

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

迭代



廖雪峰



资深软件开发工程师，业余马拉松选手。

如果给定一个 `list` 或 `tuple`，我们可以通过 `for` 循环来遍历这个 `list` 或 `tuple`，这种遍历我们称为迭代（Iteration）。

在Python中，迭代是通过 `for ... in` 来完成的，而很多语言比如C语言，迭代 `list` 是通过下标完成的，比如C代码：

```
for (i=0; i<length; i++) {  
    n = list[i];  
}
```

可以看出，Python的 `for` 循环抽象程度要高于C的 `for` 循环，因为Python的 `for` 循环不仅可以用在 `list` 或 `tuple` 上，还可以作用在其他可迭代对象上。

`list` 这种数据类型虽然有以下标，但很多其他数据类型是没有下标的，但是，只要是可迭代对象，无论有无下标，都可以迭代，比如 `dict` 就可以迭代：

```
>>> d = {'a': 1, 'b': 2, 'c': 3}  
>>> for key in d:  
...     print(key)  
...  
a  
c  
b
```

因为 `dict` 的存储不是按照 `list` 的方式顺序排列，所以，迭代出的结果顺序很可能不一样。

默认情况下，`dict` 迭代的是key。如果要迭代value，可以用 `for value in d.values()`，如果要同时迭代key和value，可以用 `for k, v in d.items()`。

由于字符串也是可迭代对象，因此，也可以作用于 `for` 循环：

```
>>> for ch in 'ABC':  
...     print(ch)  
...  
A  
B  
C
```

所以，当我们使用 `for` 循环时，只要作用于一个可迭代对象，`for` 循环就可以正常运行，而我们不太关心该对象究竟是 `list` 还是其他数据类型。

那么，如何判断一个对象是可迭代对象呢？方法是通过 `collections.abc` 模块的 `Iterable` 类型判断：

```
>>> from collections.abc import Iterable  
>>> isinstance('abc', Iterable) # str是否可迭代  
True  
>>> isinstance([1,2,3], Iterable) # list是否可迭代  
True  
>>> isinstance(123, Iterable) # 整数是否可迭代  
False
```

最后一个小问题，如果要对 `list` 实现类似Java那样的下标循环怎么办？Python内置的 `enumerate` 函数可以把一个 `list` 变成索引-元素对，这样就可以在 `for` 循环中同时迭代索引和元素本身：

```
>>> for i, value in enumerate(['A', 'B', 'C']):  
...     print(i, value)  
...  
0 A  
1 B  
2 C
```

上面的 `for` 循环里，同时引用了两个变量，在Python里是很常见的，比如下面的代码：

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

```
>>> for x, y in [(1, 1), (2, 4), (3, 9)]:
...     print(x, y)
...
1 1
2 4
3 9
```

练习

请使用迭代查找一个list中最小和最大值，并返回一个tuple：

```
def findMinAndMax(L):
    return (None, None)

# 测试
if findMinAndMax([]) != (None, None):
    print('测试失败!')
elif findMinAndMax([7]) != (7, 7):
    print('测试失败!')
elif findMinAndMax([7, 1]) != (1, 7):
    print('测试失败!')
elif findMinAndMax([7, 1, 3, 9, 5]) != (1, 9):
    print('测试失败!')
else:
    print('测试成功!')
```

参考源码

[do_iter.py](#)

小结

任何可迭代对象都可以作用于 `for` 循环，包括我们自定义的数据类型，只要符合迭代条件，就可以使用 `for` 循环。

« 切片

列表生成式 »



Comments

Loading comments...

©liaoxxuefeng.com - 微博 - GitHub - License