第5章

Python入门

5.0 引言

对于树莓派来说，有许多编程语言可用，其中最流行的就是 Python 了。事实上，树莓派名字中的 Pi 就是受到单词 Python 的启发而取的。

本章提供了大量的示例代码，来帮助你掌握树莓派的编程技术。

5.1 在 Python 2 和 Python 3 之间做出选择

面临问题

你需要使用 Python，但是具体使用哪个版本却拿不定主意。

解决方案

两个版本都用。先使用 Python 3，直到遇到只有版本 2 才能完美解决的问题时，再回到 Python 2 的怀抱。

进一步探讨

虽然 Python 的最新的版本是 Python 3，并且该版本已经发布好几年了，但是许多人仍在坚守 Python 2。实际上，在 Raspbian 发行包中，两个版本都提供了：版本 2 名为 Python，而版本 3 则名为 Python 3。本书中的示例，除非特别指明，否则都是利用 Python 3 编写的。对于绝大部分示例代码来说，无需修改即可同时运行于 Python 2 和 Python 3 环境中。就 Python 社区而言，弃用旧版本的 Python 2 的阻力在于 Python 3 所引入的某些特性无法与版本 2 相兼容。也就是说，在为 Python 2 开发的巨量第三方库中，许多都无

法在 Python 3 下正常使用。

我的策略是在有可能的情况下，尽量使用 Python 3 编写代码，只有遇到兼容性问题的时候，万不得已才重新回到 Python 2 的怀抱。

参考资料

有关 Python 2 与 Python 3 的辩论文章，请参考 Python wiki（http://wiki.Python.org/moin/Python 2orPython）。

5.2 使用 IDLE 编辑 Python 程序

面临问题

你不知道应该使用什么来编写 Python 程序。

解决方案

在各种常见的树莓派发行包中，通常都同时提供了 Python 2 和 Python 3 两个版本的 IDLE Python 开发工具。如果你使用的是 Raspbian 的话，可以在 Start 菜单的 Programming 区找到名为 Python 2 和 Python 3 的两个版本的 IDLE 的快捷方式（见图 5-1）。

图 5-1 利用 IDLE 启动 Python

进一步探讨

IDLE 和 IDLE 3 外观看起来没什么区别，只是它们使用的 Python 的版本不同而已，所以不妨打开 IDLE 3（见图 5-2）。

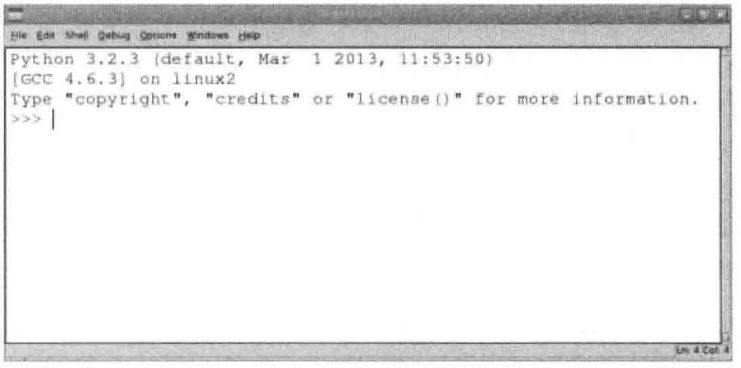

图 5-2 IDLE Python 控制台

打开的窗口名为 Python Shell。这是一个交互区域，你可以在此输入 Python 命令，然后马上就会看到返回的结果。所以，你可以尝试在该 shell 的>>>提示符后面输入下列命令。

```
>>> 2 + 2
4
>>>
```

Python 会计算表示式 2+2 的值，并返回结果 4。

Python Shell 非常适合随性的尝试，不过要想编写程序的话，你就需要使用一个编辑器了。为了使用 IDLE 打开一个新文件进行编辑，可以从 File 菜单中选择 New Window 选项（见图 5-3）。

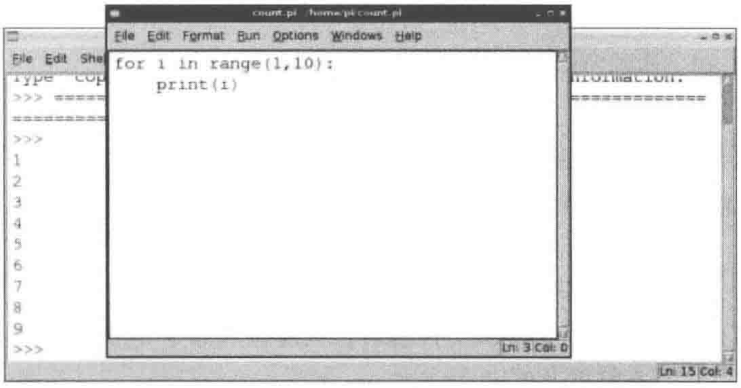

图 5-3 IDLE 编辑器

现在，你就可以在编辑器窗口中输入自己的程序代码了。作为一个尝试，你可以将下

列文本粘贴进该编辑器中，将文件保存为 count.py，然后，从 Run 菜单中选择 Run Module 选项。这样，你就可以从 Python 控制台中看到程序的运行结果了。

```
for i in range(1, 10):
    print(i)
```

作为一门编程语言，Python 的与众不同之处在于缩进是该语言的一个基本组成部分。许多基于 C 的编程语言都是利用{和}来界定代码块的，而 Python 则使用缩进级别来划分代码块。因此，对于上面的代码来说，Python 会将 print 作为循环的一部分而重复执行，因为它缩进了 4 个空格。

本书约定：使用 4 个空格作为一个缩进级别。

当你刚接触 Python 时，可能经常会遇到 IndentationError:unexpected indent 之类的错误提示，这些提示说明某些地方的缩进出现问题了。如果一切看起来都很正常的话，最好仔细检查各个缩进中是否存在制表符。要知道，Python 是会对制表符区别对待的。

此外，你还需注意观察 IDLE 是如何通过不同的颜色来高亮显示程序结构的。

参考资料

在本章中的许多示例都是通过 IDLE 来编辑 Python 代码的。

跟使用 IDLE 编辑运行 Python 文件类似，你也可以使用 nano 编辑文件（见 3.6 节），然后通过终端会话来运行它们（见 5.4 节）。

5.3 使用 Python 控制台

面临问题

你想要输入 Python 命令，而非编写完整的程序。

解决方案

你可以在 IDLE（见 5.2 节）或者终端会话中使用 Python 控制台。

进一步探讨

为了在终端窗口中启动一个 Python 2 控制台，只需输入命令 Python 即可；对于 Python 3 控制台，则需要输入命令 Python 3。

如果出现提示符>>>，则表明你可以输入 Python 命令了。如果你需要输入多行命令，那么控制台将会自动产生一个由 3 个点号指示的后续行。对于这些行来说，你仍然需要使用 4 个空格进行缩进，具体如下所示。

```
>>> for i in range(1, 10):
...   print(i)
...
1
2
3
4
5
6
7
8
9
>>>
```

在最后一条命令之后，你需要按两次回车键，控制台才能识别出这是缩进块的结束位置，继而运行这些代码。

此外，Python 控制台还提供了命令历史记录，你可以通过上下键来向前或向后选择运行过的命令。

参考资料

如果你想输入多行代码的话，最好还是使用 IDLE（见 5.2 节），将这些代码放到一个文件中进行编辑和运行。

5.4 利用终端运行 Python 程序

面临问题

虽然从 IDLE 内运行程序是个不错的方法，但是有时候你想从一个终端窗口来运行 Python 程序。

解决方案

在终端中使用 python 或者 python 3 命令，后面加上需要运行的程序文件的名称即可。

进一步探讨

为了从命令行运行一个 Python 2 程序，可以使用如下所示的代码。

```
$ python myprogram.py
```

如果你想要使用 Python 3 来运行程序的话，则需要将命令 Python 改为 Python 3。你要运行的 Python 程序应该位于扩展名为.py 的文件中。虽然你可以用普通用户的身份来运行大部分 Python 程序，但是，对于某些程序，特别是使用了 GPIO 端口的程序来说，你必须具有超级用户权限才能够运行。如果你的程序要求具有超级用户权限的话，可以在命令前面加上前缀 sudo。

```
$ sudo python myprogram.py
```

在前面的这些例子中，命令中必须包含 Python 才能运行程序，不过，你还可以在 Python 程序的开头部分添加一行内容，这样 Linux 就能够知道它是一个 Python 程序了。这一特殊行通常称为 shebang（hash-bang 的缩写）行，下面是由单行代码组成的一个示例程序，其中第一行就是 shebang 行。

```
#!/usr/bin/python
print("I'm a program, I can run all by myself")
```

为了能够从命令行直接运行这个示例程序，你必须首先通过下列命令赋予该文件写权限（请参考 3.13 节）。

```
$ chmod +x test.py
```

其中，参数+x 表示添加执行权限。

现在，你只需要单条命令就能够执行这个名为 test.py 的 Python 程序了。

```
$ ./test.py
I'm a program, I can run all by myself
$
```

注意，对于上面这条命令来说，最前边的./是搜索该文件所必需的。

参考资料

在 3.23 节中介绍了如何通过定时事件来运行一个 Python 程序的方法。

要想在引导期间自动运行程序，请参考 3.22 节。

5.5 变量

面临问题

你想要给一个值取名。

解决方案

利用=给一个值指定名称。

进一步探讨

对于 Python 来说，你无需声明变量的类型，直接给它赋值即可，具体如下所示。

```
a = 123
b = 12.34
c = "Hello"
d = 'Hello'
```

```
e = True
```

你可以使用单引号或双引号来定义字符串常量。在 Python 中，逻辑常量是 True 和 False，这里是大小写敏感的。

按照惯例，变量名以小写字母开头，如果其中包含多个单词的话，可以使用下画线将它们连接起来。给变量取一个描述性的名称，永远都是一个好主意。

像 x、total 和 number_of_chars，都是合法的变量名称。

参考资料

赋予变量的值，也可以是列表（见 6.1 节）或者字典（见 6.12 节）。

关于变量算术运算的介绍，详情请参考 5.8 节。

5.6 显示输出结果

面临问题

你想要查看变量的值。

解决方案

使用 print 命令。你可以在 Python 控制台（见 5.3 节）中尝试下面的示例代码。

```
>>> x = 10
>>> print(x)
10
>>>
```

进一步探讨

对于 Python 2 来说，你可以使用不带括号的 print 命令。但是，对于 Python 3 来说，这是不允许的，所以为了兼容这两个版本起见，请在待输出值的两边加上括号。

参考资料

要想读取用户的输入，请参考 5.7 节。

5.7 读取用户输入

面临问题

你想提示用户输入一个值。

解决方案

使用 input（Python 3）或者 raw_input（Python 2）命令。你可以在 Python 3 控制台（见 5.3 节）下面尝试下列代码。

```
>>> x = input("Enter Value:")
Enter Value:23
>>> print(x)
23
>>>
```

进一步探讨

使用 Python 2 时，必须将上面例子中的 input 替换为 raw_input。

实际上，Python 2 也有一个 input 函数，不过它的作用是验证输入，并试图将其转换为适当类型的 Python 值，而 raw_input 的作用跟 Python 3 中的 input 函数完全一致，都是返回一个字符串。

参考资料

关于 Python 2 的 input 函数的更多内容，请参考 http://docs.python.org/2/library/functions.html#input。

5.8 算术运算

面临问题

你想使用 Python 进行算术运算。

解决方案

使用运算符+、-、*和/。

进一步探讨

对于算术运算来说，最常用的运算符是+、-、*和/，分别对应于加法、减法、乘法和除法运算。

此外，你还可以像下例那样通过小括号对表达式的各个部分进行分组，在这个示例中，给出一个摄氏温度，它会把它转换为华氏度数。

```
>>> tempC = input("Enter temp in C: ")
Enter temp in C: 20
>>> tempF = (int(tempC) * 9) / 5 + 32
```

```
>>> print(tempF)
68.0
>>>
```

其他算术运算符还包括%（取模运算）和**（幂运算）。举例来说，为了求 2 的 8 次方，可以使用下列运算表达式。

```
>>> 2 ** 8
256
```

参考资料

关于 input 命令的用法，请参考 5.7 节；将输入的字符串转换为数字的方法，请参考 5.12 节。

此外，Math 库提供了许多常用的数学函数，你可以直接拿来使用，该库的地址为 http://docs.python.org/3.0/library/math.html。

5.9 创建字符串

面临问题

你需要创建字符串变量。

解决方案

要想创建一个新的字符串，可以使用赋值运算符和字符串常量。你可以使用单引号或双引号来括住字符串，但是前后要匹配。

举例来说：

```
>>> s = "abc def"
>>> print(s)
abc def
>>>
```

进一步探讨

如果字符串本身包含单引号或双引号的话，那么可以选择字符串内没有用到的那种类型的引号来括住这个字符串。例如：

```
>>> s = "Isn't it warm?"
>>> print(s)
Isn't it warm?
>>>
```

有时候，字符串内部会包含一些特殊字符，如制表符和换行符等。这时候，可以使用转义字符来满足这一需求。为了包含一个制表符，可以使用\t，对于换行符，可以使用

\n，具体如下例所示。

```
>>> s = "name\tage\nMatt\t14"
>>> print(s)
name    age
Matt    14
>>>
```

参考资料

要想了解所有的转义字符，请参考 Python 参考手册（http://bit.ly/17Xxuqf）。

5.10 连接（合并）字符串

面临问题

你想把多个字符串合并到一起。

解决方案

使用+（连结）运算符。

举例来说：

```
>>> s1 = "abc"
>>> s2 = "def"
>>> s = s1 + s2
>>> print(s)
abcdef
>>>
```

进一步探讨

对于许多编程语言来说，你可以连接一串的值，这些值中既可以有字符串类型，也可以有其他类型，比如数值类型；这种情况下，连接期间会自动将数字转换为字符串。但是，对于 Python 来说情况有所不同，如果你尝试输入下列命令的话，会得到一个错误。

```
>>> "abc" + 23
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

在连接它们之前，你需要将待连接的各个部分都转换为一个字符串，具体如下例所示。

```
>>> "abc" + str(23)
'abc23'
>>>
```

参考资料

至于利用 str 函数将数字转换为字符串的具体方法，请参考 5.11 节。

5.11 将数字转换为字符串

面临问题

你想把一个数字转换为一个字符串。

解决方案

你可以使用 Python 提供的 str 函数，具体如下所示。

```
>>> str(123)
'123'
>>>
```

进一步探讨

之所以要把数字转换为字符串，通常是因为只有如此，你才能够将它与别的字符串合并到一起（见 5.10 节）。

参考资料

关于将字符串转换为数字的逆运算，请参考 5.12 节。

5.12 将字符串转换为数字

面临问题

你想把一个字符串转换为一个数字。

解决方案

你可以使用 Python 提供的 int 或者 float 函数。

举例来说，为了将字符串-123 变成一个数字，可以使用下列代码。

```
>>> int("-123")
-123
>>>
```

这个函数可以处理所有整数，无论符号是正，还是负。

要想把一个字符串转换为一个浮点数，那就不能使用 int 函数了，而应该使用 float 函数。

```
>>> float("00123.45")
123.45
>>>
```

进一步探讨

无论 int 函数，还是 float 函数，它们不仅都能正确处理数字前面部分的前导零，而且还能够容忍数字前后多余的空格符以及其他空白字符。

此外，int 函数还可以将一个表示非十进制数字的字符串转换为数字，为此，只需将基数作为第二个参数传递给这个函数即可。

在下面的例子中，会将表示二进制数字 1001 的字符串转换为一个数字。

```
>>> int("1001", 2)
9
>>>
```

在下面第二个示例中，会将十六进制数 AFF0 转换为一个整数。

```
>>> int("AFF0", 16)
45040
>>>
```

参考资料

关于将数字转换为字符串的逆运算，请参考 5.11 节。

5.13 确定字符串的长度

面临问题

你想知道字符串中包含了多少个字符。

解决方案

你可以使用 Python 提供的 len 函数。

进一步探讨

举例来说，为了确定字符串 abcdef 的长度，你可以使用下列代码。

```
>>> len("abcdef")
6
>>>
```

参考资料

此外，len 命令还可以用于列表（见 6.3 节）。

5.14 确定某字符串在另一个字符串中的位置

面临问题

你需要获得一个字符串在另一个字符串中的位置。

解决方案

你可以使用 Python 提供的 find 函数。

举例来说，为了确定字符串 def 在字符串 abcdef ghi 中的位置，你可以使用下列代码。

```
>>> s = "abcdefghi"
>>> s.find("def")
3
>>>
```

请注意，字符串的位置是从 0 开始计数的，也就是说，位置 3 表示的是字符串中的第 4 个字符。

进一步探讨

如果待查找的字符串实际上并不存在于正在搜索的字符串中的话，那么 find 函数的返回值为-1。

参考资料

函数 replace 可以用来查找并替换字符串中所有的匹配项（见 5.16 节）。

5.15 截取部分字符串

面临问题

你想截取特定字符之间的部分字符串。

解决方案

你可以使用 Python 的[:]表达式。

举例来说，如果你想截取字符串 abcdefghi 中介于第 2 个字符到第 5 个字符之间的部分的话，可以使用下列代码。

```
>>> s = "abcdefghi"
>>> s[1:5]
'bcde'
>>>
```

需要注意的是字符的位置是从 0 开始计数的，所以，位置 1 表示字符串中的第 2 个字符，位置 5 表示字符串中的第 6 个字符，不过，由于位置的范围并不包含最右边的数字，所以这里并不包括字母 f。

进一步探讨

实际上，[:]的作用是非常强大的，其中的两个参数，任何一个都可以忽略掉，这时候字符串的开始和结束位置会视情况而定。举例来说:

```
>>> s[:5]
'abcde'
>>>
```

同时：

```
>>> s = "abcdefghi"
>>> s[3:]
'defghi'
>>>
```

同时，你还可以使用负数来表示从字符串末尾开始反向计数。这种表示方法在某些情况下是非常有用的，比如当你想要获取某个文件名后面 3 个字母组成的扩展名的时候，具体如下例所示。

```
>>> "myfile.txt"[-3:]
'txt'
```

参考资料

与本节介绍的分割字符串不同，在 5.10 节中，我们介绍了将字符串合并到一起的方法。

在 6.10 节中也会用到同样的语法，不过它不是针对字符串，而是针对列表的。

5.16 使用字符串替换另一个字符串中的内容

面临问题

你想利用一个字符串替换另一个字符串中的所有匹配项。

解决方案

你可以使用 replace 函数。

举例来说，为了利用 times 替换所有的 X，可以使用如下所示的代码。

```
>>> s = "It was the best of X. It was the worst of X"
>>> s.replace("X", "times")
'It was the best of times. It was the worst of times'
>>>
```

进一步探讨

待查找的字符串必须是严格匹配的，也就是说，不仅对大小写敏感，同时还要考虑空格。

参考资料

如果只需查找字符串，而无需进行替换的话，请参考 5.14 节。

5.17 字符串的大小写转换

面临问题

你想把字符串中的所有字符全部转换成大写字母或小写字母。

解决方案

你可以根据情况使用 upper 函数或者 lower 函数。

举例来说，为了将 aBcDe 转换为大写字母，可以使用如下所示的代码。

```
>>> "aBcDe".upper()
'ABCDE'
>>>
```

为了将这个字符串转换为小写字母，你可以使用下列代码。

```
>>> "aBcDe".lower()
'abcde'
>>>
```

进一步探讨

就像大部分处理字符串函数所做的那样，upper 和 lower 函数也不会实际修改字符串，而是返回一个修改后的字符串副本。

举例来说，下面的示例代码将会返回字符串 s 的副本，需要注意的是原始字符串本身并没有发生任何变化。

```
>>> s = "aBcDe"
>>> s.upper()
```

```
'ABCDE'
>>> s
'aBcDe'
>>>
```

如果你想把 s 的值全部转换为大写的话，可以使用如下所示的代码。

```
>>> s = "aBcDe"
>>> s = s.upper()
>>> s
'ABCDE'
>>>
```

参考资料

关于如何替换字符串中的文本的方法，请参考 5.16 节。

5.18 根据条件运行命令

面临问题

你希望只有当某些条件成立时才运行某些 Python 命令。

解决方案

你可以使用 Python 的 if 命令。

在下面的例子中，只有当 x 取值大于 100 的时候，才会输出消息 xis big。

```
>>> x = 101
>>> if x > 100:
...     print("x is big")
...
x is big
```

进一步探讨

在关键字 if 后面是一个条件表达式。这个条件表达式通常无非就是比较两个值，并返回 True 或者 False。如果返回值为 True，那么，后面缩进的各行代码就会被执行。

通常情况下，人们都是希望在条件为 True 的情况下执行某些操作，而在条件为 False 的情况下执行另外一些操作。如果遇到这种情况的话，可以将 else 命令与 if 命令联合起来使用，具体如下例所示。

```
x = 101
if x > 100:
    print("x is big")
else:
```

```
        print("x is small")

    print("This will always print")
```

你还可以利用一长串的 elif 语句将一组条件连接到一起。只要其中任何一个条件得到满足的话，就会执行对应的代码块，并且不再判断后续的条件分支。举例来说：

```
x = 90
if x > 100:
    print("x is big")
elif x < 10:
    print("x is small")
else:
    print("x is medium")
```

这段代码的输出结果是 x is medium。

参考资料

关于不同类型比较操作的详细信息，请参考 5.19 节。

5.19 值的比较

面临问题

你想对值进行比较。

解决方案

你可以使用下列比较运算符：<、>、<=、>=、==或者!=。

进一步探讨

在 5.18 节中，已经用过<（小于）和>（大于）运算符了。下面，我们将列出所有的比较运算符。

< 小于

> 大于

<= 小于等于

>=大于等于

==等于

!=不等于

虽然有些人更喜欢使用<>，而非!=，但是两者的作用是完全一样的。

你可以在 Python 控制台（见 5.3 节）下面测试这些命令，如下所示。

```
>>> 1 != 2
True
>>> 1 != 1
False
>>> 10 >= 10
True
>>> 10 >= 11
False
>>> 10 == 10
True
>>>
```

在进行比较运算时，一个常见的错误是使用了=（赋值），而非==（双等号）。这个错误通常比较隐蔽，因为如果比较运算的一边是个变量的话，那么这是完全合法的并且会被正确执行的，但是，却无法得到预期的结果。

就如同比较数字一样，你也可以使用这些操作符来比较字符串，具体用法如下所示。

```
>>> 'aa' < 'ab'
True
>>> 'aaa' < 'aa'
False
```

对字符串进行比较是按照字典方式进行的，即按照它们在字典中出现的顺序进行比较。

当然，这种方式并非十分完美，因为对于每个字母来说，通常认为大写字母要小于小写字母。每个字母都有一个值，即它的 ASCII 编码（https://en.wikipedia.org/wiki/ASCII）。

参考资料

请同时参考 5.18 节和 5.20 节的内容。

5.20 逻辑运算符

面临问题

你需要使用 if 语句描述一个复杂的条件。

解决方案

使用下列逻辑运算符：and、or 和 not。

进一步探讨

举例来说，你想要检查变量 x 的值是否介于 10 ~ 20 之间。为此，你可以使用 and 运算符。

```
>>> x = 17
```

```
>>> if x >= 10 and x <= 20:
...     print('x is in the middle')
...
x is in the middle
```

如果需要的话，你可以将任意多个 and 和 or 语句组合起来使用，同时，你还可以利用括号对复杂的表达式进行分组。

参考资料

请同时参考 5.18 节和 5.19 节。

5.21 将指令重复执行特定次数

面临问题

你需要将某些程序代码重复执行特定的次数。

解决方案

你可以使用 Python 的 for 命令，并指定迭代次数。

举例来说，如果要将一个命令重复执行 10 次，可以使用如下所示的代码。

```
>>> for i in range(1, 11):
...     print(i)
...
1
2
3
4
5
6
7
8
9
10
>>>
```

进一步探讨

这里，range 命令的第二个参数并不包括在范围之内，也就是说，如果要想计算到 10 的话，该参数的值必须指定为 11。

参考资料

对于终止循环的条件要远比这里重复执行特定次数更为复杂的情况，请参考 5.22 节。

如果你想对列表或字典逐元素执行某些命令的话，请分别参考 6.7 节或 6.15 节。

5.22 重复执行指令直到特定条件改变为止

面临问题

你需要重复执行某些程序代码，直到某些条件发生变化为止。

解决方案

你可以使用 Python 的 while 语句。该语句会重复执行嵌套在其内部的语句，直到它的条件不再成立为止。在下面的例子中，while 语句内嵌的代码会一直循环执行下去，直到用户输入 X 后才会退出循环。

```
>>> answer = ''
>>> while answer != 'X':
...     answer = input('Enter command:')
...
Enter command:A
Enter command:B
Enter command:X
>>>
```

进一步探讨

需要注意的是前面的示例中所用的 input 命令只适用于 Python 3。如果要在 Python 2 中运行该代码的话，可以使用 raw_input 函数替换掉 input 函数。

参考资料

如果你只是想让某些命令执行特定的次数的话，可以参考 5.21 节。

如果你想对列表或字典逐元素执行某些命令的话，那么请分别参考 6.7 节或 6.15 节。

5.23 跳出循环语句

面临问题

在执行循环语句的时候，如果遇到某些情况，你将需要退出循环过程。

解决方案

你可以使用 Python 的 break 语句来退出 while 循环或 for 循环。

下面例子中的代码的行为与 5.22 节中的完全一致。

```
>>> while True:
...     answer = input('Enter command:')
```

```
...     if answer == 'X':
...             break
...
Enter command:A
Enter command:B
Enter command:X
>>>
```

进一步探讨

需要注意的是这个示例中所用的 input 命令只适用于 Python 3。如果要在 Python 2 中运行该代码的话，可以使用 raw_input 函数替换掉 input 函数。这个示例代码的行为与 5.22 节中的代码的行为完全一致。不过，就本例而言，while 循环的条件为 True，所以该循环将一直进行下去，直到用户输入 X 后，才会通过 break 语句退出循环。

参考资料

此外，你还可以通过 while 语句本身的条件来退出循环，具体请参考 5.18 节。

5.24 定义 Python 函数

面临问题

你想避免在程序中一遍又一遍地重复同样的代码。

解决方案

创建一个函数，将多行代码组织在一起，以便可以从多个不同的地方来调用它。

下面的示例代码展示了如何在 Python 中创建和调用函数。

```
def count_to_10():
    for i in range(1, 11):
        print(i)

count_to_10()
```

在这个例子中，我们利用 def 命令定义了一个函数，每当调用该函数时，它会打印输出一个介于 1 到 10 之间的数字。

```
count_to_10()
```

进一步探讨

函数的命名惯例与变量完全一致，具体请参考 5.5 节。也就是说，函数名应该以小写字母开头，如果名称由多个单词组成的话，可以利用下画线来分隔单词。

上面的示例函数不是太灵活，因为它只能计算到 10。如果想要提高该函数的灵活性以便可以计算到任意数字的话，可以在函数中添加一个参数来表示最大值，具体如下所示。

```
def count_to_n(n):
    for i in range(1, n + 1):
        print(i)

count_to_n(5)
```

参数名 n 需要放到小括号中，之后，range 命令会用到这个参数，但不是直接使用它，而是在加 1 之后。

使用一个参数来表示想要计数到的数字的时候，如果通常要计数到 10，而有时候又要计数到其他数字的话，那么你就不得不每次都规定一个数字。实际上，你可以为参数指定一个默认值，这样就能够两者兼顾了，具体如下所示。

```
def count_to_n(n=10):
    for i in range(1, n + 1):
        print(i)

count_to_n()
```

这样的话，每次调用该函数时，这个参数的值都是 10，除非指定了其他数字。

如果你的函数需要多个参数的话，比如需要计算两个数字之间的数字个数的话，那么可以使用逗号来分隔参数。

```
def count(from_num=1, to_num=10):
    for i in range(from_num, to_num + 1):
                print(i)

count()
count(5)
count(5, 10)
```

上面这些示例代码中的所有函数都没有返回任何值，它们只是进行了某些计算。

如果你想让函数返回值的话，则需要使用 return 命令。

下列函数以字符串作为其参数，并在该字符串后面追加单词 please。

```
def make_polite(sentence):
    return sentence + " please"

print(make_polite("Pass the cheese"))
```

当函数返回值时，你可以将结果赋值给一个变量，或者像本例中那样，直接打印出结果。

参考资料

如果想要从一个函数中返回多个值的话，可以参见 7.3 节。