

# 人工智能程序设计

---

M1 Python程序设计基础

1 走近Python

张 莉

---



《人工智能程序设计》课程专用

# 1 人工智能程序设计 PYTHON简介

# 什么是Python

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

# Python 的历史

- 胶水语言 Glue Language
  - 很容易和其他著名的程序语言连接（如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应用实例



# 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!

```
>>> import this
```

by Tim Peters



人工智能程序设计

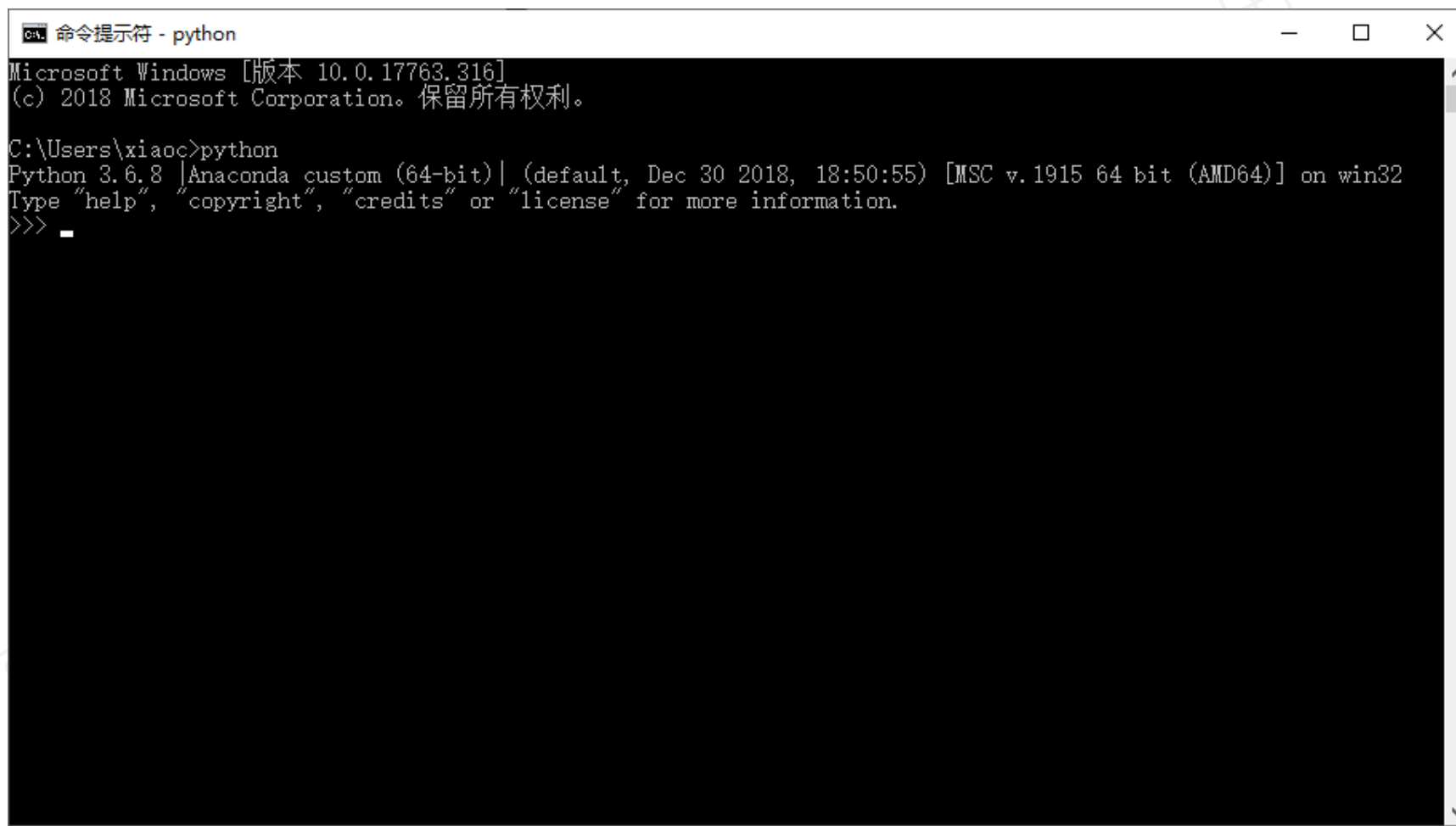
# 2 第一个PYTHON程序

# 经典 Hello World

```
myString = 'Hello, World!'

print(myString)
```

# Python环境-1



```
命令提示符 - python
Microsoft Windows [版本 10.0.17763.316]
(c) 2018 Microsoft Corporation。保留所有权利。

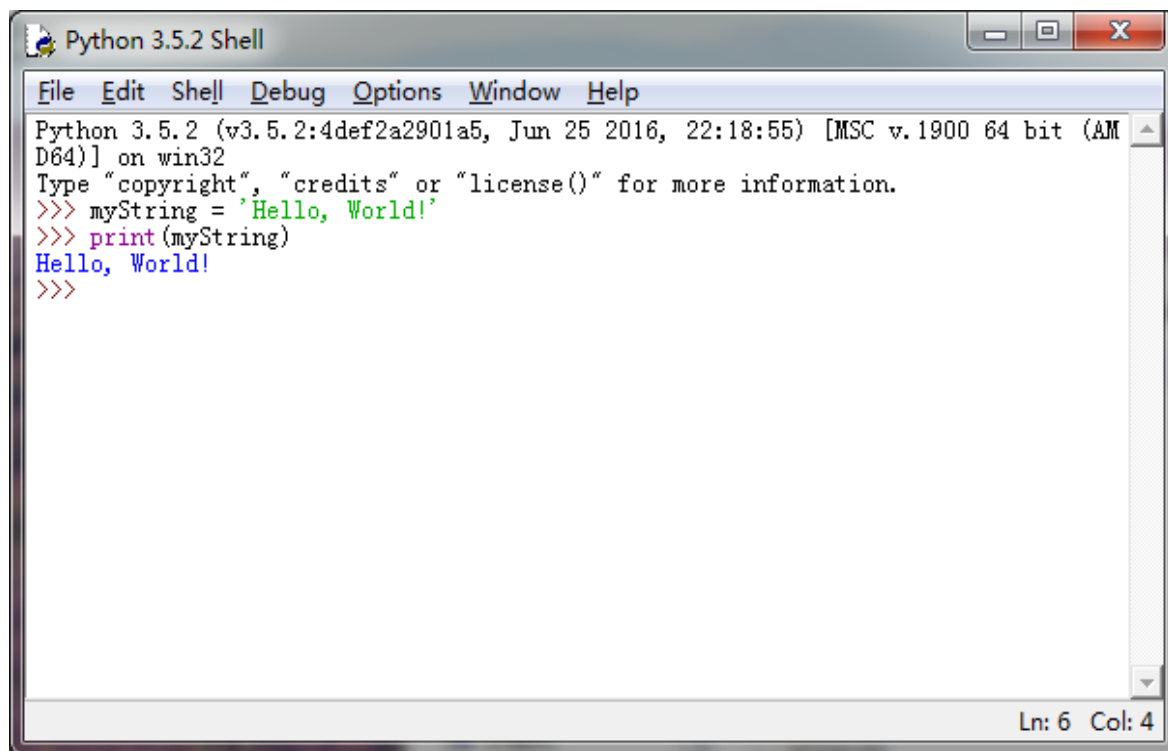
C:\Users\xiaoc>python
Python 3.6.8 |Anaconda custom (64-bit)| (default, Dec 30 2018, 18:50:55) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

# Python环境-2



# Python的运行方式（一）

## Shell方式



The screenshot shows a window titled "Python 3.5.2 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The text area contains the following content:

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> myString = 'Hello, World!'
>>> print(myString)
Hello, World!
>>>
```

The status bar at the bottom right indicates "Ln: 6 Col: 4".

- 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

# Python的运行方式（二）

## 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输出： print函数

- Python使用print函数实现输出：
  - print(变量)
  - print(字符串)



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



# Python输入：input()函数

- input()返回的类型是字符型



```
>>> price = input('input the stock price of Apple:')  
input the stock price of Apple:109
```

```
>>> price  
'109'
```


```
>>> type(price)  
<type 'str'>
```

```
>>> price = int(input('input the stock price of Apple:'))
```


```
>>> price = eval(input('input the stock price of Apple:'))
```

# Python 风格 (一)

## 单行注释



```
>>> # comment No.1
>>> print( 'hello world!' )    # comment No.2
hello world!
```



# Python 风格 (二)

续行

**S**<sub>ource</sub>

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



**S**<sub>ource</sub>

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



# Python 风格 (二)

## 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!""")
```

# Python 风格 (三)

## 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



# Python 风格 (四)

## 缩进

01

增加缩进  
表示语句  
块的开始

Python用相同  
的缩进表示同  
级别语句块

02

减少缩进  
表示语句  
块的退出

03

**S**<sub>ource</sub>

```
>>> # Indentation
```

```
>>> if signal == 'red' and car == 'moving':
```

```
    car = 'stop'
```

```
    singal = 'yellow'
```

```
elif signal == 'green' and car == 'stop':
```

```
    car = 'moving'
```

```
    singal = 'yellow'
```

# 人工智能程序设计 PYTHON语法基础

# 变量



```
>>> # variable
>>> PI = 3.14159
>>> pi = 'circumference ratio'
>>> print(PI)
3.14159
>>> print(pi)
circumference ratio
```



# 关键字

- 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

```
>>> import keyword  
>>> print(keyword.kwlist)
```

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			

# 变量的使用

- 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



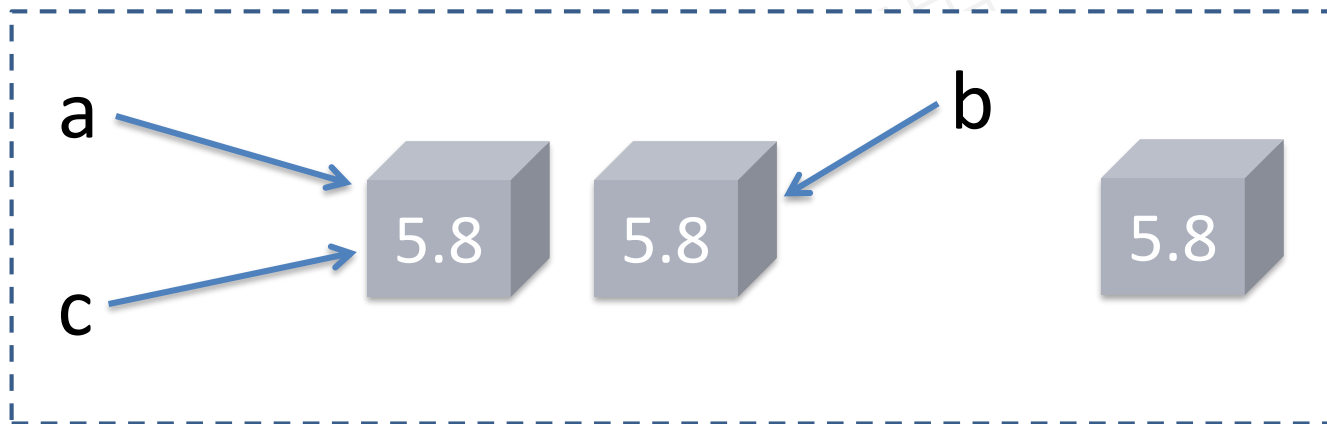
```
>>> # Identifier
>>> PI = 3.14159
>>> pi = 'one word'
>>> print(PI)
3.14159
>>> print(pi)
one word
```

# 变量的管理

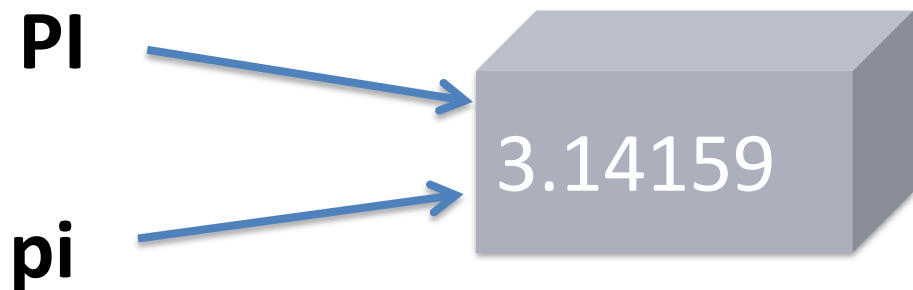
## 动态类型

### • 引用计数

- $a = c$ , 指向了相同的数据对象
- $b$ 和 $a$ 创建的是不同的5.8对象
- 单独 $\text{id}(5.8)$ 是创建的全新的对象



# 变量的管理



**S**<sub>ource</sub>

```
>>> p = 3
>>> q = 3
>>> p is q
True
```

**S**<sub>ource</sub>

```
>>> PI = 3.14159
>>> pi = 3.14159
>>> PI is pi
False
>>> pi = PI
>>> print(PI)
3.14159
>>> pi is PI
True
```

图中的形式用  
哪个语句可以  
表示?

is运算符的基  
础是id()函数

特殊情况?

# 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	>>

## 关系运算符

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

# 运算符



```
>>> 1 / 2
>>> 1 // 2
>>> 123 % 10
>>> 123 // 10 % 10
>>> 3 != 5
>>> x = 5
>>> 3 < x < 7
```

- 算术运算符 > 位运算符 > 关系运算符 > 逻辑运算符
- 算术运算符的优先级
  - $**$  >  $+$   $-$  (正负号) >  $*$   $/$   $//$   $\%$  >  $+$   $-$
- 逻辑运算符的优先级
  - $\text{not}$  >  $\text{and}$  >  $\text{or}$

# 赋值语句 增量赋值

增量赋值操作符

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



>>> *# Augmented assignment*

>>> m = 18

>>> m /= 5

>>> m

3.6

# 赋值 链式赋值



```
>>> # Chained assignment
```

```
>>> a = 1
```

```
>>> b = a = a + 1
```

```
>>> b
```

```
2
```

```
>>> a
```

```
2
```



# 赋值 多重赋值



```
>>> # Multiple assignment
```

```
>>> PI, r = 3.1415, 3
```

```
>>> PI
```

```
3.1415
```

```
>>> r
```

```
3
```

```
>>> x, y = 3, 5
```

```
>>> x, y = y, x
```

# 4 PYTHON数据类型

人工智能程序设计

《人工智能程序设计》课程专用

# Python数据类型



# 整型


- 整型和长整型并不严格区分
- Python 2支持整型值后加  
“L” 即为长整型



```
>>> # integer  
>>> type(3)  
<class 'int'>
```

# 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

 Source

```
>>> # boolean
>>> x = True
>>> type(x)
<class 'bool'>
>>> int(x)
1
>>> y = False
>>> int(y)
0
```

# 浮点型

- 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)
```

```
<class '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 $j$

  
>>> *# complex*  
>>> 2.4+5.6j  
(2.4+5.6j)  
>>> type(2.4+5.6j)  
<class 'complex'>

  
>>> *# complex*  
>>> 3j  
3j  
>>> type(3j)  
<class 'complex'>  
>>> 5+0j  
(5+0j)  
>>> type(5+0j)  
<class 'complex'>

# 5 人工智能程序设计 PYTHON函数，模块与包



# Python中的函数



# 内建函数

```
>>> dir(__builtins__)
```

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	<u>input()</u>	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()	

# 内建函数

abs()	bool()	oct()
round()	int()	hex()
divmod()	ord()	pow()
float()	chr()	complex()

数值型内建函数

dir()	input()
help()	open()
len()	range()

实用函数

# 内建函数

Source

```
>>> # round-off int
```

```
>>> int(35.4)
```

```
35
```

```
>>> int(35.5)
```

```
35
```

```
>>> int(35.8)
```

```
35
```

```
>>> type(int(35.8))
```

```
<class 'int'>
```

Source

```
>>> # ord
```

```
>>> ord('3')
```

```
51
```

```
>>> ord('a')
```

```
97
```

```
>>> ord('\n')
```

```
10
```

```
>>> type(ord('A'))
```

```
<class 'int'>
```

# 其他方式定义的函数

A

**标准库函数：**需要先导入模块再使用函数，每个库有相关的一些函数如math库中的sqrt()函数

B

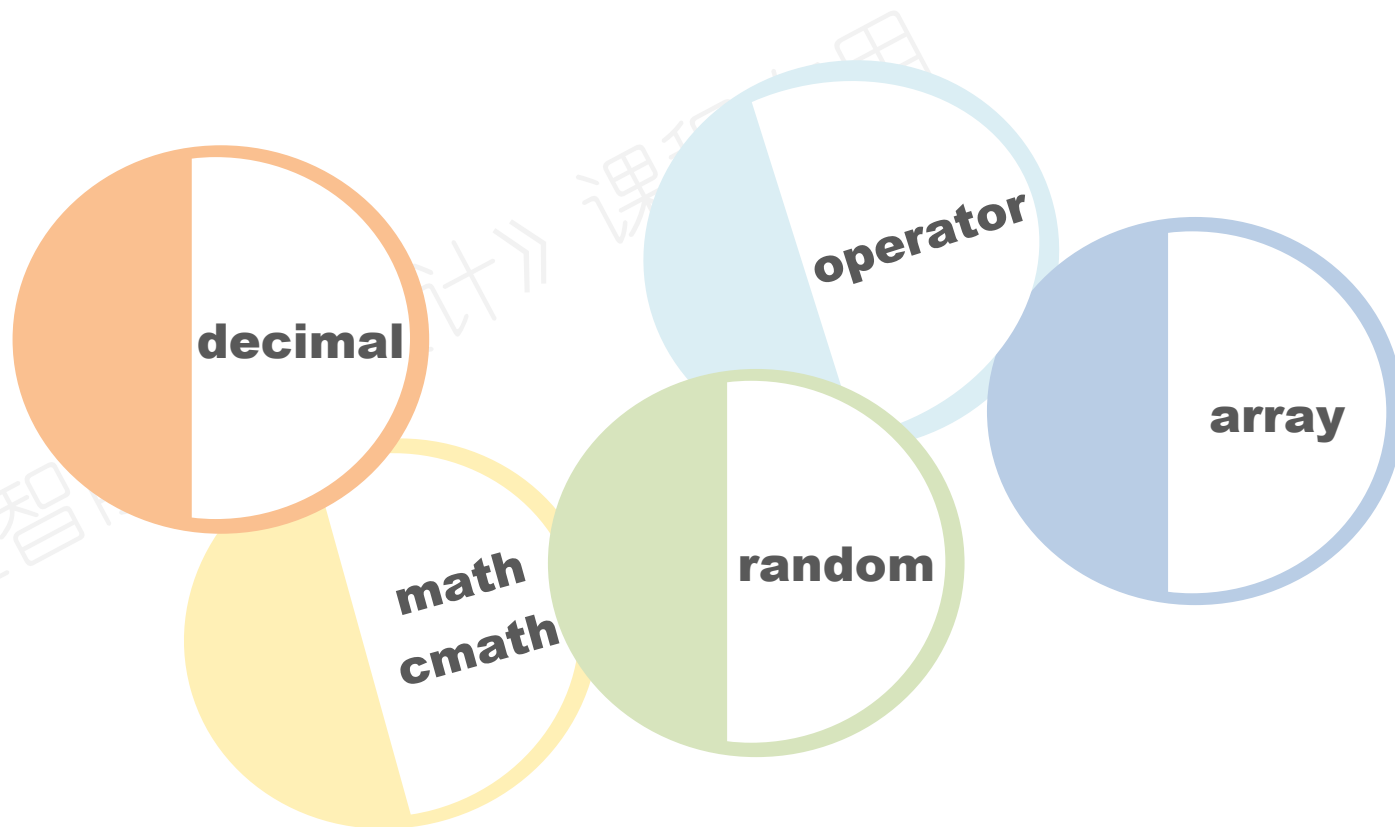
**第三方库函数：**数量非常惊人，这也是Python重要的特征和优势，例如著名的科学计算包SciPy中就包含了很多用于科学计算的函数

C

**用户自定义函数：**有固定的定义、调用和参数传递方式等

# 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

# 模块 (一)

- 非内建函数如何使用?

Source

```
>>> # 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

Source

```
>>> # round-off floor
>>> from math import *
>>> floor(-35.4)
-36
>>> floor(-35.5)
-36
>>> floor(-35.8)
-36
```

## 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



```
>>> 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
```



```
>>> import math
>>> math.log(math.e)
1.0
>>> math.sqrt(9)
3.0
>>> from math import floor
>>> floor(5.4)
5
```

# 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

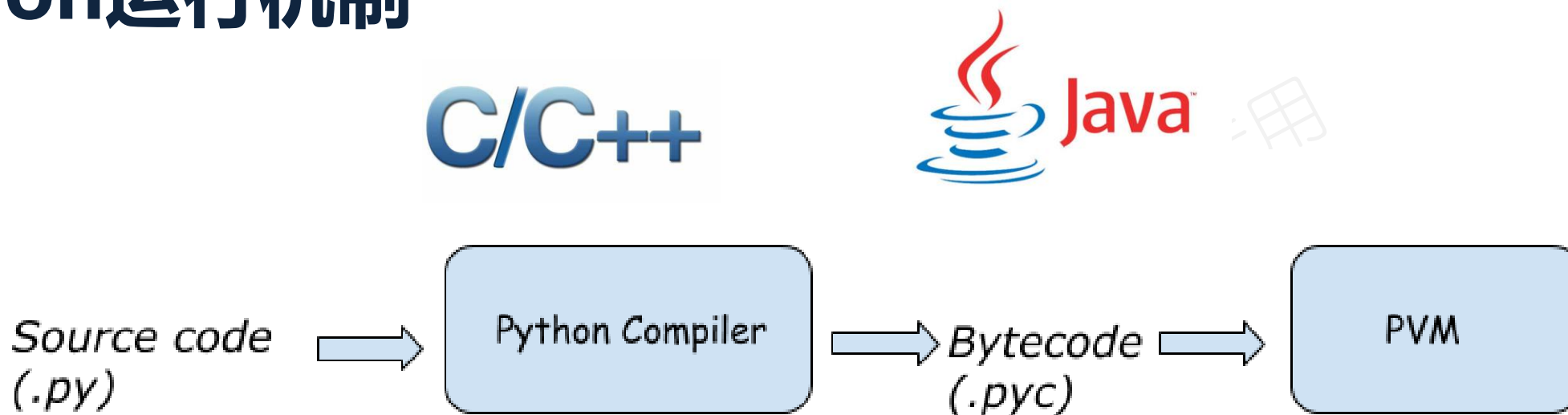
```
>>> import AAA.CCC.c1  
>>> AAA.CCC.c1.func1(123)
```

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

```
AAA/  
    __init__.py  
    b.py  
    CCC/  
        __init__.py  
        c1.py  
        c2.py  
    DDD/  
        __init__.py  
        d1.py  
        d2.py  
    ...
```

# Python的运行机制及打包

# Python运行机制



*From: <https://www.codecompiled.com/>*

什么时候会产生.pyc文件?

```
>>> import py_compile
>>> py_compile.compile('c:/users/xiaoc/test.py')
'c:/users/xiaoc\\__pycache__\\test.cpython-37.pyc'
```

*See more: <https://python-compiler.com/post/how-to-compile-py-to-pyc>*

# py文件打包成exe文件

## PyInstaller Quickstart

- Install PyInstaller from PyPI:

```
$ pip install pyinstaller -i https://pypi.tuna.tsinghua.edu.cn/simple
```

- Go to your program's directory and run:

```
$ pyinstaller yourprogram.py
```

- This will generate the bundle in a subdirectory called **dist**.

*See more: <http://www.pyinstaller.org/>*

# M1.1小结

- 01 Python简介
- 02 第一个Python程序
- 03 Python语法基础
- 04 Python数据类型
- 05 Python函数，模块与包