

# 循环

---

## 一、实验介绍

---

### 1.1 实验内容

---

在以前的例子里，有些时候我们需要多次执行相同的任务，我们使用一个计数器来检查代码需要执行的次数。这个技术被称为循环。

### 1.2 实验知识点

---

- while 循环
- print() 函数的 end 参数
- 列表
  - 索引
  - 切片
- for 循环
- range() 函数
- continue 关键字
- for 循环中的 else 关键字

### 1.3 实验环境

- python3.5
- Xfce终端
- Vim

## 1.4 适合人群

本课程属于初级级别课程，不仅适用于那些有其它语言基础的同学，对没有编程经验的同学也非常友好。

## 二、实验步骤

### 2.1 while 循环

while 语句的语法如下：

```
while condition:
    statement1
    statement2
```

想要多次执行的代码必须以正确的缩进放在 while 语句下面。在表达式 *condition* 为真的时候它们才会执行。同 *if-else* 一样，非零值为真。让我们写一个简单的代码，它按顺序打印 0 到 10 的数字：

```
>>> n = 0
>>> while n < 11:
...     print(n)
...     n += 1
...
0
1
2
3
4
5
6
7
8
9
10
```

在第一行我们使  $n = 0$  , 然后在 `while` 语句中把条件设置为  $n < 11$  , 这意味着在 `while` 语句下面缩进的所有行将会被执行, 直到  $n$  的值大于等于11。在循环里我们只是打印  $n$  的值然后令它增一。


想想如果没有循环语句, 你想要打印 0 到 10 的所有数字, 那你得手动打印 11 次!

## 2.1.1 斐波那契 ( Fibonacci ) 数列

让我们来试试打印斐波那契数列。这个数列前两项为 1, 之后的每一个项都是前两项之和。所以这个数列看起来就像这样: *1, 1, 2, 3, 5, 8, 13 .....*

```
#!/usr/bin/env python3
a, b = 0, 1
while b < 100:
    print(b)
    a, b = b, a + b
```

运行程序:

A terminal window titled "Terminal 终端 - shiyanlou@ad8b41e7ec57: ~" shows the execution of a Python script. The prompt is "shiyanlou:~/ \$ ./fibonacci1.py" and the output is a list of Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. The terminal also shows the prompt "shiyanlou:~/ \$" and a timestamp "[16:14:14]".

```
Terminal 终端 - shiyanlou@ad8b41e7ec57: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./fibonacci1.py
1
1
2
3
5
8
13
21
34
55
89
shiyanlou:~/ $
```

第一行代码中我们初始化  $a$  和  $b$ 。当  $b$  的值小于 100 的时候, 循环执行代码。循环里我们首先打印  $b$  的值, 然后在下一行将  $a + b$  的值赋值给  $b$ ,  $b$  的值赋值给  $a$ 。

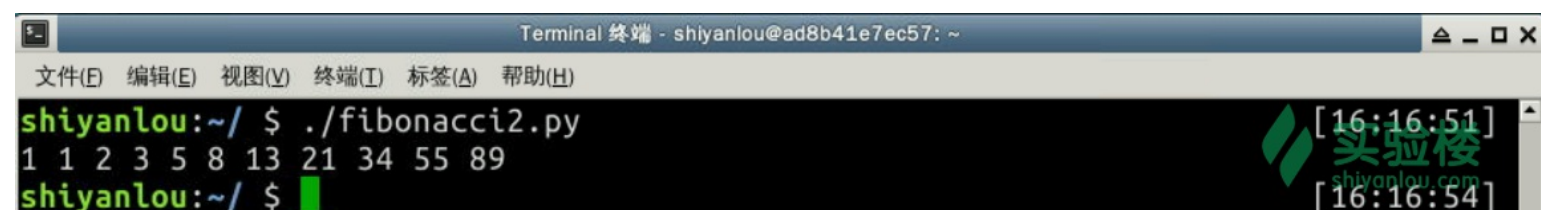
学习其他语言的同学在这里可能有些困惑, 你可以这样理解, Python 中赋值语句执行时会先对赋值运算符右边的表达式求值, 然后将这个值赋值给左边的变量。

默认情况下，`print()` 除了打印你提供的字符串之外，还会打印一个换行符，所以每调用一次 `print()` 就会换一次行，如同上面一样。

你可以通过 `print()` 的另一个参数 `end` 来替换这个换行符，就像下面这样：

```
#!/usr/bin/env python3
a, b = 0, 1
while b < 100:
    print(b, end=' ')
    a, b = b, a + b
print()
```

运行程序：



## 2.1.2 幂级数

我们来写一个程序计算幂级数：
$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} \quad (0 < x < 1)$$
。

```
#!/usr/bin/env python3
x = float(input("Enter the value of x: "))
n = term = num = 1
result = 1.0
while n <= 100:
    term *= x / n
    result += term
    n += 1
    if term < 0.0001:
        break
print("No of Times= {} and Sum= {}".format(n, result))
```

运行程序：

```
Terminal 终端 - shiyanlou@ad8b41e7ec57: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./powerseries.py [16:18:07]
Enter the value of x: 0
No of Times= 2 and Sum= 1.0
shiyanlou:~/ $ ./powerseries.py [16:18:20]
Enter the value of x: 0.1
No of Times= 5 and Sum= 1.1051708333333332
shiyanlou:~/ $ ./powerseries.py [16:18:25]
Enter the value of x: 0.5
No of Times= 7 and Sum= 1.6487196180555554
shiyanlou:~/ $ [16:18:30]
```

在这个程序里我们介绍一个新的关键字 `break`，它可以终止最里面的循环。这个例子里我们在 `if` 语句里使用 `break`：

```
if term < 0.0001:
    break
```

这意味着如果 `term` 的值小于 `0.0001`，那么终止循环。


### 2.1.3 乘法表

这个例子里我们打印 10 以内的乘法表。

```
#!/usr/bin/env python3
i = 1
print("-" * 50)
while i < 11:
    n = 1
    while n <= 10:
        print("{:4d}".format(i * n), end=' ')
        n += 1
    print()
    i += 1
print("-" * 50)
```

运行如下：

```
Terminal 终端 - shiyanlou@ad8b41e7ec57: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./multiplication.py [16:20:24]
-----
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
-----
shiyanlou:~/ $
```



这里我们在 while 循环里使用了另一个 while 循环，这被称为嵌套循环。你应该已经看到一条有趣的语句：

```
print("-" * 50)
```

字符串若是乘上整数 n，将返回由 n 个此字符串拼接起来的新字符串。

下面是一些例子：

```
>>> 's' * 10
'ssssssssss'
>>> print("*" * 10)
*****
>>> print("#" * 20)
#####
>>> print("--" * 20)
-----
>>> print("-" * 40)
-----
```

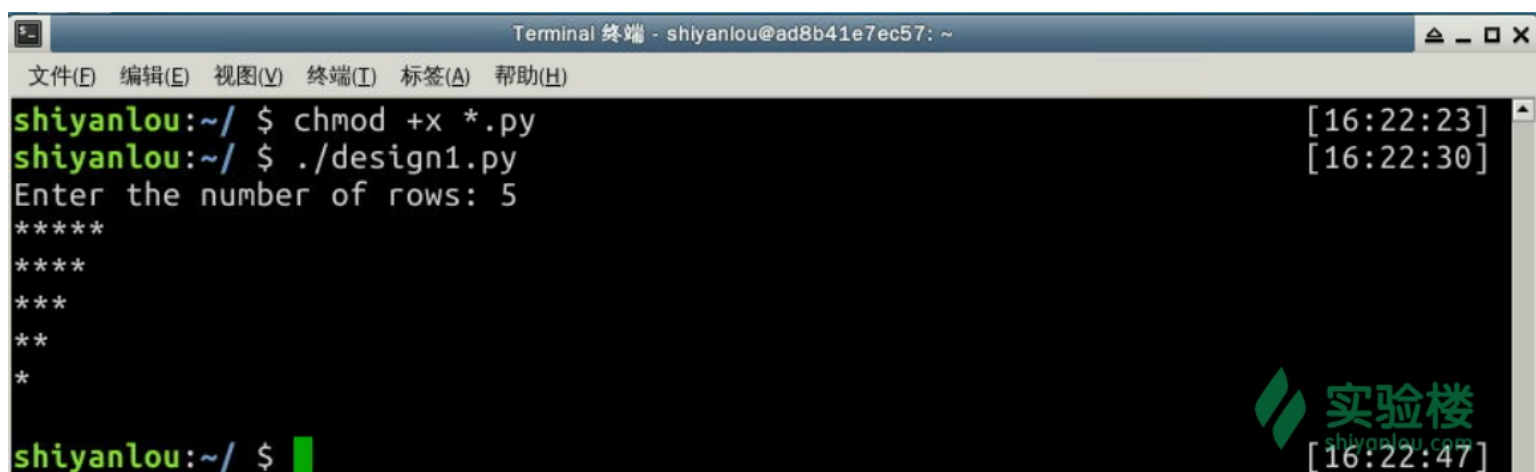
## 2.2 一些打印 \* 的例子

这里是一些你可以在大学的实验报告里经常看到的例子。

### 2.2.1. 设计 1

```
#!/usr/bin/env python3
row = int(input("Enter the number of rows: "))
n = row
while n >= 0:
    x = "*" * n
    print(x)
    n -= 1
```

运行这个程序：

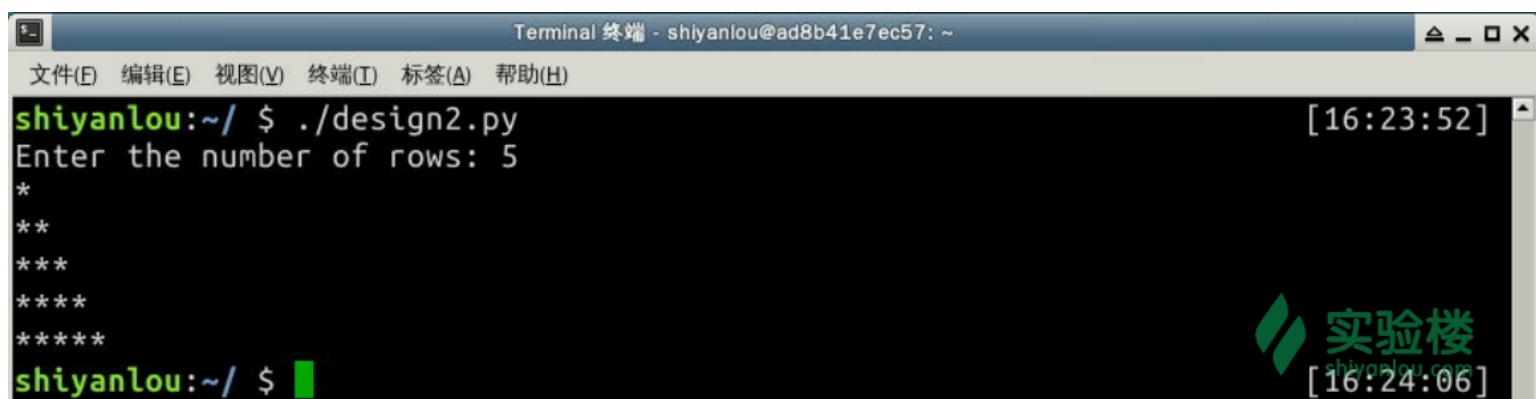


```
Terminal 终端 - shiyanlou@ad8b41e7ec57: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ chmod +x *.py
shiyanlou:~/ $ ./design1.py
Enter the number of rows: 5
*****
****
***
**
*
shiyanlou:~/ $
```

## 2.2.2. 设计 2

```
#!/usr/bin/env python3
n = int(input("Enter the number of rows: "))
i = 1
while i <= n:
    print("*" * i)
    i += 1
```

运行这个程序：

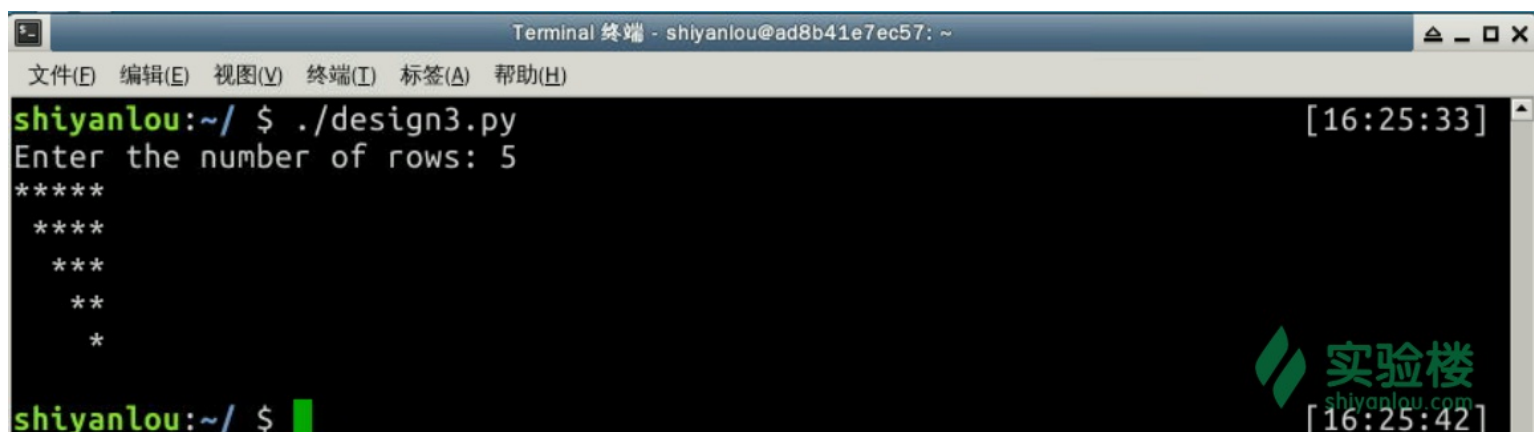


```
Terminal 终端 - shiyanlou@ad8b41e7ec57: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./design2.py
Enter the number of rows: 5
*
**
***
****
*****
shiyanlou:~/ $
```

## 2.2.3 设计 3

```
#!/usr/bin/env python3
row = int(input("Enter the number of rows: "))
n = row
while n >= 0:
    x = "*" * n
    y = " " * (row - n)
    print(y + x)
    n -= 1
```

运行这个程序：



### 3. 列表

在继续学习循环之前，我们先学习一个叫做列表的数据结构。它可以写作中括号之间的一系列逗号分隔的值。列表的元素不必是同一类型：

```
>>> a = [ 1, 342, 223, 'India', 'Fedora']
>>> a
[1, 342, 223, 'India', 'Fedora']
```

你可以将上面的列表想象为一堆有序的盒子，盒子包含有上面提到的值，每个盒子都有自己的编号（红色的数字），编号从零开始，你可以通过编号访问每一个盒子里面的值。对于列表，这里的编号称为索引。

0

1

2

3

4

1

342

223

India

Fedora



我们像下面这样通过索引来访问列表中的每一个值：

```
>>> a[0]
1
>>> a[4]
'Fedora'
```

如果我们使用负数的索引，那将会从列表的末尾开始计数，像下面这样：

```
>>> a[-1]
'Fedora'
```

你甚至可以把它切成不同的部分，这个操作称为切片，例子在下面给出：

```
>>> a[0:-1]
[1, 342, 223, 'India']
>>> a[2:-2]
[223]
```

切片并不会改变正在操作的列表，切片操作返回其子列表，这意味着下面的切片操作返回列表一个新的（浅）拷贝副本：

```
>>> a[:]
[1, 342, 223, 'India', 'Fedora']
```

切片的索引有非常有用的默认值；省略的第一个索引默认为零，省略的第二个索引默认为切片的字符串的大小：

```
>>> a[:-2]
[1, 342, 223]
>>> a[-2:]
['India', 'Fedora']
```

有个办法可以很容易地记住切片的工作方式：切片时的索引是在两个元素之间。左边第一个元素的索引为 0，而长度为 n 的列表其最后一个元素的右界索引为 n。例如：

|                                     |    |  |     |  |     |                    |
|-------------------------------------|----|--|-----|--|-----|--------------------|
| +---+-----+-----+-----+-----+-----+ |    |  |     |  |     |                    |
|                                     | 1  |  | 342 |  | 223 | 'India'   'Fedora' |
| +---+-----+-----+-----+-----+-----+ |    |  |     |  |     |                    |
| 0                                   | 1  |  | 2   |  | 3   |                    |
|                                     |    |  |     |  |     | 4                  |
|                                     |    |  |     |  |     | 5                  |
| -6                                  | -5 |  | -4  |  | -3  |                    |
|                                     |    |  |     |  |     | -2                 |
|                                     |    |  |     |  |     | -1                 |

上面的第一行数字给出列表中的索引点 0...6。第二行给出相应的负索引。切片是从 i 到 j 两个数值表示的边界之间的所有元素。

对于非负索引，如果上下都在边界内，切片长度就是两个索引之差。例如 `a[2:4]` 是 2。

试图使用太大的索引会导致错误：

```
>>> a[32]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> a[-10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Python 能够优雅地处理那些没有意义的切片索引：一个过大的索引值(即大于列表实际长度)将被列表实际长度所代替，当上边界比下边界大时(即切片左值大于右值)就返回空列表：

```
>>> a[2:32]
[223, 'India', 'Fedora']
>>> a[32:]
[]
```

切片操作还可以设置步长，就像下面这样：

```
>>> a[1::2]
[342, 'India']
```

它的意思是，从切片索引 1 到列表末尾，每隔两个元素取值。

列表也支持连接这样的操作，它返回一个新的列表：

```
>>> a + [36, 49, 64, 81, 100]
[1, 342, 223, 'India', 'Fedora', 36, 49, 64, 81, 100]
```

列表允许修改元素：

```
>>> cubes = [1, 8, 27, 65, 125]
>>> cubes[3] = 64
>>> cubes
[1, 8, 27, 64, 125]
```

也可以对切片赋值，此操作可以改变列表的尺寸，或清空它：

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # 替换某些值
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # 现在移除他们
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # 通过替换所有元素为空列表来清空这个列表
>>> letters[:] = []
>>> letters
[]
```

细心的同学可能发问了，前面不是说过切片操作不改变列表么？严格来说，这里并不算真正的切片操作，只是上面代码中赋值运算符左边的这种操作与切片操作形式一样而已。

要检查某个值是否存在于列表中，你可以这样做：

```
>>> a = ['ShiYanLou', 'is', 'cool']
>>> 'cool' in a
True
>>> 'Linux' in a
False
```

这意味着我们可以将上面的语句使用在 `if` 子句中的表达式。通过内建函数 `len()` 我们可以获得列表的长度：

```
>>> len(a)
3
```

如果你想要检查列表是否为空，请这样做：

```
if list_name: # 列表不为空
    pass
else: # 列表为空
    pass
```

列表是允许嵌套的（创建一个包含其它列表的列表），例如：

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

### 3. for 循环

通过 `for` 语句我们可以使用 `for` 循环。Python 里的 `for` 循环与 C 语言中的不同。这里的 `for` 循环遍历任何序列（比如列表和字符串）中的每一个元素。下面给出示例：

```
>>> a = ['ShiYanLou', 'is', 'powerful']
>>> for x in a:
...     print(x)
...
ShiYanLou
is
powerful
```

我们也能这样做：

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> for x in a[::2]:
...     print(x)
1
3
5
7
9
```

### 3.1. range() 函数

如果你需要一个数值序列，内置函数 `range()`

(<https://docs.python.org/3/library/stdtypes.html#range>) 会很方便，它生成一个等差数列（并不是列表）：

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> range(1, 5)
range(1, 5)
>>> list(range(1, 5))
[1, 2, 3, 4]
>>> list(range(1, 15, 3))
[1, 4, 7, 10, 13]
>>> list(range(4, 15, 2))
[4, 6, 8, 10, 12, 14]
```

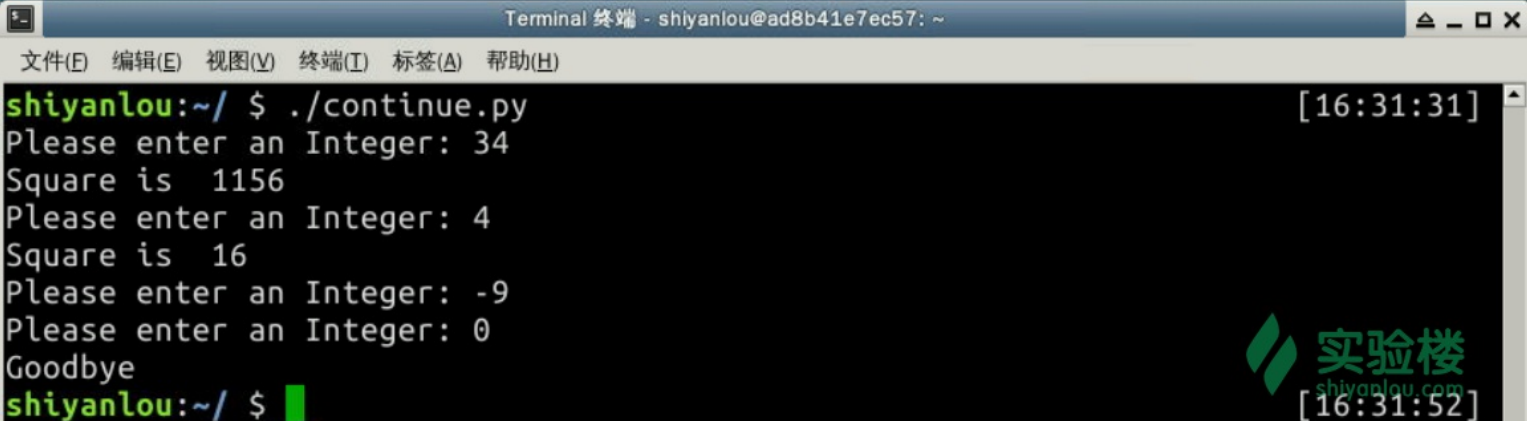
## 4. continue 语句

如同 `break`，我们可以在循环中使用另一个语句 `continue`。它会跳过其后的代码回到循环开始处执行。这意味着它可以帮助你跳过部分循环。在下面的例子中，我们要求用户输入一个整数，如果输入的是负数，那么我们会再次要求输

入，如果输入的是整数，我们计算这个数的平方。用户输入 0 来跳出这个无限循环。

```
#!/usr/bin/env python3
while True:
    n = int(input("Please enter an Integer: "))
    if n < 0:
        continue # 这会返回到循环开始处执行
    elif n == 0:
        break
    print("Square is ", n ** 2)
print("Goodbye")
```

运行程序：



```
Terminal 终端 - shiyanlou@ad8b41e7ec57: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./continue.py
Please enter an Integer: 34
Square is 1156
Please enter an Integer: 4
Square is 16
Please enter an Integer: -9
Please enter an Integer: 0
Goodbye
shiyanlou:~/ $
```

## 5. 循环的 else 语句

我们可以在循环后面使用可选的 else 语句。它将会在循环完毕后执行，除非有 break 语句终止了循环。

```
>>> for i in range(0, 5):
...     print(i)
... else:
...     print("Bye bye")
...
0
1
2
3
4
Bye bye
```

在本课程的后续内容中，我们会看到更多有关 `break` 和 `continue` 的例子。

## 6. 棍子游戏

这是一个非常简单的游戏。这里有 21 根棍子，首先用户选 1 到 4 根棍子，然后电脑选 1 到 4 根棍子。谁选到最后一根棍子谁就输。你知道哪种情况用户会赢吗？

特别说明：用户和电脑一次选的棍子总数只能是5。

```
#!/usr/bin/env python3
sticks = 21


print("There are 21 sticks, you can take 1-4 number of sticks at a time.")
print("Whoever will take the last stick will loose")

while True:
    print("Sticks left: " , sticks)
    sticks_taken = int(input("Take sticks(1-4):"))
    if sticks == 1:
        print("You took the last stick, you loose")
        break
    if sticks_taken >= 5 or sticks_taken <= 0:
        print("Wrong choice")
        continue
    print("Computer took: " , (5 - sticks_taken) , "\n")
    sticks -= 5
```

## 三、总结

这个实验中我们了解了两种循环：`while` 和 `for` 循环，其中的 `for` 循环我们通常与 `range()` 函数配合使用，要特别注意的是，`range()` 函数返回的并不是列表而是一种可迭代对象：

```
Terminal 终端 · python3
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyancelou:Desktop/ $ python3 [17:53:08]
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = range(10)
>>> a
range(0, 10)
>>> b = list(a)
>>> b
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```



python 中 for 循环的 else 子句给我们提供了检测循环是否顺利执行完毕的一种优雅方法。

*\*本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：控制流 If-else (</courses/596/labs/2039/document>)

下一节：数据结构 (</courses/596/labs/2041/document>)