



■ 第6章 函数与模块

6.1 函数的定义与调用

6.2 函数的参数

6.3 匿名函数

6.4 函数的递归

6.5 程序结构

- 掌握函数的定义与调用
- 理解并掌握函数的参数传递
- 理解变量的作用域
- 理解匿名函数的定义和调用
- 了解函数的递归
- 了解常用的内置函数



■ 内置函数

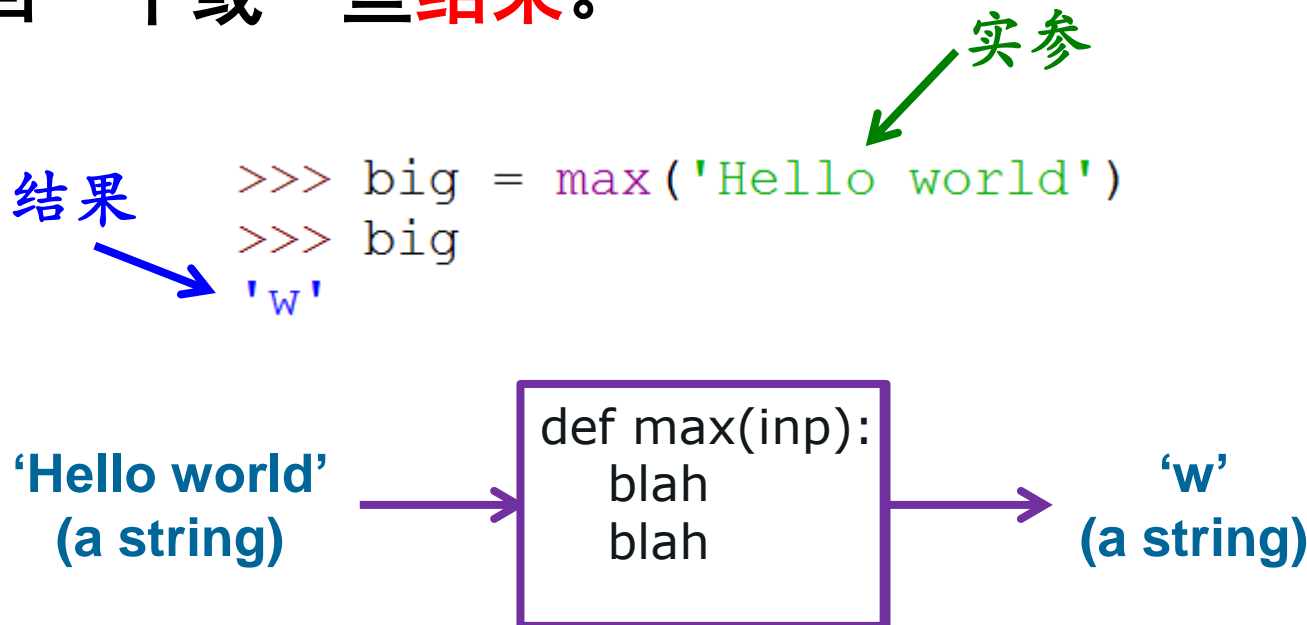
❖ Python提供了很多重要的内置函数

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	



函数的概念

- ❖ 函数是一段可重复使用的程序代码。
- ❖ 函数以**实参**作为输入，执行一些运算，然后返回一个或一些**结果**。



Guido wrote this code



```
def myfunc(x, y):  
    return x+y
```

6.1 自定义函数

- ❖ 将一些经常重复使用的程序代码定义为函数，方便重复调用执行。
- ❖ 自定义函数用
- ❖ 函数由函数名

函数名必须是
合法标识符

组成

```
def <函数名> ([形式参数列表]):
```

函数头

```
<执行语句>
```

函数体

```
[return <返回值>]
```

函数体的各语句
保持缩进



```
def myfunc(x, y):  
    return x+y
```

6.1 自定义函数

- ❖ 函数定义时，用来接收输入的参数称为**形式参数**，简称**形参**。
- ❖ 函数可以没有形参（即不需要传递数据），但要保留圆括号。
- ❖ 函数可以没有返回值（返回值为None）

```
def myprint():  
    print("I'm okey.")
```

- ❖ 占位语句pass：待以后完善

```
def emptyfunc():  
    pass
```



6.1 函数的调用

❖ 函数调用的一般形式：

<函数名>([实际参数列表])

❖ 调用函数时，实际传递给函数的参数称为**实际参数**，简称**实参**。

❖ 即使没有要传入的实参，也要留空括号。

```
def myfunc(x, y):  
    return x + y  
a, b = 2.5, 3.6  
c = myfunc(a, b)  
print('%.2f+%.2f=%.2f' % (a, b, c))
```

形参

实参



6.1 函数的调用

如何为多个朋友输出生日歌？

【例】生日歌。过生日时要为朋友唱生日歌，歌词为：

```
Happy birthday to you!  
Happy birthday to you!  
Happy bithday, dear Mike!  
Happy birthday to you!
```

```
def happy():  
    print('Happy birthday to you!')
```

```
happy()  
happy()  
print('Happy bithday, dear Mike!')  
happy()
```



6.1 函数的调用

【例】生日歌。为多个朋友输出生日歌。

```
def happy():  
    print('Happy birthday to you!')  
  
def happyB(name):  
    happy()  
    happy()  
    print('Happy birthday, dear %s!' % name)  
    happy()
```

```
happyB('Mike')  
print()  
happyB('Jane')
```

形参

实参

注意：函数定义时，函数体里面的代码并没有被执行，直到被调用时才会被执行。



■ 6.1 函数的调用

❖ 程序调用一个函数，需要进行以下4个步骤：

- ◆ 程序在调用处暂停；
- ◆ 将实参复制给函数的形参；
- ◆ 执行函数体语句；
- ◆ 调用结束给出返回值，程序回到之前暂停的位置，继续往下执行。



6.1 函数的调用

name='Mike'

```
happyB('Mike')  
print()  
happyB('Jane')
```

```
def happyB(name):  
    happy()  
    happy()  
    print('Happy birthday, dear %s!' % name)  
    happy()  
def happy():  
    print('Happy birthday')
```

◆ 程序在调用处暂停;





6.2 函数的参数传递

❖ 参数按位置依次传递

调用函数时，按照定义时的参数顺序依次传递，这是最常见的参数传递方法。

【例6-1】 带参数的函数调用示例。

```
def myfunc(x, y):  
    return x + y
```

```
a, b = 2.5, 3.6
```

```
c = myfunc(a, b)
```

```
print('% .2f+% .2f=% .2f' % (a, b, c))
```

运行结果为:

2.50+3.60=6.10



6.2 函数的参数传递

❖ 参数赋值传递

调用函数时，按照形参名称输入实参。圆括号内用“**形参变量名=实参值**”的方式传入参数。由于指定了参数名称，所以参数之间的顺序可以任意调整。

```
def func(x1, y1, x2, y2):  
    return (x1-x2)**2 + (y1-y2)**2  
  
result1 = func(1, 2, 3, 4) #按位置顺序传递  
result2 = func(x2=3, y2=4, x1=1, y1=2) #参数赋值传递
```



6.2 函数的参数传递

注意：默认参数只能定义在参数列表的最后。

❖ 参数默认值传递

在定义函数时，可定义默认参数。调用函数时，若没有传递该形式参数，则会使用默认参数值。

【例6-2】 带默认参数的函数调用示例。

```
def myfunc(x, y=2):  
    return x + y
```

```
a, b = 2.5, 3.6
```

```
c = myfunc(a)    #参数默认值传递
```

```
d = myfunc(y=b, x=a)  #参数赋值传递
```

```
print('%.2f+默认值=%.2f' % (a, c))
```

```
print('%.2f+%.2f=%.2f' % (a, b, d))
```

```
== RESTART: C:/Us  
2.50+默认值=4.50  
2.50+3.60=6.10
```



6.2 函数的参数传递

❖ 可变长参数

在定义函数时，也可以设计可变长参数，通过在参数前增加星号‘*’实现。注意：可变长参数只能出现在参数列表的最后。

◆ 元组类型的可变长参数

```
def func(a, *b):  
    pass
```

◆ 字典类型的可变长参数

```
def func(a, **c):  
    pass
```



6.2 函数的参数传递

❖ 元组类型可变长参数

调用时，这些参数被当做元组类型传递到函数中。

```
def func(a, *b):  
    print("a=", a)  
    print("b=", b)  
    print("b的类型: ", type(b))
```

```
func(10)
```

```
func(20, 30, 40, 50)
```

```
a= 10  
b= ()  
b的类型: <class 'tuple'>  
a= 20  
b= (30, 40, 50)  
b的类型: <class 'tuple'>
```



6.2 函数的参数

```
a= 1
b= (2, 3, 4, 5, 6)
b的类型: <class 'tuple'>
c= {'x': 10, 'y': 20, 'z': 30}
c的类型: <class 'dict'>
```

❖ 字典类型可变长参数

调用时，以实参变量名等于字典值的方式传递参数，实参变量名则以字符形式作为字典的键。

```
def func(a, *b, **c):
    print("a=", a)
    print("b=", b)
    print("b的类型: ", type(b))
    print("c=", c)
    print("c的类型: ", type(c))

func(1, 2, 3, 4, 5, 6, x=10, y=20, z=30)
```




6.2 函数的参数传递

❖ 高阶函数

◆ 变量可以用来指向函数。

◆ 函数名

```
>>> abs(-10)    #调用函数abs()  
10  
>>> abs         #abs是函数本身  
<built-in function abs>  
>>> f = abs     #变量f指向函数  
>>> f  
<built-in function abs>  
>>> f(-10)      #通过变量f来调用函数  
10
```



6.2 函数的参数传递

❖ 高阶函数

如果一个函数接收另一个函数作为参数，数就称为高阶函数。

#例：一个最简单的高阶函数


```
def add(x, y, f):    #形参f为一个函数对象
    return f(x) + f(y)
```

```
a, b = -5, 6
```

```
print( add(a, b, abs) ) #参数x,y,f分别接收-5,6,abs
```

```
print( add(1.8, 3.5,int) )
```

```
>>> x = -5
>>> y = 6
>>> f = abs
>>> f(x)+f(y)
11
>>> x = 1.8
>>> y = 3.5
>>> f = int
>>> f(x)+f(y)
4
```





6.2 函数的参数传递

❖ 高阶函数——内置map() 函数

- ◆ 有两个参数：一个是函数，一个是序列对象

```
ls = map(int, input().split())
```

例：有一个函数 $f(x)=x^2$ ，把这个函数作用在列表
[1,2,3,4,5,6,7,8,9]上生成一个新列表。

```
>>> def f(x):  
        return x*x  
  
>>> ls = map(f, [1,2,3,4,5,6,7,8,9])  
>>> list(ls)  
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```



6.2 函数的参数传递

❖ 函数中变量的作用域

变量的作用域是指在程序中能够对该变量进行读/写操作的**范围**。程序中的变量包括两类：

- ◆ **全局变量**：在**函数之外**定义的变量，一般无缩进，在程序执行全过程都有效。
- ◆ **局部变量**：在**函数内部**使用的变量，仅函数内部有效，当函数退出时变量不复存在。



6.2 函数的参数传递

❖ 函数中变量的作用域

```
>>> n=1    #n是全局变量
>>> def func(a, b):
        c = a*b    #c是局部变量，a和b作为形参也是局部变量
        return c
```

```
>>> s = func("knock~", 2)
```

```
>>> print(s)
```

```
knock~knock~
```

```
>>> print(c)
```

这个例子说明：当函数执行完退出后，其内部变量将被释放。

```
Traceback (most recent call last):
```

```
File "<pyshell#35>", line 1, in <module>
```

```
print(c)
```

```
NameError: name 'c' is not defined
```



6.2 函数

❖ 函数中变

函数内的局部变量n与全局变量n是不同的变量，函数执行时仅局部变量n可见，函数退出时局部变量n被释放。

```
>>> n = 1 #n是全局变量
>>> def func(a, b):
    n = b #这个n是在函数内新生成的局部变量，不是全局变量
    return a*b
>>> s = func("knock~", 2)
>>> print(s, n) #测试一下n值是否改变?
knock~knock~ 1
```

思考：函数func()内部使用了变量n，并且将b的值赋给了n，为何全局变量n值没有改变？



6.2 函数的参数传递

❖ 函数中变量的作用域

如果希望让func()函数将n当做全局变量，则需要
在变量n使用前**显式声明**该变量为全局变量。

```
>>> n = 1    #n是全局变量
>>> def func(a, b):
    global n
    n = b    #将b赋给全局变量 n
    return a*b
>>> s = func("knock~", 2)
>>> print(s, n)    #测试一下n值是否改变?
knock~knock~ 2
```



6.2 函数的参数传递

❖ 函数中变量的作用域

根据作用域不同，变量可分为：

- ◆ 函数中定义的局部变量Local
- ◆ 嵌套中父级函数内定义的局部变量Enclosing
- ◆ 模块级别定义的全局变量Global
- ◆ 内置模块中的变量Built-in

优先级高

低





■ 6.2 函数的参数传递

❖ 函数中变量的作用域

Python允许出现同名变量

- ◆ 同名变量出现在不同的函数体中，代表不同的对象，互不干扰。
- ◆ 若同名变量在同一函数体中或具有函数嵌套关系，则不同作用域的变量各自代表不同的对象，程序执行时按优先级进行访问。



6.2 函数的参数传递

【例6-6】 变量作用域测试。

```
x = 0    #global
def outer():
    x = 1    #enclosing
    def inner():
        x = 2    #local
        print('local: x =', x)
    inner()
    print('enclosing: x =', x)
outer()
print('global: x =', x)
```

```
local:  x = 2
enclosing:  x = 1
global:  x = 0
```



6.2 函数的参数传递

运行结果为：

0
5
5

【例6-7】 全局变量声明测试。

```
sum = 0
def func():
    global sum    # 用global关键字声明对全局变量的改写操作
    print(sum)    # 累加前
    for i in range(5):
        sum += 1
    print(sum)    # 累加后
func()
print(sum)       # 观察执行函数后全局变量发生变化
```