

# 干货分享 | Python从入门到编写POC之特殊函数

i春秋 昨天

前期回顾 >>

[Python介绍及安装使用](#)

[Python从入门到编写POC之读写文件](#)

Python从入门到编写POC系列文章是i春秋论坛作家「Exp1ore」表哥原创的一套完整教程，想系统学习Python技能的小伙伴，不要错过哦！

公众号旨在为大家提供更多的学习方法与技能技巧，文章仅供学习参考。



有料干货

今天分享的是Python从入门到编写POC之特殊函数、模块及包和库的相关知识点。

## 01 特殊函数

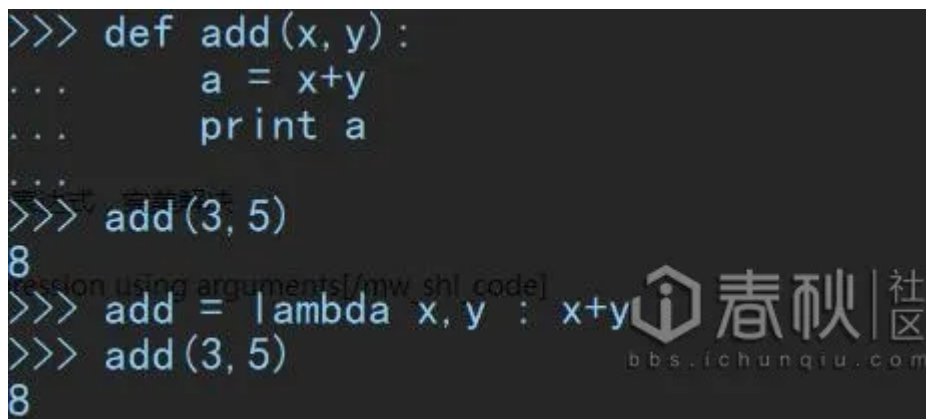
lambda函数，这个函数是一个只用一行就能解决问题的函数。

Python解释器中运行多行版：

```
1 >>> def add(x,y):
2 ...     a = x+y
3 ...     print a
4 ...
5 >>> add(3,5)
6 8
```

一行版（引入了lambda函数）：

```
1 >>> add = lambda x,y : x+y
2 >>> add(3,5)
3 8
```



```
>>> def add(x,y):
...     a = x+y
...     print a
...
>>> add(3,5)
8
>>> add = lambda x,y : x+y
>>> add(3,5)
8
```

**使用方法：**

在lambda函数后面直接加变量，变量后直接冒号，冒号后面是表达式，完美解决。

来个表达式：

```
1 lambda arg1,arg2,...,argn : expression using arguments
```

可以通过help命令来查看map函数的官方文档。

```
1 >>> help(map)
2 Help on built-in function map in module __builtin__:
3
4 map(...)
5     map(function, sequence[, sequence, ...]) -> list
6
7     Return a list of the results of applying the function to the items of
8     the argument sequence(s).  If more than one sequence is given, the
```

```
9     function is called with an argument list consisting of the corresponding
10     item of each sequence, substituting None for missing values when not all
11     sequences have the same length. If the function is None, return a list of
12     the items of the sequence (or a list of tuples if more than one sequence).
```

```
>>> help(map)
Help on built-in function map in module __builtin__:

map(...)
    map(function, sequence[, sequence, ...]) -> list

    Return a list of the results of applying the function to the items of
    the argument sequence(s). If more than one sequence is given, the
    function is called with an argument list consisting of the corresponding
    item of each sequence, substituting None for missing values when not
    sequences have the same length. If the function is None, return a list of
    the items of the sequence (or a list of tuples if more than one sequence).
```

## 使用方法:

```
1 map(函数, 序列对象)
```

举个0-10的平方数的例子:

```
1 >>> numbers = range(11)
2 >>> map(lambda x : x**2,numbers)
3 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

较为简单的方法:

```
1 >>> [x**2 for x in numbers]
2 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

reduce函数也可通过help命令学习。

filter函数, 顾名思义, 就是过滤器的意思, 用法跟map一样: filter(函数,序列对象)。

举个例子:

```
1 >>> numbers = range(-2,11)
2 >>> numbers
3 [-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4 >>> filter(lambda x:x>5,numbers)
5 [6, 7, 8, 9, 10]
```

## zip函数:

zip()是Python的一个内建函数，它接受一系列可迭代的对象作为参数，将对象中对应的元素打包成一个个tuple（元组），然后返回由这些tuples组成的list（列表）。若传入参数的长度不等，则返回list的长度和参数中长度最短的对象相同。利用\*号操作符，可以将list unzip（解压）。

通过help命令来看：

```
1 >>> help(zip)
2 Help on built-in function zip in module __builtin__:
3
4 zip(...)
5     zip(seq1 [, seq2 [...]]) -> [(seq1[0], seq2[0] ...), (...)]
6
7     Return a list of tuples, where each tuple contains the i-th element
8     from each of the argument sequences. The returned list is truncated
9     in length to the length of the shortest argument sequence.
```

举个例子：

```
1 >>> a = [1,2,3]
2 >>> b = [4,5,6]
3 >>> zip(a,b)
4 [(1, 4), (2, 5), (3, 6)]
```

长度不等时，取长度最小的。

```
1 >>> c = 'HELLØ'
2 >>> d = 'MOMØ'
3 >>> zip(c,d)
4 [('H', 'M'), ('E', 'O'), ('L', 'M'), ('L', 'O')]
```

还有个常用的是构造字典：

```
1 >>> demo1 = ['a','b','c']
2 >>> demo2 = ['d','e','f']
3 >>> demo3 = zip(demo1,demo2)
4 >>> demo = dict(demo3)
```

```
5 >>> print demo
6 {'a': 'd', 'c': 'f', 'b': 'e'}
```

```
D:\ichunqiu\items>python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> zip(a, b)
[(1, 4), (2, 5), (3, 6)]
>>> c = 'HELLO'
>>> d = 'MOMO'
>>> zip(c, d)
[('H', 'M'), ('E', 'O'), ('L', 'M'), ('L', 'O')]
>>> demo1 = ['a', 'b', 'c']
>>> demo2 = ['d', 'e', 'f']
>>> demo3 = zip(demo1, demo2)
>>> demo = dict(demo3)
>>> print demo
{'a': 'd', 'c': 'f', 'b': 'e'}
```



## 02 模块

有C语言编程经验的朋友都知道在C语言中如果要引用sqrt这个函数，必须用语句"#include<math.h>"引入math.h这个头文件，否则是无法正常进行调用的。那么在Python中，如果要引用一些内置的函数，该怎么处理呢？

在Python中有一个概念叫做模块（module），这个和C语言中的头文件以及Java中的包很类似，比如在Python中要调用sqrt函数，必须用import关键字引入math这个模块，下面就来了解一下Python中的模块。

很多熟悉import语句的朋友都知道，在前面的内容讲述中，用到过base64这个库。

例子是这样的：

```
1 #coding = utf-8
2
3 import base64
4
5 def demo():
6     str = "TU9NTyBpcyBhIGJlYXV0aWZ1bCBnaXJs"
7     result = base64.b64decode(str)
8     print result
9     return
10    print "KISS MOMO"
11
12 demo()
```

这个是base64库，Python标准库之一，总而言之，就是一个模块，用import语句引入了这个模块，然后就可以使用模块中的函数了，不用自己动手写具体的函数，节约时间。

也可以这么理解，模块是一个程序，就是.py的程序，如何引入模块？

有以下四种方法：

### 【1】导入整个模块

```
1 >>> import sys
2 >>> print sys.argv
3 ['']
```

### 【2】只导入我们要用到的

```
1 >>> from sys import argv
2 >>> print argv
3 ['']
```

### 【3】模块名太长，可以起个别名

```
1 >>> import sys as s
2 >>> print s.argv
3 ['']
```

### 【4】从模块中导入所有

```
1 >>> from sys import *
2 >>> print path #输出sys模块中的path
3 ['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload', '/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages/PILcompat', '/usr/lib/python2.7/dist-packages/gtk-2.0', '/usr/lib/python2.7/dist-packages/ubuntu-ssc']
```

最后一种导入方法不建议使用，原因：若自己定义的变量或函数与所导入模块中的变量或函数同名，易产生混淆。

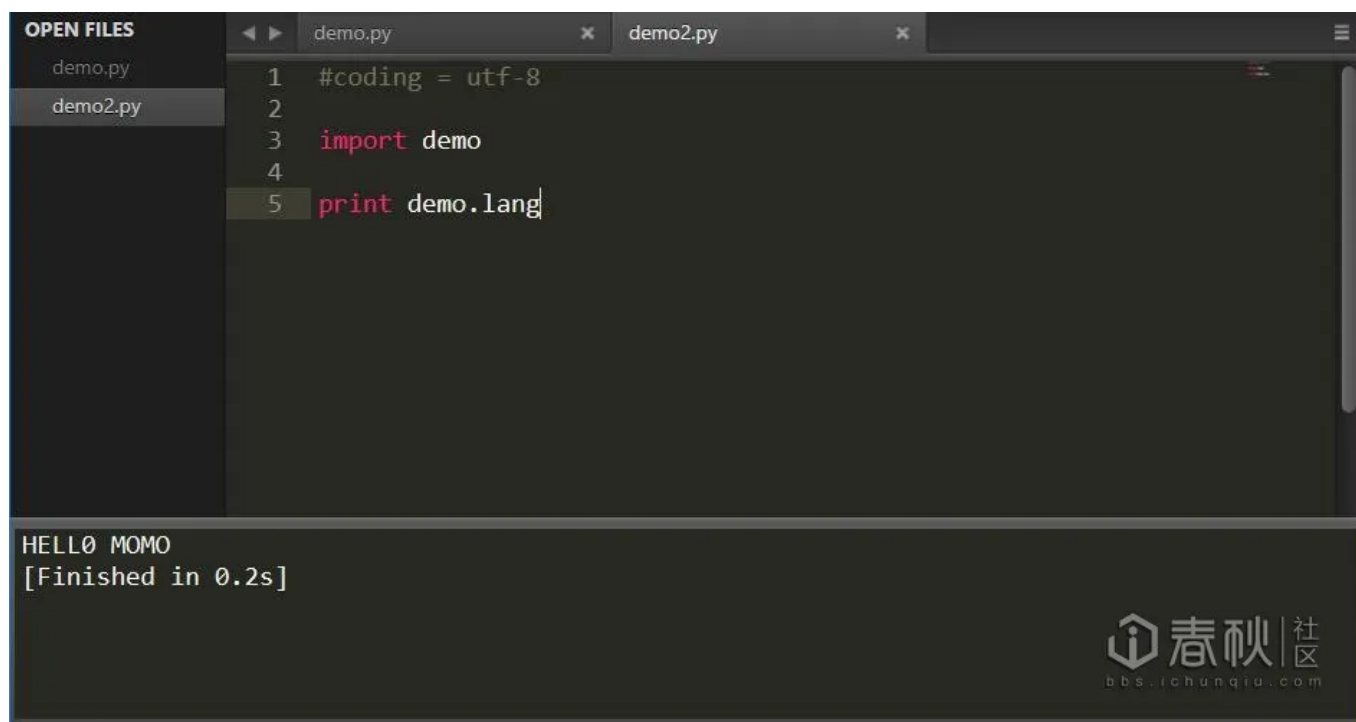
定义一个自己的模块，在一个目录下，创建了一个demo.py当作模块。

```
1 # coding = utf-8
2
3 lang = 'HELLØ MOMO'
```



接下来，创建一个py，引入这个模块：

```
1 #coding = utf-8
2
3 import demo
4
5 print demo.lang
```



如果是不同的目录该咋办呢？

这里我们就要用到sys库，把demo.py这个模块放到了c盘。



```
1 >>> import sys
2 >>> sys.path.append("C:\\Users\\20148\\Desktop\\demo.py")
```



用这种方法告诉Python解释器，我写的文件在哪里，C:\\Users\\20148\\Desktop\\demo.py是我的文件路径，然后就可以直接引入了。

```
1 >>> import demo
2 >>> demo.lang
3 'HELLO MOMO'
```



```
>>> import sys
>>> sys.path.append("C:\\Users\\20148\\Desktop\\demo.py")
>>> import demo
>>> demo.lang
'HELLO MOMO'
```

回到demo.py的储存目录，会发现多一个pyc文件。

 demo.py	2017/8	Python File	1 KB
 demo.pyc	2017/8	Compiled Pytho...	1 KB

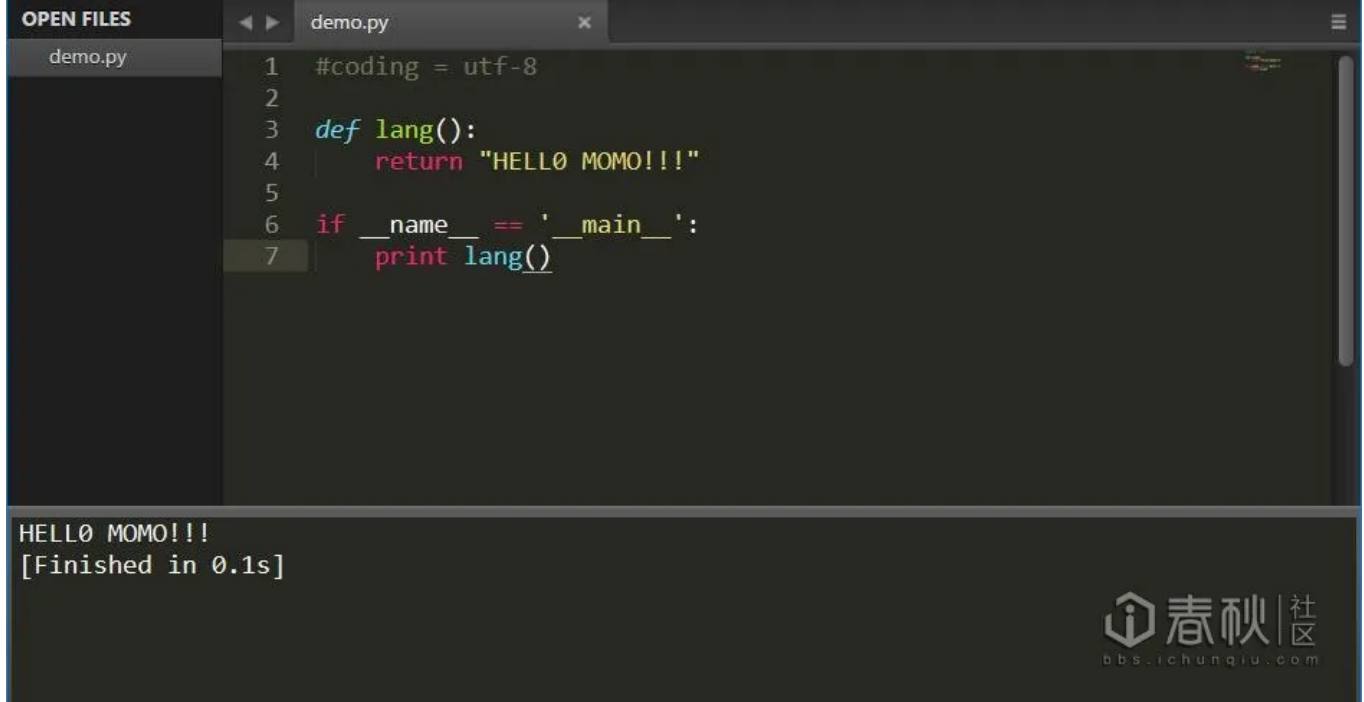
这是为什么呢？

因为当Python解释器都去了.py的文件，先将它变成由字节码组成的pyc文件，然后这个pyc文件把文件给了一个叫Python虚拟机的东西运行。

继续深入，将demo.py这个文件改造。

```
1 #coding = utf-8
2
3 def lang():
4     return "HELLO MOMO!!!"
5
6 if __name__ == '__main__':
7     print lang()
```





```
OPEN FILES
demo.py
1 #coding = utf-8
2
3 def lang():
4     return "HELLØ MOMO!!!"
5
6 if __name__ == '__main__':
7     print lang()

HELLØ MOMO!!!
[Finished in 0.1s]
```

如果将它作为模块，导入上面的方法，`sys.path.append("路径")`，然后查看模块属性，`dir()`。

```
1 >>> dir(demo)
2 ['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'lang']
```

都是一个文件，我们可以把它当作程序执行，也可以当作模块引入。

```
1 >>> __name__
2 '__main__'
3 >>> demo.__name__
4 'demo'
```

如果是程序执行的话，`__name__=="__main__"`。

如果是模块呢？`demo.__name__=="demo"`，即为模块的名称。

如果是模块的引入，就不用写`if __name__=="__main__"`了。

### 03 包和库

包(带 `__init__.py` 的文件夹)和库，比模块大了，一个包（就是咱们熟悉的目录）里面有好多的模块（即为.py文件），就跟你钱包一样，有好多钱一样，当然了，库就更大了，一个Python的标准库有好多的包，包又有一堆的模块。

举个例子：

建立一个叫ichunqiu的目录，里面放两个py文件，一个BaZong.py，一个MoMo.py，再建立一个空文件\_\_init\_\_.py。

BaZong.py源码如下：

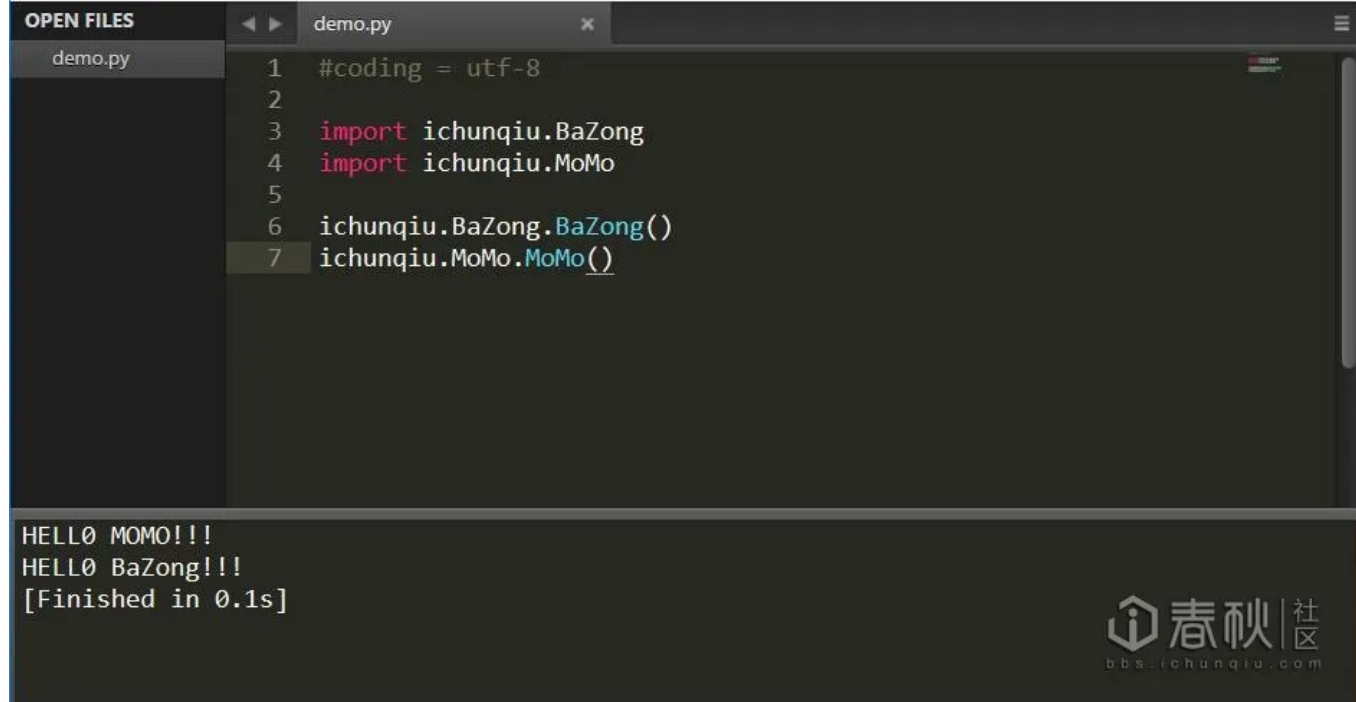
```
1  #coding = utf-8
2
3  def BaZong():
4      print 'HELL0 MOMO!!!'
```

MoMo.py源码如下：

```
1  #coding = utf-8
2
3  def MoMo():
4      print 'HELL0 BaZong!!!'
```

然后，咱们在与ichunqiu同级的目录中，创建一个demo.py调用这个ichunqiu的包。

```
1  #coding = utf-8
2
3  import ichunqiu.BaZong
4  import ichunqiu.MoMo
5
6  ichunqiu.BaZong.BaZong()
7  ichunqiu.MoMo.MoMo()
```



```
demo.py
1 #coding = utf-8
2
3 import ichunqiu.BaZong
4 import ichunqiu.MoMo
5
6 ichunqiu.BaZong.BaZong()
7 ichunqiu.MoMo.MoMo()
```

```
HELL0 MOMO!!!
HELL0 BaZong!!!
[Finished in 0.1s]
```

春秋社区  
bbs.ichunqiu.com

以上是今天要分享的内容，大家看懂了吗？

文章素材来源于i春秋社区

新来的朋友如果想要了解其他的必备技能和实用工具，可以点击菜单栏中的[入门锦囊](#)查看相关内容：



猜 你 喜 欢