函数

1. 函数形参

函数取得的参数是你提供给函数的值，这样函数就可以利用这些值 做 一些事情。这些参数就像变量一样，只不过它们的值是在我们调用函数的时候定义的，而非在函数本身内赋值。

参数在函数定义的圆括号对内指定，用逗号分割。当我们调用函数的时候，我们以同样的方式提供值。注意我们使用过的术语——函数中的参数名称为 形参 而你提供给函数调用的值称为 实参 。

```
#!/usr/bin/python
# Filename: func_param.py

def printMax(a, b):
    if a &gt; b:
        print a, 'is maximum'
    else:
        print b, 'is maximum'

printMax(3, 4) # directly give literal values

x = 5
y = 7

printMax(x, y) # give variables as arguments
```

输出

```
$ python func_param.py
4 is maximum
7 is maximum
```

它如何工作

这里，我们定义了一个称为 `printMax` 的函数，这个函数需要两个形参，叫做 `a` 和 `b`。我们使用 `if..else` 语句找出两者之中较大的一个数，并且打印较大的那个数。
在第一个 `printMax` 使用中，我们直接把数，即实参，提供给函数。在第二个使用中，我们使用变量调用函数。`printMax(x, y)`使实参 `x` 的值赋给形参 `a`，实参 `y` 的值赋给形参 `b`。在两次调用中，printMax 函数的工作完全相同。

2.局部变量

当你在函数定义内声明变量的时候，它们与函数外具有相同名称的其他变量没有任何关系，即变量名称对于函数来说是 局部 的。这称为变量的 作用域 。所有变量的作用域是它们被定义的块，从它们的名称被定义的那点开始。

```
#!/usr/bin/python
# Filename: func_local.py

def func(x):
    print 'x is', x
    x = 2
    print 'Changed local x to', x

x = 50
func(x)
print 'x is still', x
```

输出

```
$ python func_local.py
x is 50
Changed local x to 2
x is still 50
```

它如何工作

在函数中，我们第一次使用 `x` 的 值 的时候，Python 使用函数声明的形参的值。
接下来，我们把值 `2` 赋给 `x`。`x` 是函数的局部变量。所以，当我们在函数内改变 `x` 的值的时候，在主块中定义的 `x` 不受影响。
在最后一个 `print` 语句中，我们证明了主块中的 `x` 的值确实没有受到影响。
如果你想要为一个定义在函数外的变量赋值，那么你就得告诉 Python 这个变量名不是局部的，而是 全局 的。我们使用 `global` 语句完成这一功能。没有 `global` 语句，是不可能为定义在函数外的变量赋值的。
你可以使用定义在函数外的变量的值（假设在函数内没有同名的变量）。然而，我并不鼓励你这样做，并且你应该尽量避免这样做，因为这使得程序的读者会不清楚这个变量是在哪里定义的。使用 `global` 语句可以清楚地表明变量是在外面的块定义的。

```
#!/usr/bin/python
# Filename: func_global.py

def func():
    global x

    print 'x is', x
```

```
    x = 2
    print 'Changed local x to', x

x = 50
func()
print 'Value of x is', x
```

输出

```
$ python func_global.py
x is 50
Changed global x to 2
Value of x is 2
```

它如何工作

`global`语句被用来声明`x`是全局的——因此，当我们在函数内把值赋给`x`的时候，这个变化也反映在我们在主块中使用`x`的值的时候。
你可以使用同一个`global`语句指定多个全局变量。例如`global x, y, z`。

3.默认参数值

对于一些函数，你可能希望它的一些参数是 可选 的，如果用户不想要为这些参数提供值的话，这些参数就使用默认值。这个功能借助于默认参数值完成。你可以在函数定义的形参名后加上赋值运算符（=）和默认值，从而给形参指定默认参数值。

注意，默认参数值应该是一个参数。更加准确的说，默认参数值应该是不可变的——这会在后面的章节中做详细解释。从现在开始，请记住这一点。

```
#!/usr/bin/python
# Filename: func_default.py

def say(message, times = 1):
    print message * times

say('Hello')
say('World', 5)
```

输出

```
$ python func_default.py
Hello
WorldWorldWorldWorldWorld
```

它如何工作

名为 `say` 的函数用来打印一个字符串任意所需的次数。如果我们不提供一个值，那么默认地，字符串将只被打印一遍。我们通过给形参 `times` 指定默认参数值 `1` 来实现这一功能。
在第一次使用 `say` 的时候，我们只提供一个字符串，函数只打印一次字符串。在第二次使用 `say` 的时候，我们提供了字符串和参数 `5`，表明我们想要 说 这个字符串消息 5 遍。
重要 只有在形参表末尾的那些参数可以有默认参数值，即你不能在声明函数形参的时候，先声明有默认值的形参而后声明没有默认值的形参。 这是因为赋给形参的值是根据位置而赋值的。例如，`def func(a, b=5)`是有效的，但是 `def func(a=5, b)`是 无效的。

4.关键参数

如果你的某个函数有许多参数，而你只想指定其中的一部分，那么你可以通过命名来为这些参数赋值——这被称作 关键参数 ——我们使用名字（关键字）而不是位置（我们前面所一直使用的方法）来给函数指定实参。

这样做有两个 优势 ——一，由于我们不必担心参数的顺序，使用函数变得更加简单了。二、假设其他参数都有默认值，我们可以只给我们想要的那些参数赋值。

```
#!/usr/bin/python
# Filename: func_key.py

def func(a, b=5, c=10):
    print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

输出

```
$ python func_key.py
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

它如何工作

名为 `func` 的函数有一个没有默认值的参数，和两个有默认值的参数。
在第一次使用函数的时候， `func(3, 7)`，参数 `a` 得到值 `3`，参数 `b` 得到值 `7`，而参数 `c` 使用默认值 `10`。

在第二次使用函数 `func(25, c=24)`的时候，根据实参的位置变量 `a` 得到值 `25`。根据命名，即关键参数，参数 `c` 得到值 `24`。变量 `b` 根据默认值，为 `5`。
在第三次使用 `func(c=50, a=100)`的时候，我们使用关键参数来完全指定参数值。注意，尽管函数定义中，`a` 在 `c` 之前定义，我们仍然可以在 `a` 之前指定参数 `c` 的值。

5.return 语句

`return` 语句用来从一个函数 返回 即跳出函数。我们也可选从函数 返回一个值 。

```
#!/usr/bin/python
# Filename: func_return.py

def maximum(x, y):
    if x &gt; y:
        return x
    else:
        return y

print maximum(2, 3)
```

输出

```
$ python func_return.py
3
```

它如何工作

`maximum` 函数返回参数中的最大值，在这里是提供给函数的数。它使用简单的 `if..else` 语句来找出较大的值，然后 返回 那个值。
注意，没有返回值的 `return` 语句等价于 `return None`。`None` 是 Python 中表示没有任何东西的特殊类型。例如，如果一个变量的值为 `None`，可以表示它没有值。除非你提供你自己的 `return` 语句，每个函数都在结尾暗含有 `return None` 语句。通过运行 `print someFunction()`，你可以明白这一点，函数 `someFunction` 没有使用 `return` 语句，如同：

```
def someFunction():
    pass
```

`pass` 语句在 Python 中表示一个空的语句块。

6.DocStrings

Python 有一个很奇妙的特性，称为 文档字符串 ，它通常被简称为 docstrings 。DocStrings 是一个重要的工具，由于它帮助你的程序文档更加简单易懂，你应该尽量使用它。你甚至可以在程序运行的时候，从函数恢复文档字符串！

```
#!/usr/bin/python
# Filename: func_doc.py

def printMax(x, y):
    '''Prints the maximum of two numbers.

    The two values must be integers.'''
    x = int(x) # convert to integers, if possible
    y = int(y)

    if x &gt; y:
        print x, 'is maximum'
    else:
        print y, 'is maximum'

printMax(3, 5)
print printMax.__doc__
```

输出

```
$ python func_doc.py
5 is maximum
Prints the maximum of two numbers.

       The two values must be integers.
```

它如何工作

在函数的第一个逻辑行的字符串是这个函数的 文档字符串 。注意，DocStrings 也适用于模块和类，我们会在后面相应的章节学习它们。

文档字符串的惯例是一个多行字符串，它的首行以大写字母开始，句号结尾。第二行是空行，从第三行开始是详细的描述。 强烈建议 你在你的函数中使用文档字符串时遵循这个惯例。

你可以使用`__doc__`（注意双下划线）调用`printMax`函数的文档字符串属性（属于函数的名称）。请记住 Python 把 每一样东西 都作为对象，包括这个函数。我们会在后面的类一章学习更多关于对象的知识。
如果你已经在 Python 中使用过`help()`，那么你已经看到过 DocStings 的使用了！它所做的只是抓取函数的`__doc__`属性，然后整洁地展示给你。你可以对上面这个函数尝试一下——只是在你的程序中包括`help(printMax)`。记住按 **q** 退出`help`。

自动化工具也可以以同样的方式从你的程序中提取文档。因此，我 强烈建议你对你所写的任何正式函数编写文档字符串。随你的 Python 发行版附带的 **pydoc** 命令，与 `help()`类似地使用 DocStrings。