

---

title: "Python 第三课" tags: [Python, for, range(), append(), "\*\*", 循环, 列表解析] categories: 资料

## 先说一句...

---

- 这周按计划已是本学期最后一次 Python 课程。希望大家能复习过去三周内所学、重温过去三周内所思，并充满激情地完成综合实践任务。
- 请注意：**11月11日的社团活动暂停一次。后续安排见群。**
- 如果你需要下载或打印这篇文章，[点击这里下载 PDF 版本](#)。

## 迭代循环 Iterative Loop

---

### 定义

迭代是用计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机重复执行一些步骤。在可以用迭代算法解决的问题中，至少存在一个能不断由旧值递推出新值的变量，这个变量被称为迭代变量。在每次执行这些步骤时，迭代变量的原值推出它的一个新值。

在 Python 中，`for` 循环结构就是一种迭代的过程。与 C/C++ 不同的是，Python 很多情况下有关迭代变量的操作更简单。

为什么 `for` 结构被称为**迭代循环**？当然是因为还有别的循环类型存在嘛。在 Python 中，存在另一种由 `while` 结构组成的循环。详情可参考[这篇博客](#)。

### 语法

在 Python 中，`for` 循环典型语法形式如下：

```
for iterating_var in sequence:
    do_something()
```

其中

- `for` 和 `in` 为关键字，是这个结构中不可改变或去掉的部分。
- `iterating_var` 就是上面提到的递归变量。可为多种数据类型，具体取决于 `sequence`。
- `sequence` 是递归变量所存在的序列。换句话说，这个序列确定了迭代变量的原值推出它的一个新值的依据。很多情况下，`sequence` 是一个列表，而对应的迭代方式即是让每执行一次 `do_something()`，即循环体内的代码，`iterating_var` 的值变为列表中下一个元素的值。列表中有几个元素，循环就执行几遍。

另外，请仔细观察上面这个例子的代码缩进和符号 `:`。

### 使用

最简单的例子

下面的循环使用for循环打印出了列表topics中的所有元素。

```
topics = ['maker', 'arduino', 'ml', 'nl']
for topic in topics:
    print(topic)
```

再次重复：topic为迭代变量；topics是作为迭代序列的列表。与 C/C++ 不同，Python 的迭代变量不需要额外定义。

这三行代码等价于

```
topics = ['maker', 'arduino', 'ml', 'nl']
print(topics[0])
print(topics[1])
print(topics[2])
print(topics[3])
```

可见，循环语句减少了表意重复的代码量。

与range()搭配使用

**range()函数**

**定义**

顾名思义，range()的返回值是一个特定的范围。

**语法**







这是一个学习使用 Python 帮助文档的绝佳机会。☺

1. 打开一个Python Shell, 如图所示。

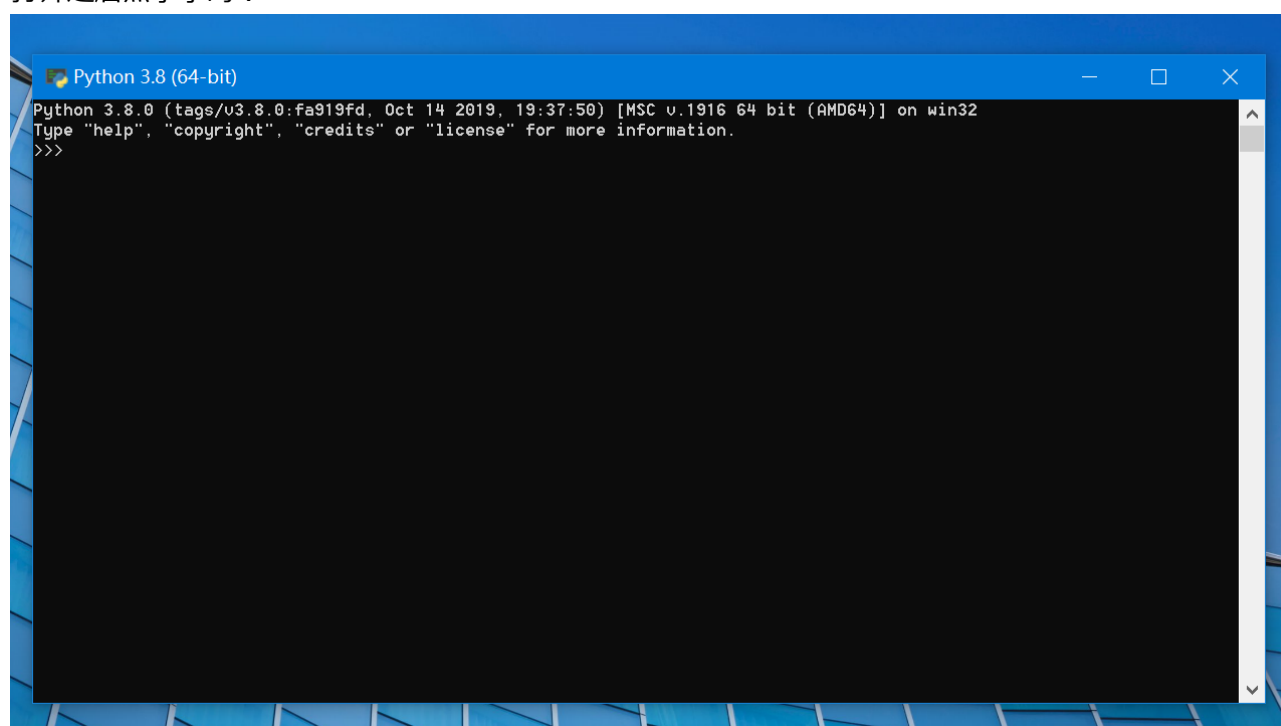


# Python 3.8 (64-bit)

应用

-  打开
-  以管理员身份运行
-  打开文件位置
-  固定到“开始”屏幕
-  固定到任务栏
-  卸载

打开之后黑乎乎的：



## 2. 输入命令：

```
help(range)
```

在`help()`函数的小括号可以里填任何你想查询的准确关键词。

## 3. 抓住关键部分：

```
Return an object that produces a sequence of integers from start (inclusive) to stop (exclusive) by step. range(i, j) produces i, i+1, i+2, ..., j-1. Start defaults to 0, and stop is omitted! range(4) produces 0, 1, 2, 3. These are exactly the valid indices for a list of 4 elements. When step is given, it specifies the increment (or decrement).
```

## 4. 有需可自行翻译。

当然，帮助文档解决不了你的疑问没关系，因为下面还要讲的嘛。

### 最简单的例子

`range()`接受一个参数即可正常工作。这个参数一般情况下是个整数：

```
for counter in range(5):  
    print(str(counter))
```

在课上我们已经注意到了，输出有点和预期不符：

```
0  
1  
2  
3  
4
```

原因很简单：此时`range()`的作用是返回从0至比参数唯一的参数j小1的值（j-1）组成的范围。这就是编程中常见的差一行为。使用`range()`时，如果输出不符合预期且不打算动脑筋排查错误，尝试将参数j的值加1或减1是极好的下下策。

### 更多参数

事实上，`range`可以接受三个参数：

```
for counter in range(1, 10, 2):  
    print(str(counter))
```

例子中的“1”(参数*i*)替换的是“最简单的例子”里“0”的功能，即范围的起始值。“2”被称为*step*，也就是步长，可理解成数学中等差数列的公差。自然，代码的输出是：

```
1
3
5
7
9
```

结论

在标准结构

```
range(i, j, step)
```

中，只有终点*j*是不可或缺的参数。三者的相对位置不能改变。只要满足这两个要求，*range()*函数随便你用啦。

举例

用*range()*生成的数字序列代替列表在一些情况下很方便。下面用两个例子具体展示。

**例：输出100以内的完全平方数**

问题等价于输出1到10的平方数。可以从1数到10，并用*print()*打印出当前迭代变量的平方数。

```
for counter in range(1, 11):
    sqr = counter ** 2
    print(str(sqr))
```

输出是

```
1
4
9
16
25
36
49
64
81
100
```

其中`**`是指数运算符。使用格式为`m ** n`，`m`是底数，`n`是指数。因此，`2 ** 5`的意思是“2的5次方”。

思考：为什么输出的是1到10的平方数，`range()`的第二个参数是11？

### 例：输出500以内能被3整除的数

#### 方法一：条件语句

`%`在 Python 中是一个运算符，表示取模。`m % n`的值是`m`除以`n`的余数。

从1数到500，每次判断当前这个数除以3余数是否为0，即是否能被3整除。若能，则用`print()`打印出这个数。

```
for counter in range(1, 501):  
    if counter % 3 == 0:  
        print(str(counter))
```

#### 方法二：巧用step

从3开始数到500，用`step`定义每隔3个数一次并打印出当前数到的数，也可以解决问题。

```
for counter in range(3, 501, 3):  
    print(str(counter))
```

两种方法等价，输出都是（...省略了一些结果）

```
3  
6  
9  
12  
15  
...  
...  
...  
492  
495  
498
```

### 嵌套循环

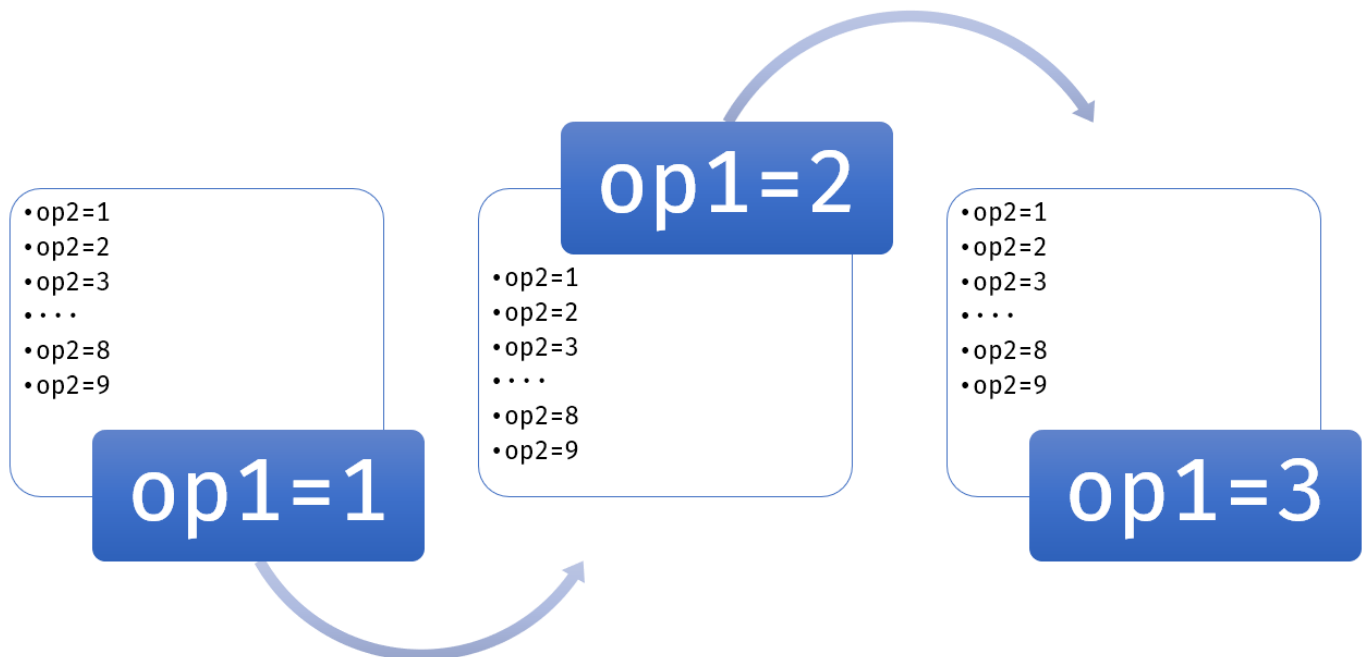
顾名思义，嵌套循环指的就是一个`for`循环结构的内部还嵌了一层或多层`for`循环。下面用一个例子具体展示。

如何帮助你小学的弟弟妹妹解决九九乘法表？不妨用 Python 打印出来一个给TA背。📖

请仔细观察代码缩进。

```
for op1 in range(1, 10):
    for op2 in range(1, 10):
        current_product = op1 * op2
        current_output = str(op1) + "*" + str(op2) + "=" + str(current_product)
        print(current_output, end=" ")
    print("\n")
```

首先，看看两个循环是怎么嵌套的：每当外循环的迭代变量`op1`从`range(1, 10)`取一个时，内循环总是完整地执行了一遍。也就是说，当`op1`数1个数时，`op2`数了9个数。下面这幅图展示了`op1`从1迭代到3的过程。



九九乘法表的每一个基本单元都是 $m*n=x$ 的形式，正好用`op1`表示 $m$ ，`op2`表示 $n$ 。

内循环中剩下的代码很好理解。`current_product`是 $x$ 。`current_output`把两个乘数和积都转换成字符串并用`+`符号拼接上乘号和等于符号。内循环每执行9次，也就是`op2`每从1数到9，会执行一次`print("\n")`来换行，使得每一横行都是九个基本单元。

另外你可能已经注意到了内循环里的`print()`有一个没见过的参数`end`。但善于观察的你一定见过 Python 其实每个`print()`之后都自动换行的现象。也就是说，

```
print("1")
print("2")
print("3")
```

的输出并不是

```
123
```

而是

```
1
2
3
```

不信你试！😏

原理：Python 的每一行`print()`出来的语句都有结尾字符，默认是`\n`。在通常情况下这个功能对懒癌有极好的促进作用，但显然在`1*1=1`和`1*1=2`之间需要的不是换行，而是空格。因此，使用`end=" "`把默认的结尾字符换成两个空格。

最后，看看输出：

```
1*1=1  1*2=2  1*3=3  1*4=4  1*5=5  1*6=6  1*7=7  1*8=8  1*9=9
2*1=2  2*2=4  2*3=6  2*4=8  2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
3*1=3  3*2=6  3*3=9  3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
4*1=4  4*2=8  4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
5*1=5  5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
6*1=6  6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
7*1=7  7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
8*1=8  8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
9*1=9  9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

## 循环与列表

列表本身就和`range()`函数一样，可以作为`for`迭代循环结构中的序列。自然而然，循环和列表有着不解之缘。

### 列表解析 list comprehension

列表解析是一种看起来有些玄学但效率很高的语法结构。在下面的例子中，一行代码更比三行强：

```
sqr = [value**2 for value in range(1, 11)]
```

生成的列表`sqr`等价于

```
sqr = []
for value in range(1, 11):
    sqr.append(value ** 2)
```



生成的列表 `sqr`。不信你可以把两种方法生成的 `sqr` 打印出来观察是否一模一样。

`append()` 方法的功能是把括号里的参数当作元素，添加到 . 之前的那个列表（此处是 `sqr`）里去。注意 Python 方法和函数在语法上的区别：方法用一个“.”（英文句号，半角）连接它的操作对象。

你能借助第二种列表生成方法自己总结出列表解析的语法规则吗？

## 遍历列表 list traversal

所谓遍历，是指沿着某条搜索路线，依次对某个数据结构中每个节点均做一次访问。对于 Python 的列表数据结构来讲，遍历指的是访问每一个从头到尾的元素。举例？你已经看过啦，就是这篇文章开头处“最简单的例子”。

## 另外...（首尾照应）

---

- 重要的事情再说一遍：**11月11日的社团活动暂停一次。后续安排见群。**
- 请**认真看完这篇并亲自写一遍并执行了一下所有程序**的你在这个页面的评论区回复一个“名字+已完成”。（“幸好翻到了这里”，你庆幸道。😊）
- 请**大家尽量自己亲自写一遍上面的代码并执行**。编程如同刷题，不过手、当“云玩家”真的很难进步！有任何问题，**都可以**必须在下面的评论区或是 Lumix 2019 QQ群里提出来！