1 组合数据类型

1.1 组合数据类型概述

计算机不仅对单个变量表示的数据进行处理，更多情况，计算机需要对一组数据进行批量处理。一些例子包括：

- 给定一组单词{python, data, function, list, loop}，计算并输出每个单词的长度；
- 给定一个学院学生信息，统计一下男女生比例；
- 一次实验产生了很多组数据，对这些大量数据进行分析；

组合数据类型能够将多个同类型或不同类型的数据组织起来，通过单一的表示使数据操作更有序更容易。

根据数据之间的关系，组合数据类型可以分为三类：

序列类型、集合类型和映射类型。

- 序列类型是一个元素向量，元素之间存在先后关系，通过序号访问，元素之间不排他。
- 集合类型是一个元素集合，元素之间无序，相同元素在集合中唯一存在。
- 映射类型是“键-值”数据项的组合，每个元素是一个键值对，表示为(key, value)。

1.2 序列类型

序列类型是一维元素向量，元素之间存在先后关系，通过序号访问。

当需要访问序列中某特定值时，只需要通过下标标出即可。
序列类型支持成员关系操作符（in）、长度计算函数（len()）、分片（[]），元素本身也可以是序列类型。

Python语言中有很多数据类型都是序列类型，其中比较重要的是：str（字符串）、tuple（元组）和list（列表）。
- 元组是包含0个或多个数据项的不可变序列类型。元组生成后是固定的，其中任何数据项不能替换或删除。

- 列表则是一个可以修改数据项的序列类型，使用也最灵活

1.3 列表定义

列表类型的概念

列表（list）是包含0个或多个对象引用的有序序列，属于序列类型。与元组不同，列表的长度和内容都是可变的，可自由对列表中数据项进行增加、删除或替换。列表没有长度限制，元素类型可以不同，使用非常灵活。

由于列表属于序列类型，所以列表也支持成员关系操作符（in）、长度计算函数（len()）、分片（[]）。列表可以同时使用正向递增序号和反向递减序号，可以采用标准的比较操作符（<、<=、==、!=、>=、>）进行比较，列表的比较实际上是单个数据项的逐个比较。

列表用中括号（[]）表示，也可以通过list()函数将元组或字符串转化成列表。直接使用list()函数会返回一个空列表。

列表用一对 `[]` 生成，中间的元素用 `,` 隔开，其中的元素不需要是同一类型，同时列表的长度也不固定。

```
mlist = [1, 2.0, 'hello']
print(mlist)
```

```
[1, 2.0, 'hello']
```

空列表可以用 `[]` 或者 `list()` 生成：

```
empty_list = []
empty_list  # print(empty_list)
```

```
[]
```

```
empty_list = list()
empty_list
```

```
[]
```

```
#需要打印所有变量（而不只是最后一个）
```

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
ls = [425, "BIT", [10, "CS"], 425]
ls
ls[2][-1][0]
list((425, "BIT", [10, "CS"], 425)) # 将元组转换为list
list("中国是一个伟大的国家")  # 将字符串转换为list
list()  # 空list
```

```
[425, 'BIT', [10, 'CS'], 425]
```

```
'C'
```

```
[425, 'BIT', [10, 'CS'], 425]
```

```
['中', '国', '是', '一', '个', '伟', '大', '的', '国', '家']
```

```
[]
```

与整数和字符串不同，列表要处理一组数据，因此，列表必须通过显式的数据赋值才能生成，简单将一个列表赋值给另一个列表不会生成新的列表对象。

```
ls = [425, "BIT", 1024]  #用数据赋值产生列表ls
```

```
lt = ls       #lt是ls所对应数据的引用，lt并不包含真实数据？ 变量赋值怎么处理的?
ls[0] = 0
lt
id(lt)
id(ls)
```

```
[0, 'BIT', 1024]
```

```
1643058231368
```

```
1643058231368
```

1.4 列表操作

与字符串类似，列表也支持以下的操作：

1.4.1 长度

用 `len` 查看列表长度：

```
len(ls)
```

```
3
```

1.4.2 加法和乘法

列表加法，相当于将两个列表按顺序连接：

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

```
b = [3.2, 'hello']
a + b
```

```
[1, 2, 3, 3.2, 'hello']
```

列表与整数相乘，相当于将列表重复相加：

```
a * 3
```

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

```
vlist = list(range(5))
vlist
2 in vlist            #判断2是否在列表vlist中
```

```
[0, 1, 2, 3, 4]




True
```

1.4.3 索引和分片

列表和字符串一样可以通过索引和分片来查看它的元素。

索引：

```
a = [10, 11, 12, 13, 14]
a[0]
```

```
10
```

反向索引：

```
a[-1]
```

```
14
```

分片：

```
a[2:-1]
```

```
[12, 13]
```

与字符串不同的是，列表可以通过索引和分片来修改。

对于字符串，如果我们通过索引或者分片来修改，**Python**会报错：

```
s = "hello world"
# 把开头的 h 改成大写
s[0] = 'H'
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-10-844622ced67a> in <module>()
      1 s = "hello world"
      2 # 把开头的 h 改成大写
----> 3 s[0] = 'H'


TypeError: 'str' object does not support item assignment
```

而这种操作对于列表来说是可以的：

```
a = [10, 11, 12, 13, 14]
a[0] = 100
print(a)
```

```
[100, 11, 12, 13, 14]
```

这种赋值也适用于分片，例如，将列表的第2，3两个元素换掉：

```
a[1:3] = [1, 2]
a
```

```
[100, 1, 2, 13, 14]
```

```
a[1:3] = ['hello', 'python']
a
```

```
[100, 1, 2, 13, 14]
```

```
[100, 'hello', 'python', 13, 14]
```

事实上，对于连续的分片（即步长为 `1` ），**Python**采用的是整段替换的方法，两者的元素个数并不需要相同，例如，将 `[11,12]` 替换为 `[1,2,3,4]`：

```
a = [10, 11, 12, 13, 14]
a[1:3] = [1, 2, 3, 4]
a
```

```
[10, 1, 2, 3, 4, 13, 14]
```

这意味着，可以用这种方法来删除列表中一个连续的分片：

```
a = [10, 1, 2, 11, 12]
a[1:3] = []
a
```

```
[10, 11, 12]
```

对于不连续（间隔step不为1）的片段进行修改时，两者的元素数目必须一致：

```
a = [10, 11, 12, 13, 14]
a[::2] = [1, 2, 3]
a
```

```
[1, 11, 2, 13, 3]
```

否则会报错：

```
a[::2] = []
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-16-7b6c4e43a9fa> in <module>()
----> 1 a[::2] = []


ValueError: attempt to assign sequence of size 0 to extended slice of size 3
```

1.4.4 删除元素

Python提供了删除列表中元素的方法 'del'。

删除列表中的第一个元素：

```
a = [1002, 'a', 'b', 'c']
del a[0]
a
```

```
['a', 'b', 'c']
```

删除第2到最后一个元素：

```
a = [1002, 'a', 'b', 'c']
del a[1:]
```

```
a
```

```
[1002]
```

删除间隔的元素：

```
a = ['a', 1, 'b', 2, 'c']
del a[::2]
a
```

```
[1, 2]
```

1.4.5 测试从属关系

用 `in` 来看某个元素是否在某个序列（不仅仅是列表）中，用not in来判断是否不在某个序列中。

```
a = [10, 11, 12, 13, 14]
print(10 in a)
print(10 not in a)
```

```
True
False
```

也可以作用于字符串：

```
s = 'hello world'
print('he' in s)
print('world' not in s)
```

```
True
False
```

列表中可以包含各种对象，甚至可以包含列表：

```
a = [10, 'eleven', [12, 13]]
```

```
a[2]
```

```
[12, 13]
```

a[2]是列表，可以对它再进行索引：

```
a[2][1]
```

```
13
```

列表可以通过for…in语句对其元素进行遍历，基本语法结构如下：

```
for <任意变量名> in <列表名>:
语句块
```

```
for e in a:
    print(e, end=" ")
```

```
10 eleven [12, 13]
```

```
for e in a:
    print(e)
```

```
10
eleven
[12, 13]
```

1.5 列表方法

1.5.1 不改变列表的方法

列表中某个元素个数count

`l.count(ob)` 返回列表中元素 `ob` 出现的次数。

```
a = [11, 12, 13, 12, 11]
a.count(11)
```

```
2
```

列表中某个元素位置index

`l.index(ob)` 返回列表中元素 `ob` 第一次出现的索引位置，如果 `ob` 不在 `l` 中会报错。

```
a.index(12)
```

```
1
```

不存在的元素会报错：

```
a.index(1)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-26-ed16592c2786> in <module>()
----> 1 a.index(1)


ValueError: 1 is not in list
```

1.5.2 改变列表的方法

向列表添加单个元素

`l.append(ob)` 将元素 `ob` 添加到列表 `l` 的最后。

```
a = [10, 11, 12]
a.append(11)
```

```
print a
```

```
[10, 11, 12, 11]
```

append每次只添加一个元素，并不会因为这个元素是序列而将其展开：

```
a.append([11, 12])
print a
```

```
[10, 11, 12, 11, [11, 12]]
```

向列表添加序列

`l.extend(lst)` 将序列 `lst` 的元素依次添加到列表 `l` 的最后，作用相当于 `l += lst`。

```
a = [10, 11, 12, 11]
a.extend([1, 2])
print(a)
```

```
[10, 11, 12, 11, 1, 2]
```

插入元素

`l.insert(idx, ob)` 在索引 `idx` 处插入 `ob` ，之后的元素依次后移。

```
a = [10, 11, 12, 13, 11]
# 在索引 3 插入 'a'
a.insert(3, 'a')
print(a)
```

```
[10, 11, 12, 'a', 13, 11]
```

移除元素

`l.remove(ob)` 会将列表中第一个出现的 `ob` 删除，如果 `ob` 不在 `l` 中会报错。

```
a = [10, 11, 12, 13, 11]
# 移除了第一个 11
a.remove(11)
print(a)
# help(list.remove)
```

```
[10, 12, 13, 11]
Help on method_descriptor:

remove(...)
    L.remove(value) -> None -- remove first occurrence of value.
    Raises ValueError if the value is not present.
```

弹出元素

`l.pop(idx)` 会将索引 `idx` 处的元素删除，并返回这个元素。

```
a = [10, 11, 12, 13, 11]
a.pop(2)
```

```
12
```

1.6 排序

`l.sort()` 会将列表中的元素按照一定的规则排序：

```
a = [10, 1, 11, 13, 11, 2]
a.sort()
print(a)
```

```
[1, 2, 10, 11, 11, 13]
```

如果不想改变原来列表中的值，可以使用 `sorted` 函数：

```
a = [10, 1, 11, 13, 11, 2]
b = sorted(a)
print(a)
```

```
print(b)
```

```
[10, 1, 11, 13, 11, 2]
[1, 2, 10, 11, 11, 13]
```

1.7 列表反向

`l.reverse()` 会将列表中的元素从后向前排列。

```
a = [1, 2, 3, 4, 5, 6]
a.reverse()
print(a)
```

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

如果不想改变原来列表中的值，可以使用这样的方法：

```
a = [1, 2, 3, 4, 5, 6]
b = a[::-1]
print(a)
print(b)
```

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

如果不清楚用法，可以查看帮助：

```
a.sort?
```

1.8 列表浅拷贝（copy），深拷贝

python——赋值与深浅拷贝 https://www.cnblogs.com/Eva-J/p/5534037.html

```
# 注意文件命名方式：不能 与关键字copy等发生冲突
```

```python
# 浅拷贝，只拷贝第一层，2层以上 都是拷贝元素的地址
list_names = ["he", "li", ["liu", "li"], "fu", "chen"]
list_names2 = list_names.copy()
list_names[3] = "平"
print(list_names)
print(list_names2)

# 只是name，指向了list_names这个列表存储地址
name = list_names
print(name)
# 多维列表：，所以2层以后的元素，会跟着原来的列表改变
list_names[2][0] = "高"
print(list_names)
print(list_names2)
```

```
['he', 'li', ['liu', 'li'], '平', 'chen']
['he', 'li', ['liu', 'li'], 'fu', 'chen']
['he', 'li', ['liu', 'li'], '平', 'chen']
['he', 'li', ['高', 'li'], '平', 'chen']
['he', 'li', ['高', 'li'], 'fu', 'chen']
```

```python
import copy

# 深拷贝：拷贝的内容 不会随原列表list_names内容的更改而更改
list_names = ["he", "li", ["liu", "li"], "fu", "chen"]
list_names2 = copy.deepcopy(list_names)
list_names[3] = "平"
print(list_names)
print(list_names2)

# 多维列表
list_names[2][0] = "高"
print(list_names)
print(list_names2)
```

```
['he', 'li', ['liu', 'li'], '平', 'chen']
['he', 'li', ['liu', 'li'], 'fu', 'chen']
['he', 'li', ['高', 'li'], '平', 'chen']
['he', 'li', ['liu', 'li'], 'fu', 'chen']
```