

2 数据类型与自带结构

8人过河问题

抽象现实问题

- 设A为河一边的状态，用集合{A} 分别表示父亲，母亲；儿子1；儿子2；女儿1；女儿2；警察；小偷的状态。=1 表示人在，=0表示人不在。
 - 则初始A=(1,1,1,1,1,1,1,1)
- 设B为河另外一边的状态数组，其含义与A数组相同。
 - 则初始B=(0,0,0,0,0,0,0,0)。
- 完成过河任务后，A=(0,0,0,0,0,0,0,0); B=(1,1,1,1,1,1,1,1)

想象一下世界中有多少种数？

整数，小数，分数.....

逻辑（正确与否）...

字，词，句....

一组数/集合...

让我从Byte(=8bit)讲起

- ▶ B->KB->MB->GB->TB->PB->EB->ZB->YB->BB->NB->DB ($2^{110}B$)
 - ▶ Byte, Kilobyte, Megabyte, Gigabyte, Terabyte, Petabyte, Exabyte, Zettabyte, Yottabyte, Brontobyte, NonaByte, DoggaByte
- ▶ 如何测量一条信息包含的信息量
 - ▶ 信息量的本质是不确定性（信息熵）
 - ▶ 2选1：不确定性为1；4选1：不确定性为2，8选1：不确定性为3 -> 不确定性公式是什么？
 - ▶ 仅用0/1这样一种最小表达及其组合就可以表示所有不确定性，定义为比特（bit）

A mathematical theory of communication

CE Shannon - Bell system technical journal, 1948 - Wiley Online Library

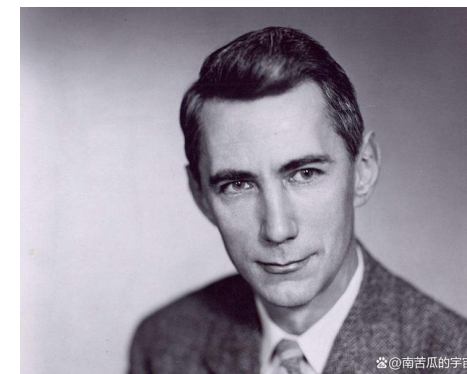
HE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist1 ...

☆ 99 Cited by 69155 Related articles All 143 versions »

[CITATION] The mathematical theory of communication

CE Shannon, W Weaver - 1998 - University of Illinois press

☆ 99 Cited by 45663 Related articles All 372 versions »



1916-2001

一个邪恶的国王地窖里有 n 瓶昂贵的酒，守卫抓到了一个要在酒里下毒的间谍。当守卫抓到他时，他已经成功地在—个瓶子里下了毒。很不幸，没有其他人知道是哪瓶。更糟糕的是，间谍使用的毒药是致命的，只要一滴，无论怎么稀释，也会致人死命。但是毒药的药性很慢，需要整整一个月才能显现出来。如何设计—种策略，可以在—个月内检查出被下毒的酒，并且使用的试酒者最少

分析：

1. 所有的酒必须在第一天检查
2. 一个月后，死亡的试酒者提供的信息必须足够用来判断被下毒的是哪瓶酒。
3. 检查的结果应只包括被下毒的那瓶酒，而不包括其他仍可饮用的酒。
4. 不等于死的人最少

可行的方案：

- 1. 让一个试酒者检查所有的酒
 - 结果：试酒者死亡，国王仍有危险
- 2. 让 J ($j < n-1$) 个试酒者，每人检查一瓶酒
 - 结果：可能有一个试酒者死亡，国王仍有危险的概率为 $1/(n-j)$
- 3. 让 n (或 $n-1$) 个试酒者，每人检查一瓶酒
 - 结果：国王安全了，一个试酒者死亡
- 4. 国王戒酒了
 - 结果：国王安全了，试酒者也安全了。双赢结果。

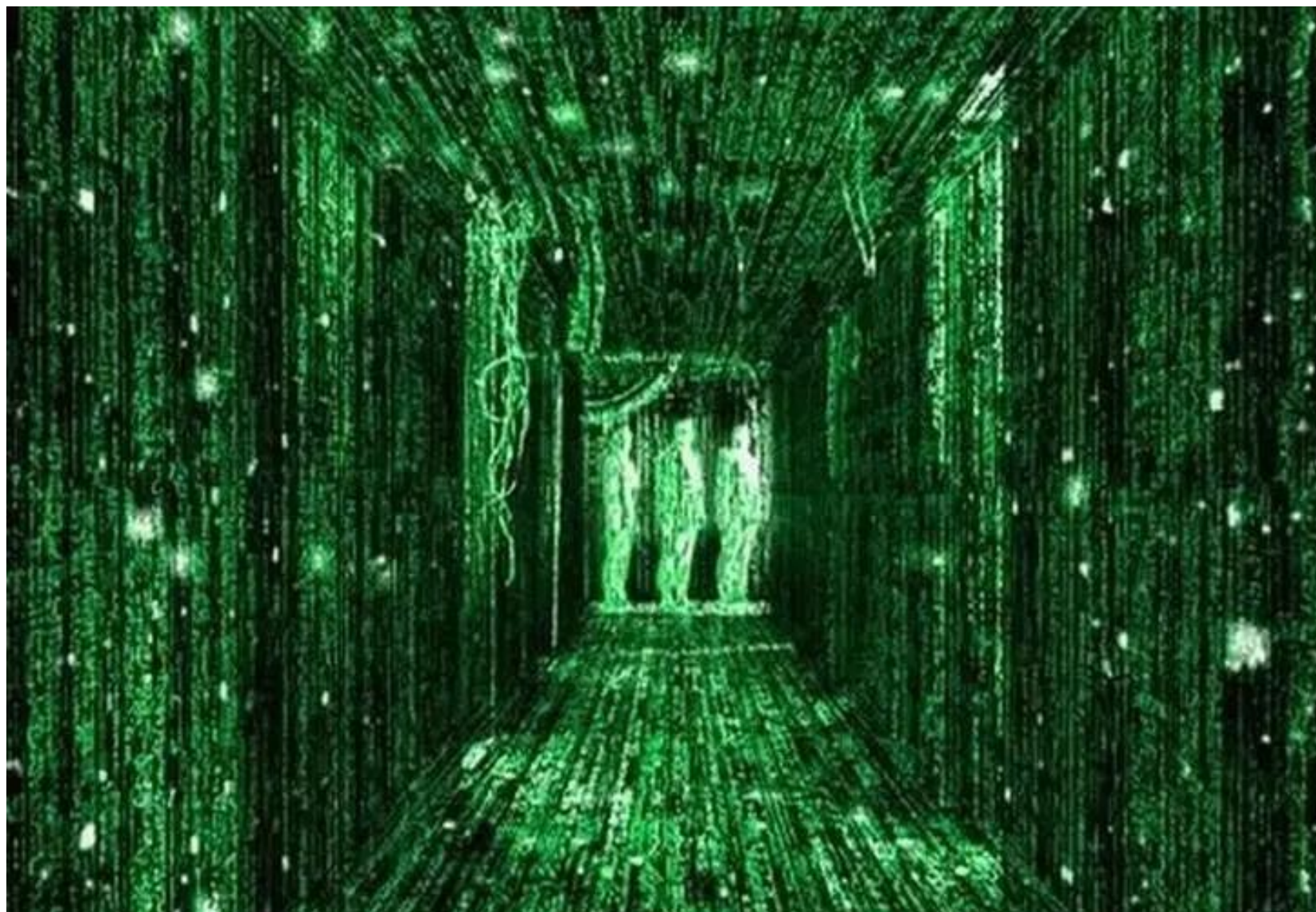
方案

- 1. 国王需要 $k = \log_2(n)$ 个试酒者 (n 的二进制表示的位数), 并给他们每人赋值, 酒也使用 k 位的二进制数字进行编码。
- 2. 在每瓶中取出一滴酒, 稀释后按照下列规则混合: 试酒者 i 的酒是所有编码第 i 位为 1 的酒的混合物。比如, 当 $n=8, k=3$
 - N 瓶酒被编码为 000, 001, 010, 011, 100, 101, 110, 111
 - 第0个试酒者: 尝 00**1**, 01**1**, 10**1**, 11**1**
 - 第1个试酒者: 尝 0**1**0, 0**1**1, 1**1**0, 1**1**1
 - 第2个试酒者: 尝 **1**00, **1**01, **1**10, **1**11
- 3. 一个月后, 部分试酒者会死亡, 在所有死亡者对应的位置标 1, 其他位置标 0, 可以得到一个数字, 该数字即毒酒的编号。
- 找到一种方法使得每个存活/死亡者的信息都确认了一个字节的值

背后的本质：1/n概率发生事件的不确定性是多少？

$\text{Log}_2(n)$

也可以用 $\text{Log}_2(n)$ 的信息量来表示



Python常用内置对象

对象类型	类型名称	示例	简要说明
数字	int float complex	1234 3.14, 1.3e5 3+4j	数字大小没有限制，内置支持复数及其运算
字符串	str	'swfu', "I'm student", '''Python ''', r'abc', R'bcd'	使用单引号、双引号、三引号作为定界符，以字母r或R引导的表示原始字符串
字节串	bytes	b'hello world'	以字母b引导，可以使用单引号、双引号、三引号作为定界符
列表	list	[1, 2, 3] ['a', 'b', ['c', 2]]	所有元素放在一对方括号中，元素之间使用逗号分隔，其中的元素可以是任意类型
字典	dict	{1:'food' ,2:'taste', 3:'import'}	所有元素放在一对大括号中，元素之间使用逗号分隔， 元素形式为“键:值”
元组	tuple	(2, -5, 6) (3,)	不可变 ，所有元素放在一对圆括号中，元素之间使用逗号分隔， 如果元组中只有一个元素的话，后面的逗号不能省略
集合	set frozenset	{'a', 'b', 'c'}	所有元素放在一对大括号中，元素之间使用逗号分隔， 元素不允许重复 ；另外，set是可变的，而frozenset是不可变的

Python常用内置对象

对象类型	类型名称	示例	简要说明
布尔型	bool	True, False	逻辑值，关系运算符、成员测试运算符、同一性测试运算符组成的表达式的值一般为True或False
空类型	NoneType	None	空值
文件		f = open('data.dat', 'rb')	open是Python内置函数，使用指定的模式打开文件，返回文件对象
异常	Exception ValueError TypeError		Python内置大量异常类，分别对应不同类型的异常
其他可迭代对象		生成器对象、range对象、zip对象、enumerate对象、map对象、filter对象等等	具有 惰性求值 的特点，除range对象之外，其他对象中的元素只能看一次
编程单元		函数（使用def定义） 类（使用class定义） 模块（类型为module）	类和函数都属于 可调用对象 ，模块用来集中存放函数、类、常量或其他对象

常量与变量

- 在Python中，**不需要事先声明变量名及其类型**，直接赋值即可创建各种类型的对象变量。这一点适用于Python任意类型的对象。

例如语句

```
>>> x = 3
```

凭空出现一个整型变量x

创建了整型变量x，并赋值为3，再例如语句

```
>>> x = 'Hello world.'
```

新的字符串变量，再也不是原来的x了

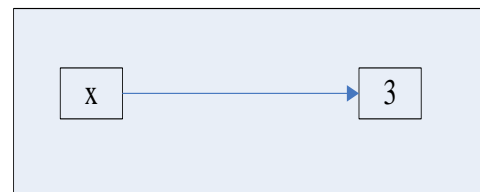
创建了字符串变量x，并赋值为'Hello world.'。

带来的挑战：随着程序越来越复杂，变量管理难度增加

常量与变量

- 赋值语句的执行过程是：首先把等号右侧表达式的值计算出来，然后在内存中寻找一个位置把值存放进去，最后创建变量并指向这个内存地址。

```
>>> x = 3
```



- Python中的变量并不直接存储值，而是存储了值的内存地址或者引用，这也是变量类型随时可以改变的原因。
- 带来的挑战：变量被使用错时，更难被机器/人类发现

深夜突发！B站崩了

播报文章



沈阳晚报

2021-07-14 18:44 | 《沈阳晚报》官方帐号

关注

7月13日晚间，许多网友突然发现**B站崩了**，消息迅速刷屏，空降热搜第一。

微博热搜

每分钟更新

中国开展第12次北极科学考察



1 b站崩了 1186万



直播

推荐

热门

追番

影视

建党百年



网络出错了 (´・Д・) error-1

bilibili

主站 番剧 游戏中心 直播 会员购 漫画 赛事 下载APP

肥肠抱歉，该页面暂时无法访问

返回上一页

休息一下，看看吧~



自我介绍

热门



1小时前 来自 iPhone 12 Pro

#b站崩了#原来没有b站的时间这么无聊

191

1472

3.1万

+关注



上海消防

7-13 23:57

+关注

【#上海#b站大楼未见火情】网传#b站崩了#有火情发生，经了解，位于上海市政立路485号国正中心内的哔哩哔哩弹幕网B站（总部）未出现火情，未接到相关报警。具体情况以站方公布为准。



原因找到了

我家阳台上拍的照片

真停电了

一个变量引发的血案

```
17  local _gcd
18  _gcd = function (a, b)
19      if b == 0 then
20          return a
21      end
22
23      return _gcd(b, a % b)
24  end
25
```

例子：18和24的最大公约数

1	第一次：a = 18 b = 24 c = a%b = 18%24 = 18
2	循环中：a = 24 b = 18

1	第二次：a = 24 b = 18 c = a%b = 24%18 = 6
2	循环中：a = 18 b = 6

1	第三次：a = 18 b = 6 c = a%b = 18%6 = 0
2	循环结束

如果大数可以整除小数，那么最大公约数为小数。
如果不能整除小数，那么大数对小数求余，小数和余数能够整除的，就是最大公约数。

可证明：Y、R之间的最大公约数也是X、Y之间的最大公约数（R为X，Y之间的余数）

近日，“B站崩了”事故闹得沸沸扬扬，虽然官方已经解释了原因，且目前已经修复完毕可正常使用，但关于B站这次出现的网络故障，不少网友还是不依不饶。



昨晚，B站的部分服务器机房发生故障，造成无法访问。技术团队随即进行了问题排查和修复，现在服务已经陆续恢复正常。
耽误大家看视频了，对不起！

21-7-14 02:20

216

412

964

昨晚，B站在官方微博发布消息，给全体会员带来了好消息，B站表示很抱歉因站点崩溃耽误大家看视频，所以将会赠送所有用户1天大会员。

付费用户2750万

< 返回

微博正文

...



哔哩哔哩弹幕网



7-14 20:18

+ 关注

很抱歉昨晚#B站崩了#，耽误小伙伴们看视频了，我们将会赠送所有用户1天大会员。
领取方式见评论

技能君看了下领取界面，B站这次不仅赠送全体用户1天大会员，还外加1天电视大会员。

如果你已经是B站大会员，若没有手动领取的话，系统也会自动发放，将你的大会员有效期自动延续1天。



常量与变量

- 在定义变量名的时候，需要注意以下问题：
 - ✓ 变量名**必须**以字母或下划线开头，但以下划线开头的变量在Python中有特殊含义；
 - ✓ 变量名中**不能**有空格以及标点符号（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等等）；
 - ✓ **不能**使用关键字作变量名，可以import keyword模块后使用print(keyword.kwlist)查看所有Python关键字；
 - ✓ 变量名对英文字母的**大小写敏感**，例如student和Student是不同的变量。
 - ✓ **不建议**使用系统内置的模块名、类型名或函数名以及已导入的模块名及其成员名作变量名，这将会改变其类型和含义，可以通过dir(__builtins__)查看所有内置模块、类型和函数；

双下划线

数字

- Python支持任意大的数字，具体可以大到什么程度仅受内存大小的限制。
- 由于精度的问题，对于实数运算可能会有一定的误差，应尽量避免在实数之间直接进行相等性测试，而是应该以二者之差的绝对值是否足够小作为两个实数是否相等的依据 (p13)
- 在数字的算术运算表达式求值时会进行隐式的类型转换，如果存在复数则都变成复数，如果没有复数但是有实数就都变成实数，如果都是整数则不进行类型转换。

数字

```
>>> 9999 ** 99                                     #这里**是幂乘运算符，等价于内置函数pow()
990148353526723487602263124753282625570559528895791057324326529121794837
894053513464422176826916433932586924386677766244032001623756821400432975
051208820204980098735552703841362304669970510691243800218202840374329378
800694920309791954185117798434329591212159106298699938669908067573374724
331208942425544893910910073205049031656789220889560732962926226305865706
593594917896276756396848514900989999
>>> 0.3 + 0.2                                     #实数相加
0.5
>>> 0.4 - 0.1                                     #实数相减，结果稍微有点偏差
0.30000000000000004
>>> 0.4 - 0.1 == 0.3                             #应尽量避免直接比较两个实数是否相等
False
>>> abs(0.4-0.1 - 0.3) < 1e-6                   #这里1e-6表示10的-6次方
True
```

关于数的极限和精度

- ▶ 《诗云》刘慈欣
- ▶ 借助伟大的技术，我写出了诗词的颠峰之作，却不可能把它们从诗云中检索出来
- ▶ <https://www.zhihu.com/tardis/sogou/art/463954871>

美国又下狠手！对中国禁售高端GPU芯片，倒逼国产替代加速！

🔊 播报文章



科技每日推送

2022-09-01 18:47 广东

关注

9月1日媒体消息，美国对华科技战进一步升级。

美国芯片厂商AMD、英伟达，相继接到美国政府下达的暂停出口命令，对中国区客户断供高端GPU芯片，以避免沦为“军事用途”。

有业内人士指出，本次美国断供高端GPU芯片，将危及中国的AI计算与超算平台，甚至波及整个互联网领域，直击我们的要害。

01

🔊 科技每日推送

AMD、英伟达均被限制

8月31日英伟达披露的一份文件，证实了这一消息的真实性。英伟达表示：

2022年8月26日，美国政府通知NVIDIA，美国政府对英伟达A100和即将推出的H100集成电路今后向中国（包括香港）和俄罗斯的任何出口提出了新的许可要求，立即生效。

数字

- Python内置支持复数类型及其运算，并且形式与数学上的复数完全一致

```
>>> x = 3 + 4j
```

#使用j或J表示复数虚部

```
>>> y = 5 + 6j
```

```
>>> x + y
```

#支持复数之间的加、减、乘、除以及幂乘等运算

```
(8+10j)
```

```
>>> x * y
```

```
(-9+38j)
```

```
>>> abs(x)
```

#内置函数abs()可用来计算复数的模

```
5.0
```

```
>>> x.imag
```

#虚部

```
4.0
```

```
>>> x.real
```

#实部

```
3.0
```

```
>>> x.conjugate()
```

#共轭复数

```
(3-4j)
```

数字

- Python 3.6.x支持在数字中间位置使用单个下划线作为分隔来提高数字的可读性，类似于数学上使用逗号作为千位分隔符。
- 在Python数字中单个下划线可以出现在中间任意位置，但不能出现开头和结尾位置，也不能使用多个连续的下划线。

```
>>> 1_000_000
```

```
1000000
```

```
>>> 1_2_3_4
```

```
1234
```

```
>>> 1_2 + 3_4j
```

```
(12+34j)
```

```
>>> 1_2.3_45
```

```
12.345
```

数字

- Python标准库fractions中的Fraction对象支持分数及其运算

```
>>> from fractions import Fraction
>>> x = Fraction(3, 5)      #创建分数对象
>>> y = Fraction(3, 7)
>>> x
Fraction(3, 5)
>>> x ** 2                  #幂运算
Fraction(9, 25)
>>> x.numerator             #查看分子
3
>>> x.denominator          #查看分母
5
>>> x + y                  #支持分数之间的四则运算，自动进行通分
Fraction(36, 35)
```


数字

- 标准库fractions和decimal中提供的Decimal类实现了更高精度实数的运算

```
>>> from fractions import Decimal
>>> 1 / 9                                #内置的实数类型
0.1111111111111111
>>> Decimal(1/9)                         #高精度实数
Decimal('0.1111111111111111104943205418749130330979824066162109375')
>>> 1 / 3
0.3333333333333333
>>> Decimal(1/3)
Decimal('0.33333333333333333314829616256247390992939472198486328125')
>>> Decimal(1/9) + Decimal(1/3)
Decimal('0.4444444444444444444197728216750')
```

字符串与字节串

- 在Python中，**单个字符也是字符串**。使用**单引号、双引号、三单引号、三双引号**作为定界符（delimiter）来表示字符串，并且**不同的定界符之间可以互相嵌套**
- Python 3.x全面支持中文，中文和英文字母都作为一个字符对待，甚至**可以使用中文作为变量名**
- 除了支持使用加号运算符连接字符串以外，Python字符串还提供了大量的方法支持格式化、查找、替换、排版等操作。

字符串与字节串

```
>>> x = 'Hello world.'
>>> x = "Python is a great language."
>>> x = '''Tom said, "Let's go."'''
>>> print(x)
Tom said, "Let's go."
>>> x = 'good ' + 'morning'
>>> x
'good morning'
>>> x = 'good ''morning'
>>> x
'good morning'
>>> x = 'good '
>>> x = x'morning'
SyntaxError: invalid syntax
>>> x = x + 'morning'
>>> x
'good morning'
```

#使用单引号作为定界符
#使用双引号作为定界符
#不同定界符之间可以互相嵌套
#如果直接显示x, 则显示转义字符\
#Tom said, "Let\'s go."
#连接字符串

#连接字符串, 仅适用于字符串常量

#不适用于字符串变量

#字符串变量之间的连接可以使用加号

字符串与字节串

- 对str类型的字符串调用其`encode()`方法进行编码得到bytes字节串，对bytes字节串调用其`decode()`方法并指定正确的编码格式则得到str字符串

```
>>> type('Hello world')  
<class 'str'>
```

#默认字符串类型为str

```
>>> type(b'Hello world')  
<class 'bytes'>
```

#在定界符前加上字母b表示字节串

```
>>> 'Hello world'.encode('utf8')  
b'Hello world'
```

#使用utf8编码格式进行编码

```
>>> 'Hello world'.encode('gbk')  
b'Hello world'
```

#使用gbk编码格式进行编码

```
>>> '张诚'.encode('utf8')  
b'\xe5\xbc\xa0\xe8\xaf\x9a'
```

#对中文进行编码

```
>>> _.decode('utf8')  
b'\xd5\xc5\xb3\xcf'
```

#一个下划线表示最后一次正确输出结果

```
>>> '张诚'.encode('cp936').decode('cp936')  
'张诚'
```

Python运算符与表达式

- 运算符优先级遵循的规则为：算术运算符优先级最高，其次是位运算符、成员测试运算符、关系运算符、逻辑运算符等，算术运算符遵循“先乘除，后加减”的基本运算原则
- 虽然Python运算符有一套严格的优先级规则，但是强烈建议在编写复杂表达式时使用圆括号来明确说明其中的逻辑来提高代码可读性

Python运算符与表达式

运算符	功能说明
+	算术加法，列表、元组、字符串合并与连接，正号
-	算术减法，集合差集，相反数
*	算术乘法，序列重复
/	真除法
//	求整商，但如果操作数中有实数的话，结果为实数形式的整数
%	求余数，字符串格式化
**	幂运算
<、<=、>、>=、==、!=	（值）大小比较，集合的包含关系比较
or	逻辑或
and	逻辑与
not	逻辑非
in	成员测试
is	对象同一性测试，即测试是否为同一个对象或内存地址是否相同
、^、&、<<、>>、~	位或、位异或、位与、左移位、右移位、位求反
&、 、^	集合交集、并集、对称差集
@	矩阵相乘运算符

算术运算符

(1) **+运算符**除了用于算术加法以外，还可以用于列表、元组、字符串的连接，但不支持不同类型的对象之间相加或连接

```
>>> [1, 2, 3] + [4, 5, 6]
```

#连接两个列表

```
[1, 2, 3, 4, 5, 6]
```

```
>>> (1, 2, 3) + (4,)
```

#连接两个元组

```
(1, 2, 3, 4)
```

```
>>> 'abcd' + '1234'
```

#连接两个字符串

```
'abcd1234'
```

```
>>> 'A' + 1
```

#不支持字符与数字相加，抛出异常

```
TypeError: Can't convert 'int' object to str implicitly
```

```
>>> True + 3
```

#Python内部把True当作1处理

```
4
```

```
>>> False + 3
```

#把False当作0处理

```
3
```

算术运算符

(2) ***运算符**除了表示算术乘法，还可用于列表、元组、字符串这几个序列类型与整数的乘法，表示序列元素的重复，生成新的序列对象。**字典和集合不支持与整数的相乘**，因为其中的元素是不允许重复的。

```
>>> True * 3
```

```
3
```

```
>>> False * 3
```

```
0
```

```
>>> [1, 2, 3] * 3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> (1, 2, 3) * 3
```

```
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

```
>>> 'abc' * 3
```

```
'abccabccabc'
```


算术运算符

(3) **运算符/和//**在Python中分别表示算术除法和算术求整商 (floor division)

```
>>> 3 / 2
```

#数学意义上的除法

```
1.5
```

```
>>> 15 // 4
```

#如果两个操作数都是整数，结果为整数

```
3
```

```
>>> 15.0 // 4
```

#如果操作数中有实数，结果为实数形式的整数值

```
3.0
```

```
>>> -15//4
```

#向下取整

```
-4
```

算术运算符

(4) **%运算符**可以用于整数或实数的求余数运算，还可以用于字符串格式化，但是这种方法并不推荐

>>> 789 % 23 #余数

7

>>> 123.45 % 3.2 #可以对实数进行余数运算，注意精度问题

1.8499999999999996

>>> '%c, %d' % (65, 65) #把65分别格式化为字符和整数

'A, 65'

>>> '%f,%s' % (65, 65) #把65分别格式化为实数和字符串

'65.000000,65'

算术运算符

(5) ****运算符**表示幂乘:

```
>>> 3 ** 2
```

```
9
```

#3的2次方, 等价于pow(3, 2)

```
>>> pow(3, 2, 8)
```

```
1
```

#等价于(3**2) % 8

```
>>> 9 ** 0.5
```

```
3.0
```

#9的0.5次方, 平方根

```
>>> (-9) ** 0.5
```

```
(1.8369701987210297e-16+3j)
```

#可以计算负数的平方根

关系运算符

- Python **关系运算符**最大的特点是**可以连用**，其含义与我们日常的理解完全一致。使用关系运算符的一个最重要的前提是，**操作数之间必须可比较大小**。例如把一个字符串和一个数字进行大小比较是毫无意义的，所以Python也不支持这样的运算

```
>>> 1 < 3 < 5                                #等价于1 < 3 and 3 < 5
True
>>> 3 < 5 > 2
True
>>> 1 > 6 < 8
False
>>> 1 > 6 < math.sqrt(9)                      #具有惰性求值或者逻辑短路的特点
False
>>> 1 < 6 < math.sqrt(9)                      #还没有导入math模块，抛出异常
NameError: name 'math' is not defined
>>> import math
>>> 1 < 6 < math.sqrt(9)
False
```

关系运算符

```
>>> 'Hello' > 'world'
```

```
False
```

#比较字符串大小

```
>>> [1, 2, 3] < [1, 2, 4]
```

```
True
```

#比较列表大小

```
>>> 'Hello' > 3
```

```
TypeError: unorderable types: str() > int()
```

#字符串和数字不能比较

```
>>> {1, 2, 3} < {1, 2, 3, 4}
```

```
True
```

#测试是否子集

```
>>> {1, 2, 3} == {3, 2, 1}
```

```
True
```

#测试两个集合是否相等

```
>>> {1, 2, 4} > {1, 2, 3}
```

```
False
```

#集合之间的包含测试

```
>>> {1, 2, 4} < {1, 2, 3}
```

```
False
```

```
>>> {1, 2, 4} == {1, 2, 3}
```

```
False
```

位运算符与集合运算符

- 位运算符只能用于整数，其内部执行过程为：首先将整数转换为二进制数，然后右对齐，必要的时候左侧补0，按位进行运算，最后再把计算结果转换为十进制数字返回
- 位与运算规则为 $1 \& 1 = 1$ 、 $1 \& 0 = 0 \& 1 = 0 \& 0 = 0$ ，位或运算规则为 $1 | 1 = 1 | 0 = 0 | 1 = 1$ 、 $0 | 0 = 0$ ，位异或运算规则为 $1 \wedge 1 = 0 \wedge 0 = 0$ 、 $1 \wedge 0 = 0 \wedge 1 = 1$ 。
- 左移位时右侧补0，每左移一位相当于乘以2；右移位时左侧补0，每右移一位相当于整除以2。

位运算符与集合运算符

>>> 3 << 2

12

#把3左移2位

>>> 3 & 7

3

#位与运算

011&111=011 (3)

>>> 3 | 8

11

#位或运算

0011|1000=1011 (11)

>>> 3 ^ 5

6

#位异或运算

011^101=110 (6)

位与运算: $1\&1=1$ 、 $1\&0=0$ 、 $0\&1=0$ 、 $0\&0=0$

位或运算: $1|1=1$ 、 $0|1=1$ 、 $1|0=1$ 、 $0|0=0$

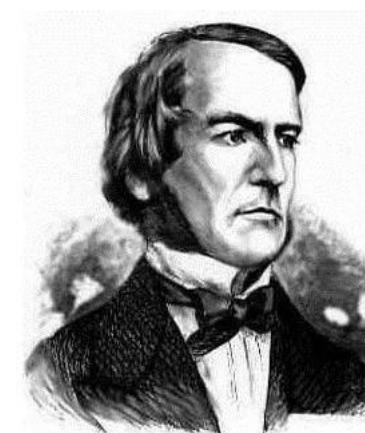
位异或运算: $1\^1=0$ 、 $0\^0=0$ 、 $1\^0=1$ 、 $0\^1=1$

左移位时右侧补0, 每左移一位相当于乘以2

移位时左侧补0, 每右移一位相当于整除以2

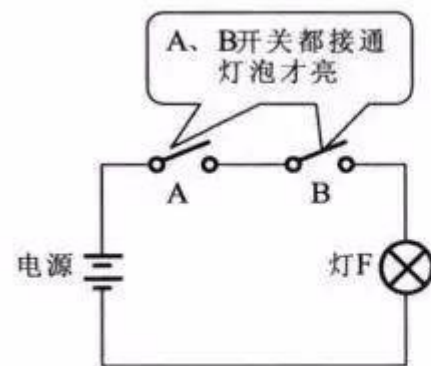
与，或，非：逻辑推理的灵魂

- ▶ 所谓逻辑，就是判断真假或者是非
- ▶ 如何通过符号和代数（运算）系统地进行逻辑推理
 - ▶ “我想寻求这样一张特殊的字母表，其元素表示的是概念。有了这样一个符号系统，我们就可以发展出一种语言，我们仅凭符号演算，就可以确定用这种语言写成的哪些句子为真，以及它们之间存在着什么样的逻辑关系。” ——莱布尼茨
- ▶ 逻辑的数学分析 The Mathematical Analysis of Logic
 - ▶ 负负得正
 - ▶ 与（且）、或、非 (and, or, not)
 - ▶ 以及由此衍生出的与非 (not and)，或非(not or)，异或 (xor)

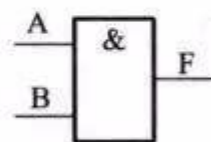


乔治·布尔 (George Boole, 1815. 11. 2~1864)，1815年11月2日生于英格兰的林肯。生前没有人认为他是数学家的伟大数学家。他1847年发表的《逻辑的数学分析》和1854年发表的《思维规律的研究》奠定了符号逻辑的基础，创建了以他的名字命名的布尔代数。

应用到世界中: 逻辑门电路

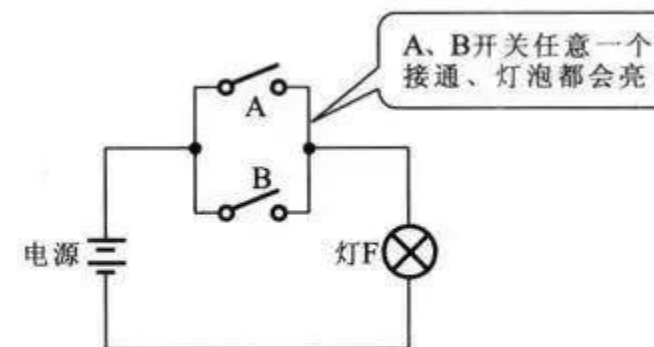


功能示意图

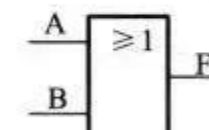


逻辑符号

与门电路的基本结构和逻辑符号

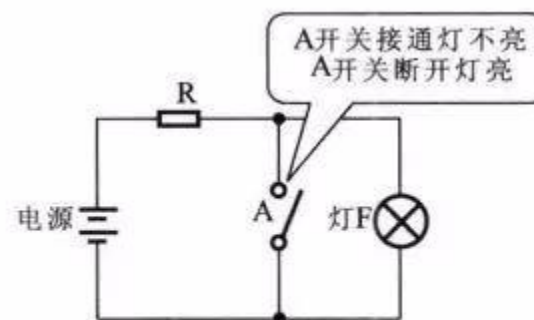


功能示意图

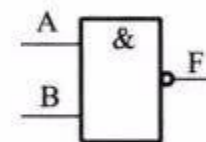


逻辑符号

或门电路的基本结构和逻辑符号



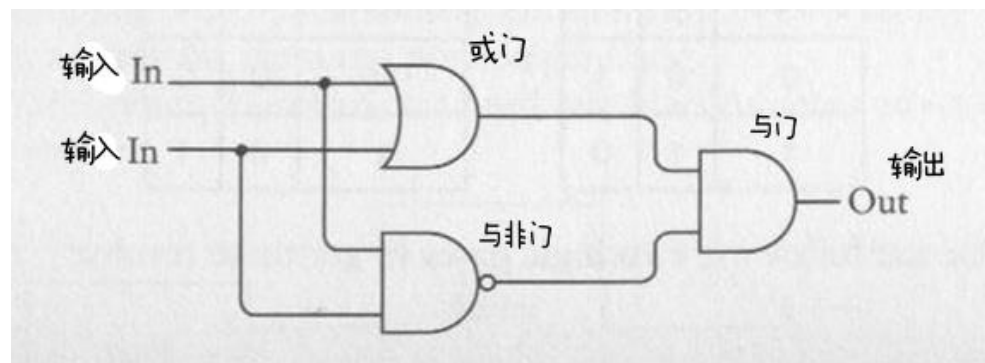
功能示意图



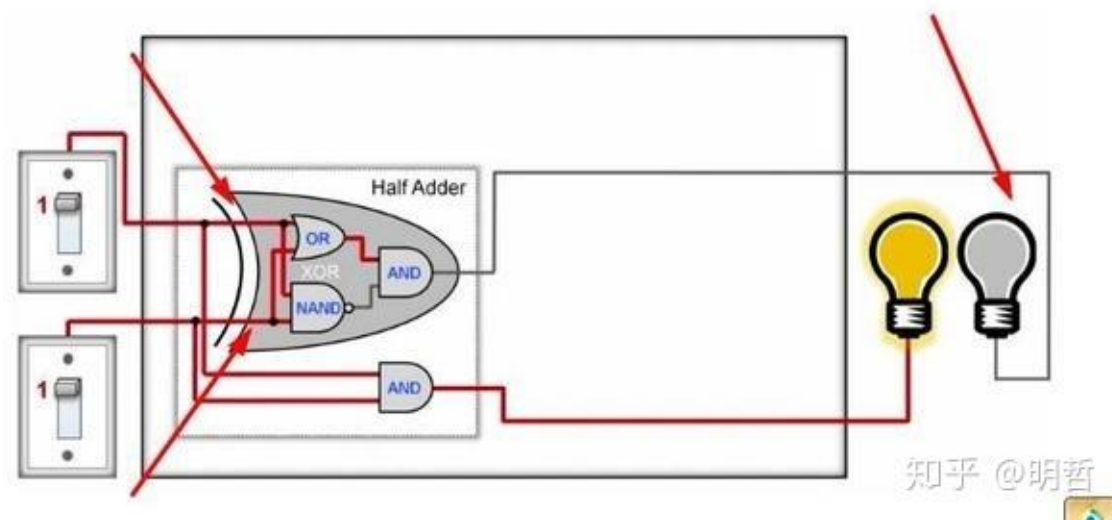
逻辑符号

非门电路的基本结构和逻辑符号

异或门



同时开关不亮，开关各异才亮



知乎 @明哲

与，或，非，异或：逻辑门电路

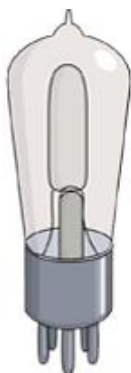
- ▶ 二进制的证明奠定了数字表达、存储和运算基础，布尔代数奠定了逻辑推理（判断）基础，代数微积分等奠定了数据计算的规则。
- ▶ 但数学公式和科学原理，并不能直接产生任何实际的用途，还必须有相应实现的基础元件
- ▶ 继电器、电子管和晶体管的发明，完成了这一基础
 - ▶ 计算机、手机、自动控制系统、人工智能，都是用集成电路——也就是电子芯片组装起来的

- ▶ 集成电路是由几千万、上亿个晶体管组成的（目前到千亿）
 - ▶ 晶体管，就是受控制的开关。可以接收控制信号，改变是否导电。我们可以假设开关接通的状态为'1'，开关不通的状态为'0'，这样就可以用开关电路模拟逻辑状态，组成逻辑电路。
 - ▶ 而最最基本的逻辑电路，我们称为逻辑门。之所以叫 "门"，是因为它能控制电流的路径。逻辑门可以由电阻、电容、二极管、三极管等分立原件构成，
 - ▶ 最基本的逻辑门：与门、或门、非门，对应布尔代数中有三个基本操作：NOT, AND 和 OR
 - ▶ 它们的组合，就形成了人们所需要的各种数字电路
 - ▶ 在编程应用层次，对应逻辑与、或、非，以及位与、位或和位非，以及各种逻辑和位的组合

芯片(比如CPU) 是什么?

- **芯片**就是以**集成电路**为核心的电子技术，它是在**电子元器件小型化、微型化**的过程中发展起来的。

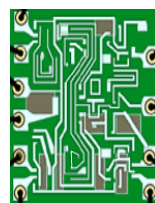
电子管
(1904)



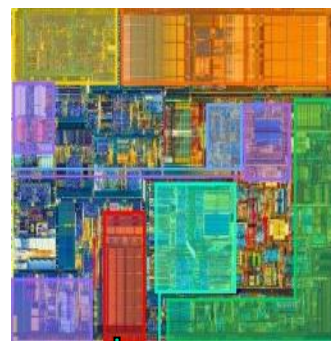
晶体管
(1948)



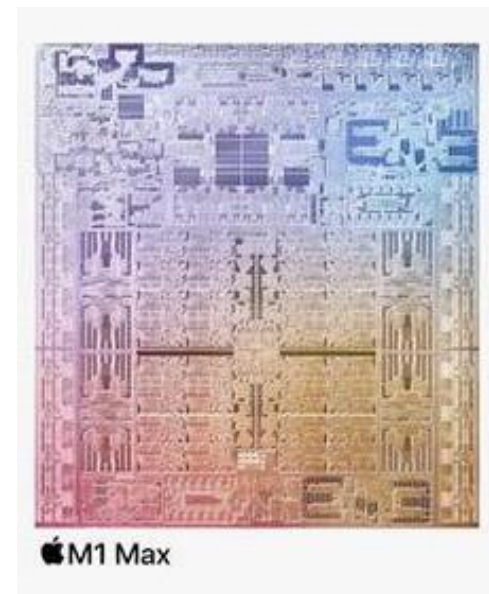
中/小规模
集成电路
(1960' s)



大规模/超大规模
集成电路(1970' s)



电子电路中元器件的发展演变过程



苹果M1 Max芯片，基于5nm工艺，集成570亿个晶体管

位计算的一种应用：求平均值，同时避免进位溢出

如何计算两个数的平均值？

- $(a + b) / 2$

如果涉及进位突破存储上限呢？

- 比如1个十进制的2位数最多表达达到99，那么 $99+99=198$ 超出区域，计算机只会读到98，那么就会得到 $(99+99) / 2=49$
- 二进制也是一样，比如一个整数用32bit存储，不考虑符号，那么就是最大值为 $2^{32}-1$ ，那么 $2^{32}-1$ +任何大于1的数会导致进位，而计算机计算出错
- 但同时你也知道：求得的平均数其实还是在范围内，因此你并不想为变量提升存储空间

如何不进位解决计算？

- ▶ $\text{low} + (\text{high} - \text{low}) / 2$
 - ▶ 需要额外加上判断哪个值大/小

- ▶ $(a / 2) + (b / 2) + (a \& b \& 1)$
 - ▶ $(a+b)/2$ 转换为 $a/2+b/2$
 - ▶ 需要考虑个位丢失的问题，比如说 $1/2$ 在整形运算当中的结果会是0，因此 $1/2+1/2$ 的结果是0而不是1，需要把两个输入的个位提取出来进行修正

SWAR法

- 按位考虑输入值与输出结果的对应关系，那么会有以下的需求要点：
 - 输入都是0，输出结果是0
 - 输入都是1，输出是1
 - 输入是一个0一个1，那么输出结果就是1/2
 - 而满足以上条件的位运算，是“位与”运算加上“位异或”运算除2的结果，
即 $(a \text{ and } b) + (a \text{ xor } b) / 2$ ：
- $(a \& b) + (a \wedge b) / 2$

位运算符与集合运算符

- 集合的交集、并集、对称差集等运算借助于位运算符来实现，而差集则使用减号运算符实现（注意，**并集运算符不是加号**）

>>> {1, 2, 3} {3, 4, 5}	#并集，自动去除重复元素
{1, 2, 3, 4, 5}	
>>> {1, 2, 3} & {3, 4, 5}	#交集
{3}	
>>> {1, 2, 3} ^ {3, 4, 5}	#对称差集
{1, 2, 4, 5}	
>>> {1, 2, 3} - {3, 4, 5}	#差集
{1, 2}	

逻辑运算符

- 逻辑运算符and、or、not常用来连接条件表达式构成更加复杂的条件表达式，并且and和or具有惰性求值或**逻辑短路**的特点，当连接多个表达式时**只计算必须要计算的值**。
- 例如表达式 “exp1 and exp2” 等价于 “exp1 if not exp1 else exp2” ，而表达式 “exp1 or exp2” 则等价于 “exp1 if exp1 else exp2” 。
- 在编写复杂条件表达式时充分利用这个特点，**合理安排不同条件的先后顺序，在一定程度上可以提高代码运行速度**。
- 另外要注意的是，**运算符and和or并不一定会返回True或False，而是得到最后一个被计算的表达式的值，但是运算符not一定会返回True或False**。

逻辑运算符

```
>>> 3>5 and a>3  
False
```

#注意, 此时并没有定义变量a

```
>>> 3>5 or a>3
```

#3>5的值为False, 所以需要计算后面表达式

```
NameError: name 'a' is not defined
```

```
>>> 3<5 or a>3
```

#3<5的值为True, 不需要计算后面表达式

```
True
```

```
>>> 3 and 5
```

#最后一个计算的表达式的值作为整个表达式的值

```
5
```

```
>>> 3 and 5>2
```

```
True
```

```
>>> 3 not in [1, 2, 3]
```

#逻辑非运算not

```
False
```

```
>>> 3 is not 5
```

#not的计算结果只能是True或False之一

```
True
```

```
>>> not 3
```

#大于0的正整数等同于True (0)

```
False
```

```
>>> not 0
```

```
True
```

```
>>> import numpy #numpy是用于科学计算的Python扩展库
>>> x = numpy.ones(3) #ones()函数用于生成全1矩阵，参数表示矩阵大小
>>> m = numpy.eye(3)*3 #eye()函数用于生成单位矩阵
>>> m[0,2] = 5 #设置矩阵指定位置上元素的值
>>> m[2, 0] =3
>>> x @ m #矩阵相乘
array([ 6.,  3.,  8.] )
```

补充说明

- Python还有赋值运算符=和+=、-=、*=、/=、//=、**=、|=、^=等大量复合赋值运算符。例如， $x += 3$ 在语法上等价（注意，在功能的细节上可能会稍有区别）于 $x = x + 3$ 。
- Python不支持++和--运算符，虽然在形式上有时候似乎可以这样用，但实际上是另外的含义，要注意和其他语言的区别。

补充说明

```
>>> i = 3
```

```
>>> ++i
```

```
3
```

```
>>> +(3)
```

```
3
```

```
>>> i++
```

```
SyntaxError: invalid syntax
```

```
>>> --i
```

```
3
```

```
>>> ---i
```

```
-3
```

```
>>> i--
```

```
SyntaxError: invalid syntax
```

#正正得正

#与++i等价

#Python不支持++运算符，语法错误

#负负得正，等价于-(-i)

#等价于-(-(-i))

#Python不支持--运算符，语法错误

Python关键字简要说明

- Python关键字只允许用来表达特定的语义，**不允许**通过任何方式改变它们的含义，也**不能**用来做变量名、函数名或类名等标识符。
- 在Python开发环境中 import keyword之后，可以使用 print(keyword.kwlist)查看所有关键字。

Python关键字简要说明

关键字	含义
False	常量，逻辑假
None	常量，空值
True	常量，逻辑真
and	逻辑与运算
as	在import或except语句中给对象起别名
assert	断言，用来确认某个条件必须满足，可用来帮助调试程序
break	用在循环中，提前结束break所在层次的循环
class	用来定义类
continue	用在循环中，提前结束本次循环
def	用来定义函数
del	用来删除对象或对象成员
elif	用在选择结构中，表示else if的意思
else	可以用在选择结构、循环结构和异常处理结构中
except	用在异常处理结构中，用来捕获特定类型的异常
finally	用在异常处理结构中，用来表示不论是否发生异常都会执行的代码
for	构造for循环，用来迭代序列或可迭代对象中的所有元素
from	明确指定从哪个模块中导入什么对象，例如from math import sin; 还可以与yield一起构成yield表达式

Python关键字简要说明

关键字	含义
global	定义或声明全局变量
if	用在选择结构中
import	用来导入模块或模块中的对象
in	成员测试
is	同一性测试
lambda	用来定义lambda表达式，类似于函数
nonlocal	用来声明nonlocal变量
not	逻辑非运算
or	逻辑或运算
pass	空语句，执行该语句时什么都不做，常用作占位符
raise	用来显式抛出异常
return	在函数中用来返回值，如果没有指定返回值，表示返回空值None
try	在异常处理结构中用来限定可能会引发异常的代码块
while	用来构造while循环结构，只要条件表达式等价于True就重复执行限定的代码块
with	上下文管理，具有自动管理资源的功能
yield	在生成器函数中用来返回值

Python常用内置函数用法精要

- 内置函数是Python内置对象类型之一，**不需要额外导入任何模块即可直接使用**，这些内置对象都封装在内置模块 `__builtins__` 之中，用C语言实现并且进行了大量优化，具有非常快的运行速度，**推荐优先使用**。使用内置函数 `dir()` 可以查看所有内置函数和内置对象：

```
>>> dir(__builtins__)
```

- 使用 `help(函数名)` 可以查看某个函数的用法。

```
>>> help(sum)
```

```
Help on built-in function sum in module builtins:
```

```
sum(iterable, start=0, /)
```

```
Return the sum of a 'start' value (default: 0) plus an iterable of numbers
```

```
When the iterable is empty, return the start value.
```

```
This function is intended specifically for use with numeric values and may reject non-numeric types.
```

Python常用内置函数用法精要

函数	功能简要说明
<code>abs(x)</code>	返回数字x的绝对值或复数x的模
<code>all(iterable)</code>	如果对于可迭代对象中所有元素x都等价于True，也就是对于所有元素x都有 <code>bool(x)</code> 等于True，则返回True。对于空的可迭代对象也返回True
<code>any(iterable)</code>	只要可迭代对象 <code>iterable</code> 中存在元素x使得 <code>bool(x)</code> 为True，则返回True。对于空的可迭代对象，返回False
<code>ascii(obj)</code>	把对象转换为ASCII码表示形式，必要的时候使用转义字符来表示特定的字符
<code>bin(x)</code>	把整数x转换为二进制串表示形式
<code>bool(x)</code>	返回与x等价的布尔值True或False
<code>bytes(x)</code>	生成字节串，或把指定对象x转换为字节串表示形式
<code>callable(obj)</code>	测试对象obj是否可调用。类和函数是可调用的，包含 <code>__call__()</code> 方法的类的对象也是可调用的
<code>compile()</code>	用于把Python代码编译成可被 <code>exec()</code> 或 <code>eval()</code> 函数执行的代码对象
<code>complex(real, [imag])</code>	返回复数
<code>chr(x)</code>	返回Unicode编码为x的字符

Python常用内置函数用法精要

函数	功能简要说明
<code>delattr(obj, name)</code>	删除属性，等价于 <code>del obj.name</code>
<code>dir(obj)</code>	返回指定对象或模块obj的成员列表，如果不带参数则返回当前作用域内所有标识符
<code>divmod(x, y)</code>	返回包含整商和余数的元组 $((x-x\%y)/y, x\%y)$
<code>enumerate(iterable[, start])</code>	返回包含元素形式为 $(0, iterable[0])$, $(1, iterable[1])$, $(2, iterable[2])$, ... 的迭代器对象
<code>eval(s[, globals[, locals]])</code>	计算并返回字符串s中表达式的值
<code>exec(x)</code>	执行代码或代码对象x
<code>exit()</code>	退出当前解释器环境
<code>filter(func, seq)</code>	返回filter对象，其中包含序列seq中使得单参数函数func返回值为True的那些元素，如果函数func为None则返回包含seq中等价于True的元素的filter对象
<code>float(x)</code>	把整数或字符串x转换为浮点数并返回
<code>frozenset([x])</code>	创建不可变的集合对象
<code>getattr(obj, name[, default])</code>	获取对象中指定属性的值，等价于 <code>obj.name</code> ，如果不存在指定属性则返回default的值，如果要访问的属性不存在并且没有指定default则抛出异常

Python常用内置函数用法精要

函数	功能简要说明
<code>globals()</code>	返回包含当前作用域内全局变量及其值的字典
<code>hasattr(obj, name)</code>	测试对象obj是否具有名为name的成员
<code>hash(x)</code>	返回对象x的哈希值，如果x不可哈希则抛出异常
<code>help(obj)</code>	返回对象obj的帮助信息
<code>hex(x)</code>	把整数x转换为十六进制串
<code>id(obj)</code>	返回对象obj的标识（内存地址）
<code>input([提示])</code>	显示提示，接收键盘输入的内容，返回字符串
<code>int(x[, d])</code>	返回实数（float）、分数（Fraction）或高精度实数（Decimal）x的整数部分，或把d进制的字符串x转换为十进制并返回，d默认为十进制
<code>isinstance(obj, class-or-type-or-tuple)</code>	测试对象obj是否属于指定类型（如果有多个类型的话需要放到元组中）的实例
<code>iter(...)</code>	返回指定对象的可迭代对象
<code>len(obj)</code>	返回对象obj包含的元素个数，适用于列表、元组、集合、字典、字符串以及range对象和其他可迭代对象

Python常用内置函数用法精要

函数	功能简要说明
<code>list([x])</code> 、 <code>set([x])</code> 、 <code>tuple([x])</code> 、 <code>dict([x])</code>	把对象x转换为列表、集合、元组或字典并返回，或生成空列表、空集合、空元组、空字典
<code>locals()</code>	返回包含当前作用域内局部变量及其值的字典
<code>map(func, *iterables)</code>	返回包含若干函数值的map对象，函数func的参数分别来自于iterables指定的每个迭代对象，
<code>max(x)</code> 、 <code>min(x)</code>	返回可迭代对象x中的最大值、最小值，要求x中的所有元素之间可比较大小，允许指定排序规则和x为空时返回的默认值
<code>next(iterator[, default])</code>	返回可迭代对象x中的下一个元素，允许指定迭代结束之后继续迭代时返回的默认值
<code>oct(x)</code>	把整数x转换为八进制串
<code>open(name[, mode])</code>	以指定模式mode打开文件name并返回文件对象
<code>ord(x)</code>	返回1个字符x的Unicode编码
<code>pow(x, y, z=None)</code>	返回x的y次方，等价于 <code>x ** y</code> 或 <code>(x ** y) % z</code>

Python常用内置函数用法精要

函数	功能简要说明
<code>print(value, ..., sep=' ', end=' \n', file=sys.stdout, flush=False)</code>	基本输出函数
<code>quit()</code>	退出当前解释器环境
<code>range([start,] end [, step])</code>	返回range对象，其中包含左闭右开区间[start, end)内以step为步长的整数
<code>reduce(func, sequence[, initial])</code>	将双参数的函数func以迭代的方式从左到右依次应用至序列seq中每个元素，最终返回单个值作为结果。在Python 2.x中该函数为内置函数，在Python 3.x中需要从functools中导入reduce函数再使用
<code>repr(obj)</code>	返回对象obj的规范化字符串表示形式，对于大多数对象有 <code>eval(repr(obj))==obj</code>
<code>reversed(seq)</code>	返回seq（可以是列表、元组、字符串、range以及其他可迭代对象）中所有元素逆序后的迭代器对象

Python常用内置函数用法精要

函数	功能简要说明
<code>round(x [, 小数位数])</code>	对x进行四舍五入，若不指定小数位数，则返回整数
<code>sorted(iterable, key=None, reverse=False)</code>	返回排序后的列表，其中iterable表示要排序的序列或迭代对象，key用来指定排序规则或依据，reverse用来指定升序或降序。该函数不改变iterable内任何元素的顺序
<code>str(obj)</code>	把对象obj直接转换为字符串
<code>sum(x, start=0)</code>	返回序列x中所有元素之和，返回start+sum(x)
<code>type(obj)</code>	返回对象obj的类型
<code>zip(seq1 [, seq2 [...]])</code>	返回zip对象，其中元素为(seq1[i], seq2[i], ...)形式的元组，最终结果中包含的元素个数取决于所有参数序列或可迭代对象中最短的那个

类型转换与类型判断

- 内置函数int()用来将其他形式的数字转换为整数，参数可以为整数、实数、分数或合法的数字字符串。当参数为数字字符串时，还允许指定第二个参数base用来说明数字字符串的进制，**base的取值应为0或2-36之间的整数**，其中0表示按数字字符串隐含的进制进行转换

```
>>> int(-3.2)                #把实数转换为整数
```

```
-3
```

```
>>> from fractions import Fraction, Decimal
```

```
>>> x = Fraction(7, 3)
```

```
>>> x
```

```
Fraction(7, 3)
```

```
>>> int(x)                    #把分数转换为整数
```

```
2
```

```
>>> x = Decimal(10/3)
```

```
>>> x
```

```
Decimal('3.3333333333333333481363069950020872056484222412109375')
```

```
>>> int(x)                    #把高精度实数转换为整数
```

```
3
```

```
>>> hex(11)
```

```
'0xb'
```

```
>>> int('0xb',16)
```

```
11
```

类型转换与类型判断

```
>>> int('0x22b', 16)
```

```
555
```

#把十六进制数转换为十进制数

```
>>> int('22b', 16)
```

```
555
```

#与上一行代码等价

```
>>> int(bin(54321), 2)
```

```
54321
```

#二进制与十进制之间的转换

```
>>> int('0b111')
```

```
ValueError: invalid literal for int() with base 10: '0b111'
```

#非十进制字符串进，必须指定第二个参数

```
>>> int('0b111', 0)
```

```
7
```

#第二个参数0表示使用字符串隐含的进制

```
>>> int('0b111', 6)
```

```
ValueError: invalid literal for int() with base 6: '0b111'
```

#第二个参数应与隐含的进制一致

```
>>> int('0b111', 2)
```

```
7
```

```
>>> int('111', 6)
```

#字符串没有隐含进制

#第二个参数可以为2-36之间的数字

```
43
```

类型转换与类型判断

- 内置函数float()用来将其他类型数据转换为实数，complex()可以用来生成复数。

```
>>> float(3)                #把整数转换为实数
3.0
>>> float('3.5')           #把数字字符串转换为实数
3.5
>>> float('inf')            #无穷大，其中inf不区分大小写
inf
>>> complex(3)              #指定实部
(3+0j)
>>> complex(3, 5)           #指定实部和虚部
(3+5j)
>>> complex('inf')         #无穷大
(inf+0j)
```

基本输入输出

- input()和print()是Python的基本输入输出函数，前者用来接收用户的键盘输入，后者用来把数据以指定的格式输出到标准控制台或指定的文件对象。
- 不论用户输入什么内容，input()一律返回字符串对待，必要的时候可以使用内置函数int()、float()或eval()对用户输入的内容进行类型转换。

基本输入输出

```
>>> x = input('Please input: ')
```

```
Please input: 345
```

```
>>> x
```

```
'345'
```

```
>>> type(x)
```

```
<class 'str'>
```

#把用户的输入作为字符串对待

```
>>> int(x)
```

#转换为整数

```
345
```

```
>>> eval(x)
```

#对字符串求值，或类型转换

```
345
```

```
>>> x = input('Please input: ')
```

```
Please input: [1, 2, 3]
```

```
>>> x
```

```
'[1, 2, 3]'
```

```
>>> type(x)
```

```
<class 'str'>
```

```
>>> eval(x)
```

```
[1, 2, 3]
```

基本输入输出

- 内置函数print()用于输出信息到标准控制台或指定文件，语法格式为：
`print(value1, value2, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
- ✓ sep参数之前为需要输出的内容（可以有多个）；
- ✓ sep参数用于指定数据之间的分隔符，默认为空格；
- ✓ end参数用于指定输出完数据之后再输出什么字符；
- ✓ file参数用于指定输出位置，默认为标准控制台，也可以重定向输出到文件。

```
>>> print(1, 3, 5, 7, sep='\t')           #修改默认分隔符
1      3      5      7
>>> for i in range(10):                   #修改end参数，每个输出之后不换行
    print(i, end=' ')
0 1 2 3 4 5 6 7 8 9
>>> with open('test.txt', 'a+') as fp:
    print('Hello world!', file=fp)        #重定向，将内容输出到文件中
```

数据结构

除了数据，还有数据之间的关系

“学生” 表格

	学 号	姓 名	性别	籍 贯	出生年月
1	98131	刘激扬	男	北 京	1979.12
2	98164	衣春生	男	青 岛	1979.07
3	98165	卢声凯	男	天 津	1981.02
4	98182	袁秋慧	女	广 州	1980.10
5	98203	林德康	男	上 海	1980.05
6	98224	洪 伟	男	太 原	1981.01
7	98236	熊南燕	女	苏 州	1980.03
8	98297	宫 力	男	北 京	1981.01
9	98310	蔡晓莉	女	昆 明	1981.02
10	98318	陈 健	男	杭 州	1979.12

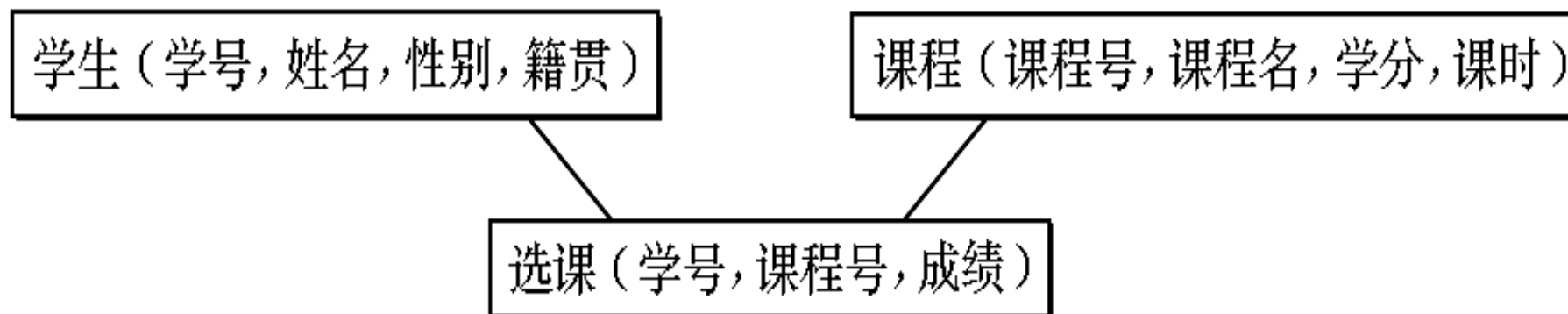
“课程” 表格

课程编号	课 程 名	学时
024002	程序设计基础	64
024010	汇编语言	48
024016	计算机原理	64
024020	数据结构	64
024021	微机技术	64
024024	操作系统	48
024026	数据库原理	48

选课单包含如下信息

学 号 课程编号 成 绩 时 间

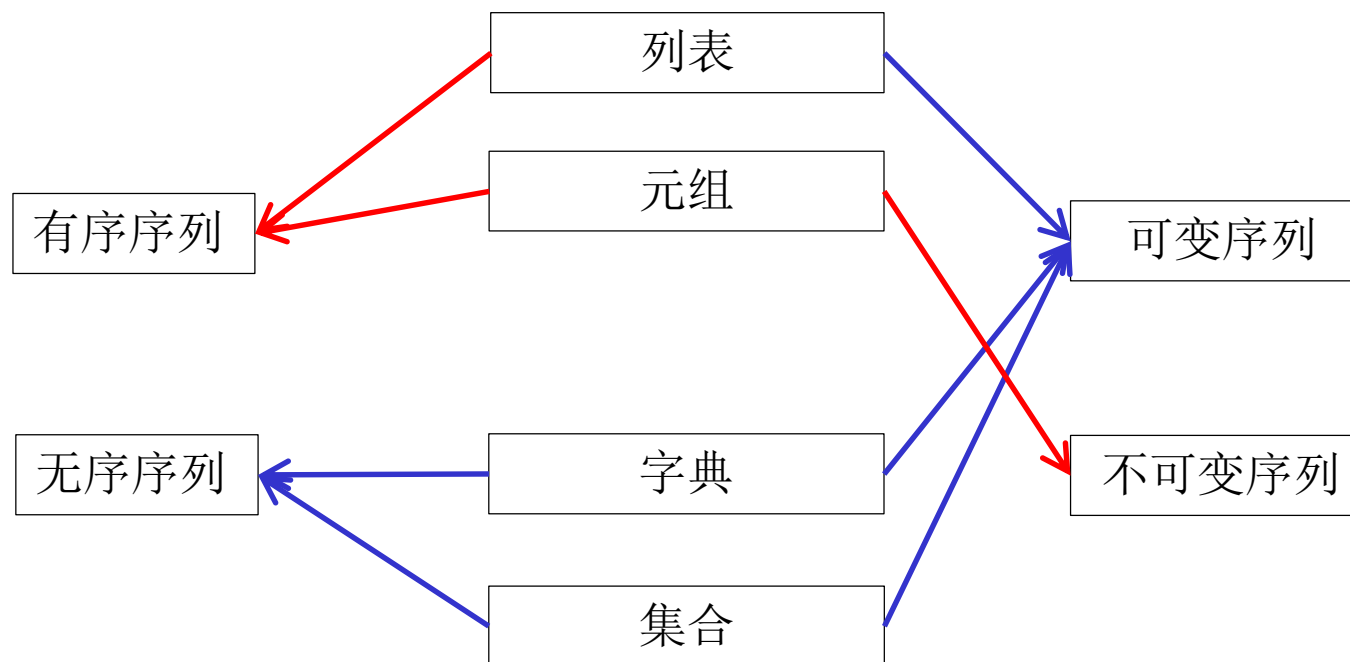
学生选课系统中实体构成的网状关系



基础数据结构：列表、元组、字典、集合

	列表	元组	字典	集合
类型名称	<code>list</code>	<code>tuple</code>	<code>dict</code>	<code>set</code>
定界符	方括号 []	圆括号 ()	大括号 {}	大括号 {}
是否可变	是	否	是	是
是否有序	是	是	否	否
是否支持下标	是（使用序号作为下标）	是（使用序号作为下标）	是（使用“键”作为下标）	否
元素分隔符	逗号	逗号	逗号	逗号
对元素形式的要求	无	无	键:值	必须可哈希
对元素值的要求	无	无	“键”必须可哈希	必须可哈希
元素是否可重复	是	是	“键”不允许重复，“值”可以重复	否
元素查找速度	非常慢	很慢	非常快	非常快
新增和删除元素速度	尾部操作快 其他位置慢	不允许	快	快

Python自带基础数据结构



列表

- ▶ 列表 (list) 是最重要的Python内置对象之一，是包含若干元素的有序连续内存空间。当列表增加或删除元素时，列表对象自动进行内存的扩展或收缩，从而保证相邻元素之间没有缝隙。
 - ▶ Python列表的这个内存自动管理功能可以大幅度减少程序员的负担，但插入和删除非尾部元素时涉及到列表中大量元素的移动，会严重影响效率
- ▶ 在非尾部位置插入和删除元素时会改变该位置后面的元素在列表中的索引，这对于某些操作可能会导致意外的错误结果。
- ▶ 除非确实有必要，否则应尽量从列表尾部进行元素的追加与删除操作。

列表

- 在形式上，列表的所有元素放在一对**方括号**[]中，相邻元素之间使用**逗号**分隔。
- 在Python中，**同一个列表中元素的数据类型可以各不相同**，可以同时包含整数、实数、字符串等基本类型的元素，也可以包含列表、元组、字典、集合、函数以及其他任意对象。简单来说，可以嵌套
- 如果只有一对方括号而没有任何元素则表示空列表。

```
[10, 20, 30, 40]
```

```
['crunchy frog', 'ram bladder', 'lark vomit']
```

```
['spam', 2.0, 5, [10, 20]]
```

```
[['file1', 200,7], ['file2', 260,9]]
```

```
[{3}, {5:6}, (1, 2, 3)]
```

列表创建与删除

- 使用 “=” 直接将一个列表赋值给变量即可创建列表对象。

```
>>> a_list = ['a', 'b', 'mpilgrim', 'z', 'example']  
>>> a_list = [] #创建空列表
```

```
left=['警察','犯人','父亲','儿子','儿子','母亲','女儿','女儿']
```

```
right=[]
```

```
ship=[]
```

```
# 或者写成一行
```

```
left, right, ship=['警察','犯人','父亲','儿子','儿子','母亲','女儿','女儿'],[],[]
```

列表创建与删除

- 也可以使用list()函数把元组、range对象、字符串、字典、集合或其他可迭代对象转换为列表。

```
>>> list((3,5,7,9,11))           #将元组转换为列表
[3, 5, 7, 9, 11]
>>> list(range(1, 10, 2))        #将range对象转换为列表
[1, 3, 5, 7, 9]
>>> list('hello world')         #将字符串转换为列表
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> list({3,7,5})               #将集合转换为列表
[3, 5, 7]
>>> list({'a':3, 'b':9, 'c':78}) #等后面再讲将字典的“键”转换为列表
['a', 'c', 'b']
>>> list({'a':3, 'b':9, 'c':78}.items()) #将字典的“键:值”对转换为列表
[('b', 9), ('c', 78), ('a', 3)]
>>> x = list()                  #创建空列表
```


列表创建与删除

- 当一个列表不再使用时，可以使用del命令将其删除，这一点适用于所有类型的Python对象。

```
>>> x = [1, 2, 3]
```

```
>>> del x
```

#删除列表对象

```
>>> x
```

#对象删除后无法再访问，抛出异常

```
NameError: name 'x' is not defined
```

列表元素访问

- 创建列表之后，可以使用**整数**作为下标来访问其中的元素，其中**0表示第1个元素**，1表示第2个元素，2表示第3个元素，以此类推；
- 列表还支持使用负整数作为下标，其中**-1表示最后1个元素**，-2表示倒数第2个元素，-3表示倒数第3个元素，以此类推

```
>>> x = list('Python')           #创建类别对象
>>> x
['P', 'y', 't', 'h', 'o', 'n']
>>> x[0]                          #下标为0的元素，第一个元素
'P'
>>> x[-1]                        #下标为-1的元素，最后一个元素
'n'
```

+	+	+	+	+	+	+	+
	'P'		'y'		't'		'h'
+	+	+	+	+	+	+	+
0	1	2	3	4	5	6	
-6	-5	-4	-3	-2	-1		

列表常用方法

方法	说明
<code>append(x)</code>	将x追加至列表尾部
<code>extend(L)</code>	将列表L中所有元素追加至列表尾部
<code>insert(index, x)</code>	在列表index位置处插入x，该位置后面的所有元素后移并且在列表中的索引加1，如果index为正数且大于列表长度则在列表尾部追加x，如果index为负数且小于列表长度的相反数则在列表头部插入元素x
<code>remove(x)</code>	在列表中删除第一个值为x的元素，该元素之后所有元素前移并且索引减1，如果列表中不存在x则抛出异常
<code>pop([index])</code>	删除并返回列表中下标为index的元素，如果不指定index则默认为-1，弹出最后一个元素；如果弹出中间位置的元素则后面的元素索引减1；如果index不是[-L, L]区间上的整数则抛出异常
<code>clear()</code>	清空列表，删除列表中所有元素，保留列表对象
<code>index(x)</code>	返回列表中第一个值为x的元素的索引，若不存在值为x的元素则抛出异常
<code>count(x)</code>	返回x在列表中的出现次数
<code>reverse()</code>	对列表所有元素进行原地逆序，首尾交换
<code>sort(key=None, reverse=False)</code>	对列表中的元素进行原地排序，key用来指定排序规则，reverse为False表示升序，True表示降序
<code>copy()</code>	返回列表的浅复制

列表常用方法

(1) append()、insert()、extend()
append()用于向列表尾部追加一个元素，insert()用于向列表任意指定位置插入一个元素，extend()用于将另一个列表中的**所有元素**追加至当前列表的尾部。这3个方法都属于**原地操作**，不影响列表对象在内存中的起始地址

```
>>> x = [1, 2, 3]
>>> id(x)
50159368
>>> x.append(4)
>>> x.insert(0, 0)
>>> x.extend([5, 6, 7])
>>> x
[0, 1, 2, 3, 4, 5, 6, 7]
>>> id(x)
50159368
```

#查看对象的内存地址

#在尾部追加元素
#在指定位置插入元素
#在尾部追加多个元素

#列表在内存中的地址不变

```
>>> x = [1, 2, 3, 4, 5, 6, 7]
>>> x.pop()
7
>>> x.pop(0)
1
>>> x.clear()
>>> x
[]
>>> x = [1, 2, 1, 1, 2]
>>> x.remove(2)
>>> del x[3]
>>> x
[1, 1, 1]
```

- ▶ left, right, ship=['警察','犯人','父亲','儿子','儿子','母亲','女儿','女儿'],[],[]
- ▶ #操作两岸和船的状态
- ▶ ship.append(), left.pop(), right.append()
- ▶ #假设i, j是记录每次登船的人
- ▶ ship.append(left.pop(i))
- ▶ ship.append(left.pop(j))
- ▶ #如果满足两岸和划船的条件
- ▶ for k in ship:
- ▶ right.append(k)

列表常用方法

(3) count()、index()

列表方法count()用于返回列表中指定元素出现的次数；index()用于返回指定元素在列表中**首次出现的位置**，如果该元素不在列表中则抛出异常

```
>>> x = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
>>> x.count(3)                #元素3在列表x中的出现次数
3
>>> x.count(5)                #不存在, 返回0
0
>>> x.index(2)                #元素2在列表x中首次出现的索引
1
>>> x.index(5)                #列表x中没有5, 抛出异常
ValueError: 5 is not in list
```

列表常用方法

(4) sort()、reverse()

列表对象的sort()方法用于按照指定的规则对所有元素进行排序；reverse()方法用于将列表所有元素逆序或翻转

```
>>> x = list(range(11))          #包含11个整数的列表
>>> import random
>>> random.shuffle(x)           #把列表x中的元素随机乱序
>>> x
[6, 0, 1, 7, 4, 3, 2, 8, 5, 10, 9]
>>> x.sort(key=lambda item:len(str(item)), reverse=True)  #p.105,按转换成字符串以后的长度,
降序排列
>>> x
[10, 6, 0, 1, 7, 4, 3, 2, 8, 5, 9]
>>> x.sort(key=str)             #按转换为字符串后的大小, 升序排序
>>> x
[0, 1, 10, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x.sort()                    #按默认规则排序
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> x.reverse()                #把所有元素翻转或逆序
>>> x
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```


列表对象支持的运算符

- 加法运算符+也可以实现列表增加元素的目的，但不属于原地操作，而是**返回新列表**，涉及大量元素的复制，效率非常低。使用复合赋值运算符+=实现列表追加元素时属于原地操作，与append()方法一样高效

```
>>> x = [1, 2, 3]
```

```
>>> id(x)
```

```
53868168
```

#获取对象的内存地址 help(id)

```
>>> x = x + [4]
```

#连接两个列表

```
>>> x
```

```
[1, 2, 3, 4]
```

```
>>> id(x)
```

#内存地址发生改变

```
53875720
```

```
>>> x += [5]
```

#为列表追加元素

```
>>> x
```

```
[1, 2, 3, 4, 5]
```

```
>>> id(x)
```

#内存地址不变

```
53875720
```

列表对象支持的运算符

- 乘法运算符*可以用于列表和整数相乘，表示序列重复，返回新列表。运算符*=也可以用于列表元素重复，属于原地操作

```
>>> x = [1, 2, 3, 4]
```

```
>>> id(x)
```

```
54497224
```

```
>>> x = x * 2
```

#元素重复，返回新列表

```
>>> x
```

```
[1, 2, 3, 4, 1, 2, 3, 4]
```

```
>>> id(x)
```

#地址发生改变

```
54603912
```

```
>>> x *= 2
```

#元素重复，原地进行

```
>>> x
```

```
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

```
>>> id(x)
```

#地址不变

```
54603912
```

列表对象支持的运算符

- 成员测试运算符in可用于测试列表中是否包含某个元素，**查询时间随着列表长度的增加而线性增加**，而同样的操作对于集合而言则是常数级的。

```
>>> 3 in [1, 2, 3]
```

```
True
```

```
>>> 3 in [1, 2, '3']
```

```
False
```

内置函数对列表的操作

- ▶ `max()`、`min()`函数用于返回列表中所有元素的最大值和最小值
- ▶ `sum()`函数用于返回列表中所有元素之和;
- ▶ `len()`函数用于返回列表中元素个数, `zip()`函数用于将多个列表中元素重新组合为元组并返回包含这些元组的zip对象;
- ▶ `enumerate()`函数返回包含若干下标和值的迭代对象;
- ▶ `map()`函数把函数映射到列表上的每个元素, `filter()`函数根据指定函数的返回值对列表元素进行过滤;
- ▶ `all()`函数用来测试列表中是否所有元素都等价于True, `any()`用来测试列表中是否有等价于True的元素。
- ▶ 标准库functools中的`reduce()`函数以及标准库itertools中的`compress()`、`groupby()`、`dropwhile()`等大量函数也可以对列表进行操作。

内置函数对列表的操作

```
>>> x = list(range(11))
```

#生成列表

```
>>> import random
```

```
>>> random.shuffle(x)
```

#打乱列表中元素顺序

```
>>> x
```

```
[0, 6, 10, 9, 8, 7, 4, 5, 2, 1, 3]
```

```
>>> all(x)
```

#测试是否所有元素都等价于True (即>0正整数)

```
False
```

```
>>> any(x)
```

#测试是否存在等价于True的元素

```
True
```

```
>>> max(x)
```

#返回最大值

```
10
```

```
>>> max(x, key=str)
```

#按指定规则返回最大值

```
9
```

```
>>> min(x)
```

```
0
```

```
>>> all([1,2,-1,0.3,-0.3,'a'])
```

#Python任何非0数均认为True, 0/0.0/为False

```
True
```

内置函数对列表的操作

```
>>> sum(x)                                #所有元素之和
55

>>> len(x)                                #列表元素个数
11

>>> list(zip(x, [1]*11))                   #多列表元素重新组合
[(0, 1), (6, 1), (10, 1), (9, 1), (8, 1), (7, 1), (4, 1), (5, 1), (2, 1),
(1, 1), (3, 1)]

>>> list(zip(range(1,4)))                  #zip()函数也可以用于一个序列或迭代对象
[(1,), (2,), (3,)]

>>> list(zip(['a', 'b', 'c'], [1, 2]))      #如果两个列表不等长, 以短的为准
[('a', 1), ('b', 2)]

>>> enumerate(x)                          #枚举列表元素, 返回enumerate对象
<enumerate object at 0x00000000030A9120>

>>> list(enumerate(x))                     #enumerate对象可以转换为列表、元组、集合
[(0, 0), (1, 6), (2, 10), (3, 9), (4, 8), (5, 7), (6, 4), (7, 5), (8, 2),
(9, 1), (10, 3)]
```

Python, 你又顽皮了: 切片操作的强大功能

- 在形式上, 切片使用2个冒号分隔的3个数字来完成。

[start:end:step]

- ✓ 第一个数字start表示切片开始位置, 默认为0;
- ✓ 第二个数字end表示切片截止 (但不包含) 位置 (默认为列表长度);
- ✓ 第三个数字step表示切片的步长 (默认为1)。
- ✓ 当start为0时可以省略, 当end为列表长度时可以省略, 当step为1时可以省略, 省略步长时还可以同时省略最后一个冒号。
- ✓ 当step为负整数时, 表示反向切片, 这时start应该在end的右侧才行。

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList[::-1] #返回包含原列表中所有元素的新列表
```

切片操作的强大功能

(1) 使用切片获取列表部分元素

使用切片可以返回列表中部分元素组成的**新列表**。与使用索引作为下标访问列表元素的方法不同，切片操作不会因为下标越界而抛出异常，而是简单地在列表尾部截断或者返回一个空列表，**代码具有更强的健壮性**。

切片操作的强大功能

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList[:]           #返回包含原列表中所有元素的新列表
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList[::-1]        #返回包含原列表中所有元素的逆序列表
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
>>> aList[::2]         #隔一个取一个，获取偶数位置的元素
[3, 5, 7, 11, 15]
>>> aList[1::2]        #隔一个取一个，获取奇数位置的元素
[4, 6, 9, 13, 17]
>>> aList[3:6]         #指定切片的开始和结束位置
[6, 7, 9]
```

切片操作的强大功能

```
>>> aList[0:100]           #切片结束位置大于列表长度时，从列表尾部
截断
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList[100]             #抛出异常，不允许越界访问
IndexError: list index out of range
>>> aList[100:]            #切片开始位置大于列表长度时，返回空列表
[]
>>> aList[-15:3]           #进行必要的截断处理
[3, 4, 5]
>>> len(aList)
10
>>> aList[3:-10:-1]        #位置3在位置-10的右侧，-1表示反向切片
[6, 5, 4]
>>> aList[3:-5]            #位置3在位置-5的左侧，正向切片
[6, 7]
```

切片操作的强大功能

(2) 使用切片为列表增加元素

可以使用切片操作在列表任意位置插入新元素，不影响列表对象的内存地址，属于原地操作。

```
>>> aList = [3, 5, 7]
>>> aList[len(aList):]
```

```
[ ]
```

```
>>> aList[len(aList):] = [9]
```

#在列表尾部增加元素

```
>>> aList[:0] = [1, 2]
```

#在列表头部插入多个元素

```
>>> aList[3:3] = [4]
```

#在列表中间位置插入元素

```
>>> aList
```

```
[1, 2, 3, 4, 5, 7, 9]
```

切片操作的强大功能

(3) 使用切片替换和修改列表中的元素

```
>>> aList = [3, 5, 7, 9]
```

```
>>> aList[:3] = [1, 2, 3]
```

#替换列表元素，等号两边的列表长度相等

```
>>> aList
```

```
[1, 2, 3, 9]
```

```
>>> aList[3:] = [4, 5, 6]
```

#切片连续，等号两边的列表长度可以不相等

```
>>> aList
```

```
[1, 2, 3, 4, 5, 6]
```

```
>>> aList[::2] = [0]*3
```

#隔一个修改一个

```
>>> aList
```

```
[0, 2, 0, 4, 0, 6]
```

```
>>> aList[::2] = ['a', 'b', 'c']
```

#隔一个修改一个

```
>>> aList
```

```
['a', 2, 'b', 4, 'c', 6]
```

切片操作的强大功能

```
>>> aList[1::2] = range(3)                                #序列解包的用法
>>> aList
['a', 0, 'b', 1, 'c', 2]
>>> aList[1::2] = map(lambda x: x!=5, range(3))
>>> aList
['a', True, 'b', True, 'c', True]
>>> aList[1::2] = zip('abc', range(3)) #map、filter、zip对象都支持这样的用法
>>> aList
['a', ('a', 0), 'b', ('b', 1), 'c', ('c', 2)]
>>> aList[::2] = [1]                                       #切片不连续时等号两边列表长度必须相等
ValueError: attempt to assign sequence of size 1 to extended slice of
size 3
```

切片操作的强大功能

(4) 使用切片删除列表中的元素

```
>>> aList = [3, 5, 7, 9]
```

```
>>> aList[:3] = []
```

#删除列表中前3个元素

```
>>> aList
```

```
[9]
```

也可以结合使用del命令与切片结合来删除列表中的部分元素，并且切片元素可以不连续。

```
>>> aList = [3, 5, 7, 9, 11]
```

```
>>> del aList[:3]
```

#切片元素连续

```
>>> aList
```

```
[9, 11]
```

```
>>> aList = [3, 5, 7, 9, 11]
```

```
>>> del aList[::2]
```

#切片元素不连续，隔一个删一个

```
>>> aList
```

```
[5, 9]
```

元组：轻量级列表

- 列表的功能虽然很强大，但负担也很重，在很大程度上影响了运行效率。有时候我们并不需要那么多功能，很希望能有个轻量级的列表，元组 (tuple) 正是这样一种类型。
- 从形式上，元组的所有元素放在一对圆括号中，元素之间使用逗号分隔，如果元组中只有一个元素则必须在最后增加一个逗号。

元组创建与元素访问

```
>>> x = (1, 2, 3)
```

```
>>> type(x)
```

```
<class 'tuple'>
```

```
>>> x[0]
```

```
1
```

```
>>> x[-1]
```

```
3
```

```
>>> x[1] = 4
```

```
TypeError: 'tuple' object does not support item assignment
```

```
>>> x = (3)
```

```
>>> x
```

```
3
```

```
>>> x = (3,)
```

```
>>> x
```

```
(3,)
```

#直接把元组赋值给一个变量

#使用type()函数查看变量类型

#元组支持使用下标访问特定位置的元素

#最后一个元素，元组也支持双向索引

#元组是不可变的

#这和x = 3是一样的

#如果元组中只有一个元素，必须在后面多写一个逗号

元组创建与元素访问

```
>>> x = ()           #空元组
>>> x = tuple()      #空元组
>>> tuple(range(5))  #将其他迭代对象转换为元组
(0, 1, 2, 3, 4)
```

元组与列表的异同点

- 列表和元组都属于有序序列，都支持使用双向索引访问其中的元素，以及使用count()方法统计指定元素的出现次数和index()方法获取指定元素的索引，len()、map()、filter()等大量内置函数和+、+=、in等运算符也都可以作用于列表和元组。

元组与列表的异同点

- 元组属于**不可变** (immutable) 序列，不可以直接修改元组中元素的值，也无法为元组增加或删除元素。
- 元组没有提供append()、extend()和insert()等方法，无法向元组中添加元素；同样，元组也没有remove()和pop()方法，也不支持对元组元素进行del操作，不能从元组中删除元素，而只能使用del命令删除整个元组。
- 元组也支持切片操作，但是只能通过切片来访问元组中的元素，而不允许使用切片来修改元组中元素的值，也不支持使用切片操作来为元组增加或删除元素。

元组与列表的异同点

- Python的内部实现对元组做了大量优化，访问速度比列表更快。如果定义了一系列常量值，主要用途仅是对它们进行遍历或其他类似用途，而不需要对其元素进行任何修改，那么一般建议使用元组而不用列表。
- 元组在内部实现上不允许修改其元素值，从而使得代码更加安全，例如调用函数时使用元组传递参数可以防止在函数中修改元组，而使用列表则很难保证这一点。

字典：反映对应关系的映射类型

- 字典 (dictionary) 是包含若干 “键:值” 元素的**无序可变序列**，字典中的**每个元素包含用冒号分隔开的 “键” 和 “值” 两部分**，表示一种映射或对应关系，也称关联数组。定义字典时，每个元素的 “键” 和 “值” 之间用冒号分隔，不同元素之间用逗号分隔，所有的元素放在一对**大括号** “ {} ” 中。
- 字典中元素的 “键” 可以是Python中任意不可变数据，例如整数、实数、复数、字符串、元组等类型等可哈希数据，但不能使用列表、集合、字典或其他可变类型作为字典的 “键”。另外，字典中的 “键” **不允许重复**，而 “值” 是可以重复的。

字典创建与删除

- 使用赋值运算符 “=” 将一个字典赋值给一个变量即可创建一个字典变量。

```
>>> aDict = {'server': 'db.diveintopython3.org', 'database': 'mysql'}
```

- 也可以使用内置类dict以不同形式创建字典。

```
>>> x = dict()
```

#空字典

```
>>> type(x)
```

#查看对象类型

```
<class 'dict'>
```

```
>>> x = {}
```

#空字典

```
>>> keys = ['a', 'b', 'c', 'd']
```

```
>>> values = [1, 2, 3, 4]
```

```
>>> dictionary = dict(zip(keys, values))
```

#根据已有数据创建字典

```
>>> dictionary
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

```
>>> d = dict(name='Dong', age=39)
```

#以关键参数的形式创建字典

```
>>> d
```

```
{'name': 'Dong', 'age': 39}
```

```
>>> aDict = dict.fromkeys(['name', 'age', 'sex ']) #以给定内容为“键”，创建“值”为空的字典
```

```
>>> aDict
```

```
{'age': None, 'name': None, 'sex': None}
```

字典元素的访问

- 字典中的每个元素表示一种映射关系或对应关系，根据提供的“键”作为下标就可以访问对应的“值”，如果字典中不存在这个“键”会抛出异常。

```
>>> aDict = {'age': 39, 'score': [98, 97], 'name': 'Dong', 'sex': 'male'}
```

```
>>> aDict['age'] #指定的“键”存在，返回对应的“值”
```

```
39
```

```
>>> aDict['address'] #指定的“键”不存在，抛出异常
```

```
KeyError: 'address'
```

字典元素的访问

- 字典对象提供了一个get()方法用来返回指定“键”对应的“值”，并且允许指定该键不存在时返回特定的“值”。例如：

```
>>> aDict.get('age') #如果字典中存在该“键” 则返回对应的“值”  
39  
>>> aDict.get('address', 'Not Exists.') #指定的“键” 不存在时返回指定的默认值  
'Not Exists.'
```

- 使用字典对象的items()方法可以返回字典的键、值对。
- 使用字典对象的keys()方法可以返回字典的键。
- 使用字典对象的values()方法可以返回字典的值。

元素添加、修改与删除

- 当以指定“键”为下标为字典元素赋值时，有两种含义：
 - 1) 若该“键”存在，则表示修改该“键”对应的值；
 - 2) 若不存在，则表示添加一个新的“键:值”对，也就是添加一个新元素。

```
>>> aDict = {'age': 35, 'name': 'Dong', 'sex': 'male'}
>>> aDict['age'] = 39                                #修改元素值
>>> aDict
{'age': 39, 'name': 'Dong', 'sex': 'male'}
>>> aDict['address'] = 'SDIBT'                        #添加新元素
>>> aDict
{'age': 39, 'address': 'SDIBT', 'name': 'Dong', 'sex': 'male'}
```

元素添加、修改与删除

- 使用字典对象的update()方法可以将另一个字典的“键:值”一次性全部添加到当前字典对象, 如果两个字典中存在相同的“键”, 则以另一个字典中的“值”为准对当前字典进行更新。

```
>>> aDict = {'age': 37, 'score': [98, 97], 'name': 'Dong', 'sex': 'male'}  
>>> aDict.update({'a':97, 'age':39}) #修改' age' 键的值, 同时添加新元素' a' :97  
>>> aDict  
{ 'age': 39, 'score': [98, 97], 'name': 'Dong', 'sex': 'male', 'a': 97}  
# 新元素排在最后
```

元素添加、修改与删除

- 如果需要删除字典中指定的元素，可以使用del命令。

```
>>> del aDict['age']           #删除字典元素
>>> aDict
{'score': [98, 97], 'sex': 'male', 'a': 97, 'name': 'Dong'}
```

- 也可以使用字典对象的pop()和popitem()方法弹出并删除指定的元素，例如：

```
>>> aDict = {'age': 37, 'score': [98, 97], 'name': 'Dong', 'sex': 'male'}
>>> aDict.popitem()           #每次删除字典中的最后一个键值对，返回这个删除的键值对
('sex': 'male')
>>> aDict.pop('age')          #弹出指定键对应的元素
'37'
>>> aDict
{'score': [98, 97], 'name': 'Dong'}
```

现在大多数资料对字典的popitem方法的解释还是“随机删除一个键值对”，但是，从Python3.6开始字典变得有序之后，popitem是固定地删除字典的最后一个键值对。

谁是亲生儿子？

	列表	元组	字典	集合
创建(设置)	=[], list()	=(), tuple()	={}, dict()	={}, set()
添加	append(), extend(), insert()		update(), setdefault()	add(), update()
删除	del, remove(), pop(), clear(),	del()	pop(), popitem()	del(), remove(), discard(), pop(), clear()
访问	x[n], index(), count()	x[n], index(), count()	x[key], get(), items(), key(), value()	x[n]
其他操作	sort(), reverse(), copy(), max(), min(), all(), any(), len(), sum(), enumerate(), zip(), filter(), map(), in, x[::]	len(), enumerate(), map(), filter(), in	sorted(), copy(), max(), min(), len(), sum(), enumerate(), map(), filter(), in	sorted(), max(), min(), len(), sum(), enumerate(), map(), filter(), in

集合

- 集合 (set) 属于Python**无序可变序列**，使用一对**大括号**作为定界符，元素之间使用**逗号**分隔，同一个集合内的每个元素都是唯一的，**元素之间不允许重复**。
- 集合中只能包含数字、字符串、元组等**不可变类型**（或者说可哈希）的数据，而不能包含列表、字典、集合等可变类型的数据。

集合对象的创建与删除

- 直接将集合赋值给变量即可创建一个集合对象。

```
>>> a = {3, 5}                                #创建集合对象
>>> type(a)                                    #查看对象类型
<class 'set'>
```

- 也可以使用函数set()函数将列表、元组、字符串、range对象等其他可迭代对象转换为集合，如果原来的数据中存在重复元素，则在转换为集合的时候只保留一个；如果原序列或迭代对象中有不可哈希的值，无法转换成为集合，抛出异常。

```
>>> a_set = set(range(8, 14))                  #把range对象转换为集合
>>> a_set
{8, 9, 10, 11, 12, 13}
>>> b_set = set([0, 1, 2, 3, 0, 1, 2, 3, 7, 8]) #转换时自动去掉重复元素
>>> b_set
{0, 1, 2, 3, 7, 8}
>>> x = set()                                  #空集合
```

集合操作与运算

(1) 集合元素增加与删除

- 使用集合对象的add()方法可以增加新元素，如果该元素已存在则忽略该操作，不会抛出异常；update()方法用于合并另外一个集合中的元素到当前集合中，并自动去除重复元素。例如：

```
>>> s = {1, 2, 3}
```

```
>>> s.add(3)
```

#添加元素，重复元素自动忽略

```
>>> s
```

```
{1, 2, 3}
```

```
>>> s.update({3,4})
```

#更新当前字典，自动忽略重复的元素

```
>>> s
```

```
{1, 2, 3, 4}
```

集合操作与运算

- pop()方法用于随机删除并返回集合中的一个元素（在调试环境下是删除左边第一个元素），如果集合为空则抛出异常；
- remove()方法用于删除集合中的元素，如果指定元素不存在则抛出异常；
- discard()用于从集合中删除一个特定元素，如果元素不在集合中则忽略该操作；
- clear()方法清空集合删除所有元素。

```
>>> s.discard(5)           #删除元素，不存在则忽略该操作
>>> s
{1, 2, 3, 4}
>>> s.remove(5)           #删除元素，不存在就抛出异常
KeyError: 5
>>> s.pop()              #删除并返回一个元素
1
```

集合的pop方法一直保持随机删除一个元素，因为集合都是无序的

集合操作与运算

(2) 集合运算

```
>>> a_set = set([8, 9, 10, 11, 12, 13])
>>> b_set = {0, 1, 2, 3, 7, 8}
>>> a_set | b_set                                #并集
{0, 1, 2, 3, 7, 8, 9, 10, 11, 12, 13}
>>> a_set.union(b_set)                           #并集
{0, 1, 2, 3, 7, 8, 9, 10, 11, 12, 13}
>>> a_set & b_set                                #交集
{8}
>>> a_set.intersection(b_set)                   #交集
{8}
>>> a_set.difference(b_set)                      #差集
{9, 10, 11, 12, 13}
>>> a_set - b_set
{9, 10, 11, 12, 13}
>>> a_set.symmetric_difference(b_set)            #对称差集
{0, 1, 2, 3, 7, 9, 10, 11, 12, 13}
>>> a_set ^ b_set
{0, 1, 2, 3, 7, 9, 10, 11, 12, 13}
```

这就是为什么“同一个集合内的每个元素都是唯一的，元素之间不允许重复”的原因

集合操作与运算

```
>>> x = {1, 2, 3}
>>> y = {1, 2, 5}
>>> z = {1, 2, 3, 4}
>>> x < y
```

False

```
>>> x < z
```

True

```
>>> y < z
```

False

```
>>> {1, 2, 3} <= {1, 2, 3}
```

True

#比较集合包含关系

#真子集

#子集

如果 $A < B$ 不成立，不代表 $A \geq B$ 一定成立

基础数据结构：列表、元组、字典、集合



	列表	元组	字典	集合
类型名称	list	tuple	dict	set
定界符	方括号[]	圆括号()	大括号{}	大括号{}
是否可变	是	否	是	是
是否有序	是	是	否	否
是否支持下标	是（使用序号作为下标）	是（使用序号作为下标）	是（使用“键”作为下标）	否
元素分隔符	逗号	逗号	逗号	逗号
对元素形式的要求	无	无	键:值	必须可哈希
对元素值的 ^值 的要求	无	无	“键”必须可哈希	必须可哈希
元素是否可重复	是	是	“键”不允许重复，“值”可以重复	否
元素查找速度	非常慢	很慢	非常快	非常快
新增和删除元素速度	尾部操作快 其他位置慢	不允许	快	快

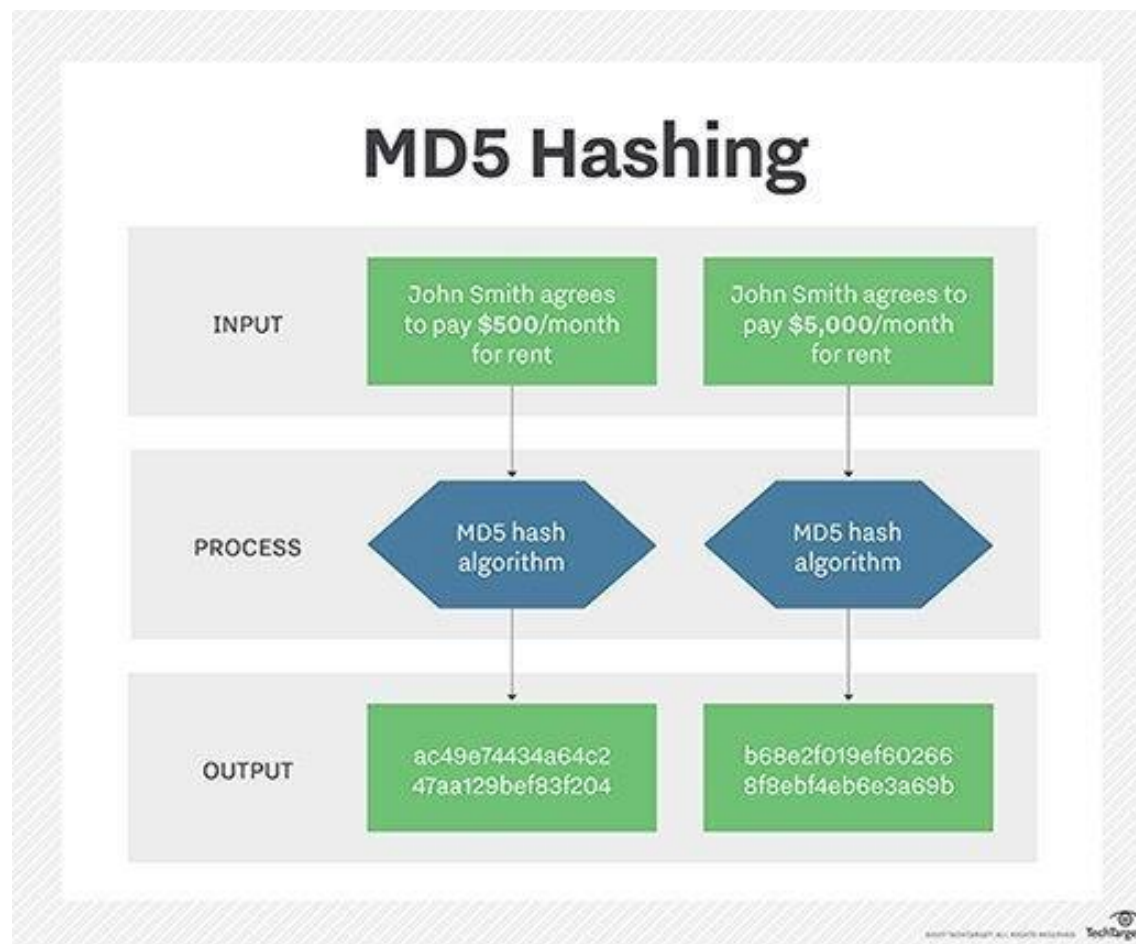
可哈希Hashable：（值）保持不变

- 通过hash函数，能产生唯一的value与key对应，如果key改变，value也应该改变，但是可变的数据结构，比如列表，它改变后，列表地址是不变的，可以理解为：不同的key指向了相同的value，这就发生了冲突，所以说列表是不可哈希的
- 散列函数（英语：Hash function）又称散列算法、哈希函数，是一种从任何一种数据中创建小的数字“指纹”的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（hash values, hash codes, hash sums, 或hashes）的指纹。散列值通常用一个短的随机字母和数字组成的字符串来代表。
- 接受任意长度的消息作为输入，并计算转换为一个固定长度的值

<https://www.python.org/downloads/release/python-3107/>

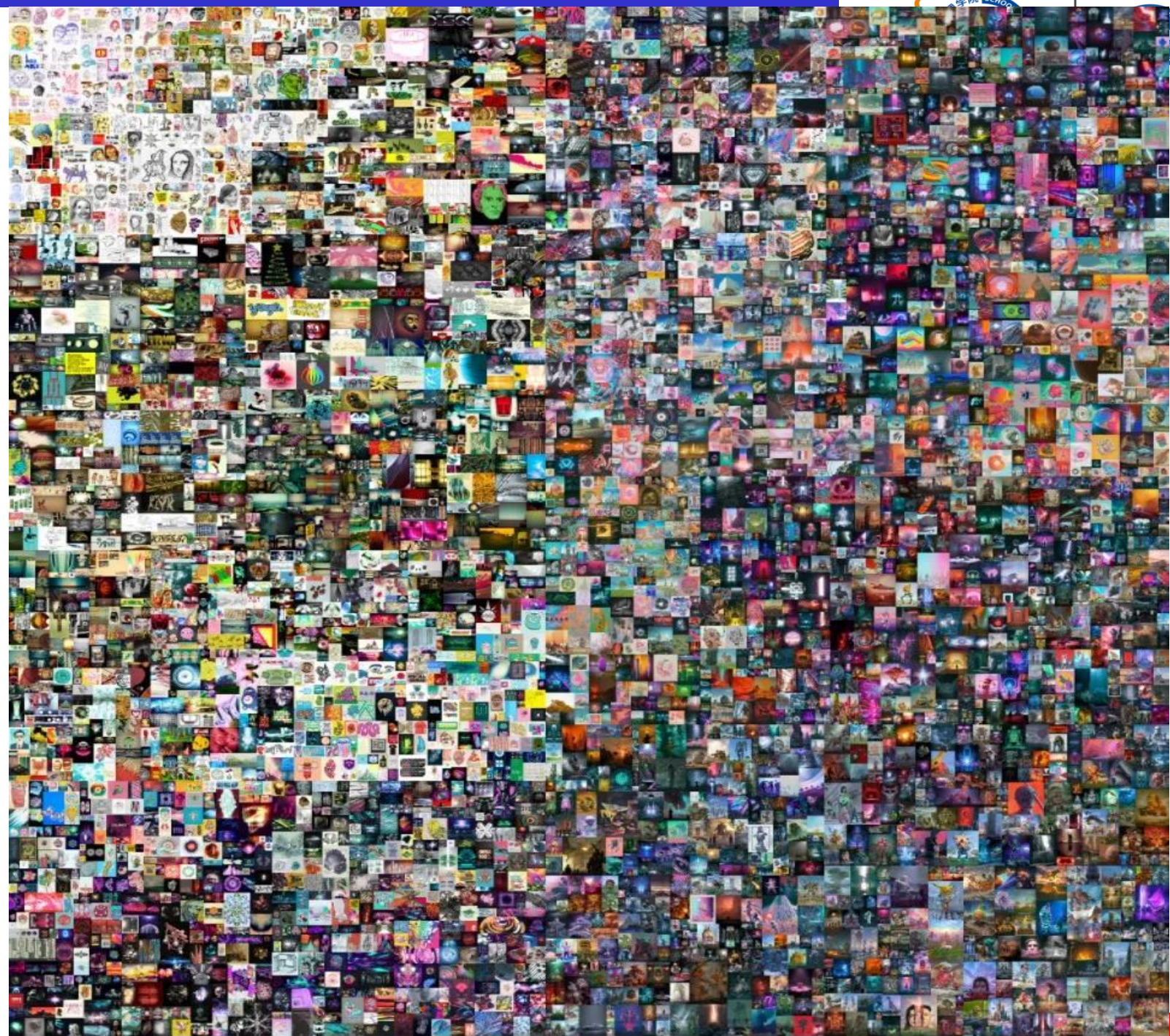
Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore	
Gzipped source tarball	Source release		1aea68575c0e97bc83ff8225977b0d46	26006589	SIG	CRT	SIG
XZ compressed source tarball	Source release		b8094f007b3a835ca3be6bdf8116cccc	19618696	SIG	CRT	SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	4c89649f6ca799ff29f1d1dffcb9393	40865361	SIG	CRT	SIG
Windows embeddable package (32-bit)	Windows		7e4de22bfe1e6d333b2c691ec2c1fcee	7615330	SIG	CRT	SIG
Windows embeddable package (64-bit)	Windows		7f90f8642c1b19cf02bce91a5f4f9263	8591256	SIG	CRT	SIG
Windows help file	Windows		643179390f5f5d9d6b1ad66355c795bb	9355326	SIG	CRT	SIG
Windows installer (32-bit)	Windows		58755d6906f825168999c83ce82315d7	27779240	SIG	CRT	SIG
Windows installer (64-bit)	Windows	Recommended	bfbe8467c7e3504f3800b0fe94d9a3e6	28953568	SIG	CRT	SIG



- ▶ 接受任意长度的消息作为输入，并计算转换为一个固定长度的值
- ▶ 因此，MD5算法（及后续的SHA-1/2/3）为任何文件（不管其大小、格式、数量）产生一个独一无二的数字指纹，借助这个数字指纹，通过检查文件前后 MD5 值是否发生了改变，就可以知道源文件是否被改动
 - ▶ 软件官网下载页面除了提供下载地址外，还会给出一串字符串，其实就是该软件的MD5 值

《每一天》 6928万美金



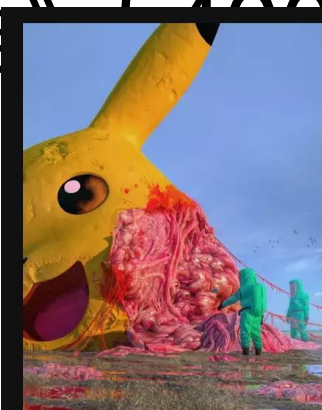
拍卖成交 5400万 “睡莲”

ENTERTAINMENT & ARTS

Monet painting of waterlilies sells for \$54 million at auction



A Claude Monet painting titled "Nymphaeas," which dates from 1906, was sold at an auction in London for \$54 million. (Andrew Cowie / AFP / Getty Images)



INFECTED CULTURE

10-Apr-20



AFTER QUARANTINE

3-Aug-20



HAPPIEST PLACE ON EARTH

20-Jul-20




POKEMON R

24-May-20

17至21岁期间，每天在电脑前自拍照片 2022年1月将933张自拍照上传至NFT交易平台

 OpenSea

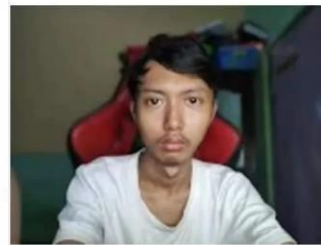




Ghozali Everyday

933 items	352 owners
0.67 floor price	68.5 volume traded


NFT艺术藏品



Ghozali E...
Ghozali_Gho
#841

Price
0.67
a day left


0



Ghozali E...
Ghozali_Gho
#430


Price
0.678
a day left

1



Ghozali E...
Ghozali_Gho
#303

Price
0.6869
Last 0.488



Ghozali E...
Ghozali_Gho
#362

Price
0.687
14 minutes left

NFT艺术藏品

销售403张，获得277 Eth（按3300美金/Eth计算，价值约89万美金）

置顶推文
Ghozali_Ghozalu
@Ghozali_Ghozalu

It's been 3 days and left 331 NFT sold out now because for the next few years I won't be listing

You can do anything like flipping or whatever but please don't abuse my photos or my parents will very disappointed to me

I believe in you guys so please take care of my photos.
由 Google 翻译自 英语

已经 3 天了，还剩 331 NFT
现在卖光了，因为在接下来的几年里我不会上市

您可以做任何事情，例如翻转或其他任何事情，但请不要滥用我的照片，否则我的父母会对我感到非常失望

我相信你们，所以请照顾好我的照片。



下午1:10 · 2022年1月12日 · Twitter Web App

NFT艺术品



利用社会热点事件成立DAO（分布式自治组织）发行虚拟币是利用新技术和新概念的诈骗手段，究其行为本质，依然在法律、行政法规所禁止的范畴。

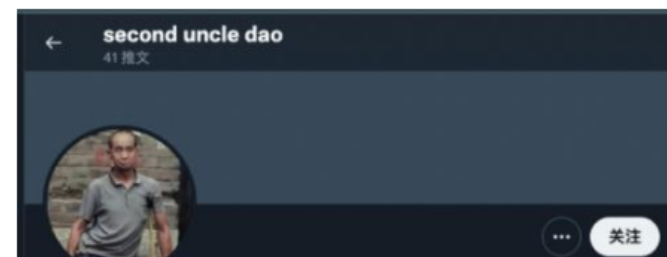
从启动到暴涨，再到崩盘，短短数日，“二舅币”创下币圈最快的崩盘纪录。更匪夷所思的是，二舅币崩盘以后，市场上又涌现出“大舅币”“二舅妈币”等跟风炒作的币种。

随着短视频《回村三天，二舅治好了我的精神内耗》近日走红，一款名为second uncle coin（二舅币，简称SUC）的虚拟币也进入大众视野，上线后被市场爆炒，短期上涨超过100倍，随后暴跌，发行人被质疑卷款跑路，网友因此戏称，“二舅治好的精神内耗，被二舅币磨光”。

事实上，蹭热点事件发币早已屡见不鲜。业内资深专家和律师对《国际金融报》记者表示，这几年，蹭社会热点事件发行的虚拟币五花八门，其中绝大部分本质上是欺诈、传销，同时还存在市场操纵等行为，扰乱了金融秩序。国内民众要打消通过非法途径买卖虚拟币、攫取高额利益的想法，要提高自己的风险意识，否则引起的损失只能自行承担。

借热点事件发币屡见不鲜

据悉，二舅币发起人是借由名为《回村三天，二舅治好了我的精神内耗》的网络热门视频，成立了second uncle DAO，于7月27日在币安智能链上创建二舅币，来吸收资金。该项目方声称，通过区块链将爱心传递，所筹集资金将会捐赠给“二舅”，为其提供养老保障。



NFT

- ▶ NFT全称Non-Fungible Token，译为非同质化代币
- ▶ 是一种基于区块链技术的数字加密货币
 - ▶ 可以随时验证真伪，了解流动和使用情况
- ▶ NFT具有可标识唯一性，不可拆分且无法造假

▶ NFT: 区块链+数字签名认证+独特的内容

▶ 客观来看

- ▶ 区块链技术是一种通用技术
- ▶ 数字签名技术（如MD5于1991年诞生并应用至今）
- ▶ 独特内容来自美或某种社会共识