



Approach Python

走近Python

Department of Computer Science and Technology
Department of University Basic Computer Teaching

用Python玩转数据

PYTHON简介

什么是Python

3

优雅

明确

简单

拥有简单脚本语言和解释型程序语言的易用性

拥有传统编译型程序语言所有强大通用的功能

Python是一种解释型的、面向对象的、带有动态语义的高级程序设计语言



Guido van Rossum

python的诞生

- 第1个Python编译器/解释器于1991年诞生
- Python名称来自Guido挚爱的电视剧Monty Python's Flying Circus
- Python介于C和Shell之间、功能全面、易学易用、可扩展

- 胶水语言

Glue Language

- 很容易和其他著名的程序语言连接（C/C++），集成封装

- 脚本语言

Script Language

- 高级脚本语言，比脚本语言只能处理简单任务强大

- 面向对象语言

Object-Oriented Language

- 完全支持继承、重载、派生、多继承

Python的特点

可移植 可升级 可扩展

健壮性 解释性 编译性

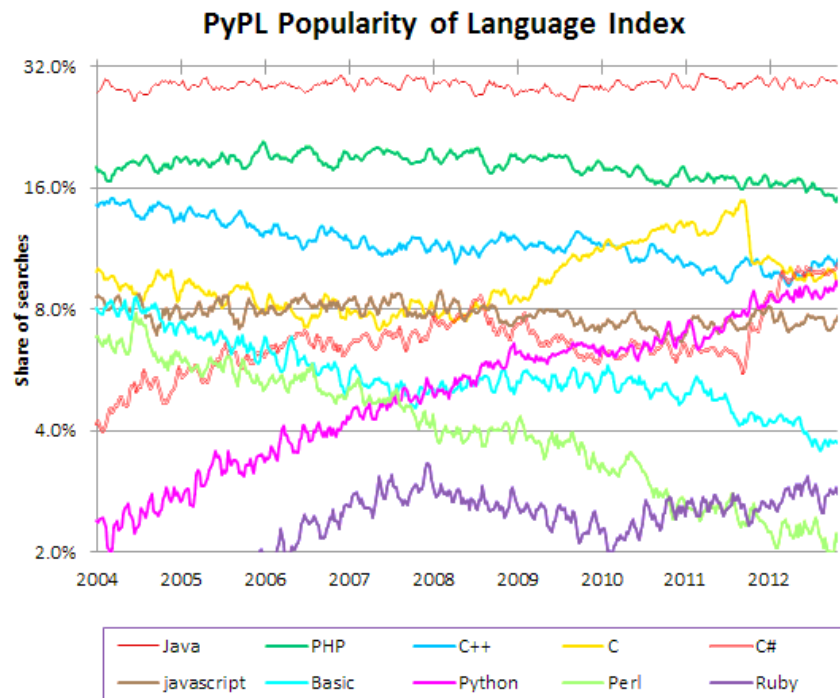
易学 易读 易维护

内存管理器

高级 面向对象

快速原型开发工具

Python的发展



编程语言流行指数 ([PyPL](#))

Python的应用（一）

8



Python定义了WSGI标准应用接口来协调http服务器与基于Python的Web程序之间的沟通

用wxPython或者PyQt来开发跨平台的桌面软件



Python的应用（二）



操作系统

大多数 Linux 发布版以及 NetBSD、OpenBSD 和 Mac OS X 都集成了 Python，Python 标准库包含了多个调用作业系统功能的库



多媒体

可用于计算机游戏三维场景制作

Python应用实例

10



Python超级程序员

11



Alex Martelli

2002 Activators' Choice Award和2006 Frank Willison award为Google开发商业智能软件



Daniel Greenfeld

之前在美国宇航局做开发，目前是Cartwheel Web的负责人



Miguel Grinberg

为Harmonic做视频软件。C++是主要语言，但用Python写的自动化单元测试框架更很有趣

Python 格言

The Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than **right** now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

by Tim Peters

用Python玩转数据

2

第一个 PYTHON程序

经典 Hello World

14

```
myString = 'Hello, World!'
```

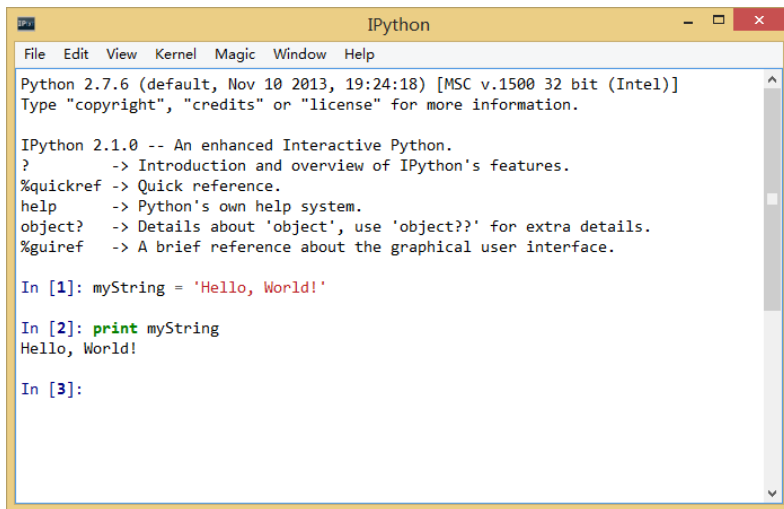
```
print myString
```

Python 3.x中print
语句用print()函
数替代

Python的运行方式（一）

15

Shell方式



```
IPython
File Edit View Kernel Magic Window Help
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 2.1.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.
%gui?            -> A brief reference about the graphical user interface.

In [1]: myString = 'Hello, World!'

In [2]: print myString
Hello, World!

In [3]:
```

- Shell是交互式的解释器
- 输入一行命令，解释器就解释运行出相应结果

文件方式

- 在Python的IDE环境中，创建一个以py为扩展名的文件
- 用Python解释器在Shell中运行出结果



经典 Hello World



```
>>> myString = 'Hello, World!'
>>> print myString
Hello World!
>>> myString
'Hello World!'
```



```
# Filename: helloworld.py
mystring = 'Hello, World!'
print mystring
```

- Python使用print语句实现输出：
 - print 变量
 - print 字符串



```
>>> myString = 'Hello World!'
>>> print myString
Hello World!
```

Python输入 : raw_input()语句

- raw_input()返回的类型是字符型



Python 3.x中raw_input()和input()整合成了input(), 返回类型为str

```
>>> price = raw_input('input the stock price of Apple:')
input the stock price of Apple:109
>>> price
'109'
>>> type(price)
<type 'str'>
```

Python 风格 (一)

20

注释



```
>>> # comment No.1
```

```
>>> print 'hello world!' # comment No.2  
hello world!
```

续行



```
>>> # long sentence
>>> if (signal == 'red') and\
    (car == 'moving'):
    car = 'stop'
elif (signal == 'green') and\
    (car == 'stop'):
    car = 'moving'
```



```
>>> # long sentence
>>> if (signal == 'red') and (car == 'moving'):
    car = 'stop'
elif (signal == 'green') and (car == 'stop'):
    car = 'moving'
```



续行

- 无需续行符可直接换行的两种情况：
 - 小括号、中括号、花括号的内部可以多行书写
 - 三引号包括下的字符串也可以跨行书写



```
>>> # triple quotes
>>> print "hi everybody,
welcome to python's MOOC course.
Here we can learn something about
python. Good lucky!"
```

一行多语句



```
>>> x = 'Today' ; y = 'is' ; z = 'Thursday' ; print (x,y,z)
('Today', 'is', 'Thursday')
```



```
>>> x = 'Today'
>>> y = 'is'
>>> z = 'Thursday'
>>> print (x,y,z)
('Today', 'is', 'Thursday')
```



缩进



S_{ource}

```
>>> # Indentation
>>> if (signal == 'red') and (car == 'moving'):
    car = 'stop'
    singal = 'yellow'
elif (signal == 'green') and (car == 'stop'):
    car = 'moving'
    singal = 'yellow'
```


用Python玩转数据

PYTHON 语法基础



```
>>> # variable
>>> p = 3.14159
>>> myString = 'is a mathematic circular constant'
>>> print p, myString
3.14159 is a mathematic circular constant
```

标识符

- 标识符是指Python语言中允许作为变量名或其他对象名称的有效符号
 - 首字符是字母或下划线
 - 其余可以是字母、下划线、数字
 - 大小写敏感(PI和pi是不同的标识符)



```
>>> # Identifier
>>> PI = 3.14159
>>> pi = 'one word'
>>> print PI
3.14159
>>> print pi
one word
```

- 关键字是Python语言的关键组成部分，不可随便作为其他对象的标识符
 - 在一门语言中关键字是基本固定的集合
 - 在 IDE 中常以不同颜色字体出现

and	as	assert	break	class	continue	def	del
elif	else	except	exec	finally	for	from	global
if	import	in	is	lambda	not	or	pass
print	raise	return	try	while	with	yield	

Python 3.x中的关键字

29

- 由Python 2.x中的31个变为33个

False	None	True	and	as	assert	break	class	continue
def	del	elif	else	except	finally	for	from	global
if	import	in	is	lambda	nonlocal	not	or	pass
raise	return	try	while	with	yield			

表达式

- 用运算符连接各种类型数据的式子就是表达式

算术运算符

乘方	**
正负号	+ -
乘除	* /
整除	//
取余	%
加减	+ -

位运算符

取反	~
与	&
或	
异或	^
左移	<<
右移	>>

比较运算符

小于	<
大于	>
小于等于	<=
大于等于	>=
等于	=
不等于	!=

逻辑运算符

非	not
与	and
或	or

表达式

- 运算符有优先级顺序
- 表达式必须有运算结果

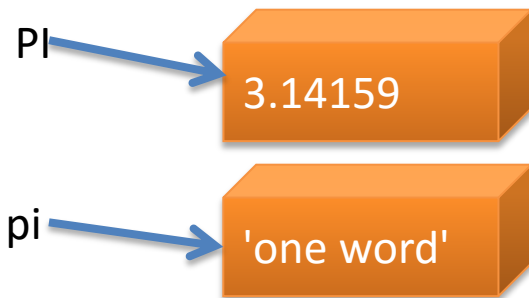
S_{ource}

```
>>> # expression
>>> PI = 3.14159
>>> r = 2
>>> c_circ = 2*PI*r
>>> print "The circle's circum : %f" %(c_circ)
```

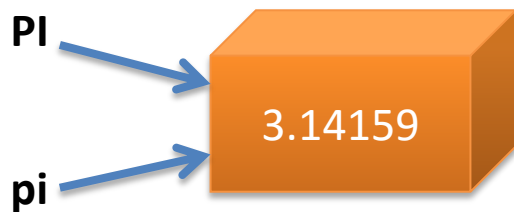
- $2*PI*r$ 是表达式
- 运算结果赋值给变量 `c_circ`

赋值

- 变量第一次赋值，同时获得类型和“值”
 - Python是动态的强类型语言
 - 不需要显式声明，根据“值”确定类型
 - 以“引用”的方式实现赋值



```
>>> # Identifier
>>> PI = 3.14159
>>> pi = 'one word'
>>> print PI
3.14159
>>> print pi
one word
```

S_{ource}

```
>>> # Identifier
>>> PI = 3.14159
>>> pi = PI
>>> print PI
3.14159
>>> print pi
3.14159
```

赋值 增量赋值

增量赋值 操作符

`+= -= *= /= %= **= <<= >>= &= ^= |=`

- `m %=5` 即 `m = m % 5`
- `m **=2` 即 `m = m ** 2`

S
ource

```
>>> # assignment
>>> m = 18
>>> m %= 5
>>> m
3
>>> m **= 2
>>> m
9
```

赋值 多重赋值

S
ource


```
>>> # assignment
>>> PI = pi = 3.14159
>>> PI
3.14159
>>> pi
3.14159
```

S
ource


```
>>> # assignment
>>> PI = 3.14159
>>> pi = PI = PI * 2
>>> pi
6.28318
```

赋值 多元赋值

- 等号左右两边都以元组的方式出现

 Source

```
>>> # assignment
>>> x = 1
>>> y = 2
>>> x, y
(1, 2)
>>> x, y = y, x
>>> x, y
(2, 1)
```

 Source

```
>>> # assignment
>>> PI, r = 3.14159, 3
>>> PI
3.14159
>>> r
3
>>> (PI, r) = (3.14159, 3) # same as no round brackets
```

- 完整执行一个任务的一行逻辑代码
 - 赋值语句完成了赋值
 - print输出语句完成了输出



S
ource

```
>>> # sentence  
>>> PI = 3.14159  
>>> r = 2  
>>> c_circ = 2*PI*r  
>>> print "The circle's circum : %f" %(c_circ)
```

语句和表达式

语句

完成一个任务

如，打印一份文件



表达式

任务中的一个具体组成部分

如，这份文件
的具体内容



用Python玩转数据

PYTHON 数据类型

- 必须有明确的数据类型，程序才能分配给常量、变量精确的存储大小，才能进行精确或高效率的运算

1	0	0	1	0	0	1	1
1	0	0	1	0	0	1	1

+

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---



1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

+

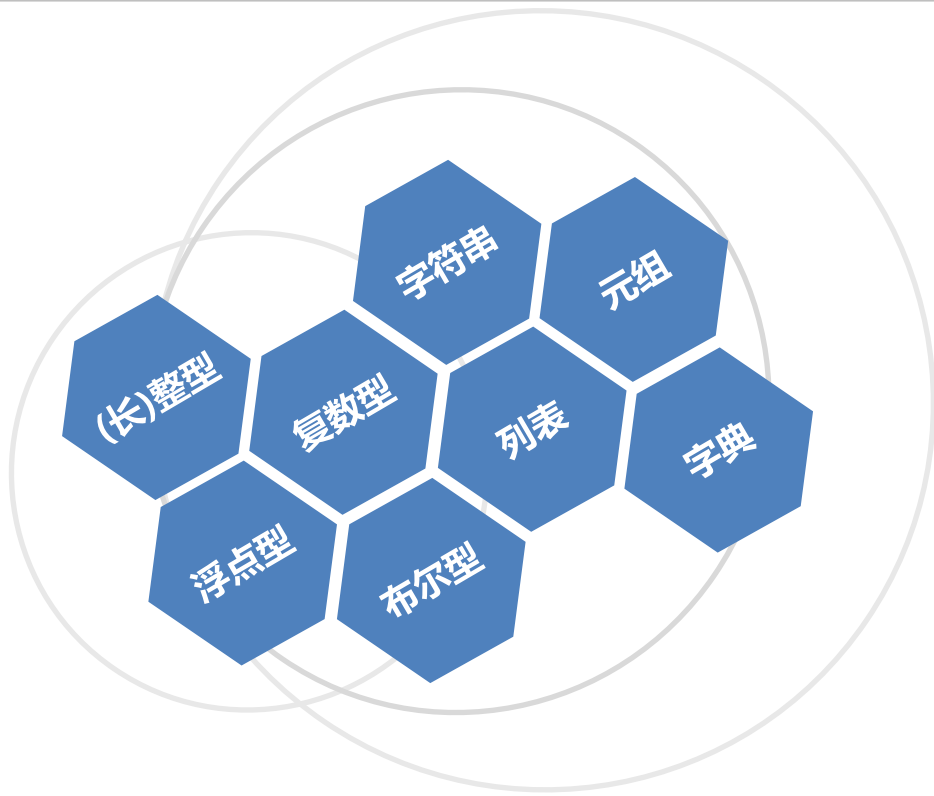
0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

1 0 0 1 1 0 1 1



Python标准数据类型

41



- 整型和长整型并不严格区分
- 整型值后加 “L” 即为长整型




```
>>> # integer  
>>> type(3L)  
<type 'long'>  
>>> type(3)  
<type 'int'>
```

Python 3.x中没有long类型，整数都是int型，但其行为模式与Python 2.x中的long差不多


布尔型

- 整型的子类
- 仅有2个值：True、False
- 本质上是用整型的1、0分别存储的



```
>>> # boolean
>>> x = True
>>> int(x)
1
>>> y = False
>>> int(y)
0
```

- 即数学中的实数
- 可以类似科学计数法表示



```
>>> # float
>>> 3.22
3.22
>>> 9.8e3
9800.0
>>> -4.78e-2
-0.0478
>>> type(-4.78e-2)
<type 'float'>
```

复数型

- $j = \sqrt{-1}$, 则 j 是虚数
- 实数+虚数 就是复数
- 虚数部分必须有 j

Source

```
>>> # complex
>>> 2.4+5.6j
(2.4+5.6j)
>>> type(2.4+5.6j)
<type 'complex'>
>>> 3j
3j
>>> type(3j)
<type 'complex'>
>>> 5+0j
(5+0j)
>>> type(5+0j)
<type 'complex'>
```

- 复数可以分离实数部分和虚数部分
 - 复数.real
 - 复数.imag
- 复数的共轭
 - 复数.conjugate()



```
>>> # complex
>>> x = 2.4+5.6j
>>> x.imag
5.6
>>> x.real
2.4
>>> x.conjugate()
(2.4-5.6j)
```

序列类型

01

字符串

单引号、双引号、三引号内的都是字符串，不可变类型

02

列表

强大的类型，用方括号 [] 界别，可变类型

元组

03

与列表相似，用小括号 () 界别，不可变类型

字符串的表示

- 单引号
- 双引号
- 三引号



```
>>> myString = 'Hello World!'
```

```
>>> print myString
```

```
Hello World!
```

```
>>> myString = "Hello World!"
```

```
>>> print myString
```

```
Hello World!
```

```
>>> myString = """Hello World!"""
```

```
>>> print myString
```

```
Hello World!
```


- 用大括号 {} 界别
- 类似于哈希表的键值对



```
>>> # dictionary
>>> d={'sine':'sin','cosine':'cos','PI':3.14159}
>>> d['sine']
'sin'
```



用Python玩转数据

PYTHON 基本运算

- 算术运算符的优先级

- 乘方**、正负号+ -、
乘除* /、整除//、
取余%、加减+ -

Python 3.x中整型除法返回浮点数，要得到整型结果使用//

Source

```
>>> # arithmetic
>>> pi = 3.14159
>>> r = 3
>>> circum = 2 * pi * r
>>> x = 1
>>> y = 2
>>> z = 3
>>> result1 = x + 3/y - z % 2
>>> result2 = (x + y**z*4)//5
>>> print circum, result1, result2
18.84954 1 6
```

- 数值的比较：按值比大小
- 字符串的比较：按ASCII码值大小



```
>>> # compare
>>> 3 < 4 < 7 # same as ( 3 < 4 ) and ( 4 < 7 )
True
>>> 4 > 3 == 3 # same as ( 4 > 3 ) and ( 3 == 3 )
True
>>> 4 < 3 < 5 != 2 < 7
False
```

Python 3.x中的不等于只用“!=”表示，不支持“<>”



```
>>> # compare
>>> 2 == 2
True
>>> 2.46 <= 8.33
True
>>> 'abc' == 'xyz'
False
>>> 'abc' > 'xyz'
False
>>> 'abc' < 'xyz'
True
```

- 逻辑运算符优先级：
 - not、and、or



```
>>> # logical
>>> x, y = 3.1415926536, -1024
>>> x < 5.0
True
>>> not (x < 5.0)
False
>>> (x < 5.0) or (y > 2.718281828)
True
>>> (x < 5.0) and (y > 2.718281828)
False
>>> not (x is y)
True
>>> 3 < 4 < 7 # same as "( 3 < 4 ) and ( 4 < 7 )"
True
```

字符运算符

- 原始字符串操作符 (r / R) :
 - 用于一些不希望转义字符起作用的地方
- Unicode 字符串操作符(u / U):
 - 转换成Unicode字符串

Source

```
>>> # u
>>> print u'Hello\nWorld'
hello
World
```

Python 3.x中的字符串均为str类型，它与Python 2.x中的Unicode非常相似

Source

```
>>> # r
>>> f = open('c:\python\test.py','w')
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    f = open('c:\python\test.py','w')
IOError: [Errno 22] invalid mode ('w') or filename: 'c:\\python\\test.py'
>>> f = open(r'c:\python\test.py','w')
>>> f = open('c:\\python\\test.py','w')
```



S_{ource}

```
>>> # mix
>>> 3 < 2 and 2 < 1 or 5 > 4
True
>>> x + 3/y - z % 2 > 2
False
>>> 3-2 << 1
2
>>> 3-2 << 1 < 3
True
```

用Python玩转数据



PYTHON的 函数、模块和包

- 函数可以看成类似于数学中的函数
- 完成一个特定功能的一段代码
 - 绝对值函数`abs(x)`
 - 类型函数`type(x)`
 - 四舍五入函数`round(x)`

- 内建函数
 - `cmp()`, `str()` 和 `type()` 适用于所有标准类型

数值型内建函数

<code>abs()</code>	<code>bool()</code>	<code>oct()</code>
<code>coerce()</code>	<code>int()</code>	<code>hex()</code>
<code>divmod()</code>	<code>long()</code>	<code>ord()</code>
<code>pow()</code>	<code>float()</code>	<code>chr()</code>
<code>round()</code>	<code>complex()</code>	

实用函数

<code>dir()</code>	<code>raw_input()</code>
<code>help()</code>	<code>open()</code>
<code>len()</code>	<code>range()</code>

内建函数

59

Built-in Functions				
<code>abs()</code>	<code>divmod()</code>	<code>input()</code>	<code>open()</code>	<code>staticmethod()</code>
<code>all()</code>	<code>enumerate()</code>	<code>int()</code>	<code>ord()</code>	<code>str()</code>
<code>any()</code>	<code>eval()</code>	<code>isinstance()</code>	<code>pow()</code>	<code>sum()</code>
<code>basestring()</code>	<code>execfile()</code>	<code>issubclass()</code>	<code>print()</code>	<code>super()</code>
<code>bin()</code>	<code>file()</code>	<code>iter()</code>	<code>property()</code>	<code>tuple()</code>
<code>bool()</code>	<code>filter()</code>	<code>len()</code>	<code>range()</code>	<code>type()</code>
<code>bytearray()</code>	<code>float()</code>	<code>list()</code>	<code>raw_input()</code>	<code>unichr()</code>
<code>callable()</code>	<code>format()</code>	<code>locals()</code>	<code>reduce()</code>	<code>unicode()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>long()</code>	<code>reload()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>map()</code>	<code>repr()</code>	<code>xrange()</code>
<code>cmp()</code>	<code>globals()</code>	<code>max()</code>	<code>reversed()</code>	<code>zip()</code>
<code>compile()</code>	<code>hasattr()</code>	<code>memoryview()</code>	<code>round()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hash()</code>	<code>min()</code>	<code>set()</code>	<code>apply()</code>
<code>delattr()</code>	<code>help()</code>	<code>next()</code>	<code>setattr()</code>	<code>buffer()</code>
<code>dict()</code>	<code>hex()</code>	<code>object()</code>	<code>slice()</code>	<code>coerce()</code>
<code>dir()</code>	<code>id()</code>	<code>oct()</code>	<code>sorted()</code>	<code>intern()</code>

Python 3.x中的内建函数

60

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()	

S
ource

```
>>> # round-off int
>>> int(35.4)
35
>>> int(35.5)
35
>>> int(35.8)
35
>>> type(int(35.8))
<type 'int'>
```

S
ource

```
>>> # round-off round
>>> round(35.4)
35.0
>>> round(35.5)
36.0
>>> round(35.8)
36.0
>>> type(round(35.8))
<type 'float'>
```

- 非内建函数如何使用？

S
ource

```
>>> # round-off floor
```

```
>>> floor(5.4)
```

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

```
File "<pyshell#0>", line 1, in <module>
```

```
    floor(5.4)
```

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

S
ource

```
>>> # round-off floor
```

```
>>> from math import *
```

```
>>> floor(-35.4)
```

```
-36.0
```

```
>>> floor(-35.5)
```

```
-36.0
```

```
>>> floor(-35.8)
```

```
-36.0
```

模块（二）

- 一个完整的Python文件即是一个模块
 - 文件：物理上的组织方式 `math.py`
 - 模块：逻辑上的组织方式 `math`
- Python通常用 “`import 模块`” 的方式将现成模块中的函数、类等重用到其他代码块中
 - `math.pi`的值可以直接使用，不需要自行定义



```
>>> # module
>>> import math
>>> math.pi
3.141592653589793
```

模块（三）

- 导入多个模块
- 模块里导入指定的模块属性，也就是把指定名称导入到当前作用域

```
>>>import ModuleName  
>>>import ModuleName1, ModuleName2, ...  
>>>from Module1 import ModuleElement
```


包 (package)

- 一个有层次的文件目录结构
- 定义了一个由模块和子包组成的 Python 应用程序执行环境

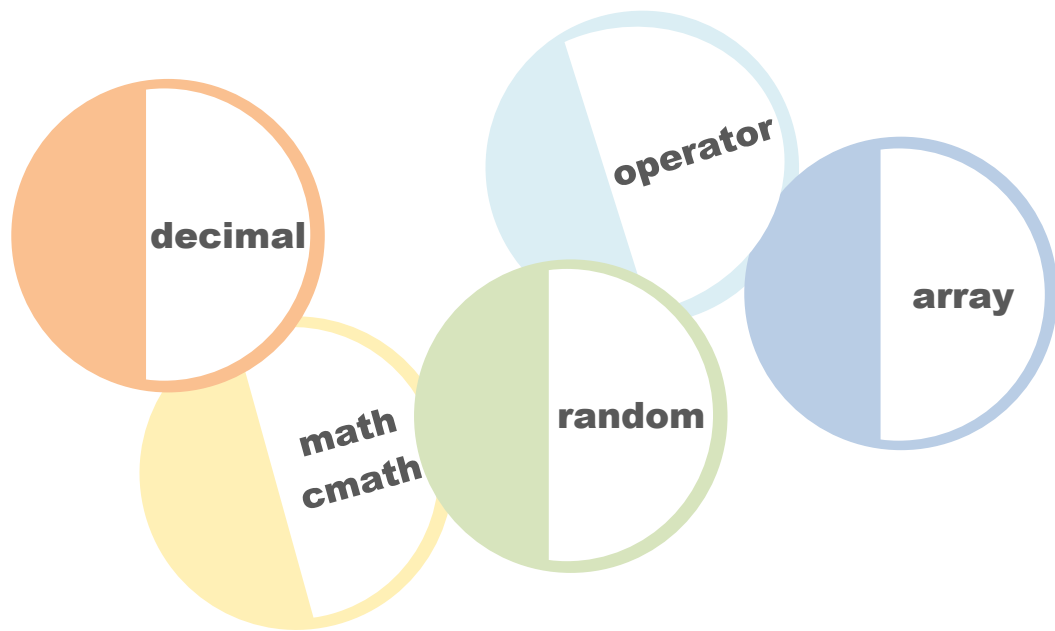
```
>>> import AAA.CCC.c1  
>>> AAA.CCC.c1.func1(123)
```

```
>>> from AAA.CCC.c1 import func1  
>>> func1(123)
```

```
AAA/  
    __init__.py  
    bbb.py  
    CCC/  
        __init__.py  
        c1.py  
        c2.py  
    DDD/  
        __init__.py  
        d1.py  
    EEE/  
    ...
```

库 (library)

- 库是一组具有相关功能的模块的集合
- Python的一大特色就是具有强大的标准库、以及第三方库、以及自定义模块



数值型相关标准库