第三章数据类型

> 本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务器进行重写。学完之后，你所收获的不仅仅是精通一门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

大家在阅读本系列课程时，最好打开python跟着老师敲敲代码看看代码的print输出加深理解。

一 数字类型

1. 数字类型 int和float

在第一章用户交互我们给大家介绍了一下基本的数据类型，这一章主要是介绍不同数据类型的使用方法和深入研究

```
数字类型即变量的值，如age=18，18则是我们保存的数据。
变量的是用来反映/保持状态以及状态变化的，毫无疑问针对不同的状态就应该用不同类型的数据去标识
```

数字类型只能存一个值，是不可变类型（不可变类型可以哈希，后面章节会讲解哈希）

```
#int整型
定义：age=10  # age=int(10)
用于标识：年龄，等级，身份证号，qq号，个数
#float浮点型
定义：salary=3.1 #salary=float(3.1)
用于标识：工资，身高，体重，
```

2. int类型常用的方法

下面代码所使用的bin()、oct()、hex()在进制转换中经常用到，大家可以了解一下：

```
# 常用操作+内置的方法
print(bin(3)) #十进制转成二进制,0b11
print(oct(8)) #十进制转成八进制,0o10
print(10) #10
print(hex(16)) #十进制转成十六进制,0x10
```

在进行用户交互程序的时候我们常用.isdigit()这个判断用户输入是不是数字，来进行下一步操作，这样就避免了程序出错，但是需要注意的是用户输入的内容默认是字符串，判断一个字符串是否是数字，我们使用这个方法。

```
age = input("your age>>:")
if age.isdigit():
    age = int(age)# 如果大家在pycharm中敲代码if语句下面是会自动缩进的，如果在普通文档编辑器中，
    敲下'Tab'键就可以缩进
    age +=1
print(age)
```

3. Python其他数字类型（了解）

```
#int（整型）
在32位机器上，整数的位数为32位，取值范围为-2**31～2**31-1，即-2147483648～2147483647
在64位系统上，整数的位数为64位，取值范围为-2**63～2**63-1，即-9223372036854775808～
9223372036854775807
#long（长整型）
跟C语言不同，Python的长整数没有指定位宽，即：Python没有限制长整数数值的大小，
但实际上由于机器内存有限，我们使用的长整数数值不可能无限大。
注意，自从Python2.2起，如果整数发生溢出，Python会自动将整数数据转换为长整数，
所以如今在长整数数据后面不加字母L也不会导致严重后果了。
注意：在Python3里不再有long类型了，全都是int
>>> a= 2**64 # >>>表示在命令行模式下进行输入
>>> type(a)  #type()是查看数据类型的方法
<type 'int'># int类型
>>> b= 2**60
>>> type(b)
<type 'int'>
#complex复数型
>>> x=1-2j
>>> x.imag
-2.0
>>> x.real
1.0
```

二 字符串类型

1. 字符串类型介绍

在python中，加了引号的字符就是字符串类型，用于标识：描述性的内容，如姓名，性别，国籍，种族，
定义形式：

```
name='Albert' # name=str('Albert')
```

字符串格式化：

```
name = input('请输入名字>>:')
print('你好，%s' % name)# %s其实可以理解为占位符，在输出时会使用字符串后面的%后面的变量的
```

值进行替换

2. 单引号，双引号和三引号的区别

```
#那单引号、双引号、多引号有什么区别呢？ 让我大声告诉你，单双引号木有任何区别，
只有下面这种情况 你需要考虑单双的配合
msg = "My name is Albert , I'm 18 years old!"
#多引号有什么作用呢？作用就是多行字符串必须用多引号
msg =
'''
今天我想写首小诗，
歌颂我的同桌，
你看他那乌黑的短发，
好像一只炸毛鸡。
'''
print(msg)
#  三引号里面写多行内容
```

3. 字符串的拼接

```
#数字可以进行加减乘除等运算，字符串呢？让我大声告诉你，也能？what ?是的，但只能进行"相
加"和"相乘"运算。
>>> name='albert'
>>> age='18'
>>> name+age #相加其实就是简单拼接
'albert18'
>>> name*5
'albertalbertalbertalbertalbert'
#注意1：
字符串1+字符串3，并不会在字符串1的基础上加字符串3，而是申请一个全新的内存空间存入字符串1和字符
串3，
相当字符串1与字符串3的空间被复制了一次
#注意2：只能字符串加字符串，不能字符串加其他类型
字符串拼接（只能在字符串之间进行，且只能相加或相乘）
```

字符串类型只能存一个值，是不可变类型，可以哈希。

4. 字符串类型常用的方法

(1) 按照索引取值

字符串是不可变类型，并不能改变字符串的值，取值时最多可以有三个参数，分别是起始位置，结束为止和步长，可以正向取值，反向取值（起始位置大于结束位置或者没有起始位置与结束位置，步长为-1表示从最后一个开始，步长为负数）

```
hobbies = "music basketball"
print(hobbies[0])
print(hobbies[1])
print(hobbies[5])
print(hobbies[0:5])
print(hobbies[:5])
print(hobbies[0:5:2]) # 从第零个元素到第四个元素，隔一个元素取一次
print(hobbies[0:5:-2])# 取不到元素，因为表示从第零个元素到第四个元素，逆序（-2为负数）隔一个元素取一次，
第零个元素前面没有元素了，所以什么都取不出来
print(hobbies[11:5:-2])# 从第11个元素到第六个元素，逆序隔一个元素取一次
print(hobbies[-1])# 取倒数第一个字符
print(hobbies[-2])# 取倒数第二个字符
```

(2) 长度运算

```
print(len('name'))# 可用来求string类型、列表等的长度
```

(3) 成员运算

```
print('a' in 'Albert')# 判断a是否在Albert中，在则返回True，否则返回False
# False
print('a' not in 'Albert')
# True
```

(4) 移除空白

```
print('   name'.lstrip())  # 移除左边空白
print('name  '.rstrip())  # 移除右边空白
print('  name  '.strip())  # 移除两边空白
```

(5) 切分

切分的作用：比如给你一条数据“Kobe 24 Lakers"你需要把Kobe的号码和所属球队都提取出来，那么你可以用以下方式进行切分,会返回一个列表。

```
print('Kobe 24 Lakes'.split(' '))
print('n a me'.split(','))
print('n,a me'.split(','))
print('n,/a /me'.split('/',1)) # 数字表示切割次数，默认全部切割
print('n,a /me'.split('/'))
print('a|b|c'.rsplit('|',1)) # 从右边开始切割
```

(6) 组合

```
print('a'.join('1234'))
print('a'.join('bcde'))
tag = ' '
print(tag.join(['I', 'say', 'hello', 'world']))  # 可迭代对象必须都是字符串
```

(7) 替换

```
print('abc name id'.replace('a', 'card')) # 将前面字符串中的a全部替换为‘card’
name = 'albert say :i have a dream,my name is albert'
# 注意此时name原来的值并没有改变，而是产生了一个替换之后新的值
print(name.replace('albert', 'Albert', 1))# 把name中的albert替换为Albert,只替换1次，默认全部替换
```

(8) 字符串其他操作

```
# find,rfind,index,rindex,count
name='albert say hello'
```

```
print(name.find('o',1,3))  # 顾头不顾尾,找不到则返回-1不会报错,找到了则显示索引
# print(name.index('e',2,4))  # 同上,但是找不到会报错
print(name.count('e',1,3))  # 顾头不顾尾,如果不指定范围则查找所有
# center,ljust,rjust,zfill
name='albert'
print(name.center(30,'-'))
print(name.ljust(30,'*'))
print(name.rjust(30,'*'))
print(name.zfill(50))  # 用0填充

# expandtabs
name='albert\thello'
print(name)
print(name.expandtabs(1))

# captalize,swapcase,title
print(name.capitalize())  # 首字母大写
print(name.swapcase())  # 大小写翻转
msg='albert say hi'
print(msg.title())  # 每个单词的首字母大写

# is数字系列
# 在python3中
num1=b'4'  # bytes
num2=u'4'  # unicode,python3中无需加u就是unicode
num3='四'  # 中文数字
num4='Ⅳ' #罗马数字

# isdigit:bytes,unicode
print(num1.isdigit())  # True
print(num2.isdigit())  # True
print(num3.isdigit())  # False
print(num4.isdigit())  # False

# isdecimal:uncicode
# bytes类型无isdecimal方法
print(num2.isdecimal())  # True
print(num3.isdecimal())  # False
print(num4.isdecimal())  # False

# isnumberic:unicode,中文数字,罗马数字
# bytes类型无isnumberic方法
print(num2.isnumeric())  # True
print(num3.isnumeric())  # True
print(num4.isnumeric())  # True
```

```
# 三者不能判断浮点数
num5='4.3'
print(num5.isdigit())
print(num5.isdecimal())
print(num5.isnumeric())

'''
总结:

    最常用的是isdigit,可以判断bytes和unicode类型,这也是最常见的数字应用场景

    如果要判断中文数字或罗马数字,则需要用到isnumeric
'''

# is其他
print('===>')
name='alb123ert'
print(name.isalnum())  # 字符串由字母或数字组成
print(name.isalpha())  # 字符串只由字母组成
print(name.isidentifier())
print(name.islower())
print(name.isupper())
print(name.isspace())
print(name.istitle())
```

三 列表

1. 列表类型基本介绍

在[]内用逗号分隔，可以存放n个任意类型的值，用于标识：存储多个值的情况，比如一个人有多个爱好，定义格式示例：

```
students=['albert','james','kd',]
# 或者students=list(['albert','james','kd',])
```

列表可以存储多个值，是有序的，是可变类型，不可以哈希。

2. 列表类型常用的方法

(1) 添加元素

append方法 在列表最后追加元素

```
l1 = ['a', 'b', 'c', 'd', ]
l1.append('e')
print(l1)
print(l1.append('e'))
# 在队尾追加，没有返回值

# ['a', 'b', 'c', 'd', 'e']
# None
```

insert方法 在列表中插入元素

```
l1 = ['a', 'b', 'c', 'd', ]
l1.insert(3,'x') # 3 指定插入索引3这个位置，就是"d"，这个位置，把'd' 顶到了后面
print(l1)

# insert可以指定插入的位置

# ['a', 'b', 'c', 'x', 'd']
```

extend方法 在列表中同时插入多个元素，extend方法使用以及与append方法的区别
append是整体添加

```
l1 = [1, 2, 3, 4, 5, ]
l1.append([6, 7, 8, 9, ])
# l1.append(*[6, 7, 8, 9, ])  # 会报错
print(l1)
l1.extend([6, 7, 8, 9])
print(l1)
```

> 只能接收一个参数，如果出现*打散的情况，还是会被识别成多个参数，因为程序执行执行是从左到右，从上倒下执行的，当出现*时这个列表已经被打散了，因而，会被程序识别成被传入了多个参数。

extend是逐个添加

```
l1 = [1, 2, 3, 4, 5, ]
l1.extend([6, 7, 8, 9])
print(l1)
l1.extend('abc')
print(l1)
l1.extend('a')  # 也是可迭代对象
print(l1)
# l1.extend(1)  # 报错，不可迭代
print(l1)
```

extend在执行添加的时候，被传入的参数必须是可迭代对象，这样通过迭代就解决了同时传入多个参数的问题，如果你还不知道可迭代对象，放心，你很快就会知道的。

(2) 删除元素

pop删除，有返回值，默认删除列表中最后一个，指定删除索引值

```
l1 = [1, 2, 3, 4, 5, 6, ]
print(l1.pop())  # 移除列表中的某个元素，默认是最后一个元素，这个是有返回值的，
因为这是一个可迭代类型和我们前面的extend一类
print(l1)
print(l1.pop(2))  # 2指定是列表中的索引值
print(l1)
```

remove 删除 没有返回值，没有默认值，指定被删除的元素

```
l1 = [1, 2, 3, 4, 5, 6, ]
print(l1.remove(2)) # 注意这个2 是列表的元素2，不是索引值，和上述pop区分清楚，
pop移除的是索引值所对应的元素
# print(l1.remove())  # 报错
print(l1)
```

clear 删除 保留列表名称，清空里面的值

```
l1 = ['a', 'b', 'c', 'd', 'e', 'b', 'c']
l1.clear()
print(l1)
```

del 删除 通用删除 但是一般不用

```
l1 = [1, 2, 3, 4, 5, 6, ]
del l1[2]  # 按索引l1[0] = 1,l1[1]=2,l1[2]=3
print(l1)
l1 = [1, 2, 3, 4, 5, 6, ]
del l1  # 在内存中删除l1 相当于没有定义l1
# print(l1)  # 报错
```

(3) 更改元素

注意与insert插入的不同，都是以原来的为位置为参照，insert是挤开，本质是添加，而这里是替换

```
l1 = [1, 2, 3, ]
l1[2] = 4# 将索引2所代表的元素替换为4
print(l1)
```

(4) 查找元素

按照索引或者指定列表中的元素查找

```
l1 = ['a', 'b', 'c', 'd', 'e', 'b', 'c']
print(l1.index('a'))  # 返回a在l1的索引值
print(l1.index('b',1,4))  # 两个数字分别对应起始位置和结束位置
print(l1[0:5])  # 按照索引取值和字符串的用法相同
print(l1[:5])
print(l1[0:5:2])
print(l1[:-1])
print(l1[:-2])
print(l1[5:1:-2])
```

把列表分段查找

```
l1 = ['a', 'b', 'c', 'd', 'e', 'b', 'c']
a = int(len(l1)/2)
print(l1[:a])  # 从中间位置开始，列出前面所有的元素
print(l1[a:])  # 从中间位置开始，列出后面所有的元素
```

使用enumerate对列表枚举

```
l1 = ['a', 'b', 'c', 'd', 'e']
for i, x in enumerate(l1, 100):  # 100指定枚举开始的数字
    print(i, x)
```

(5) 统计长度

```
print(len([1,2,3,4,]))
```

(6) 统计元素个数

```
print(['a','b'].count('a'))
```

(7) 排序

reverse 反序排序

```
l1 = ['a', 1, 'b', 'c', 'd', 'e', 'b', 'c']
l1.reverse()
print(l1)
```

sort 按照ascii码来进行排序

```
l1 = ['a', '1', 'b', 'c', 'd', 'e', 'b', 'A', 'Z', 'c']
l1.sort()
```

```
print(l1)
l1.sort(reverse=True)
print(l1)
```

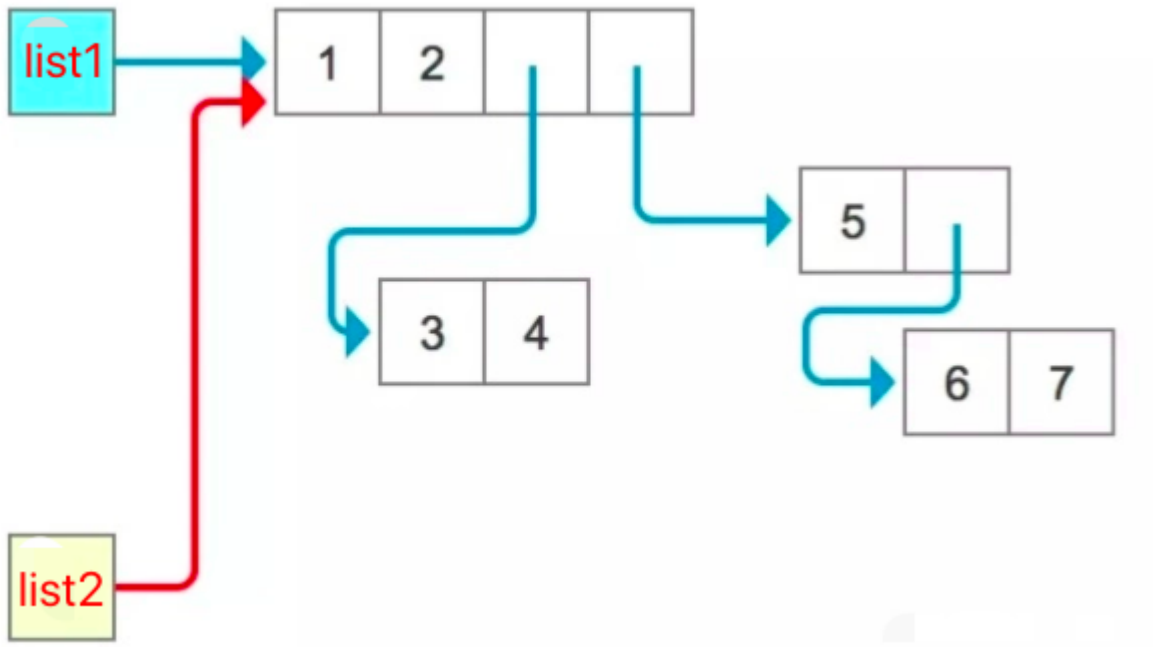
(8) 复制列表

```
l1 = ['a', '1', 'b', 'c', 'd', 'e', 'b', 'A', 'Z', 'c']
print(l1.copy())
```

(9) Python中的赋值，浅拷贝与深拷贝区别

赋值

对于复制的操作，最简单的就是赋值，指的是新建一个对象的引用，新建目标对象与原来的目标对象指向同一个内存地址，**赋值只相当于增加了一个引用，并没有开辟新的内存空间**。如下图所示



用 `id` 验证下就知道， list1 和 list2 仍然是同一个东西 。那么他们内部的元素自然也是一样的，对其中一个进行修改，另一个也会跟着变：

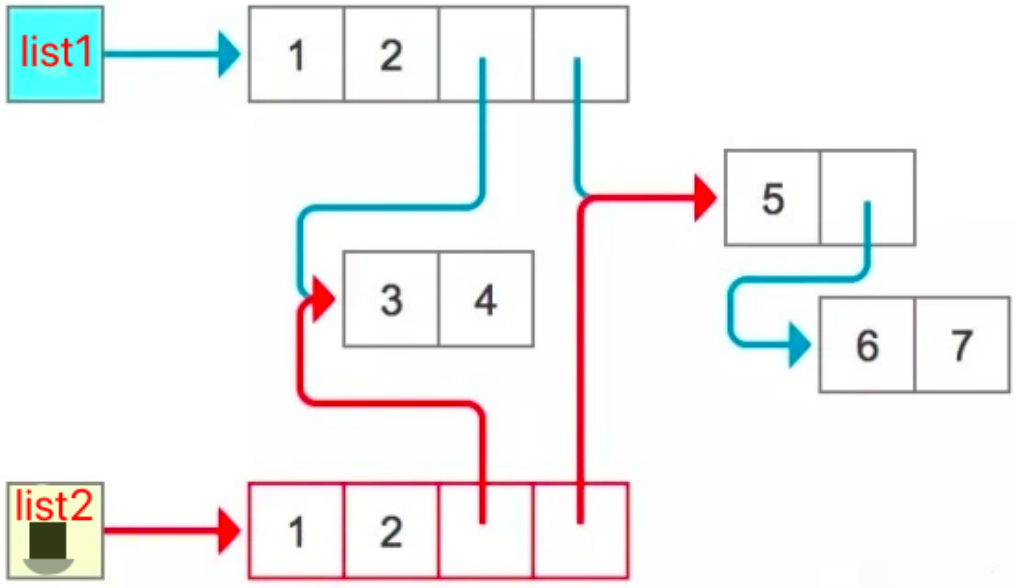
```
list1 = [1, 2, [3, 4], [5, [6, 7]]]
print('list1:', id(list1))
print([id(i) for i in list1])
list2 = list1
print('list2:', id(list2))
```

```
print([id(i) for i in list2])
print(list1 is list2)
print(list1[0] is list2[0])
print(list1[2] is list2[2])
list2[0] = -1
print(list1)
list2[2][1] = -1
print(list1)
```

因此有人将此操作称为“ 旧瓶装旧酒 ”，只是多贴了一层标签，这不能达到我们的目的。要得到一个对象的“拷贝”，我们需要用到 `copy` 方法。

浅拷贝

浅拷贝顾名思义就是拷贝的比较浅，我们可以把赋值认为是新建了一个对象的引用，把原来被对象内存空间的数据指向新的变量，这时同一块内存空间指向两个变量。浅拷贝与赋值不同，既然是拷贝，那么就是要开辟一块新的内存空间，复制的是原来被拷贝对象内多个元素对象的引用，有几个元素对象就赋值几个元素对象的引用，也有人把这种操作称为“新瓶装旧酒”。不过，如果旧的酒瓶里面如果还套了一个酒瓶子，那你就需要注意了。因此，不论是对拷贝对象或者是被拷贝对象内**一个可变类型元素内的元素**的修改（包含添加，修改和删除），都会引起另外一方的变化。但是，如果是直接对对象内的一级元素当作**一个整体进行修改**（这时对可变类型元素的元素只能是删除），虽然他们的元素有共享的内存地址，但是拷贝对象或者被拷贝对象元素内的引用都是独立的，删除了其中一方的一个引用，另外一方的引用依然存在。




```
from copy import copy

list1 = [1, 2, [3, 4], [5, [6, 7]]]
```

```
print('list1:', id(list1))
print([id(i) for i in list1])
list2 = copy(list1)
print('list2:', id(list2))
print([id(i) for i in list2])

# 1 判断论证过程
print(list1 is list2)  # id不同自然不是，id用来形象表示内存地址
print(list1[0] is list2[0])  # 两者之间对应的元素都指向同一个地址
print(list1[2] is list2[2])

# 2 对一级元素整体修改论证（其实此时可变类型元素也只能当作不可变来修改）
list1[0] = -1  # 修改列表1或2都是单独变化的
print(list1)
print(list2)

list2[1] = -2
print(list1)
print(list2)

list1[2] = -3
print(list1)
print(list2)

list2[2] = -30
print(list1)
print(list2)

# 3 给对象添加元素
list1.append('add1')
print(list1)
print(list2)

list2.append('add2')
print(list1)
print(list2)

# 4 删除
list1.pop()
print(list1)
print(list2)

list2.pop()
print(list1)
print(list2)
```

```
list1.pop()
print(list1)
print(list2)

# 5 只有对对象内一个可变类型元素内的元素的修改（包含添加，修改和删除），才会同步变化
list2[-1][0] = '5->55'
print(list1)
print(list2)

# # 换成list1也是一样，但需要把上面的部分代码注释掉，确保索引位置不会出错
# list1[3][0] = '5->55000'
# print(list1)
# print(list2)
```

深拷贝

深拷贝其实与浅拷贝有本质的区别，它不会复制任何的引用，对象内的所有元素，子元素，孙子元素，重孙元素，曾孙元素的数据都是由复制而来，因此，这样的操作被称为“新瓶装新酒”。它的实现原理就是递归，只要任意元素内仍然有子元素，就会复制子元素的数据放到新的内存地址。既然这样，在使用深拷贝后，被拷贝对象的改变，不会引起拷贝对象的任何改变。

```
import copy


list1 = [1, 2, 3, 4, 5, [6, 7, 8, ]]
list2 = copy.deepcopy(list1)
print(list1)
print(list2)

list1[5].append(9)
print(list1)
print(list2)

list1.append(6)
print(list1)
print(list2)

list1.pop()
print(list1)
print(list2)
```

复制与深浅拷贝总结：

- 赋值：新建一个原来对象内存地址的引用，对象本身不开辟新的内存空间；
- 浅拷贝：新建多个原来对象内一级子元素内存地址的引用，对象本身需要开辟新的内存空间；
- 深拷贝：复制原来对象内的所有N级子元素的数据，所有的数据都开辟新的内存空间。

四 元组

与列表类型相比，非常类似只不过[]换成()，作用：用于存储多个值，对比列表来说，元组不可变（是可以当做字典的key的），不可更改，主要是用来读。

```
age=(11,22,33,44,55)
# 本质age=tuple((11,22,33,44,55))
```

元组可以存储多个值，是有序的，是不可变类型，可以哈希。元组常用的方法，请参考列表常用方法，需要注意的是，元组只能取值，而不能改变元组的值

五 字典

1. 字典类型基本介绍

字典用于存放一个人的信息：姓名，性别，年龄，很明显是多个值，既然是存多个值，我们完全可以基于刚刚学习的列表去存放，如下：

```
# 既然如此，我们为何还要用字典？
info=['albert','male',18]
# 定义列表的目的不单单是为了存储，还要考虑取值，如果我想取出这个人的年龄，可以用
info[2]
# 但这是基于我们已经知道在第3个位置存放的是年龄的前提下，我们才知道索引2对应的是年龄，即：
#       # name, sex, age
info=['albert','male',18]
"""
而这完全只是一种假设，并没有真正意义上规定第三个位置存放的是年龄，
于是我们需要寻求一种，即可以存放多个任意类型的值，又可以硬性规定值的映射关系的类型，
比如key=value，这就用到了字典
"""
```

字典用于标识存储多个值的情况，每个值都有唯一一个对应的key，可以更为方便高效地取值，字典的定义格式如下：

```
# 在{}内用逗号分隔，可以存放多个key:value的值，key一般是字符串，value可以是任意类型
info={'name':'albert','age':18,'sex':18}
# info=dict({'name':'albert','age':18,'sex':18})
```

字典可以存储多个值，是无序的，是可变类型，不可哈希。

2. 字典类型常用的方法

(1) 增加元素

通过键值对的方式

```
l1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male'
}
l1['hobbies'] = "music"
print(l1)
```

用fromkeys构造一个字典

```
"""
第一个参数迭代循环的字典的key，第二个参数表示value，
可以多个key循环对应这个value，也可以只有一个key，也可以没有value
"""
a = l1.fromkeys(l1,'I am Albert')
print(a)

b = dict.fromkeys('name')  # 必须有一个可迭代类型，作为字典的key
print(b)

b = dict.fromkeys('e')  # 也可以迭代
print(b)
# b = dict.fromkeys(1) #报错 数字类型不可迭代
```

```
b = dict.fromkeys([1,2,3,])
print(b)
```

(2) 删除元素

del 通过字典的key删除

```
l1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male'
}

del l1['name']
print(l1)
```

pop 或者popitem删除

```
l1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male'
}

res = l1.pop('name')  # 删除指定key的value，并拿到一个返回值
print(res)
print(l1)

res2 = l1.popitem()  # 随机返回并删除字典中的一对键和值(一般删除末尾对)。
# 如果字典已经为空，却调用了此方法，就报出KeyError异常。
print(res2)
print(l1)
```

(3) 更改元素

通过键值对的方式

```
l1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male','3':3,
}

l1['name'] = '马一特'
print(l1)
```

通过setdefault 或者update操作，两者使用和区别如下：
setdefault只添加不修改

```
d1 = {
    'name': 'albert',
    'age': 18,
}

d1.setdefault('name', 'Albert')
d1.setdefault('gender', 'male')
print(d1)
```

update既添加也修改

```
d1 = {
    'name': 'albert',
    'age': 18,
}

d1.update({'name': 'Albert', 'gender': 'male'})  # 注意传参方式的不同
print(d1)
```

(4) 查找元素

通过键值对查找

```
l1 = {
    'name': 'albert',
    'age': 18,
```

```
    'gender': 'male'
}

a = l1['name']
print(a)
```

通过get方法查找

```
l1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male'
}

a = l1.get('hobbies')    # 找不到不报错
print(a)
```

通过enumerate 枚举

```
d1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male',
    '3':3,
}

for a in enumerate(d1):
    print(a)
```

通过.keys(),.values(),.items()等方法

```
d1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male',
    '3':3,
}

a = d1.keys()
```

```
print(a)
print(list(a)[0])

a = d1.values()
print(a)
print(list(a)[0])

a = d1.items()
print(a)
print(list(a)[0])
```

通过for循环遍历

```
d1 = {
    'name': 'albert',
    'age': 18,
    'gender': 'male',
    '3':3,
}

for k,v in d1.items():
    print(k,v)
```

(5) 成员运算与长度运算

参考列表的运算方法，成员运算的依据是字典的key，而不是value，长度运算都可以作为参考的依据

六 集合类型

花括号内，多个元素用逗号分割，用来存储多个值，并且是无序的，那么这多个值不能用来取值，但是我们可以使用它来进行去重(比如去掉列表中重复的元素)和关系运算。

```
# 集合的元素遵循三个原则：
                1：每个元素必须是不可变类型(可作为字典的key)
                2：没有重复的元素
                3：无序
# 注意集合的目的是将不同的值存放到一起，不同的集合间用来做关系运算，无需纠结于集合中单个值
```

集合类型可以存多个值，是无序的，是可变类型，不可以哈希。
关系运算练习

```
有如下两个集合，piano是报名钢琴课程的学员名字集合，violin是报名小提琴课程的学员名字集合
    piano={'albert','孙悟空','周星驰','朱茵','林志玲'}
    violin={'猪八戒','郭德纲','林忆莲','周星驰'}
    1. 求出即报名钢琴又报名小提琴课程的学员名字集合
    2. 求出所有报名的学生名字集合
    3. 求出只报名钢琴课程的学员名字
    4. 求出没有同时这两门课程的学员名字集合
```

参考答案

```
# 求出即报名钢琴又报名小提琴课程的学员名字集合
print(piano & violin)
# 求出所有报名的学生名字集合
print(piano | violin)
# 求出只报名钢琴课程的学员名字
print(piano - violin)
# 求出没有同时这两门课程的学员名字集合
print(piano ^ violin)
```

七 布尔类型

计算机俗称电脑，即我们编写程序让计算机运行时，应该是让计算机无限接近人脑，或者说人脑能干什么，计算机就应该能干什么，人脑的主要作用是数据运行与逻辑运算，此处的布尔类型就模拟人的逻辑运行，即判断一个条件成立时，用True标识，不成立则用False标识。

```
#布尔值，一个True一个False
>>> a=3
>>> b=5
>>> a > b #不成立就是False,即假
False
>>> a < b #成立就是True, 即真
True
# 接下来就可以根据条件结果来干不同的事情了：
if a > b:
```

```
    print(a is bigger than b )
else:
    print(a is smaller than b )
# 上面是伪代码，但意味着，计算机已经可以像人脑一样根据判断结果不同，来执行不同的动作。
```

布尔类型的重点知识

```
#所有数据类型都自带布尔值
1、None，0，空（空字符串，空列表，空字典等）三种情况下布尔值为False
2、其余均为真
```

可变类型与不可变类型（重点）

```
# 1.可变类型：在id不变的情况下，value可以变，则称为可变类型，如列表，字典
# 2. 不可变类型：value一旦改变，id也改变，则称为不可变类型（id变，意味着创建了新的内存空间）
```

八 collections容器数据类型

1. namedtuple 命名元组

```
from collections import namedtuple # 从collections库中导入namedtuple（导入外部库）
# 创建一个命名元组对象
point = namedtuple('p', ['x', 'y'])  # p代表名称，"x"和"y"为内容
p = point(1, 2)
print(p)
print(p.x)  # 1
print(p.y)  # 2
```

2. deque 超级列表

数据结构中比较常用的双向队列在python中可使用deque实现。

```
from collections import deque
# 类似列表(list)的容器，实现了在两端快速添加(append)和弹出(pop)
d = deque('abcd')
```

```python
for i in d:
    print(i)
print(d[0])
print(d[1])
d.append('e')  # 从右边加入
print(d)
d.appendleft('x')  # 从左边加入
print(d)
d.pop()  # 从右侧弹出
print(d)
d.popleft()  # 从左侧弹出
print(d)
deque(reversed(d))  # 反转顺序
print(d)
# d = list(d)  # 转化成list
# d = list(reversed(d))
# print(d)
d.extend('xyz')  # 从右侧添加
print(d)
d.extendleft('nba')  # 从左侧添加
print(d)
d.rotate(1)  # 把最右边的元素挪到最左边
print(d)
d.rotate(-1)  # 把最左边的元素挪到最右边
print(d)
d.clear()  # 清空
# d.pop()  # 报错
```

3. ChainMap 链映射

```python
from collections import ChainMap
"""
一个 ChainMap 类是为了将多个映射快速的链接到一起，这样它们就可以作为一个单元处理。
它通常比创建一个新字典和多次调用 update() 要快很多。
这个类可以用于模拟嵌套作用域，并且在模版化的时候比较有用。
"""
# 链映射的用法
dict1 = {'name': 'Albert', 'age': 18}
dict2 = {'weight': 65, 'height': 180}
res = list(ChainMap(dict1, dict2))
print(res)
```

4. Counter 计数字典

```
from collections import Counter
# 计数
cnt = Counter()
for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
    cnt[word] += 1# 对word的计数数量加
print(cnt)
# Counter({'blue': 3, 'red': 2, 'green': 1})
# 数学运算
c = Counter(a=3, b=1)
d = Counter(a=1, b=2)
print(c + d)  # 相同的部分相加
# Counter({'a': 4, 'b': 3})
print(c - d)  # 相同的部分相减
# Counter({'a': 2})
print(c & d)  # 相同的部分取最小
# Counter({'a': 1, 'b': 1})
print(c | d)  # 相同的部分取最大
# Counter({'a': 3, 'b': 2})
```

Collections容器类型一共有9种，除了上面介绍的4种之外还有OrderedDict，defaultdict，UserDict，UserList，UserString，他们的用法不是完全相同，但是使用起来并不复杂，更多Collections容器类型数据结构详见官方文档：Collections容器数据类型。