重要数据类型

列表数据类型

- 在实际开发中，经常需要将一组（不只一个）数据存储起来，以便后边的代码使用。列表就是这样的一个数据结构。且列表是Python中最基本也是最常用的数据结构之一。
- 什么是数据结构呢?
  - 通俗来讲，可以将数据结构当做是某种容器，该容器是用来装载或者存储数据的。不同的数据结构决定了对数据不同的组织方式。
  - 那么当数据被装载或者存储到了某个数据结构中后，那么就可以基于该数据结构的特性对数据进行不同形式的处理和运算。
- 列表的创建方式
  - 创建一个列表，只要把逗号分隔的不同的数据元素使用方括号括起来即可。列表内的元素，可以是其它任意类型的数据，可多层嵌套列表，元素个数无限制。

```
alist = [1,2,3,4,5]
items = [1,'bobo',12.34]
#列表中可以存储任意类型的数据
```

- 列表元素：

    - 存储在列表数据结构中的每一个数据被称为列表元素，简称元素。
- 列表索引：
    - 列表中的每个元素都被分配一个数字作为索引，用来表示该元素在列表内所排在的位置。第一个元素的索引是0，第二个索引是1，依此类推。
- 列表特性：
    - Python的列表是一个有序可重复的元素集合，可嵌套、迭代、修改、分片、追加、删除，成员判断。
- 访问列表内的元素
    - 列表从0开始为它的每一个元素顺序创建下标索引，直到总长度减一。要访问它的某个元素，以方括号加下标值的方式即可。注意要确保索引不越界，一旦访问的 索引超过范围，会抛出异常。所以，一定要记得最后一个元素的索引是len(list)-1。

```
alist = [1,12.3,'bobo']
print(alist[2])  #'bobo'
print(alist[0:2]) #[1, 12]
print(alist[6]) #使用索引和切片的时候，不可以访问超出索引范围的元素
```

- 修改元素的值
    - 直接对元素进行重新赋值

```
alist = [1,12.3,'bobo']
alist[1] = 100.123
print(alist)
```

- 删除元素
  - 使用del语句或者remove(),pop()方法删除指定的元素。

```
alist = [1,12.3,'bobo']
# del alist[0] #删除下标为0的列表元素
# alist.remove('bobo') #删除列表中bobo这个列表元素
# alist.pop() #默认情况下pop会把列表中最后一个元素删除
alist.pop(2) #将列表中下标为2的元素进行删除
print(alist)
```

- 切片
  - 切片指的是对序列进行截取，选取序列中的某一段。
    - 切片的语法是： list[start:end]

```
#同字符串的切片机制一样
```

  - 以冒号分割索引，start代表起点索引，end代表结束点索引。省略start表示以0开始，省略end表示到列表的结尾。注意，区间是左闭右开的！也就是说[1:4]会截取列表的索引为1/2/3的3个元素，不会截取索引为4的元素。

分片不会修改原有的列表，可以将结果保存到新的变量，因此切片也是一种安全操作，常被用来复制一个列表，例如newlist = lis[:]。

- 切片过程中还可以设置步长，以第二个冒号分割，例如list[3:9:2]，表示每隔多少距离取一个元素。

- 列表的内置方法
  - 上文中我们说过，数据存储到不同的数据结构中，可以基于该数据结构的特性对数据进行指定形式的操作和处理。下图中的方法是列表专有的内置方法，请熟记于心。
  -

方法	作用
append(obj)	在列表末尾添加新的对象
count(obj)	统计某个元素在列表中出现的次数
extend(seq)	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
index(obj)	从列表中找出某个值第一个匹配项的索引位置
insert(index, obj)	将对象插入列表
pop(obj=list[-1])	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
remove(obj)	移除列表中某个值的第一个匹配项
reverse()	反向列表中元素
sort([func])	对原列表进行排序
copy()	复制列表
clear()	清空列表，等于del lis[:]

```
alist = ['bobo',"18","99.5",'北京']
#将列表转换成字符串
```

```
ret = '-'.join(alist) #将列表中的每一个列表元素根据-为间隔进行拼接，返回字符串结果
print(ret)

#如何将字符串转换成列表
s = 'hello-name-bobo-age'
ret = s.split('-')
print(ret)

alist = [3,8,5,7,6,2,1]
alist.sort() #对列表元素进行排序
print(alist)

a = [1,2,3]
a.append('bobo') #向列表尾部添加一个元素
print(a)

a1 = [1,2,3]
a1.insert(1,999) #向列表下标为1的位置添加一个元素
print(a1)
```

元组数据类型

- 用方括号括起来的是列表，那么用圆括号括起来的就是元组。

- 元组也是序列结构，但是是一种不可变序列，你可以简单的理解为内容不可变的列表。除了在内部元素不可修改的区别外，元组和列表的用法差不多。

```
t = (1,2,3,4,5)
```

- 元组与列表相同的操作：
  - 使用方括号加下标访问元素
  - 切片（形成新元组对象）
  - count()/index()
  - len()/max()/min()
- 元组中不允许的操作，确切的说是元组没有的功能：
  - 修改、新增元素
  - 删除某个元素（但可以删除整个元组）
  - 所有会对元组内部元素发生修改动作的方法。例如，元组没有remove，append，pop等方法。
- 提问：
  - 在学完列表后，在使用元素我们会发现元组由于是一个不可变序列，则很多操作会受到限制，这不能，那不能，要你何用，我直接用列表不行吗?
  - 还是有用的，有些数据一旦创建之后就不允许修改了，这些数据就适合用元组来创建，比如主机地址和端口（ip，port），（"192.168.1.1"， 80），两者捆绑在一起，不允许修改。

```
ip_port = ("192.168.1.1", 8080)
```

字典数据类型

- 字典的实现机制：
  - Python的字典数据类型是基于hash散列算法实现的，采用键值对(key:value)的形式，根据key的值计算value的地址，具有非常快的查取和插入速度。
- 字典特性：
  - 字典包含的元素个数不限，值的类型可以是任何数据类型！但是字典的key必须是不可变的对象，例如整数、字符串、bytes和元组，最常见的还是将字符串作为key。列表、字典、集合等就不可以作为key。同时，同一个字典内的key必须是唯一的，但值则不必。
  - 注意：从Python3.6开始，字典是有序的！它将保持元素插入时的先后顺序！请务必清楚！
- 创建字典
  - 字典的每个键值对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号({})中 ，例如：
    - d = {key1 : value1, key2 : value2 }

```
#键值对：key : value
#key：只能使用不可变类型的数据充当，通常使用字符串
#value：任意数据类型的值充当
#字典中无法存储重复的键值对
dict_1 =
{'name':'bobo','age':18,'score':100,'age':1
8}
#注意：不要在字段中存储相同的key，value可以相同
dict_2 = {'name':'bobo','age':18,'age':20}
print(dict_2)
```

- 访问字典
  - 虽然现在的字典在访问时有序了，但字典依然是集合类型，不是序列类型，因此没有索引下标的概念，更没有切片的说法。但与list类似的地方是，字典采用把相应的键放入方括号内获取对应值的方式取值。

```
d = {'name':'bobo','age':20,"scores":
[100,120,99]}
#根据key访问对应的value值
print(d['name'],d['scores']) #依次访问name和
scores对应的value值
print(d.get('name')) #通过get使用对应的key访问
对应的value值

#注意：使用[]访问不存在的key对应的value值程序会报
错
# print(d['adress']) #程序报错

#注意：使用get访问不存在的key程序不会报错，但是会
返回None这个空值
print(d.get('address'))
```

- 添加和修改
  - 增加就是往字典插入新的键值对，修改就是给原有的键赋予新的值。由于一个key只能对应一个值，所以，多次对一个key赋值，后面的值会把前面的值冲掉。

```
d = {'name':'bobo','age':20,"scores":
[100,120,99]}
d['name'] = 'jay' #给存在的key修改对应的value值
d['address'] = 'Beijing' #给一个不存在的key赋值表
示新增键值对
del d['age'] #删除age键值对
print(d)
```

- 删除字典元素、清空字典和删除字典
  - 使用del关键字删除字典元素或者字典本身，使用字典的clear()方法清空字典。

```
d = {'name':'bobo','age':20,"scores":
[100,120,99],'name':'bobo'}
del d['name']
print(d)

d = {'name':'bobo','age':20,"scores":
[100,120,99],'name':'bobo'}
del d
print(d)

d = {'name':'bobo','age':20,"scores":
[100,120,99],'name':'bobo'}
d.clear()
print(d)
```

- 字典的重要方法

```
d = {'name':'bobo','age':20,"scores":
[100,120,99]}
print(d.keys()) #返回字典中所有的key
print(d.values()) #返回字典中所有的value
print(d.items()) #返回字典中所有的键值对
```

bytes字节类型/二进制类型

- 在Python3以后，字符串和bytes类型彻底分开了。字符串是以字符为单位进行处理的，bytes类型是以字节为单位处理的。
- bytes数据类型在所有的操作和使用甚至内置方法上和字符串数据类型基本一样，也是不可变的序列对象。
- 作用
  - bytes对象只负责以二进制字节序列的形式记录所需记录的对象。Python3中，bytes通常用于网络数据传输、二进制图片和文件的保存等等
- 创建方式

```
b = b''          # 创建一个空的bytes
b = bytes()       # 创建一个空的bytes
b = b'hello'     #  直接指定这个hello是bytes类型
b = bytes('string',encoding='编码类型')  #利用内置bytes方法，将字符串转换为指定编码的bytes
b = str.encode('编码类型')   # 利用字符串的encode方法编码成bytes，默认为utf-8类型

bytes.decode('编码类型'): 将bytes对象解码成字符串，默认使用utf-8进行解码。
```

- 对于bytes，我们只要知道在Python3中某些场合下强制使用，以及它和字符串类型之间的互相转换，其它的基本照抄字符串。

  简单的省事模式：

  `string = b'xxxxxx'.decode()` 直接以默认的utf-8编码解码bytes成string

  `b = string.encode()` 直接以默认的utf-8编码string为bytes

```
msg = b'hello' #创建了一个二进制类型的变量，存储的是hello二进制的数据

#将msg二进制类型的数据转换成字符串
s_msg = msg.decode()
print(s_msg)

#字符串转二进制类型
new_msg = s_msg.encode()
print(new_msg)

#字符串转二进制
msg = '你好吗？'
#使用gbk的编码将中文的字符串编码成二进制的形式
byte_msg = msg.encode(encoding='gbk')
print(byte_msg)
```

```
#二进制转字符串:使用gbk对二进制进行解码，还原成中文的字符串形式
ret = byte_msg.decode(encoding='gbk')
print(ret)
```

set集合

- 特性
  - set集合是一个无序不重复元素的容器，集合数据类型的核心在于自动去重。
- 创建方式
  - 集合使用大括号{}框定元素，并以逗号进行分隔。但是注意：如果要创建一个空集合，必须用 set() 而不是 {} ，因为后者创建的是一个空字典。集合除了在形式上最外层用的也是花括号外，其它的和字典没有一毛钱关系。

```
s = {1,2,3,'bobo',99.9,1,2,3}
s = set()
print(s)
```

- 注意：
  - 集合既不支持下标索引也不支持字典那样的通过键获取值。
- 作用

- 集合数据类型属于Python内置的数据类型，但不被重视，在很多书籍中甚至都看不到一点介绍。其实，集合是一种非常有用的数据结构，它的去重和集合运算是其它内置类型都不具备的功能，在很多场合有着非常重要的作用，比如网络爬虫。
- 我们都知道爬虫需要发散链接，一个页面连着另一个页面，不断爬取所有的超级链接，才能把整个站点爬取下来。然而在成千上万个页面链接中，有很大一部分可能是重复的链接或者循环互链，如果不对链接进行去重处理，那么爬虫要么陷入死循环内，要么就是出现错误。这个时候可以用集合的去重功能，保留一个曾经爬过页面的不重复的元素集合，每爬一个新链接，看看集合里是否曾经爬过，没有就开始爬，并将链接加入集合，爬过就忽略当前链接。在这里，用集合远比用列表或者字典要来得高效、节省得多。