Python 数据科学入门

1.1 课程介绍 & 为什么 Python 如此受欢迎？

(1) 课程介绍
Tips:Python 基础+数据科学基础(Numpy\Pandas\Matplotlib\Scipy)

(2) Python 为什么如此受欢迎？
Python 语言介绍：

- 面向对象，解释型计算机程序设计语言
- 1989 年 Guido van Rossum 发明，1991 年公开版本发行
- Python3.0 在 2008 年发布，为了不带来过多的累赘，没有考虑向下兼容。虽然 Python3 已经发布多年，目前版本 3.6.3，但 Python2 使用热度不减。
- 对于一般学习来讲，常见的 Python2 和 Python3 的区别之一是输出语句。
- 解析器：语言只是符号、语法、语义定义及使用规则的集合。使用这些规则编写的程序（就是 Python 程序）并不能被计算机直接执行。解析执行 Python 源程序的程序叫做 Python 解析器（Interpreter），而由解析器解析执行的过程就是 Python 的实现。Python 解析器有几种：
  --CPython 官方提供的解析器就是 C 语言实现的，所以称之为 CPython，也是最常用的 Python 实现，课程中使用的就是 CPython 作为解析器。
  --JPython 使用 Java 语言实现的 Python 解析器，将 Python 代码编程成 Java 字节码执行。
  --IronPython 是运行在微软 Net 平台上的 Python 解析器，直接把 Python 代码编译成 Net 字节码。
  --PyPy 使用 Python 语言实现的 Python 解析器。
- 广泛应用于科学计算，众多开源科学计算软件包提供 Python 接口
  (e.g.:计算机视觉库 OpenCV，三维可视化库 VTK，医学图像处理库 ITK)
- 应用：科学计算、自然语言处理、图形图像处理、脚本开发、Web 应用
- 当今最受欢迎的语言之一，2004 年后使用率线性增长 TIOBE 排名
- 美国大学计算机编程入门课程最流行的语言(前 10 名计算机系中有 8 名使用 Python）
- 设计哲学：优雅，明确，简单，可读性强

- 优点：功能强大，开发效率高，应用广泛，易上手，语法简洁

- 用途：

  网页开发
  可视化（GUI）界面开发
  网络
  系统编程
  数据分析
  机器学习
  网络爬虫
  科学计算

- 哪些著名网站是用 Python 实现的？

  成千上万，例如：
  雅虎地图
  谷歌中的很多组成部分
  Youtube
  豆瓣网

（3）补充知识点

-----python 版本

Python 2 or Python 3

Python 2.x 是早期版本，Python 3.x 是当前版本
Python 2.7 (2.x 的最终版)于 2010 年发布后很少有大的更新
Python 2.x 比 Python3.x 拥有更多的工具库
大多数 Linux 系统默认安装的仍是 Python 2.x
版本选择取决于要解决的问题

-----建议选择 Python 2.x 的情况：

部署环境不可控，Python 版本不能自行选择
某些工具库还没有提供支持 Python 3.x。
如果选择使用 Python 3.x，需要确定要用的工具库支持新版本。

----课程中会综合 python2 和 3 版本使用，大家要熟知两者的部分区别

（4）举例：

===C++===

```
#include<iostream>
using namespace std;
```

```
int main()
{
   cout<<"Hello World";
   return 0;
}
```

===Java===

```
public class Main{
   public static void main(String[] args)
   {
       System.out.println("Hello World");
   }
}
```

===Python===

```
#!/usr/bin/python
print "Hello World";
```

注：python 的结构：module 模块--->package 模块--->project 项目
python file ----- package -------project(Pycharm)

注意：
（1）脚本语言的第一行#!/usr/bin/python，只对 Linux/Unix 用户适用，用来指定本脚本用什么 interperter 来执行。
有这句的，加上执行权限后，可以直接用./执行，不然会出错，因为找不到 python 解释器
#!/usr/bin/python 是告诉操作系统执行这个脚本的时候，调用/usr/bin 下的 python 解释器；
#!/usr/bin/env python 这种用法是为了防止操作系统用户没有将 python 装在默认的/usr/bin 路径里。当系统看到这一行的时候，首先会到 env 设置里查找 python 的安装路径，再调用对应路径下的解释器程序完成操作。
#!/usr/bin/python 相当于写死了 python 路径;
#!/usr/bin/env python 会去环境设置寻找 python 目录，可以增强代码的可移植性，推荐这种写法
（2）# -*- coding:utf-8 -*-
#coding=utf-8
如果要在 python2 的 py 文件里面写中文，则必须要添加一行声明文件编码的注释，否则 python2 会默认使用 ASCII 编码。

（3）#查看编码方式

```
import sys
print(sys.getdefaultencoding())
```

Python3 字符编码

#编码方式的解读
Python3 最重要的一项改进之一就是解决了 Python2 中字符串与字符编码遗留下来的这个大坑。

#Python2 字符串设计上的一些缺陷：
1.使用 ASCII 码作为默认编码方式，对中文处理很不友好。
2.把字符串的牵强地分为 unicode 和 str 两种类型，误导开发者

#Python3 改进
当然这并不算 Bug，只要处理的时候多留心也可以避免这些坑。但在 Python3 两个问题都很好的解决了。

首先，Python3 把系统默认编码设置为 UTF-8
其次，文本字符和二进制数据区分得更清晰，分别用 str 和 bytes 表示。文本字符全部用 str 类型表示，str 能表示 Unicode 字符集中所有字符，而二进制字节数据用一种全新的数据类型，用 bytes 来表示
然后，Python3 中，在字符引号前加‘b'，明确表示这是一个 bytes 类型的对象，实际上它就是一组二进制字节序列组成的数据

最后，encode 负责字符（unicode）到字节（byte）的编码转换。默认使用 UTF-8 编码准换。
---注意区分：Unicode 和 UTF-8 的区别（utf-8 是为传送 unicode 字符的一种再编码的方式）
#（python3 下面可以演示）
```
>>> s = "Python 之禅"
>>> s.encode()
b'Python\xe4\xb9\x8b\xe7\xa6\x85'
>>> s.encode("gbk")
b'Python\xd6\xae\xec\xf8'
```

decode 负责字节（byte）到字符（unicode）的解码转换，通用使用 UTF-8 编码格式进行转换。
```
>>> b'Python\xe4\xb9\x8b\xe7\xa6\x85'.decode()
'Python 之禅'
>>> b'Python\xd6\xae\xec\xf8'.decode("gbk")
'Python 之禅'
```

（3）Pycharm 新建模板默认添加作者时间等信息
1.打开 pycharm，选择 File-Settings
2.选择 Editor--Color&Style--File and Templates--Python-Script
3.填加
```
#-*- coding: utf-8 -*-
# @Time       : ${DATE} ${TIME}
# @Author     : Z
```

@Email : S
@File : ${NAME}.py
注意：创建 python 文件而不是文件.py，否则不能识别

1.2 安装 Python 和配置环境

1.配置 Python
    1.1 下载 Python
    1.2 安装 Python

2. IDLE 介绍

3. PyDev 介绍
    3.1 Eclipse
    3.2 PyDev for Eclipse

4. 配置 Eclipse
    4.1 下载安装 Java
    4.2 下载 Eclipse

1.3 配置 PyDev

1. 介绍 Python Interpreter（python 解释器）

2. Windows 命令行中运行 Python (配置环境变量）

3. 配置 Eclipse
    3.1 配置 Eclipse 中 java 的路径
    3.2 测试 Java

4. 配置 PyDev
    4.1 在 Eclipse 里安装 PyDev
    4.2 用 PyDev 创建 Python 项目
    4.3 在 Python 项目里创建模块
    4.4 测试运行"Hello World"程序
==========
备注： eclipse 安装 pydev（如上）
    pydev 网址：http://pydev.org/updates 在线安装
==========

1.4 sublime 安装 python

见 md 文档

==========

1.5 安装 Anaconda 数据科学环境

Anaconda（水蟒）：是一个科学计算软件发行版，集成了大量常用扩展包的环境，包含了 conda、Python 等 180 多个科学计算包及其依赖项，并且支持所有操作系统平台。下载地址：https://www.continuum.io/downloads

安装包：pip install xxx,conda install xxx

卸载包：pip uninstall xxx,conda uninstall xxx

升级包：pip install upgrade xxx,conda update xxx

IDE

Jupyter Notebook：

命令：jupyter notebook

1.Anaconda 自带，无需单独安装

2.实时查看运行过程

3.基本的 web 编辑器（本地）

4..ipynb 文件分享

5.可交互式

6.记录历史运行结果

IPython：

命令：ipython

1.Anaconda 自带，无需单独安装

2.Python 的交互式命令行 Shell

3.可交互式

4.记录历史运行结果

5.及时验证想法

Spyder：

命令：spyder

1.Anaconda 自带，无需单独安装

2.完全免费，适合熟悉 Matlab 的用户

3.功能强大，使用简单的图形界面开发环境

==========

1.6 安装 pycharm 的 IDE 工具

需要自行安装：https://www.jetbrains.com/pycharm/download

PyCharm，JetBrains 的精品，全平台支持，不多解释了。

补充：PYPI：https://pypi.python.org/pypi

PyPI(Python Package Index)是 python 官方的第三方库的仓库，所有人都可以下载第三方库或上传自己开发的库到 PyPI。PyPI 推荐使用 pip 包管理器来下载第三方库。

PIP 可正常工作在 Windows、Mac OS、Unix/Linux 等操作系统上，但是需要至少 2.6+和 3.2+的 CPython 或 PyPy 的支持。python 2.7.9 和 3.4 以后的版本已经内置累 pip 程序，所以不需要安装。

2.1 Package 以及数据类型 1

1. 自带 package 和外部 package
   1.1 自带 package 举例： os; os.getwd();os.chdir(path)

2. 外部 package 以及管理系统介绍: easy_install, pip (comes with Python 3.6)

3. 环境变量中配置 easy_install, pip

4. 使用 easy_install, pip 安装 package 举例

```
>>> import requests

>>> r = requests.get('https://www.baidu.com/')

>>> r.text

>>> r.url

>>> r.encoding
```

2.2import 和 from...import 区别（补充）

语法层面：
（1）import 模块名
使用方式：模块名.函数名(参数)
模块名.变量名

（2）from 模块名 import 函数名或变量名（多个函数名用逗号分割）

```
#-*-coding:utf8-*-
#导入模块的案例
#import 和 from....import 区别
Demo1:
```

```
import math
math.sin(0.5)

from math import sin #只导入模型中的指定对象
print sin(3)

from math import sin as f
print f(3)
```

Demo2:

```
import numpy as np
a=np.array((1,2,3,4))
print a
```

2.3 数据类型 2: Numeric & String

1. Python 数据类型

1.1 总体：numerics, sequences, mappings, classes, instances, and exceptions

1.2 Numeric Types: int (包含 boolean), float, complex

1.3 int: unlimited length; float: 实现用 double in C, 可查看 sys.float_info; complex: real(实部) & imaginary(虚部）,用 z.real 和 z.imag 来取两部分

1.4 具体运算以及法则参见：https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

1.5 例子

```
id() #id 函数可以获得对象的 id 标识
type() #type 获取当前数据的数据类型
a=122
b=122
id(122)
id(a)
id(b)
```

大家认为 id 得到的值一致吗？

2.案例

```
import sys

a = 5
b = 3
```

```
e = complex(a, b)
f = complex(float(a), float(b))

print ("a is type" , type(a))
print ("b is type" , type(b))
print ("e is type" , type(e))
print ("f is type" , type(f))

print ("e's real part is: " , e.real)
print ("e's imaginary part is: " , e.imag)

+ - *乘法  /除法   //整除  **幂  %取模
print(a + b)
print(d / c)
print (b / a)
print (b // a)
print (e)
print (e + f)

+= -=   *= /= %=   **= //=
a+=b
a-=b
a*=b
a//=b
a/=b
a%=b
print a

&  按位与  |  按位或  ^按位异或  <<左移  >>  右移
a,b=5,3
#0101   0011
print a&b
print a|b
print a^b
print a<<2 #左移乘 2，这里乘 4
print a>>2 #右移除 2，这里除以 4

print (sys.float_info)
```

2.补充:输入/输出

语法 1：<变量>=input([<提示>])

提示 的信息是在屏幕上，提示用户输入什么数据，可以没有

python3 中从键盘输入的均为 string 字符串类型，如果需要整形或浮点型需要强转

代码 1：

```
a=int(input("请输入整数~"))
b=float(input("请输入浮点数"))
print(a,b)
```

语法 2：print(<输出项列表>，sep=分隔符，end=结束符)

代码 2:

```
print("Sun","Mon","Tue","Fri",sep=',',end=';')
```

注：（1）*python2 中有 raw_input 和 input
raw_input 获取到的输入永远都是 str 类型的
input 获取到的输入会自动判断其类型
***python3 将 raw_input 和 input 进行了整合，只有 input
（2）Python2.x 中，input()函数让我们明确我们输入的是数字格式还是字符格式，就是我们自己要知道我们想要的是什么，数字格式直接输入，字符格式必须加上单引号或者双引号，以确定我们输入的是字符串。

```
>>> a = input("Please input your name: ")
Please input your name: apple
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'apple' is not defined
```

从结果可以看出，提示我们输入的名字没有定义，报错了，说明不能以字符形式直接没有引号的情况下进行输入；

我们验证当我们以引号的方式输入字符串的时候，这个时候没有提示我们错误，由此可以看出，是没有问题的

```
>>> a = input("Please input your name: ")
Please input your name: "apple"
>>> a
'apple'
```

2.4 数据类型 3：字符串、变量

1. 字符串：

一串字符
显示或者打印出来文字信息
导出
编码：# -*- coding: utf-8 -*-
单引号，双引号，三引号
不可变（immutable)
Format 字符串

【Demo】

```
age = 3
name = "Tom"
print("{0} was {1} years old".format(name, age))
```

print("%s was %d years old"%(name, age))
联合：+: print(name + " was " + str(age) + " years old")
换行符: print("What's your name? \nTom")

2. 字面常量（literal constant):

可以直接以字面的意义使用它们：
如：6，2.24，3.45e-3, "This is a string"
常量：不会被改变

3. 变量：
储存信息
属于 identifier
identifier 命名规则：
第一个字符必须是字母或者下划线
其余字符可以是字母，数字，或者下划线
区分大小写
如：合法：i, name_3_4, big_bang
不合法：2people, this is tom, my-name, >123b_c2

4. 注释： #

5. 缩进(Indentation)

#参考代码：
print("hello world") #单引号
print('hello world') #双引号
print('''this is 1 line #三引号
this is 2 line
this is 3 line!''')

2.5 随机数生成的几种方式

```
#-*-coding:utf8-*-
import numpy as np
(1)使用 random
import random
x=random.random() #获取[0,1)内的随机小数
print(x)

n=random.randint(1,100) #获取[1,100]区间上的随机整数
print "n:",n
```

```
print (random.randrange(90,100))
```

(2)使用 numpy

```
#1.给定上下限范围内选取整数
x=np.random.randint(0,10,size=(1,10))  #size 指定 1 行 10 列,size 关键字可以省
y=np.random.randint(0,10,7) #0-10 之间的 7 个数
#2.产生[a,b)中均匀分布的样本值
z=np.random.uniform(-1, 5, size = (3, 4)) # 'size='可省略，（0,1）之间的均匀分布
#3.rand 产生均匀分布的样本值，3 行 4 列
t = np.random.rand(3, 4) #产生 0-1 之间的均匀分布的样本值
#4.产生二项分布的样本值
n, p = 10, .5   # number of trials, probability of each trial 试验次数，每次试验的概率
s = np.random.binomial(n, p, 1000) #产生二项分布的样本值
#5.产生高斯分布的样本值
k=np.random.normal(0,0.1,10) #参数顺序：1.均值  2.标准差  3.生成多少样本
#6.产生卡方分布的样本值
s = np.random.chisquare(2,size=(2,3)) #2 为自由度
```

3.1List 基础数据结构

1. print 中的编码：

编码：# -*- coding: utf-8 -*-

2. print 中的换行

```
print("What's your name? \nTom")
```

3. List

创建
访问
更新
删除
脚本操作符
函数方法

Code:

```
# -*- coding: utf-8 -*-

#创建一个列表

number_list = [1, 3, 5, 7, 9]
```

```
string_list = ["abc", "bbc", "python"]

mixed_list = ['python', 'java', 3, 12]


#访问列表中的值

second_num = number_list[1]

third_string = string_list[2]

fourth_mix = mixed_list[3]

print("second_num: {0} third_string: {1} fourth_mix: {2}".format(second_num, third_string, fourth_mix))

#更新列表
print("number_list before: " + str(number_list))

number_list[1] = 30

print("number_list after: " + str(number_list))

#删除列表元素
print("mixed_list before delete: " + str(mixed_list))

del mixed_list[2]

print("mixed_list after delete: " + str(mixed_list))

#Python 脚本语言

print(len([1,2,3])) #长度
print([1,2,3] + [4,5,6]) #组合
print(['Hello'] * 4) #重复
print(3 in [1,2,3]) #某元素是否在列表中

#列表的截取
abcd_list =['a', 'b', 'c', 'd']
print(abcd_list[1])
print(abcd_list[-2])
print(abcd_list[1:])

#补充
```

```
#6.7 列表创建
print    [1,'a','four'] #创建列表
print []     #创建空列表
print    list((1,2,3,3,4)) #将元组转化为列表
print list(range(1,10,2)) #将 range 对象转化为列表
print list('hello') #将字符串转化为列表
print list({3,5,7}) #将集合转化为列表
print list({'a':3,'b':9,'c':78}) #将字典的”键“转化为列表
print list({'a':3,'b':9,'c':78}.items()) #将字典的”键：值“转化为列表
#这里面注意一般将 tuple()set()dict()list()这样的函数称之为工厂函数，因为这些函数可以生成新的数据类型
#6.8 切片
aList9=[3,4,5,6,7,9,11,13,15,17]
aList9[::] #返回所有元素
aList9[::-1] #所有元素的逆序
aList9[::2]#隔一个元素获取一个元素，获取偶数位置的元素
aList9[1::2]#隔一个元素获取一个元素，获取奇数位置的元素
aList9[0:100]#切片位置大于列表长度，从源列表尾部截断
aList9[100:] #切片的开始位置大于列表长度，返回空列表
# aList9[100]#不允许越界访问，抛出 IndexError 异常
```

3.2part2-zip 函数和 enumerate 函数

```
#6.5zip 函数:用于将多个列表中元素重新组合为元组并返回包含这些元组的 zip 对象
#enumerate()函数返回包含若干下标和值的迭代对象
x=list(range(10))
import random
random.shuffle(x)
print x

print list(zip(x,[1]*11)) #zip 函数也可以用于一个序列或迭代对象
print list(zip(range(1,4)))
print list(zip(['a','b','c'],[1,2])) #如果两个列表不等长，以短的为主

print enumerate(x)   #枚举列表对象，返回 enumerate 对象
print list(enumerate(x)) #enumreate 对象可迭代
```

3.3part3_List 函数操作

```
#  列表操作包含以下函数:
# 1、cmp(list1, list2)：比较两个列表的元素
# 2、len(list)：列表元素个数
```

3、max(list)：返回列表元素最大值
4、min(list)：返回列表元素最小值
5、list(seq)：将元组转换为列表
列表操作包含以下方法:
1、list.append(obj)：在列表末尾添加新的对象
2、list.count(obj)：统计某个元素在列表中出现的次数
3、list.extend(seq)：在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4、list.index(obj)：从列表中找出某个值第一个匹配项的索引位置
5、list.insert(index, obj)：将对象插入列表
6、list.pop(obj=list[-1])：移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7、list.remove(obj)：移除列表中某个值的第一个匹配项
8、list.reverse()：反向列表中元素
9、list.sort([func])：对原列表进行排序

```
a=list(range(1,10,1))
b=list(range(2,11,2))
print cmp(a,b)
print max(a)
print a.index(2)
c=a.insert(0,"app") #c 无效
print a #对 a 本地操作
a.extend(b)
print a
```

2. 元组（tuple)
    创建
    访问
    删除
    脚本操作符
    函数方法

3.4part2 列表 list 与元祖 tuple 的比较

```
#创建只有一个元素的 tuple，需要用逗号结尾消除歧义
a_tuple = (2,)

#tuple 中的 list
mixed_tuple = (1, 2, ['a', 'b'])

print("mixed_tuple: " + str(mixed_tuple))
```

```
mixed_tuple[2][0] = 'c'
mixed_tuple[2][1] = 'd'

print("mixed_tuple: " + str(mixed_tuple))
```

Tuple 是不可变 list。 一旦创建了一个 tuple 就不能以任何方式改变它。

Tuple 与 list 的相同之处
*定义 tuple 与定义 list 的方式相同, 除了整个元素集是用小括号包围的而不是方括号。
*Tuple 的元素与 list 一样按定义的次序进行排序。
*Tuples 的索引与 list 一样从 0 开始, 所以一个非空 tuple 的第一个元素总是 t[0]。
负数索引与 list 一样从 tuple 的尾部开始计数。
*与 list 一样分片 (slice) 也可以使用。注意当分割一个 list 时, 会得到一个新的 list ； 当分割一个 tuple 时, 会得到一个新的 tuple。

Tuple 不存在的方法
*您不能向 tuple 增加元素。Tuple 没有 append 或 extend 方法。
如：

```
list_2=[1,2,3,4]
list_2.append('a')
print list_2
list_2.extend([1,2,3,4])
print list_2
```

*您不能从 tuple 删除元素。Tuple 没有 remove 或 pop 方法。
*然而, 您可以使用 in 来查看一个元素是否存在于 tuple 中。

用 Tuple 的好处
*Tuple 比 list 操作速度快。如果您定义了一个值的常量集，并且唯一要用它做的是不断地遍历它，请使用 tuple 代替 list。
*如果对不需要修改的数据进行 “写保护”，可以使代码更安全。使用 tuple 而不是 list 如同拥有一个隐含的 assert 语句，说明这一数据是常量。如果必须要改变这些值，则需要执行 tuple 到 list 的转换。

Tuple 与 list 的转换
Tuple 可以转换成 list，反之亦然。内置的 tuple 函数接收一个 list，并返回一个有着相同元素的 tuple。而 list 函数接收一个 tuple 返回一个 list。从效果上看，tuple 冻结一个 list，而 list 解冻一个 tuple。

Tuple 的其他应用
一次赋多值====序列解包

```
>>> v = ('a', 'b', 'e')
```

```
>>> (x, y, z) = v
```

解释：v 是一个三元素的 tuple，并且 (x, y, z) 是一个三变量的 tuple。将一个 tuple 赋值给另一个 tuple，会按顺序将 v 的每个值赋值给每个变量。

3.5Dictionary 字典数据结构

键(key)，对应值(value)

结构介绍

```
# -*- coding: utf-8 -*-

#创建一个词典
phone_book = {'Tom': 123, "Jerry": 456, 'Kim': 789}

mixed_dict = {"Tom": 'boy', 11: 23.5}

#访问词典里的值
print("Tom's number is " + str(phone_book['Tom']))

print('Tom is a ' + mixed_dict['Tom'])


#修改词典
phone_book['Tom'] = 999

phone_book['Heath'] = 888

print("phone_book: " + str(phone_book))

phone_book.update({'Ling':159, 'Lili':247})

print("updated phone_book: " + str(phone_book))

#删除词典元素以及词典本身
del phone_book['Tom']
print("phone_book after deleting Tom: " + str(phone_book))

#清空词典
phone_book.clear()
print("after clear: " + str(phone_book))


#删除词典
```

```
del phone_book

# print("after del: " + str(phone_book))

#不允许同一个键出现两次
rep_test = {'Name': 'aa', 'age':5, 'Name': 'bb'}
print("rep_test: " + str(rep_test))


#键必须不可变，所以可以用书，字符串或者元组充当，列表不行

list_dict = {['Name']: 'John', 'Age':13}
list_dict = {('Name'): 'John', 'Age':13}


# 六、字典内置函数&方法
# Python 字典包含了以下内置函数：
# 1、cmp(dict1, dict2)：比较两个字典元素。
# 2、len(dict)：计算字典元素个数，即键的总数。
# 3、str(dict)：输出字典可打印的字符串表示。
# 4、type(variable)：返回输入的变量类型，如果变量是字典就返回字典类型。
# Python 字典包含了以下内置方法：
# 1、radiansdict.clear()：删除字典内所有元素
# 2、radiansdict.copy()：返回一个字典的浅复制
# 3、radiansdict.fromkeys()：创建一个新字典，以序列 seq 中元素做字典的键，val 为字典所有键对应的初始值
如：print "fromkeys",dict_2.fromkeys(dict_2,10)
# 4、radiansdict.get(key, default=None)：返回指定键的值，如果值不在字典中返回 default 值
# 5、radiansdict.has_key(key)：如果键在字典 dict 里返回 true，否则返回 false
# 6、radiansdict.items()：以列表返回可遍历的(键, 值) 元组数组
# 7、radiansdict.keys()：以列表返回一个字典所有的键
# 8、radiansdict.setdefault(key, default=None)：和 get()类似, 但如果键不已经存在于字典中，将会添加键并将值设为 default
# 9、radiansdict.update(dict2)：把字典 dict2 的键/值对更新到 dict 里
# 10、radiansdict.values()：以列表返回字典中的所有值


#备注：
#Python 的最重要的数据类型-字典 dictionnary
#创建字典
dic1={"a":1,"b":2,"c":'apple'}
print dic1
#创建字典的方式 2
dict2=dict(zip(range(3),["a","b","c"]))
```

```
print dict2
print type(range(3)) #list
#创建字典的方式 3
dict3=dict(name="wang",age="23")
print dict3
#字典的遍历
for elem in dic1:
    print elem
    print dic1[elem] #切记容易出错
for elem in dic1.items():
    print elem
for key,elem in dic1.items():
    print key,elem
```

3.6 列表推导（补充）

（回顾）
(1)回顾列表的创建和切片的知识

```
#6.7 列表创建
print   [1,'a','four'] #创建列表
print []    #创建空列表
print   list((1,2,3,3,4)) #将元组转化为列表
print list(range(1,10,2)) #将 range 对象转化为列表
print list('hello') #将字符串转化为列表
print list({3,5,7}) #将集合转化为列表
print list({'a':3,'b':9,'c':78}) #将字典的”键“转化为列表
print list({'a':3,'b':9,'c':78}.items()) #将字典的”键：值“转化为列表
#这里面注意一般将 tuple()set()dict()这样的函数称之为工厂函数，因为这些函数可以生成新的数据类型

#6.8 切片
aList9=[3,4,5,6,7,9,11,13,15,17]
aList9[::] #返回所有元素
aList9[::-1] #所有元素的逆序
aList9[::2]#隔一个元素获取一个元素，获取偶数位置的元素
aList9[1::2]#隔一个元素获取一个元素，获取奇数位置的元素
aList9[0:100]#切片位置大于列表长度，从源列表尾部截断
aList9[100:] #切片的开始位置大于列表长度，返回空列表
# aList9[100]#不允许越界访问，抛出 IndexError 异常
#列表的增删改查
aList10=[3,5,7]
aList10[len(aList10):]=[9] #在尾部增加元素
aList10[:3]=[1,2,3] #替换列表中的元素
aList10[:3]=[] #删除列表元素
```

```
del aList10[:3] #删除列表元素
print aList10
bList=[3,5,7]
aList10==bList #两个列表的值是相等的
aList10 is bList   #两个列表是同一对象
id(aList10)==id(bList) #两个列表的内存地址相等
bList=aList[::] #切片，浅复制
bList.append(9)   #使用 append 增加元素
cList=[1,1,1,3,1,5,1,7,1,9]
del cList[::2]
print cList   #[1, 3, 5, 7, 9]
```

（重点）

(2)列表推导式

```
#-*-coding:utf8-*-
#列表推导式语法
#[表达式 for 变量 in 序列或迭代对象]
#快速生成满足特定需求的列表
#列表推导在逻辑上相当于一个循环，只是形式更加简洁
#Demo1
aList=[x*x for x in range(10)]
print aList
#等同于
aList1=[]
for x in range(10):
    aList1.append(x*x)
print aList1
#等同于
aList2=list(map(lambda x:x*x,range(10)))
```

```
print aList2

#Demo2 练习：
freshfruit=['banana','loganberry','passion fruit']
aList3=[]
for item in freshfruit:
    aList3.append(item.strip())
print aList3
#等同于
aList4=list(map(lambda x:x.strip(),freshfruit))
print aList4
#等同于
aList5=list(map(str.strip,freshfruit))
print aList5
```

（熟悉内容）
（加强）
（3）列表推导的其他应用

```
#1.使用列表推到式实现嵌套列表的平铺（两个 for 循环）
vec = [[1,2,3],[4,5,6],[7,8,9]]
new1=[num for elem in vec for num in elem]
print new1
#等同于
result=[]
for elem1 in vec: #[1,2,3],[4,5,6],[7,8,9]
    for num in elem1:1,2,3,4,5,6,7,8,9
        result.append(num)
print result
#2.过滤不符合条件的元素
aList5=[-1,-4,6,7.5,-2.3,9,-11]
print [i for i in aList5 if i>0]#所有大于零的元素


#3.列表推导中使用多个循环实现多序列元素任意的组合，并过滤元素
aList6=[(x,y)for x in [1,2,3] for y in [3,1,4] if x!=y]
print aList6
#等同于
result1=[]
for x in [1,2,3]:
    for y in [3,1,4]:
        if x!=y:
            result1.append((x,y))
```

```
print result1
#4.实现列表推到式实现矩阵转置
matrix=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
#先取行在取出列
aList7=[ [row[i] for row in matrix] for i in range(4)]
print aList7
#或者使用 zip 函数
aList8=list(map(list,zip(*matrix)))
print aList8
#可以 print zip(*matrix) 吗？------因为 zip 生成的是元祖而不是 list，因此需要利用 list 函数作用将类型改变为 list 类型。
#zip 函数：返回元组列表，其中每个元组包含第 i 个元素来自每个参数序列。 返回的列表被截断长度与最短参数序列的长度相同。

#5.使用列表推导生成 100 以内的所有素数
import numpy as np
# [p for p in range(2,100) if 0 not in [p%d for d in range(2,int(np.sqrt(p))+1)]]   #+1 的意思是 range 不包括末尾值
```

（非重点掌握内容）

```
#6.补充函数编程内容：
#6.1map 函数将一个函数依次作用或映射到序列或迭代器对象的每个元素上，并返回一个 map 对象作为结果
案例 1:
list(map(str,range(5))) #转化为字符串
案例 2:
def add1(v):
    return v+5
print list(map(add1,range(10)))#将单参数函数映射到 1 个序列上
#案例 3：
def add(x,y):
    return x+y
list(map(add,range(5),range(5,10)))#将双参数函数映射到两个序列上
#等同于
```

```
print list(map(lambda x,y:x+y,range(5),range(5,10)))
#等同于
print [add(x,y) for x,y in zip(range(5),range(5,10))]

#6.2reduce 函数
#标准库 functions 中的函数 reduce()可以将一个接受 2 个参数以累积的方式
#  从左到右一次作用到一个序列或迭代器对象的所有元素上
#在 Python2 中 reduce 是内置函数，不需要导入任何模块即可使用,py3 中需要导入包
from functools import reduce
print reduce(add,range(10)) #add 是上面定义的函数
seq=[1,2,3,4,5,6,7,8,9]
#将 lambda 函数应用在 seq 序列上
print reduce(lambda x,y:x+y,seq)

#6.3filter 函数将一个单参数函数作用到一个序列上，返回该序列中
#  使得该函数返回值为 True 的那些元素组成的 filter 对象
#如果指定函数为 None，则返回序列中等价于 True 的元素
#下面代码在 python3 中使用
# seq1=['foo','x41','?1','***']
# def func(x):
#         return x.isalnum()
# filter(func,seq) #返回 filter 对象
# print list(filter(func,seq)) #将 filter 转化为列表
# #使用列表推到式实现相同的功能
# print [x for x in seq if x.isalnum()]
# #使用 lambda 表达式实现相同功能
# print list(filter(lambda x:x.isalnum(),seq))
# print list(filter(None,[1,2,3,0,0,4,0,5]))
#6.4  列表的运算
print [1,2,3]*3
print [1,2,3]+[4,5,6]

#以下为内置函数对列表的操作
#6.5zip 函数:用于将多个列表中元素重新组合为元组并返回包含这些元组的 zip 对象
#enumerate()函数返回包含若干下标和值的迭代对象
x=list(range(10))
import random
random.shuffle(x)
print x
print max(x),min(x),sum(x),len(x)
print list(zip(x,[1]*11)) #zip 函数也可以用于一个序列或迭代对象
print list(zip(range(1,4)))
print list(zip(['a','b','c'],[1,2])) #如果两个列表不等长，以短的为主
```

```
print enumerate(x)   #枚举列表对象，返回 enumerate 对象
print list(enumerate(x)) #enumreate 对象可迭代
#6.6sort()和 reverse()
print "原来的： ",x

g=lambda item:len(str(item))
print g(x)   #30?
print str(x) #[0, 3, 2, 8, 1, 5, 4, 7, 6, 9]
print len(str(x))   #30? len 指的是列表元素的个数

x.sort(key=lambda item:len(str(item)),reverse=True) #按指定规则排序
print "排序的： ",x
x.reverse()
print "逆序的",x
x.sort(key=str) #按转化为字符串后的大小排序
print "按转化为字符串的大小排序",x
```

3.7 元素和生成器推导（补充）

（回顾和复习）

```
#-*-coding:utf8-*-
#元祖的创建
#元祖一旦创建，里面的元素是不可以更改的，无法为元祖增加或删除元素
#1.元祖没有提供 append(),extend()和 insert()方法
#2.也没有 remove()和 pop()方法，也不支持 del()删除元祖元素
#但是 del 可以删除整个元祖
x=(1,2,3)
type(x)
x=(3) #和 x=3 一样，这个不是元祖，有歧义
x=(3,) #如果元祖中只有一个元素，必须在后面多写一个逗号
#空元祖的表示
x=()
x=tuple()
#将其他迭代对象转化为元祖
tuple(range(5))
```

（重点）

```
#生成器推导---
#和列表推导的差别在于使用的是圆括号而不是列表推导的方括号
```

```
#生成器推导得到是一个生成器对象而不是列表或元祖，可以将生成器推导转化为元祖或列表
g=((i+2)**2 for i in range(10))
print g    #表示生成器返回的是生成器对象
print tuple(g) #将生成器对象转化为数组
print list(g) #生成器对象已遍历结束，没有元素了
g=((i+2)**2 for i in range(10))
print g.next()    #Python3 中使用 g.__next__()
print next(g)
#总结：生成器对象具有惰性求值的特点，只在需要时返回元素，比列表推导有更高的效率
```

（非重点内容）

```
#扩展：
包含 yield 语句的函数可以用来创建可迭代的生成器对象
# def  函数名(参数)：
#         ...
#         yield  变量
#普通函数使用 return 返回一个数据，生成器使用 yield 产生一系列数据
#当 f 函数被调用的时候，会输出 yield 后面跟的变量的值
#斐波那契数列 1,1,2,3,5,8,13,21
def f():
    a,b=1,1 #为多个元素赋值
    while True:
        yield a #暂停执行，需要时在产生一个新元素
        a,b=b,a+b #序列解包，继续生成新元素
a=f() #创建生成器对象
for i in range(10): #数列的前 10 个元素
    print a.next()
#常规的例子
def counter(start=0):
    while True:
        yield start
        start+=1
b=counter()
print 'first:',b.next()
print 'second:',b.next()
```

3.8 字典（补充）

（主要内容）

```
#字典推导式
f_dict={i:str(i) for i in range(1,5)}
print f_dict
x=['A','B','C','D']
y=['a','b','c','d']
e_dict={i:j for i,j in zip(x,y)}
print e_dict
#有序字典（python 默认的字典是无序的）
import collections
x=collections.OrderedDict()
x['a']=3
x['b']=5
x['c']=8
print x
```

（回顾及加强内容）

```
#-*-coding:utf8-*-
#字典：包含若干“键值对”元素的无序可变序列
#键和值用冒号分割，不同元素用逗号分割，所有元素放在{}中
#键可以是 Python 中任意不可变的类型（整数、实数、复数、字符串、元祖 ）
#键不允许重复
#字典的创建
#创建方式 1：
a_dict={1:'apple',2:'bnalna',3:'pear'}
print a_dict
#创建方式 2：
b_dict=dict(zip([1,2,3],['apple','bnalana','pear']))
print b_dict
#创建方式 3：
c_dict=dict(name='ni',age='20')
print c_dict
#创建方式 4:创建给定的内容为键，值为空的字典
d_dict=dict.fromkeys(['name','age','sex'])
print d_dict
dict_3['name']="zhao" #更新值的内容
#空字典
x=dict()
```

```
y={}
#查看字典类型
type(x)
#修改元素值
c_dict['age']=30
print c_dict
#添加新元素
c_dict['address']='SDIBT'
print c_dict
#返回所有元素
c_dict.items()
#返回所有的键
c_dict.keys()
#返回所有的值
c_dict.values()
#字典的 update 方法将另一个字典的键：值一次性添加到当前字典对象
c_dict.update({'a':97,'age':39})#修改 age 的值，同时添加键值 a：97
print c_dict
#删除字典元素
del c_dict['age']
#使用 if 判断键是否存在
if 'age' in c_dict:
    print c_dict['Age']
else:
    print 'NoExists'
#可以使用异常结构
try:
    print c_dict['address']
except:
    print 'NoExists'
#Python 提供了强大的 get 函数可以实现上述的功能
print c_dict.get('age')
print c_dict.get('Address','Not Exists.')
print c_dict
#删除整个字典
# del c_dict
#字典的遍历
for item in c_dict: #默认遍历的是字典的键
    print item
for item in c_dict.items():#明确遍历字典的所有元素
    print item
```

（非重点内容）
（加强）

```
#字典的排序
g_dict={'name': 'Dong', 'age': 30}
from operator import itemgetter
print sorted(g_dict.items(),key=itemgetter(1))#按字典的“值”排序
print sorted(g_dict.items(),key=itemgetter(0))#按字典的“键”排序
print sorted(g_dict.items(),key=lambda item:item[0])#按字典的“键”排序
#Demo
h_dict=[{'name': 'Dong', 'age': 30},{'name': 'zhang', 'age': 31},
        {'name': 'Li', 'age': 32},{'name': 'Dong', 'age': 33}]
print h_dict
#按照 key 指定排序依据，先按姓名升序，姓名相同按照年龄降序
print sorted(h_dict,key=lambda x:(x['name'],-x['age']))
```

3.9 集合补充（补充）

```
#-*-coding:utf8-*-
#集合中只能包含数字、字符串、元祖等不可变类型的数据，而不能包含列表、字典、集合等可变类型数据
#有一个方法：利用内置的 hash()函数计算对象的哈希值，凡是无法计算哈希值的对象都不能作为集合的元素，也不能作为字典对象的“键”
#字典的定义
a={3,5} #字典定义方法 1
type(a)
a_set=set(range(8,14))#将 range 对象转化为集合
b_set=set([0,1,2,3,0,1,2,3,7,8])#转化时自动去掉重复元素
c_set=set((1,2,3))
d_set=set({1:'a',2:'b'}.items())
f_set=set() #空集合
del b_set #删除集合


#集合的操作和运算
#集合元素的增加、更新和删除
s={1,2,3}
s.add(3)
print s
s.update({3,4}) #更新当前元素，自动忽略重复元素
print s
s.discard(5) #删除元素，不存在则忽略该操作
```

```
# s.remove(5) #删除元素，不存在则抛出异常
#集合的运算
（在 mspaint 中绘图解释）
d_set=set([8,9,10,11,12,13])
f_set={0,1,2,3,7,8}
d_set | f_set #并集
d_set.union(f_set) #并集
d_set & f_set #交集
d_set.intersection(f_set) #交集
d_set.difference(f_set) #差集
d_set-f_set #差集
d_set.symmetric_difference(f_set) #对称差集
d_set.issubset(f_set) #测试是否为子集
x1={1,2,3}
y1={1,2,5}
z1={1,2,3,4}
print x1<y1    #< > >=这些符号作用于集合的时候用于表示集合之间的包含关系
print x1<z1
```

4.1part1_函数

函数：程序中可重复使用的程序段

给一段程程序起一个名字，用这个名字来执行一段程序，反复使用 （调用函数）

用关键字 ‘def' 来定义，identifier(参数)

- identifier
- 参数 list
- return statement
- 局部变量 vs 全局变量

Code:

```
#-*- coding: utf-8 -*-
```

```
#没有参数和返回的函数
# def say_hi():
#         print(" hi!")
#
# say_hi()
# say_hi()
#
#
#有参数，无返回值
# def print_sum_two(a, b):
#         c = a + b
#         print(c)
#
# print_sum_two(3, 6)

# def hello_some(str):
#         print("hello " + str + "!")
#
# hello_some("China")
# hello_some("Python")


#有参数，有返回值
# def repeat_str(str, times):
#         repeated_strs = str * times
#         return repeated_strs
#
#
# repeated_strings = repeat_str("Happy Birthday!", 4)
# print(repeated_strings)


#全局变量与局部 变量
# x = 60
#
# def foo(x):
#         print("x is: " + str(x))
#         x = 3
#         print("change local x to " + str(x))
#
# foo(x)
# print('x is still', str(x))
```

```
x = 60

def foo():
    global x
    print("x is: " + str(x))
    x = 3
    print("change local x to " + str(x))

foo()
print('value of x is' , str(x))
```

4.2part2_函数

- 默认参数
- 关键字参数
- VarArgs 参数

Code:

```
#-*- coding: utf-8 -*-

# 1.默认参数，且默认值参数一定在最右端
def repeat_str(s, times = 1):
    repeated_strs = s * times
    return repeated_strs


repeated_strings = repeat_str("Happy Birthday!")
print(repeated_strings)

repeated_strings_2 = repeat_str("Happy Birthday!" , 4)
print(repeated_strings_2)

#不能在有默认参数后面跟随没有默认参数
#f(a, b =2)合法
#f(a = 2, b)非法

#2.关键字参数: 指的是调用时候的参数传递形式，可以按参数名字指定值
```

```
def func(a, b = 4, c = 8):
    print('a is', a, 'and b is', b, 'and c is', c)

func(13, 17)
func(125, c = 24)
func(c = 40, a = 80)


#3.VarArgs 参数（重要）
#*参数和**参数，前者用来接收任意多个实参并将其放在一个元祖中
#**用来接收字典类型的参数
def print_paras(fpara, *nums, **words):
    print("fpara: " + str(fpara))
    print("nums: " + str(nums))
    print("words: " + str(words))


print_paras("hello", 1, 3, 5, 7, word = "python", anohter_word = "java")
```

4.3 函数补充内容

```
3.5(补充)函数中需要注意的事项
#-*-coding:utf8-*-
#函数中需要注意的事项

#1.默认值参数，且默认值参数一定在最右端
def say(mess,times=1):
    print (mess+' ')*times
print say.func_defaults #查看函数所有默认值参数的当前值
say('hello')
say('hello',3)

#2.关键参数：指的是调用时候的参数传递形式，可以按参数名字指定值
def demo(a,b,c=5):
    print a,b,c
print demo(3,7)
print demo(a=7,b=3,c=6)
print demo(a=8,b=9,c=0)

#3.可变长参数
#*参数和**参数，前者用来接收任意多个实参并将其放在一个元祖中
#**用来接收字典类型的参数
def demo1(*p):
    print p
```

```
demo1(1,2,3)
demo1(1,2,3,34)
def demo2(**p):
    for item in p.items():
        print item
demo2(x=1,y=2,z=3) #key 和 value 形式

#4.传递参数时的序列解包
def demo3(a,b,c):
    print a+b+c
seq=[1,2,3]
demo3(*seq)

tup=(1,2,3)
demo3(*tup)

dict={1:'a',2:'b',3:'c'}
#字典对象作为实参时默认使用字典的键，如果需要将字
#  典中键值元素作为参数则需要使用 items()方法明确说明，
#  如果单独取出“值”需要调用字典的 values()方法说明
demo3(*dict)
demo3(*dict.items())

set={1,2,3}
demo(*set)

demo3(*dict.values())
```

（重点知识）

```
#5.lambda 表达式--声明匿名函数（没有函数名字的临时使用的小函数）
#lambda 表达式只可以使用一个表达式，不允许包含其他复杂语句，但是在表达式中可以调
用其他函数
#并支持默认值参数和关键参数，该表达式的计算结果相当于函数的返回值
f=lambda x,y,z:x+y+z
print f(1,2,3)

g=lambda x,y=2,z=3:x+y+z #含有默认参数
print g(1)

print g(2,4,5)
print g(2,y=4,z=5) #调用时使用关键参数

L=[(lambda x:x**2),(lambda x:x**3),(lambda x:x**4)]
```

```
print L[0](2),L[1](2),L[2](2)

L=[1,2,3,4,5]
print map((lambda x:x+10),L) #没有名字的 lambda 表达式作为函数参数
print map((lambda x,y,z:x*y*z),[1,2,3],[1,2,3],[1,2,3])


#lambda 使用自定义函数（平方）
def demo4(n):
    return n*n
demo4(5)

a_list=[1,2,3,4,5]
map(lambda x:demo4(x),a_list)

#非重点掌握
data=list(range(20))
print data
import random
random.shuffle(data)
print data
data.sort(key=lambda x:x)
print data
data.sort(key=lambda x:len(str(x))) #使用 lambda 表达式指定排序规则
print data
data.sort(key=lambda x:len(str(x)),reverse=True)
print data
```

4.4 lambda 表达式

语法；
函数名=lambda 参数：表达式
代码：

```
#1
g=lambda x,y:x+y
print(g(4,5))
#2.带默认参数的 lambda 表达式
g1=lambda x,y=0,z=0:x+y+z
print(g(5))
#3.直接使用 lambda 表达式
(lambda x,y=0,z=0:x+y+z)(4,5,6)
```

使用建议：
当函数非常简单，只有一个表达式的时候，可以考虑 lambda 表达式，否则建议用普通函数

5.1 控制流 1_if_for 语句

1. if 语句

```
if condition:
        do something
elif other_condition:
        do something
```

2. for 语句

Code:

```
#猜数字游戏，每次猜一个，告诉用户猜的数值低还是高
# #if statement example
#
# number = 59
# guess = int(input('Enter an integer : '))
#
# if guess == number:
#         # New block starts here
#         print('Bingo! you guessed it right.')
#         print('(but you do not win any prizes!)')
#         # New block ends here
# elif guess < number:
#         # Another block
#         print('No, the number is higher than that')
#         # You can do whatever you want in a block ...
# else:
#         print('No, the number is a    lower than that')
#         # you must have guessed > number to reach here
#
# print('Done')
# # This last statement is always executed,
# # after the if statement is executed.


#the for loop example
#1.for 的基本语法
# for i in range(1, 10): #在 python 中 range 是关键字，从 a 到 b 的值列表，不包括 b
#         print(i)
# else: #for 循环执行完毕之后使用 else
```

```
#       print('The for loop is over')
#
# 2.for 循环用于 list
# a_list = [1, 3, 5, 7, 9]
# for i in a_list:
#       print(i)
# 3.for 循环用于元祖
# a_tuple = (1, 3, 5, 7, 9)
# for i in a_tuple:
#       print(i)
# 4.for 循环用于字典
# a_dict = {'Tom':'111', 'Jerry':'222', 'Cathy':'333'}
# for ele in a_dict:
#       print(ele) #键
#       print(a_dict[ele]) #值
#
# for key, elem in a_dict.items(): #items 遍历的是键和值
#       print(key, elem)
```

5.2 控制流 2_while_range 语句

1. while 语句

2. range 语句

Code：

```
#2.1while example

# number = 59
# guess_flag = False
#
#
# while guess_flag == False:
#       guess = int(input('Enter an integer : '))
#       if guess == number:
#           # New block starts here
#           guess_flag = True
#
#           # New block ends here
#       elif guess < number:
#           # Another block
#           print('No, the number is higher than that, keep guessing')
#           # You can do whatever you want in a block ...
```

```
#         else:
#                 print('No, the number is a    lower than that, keep guessing')
#                 # you must have guessed > number to reach here
#
# print('Bingo! you guessed it right.')
# print('(but you do not win any prizes!)')
# print('Done')

#2.2For example 适合于固定循环次数的循环中
（仅需更改一行即可实现固定次数循环）

number = 59
num_chances = 3
print("you have only 3 chances to guess")

for i in range(1, num_chances + 1):
    print("chance " + str(i))
    guess = int(input('Enter an integer : '))
    if guess == number:
        # New block starts here
        print('Bingo! you guessed it right.')
        print('(but you do not win any prizes!)')
        break

        # New block ends here
    elif guess < number:
        # Another block
        print('No, the number is higher than that, keep guessing, you have ' + str(num_chances
- i) + ' chances left')
        # You can do whatever you want in a block ...
    else:
        print('No, the number is lower than that, keep guessing, you have ' + str(num_chances -
i) + ' chances left')
        # you must have guessed > number to reach here


print('Done')
```

5.3 控制流 2_break_continue 语句

4.3_控制流 3break_continue__pass

1. break

2. continue

3. pass

Code:

```
#break & continue example
# number = 59
#
# while True:
#         guess = int(input('Enter an integer : '))
#         if guess == number:
#                 # New block starts here
#                 break
#
#                 # New block ends here
#         if guess < number:
#                 # Another block
#                 print('No, the number is higher than that, keep guessing')
#                 continue
#                 # You can do whatever you want in a block ...
#         else:
#                 print('No, the number is a    lower than that, keep guessing')
#                 continue
#                 # you must have guessed > number to reach here
#
# print('Bingo! you guessed it right.')
# print('(but you do not win any prizes!)')
# print('Done')

#continue and pass difference

# a_list = [0, 1, 2]
#
# print("using continue:")
# for i in a_list:
#         if not i:
#                 continue
#         print(i)
#
# print("using pass:")
# for i in a_list:
```

```
#       if not i:
#           pass
#       print(i)
```

6.1 输入输出方式介绍(Output Format)

接受用户的输入： input()

输入格式：str(), format

Code:

```
str_1 = input("Enter a string: ")
str_2 = input("Enter another string: ")

print("str_1 is: " + str_1 + ". str_2 is :" + str_2)
print("str_1 is {} + str_2 is {}".format(str_1, str_2))
```

6.2_读写文件 File_IO

1. 写出文件
2. 读入文件

Code:

```
#文件写入磁盘
some_sentences = '''\
I love learning python
because python is fun
and also easy to use
'''

#Open for 'w'irting
f = open('sentences.txt', 'w')
#Write text to File
f.write(some_sentences)
f.close()

#文件从磁盘读取
#If not specifying mode, 'r'ead mode is default
f = open('sentences.txt')#默认是 r 读模式
while True:
    line = f.readline()
    #Zero length means End Of File
```

```
        if len(line) == 0:
            break
        print(line)
# close the File
f.close
```

6.3_错误与异常处理 Error__Exceptions

Python 有两种错误类型：

1. 语法错误(Syntax Errors)

2. 异常（Exceptions)

首先，try 语句下的（try 和 except 之间的代码）被执行
如果没有出现异常，except 语句将被忽略
如果 try 语句之间出现了异常，try 之下异常之后的代码被忽略，直接跳跃到 except 语句
如果异常出现，但并不属于 except 中定义的异常类型，程序将执行外围一层的 try 语句，如果异常没有被处理，将产生 unhandled exception 的错误

处理异常（Handling Exceptions)

Exception doc: https://docs.python.org/3.4/library/exceptions.html

Code:

```
#Example of Syntax errors

# while True print("Hello World!")


#Examples of exceptions

# print(8/0)

# print(hello * 4)

# num = 6
# print("Hello World " + num )
```

```
#Handling exceptions

# while True:
#         try:
#                 x = int(input("Please enter a number"))
#                 break
#         except ValueError:
#                 print("Not valid input, try again...")
```

7.1_面向对象编程 Object-Oriented

1. 面向对象编程

Python 支持面向对象编程

类(class)：现实世界中一些事物的封装 （如：学生）
类：属性 （如：名字，成绩）

类对象
实例对象

引用：通过引用对类的属性和方法进行操作
实例化：创建一个类的具体实例对象 （如：学生张三）

7.2 面向对象实现加减乘除简单四则运算(补充)

```
#-*-coding:utf8-*-
#python 的面向对象
class ArithmetricOperation:
        def __init__(self,xnew,ynew):
                self.x=xnew
                self.y=ynew
        def add(self):
                return self.x+self.y
        def sub(self):
                return self.x-self.y
        def mul(self):
                return self.x*self.y
        def div(self):
                return self.x/self.y
```

```
# obj.x=float(input("X:"))
# obj.y=float(input("Y:"))
obj=ArithmetricOperation(float(input("X:")),float(input("Y:")))
print "x+y=",obj.add()
print "x-y=",obj.sub()
print "x*y=",obj.mul()
print "x/y=",obj.div()

#需求：将上述面向对象程序单独写在一个 py 文件中，在另外一个类中使用：
from 文件名 import 类名 的方式输出对应的结果信息
```

8.1 图形界面 GUI

1. GUI： Graphical User Interface

2. tkinter: GUI library for Python

3. GUI Example

该代码在 Python3.6 中演示并讲解
Code:

```
# -*- coding: utf-8 -*-

from tkinter import *

import tkinter.simpledialog as dl
import tkinter.messagebox as mb

#tkinter GUI Input Output Example
#设置 GUI
root = Tk() #创建应用程序窗口
w = Label(root, text = "Label Title")#创建标签
w.pack() #画布框架

#欢迎消息
mb.showinfo("Welcome", "Welcome Message")#第一个参数 title，第二个参数 message
guess = dl.askinteger("Number", "Enter a number")
#第一个参数 caotion 标题，第二个参数是 prompt 提示

output = 'This is output message'
mb.showinfo("Output: ", output)
```

8.2 猜数字小游戏

8.2_猜数字游戏

1. GUI from tkinter

2. 逻辑层

Code:

```
# #设置 GUI
# root = Tk()
# w = Label(root, text = "Guess Number Game")
# w.pack()
#
# #欢迎消息
# mb.showinfo("Welcome", "Welcome to Guess Number Game")
#
#
# #处理信息
# number = 59
#
# while True:
# #让用户输入信息
#         guess = dl.askinteger("Number", "What's your guess?")
#
#         if guess == number:
#             # New block starts here
#             output = 'Bingo! you guessed it right, but you do not win any prizes!'
#             mb.showinfo("Hint: ", output)
#             break
#             # New block ends here
#         elif guess < number:
#             output = 'No, the number is a    higer than that'
#             mb.showinfo("Hint: ", output)
#         else:
#             output = 'No, the number is a    lower than that'
#             mb.showinfo("Hint: ", output)
#
# print('Done')
```

#Tips：
原始问题：下面的代码执行后为什么 x 的值是[2, 2]呢？

```
>>> x = [3, 5, 7]
>>> x = x[1:] = [2]
>>> x
[2, 2]
```

进一步，我们修改 x 列表的初始内容，会发现不管 x 的初始值是什么，执行 x = x[1:] = [2]之后的 x 的值都是[2, 2]：

```
>>> x = [1, 2, 3, 3, 4, 5, 6]
>>> x = x[1:] = [2]
>>> x
[2, 2]
```

ans：
实际上这里有个小坑，这个问题的根源在于 x = [1:] = [2]相当于 x = [2]和 x[1:] = [2]这两条语句，也就是说先创建列表 x 的值为[2]，然后使用切片为其追加一个元素 2，然后得到[2, 2]。

9.print 换行打印问题？

在 python 中使用 print 方法时，默认是换行打印的，如果我们不想换行打印呢？
这样的，在 python3.x 之前，可以通过在 print 语句之后加逗号解决

```
for i in range(1, 5):
    print i,
```

在 python3.x 之后，可以在 print()之中加 end=""来解决，可以自定义结尾字符
如：

```
for j in range(1, 5):
    print(j, end=" ")
```

Numpy

1.1Numpy 简介

Numpy（Numerical Python）

Numpy：提供了一个在 Python 中做科学计算的基础库，重在数值计算，主要用于多维数组（矩阵）处理的库。用来存储和处理大型矩阵，比 Python 自身的嵌套列表结构要高效的多。本身是由 C 语言开发，是个很基础的扩展，Python 其余的科学计算扩展大部分都是以此为基础。

1.高性能科学计算和数据分析的基础包

2.ndarray，多维数组（矩阵），具有矢量运算能力，快速、节省空间

3.矩阵运算，无需循环，可完成类似 Matlab 中的矢量运算

4.线性代数、随机数生成

5.import numpy as np

Scipy

Scipy ：基于 Numpy 提供了一个在 Python 中做科学计算的工具集，专为科学和工程设计的 Python 工具包。主要应用于统计、优化、整合、线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解、稀疏矩阵等，在数学系或者工程系相对用的多一些，和数据处理的关系不大， 我们知道即可，这里不做讲解。

1.在 NumPy 库的基础上增加了众多的数学、科学及工程常用的库函数

2.线性代数、常微分方程求解、信号处理、图像处理

3.一般的数据处理 numpy 已经够用

4.import scipy as sp

参考学习资料：

Python、NumPy 和 SciPy 介绍：http://cs231n.github.io/python-numpy-tutorial

NumPy 和 SciPy 快速入门：https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

1.2Numpy 创建随机数

```
#（补充）Numpy 随机数生成方式<注重学习方法>
#-*-coding:utf8-*-
import numpy as np
#1.给定上下限范围内选取整数
x=np.random.randint(0,10,size=(1,10))  #size 指定 1 行 10 列
y=np.random.randint(0,10,7) #0-10 之间的 7 个数
#2.产生[0,1)中均匀分布的样本值
z=np.random.uniform(-1, 5, size = (3, 4)) # 'size='可省略，（0,1）之间的均匀分布
#3.rand 产生均匀分布的样本值,3 行 4 列
t = np.random.rand(3, 4) #产生 0-1 之间的均匀分布的样本值
t = np.random.rand(3, 4) #产生 0-1 之间的均匀分布的样本值
#4.产生二项分布的样本值
n, p = 10, .5  # number of trials, probability of each trial 试验次数，每次试验的概率
s = np.random.binomial(n, p, 1000) #产生二项分布的样本值
#5.产生高斯分布的样本值
k=np.random.normal(0,0,1,10) #参数顺序：1.均值 2.标准差 3.生成多少样本
#6.产生卡方分布的样本值
s = np.random.chisquare(2,size=(2,3)) #2 为自由度
```

1.3Ndarray 创建多维数组

ndarray 多维数组(N Dimension Array)

NumPy 数组是一个多维的数组对象（矩阵），称为 ndarray，具有矢量算术运算能力和

复杂的广播能力，并具有执行速度快和节省空间的特点。

注意：ndarray 的下标从 0 开始，且数组里的所有元素必须是相同类型（同构数据多维容器）

ndarray 拥有的属性

1.ndim 属性：维度个数

2.shape 属性：维度大小

3.dtype 属性：数据类型

注：size 数组元素的个数

1. 实例代码：

```
# 导入 numpy，别名 np
import numpy as np

# 生成指定维度大小（3 行 4 列）的随机多维浮点型数据（二维），rand 固定区间 0.0 ~ 1.0
arr = np.random.rand(3, 4)
print(arr)
print(type(arr))

# 生成指定维度大小（3 行 4 列）的随机多维整型数据（二维），randint()可以指定区间（-1, 5）
arr = np.random.randint(-1, 5, size = (3, 4)) # 'size='可省略
print(arr)
print(type(arr))

# 生成指定维度大小（3 行 4 列）的随机多维浮点型数据（二维），uniform()可以指定区间（-1, 5）产生-1 到 5 之间均匀分布的样本值
arr = np.random.uniform(-1, 5, size = (3, 4)) # 'size='可省略
print(arr)
print(type(arr))

print('维度个数: ', arr.ndim)
print('维度大小: ', arr.shape)
print('数据类型: ', arr.dtype)
```

运行结果：

```
[[ 0.09371338  0.06273976  0.22748452  0.49557778]
 [ 0.30840042  0.35659161  0.54995724  0.018144  ]
 [ 0.94551493  0.70916088  0.58877255  0.90435672]]
<class 'numpy.ndarray'>

[[ 1  3  0  1]
 [ 1  4  4  3]
 [ 2  0 -1 -1]]
<class 'numpy.ndarray'>
```

```
[[ 2.25275308    1.67484038 -0.03161878 -0.44635706]
 [ 1.35459097    1.66294159    2.47419548 -0.51144655]
 [ 1.43987571    4.71505054    4.33634358    2.48202309]]
<class 'numpy.ndarray'>

维度个数:   2
维度大小:   (3, 4)
数据类型:   float64
```

2.ndarray 的序列创建

1. np.array(collection)

collection 为 序列型对象(list)、嵌套序列对象(list of list)。

示例代码：

```
# list 序列转换为 ndarray
list = range(10)
arr = np.array(list)

print(arr)                  # ndarray 数据
print(arr.ndim)             # 维度个数
print(arr.shape)       # 维度大小

# list of list 嵌套序列转换为 ndarray
lis_lis = [range(10), range(10)]
arr = np.array(lis_lis)

print(arr)                  # ndarray 数据
print(arr.ndim)             # 维度个数
print(arr.shape)       # 维度大小
```

运行结果：

```
# list 序列转换为 ndarray
[0 1 2 3 4 5 6 7 8 9]
1
(10,)

# list of list 嵌套序列转换为 ndarray
[[0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]]
2
(2, 10)
```

其他创建形式

2. np.zeros()

指定大小的全 0 数组。注意：第一个参数是元组，用来指定大小，如(3, 4)。

3. np.ones()

指定大小的全 1 数组。注意：第一个参数是元组，用来指定大小，如(3, 4)。

4. np.empty()

初始化数组，不是总是返回全 0，有时返回的是未初始的随机值（内存里的随机值）。

实战：

```
# np.zeros
zeros_arr = np.zeros((3, 4))

# np.ones
ones_arr = np.ones((2, 3))

# np.empty
empty_arr = np.empty((3, 3))

# np.empty 指定数据类型
empty_int_arr = np.empty((3, 3), int)

print('------zeros_arr-------')
print(zeros_arr)
#分别打印上述的值查看结果
```

5. np.arange() 和 reshape()

arange() 类似 python 的 range() ，创建一个一维 ndarray 数组。

reshape() 将 重新调整数组的维数。

实战代码：

```
# np.arange()
arr = np.arange(15) # 15 个元素的 一维数组
print(arr)
print(arr.reshape(3, 5)) # 3x5 个元素的 二维数组
print(arr.reshape(1, 3, 5)) # 1x3x5 个元素的 三维数组
```

6. np.arange() 和 random.shuffle()

random.shuffle() 将打乱数组序列（类似于洗牌）。

实战代码：

```
arr = np.arange(15)
print(arr)

np.random.shuffle(arr)
print(arr)
print(arr.reshape(3,5))
```

7.ndarray 的数据类型

1. dtype 参数

指定数组的数据类型，类型名+位数，如 float64, int32

2. astype 方法

转换数组的数据类型

实战：

```
# 初始化 3 行 4 列数组，数据类型为 float64
zeros_float_arr = np.zeros((3, 4), dtype=np.float64)
print(zeros_float_arr)
```

```
print(zeros_float_arr.dtype) #float64

# astype 转换数据类型，将已有的数组的数据类型转换为 int32
zeros_int_arr = zeros_float_arr.astype(np.int32)
print(zeros_int_arr)
print(zeros_int_arr.dtype)    #int32
```

2.1 创建等差数列和等比数列方式

2.1 补充等比数组和等差数组创建方式

● 先来看一个例子，我们让开始点为 0，结束点为 0，元素个数为 10，看看输出结果。为什么是这样子？难道不都是 0 吗？

```
>>> a = np.logspace(0,0,10)
>>> a
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

● 因为 logspace 中，开始点和结束点是 10 的幂，0 代表 10 的 0 次方，9 代表 10 的 9 次方。我们看下面的例子。

```
>>> a = np.logspace(0,9,10)
>>> a
array([    1.00000000e+00,     1.00000000e+01,     1.00000000e+02,
           1.00000000e+03,     1.00000000e+04,     1.00000000e+05,
           1.00000000e+06,     1.00000000e+07,     1.00000000e+08,
           1.00000000e+09])
>>> a = np.logspace(0,9,10)
>>> a
array([    1.00000000e+00,     1.00000000e+01,     1.00000000e+02,
           1.00000000e+03,     1.00000000e+04,     1.00000000e+05,
           1.00000000e+06,     1.00000000e+07,     1.00000000e+08,
           1.00000000e+09])
```

● 假如，我们想要改变基数，不让它以 10 为底数，我们可以改变 base 参数，将其设置为 2 试试。

```
>>> a = np.logspace(0,9,10,base=2)
>>> a
array([   1.,    2.,    4.,    8.,   16.,   32.,   64.,  128.,  256.,  512.])
```

numpy.linspace 是用于创建一个一维数组，并且是等差数列构成的一维数组，它最常用的有三个参数。当然它不只有三个参数，我们通过例子来了解它是如何使用的：

● 首先，我们看一下第一个例子，用到三个参数，第一个参数表示起始点，第二个参数表示终止点，第三个参数表示数列的个数。

```
>>> a = np.linspace(1,10,10)
```

```
>>> a
array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.])
```

- 我经常用它创建一个元素全部是 1 的等差数列，或者你也可以让所有的元素为 0。

```
>>> a = np.linspace(1,1,10)
>>> a
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

- 我们看一下 linspace 创建的数组元素的数据格式，当然是浮点型。

```
>>> a.dtype
dtype('float64')
```

- 我们还可以使用参数 endpoint 来决定是否包含终止值，如果不设置这个参数，默认是 True。
  我们看一下这个例子：注意步长改变了。

```
>>> a = np.linspace(1,10,10,endpoint=False)
>>> a
array([ 1. ,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1])
```

2.2matrix 创建矩阵（二维）

matrix 是 ndarray 的子类，只能生成 2 维的矩阵，而 ndarray(np.array 可以生成多个维度的),并且 matrix 还可以用 mat 的别名方式生成矩阵：

Code：

```
import numpy as np
x1=np.mat("1 2;3 4")
print x1
x2=np.matrix("1 2;3 4")
print x2
x3=np.matrix("1,2;3,4")
print x3

x4=np.matrix([[1,2,3,4],[5,6,7,8]])
print x4

x5=np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print x5

x6=np.mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print x6

#验证三维矩阵 X 是不能被 mat 或 matrix 矩阵转换的，这点需要注意
X = np.random.randint(0, 5, [3, 2, 2])
print(X)
```

```
print X.shape
#这个会报错 ValueError: shape too large to be a matrix.
# Y=np.mat(X)
# print Y.shape
```

2.3Numpy 内置函数

1.元素计算函数
np.ceil(): 向上最接近的整数，参数是 number 或 array
np.floor(): 向下最接近的整数，参数是 number 或 array
np.rint(): 四舍五入，参数是 number 或 array
np.isnan(): 判断元素是否为 NaN(Not a Number)，参数是 number 或 array
np.multiply(): 元素相乘，参数是 number 或 array
np.divide(): 元素相除，参数是 number 或 array
np.abs()：元素的绝对值，参数是 number 或 array
np.where(condition, x, y): 三元运算符，x if condition else y
测试代码：

```
    # randn() 返回具有标准正态分布的序列。
    arr = np.random.randn(2,3)

    print(arr)

    print(np.ceil(arr))

    print(np.floor(arr))

    print(np.rint(arr))

    print(np.isnan(arr))

    print(np.multiply(arr, arr))

    print(np.divide(arr, arr))

    print(np.where(arr > 0, 1, -1))
```

2.元素统计函数
np.mean(), np.sum()：所有元素的平均值，所有元素的和，参数是 number 或 array
np.max(), np.min()：所有元素的最大值，所有元素的最小值，参数是 number 或 array
np.std(), np.var()：所有元素的标准差，所有元素的方差，参数是 number 或 array
np.argmax(), np.argmin()：最大值的下标索引值，最小值的下标索引值，参数是 number 或 array
np.cumsum(), np.cumprod()：返回一个一维数组，每个元素都是之前所有元素的 累加和和 累乘积，参数是 number 或 array
多维数组默认统计全部维度，axis 参数可以按指定轴心统计，值为 0 则按列统计，值为

1 则按行统计。

测试代码：

```
arr = np.arange(12).reshape(3,4)
print(arr)

print(np.cumsum(arr)) # 返回一个一维数组，每个元素都是之前所有元素的 累加和

print(np.sum(arr)) # 所有元素的和

print(np.sum(arr, axis=0)) # 数组的按列统计和

print(np.sum(arr, axis=1)) # 数组的按行统计和
```

3.元素判断函数

np.any(): 至少有一个元素满足指定条件，返回 True

np.all(): 所有的元素满足指定条件，返回 True

测试代码：

```
arr = np.random.randn(2,3)
print(arr)

print(np.any(arr > 0))
print(np.all(arr > 0))
```

4.元素去重排序函数

np.unique():找到唯一值并返回排序结果，类似于 Python 的 set 集合

示例代码：

```
#[[1, 2, 1], [2, 3, 4]]这样声明也可以
arr = np.array([[1, 2, 1], [2, 3, 4]])
print(arr)

print(np.unique(arr))
```

3.1Numpy 线性代数和矩阵运算

3.1 Numpy 线性代数和矩阵运算

```
import numpy as np
#矩阵乘法
x=np.array([[1,2,3],[4,5,6]])
y=np.array([[6,23],[-1,7],[8,9]])
print(x)
print(y)
print(x.dot(y))
print(np.dot(x,y))
#矩阵分解运算-矩阵求逆及转置、行列式求解
from numpy.linalg import inv,qr,svd,eig
X=np.random.randn(5,5)
```

```
mat=X.T.dot(X)
print(inv(mat)) #求解方阵的逆
mat.dot(inv(mat))
q,r=qr(mat)#计算矩阵的 qr 分解-特征值和特征向量分解
print(q,r)
#qr 分解是求解一般的矩阵全部特征值最有效的方法，首先经过矩阵的正交相似变换为 Hessen 矩阵，再通过 QR 方法求特征值和特征向量，得到的是一个正规正交矩阵 Q（m*m）和一个上三角矩阵 R（m*m）
#eig 方阵的特征值和特征向量
value,vector=eig(mat)#计算矩阵的 qr 分解-特征值和特征向量分解
print(value,vector)
补充：利用特征值和特征向量求解原矩阵
matrix1=np.matrix([[1,2,3], [4,5,6], [7,8,9]])
e,v=eig(matrix1)
#  根据特征值和特征向量得到原矩阵
y = v * np.diag(e) * np.linalg.inv(v)
print y
print matrix1

#svd 奇异值分解
U,sigma,VT=svd(X)
#使用 A=mat([[1,2,3],[4,5,6]])也可测试
print("U",U)
print("sigma",sigma)
print("VT",VT)
#转换为原来的矩阵，一定记住 sigma 转化为单位矩阵
print U.dot(np.diag(sigma).dot(VT))
print np.allclose(matrix1, np.dot(U, np.dot(np.diag(sigma), VT)))
```

Pandas

1.1Pandas 介绍

Pandas
什么是 Pandas?
##Pandas 的名称来自于面板数据（panel data）和 Python 数据分析（data analysis）。
Pandas 是一个强大的分析结构化数据的工具集，基于 NumPy 构建，提供了 高级数据结构 和数据操作工具，它是使 Python 成为强大而高效的数据分析环境的重要因素之一。

1.一个强大的分析和操作大型结构化数据集所需的工具集
2.基础是 NumPy，提供了高性能矩阵的运算
3.提供了大量能够快速便捷地处理数据的函数和方法

4.应用于数据挖掘，数据分析

5.提供数据清洗功能

http://pandas.pydata.org

1.2Series

1.Pandas 的数据结构

import pandas as pd

Pandas 有两个最主要也是最重要的数据结构： Series 和 DataFrame

2.Series

简要介绍

1.Series 是一种类似于一维数组的 对象，由一组数据（各种 NumPy 数据类型）以及一组与之对应的索引（数据标签）组成。

2..类似一维数组的对象

由数据和索引组成

索引(index)在左，数据(values)在右

索引是自动创建的

2.1. 通过 list 构建 Series

ser_obj = pd.Series(range(10))

示例代码：

```
# 通过 list 构建 Series
ser_obj = pd.Series(range(10, 20))
print(ser_obj.head(3))

print(ser_obj) #打印全部的 9 个值

print(type(ser_obj))
```

2.2 获取数据和索引

ser_obj.index 和 ser_obj.values

示例代码：

```
# 获取数据
print(ser_obj.values)

# 获取索引
print(ser_obj.index)
```

运行结果：

```
[10 11 12 13 14 15 16 17 18 19]
RangeIndex(start=0, stop=10, step=1)
```

3. 通过索引获取数据

ser_obj[idx]

示例代码：

```
#通过索引获取数据
print(ser_obj[0])
print(ser_obj[8])
```

运行结果：

10

18

4. 通过 dict 构建 Series

示例代码：

```
# 通过 dict 构建 Series
year_data = {2001: 17.8, 2002: 20.1, 2003: 16.5}
ser_obj2 = pd.Series(year_data)
print(ser_obj2.head())
print(ser_obj2.index) #Int64Index([2001, 2002, 2003], dtype='int64')
```

6.name 属性

对象名：ser_obj.name

对象索引名：ser_obj.index.name

示例代码：

```
# name 属性
ser_obj2.name = 'temp'
ser_obj2.index.name = 'year'
print(ser_obj2.head())
```

举例：

```
obj=Series([4,7,-5,3])
print(obj)
print(obj.values)
print(obj.index)
obj2=Series([4,7,-5,3],index=['d','b','a','c'])
print(obj2)
obj2['a']
obj2['d']=6
obj2['c','a','b']
obj2[obj2>0]
obj2[obj2*2]
np.exp(obj2)
print('b' in obj2)
sdata={1:'a',2:'b',3:'c'}
obj3=Series(sdata)
print(obj3)
states=[3,2,1]
obj4=Seroes(sdata,index=states)
print(obj4)
pd.isnull(obj4)
pd.notnull(obj4)
```

```
obj4.isnull()
#series 会在自动对齐不同索引数据
print(obj3+obj4)
```

（补充：）计算 series 唯一值：

```
ubique 唯一值
value_count 返回一个 Series，其索引为唯一值，其值为频率，按计数值降序排序
obj=Series(['c','a','d','a','a','b','b','c','c'])
unique=obj.unique()
print(unique)
print(obj.value_counts) #计算出现频率
```

1.3Pandas-DataFrame

DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同类型的值。DataFrame 既有行索引也有列索引，它可以被看做是由 Series 组成的字典（共用同一个索引），数据是以二维结构存放的。

1.类似多维数组/表格数据 (如，excel, R 中的 data.frame)
2.每列数据可以是不同的类型
3 索引包括列索引和行索引

1. 通过 ndarray 构建 DataFrame

示例代码：

```
import numpy as np

# 通过 ndarray 构建 DataFrame
array = np.random.randn(5,4)
print(array)

df_obj = pd.DataFrame(array)
print(df_obj.head())
```

可以通过 columns 改变列名
可以通过 index 改变行的名字

2. 通过 dict 构建 DataFrame

示例代码：

```
# 通过 dict 构建 DataFrame
dict_data = {'A': 1,
             'B': pd.Timestamp('20170426'),
             'C': pd.Series(1, index=list(range(4)),dtype='float32'),
             'D': np.array([3] * 4,dtype='int32'),
             'E': ["Python","Java","C++","C"],
             'F': 'ITCast' }
```

```
#print dict_data
df_obj2 = pd.DataFrame(dict_data)
print(df_obj2)
```

3. 通过列索引获取列数据（Series 类型）

```
df_obj[col_idx]  或  df_obj.col_idx
```

示例代码：

```
# 通过列索引获取列数据
print(df_obj2['A'])
print(type(df_obj2['A']))


print(df_obj2.A)
```

4. 增加列数据

```
df_obj[new_col_idx] = data
```

类似 Python 的 dict 添加 key-value

示例代码：

```
# 增加列
df_obj2['G'] = df_obj2['D'] + 4
print(df_obj2.head())
```

运行结果：

```
   A          B    C  D       E       F  G
0  1.0 2017-01-02  1.0  3  Python  ITCast  7
1  1.0 2017-01-02  1.0  3    Java  ITCast  7
2  1.0 2017-01-02  1.0  3     C++  ITCast  7
3  1.0 2017-01-02  1.0  3       C  ITCast  7
```

5. 删除列

```
del df_obj[col_idx]
```

示例代码：

```
# 删除列
del(df_obj2['G'] )
print(df_obj2.head())
```

运行结果：

```
   A          B    C  D       E       F
0  1.0 2017-01-02  1.0  3  Python  ITCast
1  1.0 2017-01-02  1.0  3    Java  ITCast
2  1.0 2017-01-02  1.0  3     C++  ITCast
3  1.0 2017-01-02  1.0  3       C  ITCast
```

例子：obj2 使用的是上面的数据

ix 用法

方法：obj.ix[:,val]　　#选择单个列

　　obj.ix[val1,val2] #同时选取行和列

```
print(df_obj2.ix['A']) #通过 ix 获取指定索引对应的行信息，ix 表示索引字段
print(df_obj2.ix[:,'A'])#选取所有航的指定'A'列的数据

frame=DataFrame(df_obj2,columns=['Ax','Bx','Cx','Dx','Ex'],index=['one','two','three','four','five'])
print(frame['Ax'])
print(frame['Bx'])
frame['Ax']=np.arange(5.) #给 Ax 列赋予新值
```

总结：可以输入给 DataFrame 构造器数据的有如下类型：

（1）二维 ndarry
（2）由数组、列表、元祖组成的字典
（3）由 Series 组成的字典
（4）由字典组成的字典
（5）另外一个 DataFrame

```
#补充 1：索引对象是不可以修改的：如
obj=Series(range(3),index=['a','b','c'])
index=obj.index
print(index)
print(index[1:])
index[1]='d' #会报错
```

这个用法会保证 index 对象在多个数据结构之间安全共享。

```
#补充 2：pandas 对象重新索引
obj=Series(range(3),index=['a','b','c'])
obj2=obj.reindex(['c','a','b','f'])
obj3=obj.reindex(['c','a','b','f'],fill_value=0)
obj4=obj.reindex(['c','a','b','f'],method='ffill')
```

method 可选：

```
    #ffill 或 pad 前向填充值
    #bfill 或 backfill  后向填充值
#补充 3：丢弃指定轴上的项,按照索引删除
obj=Series(range(3),index=['a','b','c'])
new_obj=obj.drop('c')
print(new_obj)
new_obj2=obj.drop(['a','b'])
```

2.1Pandas 的对齐运算

是数据清洗的重要过程，可以按索引对齐进行运算，如果没对齐的位置则补 NaN，最后也可以填充 NaN

Series 的对齐运算

1. Series 按行、索引对齐

示例代码：

```
s1 = pd.Series(range(10, 20), index = range(10))
s2 = pd.Series(range(20, 25), index = range(5))

print('s1: ' )
print(s1)

print('s2: ')
print(s2)
```

2.Series 的对齐运算

示例代码：

```
# Series 对齐运算
s1 + s2    #5-9 索引的值为 Nan
```

DataFrame 的对齐运算

1. DataFrame 按行、列索引对齐

示例代码：

```
df1 = pd.DataFrame(np.ones((2,2)), columns = ['a', 'b'])
df2 = pd.DataFrame(np.ones((3,3)), columns = ['a', 'b', 'c'])

print('df1: ')
print(df1)

print('')
print('df2: ')
print(df2)
```

2.DataFrame 的对齐运算

示例代码：

```
# DataFrame 对齐操作
df1 + df2
```

运行结果：

```
     a    b   c
0  2.0  2.0 NaN
1  2.0  2.0 NaN
2  NaN  NaN NaN
```

填充未对齐的数据进行运算

1. fill_value

使用 add, sub, div, mul 的同时，

通过 fill_value 指定填充值，未对齐的数据将和填充值做运算

示例代码：

```
print(s1)
print(s2)
s1.add(s2, fill_value = -1)

print(df1)
print(df2)
df1.sub(df2, fill_value = 2.)
```

3.1Pandas 的函数应用

处理缺失数据

示例代码：

```
df_data = pd.DataFrame([np.random.randn(3), [1., 2., np.nan],
                        [np.nan, 4., np.nan], [1., 2., 3.]])
print(df_data.head())
```

运行结果：

```
          0         1         2
0 -0.281885 -0.786572  0.487126
1  1.000000  2.000000       NaN
2       NaN  4.000000       NaN
3  1.000000  2.000000  3.000000
```

1. 判断是否存在缺失值：isnull()

示例代码：

```
# isnull
print(df_data.isnull())
```

运行结果：

```
       0      1      2
0  False  False  False
1  False  False   True
2   True  False   True
3  False  False  False
```

2. 丢弃缺失数据：dropna()

根据 axis 轴方向，丢弃包含 NaN 的行或列。 示例代码：

```
# dropna
print(df_data.dropna())

print(df_data.dropna(axis=1))
```

运行结果：

```
          0         1         2
0 -0.281885 -0.786572  0.487126
3  1.000000  2.000000  3.000000

          1
0 -0.786572
1  2.000000
2  4.000000
3  2.000000
```

3. 填充缺失数据：fillna()

示例代码：

```
# fillna
print(df_data.fillna(-100.))
```

运行结果：

```
          0         1           2
0   -0.281885 -0.786572    0.487126
1    1.000000  2.000000 -100.000000
2 -100.000000  4.000000 -100.000000
3    1.000000  2.000000    3.000000
```

（补充）

applymappply 和 applymap

1. 可直接使用 NumPy 的函数

示例代码：

```
# Numpy ufunc 函数
df = pd.DataFrame(np.random.randn(5,4) - 1)
print(df)


print(np.abs(df))
```

2. 通过 apply 将函数应用到列或行上

示例代码：

```
# 使用 apply 应用行或列数据
#f = lambda x : x.max()
print(df.apply(lambda x : x.max()))
```

注意指定轴的方向，默认 axis=0，方向是列

示例代码：

```
# 指定轴方向，axis=1，方向是行
```

```
print(df.apply(lambda x : x.max(), axis=1))
```

3. 通过 applymap 将函数应用到每个数据上

示例代码：

```
# 使用 applymap 应用到每个数据
f2 = lambda x : '%.2f' % x
print(df.applymap(f2))
```

排序（补充）

1. 索引排序

sort_index()

排序默认使用升序排序，ascending=False 为降序排序

示例代码：

```
# Series
s4 = pd.Series(range(10, 15), index = np.random.randint(5, size=5))
print(s4)

# 索引排序
s4.sort_index() # 0 0 1 3 3
```

对 DataFrame 操作时注意轴方向

示例代码：

```
# DataFrame
df4 = pd.DataFrame(np.random.randn(3, 5),
                   index=np.random.randint(3, size=3),
                   columns=np.random.randint(5, size=5))
print(df4)

df4_isort = df4.sort_index(axis=1, ascending=False)
print(df4_isort) # 4 2 1 1 0
```

2. 按值排序

sort_values(by='column name')

根据某个唯一的列名进行排序，如果有其他相同列名则报错。

示例代码：

```
# 按值排序
df4_vsort = df4.sort_values(by=0, ascending=False)
print(df4_vsort)
```

3.2 层次索引

8.层级索引（hierarchical indexing）

8.1 下面创建一个 Series， 在输入索引 Index 时，输入了由两个子 list 组成的 list，第一个子 list 是外层索引，第二个 list 是内层索引。

示例代码：

```
import pandas as pd
```

```
import numpy as np

ser_obj = pd.Series(np.random.randn(12),index=[
                        ['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'c', 'd', 'd', 'd'],
                        [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]
                    ])
print(ser_obj)
```

运行结果：

```
a  0    0.099174
   1   -0.310414
   2   -0.558047
b  0    1.742445
   1    1.152924
   2   -0.725332
c  0   -0.150638
   1    0.251660
   2    0.063387
d  0    1.080605
   1    0.567547
   2   -0.154148
dtype: float64
```

8.2MultiIndex 索引对象

打印这个 Series 的索引类型，显示是 MultiIndex

直接将索引打印出来，可以看到有 levels,和 labels 两个信息。lavels 表示两个层级中分别有那些标签，labels 是每个位置分别是什么标签。

示例代码：

```
print(type(ser_obj.index))
print(ser_obj.index)
```

运行结果：

```
<class 'pandas.indexes.multi.MultiIndex'>
MultiIndex(levels=[['a', 'b', 'c', 'd'], [0, 1, 2]],
           labels=[[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3], [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]])
```

8.3 选取子集

根据索引获取数据。因为现在有两层索引，当通过外层索引获取数据的时候，可以直接利用外层索引的标签来获取。

当要通过内层索引获取数据的时候，在 list 中传入两个元素，前者是表示要选取的外层索引，后者表示要选取的内层索引。

1. 外层选取：

   ser_obj['outer_label']

   示例代码：

   ```
   # 外层选取
   print(ser_obj['c'])
   ```

   运行结果：

   ```
   0   -1.362096
   ```

1 1.558091
2 -0.452313
dtype: float64

2. 内层选取：

ser_obj[:, 'inner_label']

示例代码：

```
# 内层选取
print(ser_obj[:, 2])
```

运行结果：

a 0.826662
b 0.015426
c -0.452313
d -0.051063
dtype: float64

常用于分组操作、透视表的生成等

8.4 交换分层顺序

1. swaplevel()

.swaplevel( )交换内层与外层索引。

示例代码：

```
print(ser_obj.swaplevel())
```

运行结果：

0 a 0.099174
1 a -0.310414
2 a -0.558047
0 b 1.742445
1 b 1.152924
2 b -0.725332
0 c -0.150638
1 c 0.251660
2 c 0.063387
0 d 1.080605
1 d 0.567547
2 d -0.154148
dtype: float64

2.交换并排序分层

sortlevel()

.sortlevel( )先对外层索引进行排序，再对内层索引进行排序，默认是升序。

示例代码：

```
# 交换并排序分层
print(ser_obj.swaplevel().sortlevel())
```

运行结果：

0 a 0.099174
b 1.742445
c -0.150638

```
    d    1.080605
1   a   -0.310414
    b    1.152924
    c    0.251660
    d    0.567547
2   a   -0.558047
    b   -0.725332
    c    0.063387
    d   -0.154148
dtype: float64
```

4.1Pandas 统计计算和描述

4.1 示例代码：

```
import numpy as np
import pandas as pd

df_obj = pd.DataFrame(np.random.randn(5,4), columns = ['a', 'b', 'c', 'd'])
print(df_obj)
```

运行结果：

```
          a         b         c         d
0  1.469682  1.948965  1.373124 -0.564129
1 -1.466670 -0.494591  0.467787 -2.007771
2  1.368750  0.532142  0.487862 -1.130825
3 -0.758540 -0.479684  1.239135  1.073077
4 -0.007470  0.997034  2.669219  0.742070
```

4.2 常用的统计计算

sum, mean, max, min…

axis=0 按列统计，axis=1 按行统计

skipna 排除缺失值， 默认为 True

示例代码：

```
df_obj.sum()

df_obj.max()

df_obj.min(axis=1, skipna=False)#按行填充，NA 值自动排除
```

运行结果：

```
a    0.605751
b    2.503866
c    6.237127
d   -1.887578
dtype: float64
```

```
a       1.469682
b       1.948965
c       2.669219
d       1.073077
dtype: float64

0    -0.564129
1    -2.007771
2    -1.130825
3    -0.758540
4    -0.007470
dtype: float64
```

4.3 常用的统计描述

describe 产生多个统计数据

示例代码：

```
print(df_obj.describe())
```

运行结果：

```
              a         b         c         d
count  5.000000  5.000000  5.000000  5.000000
mean   0.180305  0.106488  0.244978  0.178046
std    0.641945  0.454340  1.064356  1.144416
min   -0.677175 -0.490278 -1.164928 -1.574556
25%   -0.064069 -0.182920 -0.464013 -0.089962
50%    0.231722  0.127846  0.355859  0.190482
75%    0.318854  0.463377  1.169750  0.983663
max    1.092195  0.614413  1.328220  1.380601
```

#描述和汇总的方法
count 非 Nan 数量
describe 针对个列汇总统计
min 和 max 最大最小值
argmin、argmax 计算最大值或最小值对应的索引位置
quantile 计算样本的分位数（0-1）
mean 均值
median 中位数
mad 平均绝对离差
var 样本方差
std 样本的标准差
skew 样本值的偏度
kurt 样本值的峰度
cumsum 样本值的累计和

Matplotlib

1.1-Matplotlib&Seaborn 数据可视化库

Matplotlib 是一个 Python 的 2D 绘图库，通过 Matplotlib，开发者可以仅需要几行代码，便可以生成绘图，直方图，功率谱，条形图，错误图，散点图等。
http://matplotlib.org
whl 文件下载地址（pypi）：https://pypi.python.org/pypi/matplotlib/

特点：
1.用于创建出版质量图表的绘图工具库
2.目的是为 Python 构建一个 Matlab 式的绘图接口
3.import matplotlib.pyplot as plt
4.pyploy 模块包含了常用的 matplotlib API 函数
Demo:

```
plt.plot([1,2,3],[3,2,1])
plt.show()

plt.plot([1,2,3],[-3,-2,-1])
plt.show()
```

#plt.show()可类比与 spark 中的 Action 算子，程序只有触发 Action 算子才会真正的执行计算或输出，而常见的 filter、map 操作都会暂存起来，当有 action 操作的时候触发。

1.2 入门案例

需求：1.打印 Matplotlib 版本
2.绘制 y=x+5 和 y=2x+5 两条曲线

```
#-*-coding:utf8-*-
#1.打印当前 matplotlib 的版本信息
import matplotlib as mpl
print "matplotlib version",mpl.__version__ #查看当前版本

#2.plot 绘制曲线图
import matplotlib.pyplot as plt
import numpy as np
#1.通过 Numpy 的 linspace 函数指定横坐标，同时规定起点和终点分别是 0 和 20
x=np.linspace(0,20)
#通过下面画线
plt.plot(x,.5+x)
```

```
plt.plot(x,5+2*x,'--')
#这里既可以用 savefig()函数把图形保存到一个文件中，也可以通过 show()函数将图像显示在屏幕上
plt.show()
```

注意：标题加中文 u+fontproperties='SimHei'这个

```
plt.title(u"一次函数",fontproperties='SimHei')
```

2.1 figure 对象及多图绘制

Matplotlib 的图像均位于 figure 对象中

创建 figure： fig = plt.figure()

```
#需求 1：示例代码：
# 引入 matplotlib 包
import matplotlib.pyplot as plt
import numpy as np

# 创建 figure 对象
fig = plt.figure()
```

运行结果：

```
<matplotlib.figure.Figure at 0x11a2dd7b8>
#需求 2：在一个图中采用面向对象的方式绘制两幅图
#绘制第 1 张图
fig=plt.figure()
ax1=fig.add_subplot(1,1,1)
ax1.plot([1,2,3],[3,2,1])
#绘制第 2 张图
fig2=plt.figure()
ax2=fig2.add_subplot(1,1,1)
ax2.plot([1,2,3],[-3,-2,-1])
plt.show()
```

2.2 通过 figure 对象创建子图(可以用来做图形的对比)

```
fig.add_subplot(a, b, c)
```

a,b 表示将 fig 分割成 a*b 的区域

c 表示当前选中要操作的区域，

注意：从 1 开始编号（不是从 0 开始）

plot 绘图的区域是最后一次指定 subplot 的位置 (jupyter notebook 里不能正确显示)

```
#需求：创建子图，并绘制+-x 和+-x 的拟合直线图（即：给定相同坐标的 x）
```

示例代码：

```
#采用的 figure 对象
```

```
fig=plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

# 在 subplot 上作图
x = np.arange(1, 100)
ax1.plot(x, x)
ax2.plot(x, -x)
ax3.plot(-x, x)
ax4.plot(-x, -x)
plt.show()
```

#采用 plt 的方式

```
x=np.arange(1,100)
plt.subplot(221)
plt.plot(x,x)

plt.subplot(222)
plt.plot(x,-x)
plt.show()
```

3.1 figure 绘制各种图形

#需求：使用 figure 的方式创建子图并做出直方图、散点图、折线图、饼状图、小提琴图

```
# 指定切分区域的位置
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)

plt.show()
```

(下面的 ax1，ax2 都是在上面定义的基础上进行的)

12.3 直方图：hist

示例代码：

```
#或者使用 plt.hist(np.random.randn(100), bins=10, color='b', alpha=0.3)
ax1.hist(np.random.randn(100), bins=10, color='b', alpha=0.3)
```

12.4 散点图：scatter

散点图是分析数据相关性常用的方法，下面绘制了余弦曲线的散点图

示例代码：

```
# 绘制散点图
x = np.arange(0,,2*np.pi,0.1)
y = np.cos(x)
```

```
    #使用 plt.scatter(x, y)
    ax2.scatter(x, y)
```

12.4 折线图

```
#画出-10 到 10 区间的二次函数图
#100 改为 10 就能看出折线图
    x=np.linspace(-10,10,100) #平均分为 100 份
    y=x**2
    ax3.plot(x,y)
```

12.5 饼状图

饼状图显示一个数据中各项的大小与各项和的比例

饼状图中显示为在整个饼状图中的比例

```
#饼状图代码
    labelx=['A','B','C','D']
    fracs=[15,30,45,10]
    #1.必须设置比例为 1:1 才能显示为圆形
    plt.axes(aspect=1)
    #2.加每一块所占有具体比例的值 autopct
    #3.突出显示 explode
    explode=[0,0.2,0,0] #块 B 远离了饼状图中心
    #3.加阴影 shadow
    ax4.pie(x=fracs,labels=labelx,autopct='%.1f%%',explode=explode,shadow=True)
    plt.show()
```

12.6 箱线图

```
#箱线图
    data=np.random.normal(size=1000,loc=0,scale=1)
    #调整异常点的形状 sym='o'
    #whis 参数表示:虚线的长度，默认 1.5(比例)，盒子距离上下四分位数的距离
    #距离越大虚线越长，设置成 0.5 和 100 分别观察
    ax4.boxplot(data,sym='o',whis=1.5)
```

4.1 网格 Grid

```
#绘制 1-5 的随机数的二次函数
#以网格的形式作为背景，可以知道坐标轴的位置
    y=np.arange(1,5)
    plt.plot(y,y**2)
    plt.grid(True)#True 表示显示网格
    plt.grid(color='r',linewidth=2,linestyle="-")
    plt.show()
#面向对象的创建方式：
#plt 在交互中使用很多，面向对象不适合用交互式的方式
    y=np.arange(1,5)
    fig=plt.figure()
    ax=fig.add_subplot(111)
```

```
ax.plot(y,y**2)
ax.grid(color='r'，linestyle='-')
plt.show()
```

5.1 图例的用法

```
x=np.arange(1,11)
plt.plot(x,x**2,label='Normal')
plt.plot(x,x**3,label="Fast")
plt.plot(x,x**4,label="Faster")
#方式 1
#ncol=3 扁平的效果，排为一列
plt.legend(loc=2,ncol=3)#1 是右上角 2 是左上角 3 左下角 4 右下角 0 是 best
#  方式 2：
# plt.legend(['Normal',"Fast","Faster"])
plt.show()
```

6.1 颜色、标记、线型

ax.plot(x, y, ‘r--’)

等价于 ax.plot(x, y, linestyle=‘--’, color=‘r’)

示例代码：

```
import matplotlib.pyplot as plt
import numpy as np

fig, axes = plt.subplots(2)
axes[0].plot(np.random.randint(0, 100, 50), 'ro--')
#  等价
axes[1].plot(np.random.randint(0, 100, 50), color='r', linestyle='dashed', marker='o')
```

常用的颜色、标记、线型：<>

```
marker
.     point
,      pixel
o     circle
v     下三角形
^     上三角形
<     左三角形

color
b：blue
g:green
r:red
c:cyan
```

m:magenta
y:yellow
k:black
w:white

linestyle
- or solid 粗线
-- or dashed dashed line
-. or dashdot dash-dotted
: or dotted dotted line
'None' draw nothing
' ' or '' 什么也不绘画

7.1Matplotlib 综合案例分析

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0,2*np.pi,500)
y1=np.sin(x)
y2=np.cos(x)
y3=np.sin(x*x)
plt.figure(1)  #创建图形
#创建三个轴
ax1=plt.subplot(2,2,1) #第一行第一列
ax2=plt.subplot(2,2,2) #第一行第二列
ax3=plt.subplot(2,1,2) #第二行
plt.sca(ax1)   #选择 ax1
plt.plot(x,y1,color='red')
plt.ylim(-1.2,1.2)
plt.sca(ax2)     #选择 ax2
plt.plot(x,y2,'b--')
plt.ylim(-1.2,1.2)
plt.sca(ax3) #选择 ax3
plt.plot(x,y3,'g--',label="sin x*x")
plt.ylim(-1.2,1.2)
plt.legend(loc=0)
plt.title("Programe")
plt.show()

#补充
 plt.subplots()
```

1.同时返回新创建的 figure 和 subplot 对象数组
2.生成 2 行 2 列 subplot:fig, subplot_arr = plt.subplots(2,2)

示例代码：

```
import matplotlib.pyplot as plt
```

```
import numpy as np

fig, subplot_arr = plt.subplots(2,2)
# bins 为显示个数，一般小于等于数值个数
subplot_arr[1,0].hist(np.random.randn(100), bins=10, color='b', alpha=0.3)
plt.show()
```

8.1（补充）刻度、标签、图例

面向对象和面向过程两种方式在设置方面的区别：

设置刻度范围

```
plt.xlim(), plt.ylim()
ax.set_xlim(), ax.set_ylim()
```

设置显示的刻度

```
plt.xticks(), plt.yticks()
ax.set_xticks(), ax.set_yticks()
```

设置刻度标签

```
ax.set_xticklabels(), ax.set_yticklabels()
```

设置坐标轴标签

```
plt.xlabel(),plt.ylabel()
ax.set_xlabel(), ax.set_ylabel()
```

设置标题

```
ax.set_title()
```

图例

```
ax.plot(label=‘legend’)
ax.legend(), plt.legend()
loc=‘best’：自动选择放置图例最佳位置
```

示例代码：

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(1)
ax.plot(np.random.randn(1000).cumsum(), label='line0')

# 设置刻度
#plt.xlim([0,500])
ax.set_xlim([0, 800])

# 设置显示的刻度
#plt.xticks([0,500])
ax.set_xticks(range(0,500,100))

# 设置刻度标签
ax.set_yticklabels(['Jan', 'Feb', 'Mar'])
```

```
# 设置坐标轴标签
#plt.xlabel('Number')
#plt.ylabel('Month')
ax.set_xlabel('Number')
ax.set_ylabel('Month')

# 设置标题
#plt.title('Example')
ax.set_title('Example')
```

9.1 补充案例

9.1.1 散点图：

https://baike.baidu.com/item/%E6%95%A3%E7%82%B9%E5%9B%BE/10065276?fr=aladdin
定义：用两组数据构成多个坐标点，考察坐标点的分布，判断两变量之间是否存在某种关联或总结坐标点的分布模式。如身高和体重的分布就可以用散点图描绘。
简单的例子：身高和体重

```
#身高
height=[160,170,182,175,173]
#体重
weight=[50,58,80,70,55]
plt.scatter(height,weight)
plt.show()
```

正相关、负相关和不相关

```
N=1000 #样本
x=np.random.randn(N)
#不相关
y1=np.random.randn(N)
plt.scatter(x,y1)
#正相关
y2=x+np.random.randn(N)*0.5
plt.scatter(x,y2)
#负相关
y2=-x+np.random.randn(N)*0.5
plt.scatter(x,y2)
plt.show()
```

9.1.2 折线图

折线图是用直线段将各数据连接起来形成的图形

常用来观察数据随时间变化的趋势
比如温度等的的变化

```
#画出-10 到 10 区间的二次函数图
#100 改为 10 就能看出折线图
x=np.linspace(-10,10,100) #平均分为 100 份
y=x**2
plt.plot(x,y)
plt.show()

#画出 0-10 的正弦函数图
x=np.linspace(0,10,100) #平均分为 100 份
y=np.sin(x)
plt.plot(x,y)
plt.show()
```

9.1.3 条形图

以长方形的长度为变量的统计图表
用来比较多个分类的数据大小
通常用于较小的数据集
如不同季度的销量
介绍：https://baike.baidu.com/item/%E6%9D%A1%E5%BD%A2%E5%9B%BE

```
#主要介绍层叠的和并列的散点图
#一组数据的条形图
    index=np.arange(5)
    y=[20,10,30,34,15]
    #left 条形图最左边的横坐标，height 条形图的高度
    # plt.bar(left=index,height=y)
    #2.外观调整 color='b',width=0.5
    # plt.bar(left=index,height=y,color='b',width=0.5)
    #3.水平条形图-加参数  bottom 和 height、width 变化
    # plt.bar(left=0,bottom=index,width=y,height=0.5,orientation='horizontal')
    plt.barh(left=0,bottom=index,width=y)
    plt.show()
##层叠图
    index=np.arange(4)
    sale1=[52,55,63,53]
    sale2=[44,66,55,41]
    bar_width=0.3
    #蓝色对应的是 sale1
    plt.bar(index,sale1,bar_width,color='b')
    #红色对应的比蓝色多 0.3 宽度
    plt.bar(index+bar_width,sale2,bar_width,color='r')
```

```
plt.show()
```

9.1.4 直方图画法

由一些列高度不等的纵向条形组成，表示数据分布的情况
例如学校身高的分布情况
和条形图的区别：直方图是连续的值，使用连续的方式分组
条形图展示不同类别数据，不连续的且不能自定义的

```
#自定义生成随机数
    mu=100
    sigma=20
    x=mu+sigma*np.random.randn(200)
    #bins 在直方图中有几个直方，color 颜色，normed 标准化(个数变为频率)
    plt.hist(x,bins=10,color='r',normed=True)
    #调整数据量，可以接近总体密度曲线
    plt.show()

#双变量的直方分布图
    x=np.random.randn(1000)+2 #中心在 2
    y=np.random.randn(1000)+3 #中心在 3
    plt.hist2d(x,y,bins=40)#颜色深浅表示频率大小
    plt.show()
```

9.1.5 饼状图

饼状图显示一个数据中各项的大小与各项和的比例
饼状图中显示为在整个饼状图中的比例

```
#饼状图代码
    labelx=['A','B','C','D']
    fracs=[15,30,45,10]
    #1.必须设置比例为 1:1 才能显示为圆形
    plt.axes(aspect=1)
    #2.加每一块所占有具体比例的值 autopct
    #3.突出显示 explode
    explode=[0,0.2,0,0] #块 B 远离了饼状图中心
    #3.加阴影 shadow
    plt.pie(x=fracs,labels=labelx,autopct='%.1f%%',explode=explode,shadow=True)
    plt.show()
```

9.1.6 箱线图 boxplot

显示数据的分散情况

上四分位数，中位数，下四分位数，下边缘，异常值（最外面的点）

```
#箱线图
    np.random.seed(10)
    data=np.random.normal(size=1000,loc=0,scale=1)
    #调整异常点的形状 sym='o'
    #whis 参数表示:虚线的长度，默认 1.5(比例)，盒子距离上下四分位数的距离
    #距离越大虚线越长，设置成 0.5 和 100 分别观察
    plt.boxplot(data,sym='o',whis=1.5)



#同时画几组数据在一个图形上
    np.random.seed(10)
    #4 列数据
    data=np.random.normal(size=(1000,4),loc=0,scale=1)
    labels=['A','B','C','D']
    plt.boxplot(data,labels=labels)
    plt.show()
```

Scipy

在 Python 科学计算中，Scipy 为数学、物理、工程方面的科学计算提供了无可代替的支持

它是一个 Numpy 的高级模块

主要凸显在符号计算、信号处理、数值优化

最重要的是向量化的思想（包括符号计算和函数向量化）

Scipy 子模块的汇总表：

scipy.cluster　主流的聚类算法

scipy.constants 数学和物理常数

scipy.fftpack 快速傅里叶变换

scipy.integrate 求解积分和常微分方程

scipy.linalg 线性代数

scipy.ndimage n 维图像处理

scipy.signal 信号处理

scipy.spatial 空间数据结构和算法

scipy.stats　　统计分布及其相关函数

```
    #-*-coding:utf8-*-
#0.常数
    from scipy import constants as C
    print (C.pi)   #圆周率
    print (C.golden)   #黄金比例
    print (C.c)    #光速
```

```
#1.特殊函数模块
    from scipy import special as S
    print (S.cbrt(8)) #8 的立方根
    print (S.exp10(3)) #10**3
    print (S.comb(5,3)) #从 5 个中任选 3 个
    print (S.perm(5,3))#排列数
#2.符号计算案例
    from scipy import poly1d
    p=poly1d([3,4,5]) #3 x^2 + 4 x + 5
    print p
    print p*p    # 9 x^4 + 24 x ^3+ 46 x ^2+ 40 x + 25
    #       4           3           2
    # 9 x + 24 x + 46 x + 40 x + 25
    print p.integ(k=6) #求解 p(x)的不定积分，指定常数项为 6
    print p.deriv() #求解 P(x)的一阶导数
    print p([1,5]) #计算每个值代入 p(x)的结果
#3.数值积分
    from scipy import integrate #导入积分函数
    def g(x):#  定义被积函数
        return (1-x**2)**0.5
    pi_2,err=integrate.quad(g,-1,1) #积分结果和误差
    print (pi_2*2) #根据微积分知识知道积分结果为圆周率一半，3.14159265359
#4.函数向量化
    import numpy as np
    def addsubtract(a,b):
        if a>b:
            return a-b
        else:
            return a+b
    vec_addsubtract=np.vectorize(addsubtract)
    print vec_addsubtract([0,3,6,9],[1,3,5,7]) #[1 6 1 2]
# 5.一维卷积运算
    from scipy import signal
    x=np.array([1,2,3])
    h=np.array([4,5,6])
    print "一维卷积运算",signal.convolve(x,h)
#5.信号处理模块-卷积处理图像
    from scipy import signal,misc
    import matplotlib.pyplot as plt
    image=misc.ascent() #二维图像，公寓图像
    w=np.zeros((50,50))
    w[0][0]=1.0 #修改参数调整滤波器
    w[49][25]=1.0 #可以根据需要调整
    image_new=signal.fftconvolve(image,w) #使用 FFT 算法进行卷积
```

```
plt.figure()
plt.imshow(image_new) #显示滤波后的图像
plt.gray()
plt.title("Filteres image!")
plt.show()
```