

5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
6.1. 调用函数
6.2. 定义函数
6.3. 函数的参数
6.4. 递归函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
8.1. 高阶函数
8.1.1. map/reduce
8.1.2. filter
8.1.3. sorted
8.2. 返回函数
8.3. 匿名函数
8.4. 装饰器
8.5. 偏函数
9. 模块
9.1. 使用模块
9.2. 安装第三方模块
10. 面向对象编程
10.1. 类和实例
10.2. 访问限制
10.3. 继承和多态
10.4. 获取对象信息
10.5. 实例属性和类属性
11. 面向对象高级编程
11.1. 使用_slots_
11.2. 使用@property
11.3. 多重继承
11.4. 定制类
11.5. 使用枚举类
11.6. 使用元类
12. 错误、调试和测试
12.1. 错误处理
12.2. 调试
12.3. 单元测试
12.4. 文档测试
13. IO编程

切片



廖雪峰 GitHub 知乎 Twitter

资深软件开发工程师，业余马拉松选手。

取一个list或tuple的部分元素是非常常见的操作。比如，一个list如下：

```
>>> L = ['Michael', 'Sarah', 'Tracy', 'Bob', 'Jack']
```

取前3个元素，应该怎么做？

笨办法：

```
>>> [L[0], L[1], L[2]]  
['Michael', 'Sarah', 'Tracy']
```

之所以是笨办法是因为扩展一下，取前N个元素就没辙了。

取前N个元素，也就是索引为0-(N-1)的元素，可以用循环：

```
>>> r = []  
>>> n = 3  
>>> for i in range(n):  
...     r.append(L[i])  
...  
>>> r  
['Michael', 'Sarah', 'Tracy']
```

对这种经常取指定索引范围的操作，用循环十分繁琐，因此，Python提供了切片（Slice）操作符，能大大简化这种操作。

对应上面的问题，取前3个元素，用一行代码就可以完成切片：

```
>>> L[0:3]  
['Michael', 'Sarah', 'Tracy']
```

L[0:3] 表示，从索引 0 开始取，直到索引 3 为止，但不包括索引 3。即索引 0, 1, 2，正好是3个元素。

如果第一个索引是 0，还可以省略：

```
>>> L[:3]  
['Michael', 'Sarah', 'Tracy']
```

也可以从索引1开始，取出2个元素出来：

```
>>> L[1:3]  
['Sarah', 'Tracy']
```

类似的，既然Python支持 L[-1] 取倒数第一个元素，那么它同样支持倒数切片，试试：

```
>>> L[-2:]  
['Bob', 'Jack']  
>>> L[-2:-1]  
['Bob']
```

记住倒数第一个元素的索引是 -1。

切片操作十分有用。我们先创建一个0-99的数列：

```
>>> L = list(range(100))  
>>> L  
[0, 1, 2, 3, ..., 99]
```

可以通过切片轻松取出某一段数列。比如前10个数：

5.3. 使用list和tuple

5.4. 条件判断

5.5. 模式匹配

5.6. 循环

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

8.1. 高阶函数

8.1.1. map/reduce

8.1.2. filter

8.1.3. sorted

8.2. 返回函数

8.3. 匿名函数

8.4. 装饰器

8.5. 偏函数

9. 模块

9.1. 使用模块

9.2. 安装第三方模块

10. 面向对象编程

10.1. 类和实例

10.2. 访问限制

10.3. 继承和多态

10.4. 获取对象信息

10.5. 实例属性和类属性

11. 面向对象高级编程

11.1. 使用_slots_

11.2. 使用@property

11.3. 多重继承

11.4. 定制类

11.5. 使用枚举类

11.6. 使用元类

12. 错误、调试和测试

12.1. 错误处理

12.2. 调试

12.3. 单元测试

12.4. 文档测试

13. IO编程

```
>>> L[:10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

后10个数:

```
>>> L[-10:]
[90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

前11-20个数:

```
>>> L[10:20]
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

前10个数, 每两个取一个:

```
>>> L[:10:2]
[0, 2, 4, 6, 8]
```

所有数, 每5个取一个:

```
>>> L[::5]
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
```

甚至什么都不写, 只写 [:] 就可以原样复制一个list:

```
>>> L[:]
[0, 1, 2, 3, ..., 99]
```

tuple也是一种list, 唯一区别是tuple不可变。因此, tuple也可以用切片操作, 只是操作的结果仍是tuple:

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

字符串 'xxx' 也可以看成是一种list, 每个元素就是一个字符。因此, 字符串也可以用切片操作, 只是操作结果仍是字符串:

```
>>> 'ABCDEFG'[::3]
'ABC'
>>> 'ABCDEFG'[::2]
'ACEG'
```

在很多编程语言中, 针对字符串提供了很多各种截取函数 (例如, substring), 其实目的就是对字符串切片。Python没有针对字符串的截取函数, 只需要切片一个操作就可以完成, 非常简单。

练习

利用切片操作, 实现一个trim()函数, 去除字符串首尾的空格, 注意不要调用str的 strip() 方法:

```
def trim(s):
    return s

# 测试:
if trim('hello ') != 'hello':
    print('测试失败!')
elif trim(' hello') != 'hello':
    print('测试失败!')
elif trim(' hello ') != 'hello':
    print('测试失败!')
elif trim(' hello world ') != 'hello world':
    print('测试失败!')
elif trim('') != '':
    print('测试失败!')
elif trim(' ') != '':
    print('测试失败!')
else:
    print('测试成功!')
```

参考源码

5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
6.1. 调用函数
6.2. 定义函数
6.3. 函数的参数
6.4. 递归函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
8.1. 高阶函数
8.1.1. map/reduce
8.1.2. filter
8.1.3. sorted
8.2. 返回函数
8.3. 匿名函数
8.4. 装饰器
8.5. 偏函数
9. 模块
9.1. 使用模块
9.2. 安装第三方模块
10. 面向对象编程
10.1. 类和实例
10.2. 访问限制
10.3. 继承和多态
10.4. 获取对象信息
10.5. 实例属性和类属性
11. 面向对象高级编程
11.1. 使用__slots__
11.2. 使用@property
11.3. 多重继承
11.4. 定制类
11.5. 使用枚举类
11.6. 使用元类
12. 错误、调试和测试
12.1. 错误处理
12.2. 调试
12.3. 单元测试
12.4. 文档测试
13. IO编程

小结

有了切片操作，很多地方循环就不再需要了。Python的切片非常灵活，一行代码就可以实现很多行循环才能完成的操作。

« 高级特性

迭代 »



Comments

>Loading comments...

©liaoxfeng.com - 微博 - GitHub - License