```
#encoding=utf-8
#*****************************************
#*****************  列表              ****************
#*****************************************
#
#像字符串类型一样,列表类型也是序列式的数据类型,可以通过下标或者切片操作来访问
#某一个或者某一块连续的元素.然而,相同的方面也就这些,字符串只能由字符组成,而且是不
#可变的(不能单独改变它的某个值),而列表则是能保留任意数目的 Python 对象的灵活的容
器。

#数组类型所有的元素只能是一种类型.列表可以执行 pop,empt,sort,reverse 等操
#作.列表也可以添加或者减少元素.还可以跟其他的列表结合或者把一个列表分成几个.可以
对
#单独一个元素或者多个元素执行 insert,update,或者 remove 操作.

#创建列表类型数据并给它赋值
#写一个列表(空的或者有值的都行)然后赋给一个变量,列表是由方括号([])来定义的.

#alist=[123,'123',456]
#print(alist)
#-->[123, '123', 456]

#访问列表中的值
#列表的切片操作就像字符串中一样;切片操作符([])和索引值或索引值范围一起使用

#alist=[123,'123',456,['dave','is','dba']]
#print(alist)
#print(alist[0])
#print(alist[0:2])
#print(alist[3][1])
#-->
#[123, '123', 456, ['dave', 'is', 'dba']]
#123
#[123, '123']
#is

#更新列表
```

```
#你可以通过在等号的左边指定一个索引或者索引范围的方式来更新一个或几个元素,你也可以用 append()方法来追加元素到列表中去.
#alist=[123,'123',456,['dave','is','dba']]
#print(alist[1])
#alist[1]='anqing'
#print(alist[1])
#alist.append('This is end')
#print(alist)
#-->
#123
#anqing
#[123, 'anqing', 456, ['dave', 'is', 'dba'], 'This is end']

#删除列表中的元素或者列表(本身)
#要删除列表中的元素,如果你确切的知道要删除元素的素引可以用 del 语句,否则可以用 remove()方法.
#alist=[123,'123',456,['dave','is','dba']]
#del alist[1]
#print(alist)
#alist.remove(123)
#print(alist)
#-->
#[123, 456, ['dave', 'is', 'dba']]
#[456, ['dave', 'is', 'dba']]

#**********Part 1：操作符  **********
#标准类型操作符
#在使用比较操作符时,比较数字和字符串是很明了的,但是用在列表上时就不是那么简单了,
#列表比较操作有些狡猾,但是合乎逻辑.比较列表时也是用的内建的 cmp()函数,基本的比较逻辑
#是这样的:两个列表的元素分别比较,直到有一方的元素胜出,比如我们上面的例子,'abc'和
#'xyz'的比较直接决定了比较结果,在'abc'<'xyz'时,list1<list2,list2>=list3,元组类型在
#进行比较操作时跟列表遵循相同的逻辑.

#list1 = ['abc', 123]
#list2 = ['xyz', 789]
#list3 = ['abc', 123]
#print(list1 < list2)
#print(list2 < list3)
#print(list2 > list3 and list1 == list3)
#-->
#True
#False
#True
```

#序列类型操作符切片([] 和[:])
#列表的切片操作跟字符串的切片操作很像,不过列表的切片操作返回的是一个对象或者是几个对象的集合,
#而不是像字符串那样,返回一个字符或者一个子串.

#列表的切片操作也遵从正负索引规则,也有开始索引值,结束索引值,如果这两个值为空,默认也会分别指到序列的开始和结束位置.

```
#num_list = [43, -1.23, -2, 6.19e5]
#str_list = ['jack', 'jumped', 'over', 'candlestick']
#mixup_list = [4.0, [1, 'x'], 'beef', -1.9+6j]
#
#print(num_list[1])
#print(str_list[1:2])
#print(mixup_list[1][1])
#-->
#-1.23
#['jumped']
#x
```

#成员关系操作(in ,not in)
#列表中(同样适用于元组),我们可以检查一个对象是否是一个列表(或者元组)的成员

```
#mixup_list = [4.0, [1, 'x'], 'beef', -1.9+6j]
#print(4 in mixup_list)
#-->True
#print('x' in mixup_list)
#-->False
```

#注意,'x'并不属于 mixup_list, 因为'x'本身并不是 mixup_list 的一个成员, 而是
#mixup_list[1]的,mixup_list[1]也是一个列表类型.成员关系操作运算同样适用于元组类型.

#连接接操作符(+)
#连接操作符允许我们把多个列表对象合并在一起.注意,列表类型的连接操作也只能在同类
#型之间进行,换句话说,你不能把两个不同类型的对象连接在一起,即便他们都是序列类型也不行.
#必须指出,连接操作符并不能实现向列表中添加新元素的操作.

```
#num_list = [43, -1.23, -2, 6.19e5]
#str_list = ['jack', 'jumped', 'over', 'candlestick']
#mixup_list = [4.0, [1, 'x'], 'beef', -1.9+6j]
#print(num_list + mixup_list)
```

```
#print(str_list + num_list)
#-->
#[43, -1.23, -2, 619000.0, 4.0, [1, 'x'], 'beef', (-1.9+6j)]
#['jack', 'jumped', 'over', 'candlestick', 43, -1.23, -2, 619000.0]


#重复操作符( * )
#重复操作符可能更多的应用在字符串类型中,不过,列表和元组跟字符串同属序列类型,所以需要的时候也可以使用这一操作.
#num_list = [43, -1.23, -2, 6.19e5]
#print(num_list*2)
#-->[43, -1.23, -2, 619000.0, 43, -1.23, -2, 619000.0]

#列表类型操作符和列表解析
#列表可以使用大部分的对象和序列类型的操作符.此外，列表类型有属于自己的方法.
#列表才有的构建--列表解析.这种方法是结合了列表的方括弧和 for 循环,在逻辑上描述要创建的列表的内容

#print([ i * 2 for i in [8, -2, 5] ])
#print([ i for i in range(8) if i % 2 == 0 ])
#-->
#[16, -4, 10]
#[0, 2, 4, 6]

#************Part 2：内建函数 **********************

#序列类型函数 len()
#对字符串来说 len()返回字符串的长度,就是字符串包含的字符个数.对列表或者元组来说,
#它会像你想像的那样返回列表或者元组的元素个数,容器里面的每个对象被作为一个项来处理.

#num_list = [43, -1.23, -2, 6.19e5]
#print(len(num_list))
#-->4

#max() and min()
#max()和 min()函数在字符串操作里面用处不大,因为它们能对字符串做的只能是找出字符串中"最大"和"最小"的字符(按词典序),
#而对列表和元组来说,它们被定义了更多的用处.比如对只包含数字和字符串对象的列表,max()和 min()函数就非常有用,重申一遍,
#混合对象的结构越复杂返回的结构准确性就越差.然而,在有些情况下(虽然很少),这样的操作可以返回你需要的结果

#num_list = [43, -1.23, -2, 6.19e5]
```

```
#mixup_list = [4.0,'beef', -1.9+6j]
#print(max(num_list))
#print(min(num_list))
#-->
#619000.0
#-2
#print(max(mixup_list))
#print(min(mixup_list))
#-->
#TypeError: unorderable types: str() > float()


#sorted() and reversed()  示例
#s = ['They', 'stamp', 'them', 'when', "they're", 'small']
#for t in reversed(s):
#       print(t)
#print(sorted(s))
#-->
#small
#they're
#when
#them
#stamp
#They
#['They', 'small', 'stamp', 'them', "they're", 'when']

#enumerate() and zip()  示例
#albums = ['tales', 'robot', 'pyramid']
#for i, album in enumerate(albums):
#       print(i, album)
#-->
#0 tales
#1 robot
#2 pyramid

#fn = ['ian', 'stuart', 'david']
#ln = ['bairnson', 'elliott', 'paton']
#
#for i, j in zip(fn, ln):
#       print(i,j,end=' ')
#-->
#ian bairnson stuart elliott david paton
```

```
#sum() 示例
#a = [6, 4, 5]
#print(sum(a))
#print(sum(a,5))
#-->
#15
#20

#list() --列表 and tuple()--元组
#list()函数和 tuple()函数接受可迭代对象(比如另一个序列)作为参数,并通过浅拷贝数据
#来创建一个新的列表或者元组.虽然字符串也是序列类型的,但是它们并不是经常用于 list()
和
#tuple(). 更多的情况下,它们用于在两种类型之间进行转换,比如你需要把一个已有的元组转
#成列表类型的(然后你就可以修改它的元素了),或者相反.

#aList = ['tao', 93, 99, 'time']
#aTuple = tuple(aList)
#print(aList, aTuple)
#-->
#['tao', 93, 99, 'time'] ('tao', 93, 99, 'time')


#************Part 3：列表类型的内建函数 **********************

#List Method                Operation
#list.append(obj)           向列表中添加一个对象 obj
#list.count(obj)            返回一个对象 obj 在列表中出现的次数
#list.extend(seq)a          把序列 seq 的内容添加到列表中
#list.index(obj, i=0,
#                  j=len(list)) 返回 list[k] == obj 的 k 值,并且 k 的范围在 i<=k<j;否则引发
ValueError 异常.
#list.insert(index, obj) 在索引量为 index 的位置插入对象 obj.
#list.pop(index=-1)         删除并返回指定位置的对象,默认是最后一个对象
#list.remove(obj)           从列表中删除对象 obj
#list.reverse()             原地翻转列表
#list.sort(func=None,key=None,
#                reverse=False) 以指定的方式排序列表中的成员,如果 func 和 key 参数指定,则
按照指定的方式比较各个元素,
#                                    如果 reverse 标志被置为 True,则列表以反序排列.

#核心笔记:那些可以改变对象值的可变对象的方法是没有返回值的!
#Python 初学者经常会陷入一个误区:调用一个方法就返回一个值.最明显的例子就是
#sort():
#>>> music_media.sort()# 没有输出?
```

```
#>>>
#在使用可变对象的方法如 sort(),extend()和 reverse()的时候要注意,这些操作会在列表
#中原地执行操作,也就是说现有的列表内容会被改变,但是没有返回值!是的,与之相反,字符
串方法确实有返回值.


#************Part 4：列表的特殊特性     **********************
#用列表构建其他数据结构
#列表有容器和可变的特性,这使得它非常灵活,用它来构建其他的数据结构不是件难事.

#堆栈
#堆栈是一个后进先出(LIFO)的数据结构.

#下面这段示例是用 Python  列表实现的一个堆栈
#encoding=utf-8

#stack = []
#
#def pushit():
#       stack.append(input('Enter new string: ').strip())
#
#def popit():
#       if len(stack) == 0:
#            print('Cannot pop from an empty stack!')
#       else:
#            print('Removed [', stack.pop(), ']')
#
#def viewstack():
#       print(stack) # calls str() internally
#
#CMDs = {'u': pushit, 'o': popit, 'v': viewstack}
#
#def showmenu():
#       pr = """
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: """
#
#       while True:
#            while True:
#                 try:
#                      choice = input(pr).strip()[0].lower()
```

```
#                     except (EOFError,KeyboardInterrupt,IndexError):
#                         choice = 'q'
#
#                     print('\nYou picked: [%s]' % choice)
#
#                     if choice not in 'uovq':
#                         print('Invalid option, try again')
#                     else:
#                             break
#
#               if choice == 'q':
#                         break
#
#               CMDs[choice]()
#
#if __name__ == '__main__':
#         showmenu()

#运行结果示例：
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: U
#
#You picked: [u]
#Enter new string: Dave
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: U
#
#You picked: [u]
#Enter new string: is
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: U
#
#You picked: [u]
```

```
#Enter new string: DBA
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: V
#
#You picked: [v]
#['Dave', 'is', 'DBA']
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: O
#
#You picked: [o]
#Removed [ DBA ]
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: V
#
#You picked: [v]
#['Dave', 'is']
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: O
#
#You picked: [o]
#Removed [ is ]
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: O
#
```

```
#You picked: [o]
#Removed [ Dave ]
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: V
#
#You picked: [v]
#[]
#
#p(U)sh
#p(O)p
#(V)iew
#(Q)uit
#Enter choice: Q
#
#You picked: [q]


#队列
#队列是一种先进先出(FIFO)的数据类型。

#Python 队列 示例：quequ.py
#encoding=utf-8

#queue = []
#
#def enQ():
#       queue.append(input('Enter new string: ').strip())
#
#def deQ():
#       if len(queue) == 0:
#            print('Cannot pop from an empty queue!')
#       else:
#            print('Removed [', queue.pop(0), ']')
#
#def viewQ():
#       print(queue) # calls str() internally
#
#CMDs = {'e': enQ, 'd': deQ, 'v': viewQ}
#
#def showmenu():
```

```
#       pr = """
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: """
#
#       while True:
#               while True:
#                       try:
#                               choice = input(pr).strip()[0].lower()
#                       except (EOFError,KeyboardInterrupt,IndexError):
#                               choice = 'q'
#
#                       print('\nYou picked: [%s]' % choice)
#
#                       if choice not in 'devq':
#                               print('Invalid option, try again')
#                       else:
#                               break
#
#               if choice == 'q':
#                               break
#
#               CMDs[choice]()
#
#if __name__ == '__main__':
# showmenu()
#
#测试数据：
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: E
#
#You picked: [e]
#Enter new string: Dave
#
# (E)nqueue
# (D)equeue
# (V)iew
```

```
# (Q)uit
#
# Enter choice: E
#
#You picked: [e]
#Enter new string: is
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: E
#
#You picked: [e]
#Enter new string: DBA
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: V
#
#You picked: [v]
#['Dave', 'is', 'DBA']
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: D
#
#You picked: [d]
#Removed [ Dave ]
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: V
#
```

```
#You picked: [v]
#['is', 'DBA']
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: D
#
#You picked: [d]
#Removed [ is ]
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: D
#
#You picked: [d]
#Removed [ DBA ]
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice: V
#
#You picked: [v]
#[]
#
# (E)nqueue
# (D)equeue
# (V)iew
# (Q)uit
#
# Enter choice:
```