# py02_day02

---

## 函数

函数创建没有先后关系，只要在调用时，所有的函数已经存在即可

```
>>> def foo():
...     print('in foo')
...     bar()
...
>>> def bar():
...     print('in bar')
...
>>> foo()
in foo
in bar
```

## 函数参数

- 只有一个参数名，称作位置参数
- 参数形式是key=val，称作关键字参数

```
>>> def get_age(name, age):
...     print('%s is %s years old' % (name, age))
...
>>> get_age('bob', 20)   # OK
bob is 20 years old
>>> get_age()   # Error，参数太少
>>> get_age('bob', 20, 100)   # Error，参数太多
>>> get_age(20, 'bob')  # 语法没有问题，但是语义不对
20 is bob years old
>>> get_age(age=20, name='bob')  # OK
bob is 20 years old
>>> get_age(age=20, 'bob')  # Error，关键字参数必须在后
>>> get_age(20, name='bob')  # Error，name得到了多个值
>>> get_age('bob', age=20)   # OK
bob is 20 years old
```

## 参数组

如果函数的参数个数是不固定的，可以使用参数组接收参数

- 在参数名前加*表示参数是元组

- 在参数名前加**表示参数是字典

```
>>> def func1(*args):
...     print(args)
...
>>> func1()
()
>>> func1('bob')
('bob',)
>>> func1('bob', 123)
('bob', 123)
>>> func1('bob', 123, 'hello', 'aaa')
('bob', 123, 'hello', 'aaa')
>>> def func2(**kwargs):
...     print(kwargs)
...
>>> func2()
{}
>>> func2(name='bob', age=20)
{'name': 'bob', 'age': 20}
```

调用函数时，如果参数有*号，表示把参数拆开。

```
>>> def add(x, y):
...     print(x + y)
...
>>> nums = [10, 20]
>>> num_dict = {'x': 100, 'y': 200}
>>> add(nums)  # Error，因为nums传给了x，y没有得到值
>>> add(num_dict)  # Error，同上
>>> add(*nums)  # nums拆成了10, 20
30
>>> add(**num_dict)  # 拆成了x=100, y=200
300
```

## 加减法小程序

```
5 + 5 = 10
Very Good!!!
Continute(y/n)? y
85 - 23 = 1
Wrong answer.
85 - 23 = 10
Wrong answer.
85 - 23 = 30
Wrong answer.
85 - 23 = 62  (系统给出正确答案)
Continute(y/n)? n
Bye-bye
```

## 匿名函数

如果函数的代码块非常简单，只有一行，可以使用匿名函数。

匿名函数使用lambda关键字定义。

```
>>> def add(x, y):
...     return x + y
...
>>> lambda x, y: x + y
<function <lambda> at 0x7f0b96735c80>
>>>
>>> myadd = lambda x, y: x + y
>>> myadd(10, 20)
30
```

## filter函数

filter函数可以接受两个参数，第一个参数是函数，它必须返回真或假；第二个参数是个可迭代对象。filter的作用是，将第二个参数中的每一项交给第一个参数（函数）处理，返回值为真保留，否则过滤。

## map函数

map函数可以接受两个参数，第一个参数是函数，第二个参数是个可迭代对象。将可迭代对象中的每项交由map处理，处理的结果保存下来。

## 变量

### 全局变量

在函数外面定义的变量。从定义起始位置开始，一直到程序结束，任何地方均可见可用。

### 局部变量

在函数内部定义的变量。只在函数内部使用，离开函数不能使用。函数的参数也可以看作是局部变量。

```
>>> x = 10
>>> def fn1():
...     print(x)
...
>>> fn1()
10
>>> def fn2():
...     x = 'hello'
...     print(x)  # 局部和全局有相同的变量，局部遮盖住全局变量
...
>>> fn2()
hello
>>> x
10
>>> def fn3():
...     global x  # 声明函数内部使用的是全局变量x
...     x = 'ni hao'
...     print(x)
...
>>> fn3()
ni hao
>>> x
'ni hao'
```

名称查找的时候，先查局部，再查全局，最后查内建。

```
>>> def fn4():
...     print(len('abc'))
...
>>> fn4()  # 查到了内建len函数
3
>>> len = 10
>>> fn4()  # 在全局查到len值是10，数字不能调用
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in fn4
TypeError: 'int' object is not callable
```

## 偏函数

可以理解为，改造现有函数，将现有函数中的一些参数赋值，生成新函数。

```
>>> def add(a, b, c, d, e):
...     return a + b + c + d + e
...
>>> add(10, 20, 30, 40, 5)
105
>>> add(10, 20, 30, 40, 8)
108
>>> add(10, 20, 30, 40, 13)
113
>>> from functools import partial
>>> myadd = partial(add, 10, 20, 30, 40)
>>> myadd(5)
105
>>> myadd(8)
108
```

## 递归函数

一个函数又包括对自身的调用，就是递归函数。

数学上的阶乘就是递归。

```
5! = 5 x 4 x 3 x 2 x 1
5! = 5 x 4!
5! = 5 x 4 x 3!
5! = 5 x 4 x 3 x 2!
5! = 5 x 4 x 3 x 2 x 1!
1! = 1
```

### 快速排序

假定第一个数是中间值。把比这个数小的放到smaller列表，把比它大的数放到larger列表。最后把这三个部分拼接起来。smaller和larger采用相同的办法继续排序，直到列表的长度是0或1结束。

## 生成器

生成器本质上也是函数。和普通函数不同，普通函数只能通过return返回一个值，但是生成器可以通过yield返回多个中间值。

生成器对象只能使用一次。

```
>>> def mygen():
```

```
...     yield 'hello'
...     a = 10 + 20
...     yield a
...     yield 100
...
>>> mg
<generator object mygen at 0x7faa8cf631a8>
>>> list(mg)
['hello', 30, 100]
>>> list(mg)
[]
>>> mg = mygen()
>>> for i in mg:
...     print(i)
...
hello
30
100
```

生成器还可以使用生成器表达式

```
>>> from random import randint
>>> nums = (randint(1, 100) for i in range(10))
>>> nums
<generator object <genexpr> at 0x7faa8cf63eb8>
>>> list(nums)
[20, 97, 30, 67, 82, 99, 46, 81, 35, 71]
>>> ips = ('192.168.1.%s' % i for i in range(1, 255))
>>> ips
>>> for ip in ips:
...     print(ip)
```