

第三讲 - Python 序列

张建章

阿里巴巴商学院

杭州师范大学

2022-09



- 1 Python 常见的内置数据类型
- 2 列表
- 3 列表的基本操作
- 4 常用的操作列表的内置函数
- 5 常用的列表方法
- 6 内置函数与列表方法的区别
- 7 多维列表
- 8 元组
- 9 元组封装与序列拆封
- 10 元组与列表的比较

序列：列表 (可变)，元组 (不可变)，字符串 (不可变)，序列中的内容是有顺序的，其正向和反向查找索引如下图所示：

G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

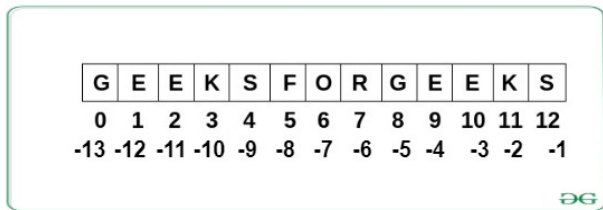


图 1: 序列的正向索引和反向索引

集合：无重复元素；

字典：存放具有映射关系的数据；

列表 (list): 一对 `[]` 中包含由 `,` 分割的一系列元素，一个列表中
可以包含任意多个 (≥ 0) 任意类型的数据，每一个数据称为一个元素。

创建列表的两种常用方式:

① 最简单方法是将列表元素放在一对方括号 `[]` 内，各元素之间以
逗号分隔，并用 `=` 将一个列表赋值给某个变量。如，`list1 = [1,3,5]`;

② 使用内置的 `list` 函数创建列表，如 `list2 = list('Python')`，
`list2` 的内容就是列表 `['P', 'y', 't', 'h', 'o', 'n']`。

注意: Python 允许同一个列表中各元素的数据类型不相同，可以是
整数、字符串等基本类型，也可以是列表、集合及其他类型的对象，
如 `list3 = ['python', True, 99, ['good', 'boy']]`。

列表访问

可以通过索引访问列表 (长度为 n) 中的元素, 索引分为正向索引 (范围为: $0 \sim n - 1$) 和反向索引 (范围为: $-1 \sim -n$), 如下图所示:

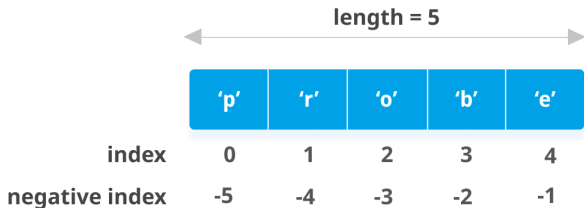


图 2: 列表的正向索引和反向索引

`list[index]` 可以像变量一样使用, 进行读取 (读取列表中索引 *index* 对应的元素值) 或写入 (更改列表中索引 *index* 对应的元素值), 所以它也被称为下标变量。

成员判断和切片

使用 `in` 和 `not in` 运算符可以判断一个元素是否在列表中。

列表切片：使用语法 `list[start:end]` 返回列表 `list` 的一个片段，其结果是 `list` 中索引 `start` 到 `end-1` 对应的元素所构成的一个新列表。

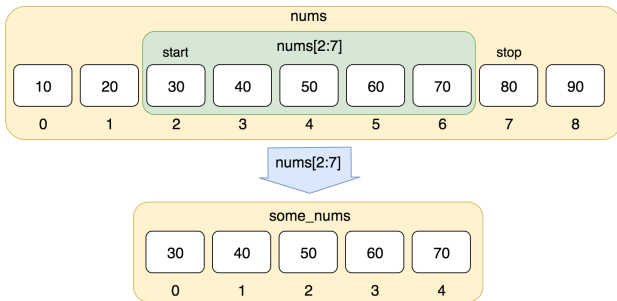


图 3: 列表的切片操作

3. 列表的基本操作

在切片操作中，起始下标 **start** 和结束下标 **end** 可以省略：① 如省略 **start**，则起始下标默认为 0，即从列表第一个元素开始截取；② 如省略 **end**，则结束下标默认为列表长度 n ，即截取到列表最后一个元素。下图中第三个参数 **-2** 表示切片步长，步长为正，要求 **start** < **end**，步长为负，要求 **start** > **end**。**list[::-1]** 可以实现列表快速翻转。

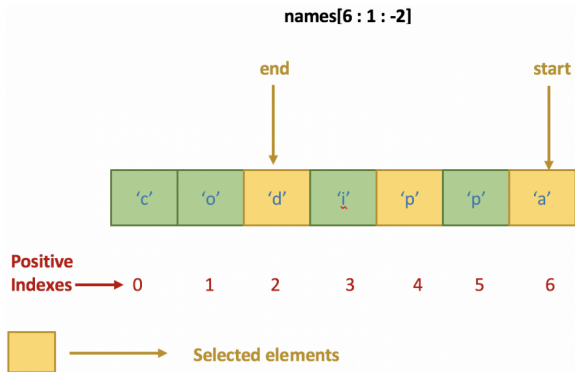


图 4: 带步长的列表的切片操作

修改列表元素值

可以通过**列表元素访问**和**切片操作**修改列表单个元素的值和同时修改多个元素的值，如下：

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

# 通过索引访问列表元素，修改单个元素值
x[2] = 100 # [1,2,100,4,5,6]

# 通过列表切片，同时修改多个元素值
x[-1:-3] = [200, 300] # [1, 2, 3, 4, 300, 200]
```

因为列表切片的结果是一个**列表**，所以上面切片赋值操作中等号右侧的内容一定是一个列表。

通过切片扩展、删减和复制列表

```
x = [1, 'a', 'b', 'c', 5, 6]
```

```
# 扩展列表
```

```
x[1:1] = [7, 8, 9] # [1, 7, 8, 9, 'a', 'b', 'c', 5, 6]
```

```
# 删减列表
```

```
x[1:4] = [] # [1, 'a', 'b', 'c', 5, 6]
```

```
# 复制列表-深复制
```

```
y = x[:] # [1, 'a', 'b', 'c', 5, 6]
```

```
# 赋值列表-浅复制 (不推荐使用)
```

```
z = x # [1, 'a', 'b', 'c', 5, 6]
```

```
# 判断三个变量的值以及指向的内存空间是否相同
```

```
x == y == z # True
```

```
x is y # False
```

```
x is z # True
```

列表的拼接和复制

Python 中，使用 `+` 号拼接两个列表，并返回一个新列表，使用 `*` 复制一个列表若干次，返回一个新列表，如下：

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

```
y = x*2 # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
x[0] = 0 # 复制返回的是新列表，故y中的元素保持不变
```

```
z = x*2 + y # [0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
# 复制和拼接都是返回新列表，故z中的元素保持不变
```

```
x[0] = 111
```

```
y[0] = 222
```

列表的比较

使用关系运算符 `<`, `>`, `==`, `<=`, `>=`, `!=` 对列表进行比较。

两个列表的比较规则如下：

- ① 比较两个列表的第一个元素，如果两个元素相同，则继续比较下面两个元素；
- ② 如果两个元素不同，则返回两个元素的比较结果；
- ③ 一直重复这个过程直到有不同的元素或比较完所有的元素为止。

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

```
y = [1, 2, 3, 100]
```

```
x < y # True
```

```
xstr = ['阿', 'z']
```

```
ystr = ['a', '里']
```

```
xstr > ystr # True
```

列表推导式

列表推导式是一个生成新列表的简洁方法。一个列表推导式由方括号括起来，方括号内包含跟着一个 **for** 子句的表达式，之后可以接 **0** 到多个 **for** 或 **if** 子句。列表推导式可以产生一个由表达式求值结果作为元素的新列表。

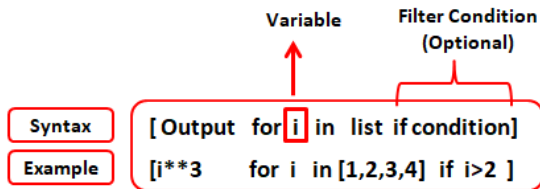


图 5: 列表推导式

4. 常用的操作列表的内置函数

- `all(iterable)`，全部元素为 `True`，返回 `True`；
- `any(iterable)`，有一个元素为 `True`，返回 `True`；
- `list(s)`，将可迭代对象 `s` 变为列表；
- `len(s)`，计算列表的长度；
- `max(iterable)`，返回列表中最大的元素；
- `min(iterable)`，返回列表中最小的元素；
- `sorted(iterable[, cmp[, key[, reverse]]])`，对列表进行排序；
- `sum(iterable[, start])`，对列表进行求和；
- `reversed(iterable)`，翻转列表。

上述内置函数不仅可以用于操作列表，还可用于操作其他可迭代对象，使用 `help` 函数可查看其详细用法，如 `help(all)`。

- `list.append(x)`，在列表末尾增加一个元素 `x`；
- `list.extend(L)`，在列表末尾再拼接一个列表 `L`；
- `list.insert(i, x)`，在索引为 `i` 的位置插入一个元素 `x`；
- `list.remove(x)`，从列表中移除元素 `x`；
- `list.pop(i)`，删除索引 `i` 对应的元素；
- `list.index(x)`，返回元素 `x` 对应的索引值；
- `list.count(x)`，计数元素 `x` 在列表中出现的次数；
- `list.sort(key=None, reverse=False)`，对列表进行排序；
- `list.reverse()`，将列表翻转。

上述常用的列表方法只能用于操作列表，使用 `help` 函数可查看其详细用法，如 `help(list.append)`，`help(list)` 查看列表的全部方法。

- **基本用法:** 操作列表的内置函数的用法为 `function_name(list)`, 列表方法的用法为 `list.method_name(parameters)`;
- **结果:** 操作列表的内置函数不改变列表的内容, 如 `reversed(list)` 其返回结果是一个新的列表, 而列表 `list` 不发生变化, 列表方法会直接改变表列表的内容, 如 `list.reverse()` 执行后, 列表 `list` 中的元素将会翻转, 并不会返回一个新列表。

注意: 当元素 `x` 在列表 `list` 中多次出现时, `list.remove(x)` 只能移除第一个 `x`(索引最小的那个 `x`), `list.index(x)` 只能返回第一个 `x` 的索引值 (`x` 对应的最小索引值)。

7. 多维列表

多维列表：列表中嵌套列表，比较常用的是二维列表 (矩阵) 和三维列表 (张量)。数值 (矩阵、张量) 运算中通常使用科学计算包 `numpy` 中的 `array` 数据类型来表示多维数值列表。

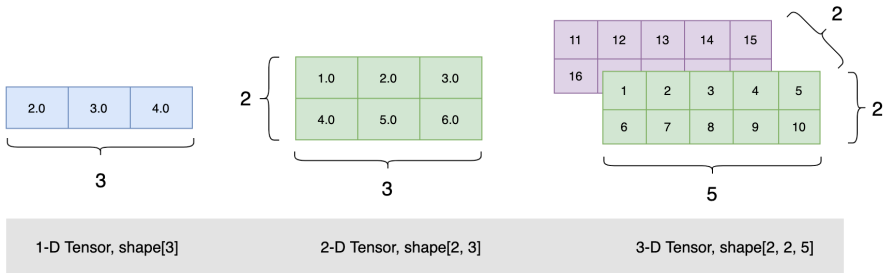


图 6: 多维数值列表 (张量)

访问多维列表中的元素：① 使用多重 `for` 循环遍历每个元素，① 使用多级索引访问某个元素，`tensor3[1][1][3]` 可访问上图最右侧三维列表中的元素 19。

7. 多维列表

- 多维列表本质也是一个列表，可通过 `type` 函数可查看其数据类型，其包含的元素也是列表；
- 前述的操作列表的内置函数和列表的方法也适用于多维列表；
- 多维列表可以表示数学上的张量，也可以表示其他数据，如二维表(数据库表，简单 Excel 表格)，其每一个维度值是灵活的，如下例：

```
# 下面的多维列表表示了下图中矩阵的左上角  
left_top = [[1, 2, 3], [4, 5], [7]]
```

1	2	3
4	5	6
7	8	9

元组 (tuple): 一对 `()` 中包含由 `,` 分割的一系列元素，一个元组中可以包含任意多个 (≥ 0) 任意类型的数据，每一个数据称为一个元素，元组中的元素不可变。

创建元组的两种常用方式：

① 最简单方法是将元组元素放在一对圆括号 `()` 内，各元素之间以逗号分隔，并用 `=` 将一个元组赋值给某个变量。如，`tuple1 = (1,3,5)`；

② 使用内置的 `tuple` 函数创建列表，
如 `tuple2 = tuple('Python')`，`tuple2` 的内容就是列表 `('P', 'y', 't', 'h', 'o', 'n')`。

注意：Python 允许同一个元组中各元素的数据类型不相同，可以是整数、字符串等基本类型，也可以是列表、集合及其他类型的对象，如 `tuple3 = ('python', True, 99, ['good', 'boy'])`。

正确理解元组中的数据元素不可变

① 元组一旦创建，其长度不可变，不能删减、增加元素，不能重新赋值元素；

② **不能重新赋值元素**：元组中的元素是不可变数据类型时，该元素不可变 (重新赋值)，如 2-Introduction.pdf 中介绍过的基本数据类型 (整数、浮点数、字符串、布尔值、空值)；如果元素是可变数据类型，则可以改变该元素的值，但是不可以改变该元素的数据类型 (重新赋值)，具体看下面示例：

```
tuple4 = (1, 2, [1, 2, 3])
```

```
tuple4[2].append(4) # (1, 2, [1, 2, 3, 4])
```

```
# Error: 'tuple' object does not support item assignment
```

```
tuple4[2] = True
```

```
# Error: 'tuple' object does not support item assignment
```

```
tuple4[0] = 'good'
```

元组相关操作

可以使用 `tuple` 函数将列表、字符串等元素转换为元组；元组也是序列，一些用于列表的基本操作也可以用在元组上，如索引访问、成员判断、切片等。

```
# 使用tuple函数将列表转换为一个元组
list_1 = [1, 'Lisa', True, 6.6]
tuple_2 = tuple(list_1)

# 使用tuple函数将字符串转换为一个元组
string_1 = '阿里巴巴商学院'
tuple_3 = tuple(string_1)

# 判断元素是否在元组中
'巴' in tuple_3
# 使用索引访问元组中的元素
tuple_3[0]
# 对元组进行切片
tuple_3[1:3]
```

元组封装：将多个值自动封装到一个元组中；

序列拆封：将一个封装起来的序列自动拆分为若干个基本数据。

为多个变量同时赋值：结合了元组封装和序列拆封操作。

`a, b, c, d = 4, 'Tony', True, 6.6` 等价
于 `my_tuple = 4, 'Tony', True, 6.6` 和 `a, b, c, d = my_tuple`

```
# 元组封装
```

```
tuple_4 = 4, 'Tony', True, 6.6 # (4, 'Tony', True, 6.6)
```

```
# 元组拆封
```

```
a, b, c, d = tuple_4
```

```
# 列表拆封
```

```
list_2 = [8, 'Dan', False, 8.8]
```

```
a, b, c, d = list_2
```

相同点：均为序列数据类型，除元素修改操作外，其他操作，如索引访问、计数、切片等，两者用法相同；

不同点：元组属于不可变序列，一旦创建，便不允许进行元素的增加、删除和重新赋值，因此，元组没有 `append`、`extend`、`insert` 方法；列表是可变序列，可以对其中元素进行增删改操作；

应用场景不同：元组的访问和处理速度比列表更快。因此，如果所定义的序列内容不会进行修改，最好使用元组而不是列表。另外，使用元组也可以使元素在实现上无法被修改，从而使代码更加安全。

THE END