

第二讲 - 列表

张建章

阿里巴巴商学院

杭州师范大学

2024-09



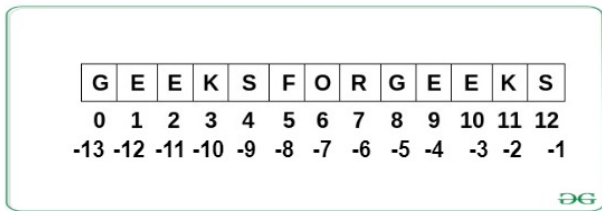
1 序列概述

2 创建列表

3 列表的基本操作

1. 序列概述

在 Python 中，**序列 (sequence)** 数据类型是一类用于存储有序数据的容器，能够通过整数索引访问其元素。常见的序列类型包括字符串 (string)、列表 (list)、元组 (tuple)。这些序列类型有一些共同特征：它们的元素是有序的，可以通过索引进行访问。



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

Python 的序列类型提供了丰富的操作，如切片（提取子序列）、连接（使用 `+` 运算符连接多个序列）、重复（使用 `*` 运算符重复序列）和成员资格测试（使用 `in` 和 `not in` 测试元素是否在序列中）。

定义列表

列表（list）是一种**有序且可变**的数据结构，能够存储多个元素，并允许对这些元素进行动态操作，如添加、删除或修改。

Python 列表使用**方括号**定义，并通过逗号分隔元素。它可以包含各种类型的数据，例如数字、字符串、甚至其他列表。这使得列表能够存储复杂的数据结构，如订单记录、财务数据或客户反馈等。

```
# 示例：存储销售订单数据
orders = [" 订单 A", " 订单 B", " 订单 C"]
orders.append(" 订单 D") # 添加新订单
print(orders) # 输出 ['订单 A', '订单 B', '订单 C', '订单 D']
```

list 函数

`list()` 是 Python 中的内置函数，用于将可迭代对象（如字符串、元组、集合等）转换为列表。该函数可以创建一个新的空列表，或者通过传递一个可迭代对象来初始化列表。

```
# 创建空列表
empty_list = list()
print(empty_list) # 输出: []

# 从字符串创建列表
string = "hello"
char_list = list(string)
print(char_list) # 输出: ['h', 'e', 'l', 'l', 'o']

# 从元组创建列表
tuple_data = (1, 2, 3)
list_from_tuple = list(tuple_data)
print(list_from_tuple) # 输出: [1, 2, 3]
```

列表的多维结构

Python 列表还支持多维结构，即列表的元素可以是另一个列表，使得它在表示复杂的商业数据时非常有用。例如，在一个订单系统中，每个订单可能包含多个产品，每个产品又有自己的属性（如名称、价格、数量）。使用嵌套列表可以很好地表示这种结构：

```
# 示例：存储订单中包含的产品信息
order_details = [
    [" 产品 A", 100, 2], # 产品名称、单价、数量
    [" 产品 B", 200, 1],
    [" 产品 C", 150, 5]
]
print(order_details[0]) # 输出 ['产品 A', 100, 2]
```

在这个示例中，每个子列表代表一个产品的详细信息，而整个列表表示一个订单的产品清单。这种嵌套结构非常适合用于管理诸如采购订单、库存列表等复杂的数据。

索引操作

1. 使用正索引访问元素

Python 列表中的元素可以通过方括号 `[]` 内的索引值进行访问。例如，有一个包含水果的列表：

```
fruits = ['apple', 'banana', 'mango', 'orange']  
# 访问第二个元素  
print(fruits[1]) # 输出: 'banana'  
# 访问最后一个元素  
print(fruits[-1]) # 输出: 'orange'
```

`fruits[1]` 访问的是列表中的第二个元素（索引从 0 开始）。

2. 使用负索引访问元素

负索引用于从列表的末尾开始计数，`-1` 表示最后一个元素，`-2` 表示倒数第二个元素，以此类推。这种方式在不确定列表长度时特别有用，方便访问列表末尾的元素。

3. 修改列表中的元素

列表是可变的数据结构，因此可以通过索引直接修改其中的元素。
例如，修改上例中第二个元素为 `'strawberry'`：

```
fruits[1] = 'strawberry'  
print(fruits)  # 输出: ['apple', 'strawberry', 'mango', 'orange']
```

同样地，也可以使用负索引来修改末尾的元素。

切片操作

在 Python 中，列表切片是从一个列表中提取部分元素的常用操作。其基本语法是：

```
list[start:stop:step]
```

其中，`start` 表示切片的起始索引（包含该索引），`stop` 表示结束索引（不包含该索引），而 `step` 表示每次跳过的步长。

1. 基本切片操作

最常见的切片是使用 `start` 和 `stop` 两个参数，从指定的起始位置到结束位置提取子列表。

```
fibonacci_sequence = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
sliced_list = fibonacci_sequence[2:5]
print(sliced_list) # 输出: [1, 2, 3]
```

这里，从索引 2 开始提取，到索引 5 结束（不包括索引 5）。

2. 省略 start 或 stop 参数

如果省略 start 参数，默认从列表的第一个元素开始；如果省略 stop，则切片会一直到列表的末尾。

```
sliced_list = fibonacci_sequence[:4]
print(sliced_list) # 输出: [0, 1, 1, 2]
```

此时提取的是从列表开头到索引 4 之前的所有元素。

3. 使用 step 参数

step 参数允许我们指定切片时的步长，从而可以跳过一些元素。例如，以下代码每隔一个元素提取一次：

```
sliced_list = fibonacci_sequence[1:8:2]
print(sliced_list) # 输出: [1, 2, 5, 13]
```

step 为 2，因此在指定范围内每隔一个元素提取一次。

4. 负索引和反转

Python 允许使用负索引来从列表末尾进行切片。此外，可以通过负 `step` 来反转列表：

```
reversed_list = fibonacci_sequence[::-1]
print(reversed_list)  # 输出: [34, 21, 13, 8, 5, 3, 2, 1, 1, 0]
```

这种方式可以轻松实现列表的反转。

5. 使用切片插入元素

切片可以用来在列表中插入元素，而不替换现有元素。通过设置起始索引和结束索引相同的方式，可以在指定位置插入新元素。

```
numbers = [1, 2, 3, 6, 7]
numbers[3:3] = [4, 5]  # 在索引 3 处插入元素
print(numbers)  # 输出: [1, 2, 3, 4, 5, 6, 7]
```

通过 `[3:3]` 在索引 3 的位置插入了元素 `[4, 5]`。

6. 使用切片替换元素

切片也可以用来替换列表中的一部分元素，只需将指定范围内的元素替换为新的值。

```
colors = ['red', 'orange', 'yellow', 'green', 'blue']  
colors[1:3] = ['purple', 'pink'] # 替换索引 1 到 2 的元素  
print(colors) # 输出: ['red', 'purple', 'pink', 'green', 'blue']
```

在此操作中，列表中索引 1 和 2 的元素
('orange' 和 'yellow') 被新值 'purple' 和 'pink' 替换。

7. 使用切片删除元素

将某一范围内的元素替换为空列表，可以删除列表中的一部分元素。

```
colors = ['red', 'orange', 'yellow', 'green', 'blue']  
colors[1:3] = [] # 删除索引 1 到 2 的元素  
print(colors) # 输出: ['red', 'green', 'blue']
```

列表拼接

在 Python 中，使用加法运算符 `+` 将两个列表拼接在一起是一种简单、直接的方式。通过这个操作，两个列表会合并为一个新的列表，而原始列表不会被修改。

```
list1 = ['a', 'b', 'c']
list2 = ['d', 'e', 'f']

combined_list = list1 + list2
print(combined_list) # 输出: ['a', 'b', 'c', 'd', 'e', 'f']
```

注意：使用 `+` 运算符时，会生成一个新的列表对象。因此，对于非常大的列表，可能会消耗额外的内存。对于需要频繁拼接的大型数据集，可以考虑使用其他方法，如 `extend()` 方法，它直接修改现有列表，避免创建新的列表。

列表乘法

列表乘法是一种通过重复列表中的元素来生成新列表的操作。它使用星号运算符（`*`）来实现，基本语法如下：

```
my_list = [1, 2, 3]
new_list = my_list * 3
print(new_list)  # 输出: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

当列表包含可变对象（如嵌套列表）时，乘法操作会重复引用而不是创建独立的副本。这意味着修改其中一个元素会影响所有引用的对象。

```
grid = [[0] * 3] * 4
print(grid)  # 输出: [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
# 修改第一个子列表
grid[0][0] = 1
print(grid)  # 所有子列表的第一个元素都被修改了
# 输出: [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]]
```

修改列表元素

1. 通过索引直接赋值：最常见的修改方法是使用列表的索引来替换特定位置的元素。

```
my_list = [1, 2, 3, 4, 5]
my_list[1] = 10
print(my_list)  # 输出: [1, 10, 3, 4, 5]
```

通过 `my_list[1]` 访问列表中的第二个元素，并将其修改为 10。这种方法简单直接，适用于已知索引的位置。

2. 使用切片修改多个元素：

```
my_list = [1, 2, 3, 4, 5]
my_list[3:] = [6, 7]
print(my_list)  # 输出: [1, 2, 3, 6, 7]
```

通过切片，可以直接修改指定范围内的多个元素。

删除列表元素

删除列表中的元素可以通过多种方法实现，主要包括以下几种常用方式：

1. 使用 `remove()` 方法

`remove()` 方法根据元素的值来删除列表中的第一个匹配项。如果列表中有重复的元素，`remove()` 只会删除第一个出现的值。如果指定的值不在列表中，则会抛出 `ValueError`。

```
# 创建一个包含多个元素的列表
fruits = ['apple', 'banana', 'cherry', 'banana']

# 删除第一个 'banana'
fruits.remove('banana')

# 输出更新后的列表
print(fruits) # 输出: ['apple', 'cherry', 'banana']
```


2. 使用 `pop()` 方法

`pop()` 方法通过索引删除元素。默认情况下，它删除并返回列表的最后一个元素。如果提供了索引参数，则删除并返回对应位置的元素。

```
# 创建一个列表
numbers = [10, 20, 30, 40]

# 删除并返回索引为 2 的元素
removed_item = numbers.pop(2)

# 输出被删除的元素和更新后的列表
print(removed_item) # 输出: 30
print(numbers)     # 输出: [10, 20, 40]
```

3. 使用 `del` 语句

`del` 语句可以通过索引删除列表中的元素，也可以删除整个列表的一部分或全部。

```
# 创建一个列表
languages = ['Python', 'Java', 'C++', 'Ruby']

# 删除索引为 1 的元素
del languages[1]

# 输出更新后的列表
print(languages) # 输出: ['Python', 'C++', 'Ruby']
```

4. 使用切片删除多个元素

如果需要删除列表中多个连续的元素，可以使用切片赋值空列表方式或者结合 `del` 语句与切片操作。

```
# 创建一个列表
nums = [1, 2, 3, 4, 5, 6]

# 删除索引 2 到 4 的元素
nums[2:5] = [] # 等价于 del nums[2:5]

# 输出更新后的列表
print(nums) # 输出: [1, 2, 6]
```

未完待续