列表导读

从列表开始，我们将陆续学习元组，集合，字典这4中容器类型数据。容器类型数据，只是望文生义，你大概能够猜测出这4种数据类型的作用，他们如同容器一样，可以存储int，float，bool，str。每一种容器类型数据都有各自的存储方式和要求，因此，他们适用于不同的场景。

什么是列表?

在讲解列表之前，我们先做一个简单的智力测验，我每天给你一个小球，每个小球都一模一样，且不允许你做任何标记，在第100天的时候，我要求你拿出来我第35天给你的那个小球，请你思考，你该如何存放小球，才能保证我说出天数，你拿出天数所对应的小球。

稍动一下脑筋，你就应该可以想出方法来，所有的小球都按顺序排放，第1天的小球排第1位，第2天的小球排第2位，以此类推，那么当我要求你找出第35天给你的小球时，你需要做的是从第1个小球开始数数，数到35时，这个小球就是我想要的。

列表的数据存储与访问与刚才所讲的小球摆放有着相同的道理，一切只与顺序有关，提到顺序，是不是想到了有序，是不是想到了索引？没错，列表里也有索引的概念，列表是数据的有序集合。

列表 (list)

python的列表(list)是使用频率最高的一种数据结构， 很像C++中的vector 和 java 中的ArrayList， 是大小可动态变换的数组。 单个列表可以同时存储多种数据类型， 而不一定非得是同一种数据类型。

列表是有序的， 它根据确定的序列对列表里的数据进行索引， 和其他语言一样， 索引从0开始， 列表里每个数据都有自己确切的位置， 对列表里元素的任何操作都离不开索引， 因此， 对列表进行迭代是非常常见的操作。

1. 创建列表

```
list1 = [1, 2, 3, 4, 5 ]
list2 = [1, 2, '3', True]
list3 = [[1,2,3], True, 1]
```

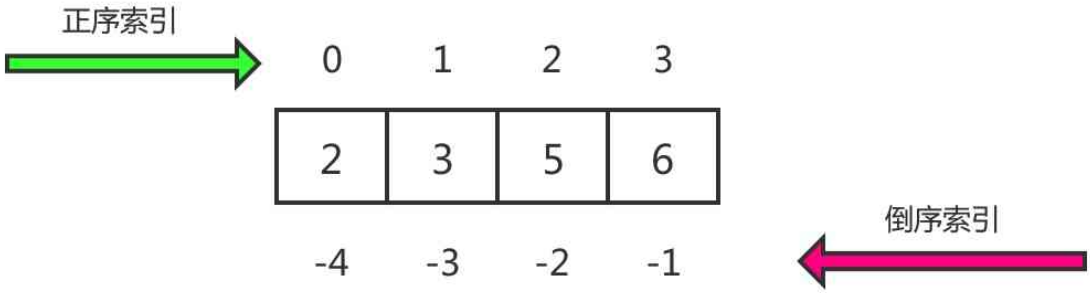
列表里可以存放任意类型的数据，每个数据之间，用逗号分隔，整个列表被包裹在一对中括号[]里，如果你定义lst = [],[]表示一个空列表。

2. 索引的概念

列表是数据的有序集合，对于列表里数据的操作，都必须通过索引来完成，回想一下上一篇的智力测试，所有的小球按顺序排放，顺序就是小球的索引，你可以将索引就理解为顺序，只是有一点务必牢记，计算机里的索引，或者说顺序，都是从0开始的

```
lst = [2, 3, 5, 6]
```

下面这张图很好的阐述了索引的概念



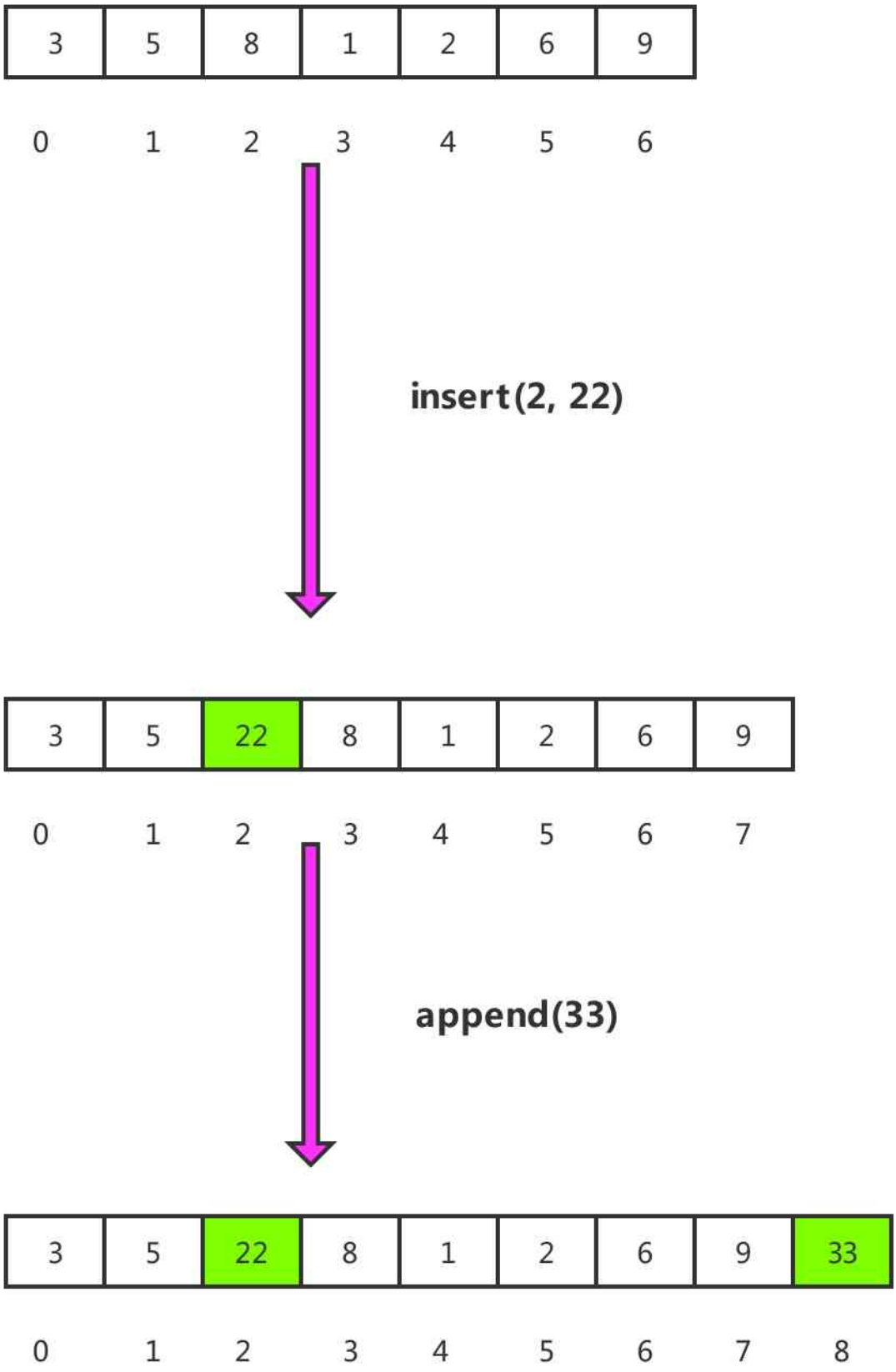
所谓正序索引，就是从左向右看，倒序索引就是从右向坐看，由左至右，从0开始递增，从右向左，从-1开始递减，python既支持正序索引，也支持倒序索引。

3.向列表中新增数据

向列表中新增数据有两种方法，一个是insert,一个是append,前者需要指定插入的位置，而append则默认在列表的尾部插入数据，下面的代码演示如何使用这两个方法新增数据

```
lst = [3, 5, 8, 1, 2, 6, 9]
lst.insert(2, 22)  # 将22插入到索引为2的位置上
lst.append(33)     # 将33增加到列表尾部
print(lst)
```

下图很好的解释了这两种新增方法的区别



4. 访问列表里的数据

访问列表里的数据，必须提供索引，假设你想输出列表里的第0个元素，和倒数第一个元素，你应该这样操作

```
lst = [3, 5, 8, 1, 2, 6, 9]

print(lst[0])
print(lst[-1])
```

程序输出结果为

```
3
9
```

根据索引获取列表里的数据，需要使用中括号[],在[]里填写你想要获取数据的索引，如果这个索引超出了范围，就会报错

```
lst = [3, 5, 8, 1, 2, 6, 9]
print(lst[12])
```

报错内容为

```
Traceback (most recent call last):
  File "/Users/kwsy/PycharmProjects/pythonclass/wx_monitor/main.py", line 2, in <module>
    print(lst[12])
IndexError: list index out of range
```

这是因为列表里根本没有这个索引，你想要找的索引已经超出了列表索引的范围。

5. 遍历列表

请注意, for循环的内容要等到第5章程序控制章节才会讲解, 如果你还没有掌握for循环, 这段内容你可以等到学完第5章循环控制以后在来学习

遍历索引，需要使用for循环，你可以写出至少3种for循环，每一个都可以遍历列表里所有的元素

方法1，通过索引遍历

```
lst = [3, 5, 8, 1, 2, 6, 9]

for i in range(len(lst)):
    print(lst[i])
```

方法2，通过迭代器遍历

```
lst = [3, 5, 8, 1, 2, 6, 9]

for item in lst:
    print(item)
```

方法3， 通过enumerate函数遍历

```
lst = [3, 5, 8, 1, 2, 6, 9]

for index, item in enumerate(lst):
    print(index, item)
```

关于enumerate函数，可以在内置函数章节里学习

6. 删除列表里的元素

删除列表里的元素有两种方法，一种是根据索引来删除，一种是根据值来删除，先说根据索引删除

根据索引删除列表里的元素

列表的pop方法可以删除指定索引位置上的数据

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

lst.pop(0)      # 删除索引为0的元素
print(lst)
```

pop只会删除指定索引位置的元素，程序输出结果为

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

除了使用pop方法，你还可以使用del 关键字

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

del lst[0]
print(lst)
```

del lst[0] 等价于 lst.pop(0)

根据值删除列表里的元素

根据值删除，使用remove方法，remove会删除列表里指定的值，比如你想删除4

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

lst.remove(4)
print(lst)
```

需要注意的是，remove方法一次只会删除一个值，列表里有两个4，它会删除索引最小的那个4，程序输出结果为

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

7. 和列表相关的几个重要内置函数

7.1 len,计算列表长度

len函数已经不是第一次接触了，len函数不仅能求列表的长度，还能求字符串的长度，集合的长度，元组的长度，字典的长度，他们的用法如此统一，减轻了我们学习的难度

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

7.2 max，返回数列的最大值

max不仅可以作用于列表，也可以作用于元组，它会返回数列里的最大值

```
lst = [3, 4, 1, 4, 2]
print(max(lst))
```

肉眼可见，4是列表的最大值，max函数返回值为4

7.3 min, 返回数列的最小值

min可以作用于列表，也可以作用于元组，它返回数列里的最小值

```
lst = [3, 4, 1, 4, 2]
print(min(lst))
```

1是列表的最小值，min函数返回值为1

7.4 sum，返回数列里所有数据的和

同样，sum函数可以作用于列表，也可以作用于元组，它返回数列里所有数的总和

```
lst = [3, 4, 1, 4, 2]
print(sum(lst))
```

列表里所有元素的总和是14，这正是sum函数的返回值

8. 列表操作符

操作符	功能作用
+	连接两个列表
*	重复列表内容
in	成员操作符，判断某个数据是否在列表中
not in	成员操作符，判断某个数据是否在列表中

列表对 + 和 * 的操作符与字符串相似,现在，在交互式解释器里跟随我一起操作

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [4, 5, 6]
>>> lst1 + lst2
[1, 2, 3, 4, 5, 6]
>>> lst1*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 3 in lst1
True
>>> 4 not in lst2
False
```

嵌套列表

python嵌套列表是一个对初学者稍有困难的知识点，因此，我专门用一篇教程来对它进行讲解。所谓嵌套，是指列表里出现了其他容器类型数据，比如下面的例子

```
lst = [1, [1, 5], 3, [9, 8, [1, 3]], 5]
print(lst[3][2][0])
print(lst[1:4])
```

面对这种嵌套的列表，很多人表示很懵逼，现在，跟着我的节奏，一点点搞清楚它。

编程是一件很枯燥的事情，因为它要求你逻辑严谨，一丝不苟，这样，肯定没有天马行空的幻想来的让人舒服。

在前面讲列表的创建时，特别强调了，列表用中括号创建，列表里的数据用逗号分隔，从左往右看，第一个数据是1， 这个没有问题，关键是第二个数据，到底是[1, 5], 还是[1 ? 他们都被逗号分隔了

如果你认真思考就明白，第二个数据是[1, 5],因为[1, 5]是一个列表，是一个数据，而[1 不是一个数据，我们已经学过的数据类型里没有这种数据类型。

按照上面的思路去思考，列表里的数据如下

索引	数据
0	1
1	[1, 5]
2	3
3	[9, 8, [1, 3]]
4	5

在此基础上理解lst[3][2][0]

1. lst[3] 的值是[9, 8, [1, 3]]
2. [9, 8, [1, 3]] 是一个列表，列表里有3个数据，索引为2的数据是[1, 3]
3. [1, 3]是一个列表，列表里有两个数据，索引为0的数据是1
4. print(lst[3][2][0]) 语句输出的结果是1

现在，请不用代码，自己手写出下面语句的结果

1. print(lst[1:4])
2. print(lst[3][2:])
3. print(lst[-2][1:2])

答案是

```
[[1, 5], 3, [9, 8, [1, 3]]]
[[1, 3]]
[8]
```

列表切片操作

python列表的切片操作，是一个使用频率非常高的技术，你应该认真学习掌握

所谓切片，就是从列表中截取某一部分，它的一般形式为 [start:end), 左闭右开. 对于列表的操作，离不开索引，做切片操作时，就必须指定截取的起始位置和结束位置

1. 指定开始和结束位置

```
lst = [3, 4, 1, 4, 2, 5, 8]

lst1 = lst[3:6]
print(lst1)
```

程序输出结果为

```
[4, 2, 5]
```

切片操作的一般模式是[start:end],shart 和 end 所指的都是索引，截取时，end索引位置的元素不会被截取，这一点尤其要注意，比如上面的示例中

```
lst1 = lst[3:6]
```

索引为6的元素是8，8没有出现在结果里，截取范围是3到6，6-3=3，最终切片截取的列表长度也正好是3

在使用切片时，也可以使用倒序索引

```
lst = [3, 4, 1, 4, 2, 5, 8]

lst1 = lst[2:-2]
print(lst1)
```

程序输出结果

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

2. 指定开始位置，不指定结束位置

```
lst = [3, 4, 1, 4, 2, 5, 8]

lst1 = lst[2:]
print(lst1)
```

如果不指定结束位置，那么切片操作会从开始位置截取到列表末尾，程序输出结果为

```
[1, 4, 2, 5, 8]
```

3. 指定结束位置，不指定开始位置

```
lst = [3, 4, 1, 4, 2, 5, 8]

lst1 = lst[:5]
print(lst1)
```

如果不指定开始位置，那么默认开始位置为0，程序输出结果为

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

4. 切片操作允许索引超出范围

```
lst = [3, 4, 1, 4, 2, 5, 8]

lst1 = lst[:11]
print(lst1)
```

上面的代码中，结束位置的索引设置为11，显然已经超出了列表索引的范围，但是却没有引发错误，对于这种情况，切片操作自身做了处理，如果结束位置的索引超出索引范围，那么就以列表长度作为结束位置

5. 指定切片间隔

关于切片间隔，已经在[字符串切片操作]做过讲解，本文不做赘述，只给出示例代码，想要了解切片间隔，可以去看这篇文章。

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

print(lst[::-2])    # [5, 3, 1]
print(lst[::-1])    # [5, 4, 3, 2, 1]
```

6. 切片操作应用示例---分组

```
lst = [2, 3, 4, 1, 5, 2, 7, 1, 8, 9, 10, 31, 22, 34]
```

上面定义了一个列表，现在要求你从第一个元素开始，每三个一组求和，将所求得的和放入新列表sum_lst

示例代码

```
lst = [2, 3, 4, 1, 5, 2, 7, 1, 8, 9, 10, 31, 22, 34]

step = 3
sum_lst = []
for i in range(0, len(lst), step):
    tmp = lst[i: i+step]
    sum_lst.append(sum(tmp))

print(sum_lst)
```

列表方法详解

下表是列表方法及功能说明

方法	功能
count()	统计某个元素在列表中出现的次数
append()	在列表末尾添加新的对象
extend	在列表末尾一次性追加另一个序列中的多个值
index	返回一个元素在列表中第一次出现的位置索引
insert()	将对象插入列表中的指定位置
pop()	删除列表中指定索引的元素，默认删除最后一个并返回该值
remove()	移除列表中某个值的第一个匹配项
reverse()	翻转列表
sort()	对列表进行排序

count方法

python列表(list)的count方法可以统计出某个元素在列表中出现的次数, 如果元素不存在于列表中，则返回0

```
lst = [1, 1, 2, 2, 3, 3, 3]
print(lst.count(1))     # 2
print(lst.count(2))     # 2
print(lst.count(4))     # 0
```

你可以自己实现一个相同功能的函数，来提升自己的能力，不过这要等到你学会函数以后，本文提供一份参考代码

```
lst = [1, 1, 2, 2, 3, 3, 3]

def my_count(lst, target):
    count = 0
    for item in lst:
        if item == target:
            count += 1

    return count

print(my_count(lst, 1))     # 2
```

append()方法

python列表的append方法可以在列表末尾向列表增加新的元素，列表是一个有索引的数据结构，索引从0开始，因此会有头尾的概念，与之相对应的，还有一个insert方法，可以在指定索引位置插入新的元素

```python
lst = []
lst.append(3)
lst.append(2)
lst.append(1)
print(lst)   # [3, 2, 1]
```

append是追加的意思，使用该方法时，无需关心列表目前的元素个数，只需明确一点，新的元素一定是在列表末尾增加， append方法没有返回值，但会修改原来的列表

extend方法

python列表(list)的extend方法可以在列表末尾一次性追加另一个序列中的多个值，这里要特别强调一点，extend方法的参数不仅限于列表，还可以是元组，字符串，集合，字典， 等序列

```python
lst = []
lst.extend([1, 2])      # 列表
print(lst)

lst.extend((3, 4))      # 元组
print(lst)

lst.extend('56')        # 字符串
print(lst)

lst.extend(set([1, 3, 4]))      # 集合
print(lst)

lst.extend({'age': 5})     # 字典
print(lst)
```

程序输出结果为

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

当参数是字典时，只会将字典的key增加到列表中， extend方法没有返回值，但会在原列表里添加新的列表内容， 可以扩增原列表的数据

index方法

python列表(list)的index方法返回一个元素在列表中第一次出现的位置索引，如果这个元素不存在，则会引发ValueError异常，下面就是一个会引发异常的示例

```python
lst = [3, 5, 3, 6, 5]
print(lst.index(2))
```

2 在列表中并不存在，index方法会抛出ValueError异常。

如果元素存在于列表中，则会返回第一次出现时所在的索引

```
lst = [3, 5, 3, 6, 5]
print(lst.index(5))     # 1
```

列表中有两个5， 他们的索引分别是1 和 4， index方法返回5第一次出现时所在的索引

insert方法

列表的insert方法可以将元素插入到指定索引位置，如果索引不存在，则会插入到列表的末尾，我们可以将append方法看做是insert方法的一种特例，append永远将新的元素写入到列表的末尾。

```
lst = [1, 2, 3]
lst.insert(0, 6)

print(lst)      # [6, 1, 2, 3]
```

列表原本有3个元素，最大索引为2， 如果使用insert方法时，所指定的索引大于2，那么就会默认写入到列表的末尾；由于列表还支持反向索引，因此，如果索引位置小于-3， 那么就会在列表的头部写入新数据

```
lst = [1, 2, 3]
lst.insert(-4, 6)

print(lst)      # [6, 1, 2, 3]
```

pop方法

python列表 (list) pop方法删除列表中指定索引的元素并且返回该元素，如果不指定索引位置，则默认删除列表中最后一个元素

```
lst = [1, 2, 3]
print(lst.pop())    # 默认删除最后一个元素, 3
print(lst.pop(0))       # 删除索引为0的元素, 1
print(lst)          # [2]
```

如果列表是空的，使用pop方法会怎样呢？由于没有元素可以被删除，因此程序会报错

```
lst = []
lst.pop()
```

报错内容为

```
Traceback (most recent call last):
  File "/Users/kwsy/kwsy/coolpython/demo.py", line 2, in <module>
    lst.pop()
IndexError: pop from empty list
```

一个空列表使用pop方法会引发IndexError异常。

remove方法

列表的remove方法会将指定元素从列表中删除，如果这个元素在列表中存在多个，则删除索引最小的那一个，也就是列表中第一个与指定元素相同的数据。

```
lst = [3, 5, 3, 6, 5]

lst.remove(6)
print(lst)   # [3, 5, 3, 5]

lst.remove(5)
print(lst)   # [3, 3, 5]
```

如果被删除的元素在列表中不存在，则会引发ValueError异常

```
lst = [3, 5, 3, 6, 5]
lst.remove(7)   # 列表中没有7
```

reverse方法

列表reverse方法可以将列表里的元素翻转，这种翻转只是将首尾元素对换位置，千万不要把列表翻转和列表排序联系在一起，他们没有任何关系

```
lst = [1, 2, 3]
lst.reverse()
print(lst)      # [3, 2, 1]
```

其实，翻转的算法非常容易就能写出来，下面是一个简单的实现

```
lst = [1, 2, 3]

for i in range(0, len(lst)//2):
    lst[i], lst[len(lst)-1-i] = lst[len(lst)-1-i], lst[i]

print(lst)
```

sort方法

列表的sort方法可以对列表里的元素进行排序， 它有两个重要的参数，一个是key， 一个是reverse，key来设置一个函数用于返回用于比较大小的数值， reverse参数决定排序是从小到大还是从大到小， 下面是一个简单的sort方法使用示例。

```
lst = [2, 1, 4, 3]
lst.sort()
print(lst)  # [1, 2, 3, 4]
```

sort方法的定义如下

```
def sort(self, key=None, reverse=False):
    pass
```

参数key指定了排序所用的数值，reverse设置排序的方法，默认为False表示从小到大排序，对上面的列表从大到小排序可以这样写

```
lst = [2, 1, 4, 3]
lst.sort(reverse=True)
print(lst)      # [4, 3, 2, 1]
```

列表里的元素都是整数，因此用不上key这个参数，如果列表里的元素无法直接进行大小比较，就必须指定参数key，参数key必须是一个函数，返回列表里元素用于比较大小的值

```
lst = ['3', 5, '1', 2]
lst.sort(key=lambda x:str(x))
print(lst)   # ['1', 2, '3', 5]
```

列表中的元素有字符串（可以转换成int），也有整数，字符串与整数无法直接比较大小，在比较大小时，比较他们转成字符串之后的结果。除了使用lambda表达式，还可以使用自定义函数

```
lst = [(1, 2), (3, 2), (2, 4)]
def compare(x):
    return x[0]

lst.sort(key=compare)
print(lst)  # [(1, 2), (2, 4), (3, 2)]
```

列表里的数据是元组，元组之间无法直接比较大小，因此使用自定义函数compare指定使用元组的第一个元素代表元组进行大小比较来排序

python列表练习题

列表的练习题非常多，多到我相信会有很大一部分人会跳过这篇教程，不过请放心，不论你是因为图省事还是因为太懒了而跳过本篇教程，将来你都会再回到这里，乖乖的做这些练习题。

编程，既是一门知识，也是一项技能，仅从学习知识的角度看，许多知识都是一看就懂的，但作为一项技能，它需要你反复练习以达到熟练的程度。

就好比骑自行车，坐上去，两脚蹬脚踏板，自行车就可以移动了，这是知识，别人一说你就懂。但是，骑上去就发现，你无法掌握平衡，只有多加练习，才能真正的掌握骑自行车的技术。

1. 列表基础考察

已知一个列表
lst = [1,2,3,4,5]

1. 求列表的长度
2. 判断6 是否在列表中
3. lst + [6, 7, 8] 的结果是什么?
4. lst*2 的结果是什么
5. 列表里元素的最大值是多少
6. 列表里元素的最小值是多少
7. 列表里所有元素的和是多少
8. 在索引1的位置新增一个的元素10

9. 在列表的末尾新增一个元素20

答案如下

```
。。。。。。
```

以上都是对列表基础操作，所用到的每一个函数，列表的每一个方法，都是需要你熟记于心的

2. 修改列表

lst = [1, [4, 6], True]
请将列表里所有数字修改成原来的两倍

答案如下

```
。。。。。
```

你以为存在一个函数，其功能便是将列表里所有的数据都变成原来的两倍，这样才显得变成语言是一个非常神奇的东西，但是很遗憾的告诉你，那些神奇的东西都是程序员自己实现的。

想要修改列表里的数据，必须通过索引对其重新赋值，上面的方法很low，你也可以写一个函数来实现这个功能，我们假设要处理的列表里只int,float,bool,和list数据，不管嵌套几层list，这个函数都应该能正确处理，下面是一段示例代码

```
def double_list(lst):
    for index, item in enumerate(lst):
        if isinstance(item, bool):
            continue
        if isinstance(item, (int, float)):
            lst[index] *= 2
        if isinstance(item, list):
            double_list(item)


if __name__ == '__main__':
    lst = [1, [4, 6], True]
    double_list(lst)
    print(lst)
```

现在，我们还没有学习到函数，更没有学习到递归函数，这个练习题，你只掌握直接通过索引修改列表即可，等到学习函数后，可以再回到这里做这个练习题。

3. 合并列表

lst = [1,2,3]
lst2 = [4,5,6]
不使用 + 号运算符，将lst2合并到lst的末尾，并思考，这个过程中，是否产生了新的列表

答案

```
。。。。。
```

这个过程中不会产生新的列表，最直观的检验方式就是print(id(lst)),合并前后，lst的内存地址都没有发生变化，只是列表里的内容发生了变化

4. 统计练习

列表lst 内容如下

```
lst = [2, 5, 6, 7, 8, 9, 2, 9, 9]
```

请写程序完成下列题目

1. 找出列表里的最大值
2. 找出列表里的最小值
3. 找出列表里最大值的个数
4. 计算列表里所有元素的和
5. 计算列表里元素的平均值
6. 计算列表的长度
7. 找出元素6在列表中的索引

答案

```
。。。。。
```

这道题考察的是你对内置函数的理解和运用

下面的题目不允许写代码，仅凭思考来回答

1. lst[2:4] 的值是什么
2. lst[1: -3]的值是什么
3. lst[-5]的值是什么
4. lst[:-4] 的值是什么
5. lst[-4:] 的值是什么

这个题目主要考察你对列表切片操作的理解

```
。。。。。
```

列表的切片操作，最关键的一点在于左闭右开，结束位置的数据不会列入结果中

5. 列表操作练习

列表lst 内容如下

```
lst = [2, 5, 6, 7, 8, 9, 2, 9, 9]
```

请写程序完成下列操作

1. 在列表的末尾增加元素15
2. 在列表的中间位置插入元素20
3. 将列表[2, 5, 6]合并到lst中
4. 移除列表中索引为3的元素
5. 翻转列表里的所有元素
6. 对列表里的元素进行排序，从小到大一次，从大到小一次

答案

```
。。。。。。
```

6. 复杂列表练习

列表lst 内容如下

```
lst = [1, 4, 5, [1, 3, 5, 6, [8, 9, 10, 12]]]
```

不写任何代码，仅凭思考推理来回答下列问题

1. 列表lst的长度是多少
2. 列表lst中有几个元素
3. lst[1] 的数据类型是什么
4. lst[3]的数据类型是什么
5. lst[3][4] 的值是什么
6. 如果才能访问到 9 这个值
7. 执行lst[3][4].append([5, 6])后，列表lst的内容是什么，手写出来
8. lst[-1][-1][-2]的值是什么
9. lst[-2]的值是什么
10. len(lst[-1]) 的值是什么
11. len(lst[-1][-1])的值是什么
12. lst[-1][1:3] 的值是什么
13. lst[-1][-1][1:-2]的值是什么

```
答案：
。。。
```