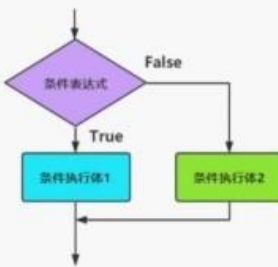
如果中奖就领奖，没中奖就不领

如果是妖怪就打，不是就不打

如果是周末不上班，不是就上班

- 语法结构:

```
if 条件表达式 :  
    条件执行体1  
else:  
    条件执行体2
```



- 多分支结构

- 解决连续的区间段的问题

成绩在90分以上吗？不是

成绩是80到90分之间吗？不是

成绩是70到80分之间吗？不是

成绩是60到70分之间吗？不是

成绩是60分以下吗？是

- 多分支结构

- 中文语义:

成绩在90分以上吗？不是

成绩是80到90分之间吗？不是

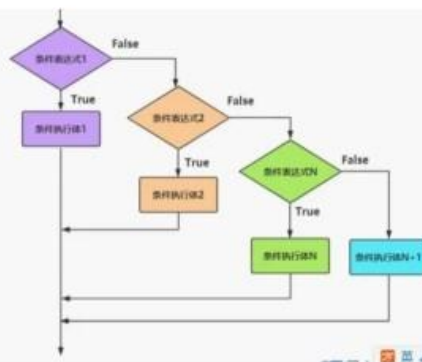
成绩是70到80分之间吗？不是

成绩是60到70分之间吗？不是

成绩是60分以下吗？是

- 语法结构:

```
if 条件表达式1 :  
    条件执行体1  
elif 条件表达式2 :  
    条件执行体2  
elif 条件表达式N :  
    条件执行体N  
[else:]  
    条件执行体N+1
```



- 嵌套 if

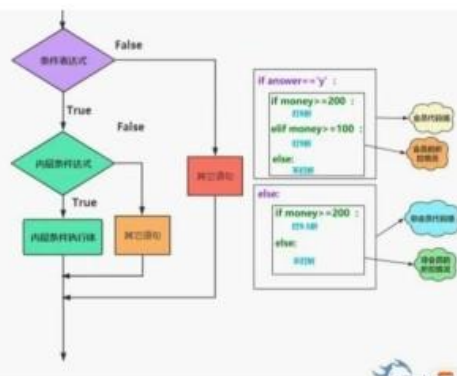
- 

嵌套if

- 嵌套if

- 语法结构:

```
if 条件表达式1:  
    if 内层条件表达式:  
        内存条件执行体1  
    else:  
        内存条件执行体2  
else:  
    条件执行体
```



- 条件表达式

- 

- 条件表达式

- 条件表达式是if.....else的简写

- 语法结构:

x if 判断条件 else y

- 运算规则

如果判断条件的布尔值为True，条件表达式的返回值为x，否则条件表达式的返回值为False

- pass 语句

- **pass语句**

- 语句什么都不做，只是一个占位符，用在语法上需要语句的地方

- **什么时候使用:**

先搭建语法结构，还没想好代码怎么写的时候

- **哪些语句一起使用**

- if语句的条件执行体
- for-in语句的循环体
- 定义函数时的函数体

- **循环结构**

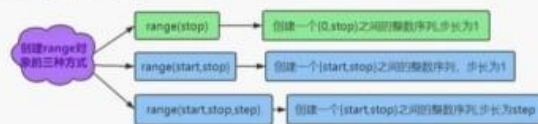
while 语句，for-in 语句

- **chap5 循环**

- **内置函数 range**

- **range()函数**

- 用于生成一个整数序列
- 创建range对象的三种方式



- 返回值是一个迭代器对象
- range类型的优点: 不管range对象表示的整数序列有多长，所有range对象占用的内存空间都是相同的，因为只需要存储start, stop和step。只有当用到range对象时，才会去计算序列中的相关元素
- in与not in 判断整数序列中是否存在（不存在）指定的整数

- **循环结构**

- 反复做同一件事情的情况，称为循环

- 循环结构的流程图

- **循环的分类**

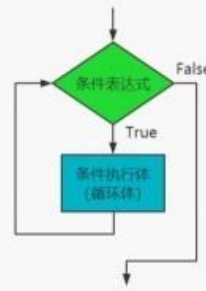
- while
- for -in

- **语法结构**

**while** 条件表达式:  
条件执行体(循环体)

- **选择结构的if与循环结构while的区别**

- if是判断一次，条件为True执行一行
- while是判断N+1次，条件为True执行N次

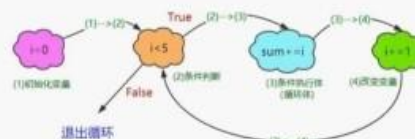


- **while 循环的执行过程**

- **四步循环法**

- 初始化变量
- 条件判断
- 条件执行体（循环体）
- i改变变量

- **while循环的执行流程**



用于次数不固定的循环



- for in 循环

- for-in循环
  - in表达从（字符串、序列等）中依次取值，又称为遍历
  - for-in遍历的对象必须是可迭代对象
- for-in的语法结构  
`for` 自定义的变量 `in` 可迭代对象:  
循环体
- for-in的执行图
- 循环体内不需要访问自定义变量，可以将自定义变量替代为下划线

```
graph TD; A{可迭代对象还有元素吗?} -- 有 --> B[自定义变量自动被赋予当前迭代对象的元素值]; B --> C[循环体]; C --> A; A -- 没有 --> D[ ];
```

用于次数固定的循环

- 可迭代对象（目前）：字符串，序列

- 流程控制语句 break

- break语句
  - 用于结束循环结构，通常与分支结构if一起使用

```
graph LR; subgraph For_Loop [for ... in ...]; direction TB; F1[for ... in ...]; F2[.....]; F3[if ...]; F4[break]; end; subgraph While_Loop [while (条件)]; direction TB; W1[while (条件)]; W2[.....]; W3[if ...]; W4[break]; end;
```

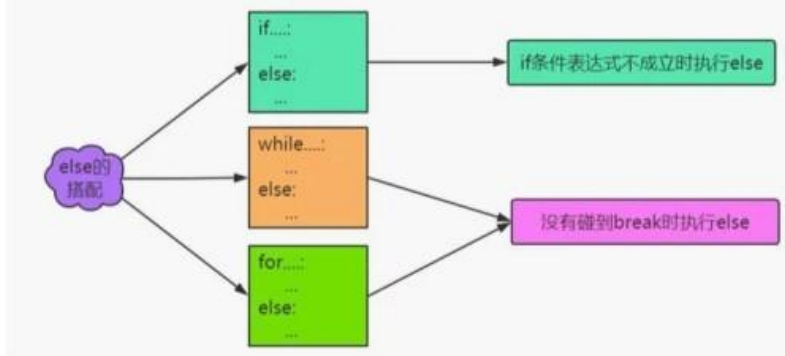
- 流程控制语句 continue

- continue语句
  - 用于结束当前循环，进入下一次循环,通常与分支结构中的if一起使用

```
graph LR; subgraph For_Loop [for ... in ...]; direction TB; F1[for ... in ...]; F2[.....]; F3[if ...]; F4[continue]; F5[.....]; end; subgraph While_Loop [while (条件)]; direction TB; W1[while (条件)]; W2[.....]; W3[if ...]; W4[continue]; W5[.....]; end;
```

- else 语句

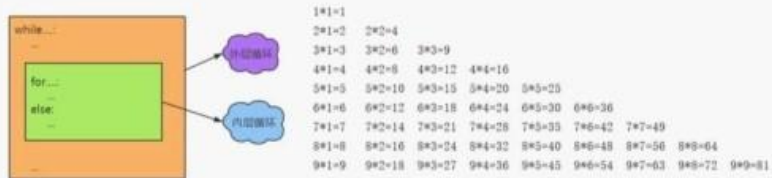
- 与else语句配合使用的三种情况



- 嵌套循环

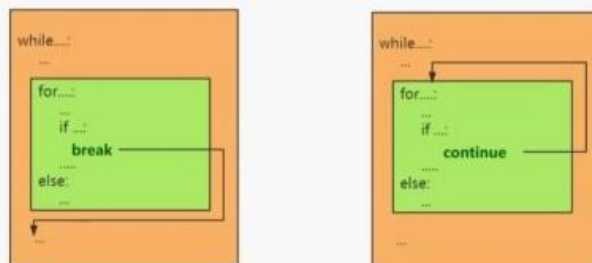
- 嵌套循环

- 循环结构中又嵌套了另外的完整的循环结构，其中内层循环做为外层循环的循环体执行



- 二重循环中的 break 和 continue

- 二重循环中的break和continue用于控制本层循环



break:退出内层循环，到外层循环 continue:回到内循环

- chap6 列表[]

- 为什么需要列表

- 变量可以存储一个元素，而列表是一个“大容器”可以存储N多个元素，程序可以方便地对这些数据进行整体操作

- 列表相当于其它语言中的数组

- 列表示意图

列表

索引	-7	-6	-5	-4	-3	-2	-1
数据	"hello"	"world"	123	00.6	"world"	125	"world"
索引	0	1	2	3	4	5	6

- 列表的创建

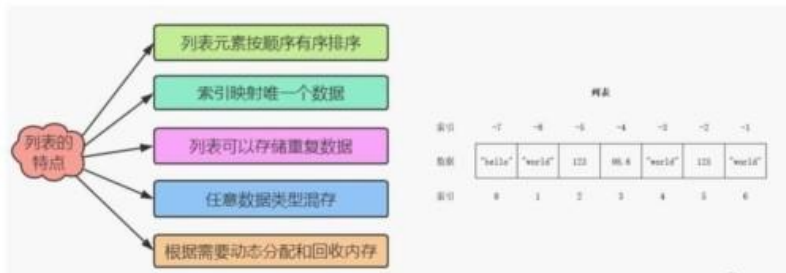
- 列表需要使用中括号[],元素之间使用英文的逗号进行分隔



- 列表的创建方式
  - 使用中括号
  - 调用内置函数list()

```
lst=['大圣','娟子姐']
lst2=list(['大圣','娟子姐'])
```

## 列表的特点



## 列表的查询操作

- 获取列表中指定元素的索引或指定元素

- 获取列表中指定元素的索引



- 获取列表中的单个元素

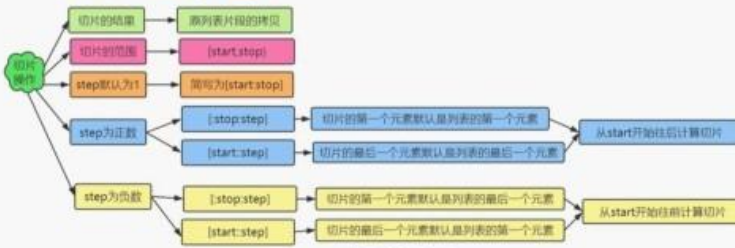


- 获取列表中的多个元素

- 获取列表中的多个元素

- 语法格式

列表名[ start : stop : step ]



- 判断指定元素在列表中是否存在&列表元素的遍历

- 判断指定元素在列表中是否存在

元素 **in** 列表名  
元素 **not in** 列表名

- 列表元素的遍历

for 迭代变量 in 列表名 :  
    操作

- 列表的增删改操作

- 列表元素的增加操作

	方法/其它	操作描述
增加操作	append()	在列表的末尾添加一个元素
	extend()	在列表的末尾至少添加一个元素
	insert()	在列表的任意位置添加一个元素
	切片	在列表的任意位置添加至少一个元素

- 列表元素的删除操作

	方法/其它	操作描述
删除操作	remove()	一次删除一个元素
		重复元素只删除第一个
		元素不存在抛出ValueError
	pop()	删除一个指定索引位置上的元素
		指定索引不存在抛出IndexError
		不指定索引，删除列表中最后一个元素
	切片	一次至少删除一个元素
	clear()	清空列表
	del	删除列表

- 列表元素的修改操作

## • 列表元素的修改操作

- 为指定索引的元素赋予一个新值
- 为指定的切片赋予一个新值

## • 列表元素的排序操作

### • 列表元素的排序操作

#### • 常见的两种方式

- 调用sort()方法，列表中的所有元素默认按照从小到大的顺序进行排序，可以指定reverse=True，进行降序排序
- 调用内置函数sorted()，可以指定reverse=True，进行降序排序，原列表不发生改变

## • 列表生成式

### • 列表生成式简称“生成列表的公式”

#### • 语法格式:



- 注意事项：“表示列表元素的表达式”中通常包含自定义变量

## • 总结



## • chap7 字典{}

### • 什么是字典

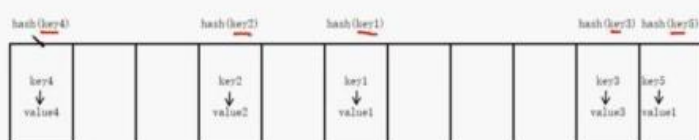
#### • 字典

- Python内置的数据结构之一，与列表一样是一个可变序列
- 以键值对的方式存储数据，字典是一个无序的序列



### • 示意图

## 字典示意图



## 字典的实现原理

- 字典的实现原理与查字典类似，查字典是先根据部首或拼音查找应的页码，Python中的字典是根据key查找value所在的位置

键值对的位置是用过 hash 函数的计算得来，所以 key 必须是不可变序列——str 和 int，不可变序列不能执行增删改操作。现在学到的可变序列：列表和字典

## 字典的创建

### 字典的创建

- 最常用的方式：使用花括号

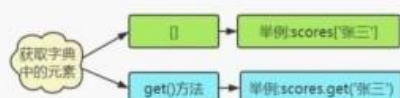
```
scores={ '张三': 100, '李四': 98, '王五': 45 }
```

- 使用内置函数dict()

```
dict( name='jack' , age=20 )
```

## 字典中元素的获取

### 字典中元素的获取



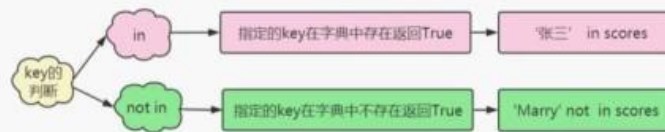
### []取值与使用get()取值的区别

- []如果字典中不存在指定的key，抛出keyError异常
- get()方法取值，如果字典中不存在指定的key，并不会抛出KeyError而是返回None，可以通过参数设置默认的值，以便指定的key不存在时返回

## 增删改



- key的判断



- 字典元素的删除

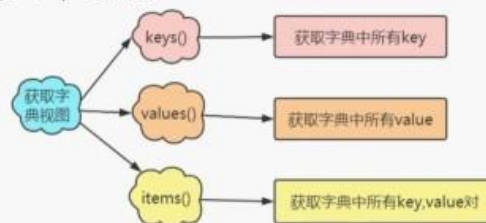
```
del scores[ '张三' ]
```

- 字典元素的新增

```
scores[ 'Jack' ] = 90
```

- 获取字典视图

- 获取字典视图的三个方法



- 字典元素的遍历

- 字典元素的遍历

```
for item in scores :  
    print( item )
```

- 字典的特点

- 字典的特点

- 字典中的所有元素都是一个 key-value对，key不允许重复，value可以重复
- 字典中的元素是无序的
- 字典中的key必须是不可变对象
- 字典也可以根据需要动态地伸缩
- 字典会浪费较大的内存，是一种使用空间换时间的数据结构

- 字典生成式

-



- 内置函数zip()
  - 用于将可迭代的对象作为参数，将对象中对应的元素打包成一个元组，然后返回由这些元组组成的列表

•



- 字典生成式

## • 总结



## • chap8 元组()与集合{}

### • 什么是元组



### • 元组的创建

- 元组的创建方式

- 直接小括号

```
t=( 'Python', 'hello', 90 )
```

- 使用内置函数tuple()

```
t= tuple( ('Python', 'hello' , 90) )
```

- 只包含一个元组的元素需要使用逗号和小括号

```
t=( 10 , )
```

- 为什么要将元组设计成不可变序列

- 

- 为什么要将元组设计成不可变序列

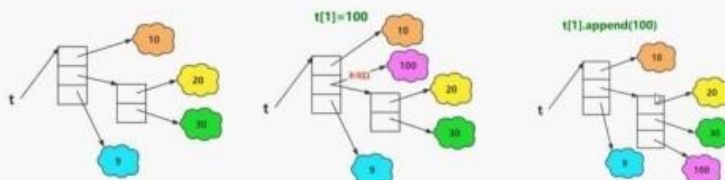
- 在多任务环境下，同时操作对象时不需要加锁
- 因此，在程序中尽量使用不可变序列

- **注意事项：**元组中存储的是对象的引用

- a)如果元组中对象本身不可对象，则不能再引用其它对象
- b)如果元组中的对象是可变对象，则可变对象的引用不允许改变，但数据可以改变

- 

```
t=( 10 ,[20,30] , 9 )
```



- 元组的遍历

- 元组是可迭代对象，所以可以使用for...in进行遍历

```
t=tuple(('Python','hello',90))
for item in t:
    print(item)
```

- 什么是集合

- 集合

- Python语言提供的内置数据结构
- 与列表、字典一样都属于可变类型的序列
- 集合是没有value的字典



- 集合的创建方式

- 集合的创建方式

- 直接{}

```
s={'Python' , 'hello' , 90 }
```

- 使用内置函数set()

```
s=set(range(6))
print(s)
print(set([3,4,53,56]))
print(set((3,4,43,435)))
print(set('Python'))
print(set({124,3,4,4,5}))
print(set())
```

- 集合的相关操作

- 集合元素的判断操作

- in或not in

- 集合元素的新增操作

- 调用add()方法，一次添中一个元素
- 调用update()方法至少添中一个元素

- 集合元素的删除操作

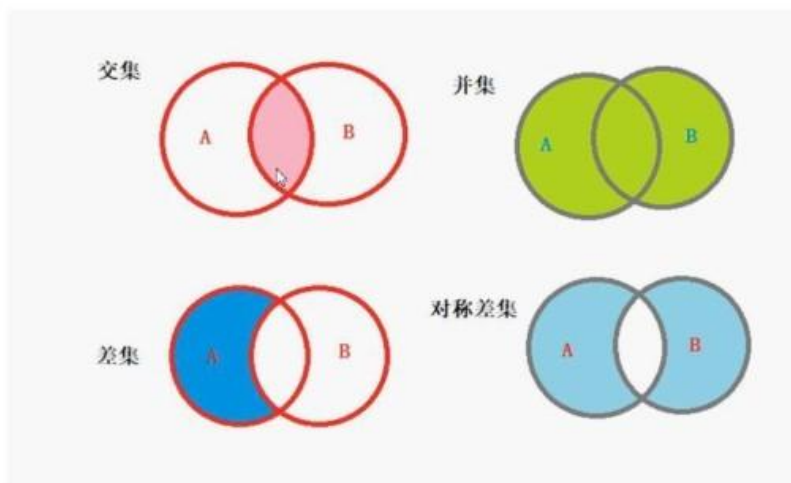
- 调用remove()方法，一次删除一个指定元素，如果指定的元素不存在抛出KeyError
- 调用discard()方法，一次删除一个指定元素，如果指定的元素不存在不抛出异常
- 调用pop()方法，一次只删除一个任意元素
- 调用clear()方法，清空集合

- 集合间的关系

- 两个集合是否相等
  - 可以使用运算符`==`或`!=`进行判断
- 一个集合是否是另一个集合的子集
  - 可以调用方法`issubset`进行判断
  - B是A的子集
- 一个集合是否是另一个集合的超集
  - 可以调用方法`issuperset`进行判断
  - A是B的超集
- 两个集合是否没有交集
  - 可以调用方法`isdisjoint`进行判断



## • 集合的数学操作



## • 集合生成式

### • 用于生成集合的公式

```
{ i*i for i in range(1,10) }
```



- 将`{}`修改为`[]`就是列表生成式`^_^`
- 没有元组生成式

## • 列表、元组、字典、集合总结

数据结构	是否可变	是否重复	是否有序	定义符号
列表(list)	可变	可重复	有序	[]
元组(tuple)	不可变	可重复	有序	()
字典(dict)	可变	key不可重复 value可重复	无序	{key:value}
集合(set)	可变	不可重复	无序	{}

不可变就是不能增删改只能遍历

## • 总结



## • chap9 字符串

### • 字符串的创建与驻留机制

•

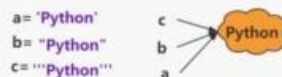
## 字符串的驻留机制

### • 字符串

- 在Python中字符串是基本数据类型，是一个不可变的字符序列

### • 什么叫字符串驻留机制呢？

- 仅保存一份相同且不可变字符串的方法，不同的值被存放在字符串的驻留池中，Python的驻留机制对相同的字符串只保留一份拷贝，后续创建相同字符串时，不会开辟新空间，而是把该字符串的地址赋给新创建的变量



### • 驻留机制的几种情况(交互模式)

- 字符串的长度为0或1时
- 符合标识符的字符串
- 字符串只在编译时进行驻留，而非运行时
- [-5,256]之间的整数数字

- sys中的intern方法强制2个字符串指向同一个对象
- PyCharm对字符串进行了优化处理



```

Python 3.6.0 (tags/v3.6.0:iaa
Type "help", "copyright", "ci
>>> s1=' '
>>> s2=' '
>>> s1 is s2
True
>>> s1='%'
>>> s2='%'
>>> s1 is s2
True
>>> s1='abc%'
>>> s2='abc%'
>>> s1==s2
True
>>> s1 is s2
False
>>> id(s1)
1639955366320
>>> id(s2)
1639955366384
>>> s1='abcx'
>>> s2='abcx'
>>> s1 is s2
True
>>> id(s1)
1639955366448
>>> id(s2)
1639955366448
>>>

```

#### • 字符串驻留机制的优缺点

- 当需要值相同的字符串时，可以直接从字符串池里拿来使用，避免频繁的创建和销毁，提升效率和节约内存，因此拼接字符串和修改字符串是会比较影响性能的。
- 在需要进行字符串拼接时建议使用 str 类型的 join 方法，而非 +，因为 join() 方法是先计算出所有字符中的长度，然后再拷贝，只 new 一次对象，效率要比 "+" 效率高

符合标识符的字符串：含有数字、字母、下划线

#### • 字符串的常用操作

##### • 查询操作

功能	方法名称	作用
查询方法	index()	查找子串 substr 第一次出现的位置, 如果查找的子串不存在时, 则抛出 ValueError
	rindex()	查找子串 substr 最后一次出现的位置, 如果查找的子串不存在时, 则抛出 ValueError
	find()	查找子串 substr 第一次出现的位置, 如果查找的子串不存在时, 则返回 -1
	rfind()	查找子串 substr 最后一次出现的位置, 如果查找的子串不存在时, 则返回 -1

##### • 字符串的大小写转换

功能	方法名称	作用
大小写转换	upper()	把字符串中所有字符都转成大写字母
	lower()	把字符串中所有字符都转成小写字母
	swapcase()	把字符串中所有大写字母转成小写字母, 把所有小写字母都转成大写字母
	capitalize()	把第一个字符转换为大写, 把其余字符转换为小写
	title()	把每个单词的第一个字符转换为大写, 把每个单词的剩余字符转换为小写

##### • 字符串内容对齐

功能	方法名称	作用
字符串对齐	center()	居中对齐, 第1个参数指定宽度, 第2个参数指定填充符, 第2个参数是可选的, 默认是空格, 如果设置宽度小于实际宽度则返回原字符串
	ljust()	左对齐, 第1个参数指定宽度, 第2个参数指定填充符, 第2个参数是可选的, 默认是空格, 如果设置宽度小于实际宽度则返回原字符串
	rjust()	右对齐, 第1个参数指定宽度, 第2个参数指定填充符, 第2个参数是可选的, 默认是空格, 如果设置宽度小于实际宽度则返回原字符串
	zfill()	右对齐, 左边用0填充, 该方法只接收一个参数, 用于指定字符串的宽度, 如果指定的宽度小于等于字符串的长度, 返回字符串本身

## • 字符串劈分

功能	方法名称	作用
字符串的劈分	split()	从字符串的左边开始劈分, 默认的劈分字符是空格字符串, 返回的值都是一个列表
		以通过参数sep指定劈分字符串的劈分符
		通过参数maxsplit指定劈分字符串时的最大劈分次数, 在经过最大次劈分之后, 剩余的子串会单独做为一部分
	rsplit()	从字符串的右边开始劈分, 默认的劈分字符是空格字符串, 返回的值都是一个列表
		以通过参数sep指定劈分字符串的劈分符
		通过参数maxsplit指定劈分字符串时的最大劈分次数, 在经过最大次劈分之后, 剩余的子串会单独做为一部分

## • 判断字符串

功能	方法名称	作用
判断字符串的方法	isidentifier()	判断指定的字符串是不是合法的标识符
	isspace()	判断指定的字符串是否全部由空白字符组成(回车、换行, 水平制表符)
	isalpha()	判断指定的字符串是否全部由字母组成
	isdecimal()	判断指定字符串是否全部由十进制的数字组成
	isnumeric()	判断指定的字符串是否全部由数字组成
	isalnum()	判断指定字符串是否全部由字母和数字组成

## • 字符串的替换与合并

功能	方法名称	作用
字符串替换	replace()	第1个参数指定被替换的子串, 第2个参数指定替换子串的字符串, 该方法返回替换后得到的字符串, 替换前的字符串不发生变化, 调用该方法时可以通过第3个参数指定最大替换次数
字符串的合并	join()	将列表或元组中的字符串合并成一个字符串

## • 字符串的比较操作

### •

#### • 字符串的比较操作

• 运算符: >, >=, <, <=, ==, !=

• 比较规则: 首先比较两个字符串中的第一个字符, 如果相等则继续比较下一个字符, 依次比较下去, 直到两个字符串中的字符不相等时, 其比较结果就是两个字符串的比较结果, 两个字符串中的所有后续字符将不再被比较

• 比较原理: 两上字符进行比较时, 比较的是其ordinal value(原始值), 调用内置函数ord可以得到指定字符的ordinal value。与内置函数ord对应的是内置函数chr, 调用内置函数chr时指定ordinal value可以得到其对应的字符

## • 字符串的切片操作

### •

## 字符串的切片操作

- 字符串是不可变类型
  - 不具备增、删、改等操作
  - 切片操作将产生新的对象



## • 格式化字符串

- 为什么需要格式化

## • 为什么需要格式化字符串

证明。

兹证明 XXX，身份证号：XXX，现居家庭住址：XXX，为我公司在职员工，工作地点为：XXX，单位负责人：XXX，座机联系电话：XXX，因工作需要，每天需出入该小区，本单元每天早晚有定时消毒、测量体温、出入登记等防控措施。

特此证明。

公司名称：XXX

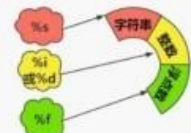
XXX 年 XX 月 XX 日。

因为有的地方需要变有的地方不用变，这就是字符串的拼接

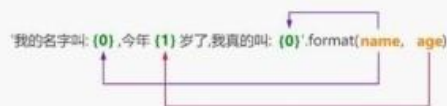
## • 格式化的两种方式

## • 格式化字符串的两种方式

- %作占位符



- {}作占位符



## • 字符串的编码转换

-

## • 为什么需要字符串的编码转换

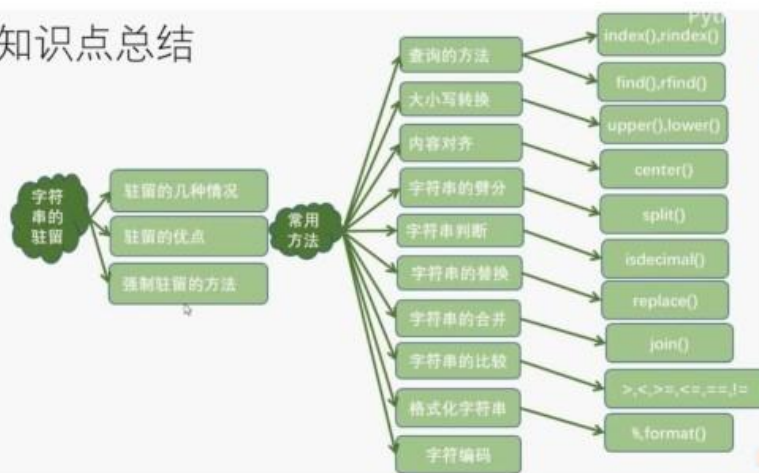


## • 编码与解码的方式

- **编码**：将字符串转换为二进制数据(bytes)
- **解码**：将bytes类型的数据转换成字符串类型

## • 总结

### 知识点总结



## • chap10 函数

### • 函数的创建和调用

## • 什么是函数

- 函数就是执行特定任务和以完成特定功能的一段代码

## • 为什么需要函数

- 复用代码
- 隐藏实现细节
- 提高可维护性
- 提高可读性便于调试

## • 函数的创建

```
def 函数名 ([输入参数]):  
    函数体  
    [return xxx]
```

## • 函数的创建

```
def calc(a,b):
    c=a+b
    return c
```

## • 函数的调用

函数名 ( [实际参数])

```
result=calc(10,20)
print(result)
```



## • 函数的参数传递

- 形参和实参可以名称不相同

## • 函数调用的参数传递

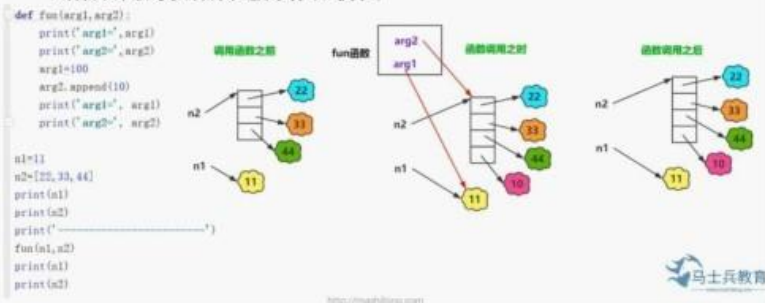
- 位置实参
  - 根据形参对应的位置进行实参传递

calc( 10, 20 )  
def calc( a , b ):

- 关键字实参
  - 根据形参名称进行实参传递

calc( b=10 , a=20 )  
def calc( a , b ):

## • 函数调用的参数传递内存分析图



## • 函数的返回值

- 函数返回多个值时，结果为元组

```
def fun(num):
    odd=[] #存奇数
    even=[] #存偶数
    for i in num:
        if i%2:
            odd.append(i)
        else:
            even.append(i)
    return odd,even

print(fun([10,29,34,23,44,53,55]))
```

```
Run: demo1
"C:\Program Files\Python38\python.exe"
E:/vippython/chap10/demo3.py
([29, 23, 53, 55], [10, 34, 44])
```

## • 函数的参数定义



• 默认值参数

• 函数定义默认值参数

- 函数定义时，给形参设置默认值，只有与默认值不符的时候才需要传递实参



• 个数可变的位置

• 个数可变的位置参数

- 定义函数时，可能无法事先确定传递的位置实参的个数时，使用可变的位置参数
- 使用\*定义个数可变的位置形参
- 结果为一个元组

```
def fun(*args):  
    print(args)  
fun(10)  
fun(10, 20, 30)
```

```
E:/dream/chapfile/d.py  
('10,')  
(10, 20, 30)
```

• 个数可变的关键字形参

- 定义函数时，无法事先确定传递的关键字实参的个数时，使用可变的关键字形参
- 使用\*\*定义个数可变的关键字形参
- 结果为一个字典

```
def fun(**args):  
    print(args)  
fun(a=10)  
fun(n=10, b=20, c=30)
```

```
E:/dream/chapfile/d.py  
{'a': 10}  
{'a': 10, 'b': 20, 'c': 30}
```

• 函数的参数总结

序号	参数的类型	函数的定义	函数的调用	备注
1	位置实参		√	
	将序列中的每个元素都转换为位置实参		√	使用*
2	关键字实参		√	
	将字典中的每个键值对都转换为关键字实参		√	使用**
3	默认值形参	√		
4	关键字形参	√		使用*
5	个数可变的位置形参	√		使用*
6	个数可变的关键字形参	√		使用**

• 变量的作用域

•

• 变量的作用域

- 程序代码能访问该变量的区域
- 根据变量的有效范围可分为
  - 局部变量
    - 在函数内定义并使用的变量，只在函数内部有效，局部变量使用global声明，这个变量就会就成全局变量
  - 全局变量
    - 函数体外定义的变量，可作用于函数内外

• 递归函数

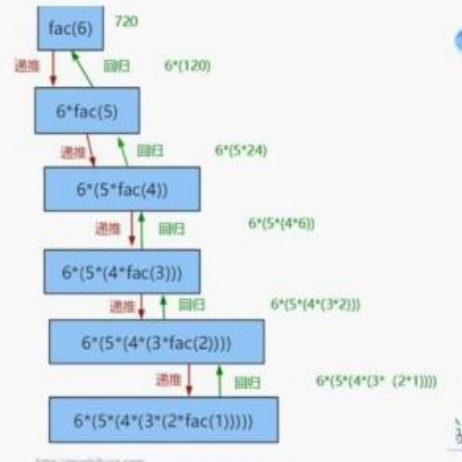
•



- 什么是递归函数
  - 如果在一个函数的函数体内调用了该函数本身，这个函数就称为递归函数
- 递归的组成部分
  - 递归调用与递归终止条件
- 递归的调用过程
  - 每递归调用一次函数，都会在栈内存分配一个栈帧，
  - 每执行完一次函数，都会释放相应的空间
- 递归的优缺点
  - 缺点：占用内存多，效率低下
  - 优点：思路 and 代码简单

## 递归函数

- 使用递归来计算阶乘



- 总结

•

## 知识点总结



- chap11 bug

- bug 的常见类型
  - 语法错误

## • Bug的常见类型

- 粗心导致的语法错误 SyntaxError

(1)

```
age=input('请输入你的年龄:')
if age>=18
    print('成年人,做事需要负法律责任了')
```

(2)

```
while i<10:
    print(i)
```

## • 粗心导致错误的自查宝典

- 1.漏了**末尾的冒号**,如if语句,循环语句,else子句等
- 2.**缩进错误**,该缩进的没缩进,不该缩进的瞎缩进
- 3.把**英文符号**写成中文符号,比如说:引号,冒号,括号
- 4.字符串拼接的时候,把**字符串和数字**拼在一起
- 5.没有**定义变量**,比如说while的循环条件的变量
- 6."**==**"比较运算符和"**=**"赋值运算符的混用

input 输入的都是 str 类型

## • 思路不清

### • 思路不清导致的问题

```
name=input('请输入你要查询的演员:')
for item in lst:
    for movie in item:
        actors=movie['actors']
        if name in actors:
            print(name+'出演了:'+movie)
```

```
E:/dream/chap11/demol.py
请输入你要查询的演员:张国荣
Traceback (most recent call last):
  File "E:/dream/chap11/demol.py", line 13, in <module>
    actors=movie['actors']
TypeError: string indices must be integers
```

### • 使用print()输出item以及moive的值

```
"E:/Program Files/Python37/python.exe" E:/dream/chap11/demol.py
请输入你要查询的演员:张国荣
Traceback (most recent call last):
  File "E:/dream/chap11/demol.py", line 13, in <module>
    actors=movie['actors']
TypeError: string indices must be integers
```

### • 思路不清导致的问题

- 第一层for循环遍历列表可以得到每一部电影,而每一部电影又是一个字典,只需要根据key在字典中取值即可。根据演员的键actors取出演员的列表,使用判断name在列表中是否存在,最后根据电影名称的键title取出电影的名称,进行输出

```
name=input('请输入你要查询的演员:')
for item in lst:
    #print(item)
    actors=item['actors']
    if name in actors:
        print(name+'出演了:'+item['title'])
```

```
E:/dream/chap11/demol.py
请输入你要查询的演员:李健,赵华
李健,赵华出演了:控方证人
```

## • 被动掉坑

- 被动掉坑:程序代码逻辑没有错,只是因为用户错误操作或者一些“例外情况”而导致的程序崩溃
- 题目要求:输入两个整数并进行除法运算

```

E:/dream/chap11/demo1.py
请输入一个整数:10
请输入另一个整数:5
结果为: 2.0

```

```

请输入一个整数:a
Traceback (most recent call last):
  File "E:/dream/chap11/demo1.py", line 5, in <module>
    n1=int(input('请输入一个整数:'))
ValueError: invalid literal for int() with base 10: 'a'

```

```

E:/dream/chap11/demo1.py
请输入一个整数:10
请输入另一个整数:#
Traceback (most recent call last):
  File "E:/dream/chap11/demo1.py", line 7, in <module>
    result=n1/n2
ZeroDivisionError: division by zero

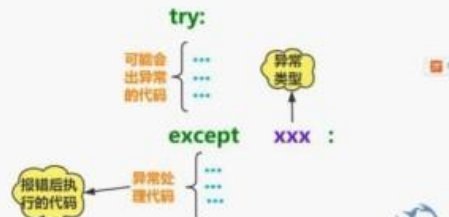
```

- 被动掉坑问题的解决方案
- Python提供了异常处理机制,可以在异常出现时即时捕获,然后内部“消化”,让程序继续运行

```

try:
    n1=int(input('请输入一个整数:'))
    n2=int(input('请输入另一个整数:'))
    result=n1/n2
    print('结果为:',result)
except ZeroDivisionError:
    print('除数不能为0哦!!!')

```



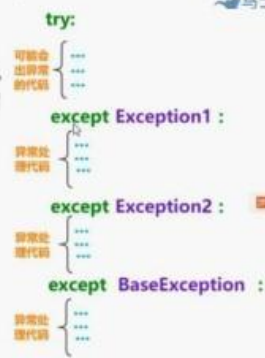
### • 多个except结构

- 捕获异常的顺序按照先子类后父类顺序,为了避免遗漏可能出现的异常,可以在最后增加BaseException

```

try:
    n1=int(input('请输入一个整数:'))
    n2=int(input('请输入另一个整数:'))
    result=n1/n2
    print('结果为:',result)
except ZeroDivisionError:
    print('除数不能为0哦!!!')
except ValueError:
    print('不能将字符串转换为数字')
except BaseException as e:
    print(e)

```



## • python 的异常处理机制

### • try...except...else结构

- 如果try块中没有抛出异常,则执行else块,如果try中抛出异常,except块

```

try:
    n1=int(input('请输入一个整数:'))
    n2=int(input('请输入另一个整数:'))
    result=n1/n2
except BaseException as e:
    print('出错了')
    print(e)
else:
    print('结果为:',result)

```

### • try...except...else...finally结构

- finally块无论是否发生异常都会被执行,能常用来释放try块中申请的资源

```

try:
    n1=int(input('请输入一个整数:'))
    n2=int(input('请输入另一个整数:'))
    result=n1/n2
except BaseException as e:
    print('出错了')
    print(e)
else:
    print('结果为:',result)
finally:
    print('无论是否产生异常,总会被执行的代码')
    print('程序结束')

```



• 常见异常类型

•

序号	异常类型	描述
1	ZeroDivisionError	除(或取模)零 (所有数据类型)
2	IndexError	序列中没有此索引(index)
3	KeyError	映射中没有这个键
4	NameError	未声明/初始化对象 (没有属性)
5	SyntaxError	Python 语法错误
6	ValueError	传入无效的参数

• traceback

•

• traceback模块

- 使用traceback模块打印异常信息

```
import traceback
try:
    print('1.-----')
    num=10/0
except:
    traceback.print_exc()
```

```
1.-----
Traceback (most recent call last):
  File "E:/dream/chap11/demol.py", line 7, in <module>
    num=10/0
ZeroDivisionError: division by zero
```

• pycharm 的程序调试

•


• 断点

- 程序运行到此处，暂时挂起，停止执行。此时可以详细观察程序的运行情况，方便做出进一步的判断

```
1 num=int(input('请输入年龄:'))
2 print(num)
```

• 进入调试视图

- 进入调试视图的三种方式

- (1)单击工具栏上的按钮 
- (2)右键单击编辑区：点击：debug'模块名'
- (3)快捷键shift+F9

• 总结

•



• chap12 编程

• 编程的两大思想

•

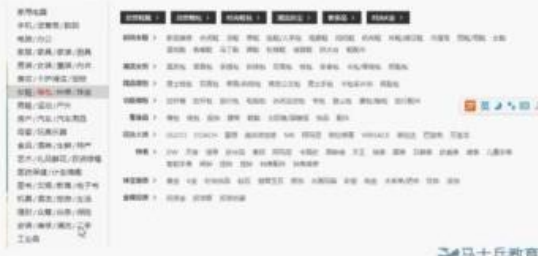
	面向过程	面向对象
区别	事物比较简单，可以用线性的思维去解决	事物比较复杂，使用简单的线性思维无法解决
共同点	面向过程和面向对象都是解决实际问题的一种思维方式	
	二者相辅相成，并不是对立的 解决复杂问题，通过面向对象方式便于我们从宏观上把握事物之间复杂的关系、方便我们分析整个系统；具体到微观操作，仍然使用面向过程方式来处理	

## • 类与对象

### • 类

- 类别，分门别类，物以类聚，人类，鸟类，动物类，植物类.....

• 类是多个类似事物组成的群体的统称。能够帮助我们快速理解和判断事物的性质



### • 数据类型

- 不同的数据类型属于不同的类
- 使用内置函数查看数据类型

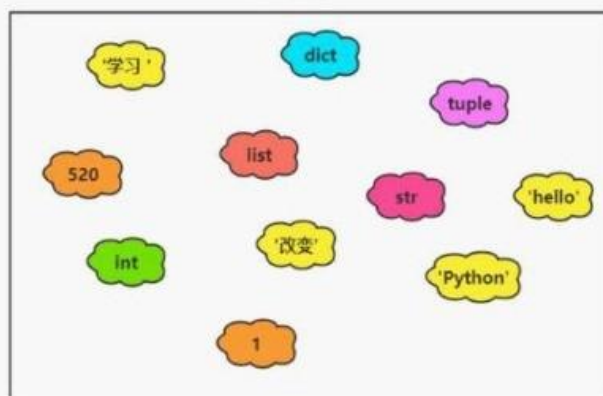
```
print(type(100))    <class 'int'>
print(type(99))     <class 'int'>
print(type(520))    <class 'int'>
```

### • 对象

- 100、99、520都是int类之下包含的相似的不同个例，这个个例专业术语称为实例或对象

# Python中一切皆对象

一切皆对象



## • 类的创建



## • 创建类的语法

```
class Student:
    pass
```

## • 类的组成

- 类属性
- 实例方法
- 静态方法
- 类方法

```
class Student:
    native_place = '吉林' #类属性
    def __init__(self, name, age): #name, age为实例属性
        self.name = name
        self.age = age
    #实例方法
    def info(self):
        print('我的名字叫:', self.name, '年龄是:', self.age)
    #类方法
    @classmethod
    def cm(cls):
        print('类方法')
    #静态方法
    @staticmethod
    def sm():
        print('静态方法')
```



## • 对象的创建

超纲，不用看

## 对象的创建

- 对象的创建又称为类的实例化

- 语法:

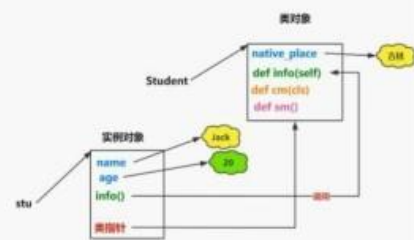
实例名 = 类名 ()

- 例子:

```
#创建Student类的实例对象
stu = Student('Jack', 20)
print(stu.name) #实例属性
print(stu.age) #实例属性
stu.info() #实例方法
```

stu = Student()

- 意义: 有了实例, 就可以调用类中的内容



## • 类属性、类方法、静态方法

- 类属性: 类中方法外的变量称为类属性, 被该类的所有对象所共享
- 类方法: 使用@classmethod修饰的方法, 使用类名直接访问的方法
- 静态方法: 使用@staticmethod修饰的主法, 使用类名直接访问的方法

```
print(Student.native_place) #访问类属性
Student.cm() #调用类方法
Student.sm() #调用静态方法
```

## • 动态绑定属性和方法

超纲，不用看

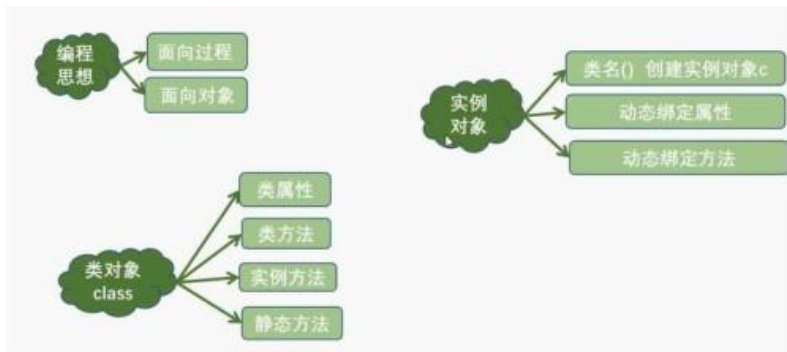
- Python是动态语言, 在创建对象之后, 可以动态地绑定属性和方法

```
def show():
    print('我是一函数')

stu = Student('Jack', 20)
stu.gender = '男' #动态绑定性别
print(stu.name, stu.age, stu.gender)
stu.show = show #动态绑定方法
stu.show()
```

## • 总结





## • chap13 对象

### • 面向对象的三大特征

**超纲，不用看**

#### • 封装

#### • 面向对象的三大特征

- 封装：提高程序的安全性
  - 将数据（属性）和行为（方法）包装到类对象中。在方法内部对属性进行操作，在类对象的外部调用方法。这样，无需关心方法内部的具体实现细节，从而隔离了复杂度。
  - 在Python中没有专门的修饰符用于属性的私有，如果该属性不希望在类对象外部被访问，前边使用两个“\_”。
- 继承：提高代码的复用性
- 多态：提高程序的可扩展性和可维护性

不关心类的内部，把类包起来直接使用即可

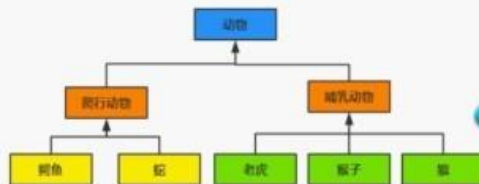
#### • 继承

##### • 继承

##### • 语法格式

```
class 子类类名( 父类1,父类2,... ) :
    pass
```

- 如果一个类没有继承任何类，则默认继承object



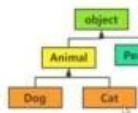
- Python支持多继承
- 定义子类时，必须在其构造函数中调用父类的构造函数

#### • 多态

##### • 多态

- 简单地说，多态就是“具有多种形态”，它指的是：即便不知道一个变量所引用的对象到底是什么类型，仍然可以通过这个变量调用方法，在运行过程中根据变量所引用对象的类型，动态决定调用哪个对象中的方法。

```
class Animal(object):
    def eat(self):
        print('动物要吃东西')
class Dog(Animal):
    def eat(self):
        print('狗吃肉')
class Cat(Animal):
    def eat(self):
        print('猫吃鱼')
class Person(object):
    def eat(self):
        print('人吃五谷杂粮')
```



```
def fun(animl):
    animl.eat()
fun(Dog())
fun(Cat())
fun(Person())
```

```
狗吃肉
猫吃鱼
人吃五谷杂粮
```

<http://mashibing.com>



## 静态语言与动态语言



超纲，不用看

### • 静态语言和动态语言关于多态的区别

#### • 静态语言实现多态的三个必要条件

- 继承
- 方法重写
- 父类引用指向子类对象

- 动态语言的多态崇尚“鸭子类型”当看到一只鸟走过来像鸭子、游泳起来像鸭子、收起来也像鸭子，那么这只鸟就可以被称为鸭子。在鸭子类型中，不需要关心对象是什么类型，到底是不是鸭子，只关心对象的行为。

传入对象，只要具备方法就可以实现，与是否有继承关系无关 Python 是动态语言，可以在创建对象之后动态绑定属性和方法

### • 方法重写（继承）

•

## 方法重写

### • 方法重写

- 如果子类对继承自父类的某个属性或方法不满意，可以在子类中对其进行（方法体）进行重新编写
- 子类重写后的方法中可以通过 `super().xxx()` 调用父类中被重写的方法

```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def info(self):
        print('姓名: {0}, 年龄: {1}'.format(self.name, self.age))

# 定义子类
class Student(Person):
    def __init__(self, name, age, score):
        super().__init__(name, age)
        self.score = score
    def info(self):
        super().info()
        print('学号: {0}'.format(self.score))

# 测试
stu = Student('Jack', 20, '1001')
stu.info()
```

### • object 类

•

## object 类



超纲，不用看

### • object 类

- object 类是所有类的父类，因此所有类都有 object 类的属性和方法。
- 内置函数 `dir()` 可以查看指定对象所有属性
- Object 有一个 `__str__()` 方法，用于返回一个对于“对象的描述”，对应于内置函数 `str()` 经常用于 `print()` 方法，帮助我们查看对象的信息，所以我们经常会对 `__str__()` 进行重写

```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def info(self):
        print('姓名: {0}, 年龄: {1}'.format(self.name, self.age))
    def __str__(self):
        return '姓名: {0}, 年龄: {1}'.format(self.name, self.age)

o = object()
p = Person('Jack', 20)
print(dir(o))
print(dir(p))
print(p)
```

### • 特殊方法和特殊属性

- 带两个下划线的就是特殊属性和特殊方法

## 特殊方法和特殊属性

✗ 超纲，不用看

	名称	描述
特殊属性	<code>__dict__</code>	获得类对象或实例对象所绑定的所有属性和方法的字典
特殊方法	<code>__len__()</code>	通过重写 <code>__len__()</code> 方法，让内置函数 <code>len()</code> 的参数可以是自定义类型
	<code>__add__()</code>	通过重写 <code>__add__()</code> 方法，可使用自定义对象具有“+”功能
	<code>__new__()</code>	用于创建对象
	<code>__init__()</code>	对创建的对象进行初始化

- 常用`__str__()`方法来输出属性值而不是内存地址
- 类的赋值与浅拷贝深拷贝

## 类的浅拷贝与深拷贝

### • 变量的赋值操作

- 只是形成两个变量，实际上还是指向同一个对象

### • 浅拷贝

- Python拷贝一般都是浅拷贝，拷贝时，对象包含的子对象内容不拷贝，因此，源对象与拷贝对象会引用同一个子对象

### • 深拷贝

- 使用`copy`模块的`deepcopy`函数，递归拷贝对象中包含的子对象，源对象和拷贝对象所有的子对象也不相同

## • 总结

- 三大特征与语言无关



## • chap14 模块

### • 什么是模块

- `.py` 文件就是一个模块

## • 模块

- 模块英文为Modules
- 函数与模块的关系
  - 一个模块中可以包含N多个函数
- 在Python中一个扩展名为.py的文件就是一个模块
- 使用模块的好处
  - 方便其它程序和脚本的导入并使用
  - 避免函数名和变量名冲突
  - 提高代码的可维护性
  - 提高代码的可重用性



## • 模块

- 函数
- 类
  - 类属性
  - 类方法
  - 静态方法
  - 实例属性
  - 属性
- 语句
- 程序由若干个模块组成

## • 模块的导入

•

## 自定义模块

### • 创建模块

- 新建一个.py文件，名称尽量不要与Python自带的标准模块名称相同

### • 导入模块

```
import 模块名称 [as 别名]
```

```
from 模块名称 import 函数/变量/类
```

## • 以主程序方式运行

•

### • 以主程序形式运行

- 在每个模块的定义中都包括一个记录模块名称的变量\_\_name\_\_，程序可以检查该变量，以确定他们在哪个模块中执行。如果一个模块不是被导入到其它程序中执行，那么它可能在解释器的顶级模块中执行。顶级模块的\_\_name\_\_变量的值为\_\_main\_\_

```
if __name__ == '__main__':  
    pass
```

- python 中的包
  - Python中的包
    - 包是一个分层次的目录结构，它将一组功能相近的模块组织在一个目录下
    - 作用：
      - 代码规范
      - 避免模块名称冲突
    - 包与目录的区别
      - 包含\_\_init\_\_.py文件的目录称为包
      - 目录里通常不包含\_\_init\_\_.py文件
    - 包的导入
 

```
import 包名.模块名
```



- python 程序
  - 包 1
    - 功能相近的若干模块
      - 函数
      - 类
      - 语句
  - 包 2
    - 功能相近的若干模块
  - 若干包
- python 中常用的内置模块
  -

✗ 超纲，不用看

## Python中常用的内置模块

模块名	描述
sys	与Python解释器及其环境操作相关的标准库
time	提供与时间相关的各种函数的标准库
os	提供了访问操作系统服务功能的标准库
calendar	提供与日期相关的各种函数的标准库
urllib	用于读取来自网上（服务器）的数据标准库
json	用于使用JSON序列化和反序列化对象
re	用于在字符串中执行正则表达式匹配和替换
math	提供标准算术运算函数的标准库
decimal	用于进行精确控制运算精度、有效数位和四舍五入操作的十进制运算
logging	提供了灵活的记录事件、错误、警告和调试信息等日志信息的功能

- 第三方模块的安装及使用

- 第三方模块的安装

**pip install 模块名**

```

C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.17763.195]
(c) 2018 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>pip install xlswriter -i http://pppi.douhan.com/simple --trusted-host pypi.douhan.com
Looking in indexes: http://pppi.douhan.com/simple
Collecting xlswriter
  Downloading http://pppi.douhan.com/packages/0d/0c/4376cd9d0773c99692714194c843bf96845d31a553f5abb6229e74ac1e/Xlswriter-1.2.7-m2-0r3-00w-mv.whl (141kB)
    143kB 1.7MB/s
Installing collected packages: xlswriter
Successfully installed xlswriter-1.2.7
  
```

- 第三方模块的使用

**import 模块名**

- 总结



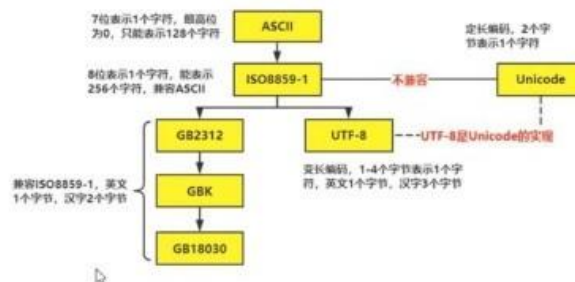
- chap15 文件操作

- 编码格式介绍

- 不同编码格式占用的内存大小不同

- 常见的字符编码格式

- Python的解释器使用的是Unicode（内存）
- .py文件在磁盘上使用UTF-8存储（外存）

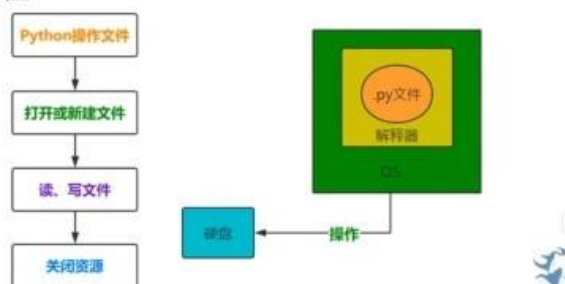


- 文件读写的原理



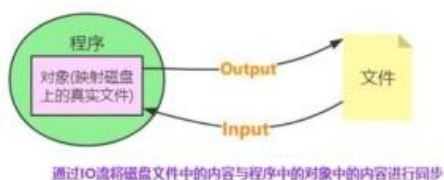
## 文件的读写原理

- 文件的读写俗称“IO操作”
- 文件读写操作流程
- 操作原理



## 文件的读写操作

- 内置函数open()创建文件对象



- 语法规则



- 常用的文件打开模式

## 常用的文件打开模式

- 文件的类型

- 按文件中数据的组织形式，文件分为以下两大类
  - 文本文件：存储的是普通“字符”文本，默认为unicode字符集，可以使用记事本程序打开
  - 二进制文件：把数据内容用“字节”进行存储，无法用记事本打开，必须使用专用的软件打开，举例：mp3音频文件.jpg图片 .doc文档等

打开模式	描述
r	以只读模式打开文件。文件的指针将会放在文件的开头。
w	以只写模式打开文件，如果文件不存在则创建。如果文件存在，则覆盖原有内容。文件指针在文件的开头。
a	以追加模式打开文件，如果文件不存在则创建。文件指针在文件开头，如果文件存在，则在文件末尾追加内容。文件指针在原文件末尾。
b	以二进制方式打开文件。不能单独使用，需要与其它模式一起使用，rb，或者wb
+>	以读写方式打开文件。不能单独使用，需要与其它模式一起使用，a+>

- 文件对象的常用方法

## • 文件对象的常用方法

方法名	说明
<code>read([size])</code>	从文件中读取size个字节或字符的内容返回。若省略[size]，则读取到文件末尾，即一次读取文件所有内容
<code>readline()</code>	从文本文件中读取一行内容
<code>readlines()</code>	把文本文件中每一行都作为独立的字符串对象，并将这些对象放入列表返回
<code>write(str)</code>	将字符串str内容写入文件
<code>writelines(s_list)</code>	将字符串列表s_list写入文本文件，不添加换行符
<code>seek(offset[,whence])</code>	把文件指针移动到新的位置，offset表示相对于whence的位置： offset: 为正往结束方向移动，为负往开始方向移动 whence不同的值代表不同含义： 0: 从文件头开始计算（默认值） 1: 从当前位置开始计算 2: 从文件尾开始计算
<code>tell()</code>	返回文件指针的当前位置
<code>flush()</code>	把缓冲区的内容写入文件，但不关闭文件
<code>close()</code>	把缓冲区的内容写入文件，同时关闭文件，释放文件对象相关资源

## • with 语句

•

## with语句(上下文管理器)

- with语句可以自动管理上下文资源，不论什么原因跳出with块，都能确保文件正确的关闭，以此来达到释放资源的目的



## • os 模块的常用语句

•

## 目录操作



超纲，不用看

- os模块是Python内置的与操作系统功能和文件系统相关的模块，该模块中的语句的执行结果通常与操作系统有关，在不同的操作系统上运行，得到的结果可能不一样。
- os模块与os.path模块用于对目录或文件进行操作

## os模块操作目录相关函数

函数	说明
<code>getcwd()</code>	返回当前的工作目录
<code>listdir(path)</code>	返回指定路径下的文件和目录信息
<code>mkdir(path[, mode])</code>	创建目录
<code>makedirs(path1/path2...[, mode])</code>	创建多级目录
<code>rmdir(path)</code>	删除目录
<code>removedirs(path1/path2...)</code>	删除多级目录
<code>chdir(path)</code>	将path设置为当前工作目录

- os.path 模块操作目录相关函数



超纲，不用看

- walk 可以把包括子目录下的所有文件都遍历

## os.path模块操作目录相关函数

函数	说明
<code>abspath(path)</code>	用于获取文件或目录的绝对路径
<code>exists(path)</code>	用于判断文件或目录是否存在，如果存在返回True，否则返回False
<code>join(path, name)</code>	将目录与目录或者文件名拼接起来
<code>splitext()</code>	分离文件名和扩展名
<code>basename(path)</code>	从一个目录中提取文件名
<code>dirname(path)</code>	从一个路径中提取文件路径，不包括文件名
<code>isdir(path)</code>	用于判断是否为路径

- 总结

•

## 知识点总结



