# ch8函数



# 传参

## 关键字实参

```
def describe_pet(animal_type, pet_name):
    """显示宠物的信息"""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")
describe_pet(animal_type='hamster', pet_name='harry')

#I have a hamster.
#My hamster's name is Harry.
```

### 默认值

```
def describe_pet(pet_name, animal_type='dog'):
    """显示宠物的信息"""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")
describe_pet(pet_name='willie')

#I have a dog.
#My dog's name is Willie.
```

使用这个函数的最简单的方式是，在函数调用中只提供小狗的名字：

```
describe_pet('willie')
```

如果要描述的动物不是小狗，可使用类似于下面的函数调用：

```
describe_pet(pet_name='harry', animal_type='hamster')
```

也可以这样,注意顺序：

```
describe_pet('miao','cat')
```

注意　使用默认值时，在形参列表中必须先列出没有默认值的形参，再列出有默认值的实参。这让Python依然能够正确地解读位置实参。

### 禁止函数修改列表

要将列表的副本传递给函数，可以像下面这样做：

```
function_name(list_name[:])
```

虽然向函数传递列表的副本可保留原始列表的内容，但除非有充分的理由需要传递副本，否则还是应该将原始列表传递给函数，因为让函数使用现成列表可避免花时间和内存创建副本，从而提高效率，在处理大型列表时尤其如此。

## 数量不固定的实参

### 传递任意数量的实参

下面的函数只有一个形参*toppings ，但不管调用语句提供了多少实参，这个形参都将它们统统收入囊中：

```
def make_pizza(*toppings):
    """打印顾客点的所有配料"""
    print(toppings)
make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

形参名*toppings 中的**星号让**Python**创建一个名为**toppings**的空元组**，并将收到的所有值都封装到这个元组中。
函数体内的print 语句通过生成输出来证明Python能够处理使用一个值调用函数的情形，也能处理使用三个值来调用函数的情形。它以类似的方式处理不同的调用
注意，Python将实参封装到一个元组中，即便函数只收到一个值也如

可以循环

```
def make_pizza(*toppings):
    """概述要制作的比萨"""
    print("\nMaking a pizza with the following toppings:")
    for topping in toppings:
    print("- " + topping)
make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

### 使用任意数量的关键字实参

```
def build_profile(first, last, **user_info):
    """创建一个字典，其中包含我们知道的有关用户的一切"""
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
        profile[key] = value
    return profile

user_profile = build_profile('albert', 'einstein',location='princeton',field='physics')
print(user_profile)
```

# 将函数存储在模块中

模块 是扩展名为.py的文件，包含要导入到程序中的代码

import `pizza` pizza是文件/模块名

from `module_nam`e import `function_name`

from `module_name` import `function_0, function_1, function_2`

from `pizza` import `make_pizza`

from `pizza` import `make_pizza as mp`
from `module_name` import `function_name as fn`

import `pizza as p`
import `module_name as mn`

import 语句中的星号让Python将模块pizza 中的每个函数都复制到这个程序文件中。由于导入了每个函数，可通过名称来调用每个函数，而无需使用句点表示法。
然而，使用并非自己编写的大型模块时，最好不要采用这种导入方法：
如果模块中有函数的名称与你的项目中使用的名称相同，可能导致意想不到的结果：Python可能遇到多个名称相同的函数或变量，进而覆盖函数，而不是分别导入所有的函数。
最佳的做法是，要么只导入你需要使用的函数，要么导入整个模块并使用句点表示法。这能让代码更清晰，更容易阅读和理解。这里之所以介绍这种导入方法，只是想让你在阅读别人编写的代码时，如果遇到类似于下面的import 语句，能够理解它们：

```
from module_name import *
```

注意:
给形参指定默认值时，等号两边不要有空格：

```
def function_name(parameter_0, parameter_1='default value')
```

对于函数调用中的关键字实参，也应遵循这种约定：

```
function_name(value_0, parameter_1='value')
```