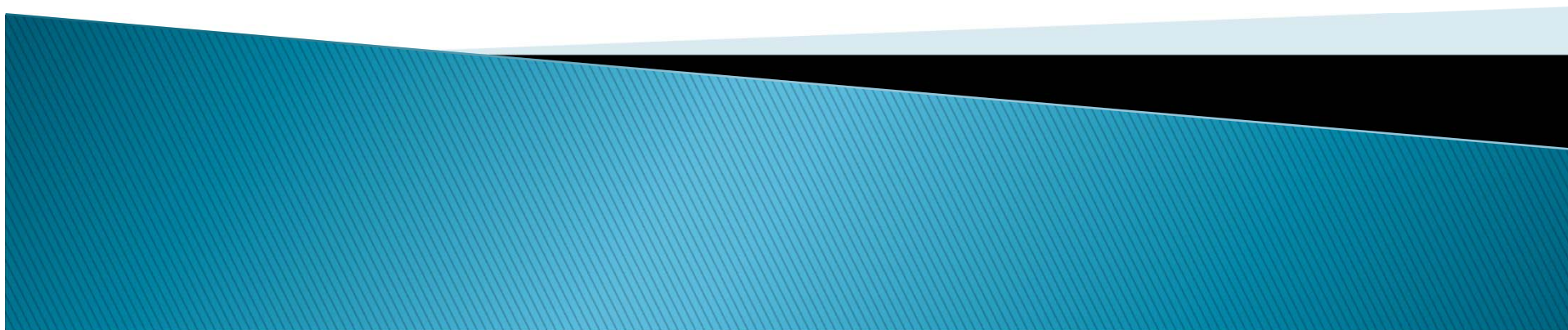
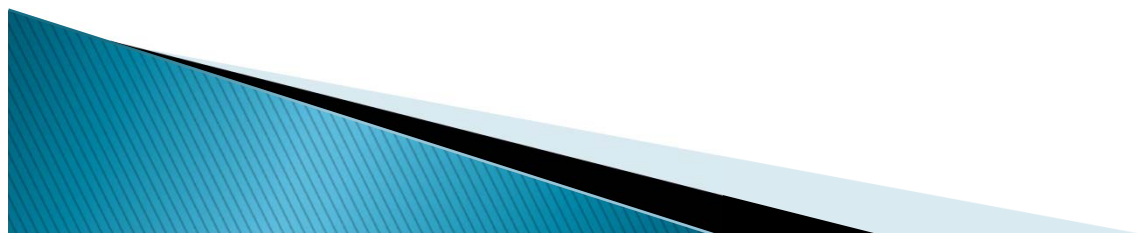


# 第八讲：函数



# 为什么需要函数？

- ▶ 当程序需要多次执行同一任务时，可将任务封装到函数中，这样程序每次需要执行这一任务，只需调用这一函数即可
- ▶ 使用函数，程序的编写、阅读、测试和修复都会变得更容易



# 函数的定义和调用

- ▶ 定义函数：

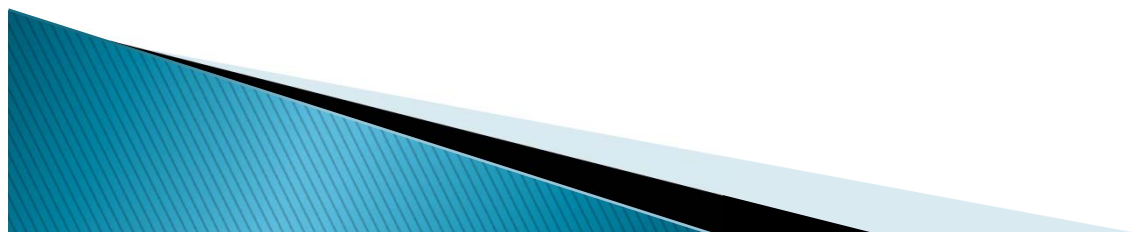
def 函数名（形式参数1，形式参数2， ...）：

语句1

语句2

...

- ▶ 调用函数：函数名（实际参数1，实际参数2， ...）



# 函数的定义和调用：例子

```
1 def greet_user(username):  
2     """Display a simple greeting."""  
3     print("Hello, " + username.title() + "!")  
4  
5 greet_user('jesse')
```

运行结果: Hello, Jesse!

- ▶ 这种传递参数的方式，在Python中称为位置实参，因为传递参数时，实际参数的顺序必须和形式参数一致；另外，Python还支持用关键字实参的方式来传递参数

# 关键字实参

- ▶ 函数调用时，可以通过 参数名=参数值 的方式传递参数，这种参数称之为关键字实参

```
1 def describe_pet(pet_name, animal_type):  
2     print("\nI have a " + animal_type + ".")  
3     print("My " + animal_type + "'s name is " + pet_name.title() + ".")  
4  
5     describe_pet('harry', 'hamster')  
6     describe_pet(pet_name='harry', animal_type='hamster')  
7     describe_pet(animal_type='hamster', pet_name='harry')
```

运行结果：

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

# 关键字实参

- ▶ 用关键字实参的方式传递参数时，参数可以按任何顺序传递；不指定关键字的传递参数方式称为位置实参，位置实参必须按形参的顺序传递
- ▶ 调用函数时，关键字实参和位置实参可以在一定程度上混合使用，但前提是所有的位置实参都必须在关键字实参的前面



# 关键字实参

```
1 def describe_pet(pet_name, animal_type):  
2     print("\nI have a " + animal_type + ".")  
3     print("My " + animal_type + "'s name is " + pet_name.title() + ".")  
4  
5 describe_pet('willie', animal_type = 'cat')
```

运行结果:

```
I have a cat.  
My cat's name is Willie.
```

```
1 def describe_pet(pet_name, animal_type):  
2     print("\nI have a " + animal_type + ".")  
3     print("My " + animal_type + "'s name is " + pet_name.title() + ".")  
4  
5 describe_pet(pet_name = 'willie', 'cat')
```

运行结果:

```
File "pets.py", line 5  
    describe_pet(pet_name = 'willie', 'cat')  
SyntaxError: positional argument follows keyword argument
```

# 参数默认值

- 在定义函数时，可以给参数指定默认值，这样在调用函数时如果没有提供实际参数，就会使用这一默认值

```
1 def describe_pet(pet_name, animal_type = 'dog'):  
2     print("\nI have a " + animal_type + ".")  
3     print("My " + animal_type + "'s name is " + pet_name.title() + ".")  
4  
5 describe_pet('willie')  
6 describe_pet(pet_name = 'willie')  
7 describe_pet('willie', 'cat')
```

运行结果：

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

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。

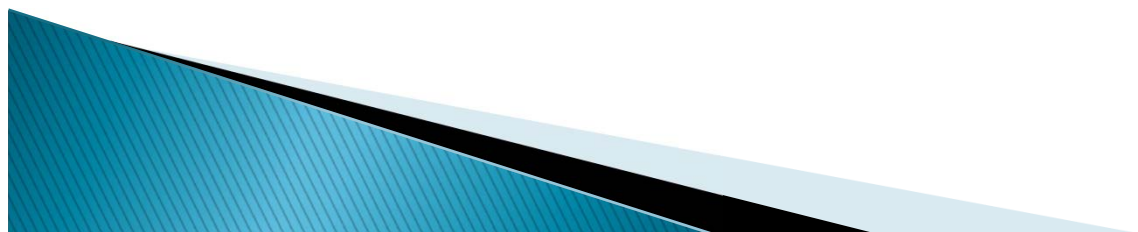


# 返回值

- ▶ 函数中可以使用return来返回值给函数的调用者

```
1 def get_formatted_name(first_name, last_name):  
2     full_name = first_name + ' ' + last_name  
3     return full_name.title()  
4  
5 musician = get_formatted_name('jimi', 'hendrix')  
6 print(musician)
```

运行结果: **Jimi Hendrix**



# 返回值

- ▶ 函数需要返回多个值，可以用以下方式：  
    return 返回值1, 返回值2, ...
- ▶ 获得函数的多个返回值，可以用以下方式：  
    变量1, 变量2, ... = 函数名(...)
- ▶ 这种方式，函数的返回值实际上是元组类型

```
1 def mul(x, y, k):  
2     return x * k, y * k  
3  
4 x, y = mul(3, 5, 2)  
5 print(x, y)  
6 print(mul(3, 5, 2))
```

运行结果：

```
6 10  
(6, 10)
```

# 可选实参

- ▶ 如果我们希望函数的参数是可选的，可以通过设定参数默认值的方式来实现

```
1 def get_formatted_name(first_name, last_name, middle_name=''):
2     if middle_name:
3         full_name = first_name + ' ' + middle_name + ' ' + last_name
4     else:
5         full_name = first_name + ' ' + last_name
6     return full_name.title()
7
8 musician = get_formatted_name('jimi', 'hendrix')
9 print(musician)
10
11 musician = get_formatted_name('john', 'hooker', 'lee')
12 print(musician)
```

运行结果:

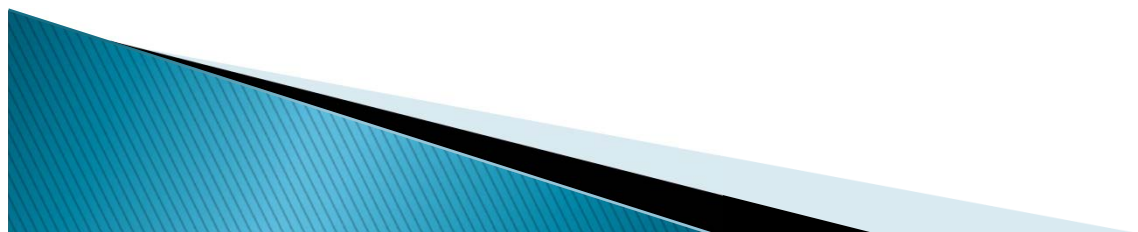
```
Jimi Hendrix
John Lee Hooker
```



# 返回字典

```
1 def build_person(first_name, last_name, age=''):
2     person = {'first': first_name, 'last': last_name}
3     if age:
4         person['age'] = age
5     return person
6
7 musician = build_person('jimi', 'hendrix', age=27)
8 print(musician)
```

运行结果: `{'first': 'jimi', 'last': 'hendrix', 'age': 27}`



# 关于参数传递的一些重要问题

- ▶ Q: 参数传递进函数后，如果函数发生改变，传进来的变量在函数外的值是否也跟着改变？
- ▶ A: 取决于变量的类型，目前学过的类型中，列表和字典会跟着变，其它类型不会

```
1 def modify_value(x):  
2     x = x * 2  
3  
4 x = 1  
5 modify_value(x)  
6 print(x)
```

运行结果:

1

# 在函数中修改列表

```
1 def print_models(unprinted_designs, completed_models):
2     while unprinted_designs:
3         current_design = unprinted_designs.pop()
4
5         print("Printing model: " + current_design)
6         completed_models.append(current_design)
7
8 def show_completed_models(completed_models):
9     print("\nThe following models have been printed:")
10    for completed_model in completed_models:
11        print(completed_model)
12
13
14    unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
15    completed_models = []
16
17    print_models(unprinted_designs, completed_models)
18    show_completed_models(completed_models)
```

运行结果:

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: iphone case

The following models have been printed:
dodecahedron
robot pendant
iphone case
```

# 防止函数修改列表

- ▶ 通过以下方式调用函数，传进函数的列表在函数中修改了，也不会影响到函数外列表变量中的内容：

函数名(列表名[:])

- ▶ 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，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



# 防止函数修改列表

```
1 def print_models(unprinted_designs, completed_models):
2     while unprinted_designs:
3         current_design = unprinted_designs.pop()
4
5         print("Printing model: " + current_design)
6         completed_models.append(current_design)
7
8 def show_completed_models(completed_models):
9     print("\nThe following models have been printed:")
10    for completed_model in completed_models:
11        print(completed_model)
12
13
14    unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
15    completed_models = []
16
17    print_models(unprinted_designs, completed_models[:])
18    show_completed_models(completed_models)
```

运行结果:

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: iphone case

The following models have been printed:
```



# 关于参数传递的一些重要问题

- ▶ Q: 那么对于像数值、字符串等类型的变量，如果希望在函数中修改这些变量的值，这些变量在函数外的值也跟着改变，应该怎么做？
- ▶ A: 一种方式是通过返回值来实现

```
1 def mul(x, y, k):  
2     return x * k, y * k  
3  
4 x = 3  
5 y = 5  
6 x, y = mul(x, y, 2)  
7 print(x)  
8 print(y)
```

运行结果:

6  
10

# 传递任意数量的实参

- ▶ 形式参数前加\*号，可以传递任意数量的实际参数

```
1 def make_pizza(*toppings):  
2     print("\nMaking a pizza with the following toppings:")  
3     for topping in toppings:  
4         print("- " + topping)  
5  
6 make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

运行结果:

```
Making a pizza with the following toppings:  
- mushrooms  
- green peppers  
- extra cheese
```



# 位置参数和任意数量参数结合使用

```
1 def make_pizza(size, *toppings):
2     print("\nMaking a " + str(size) +
3         "-inch pizza with the following toppings:")
4     for topping in toppings:
5         print("- " + topping)
6
7 make_pizza(16, 'pepperoni')
8 make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

运行结果:

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```



# 传递任意数量的关键字实参

- ▶ 形式参数前加\*\*号，可以传递任意数量的关键字实参

```
1 def build_profile(first, last, **user_info):
2     profile = {}
3     profile['first_name'] = first
4     profile['last_name'] = last
5     for key, value in user_info.items():
6         profile[key] = value
7     return profile
8
9 user_profile = build_profile('albert', 'einstein',
10                             location='princeton',
11                             field='physics')
12 print(user_profile)
```

运行结果:


```
{'first_name': 'albert', 'last_name': 'einstein', 'location': 'princeton', 'field': 'physics'}
```

# 将函数放在模块中

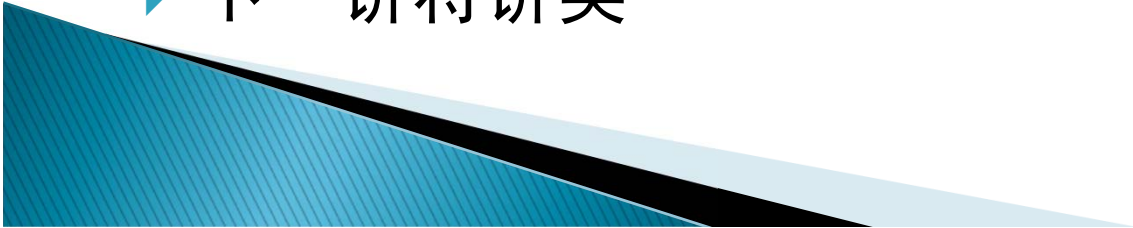
- ▶ 将不同函数放在不同文件（又称模块）中，可以更好的组织我们的程序
- ▶ 当前程序需要调用其它模块中的程序时，需要进行导入整个模块或模块中所需的函数



# 将函数放在模块中

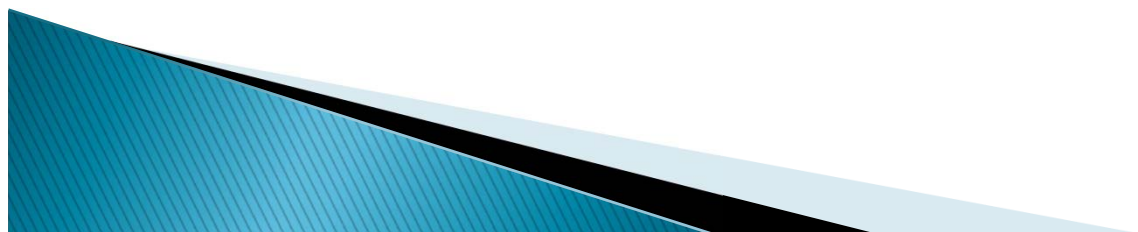
- ▶ `import 模块名` – 导入整个模块，此时调用模块中的函数需要用 `模块名.函数名` 的形式
  - ▶ `import 模块名 as 模块别名` – 导入整个模块，此时调用模块中的函数可以用 `模块别名.函数名` 的形式
  - ▶ `from 模块名 import 函数名` – 导入模块中的某个函数，此时可以只通过函数名调用函数，但只能调用导入的函数，不能调用模块中其它未导入的函数
  - ▶ `from 模块名 import *` – 可以通过函数名调用模块中的全部函数（容易造成混淆，不推荐）
  - ▶ 例子参见课本8.6节
- 

# 总结

- ▶ 函数的声明
  - ▶ 位置实参和关键字实参
  - ▶ 函数返回值
  - ▶ 参数传递的方式
  - ▶ 任意数量实参
  - ▶ 模块
  - ▶ 下一讲将讲类
- 

# 作业

- ▶ 教材中第8章课后的练习，选一些写到你的博客上





谢谢！

