0 数据类型

1 按是否是容器分

简单数据类型

- 整型`<class 'int'>`
- 浮点型`<class 'float'>`
- 布尔型`<class 'bool'>`

容器数据类型

- 列表`<class 'list'>`
- 元组`<class 'tuple'>`
- 字典`<class 'dict'>`
- 集合`<class 'set'>`
- 字符串`<class 'str'>`

2 按是否是可变数据

不可变数据（3个）：

- Number（数字）
- String（字符串）
- Tuple（元组）

可变数据（3个）：

- List（列表）
- Dictionary（字典）
- Set（集合）

1. 列表的定义

列表是有序集合，没有固定大小，能够保存任意数量任意类型的 Python 对象，语法为 `[元素1，元素2，...，元素n]`。

1. 关键点是中括号 []和逗号 ,
2. 中括号把所有元素绑在一起
3. 逗号 将每个元素一一分开

2. 列表的创建

- 创建一个普通列表

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(x, type(x))
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'] <class 'list'>

x = [2, 3, 4, 5, 6, 7]
print(x, type(x))
# [2, 3, 4, 5, 6, 7] <class 'list'>
```

- 利用`range()`创建列表

【例子】

```
x = list(range(10))
print(x, type(x))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>

x = list(range(1, 11, 2))
print(x, type(x))
# [1, 3, 5, 7, 9] <class 'list'>

x = list(range(10, 1, -2))
print(x, type(x))
# [10, 8, 6, 4, 2] <class 'list'>
```

- 利用推导式创建列表

【例子】

```
x = [0] * 5
print(x, type(x))
# [0, 0, 0, 0, 0] <class 'list'>

x = [0 for i in range(5)]
print(x, type(x))
# [0, 0, 0, 0, 0] <class 'list'>

x = [i for i in range(10)]
print(x, type(x))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>

x = [i for i in range(1, 10, 2)]
print(x, type(x))
# [1, 3, 5, 7, 9] <class 'list'>

x = [i for i in range(10, 1, -2)]
print(x, type(x))
# [10, 8, 6, 4, 2] <class 'list'>

x = [i ** 2 for i in range(1, 10)]
print(x, type(x))
# [1, 4, 9, 16, 25, 36, 49, 64, 81] <class 'list'>

x = [i for i in range(100) if (i % 2) != 0 and (i % 3) == 0]
print(x, type(x))

# [3, 9, 15, 21, 27, 33, 39, 45, 51, 57, 63, 69, 75, 81, 87, 93, 99] <class 'list'>
```

- 创建一个 4×3的二维数组

【例子】

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [0, 0, 0]]
print(x, type(x))
# [[1, 2, 3], [4, 5, 6], [7, 8, 9], [0, 0, 0]] <class 'list'>

for i in x:
    print(i, type(i))
# [1, 2, 3] <class 'list'>
# [4, 5, 6] <class 'list'>
# [7, 8, 9] <class 'list'>
# [0, 0, 0] <class 'list'>

x = [[0 for col in range(3)] for row in range(4)]
print(x, type(x))
# [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x[0][0] = 1
print(x, type(x))
# [[1, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x = [[0] * 3 for row in range(4)]
print(x, type(x))
# [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x[1][1] = 1
print(x, type(x))
# [[0, 0, 0], [0, 1, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>
```

注意：

由于list的元素可以是任何对象，因此列表中所保存的是对象的指针。即使保存一个简单的`[1,2,3]`，也有3个指针和3个整数对象。

`x = [a] * 4`操作中，只是创建4个指向list的引用，所以一旦`a`改变，`x`中4个`a`也会随之改变。

【例子】

```
x = [[0] * 3] * 4
print(x, type(x))
# [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x[0][0] = 1
print(x, type(x))
```

```
# [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]] <class 'list'>

a = [0] * 3
x = [a] * 4
print(x, type(x))
# [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x[0][0] = 1
print(x, type(x))
# [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]] <class 'list'>
```

- 创建一个混合列表

【例子】

```
mix = [1, 'lsgo', 3.14, [1, 2, 3]]
print(mix, type(mix))
# [1, 'lsgo', 3.14, [1, 2, 3]] <class 'list'>
```

- 创建一个空列表

【例子】

```
empty = []
print(empty, type(empty))  # [] <class 'list'>
```

3. 向列表中添加元素

列表不像元组，列表内容可更改 (mutable)，因此附加 (append, extend)、插入 (insert)、删除 (remove, pop) 这些操作都可以用在它身上。

3.1 append()

list.append(obj) 在列表末尾添加新的对象，只接受一个参数，参数可以是任何数据类型，被追加的元素在 list 中保持着原结构类型。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.append('Thursday')
print(x)
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Thursday']

print(len(x))  # 6
```

此元素如果是一个 list，那么这个 list 将作为一个整体进行追加，注意append()和extend()的区别。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.append(['Thursday', 'Sunday'])
print(x)
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', ['Thursday', 'Sunday']]

print(len(x))  # 6
```

3.2 extend()

list.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。

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.extend(['Thursday', 'Sunday'])
print(x)
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Thursday', 'Sunday']

print(len(x))  # 7
```

严格来说 append 是追加，把一个东西整体添加在列表后，而 extend 是扩展，把一个东西里的所有元素添加在列表后。

3.3 insert()

list.insert(index, obj) 在编号 index 位置插入 obj。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.insert(2, 'Sunday')
print(x)
# ['Monday', 'Tuesday', 'Sunday', 'Wednesday', 'Thursday', 'Friday']

print(len(x))  # 6
```

4. 删除列表中的元素

4.1 remove()

`list.remove(obj)` 移除列表中某个值的第一个匹配项。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.remove('Monday')
print(x)  # ['Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

4.2 pop()

`list.pop([index=-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。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
y = x.pop()
print(y)  # Friday

y = x.pop(0)
print(y)  # Monday

y = x.pop(-2)
print(y)  # Wednesday
print(x)  # ['Tuesday', 'Thursday']
```

`remove` 和 `pop` 都可以删除元素，前者是指定具体要删除的元素，后者是指定一个索引。

4.3 del()

- `del var1[, var2 ……]` 删除单个或多个对象。

【例子】

如果知道要删除的元素在列表中的位置，可使用`del`语句。

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
del x[0:2]
print(x)  # ['Wednesday', 'Thursday', 'Friday']
```

如果你要从列表中删除一个元素，且不再以任何方式使用它，就使用`del`语句；如果你要在删除元素后还能继续使用它，就使用方法`pop()`。

5. 获取列表中的元素

5.1 按索引值获取元素

- 通过元素的索引值，从列表获取单个元素，注意，列表索引值是从0开始的。
- 通过将索引指定为-1，可让Python返回最后一个列表元素，索引 -2 返回倒数第二个列表元素，以此类推。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', ['Thursday', 'Friday']]
print(x[0], type(x[0]))  # Monday <class 'str'>
print(x[-1], type(x[-1]))  # ['Thursday', 'Friday'] <class 'list'>
print(x[-2], type(x[-2]))  # Wednesday <class 'str'>
```

5.2 切片

切片的通用写法是 `start : stop : step` （左闭右开）

- 情况 1 - “start :”

- 以 step 为 1 (默认) 从编号 start 往列表尾部切片。

【例子】

```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(x[3:])  # ['Thursday', 'Friday']
print(x[-3:])  # ['Wednesday', 'Thursday', 'Friday']
```

- 情况 2 - ": stop"
- 以 step 为 1 (默认) 从列表头部往编号 stop 切片。

【例子】

```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[:3])  # ['Monday', 'Tuesday', 'Wednesday']
print(week[:-3])  # ['Monday', 'Tuesday']
```

- 情况 3 - "start : stop"
- 以 step 为 1 (默认) 从编号 start 往编号 stop 切片。

【例子】

```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[1:3])  # ['Tuesday', 'Wednesday']
print(week[-3:-1])  # ['Wednesday', 'Thursday']
```

- 情况 4 - "start : stop : step"
- 以具体的 step 从编号 start 往编号 stop 切片。注意最后把 step 设为 -1，相当于将列表反向排列。

【例子】

```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[1:4:2])  # ['Tuesday', 'Thursday']
print(week[:4:2])  # ['Monday', 'Wednesday']
print(week[1::2])  # ['Tuesday', 'Thursday']
print(week[::-1])
# ['Friday', 'Thursday', 'Wednesday', 'Tuesday', 'Monday']
```

- 情况 5 - ":"
- 复制列表中的所有元素（浅拷贝）。

【例子】

```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[:])
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

【例子】浅拷贝与深拷贝

```
list1 = [123, 456, 789, 213]
list2 = list1
list3 = list1[:]

print(list2)  # [123, 456, 789, 213]
print(list3)  # [123, 456, 789, 213]
list1.sort()
print(list2)  # [123, 213, 456, 789]
print(list3)  # [123, 456, 789, 213]

list1 = [[123, 456], [789, 213]]
list2 = list1
list3 = list1[:]
print(list2)  # [[123, 456], [789, 213]]
print(list3)  # [[123, 456], [789, 213]]
list1[0][0] = 111
print(list2)  # [[111, 456], [789, 213]]
print(list3)  # [[111, 456], [789, 213]]
```

6 浅拷贝和深拷贝

6.1 浅拷贝

- 1、对于 **不可** 变类型 Number String Tuple,浅复制仅仅是地址指向，不会开辟新空间。
- 2、对于 **可** 变类型 List、Dictionary、Set，浅复制会开辟新的空间地址(仅仅是最顶层开辟了新的空间，里层的元素地址还是一样的)，进行浅拷贝

- 3、浅拷贝后，改变原始对象中为可变类型的元素的值，会同时影响拷贝对象的；改变原始对象中为不可变类型的元素的值，只有原始类型受影响。（操作拷贝对象对原始对象的也是同理）

6.2 深拷贝

- 1、浅拷贝，除了顶层拷贝，还对子元素也进行了拷贝（本质上递归浅拷贝）
- 2、经过深拷贝后，原始对象和拷贝对象所有的子元素地址都是独立的了
- 3、可以用分片表达式进行深拷贝
- 4、字典的copy方法可以拷贝一个字典

7. 列表的常用操作符

- 等号操作符：==
- 连接操作符 +
- 重复操作符 *
- 成员关系操作符 in、not in

「等号 ==」，只有成员、成员位置都相同时才返回True。

列表拼接有两种方式，用「加号 +」和「乘号 *」，前者首尾拼接，后者复制拼接。

```
list1 = [123, 456]
list2 = [456, 123]
list3 = [123, 456]

print(list1 == list2)  # False
print(list1 == list3)  # True

list4 = list1 + list2  # extend()
print(list4)  # [123, 456, 456, 123]

list5 = list3 * 3
print(list5)  # [123, 456, 123, 456, 123, 456]

list3 *= 3
print(list3)  # [123, 456, 123, 456, 123, 456]

print(123 in list3)  # True
print(456 not in list3)  # False
```

前面三种方法（append, extend, insert）可对列表增加元素，它们没有返回值，是直接修改了原数据对象。 而将两个list相加，需要创建新的 list 对象，从而需要消耗额外的内存，特别是当 list 较大时，尽量不要使用 “+” 来添加list。

8. 列表的其它方法

list.count(obj) 统计某个元素在列表中出现的次数

```
list1 = [123, 456] * 3
print(list1)  # [123, 456, 123, 456, 123, 456]
num = list1.count(123)
print(num)  # 3
```

list.index(x[, start[, end]]) 从列表中找出某个值第一个匹配项的索引位置

```
list1 = [123, 456] * 5
print(list1.index(123))  # 0
print(list1.index(123, 1))  # 2
print(list1.index(123, 3, 7))  # 4
```

list.reverse() 反向列表中元素

```
x = [123, 456, 789]
x.reverse()
print(x)  # [789, 456, 123]
```

list.sort(key=None, reverse=False) 对原列表进行排序。

- key – 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。
- reverse – 排序规则，reverse = True 降序， reverse = False 升序（默认）。
- 该方法没有返回值，但是会对列表的对象进行排序。

```
x = [123, 456, 789, 213]
```

```
x.sort()
print(x)
# [123, 213, 456, 789]

x.sort(reverse=True)
print(x)
# [789, 456, 213, 123]


# 获取列表的第二个元素
def takeSecond(elem):
    return elem[1]


x = [(2, 2), (3, 4), (4, 1), (1, 3)]
x.sort(key=takeSecond)
print(x)
# [(4, 1), (2, 2), (1, 3), (3, 4)]

x.sort(key=lambda a: a[0])
print(x)
# [(1, 3), (2, 2), (3, 4), (4, 1)]
```

9.练习题

1、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

```
lst = [2, 5, 6, 7, 8, 9, 2, 9, 9]
lst.append(15)
lst.insert(5, 20)
lst.extend([2, 5, 6])
lst.pop(3)
lst.reverse()
lst.sort(reverse = True)
lst.sort(reverse = False)
print(lst)
```

2、修改列表

问题描述：

lst = [1, [4, 6], True]

请将列表里所有数字修改成原来的两倍

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

lst = [1, [4, 6], True]
fun(lst)
print(lst)
```

3、leetcode 852题 山脉数组的峰顶索引

如果一个数组k符合下面两个属性，则称之为山脉数组

数组的长度大于等于3

存在 iii， iii>0 且 i < len⁡(k) − 1 i<\operatorname{len}(k)-1 i < len(k) − 1， 使得 k [0] < k [1] < … < k [i − 1] < k [j] > k [i + 1] … > k [len⁡(k) − 1] \mathrm{k}[0]<\mathrm{k}[1]<\ldots<\mathrm{k}[\mathrm{i}-1]<\mathrm{k}[\mathrm{j}]>\mathrm{k}[\mathrm{i}+1] \ldots>\mathrm{k}[\operatorname{len}(\mathrm{k})-1] k[0] < k[1] < … < k[i − 1] < k[j] > k[i + 1] … > k[len(k) − 1]

这个 iii就是顶峰索引。

现在，给定一个山脉数组，求顶峰索引。

示例:

输入：[1, 3, 4, 5, 3]

输出：True

输入：[1, 2, 4, 6, 4, 5]

输出：False

```
class Solution:
    def peakIndexInMountainArray(self, A: List[int]) -> int:

    # your code here
    """
    :type A: List[int]
    :rtype: int
    """
    left = 0
    right = len(A)
    while left <= right:
        mid = (left + right) // 2
        mid_l = mid - 1
        mid_r = mid + 1
        if A[mid] > A[mid_l] and A[mid] > A[mid_r]:
            return mid
        elif A[mid] < A[mid_l]:
            right = mid
        elif A[mid] < A[mid_r]:
            left = mid
```