Python的列表(List)操作

|  |  |
| --- | --- |
| 列表的申明 | example\_list = [“a”,1,{“a”:”b”}] |
| 列表增加元素 | .insert(index,obj)向指定index位置处添加元素  .append()在列表末尾添加元素 |
| 列表删除元素 | .remove(obj)删除第一个符合的元素  del example\_list[index] 删除index下标的元素  .pop(index) 删除index下标的元素  .pop() 删除列表最后一个元素  .clear() 清空列表 |
| 列表改变元素值 | Example\_list[index]=obj |
| 列表切片 | Example\_list[starting\_index,end\_index] 取出包含starting\_index直到但不包含end\_index的子列表 |
| 列表判断元素是否存在 | Obj in example\_list |
| 列表判断有多少个特定的元素 | Example\_list.count(obj) |
| 列表查找第一个符合指定元素下标 | Example\_list.index(obj)  如果没有该元素则报错。 |
| 列表扩充列表 | Example\_list.extend(another\_list) |
| 列表排序 | Example\_list.sort() |
| 列表反转 | Example\_list.reverse() |
| 列表的复制 | Example\_list.copy()  Copy函数复制一份完整的列表，从而复制以后列表于源列表指向不同地址，不相互影响。  1. 直接列表赋值，example\_list2 = example\_list 只会赋值地址，两个列表指向同一内存空间。  2. 列表的Copy函数默认只复制表层简单数据，复制前后的变量指向不同的内存地址。但是如果列表中含有列表则只会对内嵌列表复制地址。如果要完完全全对嵌套的列表也复制一份，则要引入原生库Copy，使用方法copy.deepcopy(obj)才会完完全全的复制。默认浅复制的原因是防止一下完全深复制，因为嵌套所占用的内存太多。 |

P.s

🡪

tuple(元组) 为只读列表，只有列表中index,count方法。

🡪

去除数组中所有的空格 (count () 与remove () 方法的联动) ：

Count () 可以用于返回有多少个空格，和remove () 每一次去除一个空格

for i in list (range (list.count(“ “))):

list.remove(“ “)

🡪

列表的排序使用内置函数sorted

sorted(iterable[,cmp,[,key[,reverse=True]]])

sorted函数第一个参数是一个iterable对象(可迭代对象包括列表，元组，字典等)，返回值是一个对iterable中元素进行排序后的列表，原对象不发生改变。默认排序为从小到大。

Python3中sorted只有两个命名参数key和reverse。

*key* **指定（非调用）**一个接受一个参数的函数作为key，用于表明比较的标准。

该函数接受一个参数，并且该函数的参数为iterable对象中的每一个对象。要求该函数返回一个值作为比较的依据

如果从Python2中迁移代码，使用cmp函数自定义比较规则，则使用functools.cmp\_to\_key()方法将原本的cmp函数包裹一下传入sorted的命名参数key中。

*例如：*

*>>> student\_tuples = [*

*... ('john', 'A', 15),*

*... ('jane', 'B', 12),*

*... ('dave', 'B', 10),*

*... ]*

*>>> sorted (student\_tuples, key=lambda student: student[2]) # 通过key指定的匿名函数应用在每一个iterable元素中，返回某个元素的index=2作为比较。所以等于执行按照学生年龄排序*

*[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]*

*reverse* 是一个布尔值。如果设置为 True，那么列表元素将按照每个比较反转进行排序。