目录

13 这么多的数据结构（一）：列表、元祖、字符串

更新时间：2019-09-11 10:02:15

> 人不可有傲气，但不可无傲骨。
>
> ——徐悲鸿

我们之前学习了字符串和列表，除此之外 Python 中还内置有元组、字典、集合等常用的数据类型，它们都被用于存放批量数据。

存放批量数据用列表不就行了吗，为什么还要这么多数据类型？这是因为在不同的场景下，对数据的读取和修改效率以及内存使用情况的要求是不一样的，为了满足不同的场景需求，需要以不同的组织形式来存放数据。上面所述的那些数据类型，本质上就是不同的数据组织形式，Python 直接为我们提供了它们的现成的实现，我们拿来即可使用，轻而易举地获取各种不同的存放、访问和修改数据的能力。

列表、元祖、字典、集合、字符串这些数据类型中所存放的一个个单独的数据，叫作项（Item）或元素（Element）。这些数据类型除了可以存放元素以外，还能通过调用对象方法来操作管理其中的元素。

我们来详细学习下这五种内置数据类型。

一、列表

列表是 Python 中非常常用的数据类型。之前的章节中我们学习过列表的一些基础知识，这个小节将会更深入地介绍列表的各种功能。

列表是用于存放若干元素的**有序序列**。列表使用方括号（ [] ）来表示，其中的元素写入方括号中，多个元素时用逗号分隔，如 `[1, 'go', [0.1, 0.2]]` 。它的元素可以是任意数据类型，甚至也可以是个列表。

列表被创建之后，我们可以对它做很多操作，包括添加元素，删除元素，修改元素，查找元素等。

创建列表

1. 创建空的列表：

```
列表 = []
```

```
>>> items = []
>>> items
[]
```

2. 创建包含元素的列表：

```
列表 = [元素1, 元素2, ..., 元素N]
```

```
>>> items = [1, 2, 3]
>>> items
[1, 2, 3]
```

列表元素的获取

1. 通过索引获取元素

```
元素 = 列表[索引]
```

```
>>> letters = [‘a’, ‘b’, ‘c’]
>>> letters[2]
’c’
```

2. 通过元素获取索引

这种方式和上面相反，首先在列表中寻找元素，然后返回元素对应的索引。

```
索引 = 列表.index(元素)
```

```
>>> letters = [‘a’, ‘b’, ‘c’]
>>> letters.index(‘c’)
2
```

要想查看元素是否存在于列表中，需要借助 Python 的关键字 in ，使用如下：

```
布尔值 = 元素 in 列表
```

```
>>> letters = [‘a’, ‘b’, ‘c’]
>>> ‘a’ in letters
True
>>> ‘z’ in letters
False
```

4. 统计元素在列表中的个数

统计元素在列表中的个数，或者说是元素在列表中出现的次数。

```
个数 = 列表.count(元素)
```

```
>>> numbers = [1, 2, 2, 3, 4, 5, 5, 7]
>>> numbers.count(5)
2
```

列表元素的添加

我们可以很灵活地向列表添加元素，如以追加的形式向列表末尾添加一个元素；以插入的形式向列表的任意位置插入元素；或者将一个列表中的所有元素批量的添加到另一个列表中。

1. 向列表末尾追加元素

```
列表.append(元素)
```

```
>>> letters = [‘a’, ‘b’]
>>> letters.append(‘c’)
>>> letters
[‘a’, ‘b’, ‘c’]
```

2. 向列表的任意位置插入元素

```
列表.insert(索引, 元素)
```

```
>>> letters = [‘a’, ‘b’]
```

```
[ ‘c’ , ‘a’ , ‘b’ ]

>>> letters.insert(2, ‘d’ )
>>> letters
[ ‘c’ , ‘a’ , ‘d’ , ‘b’ ]
```

3. 列表末尾追加另一个列表的所有元素

```
列表.extend(另一列表)
```

```
>>> letters = [ ‘a’ , ‘b’ ]
>>> letters.extend([ ‘c’ , ‘d’ , ‘e’ ])
>>> letters
[ ‘a’ , ‘b’ , ‘c’ , ‘d’ , ‘e’ ]
```

列表元素的删除

删除元素的方式同样很灵活。

1. 按索引删除元素

```
元素 = 列表.pop(索引)
```

pop(索引) 会将索引对应的元素从列表中删除，同时返回这个元素。

```
>>> letters = [ ‘a’ , ‘b’ , ‘c’ ]
>>> letters.pop(0)
’ a’
>>> letters
[ ‘b’ , ‘c’ ]
```

也可以不传递索引，这样的话默认删除并返回最后一个元素。

```
>>> letters = [ ‘a’ , ‘b’ , ‘c’ ]
>>> letters.pop()
’ c’
>>> letters
[ ‘a’ , ‘b’ ]
```

2. 按索引删除元素（ del 方法)

删除一个列表元素也可以使用 Python 中的 del 关键字，如下：

```
>>> letters = [ ‘a’ , ‘b’ , ‘c’ ]
>>> del  letters[0]
>>> letters
[ ‘b’ , ‘c’ ]
```

3. 直接删除元素

   直接删除元素时，Python 会先在列表中遍历该元素，然后将匹配到的第一个元素删除。

```
列表.remove(元素)
```

```
>>> letters = [ ‘a’ , ‘b’ , ‘c’ ]
>>> letters.remove( ‘b’ )
>>> letters
[ ‘a’ , ‘c’ ]
```

4. 清空所有元素

   清空所有元素即是把列表元素全部删除，最后仅为列表仅为 [] 。

```
列表.clear()
```

```
>>> letters = [ ‘a’ , ‘b’ , ‘c’ ]
>>> letters.clear()
>>> letters
[]
```

列表元素的修改

1. 通过赋值修改列表元素

```
列表[索引] = 新元素
```

```
>>> letters = [ ‘a’ , ‘b’ , ‘c’ ]
>>> letters[2] = 'd’
>>> letters
[ ‘a’ , ‘b’ , ‘d’ ]
```

2. 反转整个列表

```
列表.reverse()
```

```
>>> letters = [‘a’，‘b’，‘c’]
>>> letters.reverse()
>>> letters
[‘c’，‘b’，‘a’]
```

3. 列表元素排序

```
列表.sort()
```

```
>>> numbers = [2, 4, 5, 2, 1, 5, 7, 3]
>>> numbers.sort()
>>> numbers
[1, 2, 2, 3, 4, 5, 5, 7]
```

也可以通过指定 sort 方法的 reverse 参数来倒序排列。

```
列表.sort(reverse=True)
```

```
>>> numbers = [2, 4, 5, 2, 1, 5, 7, 3]
>>> numbers.sort(reverse=True)
>>> numbers
[7, 5, 5, 4, 3, 2, 2, 1]
```

二、元组

元组和列表非常相似，也是用于存放元素的有序序列。它用的圆括号（ () ）表示，元素写入圆括号中，多个元素时用逗号分隔，如 (1, 2, 3) 。

元组同样具有索引，索引使用方式与列表一致。其元素同样可以是任意类型。

看起来元组就是披着圆括号外衣的列表嘛！有什么区别？

元组创建完成后，便不能向其中添加元素，也不能修改和删除其中的任何一个元素。所以它与列表相比，只能查找元素，也就是说只具备读的功能，不具备写的功能。元组的这一特性叫作**不可变（性）**（Immutable），而列表是**可变的**（Mutable）。

创建元组

1. 创建空的元组：

```
元组 = ()
```

```
>>> items = ()
>>> items
()
```

2. 创建包含多个元素的元组：

```
元组 = (元素1, 元素2, ..., 元素N)
```

```
>>> items = (1，2，3)
>>> items
(1，2，3)
```

3. 创建只包含一个元素的元组

只包含一个元素的情况需要单独说明一下，因为它的形式与直觉不相符。

创建只包含一个元素的元组，需要在唯一的那个元素后面**加上逗号**，如：

```
元组 = (元素,)
```

```
>>> items = (1,)
>>> items
(1,)
```

这是因为，如果括号中只有一个元素，那么 Python 会将这个括号当作优先级符号进行处理（像数学中的那样），而不是当作元组。可以试一下：

```
>>> items = (1)
>>> items
1
>>> type(items)
<class ‘int’>
```

元组元素的获取

1. 通过索引获取元素

```
元素 = 元组[索引]
```

```
>>> letters[2]
’c’
```

2. 通过元素获取索引

```
索引 = 元组.index(元素)
```

```
>>> letters = (‘a’, ‘b’, ‘c’)
>>> letters.index(‘c’)
2
```

3. 查看元素是否存在于元组中

```
布尔值 = 元素 in 元组
```

```
>>> letters = (‘a’, ‘b’, ‘c’)
>>> ‘a’ in letters
True
>>> ‘z’ in letters
False
```

4. 统计元素在元组中出现的个数

```
个数 = 元组.count(元素)
```

```
>>> numbers = (1, 2, 2, 3, 4, 5, 5, 7)
>>> numbers.count(5)
2
```

元组和列表的差别

我们可以看到，元组所具有的操作在使用方式上与和列表非常相似，甚至在一定程度上可以将元组看作是列表的精简版，但它们之间也有明显的差别。

- 元组是不可变的（Immutable），列表是可变的（Mutable），元组在被创建之后，就不能添加、删除和修改元素，而列表可以
- 一般情况下元组的性能在略高于列表

我们在什么时候用列表，什么时候用元组？

列表；当我们希望所有元素在创建之后便不再改变，可使用元组。

三、字符串

字符串也是 Python 中非常常用的内置数据类型。我们之前学习过字符串的一些内容，现在来深入的了解下。

字符串是 Python 中用来存放字符序列的数据类型，其中的元素只能是字符。字符串使用单引号或双引号来表示，如 'pick' ， "cherry" ，通常我们首先使用单引号。

字符串是有序序列，可以使用索引来获取其中某个位置的元素。它是不可变的，被创建之后其中的元素（也就是字符）不能被修改和删除。

创建字符串

1. 创建空字符串（即不包含字符的字符串）：

```
字符串 = ''
```

```
>>> string = ''
>>> string
' '
```

2. 创建包含元素的字符串：

```
字符串 = '若干字符'
```

```
>>> string = 'happy'
>>> string
' happy'
```

字符的获取

1. 通过索引获取字符

```
字符 = 字符串[索引]
```

```
>>> string = 'happy'
>>> string[2]
' p'
```

2. 通过子串获取索引

的是子串的第一个字符的索引。

```
索引 = 字符串.index(字符)
```

```
>>> string = ‘happy’
>>> string.index(‘p’)
2
```

```
>>> string = ‘happy’
>>> string.index(‘app’)
1
```

当字符或子串不存在时， index 方法将抛出 ValueError 错误。

也可采用字符串的 find 方法来查找子串，使用方式与 index 一致，不同点在于 find 方法未找到子串时返回数字 -1 ，而不抛异常。

```
>>> string = ‘happy’
>>> string.find(‘app’)
1
>>> string.find(‘z’)
-1
```

3. 查看字符是否存在于字符串中

查看字符是否存在于字符串中，需要借助 Python 的关键字 in ，如下：

```
布尔值 = 字符 in 字符串
```

```
>>> string = ‘happy’
>>> ‘a’ in string
True
>>> ‘z’ in string
False
```

4. 统计字符在字符串中的个数

```
个数 = 字符串.count(字符)
```

```
>>> string.count(‘p’)
2
```

字符串的处理

字符串自带的方法非常多，除了上面介绍的几个之外还有四十多个，这是因为字符处理是编程时的一项高频工作。Python 将这些字符处理相关的功能以方法的形式集成在字符串里。

这里列举几个常见的方法：

- startswith：判断字符串是否以某个子串开头，返回布尔值

```
>>> string = ‘happy’
>>> string.startswith(‘ha’)
True
```

- endswith：判断字符串是否以某个子串结尾，返回布尔值

```
>>> string = ‘happy’
>>> string.endswith(‘y’)
True
```

- replace：将字符串的子串用一个另一个字符串替换，返回一个新的字符串

```
>>> string = ‘happy’
>>> string.replace(‘y’, ‘iness’)
’happiness’
```

- strip：去除字符串前后的空白符号，如空格、换行符、制表符，返回一个新的字符串

```
>>> string = ’ \t happy \n’
>>> string.strip()
’happy’
```

- split：将字符串用某个子串分隔开，分隔后的各个部分放入列表中，并返回这个列表

```
>>> string = ‘I am happy’
>>> string.split(’ ')
[‘I’, ‘am’, ‘happy’]
```

符串

```
>>> words = [‘I’，‘am’，‘happy’]
>>> ’ '.join(words)
’I am happy’
```

- upper：将字符串转化为大写字母形式，返回一个新的字符串

```
>>> string = ‘happy’
>>> string.upper()
’HAPPY’
```

- lower：将字符串转化为小写字母形式，返回一个新的字符串

```
>>> string = ‘HAPPY’
>>> string.lower()
’happy’
```

注意上面的这些字符处理功能，对字符串作处理后都是返回一个新的字符串，而不会直接修改原有的字符串。为什么呢？字符串不可变呀！

字符转义

我们在创建字符串时，有一些字符是没法直接在引号中表示的，如单引号或双引号，因为这和表示字符串本身的符号冲突了，如：

```
>>> string = 'I’m happy’
File “ ”, line 1
    string = ‘I’m happy’
                ^
SyntaxError: invalid syntax
```

抛出 SyntaxError 语法错误异常，因为 Python 将 string = 'I' 看作字符串的赋值，而后面的 m happy' 就无法解析了，因为不符合任何语法。

这时就需要使用**字符转义**了，我们在这类无法直接在字符串中表示的字符前加上 \ 符号，形如 \' ，这样 Python 在解析时就能理解这是嵌入在字符串中的单引号 ' ：

```
>>> string
"I’m happy"
```

像 \' 这样在前面加了反斜杠（\），为了能在字符串中正常表示的字符，叫作**转义字符**。而这个转化的行为叫作**字符转义**。

与单引号的用法相同，双引号用 \" 来转义。

字符串中的 \ 用来做字符转义了，那怎么在字符串中表示斜杆 \ 这个字符呢？使用 \\ ，将斜杆 \ 转义一下。除此之外，有一些空白符号也需要用转义字符来表示，因为我们没办法直接在字符串中表示它们，如常见的换行（ \n ），制表符（ \t ），回车（ \r ）。

常用的转义字符	含义
\'	单引号
\"	双引号
\\	反斜杠
\n	换行符
\t	制表符（Tab）
\r	回车

举个例子，如果在字符串使用了 \n ，那么在用 print() 输出字符串的时候，这个字符串会被换行输出。如：

```
>>> print(‘第一行\n第二行’)
第一行
第二行
```

使用 \n 换行符使得我们能够在一行的字符串来表示多行的内容。

```
说明：转义字符虽然在书写时使用了两个字符，但是在程序中它只是一个字符。可以自己来试验下：
>>> len(’\n’)
1
>>> len(’\’’)
1
```

如果我们就想在字符串中表示 \n 这两个字符，而不是让它表示换行，该怎么办？有两种方式：

1. 使用 \\n ，将 \n 前面的反斜杠转义

2. 使用原始字符串

原始字符串

原始字符串就是在字符串的起始引号前加上一个 r 或 R 字母，这样字符串中的内容将不会被转义，将按照原样输出。使用方法：

```
r'字符串内容'
```

```
>> print(r’第一行\n第二行’)
第一行\n第二行
```

多行字符串

我们之前所使用的字符串都被书写成一行，要想让字符串可以跨行书写，写成多行的形式，有两种方法：

1. 字符串的每行末尾使用 \ 续行

   以多行的形式书写字符串，每行的末尾使用 \ 续行。需要注意输出内容为一行。

```
string = '第一行\
第二行\
第三行'
```

```
>>> string = '第一行\
… 第二行\
… 第三行’
>>> print(string)
‘第一行第二行第三行’
```

可以看到这种方式可以让字符串以多行的方式来书写，但是输出内容还是被当作一行。如果想要输出内容为多行，需要在字符串中显式地使用 \n 进行换行。

```
>>> string = '第一行\n\
… 第二行\n\
… 第三行’
>>> print(string)
第一行
第二行
第三行
```

在 Python 中字符串也可以使用三个单引号或三个双引号来表示字符串，这样字符串中的内容就可以多行书写，**并且被多行输出**。

```
string = '''第一行
第二行
第三行'''
```

```
>>> string = ‘’‘第一行
… 第二行
… 第三行’’’
>>> print(string)
第一行
第二行
第三行
```

使用三引号的方式，字符串可被多行书写，且被多行输出，其中不需要显式地指明 \n 换行。

列表、元组、字符串的通用操作

我们把列表、元组、字符串统称为**序列**。

1. 使用 len() 函数获取序列长度

```
>>> letters = (‘a’, ‘b’)
>>> len(letters)
2
```

```
>>> letters = ‘abcd’
>>> len(letters)
4
```

2. 获取序列中的一个子序列

获取序列中的子序列可以使用**切片**，以 [起始索引:结束索引] 表示。切片其实代表一个索引区间，这个区间是一个**左开右闭区间**，该区间内的所有元素作为子序列被返回。如：

```
>>> numbers = (1, 2, 3, 4, 5)
>>> numbers[0:2]
(1, 2)
```

目录


```
>>> letters = ‘abcd’
>>> letters[1:5]
’ bcd’
```

3. 使用 + 符号来拼接两个序列

```
>>> letters_1 = (‘a’, ‘b’)
>>> letters_2 = (‘c’, ‘d’, ‘e’)
>>> letters_1 + letters_2
(‘a’, ‘b’, ‘c’, ‘d’, ‘e’)
```

```
>>> letters_1 = ‘ab’
>>> letters_2 = ‘cde’
>>> letters_1 + letters_2
’ abcde’
```

4. 使用 * 符号来重复序列中的元素

```
>>> letters = (‘a’, ‘b’)
>>> letters * 3
(‘a’, ‘b’, ‘a’, ‘b’, ‘a’, ‘b’)
```

```
>>> letters = ‘abcd’
>>> letters * 2
’ abcdabcd’
```

注意上面的操作结果都是返回一个新的序列，不会对修改序列的内部元素。

总结

列表、元组、字符串都是有序序列，都可以使用索引。

列表和元组中可以存放任意数据类型的元素，而字符串中只能存放字符。

列表是可变的，而元组和字符串是不可变的。

目录

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

Leo梁

列表.sort(reverse=True)，这个如何理解？不懂

0 回复 2019-11-29

千学不如一看，千看不如一练