# nsd1902_py02_day02

## 函数参数

直接给定一个参数名称为位置参数，给定了key=val形式的参数称作关键字参数

```
>>> def get_info(name, age):
...   print('%s is %s years old.' % (name, age))
>>> get_info('bob', 20)  # OK
>>> get_info(20, 'bob')  # 语法正确，语义不对
>>> get_info(age=20, 'bob')  # Error，关键字参数必须在后
>>> get_info(20, name='bob')  # Error，name得到了多个值
>>> get_info('bob', age=20)  # OK
>>> get_info(age=20, name='bob')  # OK
>>> get_info('bob')  # Error，参数不够
>>> get_info('bob', 20, 100)  # Error，参数太多了
```

参数个数不确定的函数

```
# 参数名前加*号，表示用元组接收参数
>>> def func1(*args):
...   print(args)
...
>>> func1()
()
>>> func1('hao')
('hao',)
>>> func1('hao', 123)
('hao', 123)
# 参数名前加**号，表示用字典接收参数
>>> def func2(**kwargs):
...   print(kwargs)
...
>>> func2()
{}
>>> func2(name='bob', age=20)
{'name': 'bob', 'age': 20}

# 传参时加上*号或**号表示把序列或字典拆开
>>> def add(a, b):
...   return a + b
...
>>> nums = [10, 20]
>>> add(nums)  # 报错，把nums传给a，b没有得到数据
>>> add(*nums)  # 将nums拆开，得到10和20，分别赋值给a和b
30
```

数学加减法运行过程：

```
10 + 5 = 15
Very Good!
Continue(y/n)? y
13 + 12 = 15
Wrong Answer.
13 + 12 = 16
Wrong Answer.
13 + 12 = 17
Wrong Answer.
The Anser is: 13 + 12 = 25
Continue(y/n)? n
Bye-bye
```

# 匿名函数

```
>>> def add(x, y):
...   return x + y
...
>>> myadd = lambda x, y: x + y
>>> myadd(10, 20)
30
>>> add(20, 30)
50
```

## filter函数

用于过滤数据。filter(func, seq)，将seq中的每一项作为func函数的参数进行过滤，如果func的返回值是True就留下来，否则过滤掉

```
>>> from random import randint
>>> nums = [randint(1, 100) for i in range(10)]
>>> nums
[89, 67, 56, 63, 66, 23, 54, 40, 69, 6]
>>> def func1(x):
...   return True if x % 2 == 0 else False
>>> list(filter(func1, nums))  # 将奇数过滤掉
[56, 66, 54, 40, 6]
>>> list(filter(lambda x: True if x % 2 == 0 else False, nums))
[56, 66, 54, 40, 6]
```

## map函数

用于加工数据。map(func, seq)，将seq中的每一项作为func的参数，func将数据加工处理后返回。

```
>>> def func2(x):
...   return x * 2
>>> list(map(func2, nums))
[178, 134, 112, 126, 132, 46, 108, 80, 138, 12]
>>> list(map(lambda x: x * 2, nums))
[178, 134, 112, 126, 132, 46, 108, 80, 138, 12]
```

## 变量作用域

- 在函数外面的变量是全局变量，它从定义开始到程序结束一直可见可用

```
>>> x = 10
>>> def func1():
...   print(x)
...
>>> func1()
10
```

- 函数内部的变量是局部变量，只能在函数内部使用

```
>>> def func2():
...   y = 100
...   print(y)
...
>>> func2()
100
>>> print(y)   # NameError，局部变量不能在全局使用
```

- 局部如果和全局有同名变量，函数调用时局部变量将会遮盖住全局变量

```
>>> x = 10
>>> def func3():
...   x = 'hello world'
...   print(x)
...
>>> func3()
hello world
>>> print(x)
10
```

- 如果需要在局部改变全局变量，需要使用global关键字

```
>>> x = 10
>>> def func4():
...   global x
...   x = 'hello world'
...   print(x)
...
>>> func4()
hello world
>>> print(x)
hello world
```

程序在运行时，将会按这样的顺序查找名称：局部、全局、内建。

## 偏函数

偏函数是指通过fuctools.partial进行改造现有函数，生成新函数。

```
# int()函数默认可以将字符类型的数字转成10进制整数
>>> int('1010')
1010    # 一千零一十
>>> int('1010', base=2)   # 通过base=2说明1010是2进制数
10   # 输出为10进制数

# 改造int函数，把base=2固定下来，生成名为int2的新函数
>>> from functools import partial
>>> int2 = partial(int, base=2)
>>> int2('1010')
10

# 改造函数，将参数固定下来
>>> def add(a, b, c, d, e):
...   return a + b + c + d + e
...
>>> add(10, 20, 30, 40, 1)   # 每次调用函数，前4项的值都是一样的
101
>>> add(10, 20, 30, 40, 2)
102
>>> myadd = partial(add, 10, 20, 30, 40)  # 改造add函数，固定前4个参数值
>>> myadd(1)
101
>>> myadd(2)
102
```

递归函数：了解性内容

如果一个函数的内部又包括了对自身的调用就是递归函数。一般来说，递归可以用循环替代。

```
5!=5x4x3x2x1  # 5的阶乘
5!=5x4!
5!=5x4x3!
5!=5x4x3x2!
5!=5x4x3x2x1!
```

快速排序：

- 假设第一个数是中间值，赋值给middle
- 遍历剩余的数字，比middle小的放到smaller列表，比middle大的放到larger列表
- 把smaller、middle和larger拼接起来
- smaller和larger继续使用相同的方法进行排序
- 如果列表只有一项或是空的就不用再排了，直接将列表返回

## 生成器

- 生成器对象

```
# 与列表解析语法一样，只是把[]换成()
>>> ('192.168.1.%s' % i for i in range(1, 255))
<generator object <genexpr> at 0x7ffa5e95e780>
>>> ips = ('192.168.1.%s' % i for i in range(1, 255))
>>> for ip in ips:
...   print(ip)
```

- 函数的形式

与普通的函数有所区别。一般来说，函数通过return返回一个值；生成器函数可以通过yield关键字返回很多中间结果。

```
>>> def mygen():
...   yield 100
...   a = 10 + 20
...   yield a
...   yield 300
>>> mg = mygen()
>>> for n in mg:
...   print(n)
...
100
30
300
```

# 模块

导入模块时，Python将会到sys.path定义的路径下查找模块，如果查到则导入，否则报错。

```
>>> import sys
>>> sys.path   # 空串表示当前路径
['', '/usr/local/lib/python36.zip', '/usr/local/lib/python3.6',
'/usr/local/lib/python3.6/lib-dynload', '/usr/local/lib/python3.6/site-packages']
```

当我们自己写的文件需要像标准模块一样，能在任意位置导入，可以

- 方法一：将自己写的模块文件放到site-packages中
- 方法二：定义环境变量PYTHONPATH=/path/to/your/modules

```
[root@room8pc16 day02]# pwd
/var/ftp/nsd2019/nsd1902/python02/day02
[root@room8pc16 day02]# ls qsort.py
qsort.py
[root@room8pc16 day02]# export \  PYTHONPATH=/var/ftp/nsd2019/nsd1902/python02/day02
[root@room8pc16 day02]# cd /tmp/
[root@room8pc16 tmp]# python3
>>> import qsort   # 成功导入
```

# hashlib模块

用于计算文件的哈希值：md5 / sha / sha256 / sha512

```
[root@room8pc16 day02]# md5sum /etc/passwd
```

通过hashlib计算md5：

```
>>> import hashlib
>>> m = hashlib.md5(b'123456')
>>> m.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'

# 计算文件的md5值
>>> with open('/etc/passwd', 'rb') as fobj:
...   data = fobj.read()
...
>>> m = hashlib.md5(data)
>>> m.hexdigest()
'decb544ed171583bb1d7722500910d9e'

# 多次更新，计算数据的md5值
>>> m1 = hashlib.md5()
>>> m1.update(b'12')
>>> m1.update(b'34')
>>> m1.update(b'56')
>>> m1.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'
```

练习：编写checkmd5.py

1. 通过命令行的位置参数给定文件名
2. 计算出文件的md5值，打印到屏幕上