

07 | 修炼基本功：条件与循环

2019-05-24 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 09:48 大小 8.98M



你好，我是景霄。


前面几节，我们一起学习了列表、元祖、字典、集合和字符串等一系列 Python 的基本数据类型。但是，如何把这一个个基本的数据结构类型串接起来，组成一手漂亮的代码呢？这就是我们今天所要讨论的“条件与循环”。

我习惯把“条件与循环”，叫做编程中的基本功。为什么称它为基本功呢？因为它控制着代码的逻辑，可以说是程序的中枢系统。如果把写程序比作盖楼房，那么条件与循环就是楼房的根基，其他所有东西都是在此基础上构建而成。

毫不夸张地说，写一手简洁易读的条件与循环代码，对提高程序整体的质量至关重要。


条件语句

首先，我们一起来看一下 Python 的条件语句，用法很简单。比如，我想要表示 $y=|x|$ 这个函数，那么相应的代码便是：

 复制代码

```
1 # y = |x|
2 if x < 0:
3     y = -x
4 else:
5     y = x
```


和其他语言不一样，我们不能在条件语句中加括号，写成下面这样的格式。

 复制代码

```
1 if (x < 0)
```

但需要注意的是，在条件语句的末尾必须加上冒号（:），这是 Python 特定的语法规则。

由于 Python 不支持 switch 语句，因此，当存在多个条件判断时，我们需要用 else if 来实现，这在 Python 中的表达是 **elif**。语法如下：

 复制代码


```
1 if condition_1:
2     statement_1
3 elif condition_2:
4     statement_2
5 ...
6 elif condition_i:
7     statement_i
8 else:
9     statement_n
```

整个条件语句是顺序执行的，如果遇到一个条件满足，比如 `condition_i` 满足时，在执行完 `statement_i` 后，便会退出整个 if、elif、else 条件语句，而不会继续向下执行。这个语句

在工作中很常用，比如下面的这个例子。

实际工作中，我们经常用 ID 表示一个事物的属性，然后进行条件判断并且输出。比如，在 integrity 的工作中，通常用 0、1、2 分别表示一部电影的色情暴力程度。其中，0 的程度最高，是 red 级别；1 其次，是 yellow 级别；2 代表没有质量问题，属于 green。


如果给定一个 ID，要求输出某部电影的质量评级，则代码如下：

 复制代码

```
1 if id == 0:
2     print('red')
3 elif id == 1:
4     print('yellow')
5 else:
6     print('green')
```

不过要注意，if 语句是可以单独使用的，但 elif、else 都必须和 if 成对使用。

另外，在我们进行条件判断时，不少人喜欢省略判断的条件，比如写成下面这样：

 复制代码


```
1 if s: # s is a string
2     ...
3 if l: # l is a list
4     ...
5 if i: # i is an int
6     ...
7 ...
```

关于省略判断条件的常见用法，我大概总结了一下：

判断条件的省略用法


数据类型	结果
String	空字符串解析为False，其余为True
Int	0解析为False，其余为True
Bool	True为True，False为False
list/tuple/dict/set	Iterable为空解析为False，其余为True
Object	None解析为False，其余为True

不过，切记，在实际写代码时，我们鼓励，除了 boolean 类型的数据，条件判断最好是显性的。比如，在判断一个整型数是否为 0 时，我们最好写出判断的条件：

 复制代码

```
1 if i != 0:
2     ...
```

而不是只写出变量名：


 复制代码

```
1 if i:
2     ...
```

循环语句

讲完了条件语句，我们接着来看循环语句。所谓循环，顾名思义，本质上就是遍历集合中的元素。和其他语言一样，Python 中的循环一般通过 for 循环和 while 循环实现。

比如，我们有一个列表，需要遍历列表中的所有元素并打印输出，代码如下：


 复制代码

```
1 l = [1, 2, 3, 4]
2 for item in l:
3     print(item)
```

```
4 1
5 2
6 3
7 4
```

你看，是不是很简单呢？

其实，Python 中的数据结构只要是可迭代的（iterable），比如列表、集合等等，那么都可以通过下面这种方式遍历：

 复制代码

```
1 for item in <iterable>:
2     ...
```


这里需要单独强调一下字典。字典本身只有键是可迭代的，如果我们要遍历它的值或者是键值对，就需要通过其内置的函数 `values()` 或者 `items()` 实现。其中，`values()` 返回字典的值的集合，`items()` 返回键值对的集合。

 复制代码

```
1 d = {'name': 'jason', 'dob': '2000-01-01', 'gender': 'male'}
2 for k in d: # 遍历字典的键
3     print(k)
4 name
5 dob
6 gender
7
8 for v in d.values(): # 遍历字典的值
9     print(v)
10 jason
11 2000-01-01
12 male
13
14 for k, v in d.items(): # 遍历字典的键值对
15     print('key: {}, value: {}'.format(k, v))
16 key: name, value: jason
17 key: dob, value: 2000-01-01
18 key: gender, value: male
```


看到这里你也许会问，有没有办法通过集合中的索引来遍历元素呢？当然可以，其实这种情况在实际工作中还是很常见的，甚至很多时候，我们还得根据索引来做一些条件判断。

我们通常通过 `range()` 这个函数，拿到索引，再去遍历访问集合中的元素。比如下面的代码，遍历一个列表中的元素，当索引小于 5 时，打印输出：

 复制代码

```
1 l = [1, 2, 3, 4, 5, 6, 7]
2 for index in range(0, len(l)):
3     if index < 5:
4         print(l[index])
5
6 1
7 2
8 3
9 4
10 5
```


当我们同时需要索引和元素时，还有一种更简洁的方式，那就是通过 Python 内置的函数 `enumerate()`。用它来遍历集合，不仅返回每个元素，并且还返回其对应的索引，这样一来，上面的例子就可以写成：

 复制代码

```
1 l = [1, 2, 3, 4, 5, 6, 7]
2 for index, item in enumerate(l):
3     if index < 5:
4         print(item)
5
6 1
7 2
8 3
9 4
10 5
```


在循环语句中，我们还常常搭配 `continue` 和 `break` 一起使用。所谓 `continue`，就是让程序跳过当前这层循环，继续执行下面的循环；而 `break` 则是指完全跳出所在的整个循环体。在循环中适当加入 `continue` 和 `break`，往往能使程序更加简洁、易读。

比如，给定两个字典，分别是产品名称到价格的映射，和产品名称到颜色列表的映射。我们要找出价格小于 1000，并且颜色不是红色的所有产品名称和颜色的组合。如果不用 `continue`，代码应该是下面这样的：

 复制代码

```
1 # name_price: 产品名称 (str) 到价格 (int) 的映射字典
2 # name_color: 产品名字 (str) 到颜色 (list of str) 的映射字典
3 for name, price in name_price.items():
4     if price < 1000:
5         if name in name_color:
6             for color in name_color[name]:
7                 if color != 'red':
8                     print('name: {}, color: {}'.format(name, color))
9         else:
10            print('name: {}, color: {}'.format(name, 'None'))
```

而加入 `continue` 后，代码显然清晰了很多：


 复制代码

```
1 # name_price: 产品名称 (str) 到价格 (int) 的映射字典
2 # name_color: 产品名字 (str) 到颜色 (list of str) 的映射字典
3 for name, price in name_price.items():
4     if price >= 1000:
5         continue
6     if name not in name_color:
7         print('name: {}, color: {}'.format(name, 'None'))
8         continue
9     for color in name_color[name]:
10        if color == 'red':
11            continue
12        print('name: {}, color: {}'.format(name, color))
```

我们可以看到，按照第一个版本的写法，从开始一直到打印输出符合条件的产品名称和颜色，共有 5 层 `for` 或者 `if` 的嵌套；但第二个版本加入了 `continue` 后，只有 3 层嵌套。


显然，如果代码中出现嵌套里还有嵌套的情况，代码便会变得非常冗余、难读，也不利于后续的调试、修改。因此，我们要尽量避免这种多层嵌套的情况。

前面讲了 for 循环，对于 while 循环，原理也是一样的。它表示当 condition 满足时，一直重复循环内部的操作，直到 condition 不再满足，就跳出循环体。

 复制代码

```
1 while condition:
2     ....
```

很多时候，for 循环和 while 循环可以互相转换，比如要遍历一个列表，我们用 while 循环同样可以完成：


 复制代码

```
1 l = [1, 2, 3, 4]
2 index = 0
3 while index < len(l):
4     print(l[index])
5     index += 1
```

那么，两者的使用场景又有什么区别呢？

通常来说，如果你只是遍历一个已知的集合，找出满足条件的元素，并进行相应的操作，那么使用 for 循环更加简洁。但如果你需要在满足某个条件前，不停地重复某些操作，并且没有特定的集合需要去遍历，那么一般则会使用 while 循环。

比如，某个交互式问答系统，用户输入文字，系统会根据内容做出相应的回答。为了实现这个功能，我们一般会使用 while 循环，大致代码如下：


 复制代码

```
1 while True:
2     try:
3         text = input('Please enter your questions, enter "q" to exit')
4         if text == 'q':
5             print('Exit system')
6             break
7         ...
8         ...
9         print(response)
10    except as err:
```




```
11         print('Encountered error: {}'.format(err))
12         break
```

同时需要注意的是，for 循环和 while 循环的效率问题。比如下面的 while 循环：

 复制代码

```
1 i = 0
2 while i < 1000000:
3     i += 1
```

和等价的 for 循环：

 复制代码

```
1 for i in range(0, 1000000):
2     pass
```


究竟哪个效率高呢？

要知道，range() 函数是直接由 C 语言写的，调用它速度非常快。而 while 循环中的 “i += 1” 这个操作，得通过 Python 的解释器间接调用底层的 C 语言；并且这个简单的操作，又涉及到了对象的创建和删除（因为 i 是整型，是 immutable，i += 1 相当于 i = new int(i + 1)）。所以，显然，for 循环的效率更胜一筹。

条件与循环的复用

前面两部分讲了条件与循环的一些基本操作，接下来，我们重点来看它们的进阶操作，让程序变得更简洁高效。

在阅读代码的时候，你应该常常会发现，有很多将条件与循环并做一行的操作，例如：

 复制代码

```
1 expression1 if condition else expression2 for item in iterable
```

将这个表达式分解开来，其实就等同于下面这样的嵌套结构：

 复制代码


```
1 for item in iterable:
2     if condition:
3         expression1
4     else:
5         expression2
```

而如果没有 else 语句，则需要写成：

 复制代码


```
1 expression for item in iterable if condition
```

举个例子，比如我们要绘制 $y = 2*|x| + 5$ 的函数图像，给定集合 x 的数据点，需要计算出 y 的数据集合，那么只用一行代码，就可以很轻松地解决问题了：

 复制代码


```
1 y = [value * 2 + 5 if value > 0 else -value * 2 + 5 for value in x]
```

再比如我们在处理文件中的字符串时，常常遇到的一个场景：将文件中逐行读取的一个完整语句，按逗号分割单词，去掉首位的空字符，并过滤掉长度小于等于 3 的单词，最后返回由单词组成的列表。这同样可以简洁地表达成一行：

 复制代码


```
1 text = ' Today, is, Sunday'
2 text_list = [s.strip() for s in text.split(',') if len(s.strip()) > 3]
3 print(text_list)
4 ['Today', 'Sunday']
```

当然，这样的复用并不仅仅局限于一个循环。比如，给定两个列表 `x`、`y`，要求返回 `x`、`y` 中所有元素对组成的元祖，相等情况除外。那么，你也可以很容易表示出来：

 复制代码

```
1 [(xx, yy) for xx in x for yy in y if xx != yy]
```

这样的写法就等价于：

 复制代码

```
1 l = []
2 for xx in x:
3     for yy in y:
4         if xx != yy:
5             l.append((xx, yy))
```

熟练之后，你会发现这种写法非常方便。当然，如果遇到逻辑很复杂的复用，你可能会觉得写成一行难以理解、容易出错。那种情况下，用正常的形式表达，也不失为一种好的规范和选择。

总结

今天这节课，我们一起学习了条件与循环的基本概念、进阶用法以及相应的应用。这里，我重点强调几个易错的地方。

在条件语句中，`if` 可以单独使用，但是 `elif` 和 `else` 必须和 `if` 同时搭配使用；而 `If` 条件语句的判断，除了 `boolean` 类型外，其他的最好显示出来。


在 `for` 循环中，如果需要同时访问索引和元素，你可以使用 `enumerate()` 函数来简化代码。

写条件与循环时，合理利用 `continue` 或者 `break` 来避免复杂的嵌套，是十分重要的。

要注意条件与循环的复用，简单功能往往可以用一行直接完成，极大地提高代码质量与效率。

思考题

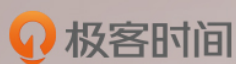
最后给你留一个思考题。给定下面两个列表 `attributes` 和 `values`，要求针对 `values` 中每一组子列表 `value`，输出其和 `attributes` 中的键对应后的字典，最后返回字典组成的列表。

 复制代码

```
1 attributes = ['name', 'dob', 'gender']
2 values = [['jason', '2000-01-01', 'male'],
3 ['mike', '1999-01-01', 'male'],
4 ['nancy', '2001-02-01', 'female']
5 ]
6
7 # expected outout:
8 [{ 'name': 'jason', 'dob': '2000-01-01', 'gender': 'male' },
9 { 'name': 'mike', 'dob': '1999-01-01', 'gender': 'male' },
10 { 'name': 'nancy', 'dob': '2001-02-01', 'gender': 'female' }]
```

你能分别用一行和多行条件循环语句，来实现这个功能吗？

欢迎在留言区写下你的答案，还有你今天学习的心得和疑惑，也欢迎你把这篇文章分享给你的同事、朋友。



Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

上一篇 06 | Python “黑箱”：输入与输出

下一篇 08 | 异常处理：如何提高程序的稳定性？

精选留言 (82)

写留言



yshan

2019-05-24

33

`[(xx, yy) for xx in x for yy in y if x != y]`

应该是 `if xx != yy]` 吧

思考题一行:

`[dict(zip(attributes,v)) for v in values]`

...

展开

作者回复: 嗯嗯，笔误已更新。

思考题



AllenGFLiu

2019-05-25

14

思考题

一行版:

`[dict(zip(attributes, value)) for value in values]`

循环版: ...

展开



LiANGZE

2019-05-24

12

课后的练习题，手机打的，格式可能不好看

`print([{ attributes[i]: value[i] for i in range(len(attributes)) } for value in values])`



Fergus

2019-05-26

👍 6

Ask

```
attributes = ['name', 'dob', 'gender']
values = [['jason', '2000-01-01', 'male'],
          ['mike', '1999-01-01', 'male'],
          ['nancy', '2001-02-01', 'female']]...
```

展开 ▾



Deed

2019-05-24

👍 4

```
[dict(zip(attributes, value)) for value in values]
```

展开 ▾



武林秀才

2019-05-28

👍 3

一行的

```
[dict([(attributes[j], values[i][j]) for j in range(len(attributes))]) for i in
range(len(values))]
```

展开 ▾



k8scloud

2019-05-26

👍 2

```
attributes = ['name', 'dob', 'gender']
values = [['jason', '2000-1-01', 'male'], ['mike', '1999-01-1', 'male'], ['nancy', '201-02-01', 'female']]
output = []
for v in values:...
```

展开 ▾



夜下凝月

2019-05-25

👍 2

Python中没有switch，但是可以用字典来代替。

#不调用函数

```
level = {0:'red', 1:'yellow', 2:'green'}
```

try:

level(id)...

展开 ▾



夏秋冬的春...

2019-05-31

👍 1

多行

```
output_dist = []
```

```
for v in values:
```

```
    dict_at = {}
```

```
    for i in range(len(attributes)):
```

展开 ▾



Geek_d848f...

2019-05-29

👍 1

#多行

```
res=[]
```

```
for value in values:
```

```
    if len(attributes)!=len(value):
```

```
        continue
```

```
    temp={}
```

```
    for index,each in enumerate(value):
```

```
        temp[attributes[index]]=each
```

```
    res.append(temp)
```

```
print(res)
```

#使用zip

```
s=[dict(zip(attributes,value)) for value in values if len(attributes)==len(value)]
```



Wing•三金

2019-05-27

👍 1

多行版本

```
combined = []
```

```
for value in values:
```

```
    temp_dict = {}
```

```
    for index, attr in enumerate(attributes):
```

展开 ▾



👍 1

展开 ∨



 1

多行:

```
for value in values:
```

展开 ∨



👍 1

```
temp_dict={}...
```

展开 ∨



```
for index , k in enumerate(attributes):...
```

展开 ∨



```
get_dict={}
```

```
get_arr=[]  
for index, value in enumerate(values):  
    for index1,key in enumerate(attributes):...
```

展开 ▾



supermouse

2019-06-05



多行的:

```
attributes = ['name', 'dob', 'gender']  
values = [['jason', '2000-01-01', 'male'],  
['mike', '1999-01-01', 'male'],  
['nancy', '2001-02-01', 'female']]...
```

展开 ▾



A0.何文祥

2019-06-05



和其他语言不一样，我们不能在条件语句中加括号，写成下面这样的格式。

可以，但是不必要，用不能不是很妥。



林子里

2019-06-03



```
[dict(zip(attributes,values[i])) for i in range(0,len(values))]
```

展开 ▾



易拉罐

2019-06-03



general.

```
expected_output = []
```

```
for v in values:
```

```
    expected_output.append(dict(zip(attributes, v)))
```

```
print(expected_output)...
```

展开 ▾

