

Python程序设计与数据科学导论

信息科学技术学院

胡俊峰



课程的定位：

- 面向计算机、数据科学及相关专业的专业选修课。
- 覆盖编程技术与实现、问题建模方案及数学原理。
- 侧重数据挖掘、机器学习方向的应用技术练习。



课程先修要求

- 学习过基础python编程
- 了解面向对象编程基础知识
- 具备一定的高等代数（向量空间模型及线性变换），高等数学（导数、偏导数、卷积）及 概率统计（概率、条件概率、贝叶斯分析、假设检验）等相关基础



课程主要内容及安排（22次课）

- Python程序设计（5次课 作业+2）
- Numpy、Pandas 技术与建模（4次课 作业+2）
- 机器学习原理与经典模型实践（3次课 作业+1）
- 图像处理、自然语言处理、时间序列分析原理与技术（3次课 作业+1）
- 期中作业讲评交流（1次课）
- 深度学习模型原理与Pytorch编程基础（2次课 作业+1）
- 卷积神经网络、递归神经网络模型及问题建模（2次课 作业+1）
- 预训练模型与应用（1次课）
- 期末交流与总结（1次课）



考试及成绩认定

- 平时作业成绩：20%（8次）
- 期中成绩（大作业）：30%
- 期末考试（笔试）-大作业（6% 可选）：50%



作业批改与成绩评定：

第一次作业	第二次作业	第三次作业	第四次作业	第五次作业	第六次作业	第七次作业
9.00	9.50	10.00	10.00	9.00	10.00	10.00
9.00	10.00	10.00	11.00	10.00	9.00	10.00
9.00	10.00	11.00	12.00	10.00	9.00	9.50
10.00	10.00	11.00	11.00	12.00	9.00	10.00
9.50	10.00	11.00	12.00	12.00	8.50	10.00
10.00	10.00	11.00	10.50	11.00	9.00	10.00
9.00	10.00	11.00	10.00	9.00	9.00	10.00
10.00	10.00	10.00	10.00	10.00	8.50	10.00
9.50	10.00	10.00	11.00	12.00	9.00	10.00
7.00	10.00	10.00	11.00	11.00	9.00	8.00

五分制

选做加分0.5-1分，
累计不超过4分



课程组织形式：

- 课堂授课
- 课后练习-作业-大作业及交流讲评
- 每次课后都有课堂小练习（作为作业的一部分提交）
 - 认真及时完成每一次课后练习是课程方案的重要环节。对于理解后继课程内容至关重要。
- 每个阶段内容（2-3次课）安排一次评分作业，会包含选做题。
- 作业会一般在对应阶段课程的第一次课后发布



教学团队：

- 任课教师：胡俊峰
- 助教团队：刘华秋、王瑞环、熊江凯

群聊：2024Python与数据科学导论



该二维码7天内(2月25日前)有效，重新进入将更新



课程软件环境：

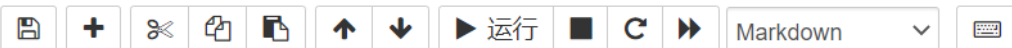
- Anaconda + python, pytorch等相关软件包
- 平时作业用Jupyter notebook文件形式布置和提交



作业示例

不可信

Python 3



numpy矩阵操作

首先请你创建三个矩阵 A, B, C, D , 它们都是服从标准正态分布的矩阵, 其中 A 的大小为 20×40 , B 的大小为 40×40 , C 的大小为 40×1 , D 的大小为 40×1 .

```
In [45]: import numpy as np
np.random.seed(1)
# 作业: 添加你的代码
```

Q1.1 计算 $A + A, AA^T, A^T A, AB$. 然后写一个函数对于输入参数 λ , 计算 $A(B - \lambda I)$.

```
In [46]: # 作业: 添加你的代码
```

Q1.2 计算 $DB^{-1}x = C$

```
In [47]: # 作业: 添加你的代码
```

Q1.3 请通过numpy的条件约束实现RELU函数, 并输出RELU(A)

RELU参考: <https://zh.wikipedia.org/wiki/%E7%BA%BF%E6%80%A7%E6%95%B4%E6%B5%81%E5%87%BD%E6%95%B0>

```
In [48]: # 作业: 添加你的代码
```

课程参考资料：

- [3.12.2 Documentation \(python.org\)](https://docs.python.org/3.12.2/) (Python官方文档及教程)
 - [The Python Tutorial — Python 3.12.2 documentation](https://docs.python.org/3.12.2/tutorial/)
 - Numpy, Pandas网上学习参考材料
 - 其他对应软件包网上学习材料
- <https://github.com/jakevdp/PythonDataScienceHandbook> (数据科学教程)
 - <https://github.com/cuttlefishh/python-for-data-analysis> (数据分析技术教程)
- <https://pytorch.org/tutorials/beginner/basics/intro.html> (Pytorch官方基础教程)
 - https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

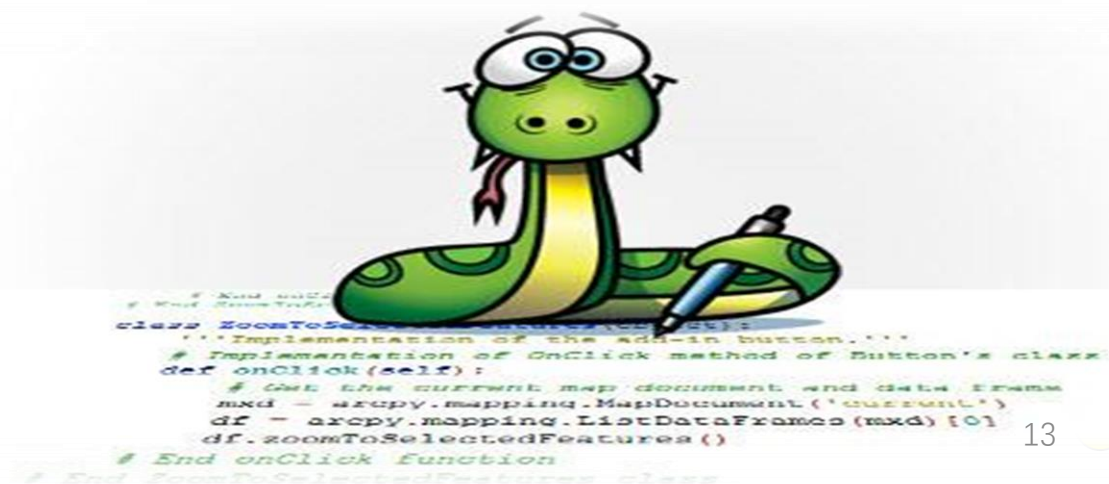


Python语言基础 C01

第一讲 Python 之 数据类型与容器类

Python语言的特色与课程教学目标

- 是一种胶水（标签-认领）型语言
 - 能够支持、派生出丰富多样的周边软件包体系
- 具有自己独特的堪称典范的语言机制
 - 数据交换机制
 - 软件融入机制



Python很容易学么？

- 入门相对容易，真正理解掌握比较难
- 写Python代码，需要查文档或上网搜索是正常情况
- Python程序写的慢不见得是坏事。代码逻辑清晰，可读性好很重要。



本次课内容提要

- Jupyter notebook简介
- Python的基本语言类型与表达式
- Python的容器类
- 基于数据流的计算：
 - 迭代器、生成器（函数）



Jupyter notebook:

一种混合了描述文本和代码实现的交互编程环境



什么是Jupyter Notebook

Jupyter Notebook是基于**网页**的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。

优势

- 轻便
- 交互性好，同时支持文本和代码
- 可以在服务器上部署，远程登录使用

基于B/S架构的软件在线编程环境。

劣势

- 功能简单，不适合大规模工程项目

In [1]: `#演示交互性`
`string = '人生苦短，我学Python'`

In [2]: `print(string)`

人生苦短，我学Python

```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Py
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 1
Type "help", "copyright", "credits" or "license" fo
>>> a = [1,2,'hello',3.9]
>>> b = a[3:]
>>> print(b)
[3.9]
>>>
```

基于本地服务 (local host) 的jupyter notebook

```
命令提示符 - jupyter notebook

Microsoft Windows [版本 10.0.18362.592]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\lajiaojiang>jupyter notebook
[I 17:46:39.322 NotebookApp] [jupyter_nbextensions_configurator] enabled 0.4.1
[I 17:46:39.366 NotebookApp] JupyterLab extension loaded from D:\anaconda\lib\site-packages\jupyterlab
[I 17:46:39.366 NotebookApp] JupyterLab application directory is D:\anaconda\share\jupyter\lab
[I 17:46:39.370 NotebookApp] Serving notebooks from local directory: E:/Jupyter
[I 17:46:39.370 NotebookApp] The Jupyter Notebook is running at:
[I 17:46:39.370 NotebookApp] http://localhost:8888/?token=8b43fb6b40f87511a583e61df312717620612aaf1c8d3c87
[I 17:46:39.370 NotebookApp] or http://127.0.0.1:8888/?token=8b43fb6b40f87511a583e61df312717620612aaf1c8d3c87
[I 17:46:39.370 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 17:46:39.414 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/lajiaojiang/AppData/Roaming/jupyter/runtime/nbserver-628-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=8b43fb6b40f87511a583e61df312717620612aaf1c8d3c87
    or http://127.0.0.1:8888/?token=8b43fb6b40f87511a583e61df312717620612aaf1c8d3c87
```

File Edit View Insert Cell Kernel Widgets Help

可信的

Python 3



Cut Cells

Copy Cells

Paste Cells Above

Paste Cells Below

Paste Cells & Replace

Delete Cells

Undo Delete Cells

Split Cell

Merge Cell Above

Merge Cell Below

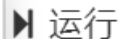
Move Cell Up

Move Cell Down

Edit Notebook Metadata

Find and Replace

Cut Cell Attachments



运行



标记



ter Notebook

是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。

支持文本和代码

部署，远程登录使用

适合大规模工程项目(可通过安装拓展包弥补部分功能)

告短，我学Python'

Notebook 编辑与常用快捷键

两种编辑模式:

内容编辑:

单元编辑:

`ctrl+enter` 运行当前cell;

`shift+enter` 运行当前cell并跳到下一个cell ;

`ctrl+[` 向左缩进;

`ctrl+]` 向右缩进

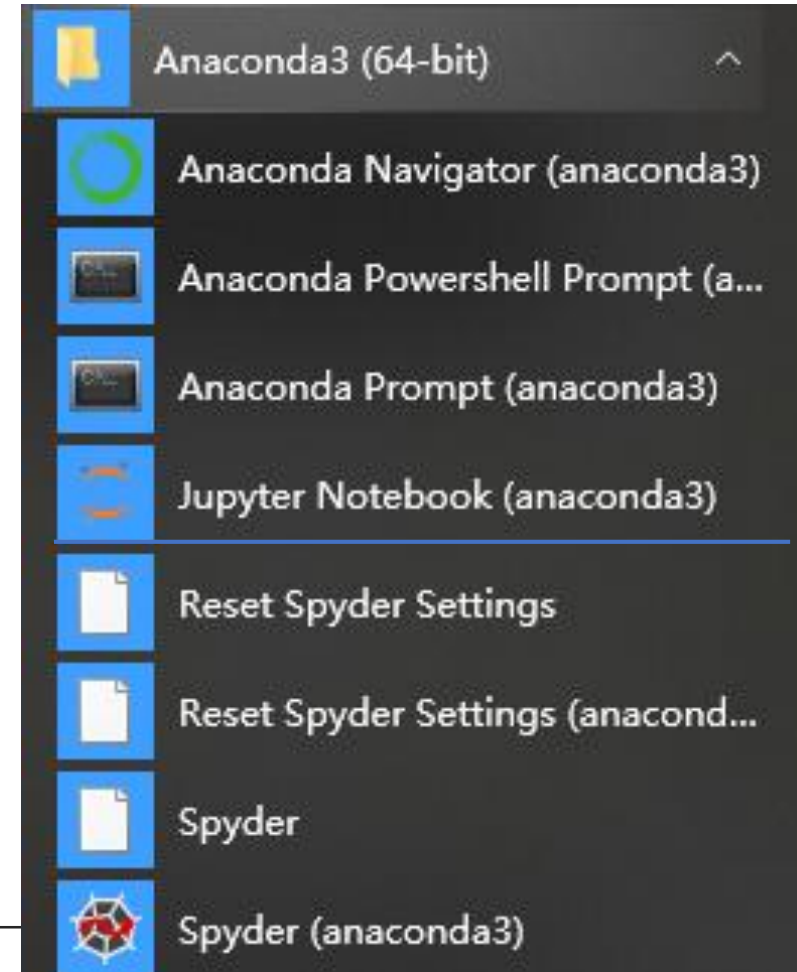


Anaconda | The World's Most Popular Data Science Platform

<https://www.anaconda.com> ▼

Supercharge your data science
efforts with **Anaconda**.

Get Started



命令行运行jupyter notebook --generate-config生成配置文件

```
C:\> 命令提示符 - jupyter notebook --generate-config

Requirement already satisfied: python-dateutil>=2.1 in d:\anaconda\lib\site-packages (from jupyter-client>=5.3.1->notebook>=4.0->jupyter_contrib_nbextensions) (2.8.0)
Requirement already satisfied: jedi>=0.10 in d:\anaconda\lib\site-packages (from ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (0.15.1)
Requirement already satisfied: backcall in d:\anaconda\lib\site-packages (from ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (0.1.0)
Requirement already satisfied: colorama; sys_platform == "win32" in d:\anaconda\lib\site-packages (from ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (0.4.1)
Requirement already satisfied: pickleshare in d:\anaconda\lib\site-packages (from ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (0.7.5)
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in d:\anaconda\lib\site-packages (from ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (2.0.10)
Requirement already satisfied: attrs>=17.4.0 in d:\anaconda\lib\site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert>=4.2->jupyter_contrib_nbextensions) (19.2.0)
Requirement already satisfied: pyparsing>=2.0.2 in d:\anaconda\lib\site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert>=4.2->jupyter_contrib_nbextensions) (2.4.6)
Requirement already satisfied: pyrsistent>=0.14.0 in d:\anaconda\lib\site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert>=4.2->jupyter_contrib_nbextensions) (0.15.4)
Requirement already satisfied: parso>=0.5.0 in d:\anaconda\lib\site-packages (from jedi>=0.10->ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (0.5.1)
Requirement already satisfied: wcwidth in d:\anaconda\lib\site-packages (from prompt-toolkit<2.1.0,>=2.0.0->ipython->jupyter-latex-envs>=1.3.8->jupyter_contrib_nbextensions) (0.1.7)

C:\Users\lajiaojiang>jupyter notebook --generate-config
Overwrite C:\Users\lajiaojiang\.jupyter\jupyter_notebook_config.py with default config? [y/N]
```


修改默认工作目录及其他配置：

```
## Dict of Python modules to load as notebook server extensions.Entry values can  
# be used to enable and disable the loading of the extensions. The extensions  
# will be loaded in alphabetical order.  
#c.NotebookApp.nbserver_extensions = {}  
  
## The directory to use for notebooks and kernels.  
c.NotebookApp.notebook_dir = 'E:/Jupyter'  
  
## Whether to open in a browser after starting. The specific browser used is
```



Python语言的基础部分：

- 基本数据类型
- 表达式
- 分支流程



Python的基础数据类型

- 数值类型： 整数、浮点数、（复数）
- 字符串
- 布尔类型

形式上基本继承了C语言的表达方案，但本质上是面向对象的。
底层有比较复杂的实现规范。



整数

[illegible][illegible]

```
1 i2 = 107
2 print(i2 == 107) # 由于上不封顶，实际实现是通过一个类似 $2^{63}-1$ 取模这样的hash操作来实现的
```

True

```
1 print(i2 is 107)  # 3.8以后, 建议采用 ==, != 代替 is 和 is not
```

True

```
<>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
<>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
<ipython-input-48-971acdacfad0>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
    print(i2 is 107)
```

Integer Objects

All integers are implemented as “long” integer objects of arbitrary size.

On error, most `PyLong_As*` APIs return `(return_type)-1` which cannot be distinguished from a number. Use `PyErr_Occurred()` to disambiguate.

```
type PyLongObject
```

Part of the *Limited API* (as an opaque struct).

This subtype of `PyObject` represents a Python integer object.

PyTypeObject PyLong_Type

Part of the Stable ABI.

This instance of `PyTypeObject` represents the Python integer type. This is the same object as `int` in the Python layer.

```
int PyLong_Check(PyObject *p)
```

Return true if its argument is a `PyLongObject` or a subtype of `PyLongObject`. This function always succeeds.



浮点数

```
num1 = 101.1                # 64位浮点数编码
num2 = 100.000000000000001  # 会产生精度误差
print(num1-num2)
num1 = 101.000000000000001
num2 = 100.000000000000001  # 会有截断误差
print(num1-num2)
num2 = 1E-200               # 科学计数法
print(num1 / num2)
```

```
1.09999999999999943
1.0
1.01e+202
```

Integers可以任意长度，由内存大小决定
floating point number精确到15位十进制小数



算术表达式运算:

表达式计算

```
1 (10 + 12) // 4 % 3 # 计算器 // 表示整除 % 取模
```

2

变量类型?

变量类型不需
预先定义

```
3 a = 2 + 10 # 变量名直接使用, 解释性语言特色
4 print("a=", (a + 1) / 3) # 表达式做函数参数
   b = 3**2 + 1.3 # ** 乘方运算
   print("b=", b)
```

a= 4.333333333333333

b= 10.3



内置算数运算	结果	注释	完整文档
<code>x + y</code>	<code>x</code> 和 <code>y</code> 的和		
<code>x - y</code>	<code>x</code> 和 <code>y</code> 的差		
<code>x * y</code>	<code>x</code> 和 <code>y</code> 的乘积		
<code>x / y</code>	<code>x</code> 和 <code>y</code> 的商		
<code>x // y</code>	<code>x</code> 和 <code>y</code> 的商数	(1)	
<code>x % y</code>	<code>x / y</code> 的余数	(2)	
<code>-x</code>	<code>x</code> 取反		
<code>+x</code>	<code>x</code> 不变		
<code>abs(x)</code>	<code>x</code> 的绝对值或大小		<code>abs()</code>
<code>int(x)</code>	将 <code>x</code> 转换为整数	(3)(6)	<code>int()</code>
<code>float(x)</code>	将 <code>x</code> 转换为浮点数	(4)(6)	<code>float()</code>
<code>complex(re, im)</code>	一个带有实部 <i>re</i> 和虚部 <i>im</i> 的复数。 <i>im</i> 默认为0。	(6)	<code>complex()</code>
<code>c.conjugate()</code>	复数 <code>c</code> 的共轭		
<code>divmod(x, y)</code>	<code>(x // y, x % y)</code>	(2)	<code>divmod()</code>
<code>pow(x, y)</code>	<code>x</code> 的 <code>y</code> 次幂	(5)	<code>pow()</code>
<code>x ** y</code>	<code>x</code> 的 <code>y</code> 次幂	(5)	



字符串（常量）

```
'''
```

单引号-双引号 表达字符串常量
连续三个引号开启一段注释

```
'''
```

```
st1 = ' "Hello"'
```

```
st2 = "Python!"
```

```
print(st1+ ' ' + st2)
```

"Hello" Python!

```
: str = 'Do U have a nice vacation?'
```

```
str += '\n' + "No I don't."
```

反斜杠转义字符

```
print (str)
```

```
# 第一个例子 Simple output (with Unicode)
str1 = "人生苦短，我用Python!"

str1
```

'人生苦短，我用Python!'

```
str2 = '用了Python。' \ ←
      '更觉人生苦短。'

str2
```

'用了Python。更觉人生苦短。'

```
str3 = '''
Python虽然好，可惜不牢靠。
模块各自乱升级，代码隔年跑不了，平空添烦恼。
''' # 多行连续字符串，也可以用3个单引号括多行起来

print(str3)
```

Python虽然好，可惜不牢靠。
有事没事乱升级，代码隔年跑不了，平空添烦恼。



布尔类型与布尔表达式运算:

Booleans 布尔类型

```
1 t, f = True, False ← 布尔常量
2 print (type(t)) # Prints "<type 'bool'>"
```

<class 'bool'>

```
1 print (t and f)    # Logical AND;
2 print (t or f)     # Logical OR;
3 print (not t)      # Logical NOT;
4 print (t != f)     # Logical XOR;
```

False

True

False

True



布尔表达式以及短路运算

```
a = 0    # 类型可以自定义 __bool__() 返回的布尔值, len() = 0 对应 false, 其他 True
print(not a)
```

True

```
s1 = '123'
s2 = s1 or 'abc'    # or, and 为短路运算 不强制返回布尔值
print(s2)
```

```
s1 = ''
s2 = s1 or 'abc'
s2
```

123

'abc'

python中的对象

- 唯一的标识码(identity)
- 类型
- 内容（或称为值）

一旦对象被创建，它的标识码就不允许更改。对象的标识码可以有内建函数id()获取，它是一个整型数。您可以将它想象为该对象在内存中的地址，其实在目前的实现中标识码也就是该对象的内存地址。

- 当用is操作符比较两个对象时，就是在比较它们的标识码。更确切地说，is操作符是在判断两个对象是否是同一个对象。
- 当用 == 操作符比较两个对象，是在比较他们的值。

```
: a = 2.0  
  b = 2.0  
  
print(a is b)  
print(a == b)
```

False
True



Python的赋值与引用绑定

- 名字(id)是对一个对象的称呼，给对象取一个名字的操作叫作命名，python将赋值语句认为是一个命名操作（或者称为名字绑定）。
- 而对于对象来说，每绑定一个名字，就在属性的引用计数上进行+1操作。
每取消一个绑定就把引用计数-1.
- 环境会自动检测对象的引用计数来进行垃圾回收。因此，python的对象一般不需要显式的进行释放操作。



条件表达式与分支 (if-elif) 语句

- 布尔类型与逻辑表达式
 - not ; and; or
- 条件表达式：返回布尔值
- 分支流程（语句）



比较（关系）运算

<	严格小于
<=	小于或等于
>	严格大于
>=	大于或等于
==	等于
!=	不等于
is	是同一个对象（标识）
is not	不是同一个对象（标识）



第一部分：基础练习

1.1 基本数据类型，表达式，简单输入输出

输入三角形的三边长，计算三角形的面积。如果输入不合法，请输出'Whoops!'

例如：输入：2 输出：1.7320508075688772

2

2

```
|: a = float(input())
b = float(input())
c = float(input())
if (a+b<c) or (b+c<a) or (c+a<b):
    print('Whoops!')
else:
    p = (a+b+c)/2
    s = (p*(p-a)*(p-b)*(p-c))**0.5
    print(s)
```

2

3

4

2.9047375096555625



运算符优先级	描述
<code>:=</code>	赋值表达式
<code>lambda</code>	lambda 表达式
<code>if -- else</code>	条件表达式
<code>or</code>	布尔逻辑或 OR
<code>and</code>	布尔逻辑与 AND
<code>not x</code>	布尔逻辑非 NOT
<code>in, not in, is, is not, <, <=, >, >=, !=, ==</code>	比较运算，包括成员检测和标识号检测
<code> </code>	按位或 OR
<code>^</code>	按位异或 XOR
<code>&</code>	按位与 AND
<code><<, >></code>	移位
<code>+, -</code>	加和减
<code>*, @, /, //, %</code>	乘，矩阵乘，除，整除，取余 [5]
<code>+x, -x, ~x</code>	正，负，按位非 NOT
<code>**</code>	乘方 [6]



小结一下：

- Python语言基本继承了传统C的变量类型，表达式计算及流程方案。语言格式规范上有调整。
- 在此基础上理解、实现传统的编程方案基本没有太大困难。
- 效率会比较低。



Python的序列类型与顺序计算流程

- 序列类型：
 - 列表 List, 元组tuple, range,
 - 字符串(str)类型
 - 字节流类型



列表 List

- 本质：对象引用的序列
- 特点：类型兼容性好、结构灵活、可容纳复合类型



线性序列类型之一：列表

- **List (列表)**

不同类型能混在一起

```
1 a = [1, 2, 'aaa', 1.23]
2
3 print('a = ', a)
4 print("a[1] =", a[1], "; a[2] =", a[2])
5 print('len(a) = ', len(a))
```

看到的数据对象并不存放在list结构中，实际存放的是引用列表

```
a = [1, 2, 'aaa', 1.23]
a[1] = 2 ; a[2] = aaa
len(a) = 4
```

下标访问为按对应的引用取出值

列表元素的赋值： 指向新的对象

```
1 a = [1, 2, 'aaa', 1.23]
2 b = a ← 添加一个新的引用b
3 print(b)
```

[1, 2, 'aaa', 1.23]

```
1 b[1] = 'bbb' # 此时修改的是将2号单元的应引用指向新的字符串对象'bbb'
2 print(a)
```

[1, 'bbb', 'aaa', 1.23]



- List对象常用方法：增、删、查、改

添加元素 `append()`

```
1 a = []          # 空列表
2
3 a.append(20.5)
4 a.append(30)
5 a.append('hello wrold!')
6 print(a)
```

← 尾部添加元素

[20.5, 30, 'hello wrold!']

插入元素 `insert()`

```
1 a.insert(2, 'beautiful')
2 print(a)
3
4 a.insert(-1, 24)
5 print(a)
```

← 第3个位置（下标从0开始）处插入元素

← 倒数第2个位置处插入元素

[20.5, 30, 'beautiful', 'hello wrold!']
[20.5, 30, 'beautiful', 24, 'hello wrold!']



► List对象常用方法：查找、定位

判断是否在list中

```
1 a = [1, 3, 5, 7, 9]
2 print(1 in a)
3
4 d = a.index(5)
5 print ('a.index(5) = ', d)
6 # print ('a.index(4) = ', a.index(4)) # ValueError: 4 is not in list
```

True

a.index(5) = 2

其他常用方法：计数元素个数、排序sort、逆序reverse、清空列表clear 等

```
1 fruits = ['orange', 'apple', ['apple', 'banana'], 'kiwi', 'apple', 'banana']
2 fruits.count('apple')
```

list.pop(*i*)

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.clear()

Remove all items from the list. Equivalent to `del a[:]`.

list.index(*x*[, *start*[, *end*]])

Return zero-based index in the list of the first item whose value is equal to *x*. Raises a `ValueError` if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

list.count(*x*)

Return the number of times *x* appears in the list.

list.sort(*, *key*=None, *reverse*=False)

Sort the items of the list in place (the arguments can be used for sort customization, see `sorted()` for their explanation).

list.reverse()

Reverse the elements of the list in place.

list.copy()

Return a shallow copy of the list. Equivalent to `a[:]`.

线性序列类型之：元组

- Immutable（不可修改）对象：创建之后只能访问，不能修改。
- tuple, frozenset
- mutable（可修改）对象：list, set, int, float...



➡ Tuple Tuple (元组)

- 和List不同在于，一旦初始化后不可更改
- 元组虽不可变，但其中嵌套元素可变（因为本身只是引用）

```
1 t = (1, 2)
2
3 print(t)
4
5 print(t[1])
```

(1, 2)
2

```
1 t = ('a', 'b', ['A', 'B'])
2
3 t[2][0] = 'X'
4
5 print(t)
6
7 # t[0] = 'd' # this may cause an error
```

引用了一个list对象

('a', 'b', ['X', 'B'])

- Tuple (list) 的拆分合并运算

```
1  #元组展开
2  a = tuple('1234')  # 展开为一个tuple
3  print(a)
4
5  c, d, *_ = a      # unpack
6  c, d
```

('1', '2', '3', '4')

('1', '2')

```
1  # 元组 合并
2
3  t2 = (3, 4)
4  t = (1, 2, 3)
5
6  print(t+t2)
```

(1, 2, 3, 3, 4)

string对象常用方法：string对象可读，可下标访问，但不可修改

```
1 str = 'abcd'
2 print (str[1])
3 str[1] = 'q'
```

b

```
TypeError                                Traceback (most recent call last)
<ipython-input-75-5b315d8a9b2a> in <module>()
      1 str = 'abcd'
      2 print (str[1])
----> 3 str[1] = 'q'
```

TypeError: 'str' object does not support item assignment

```
1 str = 'Do U have a nice sleep?'
2 str += '\n' + "No I don't."
3 print (str)
```

反斜杠的用法和C语言一致

```
Do U have a nice sleep?
No I don't.
```

```
1 s = "    11123123abc"      #str.strip([chars]) 去前导空格以及特殊符号
2 s = s.strip(' ')
3 print(s)
4 print(s.strip('1'))        ← 这里s的前导'1'并没有真正去除, 表达式是一个新对象
5
6 print('s.find = ', s.find('23'))
7 i = s.find('23', 4)         # 从特定下标开始定位串
8 print (i, s.count('11'))    ← 结果说明是find()函数的复用
```

```
11123123abc
23123abc
s.find = 3
6 1
```



下标访问的另一个常用方式：切片

List切片

```
1 a = [1, 2, 3, 4, 5]
2
3 b = a[1:3]
4 c = a[1:-1]
5 d = a[1:]
6
7 print(b)
8 print(c)
9 print(d)
```

[2, 3]

[2, 3, 4]

[2, 3, 4, 5]

```
: 1 d = a[::2]
   2
   3 d
```

[1, 3, 5]

str切片

- 字符串不可更改内容

```
1 string = "ursoooo cute!"
2
3 string[1:4]
```

'rso'

tuple切片

```
1 t = (1, 2, 3, 4, 5, 6)
2
3 t1 = t[1:3]
4
5 t1
```

(2, 3)



Sequence Types — `list`, `tuple`, `range`

There are three basic sequence types: lists, tuples, and `range` objects. Additional sequence types tailored for processing of `binary data` and `text strings` are described in dedicated sections.

Common Sequence Operations

The operations in the following table are supported by most sequence types, both mutable and immutable. The `collections.abc.Sequence` ABC is provided to make it easier to correctly implement these operations on custom sequence types.

This table lists the sequence operations sorted in ascending priority. In the table, *s* and *t* are sequences of the same type, *n*, *i*, *j* and *k* are integers and *x* is an arbitrary object that meets any type and value restrictions imposed by *s*.

The `in` and `not in` operations have the same priorities as the comparison operations. The `+` (concatenation) and `*` (repetition) operations have the same priority as the corresponding numeric operations. [3]

顺序结构对象:	Operation	Result
	<code>x in s</code>	True if an item of <i>s</i> is equal to <i>x</i> , else False
	<code>x not in s</code>	False if an item of <i>s</i> is equal to <i>x</i> , else True
	<code>s + t</code>	the concatenation of <i>s</i> and <i>t</i>
	<code>s * n, n * s</code>	equivalent to adding <i>s</i> to itself <i>n</i> times
	<code>s[i]</code>	<i>i</i> th item of <i>s</i> , origin 0
切片操作 ➡	<code>s[i:j]</code>	slice of <i>s</i> from <i>i</i> to <i>j</i>
	<code>s[i:j:k]</code>	slice of <i>s</i> from <i>i</i> to <i>j</i> with step <i>k</i>
	<code>len(s)</code>	length of <i>s</i>
	<code>min(s)</code>	smallest item of <i>s</i>
	<code>max(s)</code>	largest item of <i>s</i>
	<code>s.index(x)</code>	index of the first occurrence of <i>x</i> in <i>s</i>
	<code>s.count(x)</code>	total number of occurrences of <i>x</i> in <i>s</i> ⁵⁵

- Containers → List; (可迭代、可写、可直接存取: Mutable

Operation	Result
<code>s[i] = x</code>	item <i>i</i> of <i>s</i> is replaced by <i>x</i>
<code>s[i:j] = t</code>	slice of <i>s</i> from <i>i</i> to <i>j</i> is replaced by the contents of the iterable <i>t</i>
<code>del s[i:j]</code>	same as <code>s[i:j] = []</code>
<code>s[i:j:k] = t</code>	the elements of <code>s[i:j:k]</code> are replaced by those of <i>t</i>
<code>del s[i:j:k]</code>	removes the elements of <code>s[i:j:k]</code> from the list
<code>s.append(x)</code>	same as <code>s[len(s):len(s)] = [x]</code>
<code>s.extend(t)</code> or <code>s += t</code>	for the most part the same as <code>s[len(s):len(s)] = t</code>
<code>s *= n</code>	updates <i>s</i> with its contents repeated <i>n</i> times
<code>s.count(x)</code>	return number of <i>i</i> 's for which <code>s[i] == x</code>
<code>s.index(x[, i[, j]])</code>	return smallest <i>k</i> such that <code>s[k] == x</code> and <code>i <= k < j</code>
<code>s.insert(i, x)</code>	same as <code>s[i:i] = [x]</code>
<code>s.pop([i])</code> ← 可以用来实现栈、队列	same as <code>x = s[i]; del s[i]; return x</code>
<code>s.remove(x)</code>	same as <code>del s[s.index(x)]</code>
<code>s.reverse()</code>	reverses the items of <i>s</i> in place
<code>s.sort([cmp[, key[, reverse]]])</code>	sort the items of <i>s</i> in place

迭代器（容器）类型与顺序计算逻辑

- Iterable类型（iterator）
- for ... in (容器、生成器)



遍历计算容器内元素：

```
1 a = list('12345') # 列表, 元组, 集合, 词典: 容器
2
3 for i in a:      #
4     print(int(i) * 8)
```

8

16

24

32

40



Range(start, end, step) 函数, 左闭右开

```
sum = 0
for i in range(0, 101):
    sum += i
    j = i
print(sum, j)
```

冒号+缩进 声明语句块

+= 运算 *sum*必须先定义

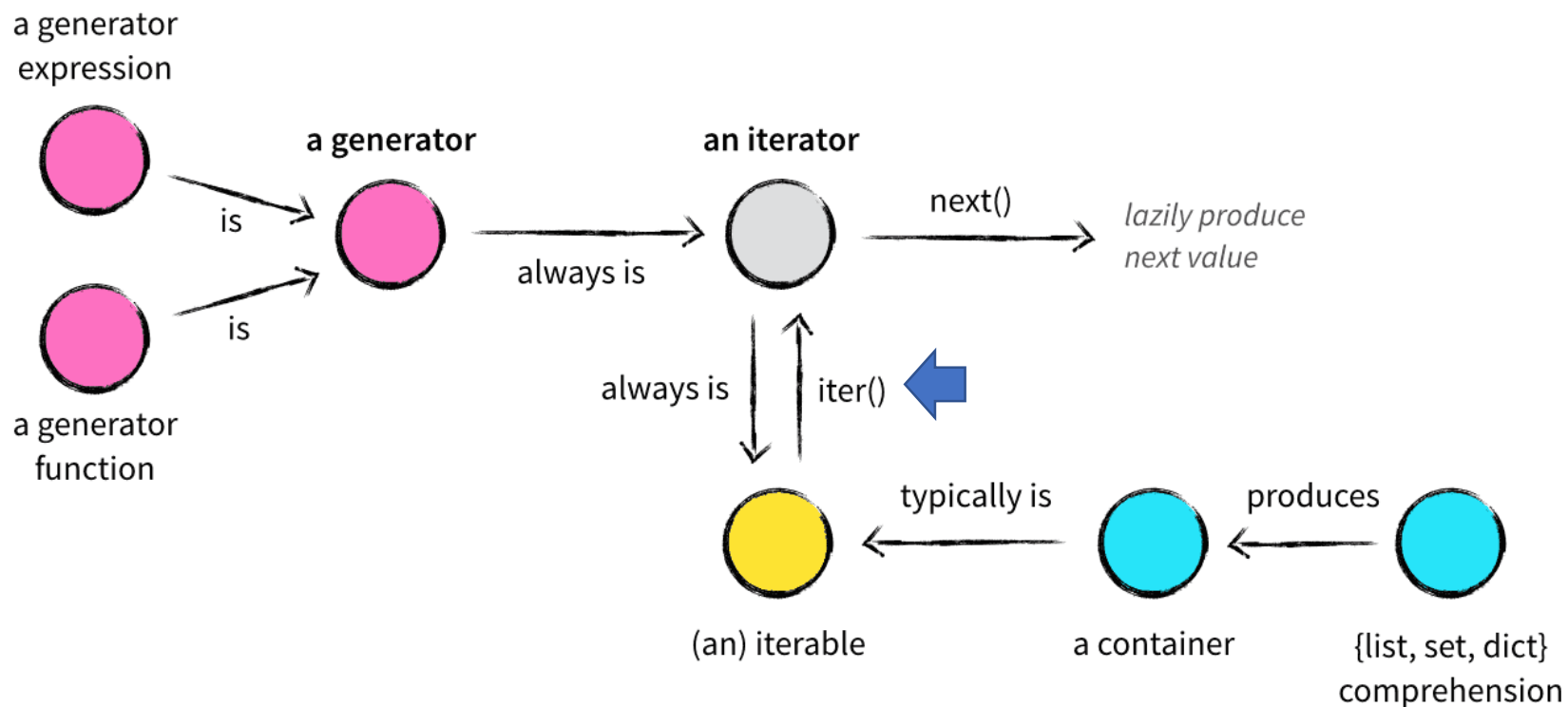
取消缩进, 退出循环体

5050 100

语句块 (for) 并不产生独立的命名空间

生成器、迭代器

- 在Python中，依托循环结构与生成表达式惰性产生数据的对象被称为生成器：**generator**
- Python的**Iterator**对象表示的是一个数据流，Iterator对象可以被next()函数调用并不断返回下一个数据



迭代器:

- 迭代器不但可以作用于for循环，还可以被next()函数不断调用并返回下一个值，直到最后抛出StopIteration错误表示无法继续返回下一个值。

```
1 x = [1, 2, 3]
2
3 y = iter(x) ←
4
5 print(next(y), next(y))
```

1 2

```
1 type(x)
```

list

```
1 type(y)
```

list_iterator

```
1 x = [1, 2, 3]
2 x1 = [] # if x1 not defined
3 y = iter(x)
4
5 for i in y:
6     x1.append(i)
7 print(x1)
```

[1, 2, 3]

深入理解迭代器的本质：

- Request ==》 Answer



How for loop actually works

```
# create an iterator object from that iterable
```

```
iter_obj = iter(iterable)
```

```
# infinite loop
```

```
while True:
```

```
    try:
```

```
        # get the next item
```

```
        element = next(iter_obj)
```

```
        # do something with element
```

```
    except StopIteration:
```

```
        # if StopIteration is raised, break from loop
```

```
        break
```



• 生成器表达式:

```
: 1 b = [int(i)**2 for i in a] # 列表生成式  
2 b
```

```
: [1, 4, 9, 16, 25]
```

```
: 1 c = (int(i)**2 for i in a) # 生成器表达式  
2 c
```

```
: <generator object <genexpr> at 0x0000021425E4E3C0>
```

```
: 1 for i in c:  
2     print(i)
```

```
1  
4  
9  
16  
25
```



set集合

- class set([iterable])
- **unordered** collection of distinct hashable objects

```
1 s1 = {1, 2, 3, 4, 5, 5, 1} ← 花括弧表示集合
2 s2 = set('abcaabbcc')  #unique letters in s
3 print(s1, '\n', s2)
```

{1, 2, 3, 4, 5}
{ 'a', 'c', 'b' }

```
1 s1.add(7)
2 s1.remove(4)
3 s1
```

{1, 2, 3, 5, 7}

```
1 s1 = set([1, 2, 3, 3, 2]) ← 实现类型转换
2
3 s2 = set([2, 3, 4])
4
5 s1 & s2 # also have s1 / s2; s1 - s2
```

{2, 3}

Dict (key-value)

- 即为词典，按关键词查询，删除，修改。可以类比C++语言的map

```
1 d = {'Michael': 95, 'Bob': 75, '1': 85}
2 print('Michael = ', d['Michael'])
3
4 d['Bob'] = 'fail'
5 print('Bob = ', d['Bob'])
```

关键词做下标，直接存取

dict根据key的内容来哈希，因此 key值不可变

```
Michael = 95
Bob = fail
```

```
: 1 d['Xiao Li'] = '90'
   2 print ('Xiao Li' in d)
   3
   4 for k in d:
   5     print (k, d[k])    # case sensitive
   6
```

```
True
Michael 95
Bob fail
1 85
Laohu 80
Xiao Li 90
```

容器类、可迭代对象、迭代器 与 迭代操作

- 容器 (container) 类 (如: list, string, set, tuple, dict...) 都内建有一个操作函数来返回一个迭代器 (iterator)。该迭代器被用来支持 for 循环语句, 以及 in 操作。同时还支持 .next() 接口, 返回下一个对象。

```
iter(object[, sentinel])
```

Return an **iterator** object. The first argument is interpreted very differently depending on the presence of the second argument. Without a second argument, *object* must be a collection object which supports the **iterable** protocol (the **__iter__()** method), or it must support the sequence protocol (the **__getitem__()** method with integer arguments starting at 0). If it does not support either of those protocols, **TypeError** is raised. If the second argument, *sentinel*, is given, then *object* must be a callable object. The **iterator** created in this case will call *object* with no arguments for each call to its **__next__()** method; if the value returned is equal to *sentinel*, **StopIteration** will be raised, otherwise the value will be returned.



循环与 `enumerate`（枚举） 函数:

```
1 animals = ['cat', 'dog', 'monkey']  
2 for idx, animal in enumerate(animals):  
3     print( animals[idx])
```

cat

dog

monkey



词典类型迭代器方法：按关键字遍历

```
for k in d:                # 等效 for k in d.keys():  
    print (k, d[k])
```

↑
迭代器?

```
d.keys()
```


```
小李 95
```

```
John 75
```

```
1 fail
```

按内容 (values) 和条目 (items) 遍历

```
for val in d.values():    # values() 返回一个value的list (view)  
    print(val)
```



95

75

fail

```
for (key, val) in d.items():    # items() 返回一个dict_item的list  
    print(key, val)
```

小李 95

John 75

1 fail



容器类型数据的相互转换

```
print(int(10.6))           # 类型转换函数只接受一个输入参数

print(set([1, 2, 3, 3, 2])) # set类型会自动去重

print(list('hello'))       # 字符串也可以看作是一个容器类型

print(list({1:2, 'a':3}))   # 返回词典的index list

print(dict([[1, 2], [3, 4]])) # 一种初始化词典的方法
```

10

{1, 2, 3}

['h', 'e', 'l', 'l', 'o']

[1, 'a']

{1: 2, 3: 4}



加包-解包运算

```
1 a = 1, 2, 'ww', 3, 4, # 自动加包, tuple
2 a
```

(1, 2, 'ww', 3, 4)

```
1 *_ , b = a # 自动解包
2 b
```

4

```
1 c, d, *_ , f = [*[1, 2, 3], [4, 5, 6]]
2 c, d, f
```

(1, 2, [4, 5, 6])



列表生成式

```
1 b = [i for i in range(5)]
2 b *= 2
3 b
```

Range函数用来创建一个整数列表
for i in ... 生成式

[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]

```
1 [x * x for x in range(1, 11)]
```

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

```
1 [x * x for x in range(1, 11) if x % 2 == 0]
```

[4, 16, 36, 64, 100]

```
1 [m + n for m in 'ABC' for n in 'XYZ']
```

分别生成字符串序列，相加

['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']

```
1 d = {'x': 'A', 'y': 'B', 'z': 'C'} # 词典
2
3 [k + '=' + v for k, v in d.items()]
```

词典容器迭代读出数据，字符串相加

['x=A', 'y=B', 'z=C']

嵌套的列表生成式

```
1 # 二维列表
2 matrix = [[1, 2, 7], [4, 9], [6, 5, 4, 3]]
3
4 flatten_matrix = [val
5                     for sublist in matrix      # 数据源列表
6                     for val in sublist if val < 6 # 遍历每个数据源
7                     ]
8
9 print(flatten_matrix)
10
```

[1, 2, 4, 5, 4, 3]



小结一下：

- 赋值语句是加引用
- 迭代器逻辑上可以看作是一个协议（protocol）
- 迭代器、生成器使用中常作为一个流式数据源



- 容器：可以支持迭代操作，运算逻辑是对遍历区间内所有元素完成指定的计算操作
- range函数：返回一个列表生成式，可以对应其他语言中的循环变量i
- 生成器表达式本质上并不需要事先枚举所有元素。也能实现顺序计算逻辑



接受函数的函数——高阶函数

- map()函数接收两个参数，一个是函数，一个是Iterable的对象，map将传入的函数依次作用到序列的每个元素，并把结果作为新的Iterator返回。

```
1  def f(x):  
2  
3      return x * x  
4  
5  r = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])  
6  
7  list(r)
```

———— [1, 4, 9, 16, 25, 36, 49, 64, 81]

filter()函数

- 过滤序列，filter()接收一个函数和一个序列。把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。

```
1 def is_odd(n):  
2  
3     return n % 2 == 1  
4  
5 list(filter(is_odd, [1, 2, 4, 5, 6, 9, 10, 15]))
```

[1, 5, 9, 15]

```
1 fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
2  
3 odd_numbers = list(filter(lambda x: x % 2, fibonacci))  
4  
5 odd_numbers
```

[1, 1, 3, 5, 13, 21, 55]



zip() 函数用于将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元组，然后返回由这些元组组成的列表。

```
1 str = [[i, j] for i, j in zip('abc', 'bcd')]  
2 print (str)
```

```
[['a', 'b'], ['b', 'c'], ['c', 'd']]
```

join() 方法用于将序列中的元素以指定的字符连接生成一个新的字符串。split用来分割字符串 input()函数用来读入一个字符串

```
1 a, b, c, *_ = map(float, input().split(' ')) #可以读取用空格分开的前三个浮点数  
2 print(a, b, c)
```

```
2.3 4 1.101  
2.3 4.0 1.101
```

```
1 print( ''.join(['0', '1'][i==j] for i, j in zip(input(), input()))) # 布尔量做下标
```

```
hello world!  
hell o w ld  
11110000011
```

布尔量做下标

```

1  # Program to multiply two matrices using list comprehension
2
3
4  X = [[12, 7, 3], [4 , 5, 6], [7 , 8, 9]] # 3x3 matrix
5  Y = [[5, 8, 1, 2], [6, 7, 3, 0], [4, 5, 9, 1]] # 3x4 matrix
6  result = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]] # result is 3x4
7
8  result = [[sum(a*b for a, b in zip(X_row, Y_col))
9              for Y_col in zip(*Y)] # 解包再zip == 转置
10             for X_row in X]
11
12  for r in result:
13      print(r)

```

```

[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]

```


Sorted()函数:

接受一个可迭代对象, 返回一个排好序的list

```
In [10]: ls = [5, 2, 3, 1, 4]
```

```
new_ls = sorted(ls)  
ls
```

```
Out[10]: [5, 2, 3, 1, 4]
```

```
In [11]: sorted?
```

Signature: `sorted(iterable, /, *, key=None, reverse=False)`

Docstring:

Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.

Type: `builtin_function_or_method`

Sorted()用例:

```
1 ids = ['id1', 'id2', 'id30', 'id3', 'id22', 'id100']  
2 print(sorted(ids))  
3 print(sorted(ids, reverse=True))  
4 print(sorted(ids, key=lambda x: int(x[2:])))  
5 ids
```

['id1', 'id100', 'id2', 'id22', 'id3', 'id30']

['id30', 'id3', 'id22', 'id2', 'id100', 'id1']

['id1', 'id2', 'id3', 'id22', 'id30', 'id100']

- ['id1', 'id2', 'id30', 'id3', 'id22', 'id100']

对象自带的Sort方法：

```
In [13]: ls.sort?  
ls.|
```



Tab键进行代码提示和补全

Signature: `ls.sort(*, key=None, reverse=False)`

Docstring: Stable sort *IN PLACE*.

Type: builtin_function_or_method



拆分-合并字符串列表

str.split(sep=None, maxsplit=-1)

- Return a **list** of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements). If maxsplit is not specified or -1, then there is no limit on the number of splits (all possible splits are made).

```
] : s = 'ab,cde, fgh, ijk'

print(s.split(',')) # 切分开逗号分割的串
print(s.split(',', maxsplit= 2))

['ab', 'cde', 'fgh', 'ijk']
['ab', 'cde', 'fgh,ijk']
```

str.join(iterable)

- Join a list of words into a string.

```
] : delimiter = ':'

mylist = ['Brazil', 'Russia', 'India', 'China']

print(delimiter.join(mylist))

Brazil:Russia:India:China
```

```
1  # Program to multiply two matrices using list comprehension
2
3
4  X = [[12, 7, 3], [4 , 5, 6], [7 , 8, 9]] # 3x3 matrix
5  Y = [[5, 8, 1, 2], [6, 7, 3, 0], [4, 5, 9, 1]] # 3x4 matrix
6  result = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]] # result is 3x4
7
8  result = [[sum(a*b for a, b in zip(X_row, Y_col))
9             for Y_col in zip(*Y)] # 解包再zip == 转置
10           for X_row in X]
11
12  for r in result:
13      print(r)
```

[114, 160, 60, 27]

[74, 97, 73, 14]

[119, 157, 112, 23]

课后练习明天发布

