第1章 Python语言入门概览

浙江工商大学计算机与信息工程学院

提纲

- 1.1 Python语言概述
- 1.2 Python能做什么？
- 1.3 Python的安装、运行与IDE
- 1.4最简单的Python代码
- 1.5 Python语言风格
- 1.6 课程安排

1.1 Python语言概述

- 听说过Python吗？

1.1-1 Python的面世

- 1989年,Guido van Rossum在阿姆斯特丹完成，第一个公开版发行于1991年。
- Guido为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，做为ABC语言的一种继承
  - ABC是由Guido参加设计的一种教学语言非常优美和强大，是专门为非专业程序员设计的
  - 使用Python作为语言的名字，因为是英国幽默剧团:"Monty Python飞行马戏团"的fans

创始人：

吉多·范罗苏姆

Guido van Rossum

1.1-1 Python的面世

- Python 2.0于2000年10月16日发布，主要是实现了完整的垃圾回收，并且支持Unicode。
- Python 3.0于2008年12月3日发布，此版不完全兼容之前的Python源代码。
- 目前Python2使用最广泛的版本是2.7，Python3的最新版本为3.7.2（截止2019.02）
- Python3不兼容Python2。本课程仅考虑Python3.

1.1-2 Python解释器

- Python是一门跨平台的脚本语言，Python规定了一个Python语法规则，实现Python语法的解释程序就成为Python的解释器
  - CPython（ClassicPython, 最原始python的实现，需要区别于其他实现的时候才以CPython称呼;或用C语言实现的的Python）
  - Jython （Java语言实现的Python ）
  - Ironpython (面向.NET和ECMA CLI的Python实现)
  - PyPy (使用Python语言写的Python)
  - Zhpy（支持繁/简中文语句编写程序的Python语言）

1.1-3 Python的特色

- 容易上手
  - 提供交互式环境
  - 语法简洁
    - ✓高级数据结构简洁地表达复杂的操作
    - ✓语句组织依赖于缩进
    - ✓参数或变量不需要声明
- 火力强大
  - 易学但不简单，从桌面程序，到网络互联，图形处理，科学计算，机器学习/深度学习，实时控制，到处都有Python的身影
  - 跨平台的（Windows, Unix, Macantoshi）
  - 面向对象

1.1-3 Python的特色

- 快速开发
  - Python内建的高层次数据结构，以及动态类型和动态邦定，非常适合于快速应用开发
  - Python语法强调可读性，降低了程序的维护费用
  - Python支持模块和包，并鼓励程序模块化和代码重用。

1.1-3 Python的特色

- 丰富的库
  - Python 标准库已经很庞大。可帮你处理各种工作：正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、 FTP、电子邮件、XML文件、GUI和其他与系统有关的操作。
  - Python开源、免费，在“百花齐放”式地发展中，已经产生大量的高质量库，如Pygame、 matplotlib 、scipy、numpy、sklearn、tensorflow、pytorch等等
- 可扩展、可嵌入
  - 对性能有特殊要求的部分代码可用C或C++编写
  - 可以把Python嵌入你的C/C++程序，从而向你的程序用户提供脚本功能
- 解释性
  - python程序不需要编译成二进制代码，可以直接从源代码运行程序。使得Python程序更加易于移植。

1.2 Python能做什么?

- Python既具备高性能且能使编程简单有趣的特点
- Python能做什么呢？
  - 系统编程
  - 用户图形接口
  - Internet通信
  - 数据库编程
  - 数值计算和科学计算
  - 游戏开发、图像处理、人工智能、机器人等
  - 。。。。。。。

Python确实是一种十分精彩又强大的语言。
Python似乎是万能的。

1.2-1 Python可做的事

- Internet通信
  - Python提供了标准Internet模块，可以广泛应用到各种网络任务中，无论服务端还是在客户端
  - 套接字
  - XML-RPC、SOAP、Telnet等通信
  - Twisted
  - 网站开发工具：HTMLGen、mod_python、Django、TurboGears、Zop，使Python能够快速构建功能完善和高质量的网站
- 数据库编程
  - 支持所有主流数据库：Oracle、Sybase、MySQL、PostgreSQL、Informix、SQLite
  - 定义了标准的、可移植的数据库API
  - Pickle
  - 面向对象数据库系统：ZODB
  - 从关系数据库映射到Python类（ORM）： SQLAlchemy 、SQLObject

1.2-1 Python可做的事

- 数据分析与数据科学
  - 科学计算：NumPy，Scipy
  - 数据分析：Pandas
  - 画图：matplotlib
  - 数据挖掘：scikit-learn，nltk，tensorflow（深度学习）等
  - 大数据：支持hadoop、spark等大数据平台的python编程
- 游戏、图像处理、机器人等
  - 游戏：Pygame/Bigworld
  - 图像：PIL
  - 机器人控制：PyRO

1.2-2 Python和其他语言的比较

- 比TCL强大，支持“大规模编程”，适宜于开发大型系统
- 比Perl语法简洁，更具可读性、更易于维护，有助于减少Bug
- 比Java更简单、更易于使用
- 比C++更简单、更易于使用，不与之竞争
- 比VB更强大也更具备跨平台特性。(中学开始推广Python课程)
- 比Ruby更成熟、语法更具可读性

1.2-3 谁在用Python?

典型几个国外公司

1.2-3 谁在用Python?

国内谁在用Python

豆瓣douban 知乎 易度云办公 everydo.com 搜狐 SOHU.com

网易游戏 NIE.163.COM Sina App Engine 腾讯网 QQ.com

42 果壳 科技有意思 Guokr.com 快玩游戏 kuaiwan.com

典型几个国内公司

1.3 Python的安装、运行与IDE

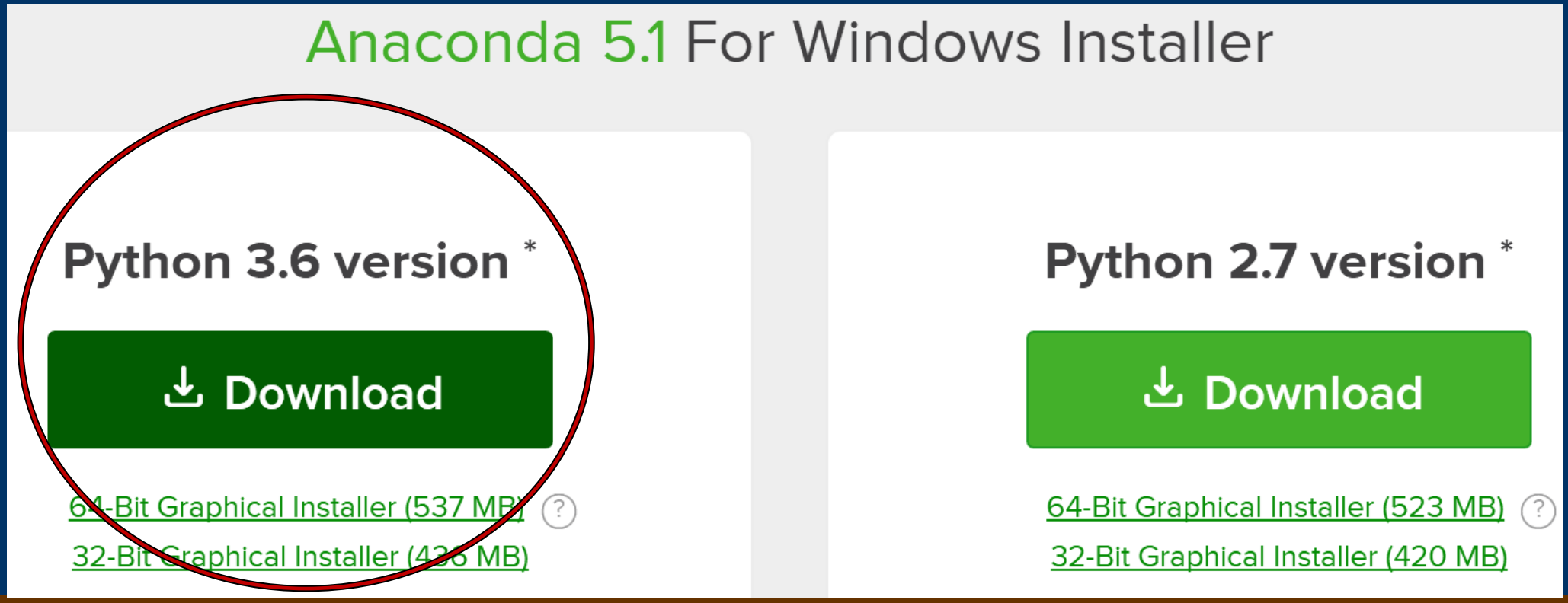
- Python官网(https://www.python.org/downloads/)下载界面

1.3-1 Python的安装

- 安装方式(一)：较为繁琐不推荐！
  - 官网下载Python核心
    - ✓python-2.7.14.msi
    - ✓python-3.6.4.msi　推荐！
  - 常用第三方库下载
    - ✓Python package index (pypi):
      https://pypi.python.org/pypi
    - ✓numpy、 scipy　科学计算
    - ✓pandas　数据分析
    - ✓matplotlib　二维、三维画图
    - ✓pygame　游戏开发
    - ✓wxpython　图形用户界面开发
    - ✓django　web开发
    - ✓scikit-learn　数据挖掘

1.3-1 Python的安装

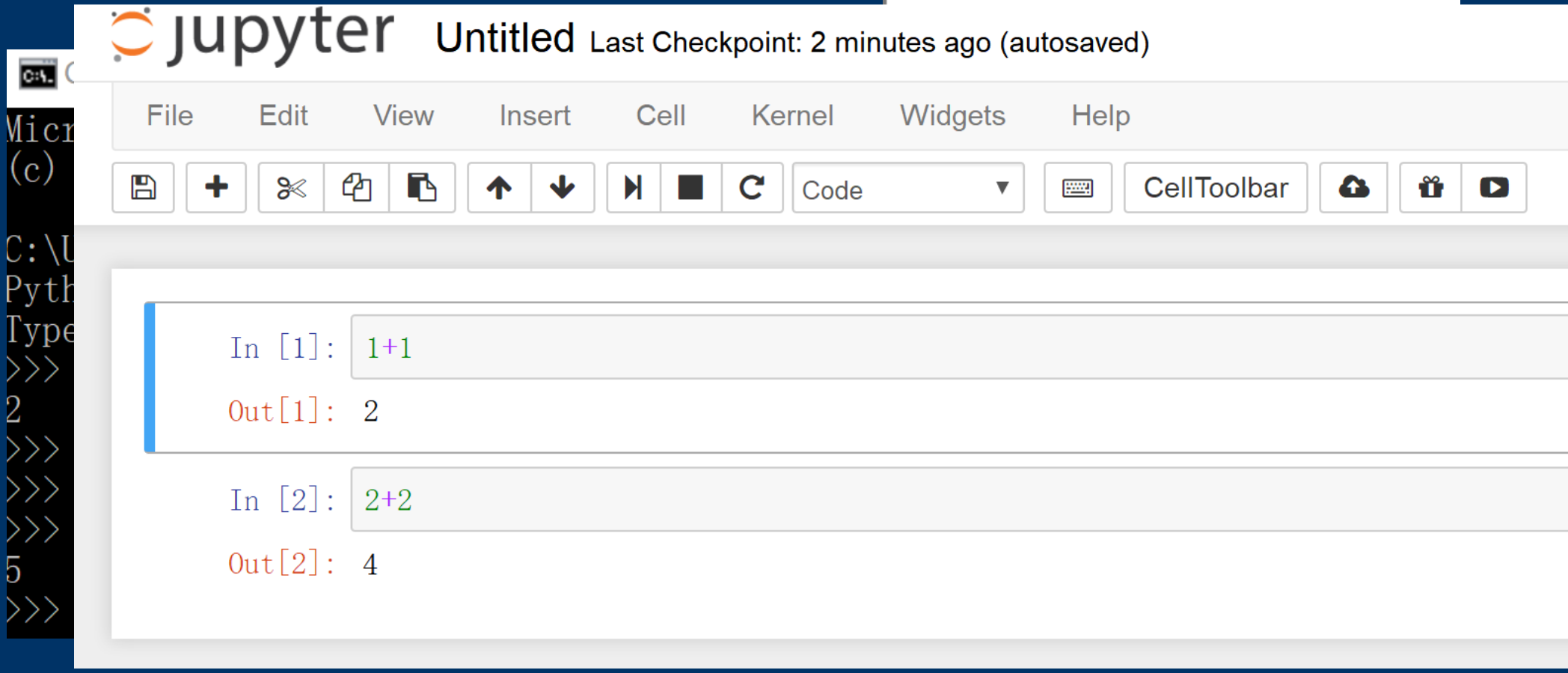
- 安装方式(二)---强烈推荐！
  - 安装Anaconda
  - 官网：https://www.continuum.io/why-anaconda
  - 集成1000多个Python科学计算包
  - 安装完anaconda，就相当于安装了Python、IPython、集成开发环境Spyder、交互式笔记本Jupyter Notebook、科学计算numpy和scipy、可视化matplotlib、数据分析Pandas等等



1.3-2 Python的运行

交互式shell

- 方式1：打开cmd，敲入python，或使用pyhton自带的ipython
- 方式2：IDE自带交互式shell，如spyder中的shell
- 方式3：Jupyter notebook(功能很强大的交互式笔记本)



1.3-2 Python的运行

- 编写源文件
  - Python源文件是以.py为后缀的文本文件
  - 可以用任何编辑文本的软件进行编写，如记事本，UltraEdit，EditPlus等
  - 也可以使用专业的python开发环境编写，如IDLE，Spyder，pycharm等

- 运行源文件
  - 如编写名为“hi.py”的python源文件，内容如下：
    print(“hello world!”)
  - 直接在编辑器（开发环境）中执行，如spyder（anaconda自带）
  - 命令行形式： python hi.py

1.3-3 Python开发环境IDE

- IDLE：Python内置IDE (随python安装包提供)

- PyCharm：由著名的JetBrains公司开发，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工 具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发，推荐！

- Spyder：Spyder是Python(x,y)的作者为它开发的一个简单的集成开发环境。和其他的Python开发环境相比，它最大的优点就是模仿MATLAB的“工作空间”的功能，可以很方便地观察和修改数组的值。安装Anaconda时自带，强烈推荐！

  - 注意，PyCharm community版本免费，Spyder在安装Anaconda时自带。

1.4最简单的Python代码

- Python的设计哲学：

- Python开发者的哲学：

“用一种方法，最好是只有一种方法来做一件事”。

在设计Python语言时，如果面临多种选择，Python开发者总会拒绝花哨的语法，而选择明确的没有或者很少有歧义的语法。

1.4-1 语法入门

- 变量
  - 使用前不用类型声明
  - 在使用过程中类型可以变化
- 标识符命名规则：
  - 必须有字母（大小写）、数字、下划线组成，但首字符必须是字母或下划线
    例如，_abc，abc， abc_1为合法标识符，而 1_abc， abc$， @#!为不合法标识符
  - 标识符大小写敏感
    例如，Abc 与 abc 不是一个标识符
  - 标识符不能为 Python 保留字

1.4-1 语法入门

- 表达式
  - 2+3, 100000000000000000000+1
  - 2+3*5，3**5
  - 2/3, 2//3, 5//4.5
  - “hello”+‘world’
- 逻辑表达式
  - C语言： &&　　||　　!
  - Python： and　　or　　not
  - Python例子：a==1 and b==5

```
In [8]: 3**5
Out[8]: 243

In [9]: 2/3
Out[9]: 0.6666666666666666

In [10]: 2//3
Out[10]: 0

In [11]: "hello"+"world"
Out[11]: 'helloworld'

In [12]:
```

1.4-1 语法入门

- 懒人的python：能少敲就少敲点
  - 句子一般不用分号结束
  - 缩进代替C语言中的花括号{ }（有点霸道！但对养成良好的编程习惯有很大帮助，程序结构变得很清晰）。因此行首的空白是有意义的
  - 用“#”开始的行作为注释

```
# test.py using python

a=1

if a==1:

    print(a)
```

```
/* test.c using C*/

#include<stdio.h>

void main (){

    int a;

    a=1;

    if (a==1)

        printf(“%d”,a);

}
```

1.4-1 语法入门

- 强悍的数据结构之列表(list): 最常用，灵活，功能强
  - 用中括号[ ]表示
    - ✓a=[1,2,3,4]
    - ✓a=[1,2,3.4,'a','hello']
    - ✓a=[[1,2],[3,4]]
    - ✓[1,2]+[3,4]+['a','b']
  - 用下标法引用一个元素
    - ✓若a=[1,2,3,4]，与C语言相同，下标从0开始，a[0]的值为1，a[3]为4，a[-1]为4
  - 用切片法引用子列
    - ✓若a=[1,2,3,4]，那么a[0:2]为列表[1,2]，a[1:-1]为列表[2,3]

1.4-1 语法入门

- 强悍的数据结构之字符串
  - 用单引号或双引号括起来的一组字符
    - ✓ ‘a’，“a”，’ helo‘
    - ✓ “Hello”与‘Hello’等价
    - ✓ ‘hello’*3，‘hello’+‘hi’
  - 用下标法获取其中字符
    - ✓若a=“string”，a[0]为字符’s’，a[5]为字符’g’，a[-1]为字符’g’
  - 用切片法截取子串
    - ✓若a=“string”, a[0:3]为子串“str”，a[1:-1]为子串”trin”

1.4-1 语法入门

- 选择语句
  - if ***:
    ******
  - if ***:
    ******
    else:
    ******
  - if ***:
    ******
    elif ***:
    ******
    else:
    ******
  - 条件表达式

```
a=1
if a==1:
```

```
a=1
if a==1:
    print("hello")
else:
    print("byebye")
```

```
elif a>1:
    print("hi")
```

C语言版本：x = A ? B : C

python版本：x = B if A else C

1.4-1 语法入门

- 循环结构
  - for x in ***:
    ****
  - while ***:
    ****

```
for x in [1,2,3]:
    print(x)
```

```
n=5
print(range(n))
for x in range(n):
    print(x)
```

```
i=10
while i>=1:
    print(i**2)
    i-=1
```

```
for x in [1,2,3]:
    for y in “abcd”:
        print(y*x)
```

1.4-1 语法入门

函数

- def语句定义

```
def 函数名(形式参数表):
    函数体
```

- 调用

```
函数名(实际参数表)
```

1.4-1 语法入门

- 函数分类
  - 有参函数
  - 无参函数
  - 有返回值函数
  - 无返回值函数
  - 什么都不做的函数

```
#有参有返回值函数sum：
def sum(x,y):
    z=x*x+y*y
    return z
```

```
#有参有返回值函数num：
def num(n):
    return range(1,n)
```

```
#无参无返回值函数myprint：
def myprint():
    print("hello world")
```

```
#什么都不做函数fun：
def fun():
    pass
```

1.4-2 自学Python

- 恭喜! 你已经入门python；接下去，完全可以自学成为一名优秀的pythoner
  - 啄木鸟社区

    http://woodpecker.org.cn/
  - 简明python教程

    http://woodpecker.org.cn/abyteofpython_cn/chinese/
  - Dive Into Python 中文版

    http://woodpecker.org.cn/diveintopython/toc/index.html
  - Elias的python学习资料http://www.elias.cn/Python/HomePage
  - Python自带手册python manuals以及第三方库(pypi)

指定教材？！
还不如上网search…

1.4-3 简单例子

- 例：计算在小于n的正整数中，能被3或5整除的所有数之和。例如，n=10，结果是3,5,6,9满足要求，和值为23。若取n=1000，和值为多少？

C语言代码？

头脑会变大！

```
#ch1: sum_divide_3_5.py
s=0
for i in range(1,1000):
    if i%3==0 or i%5==0:
        s+=i
print(s)

ls=[i for i in range(1,1000) if i%3==0 or i%5==0]
print(sum(ls))
```

方法1

方法2

1.5 Python语言风格

- Python语法的几个显著特征
  - 变量类型动态、无需声明、随时定义
  - 行首缩进表示语句块
  - 没有指针，数据结构功能强大
  - 没有switch选择语句, 没有do *** while循环结构
  - 函数随时随地可以定义，定义中形参无需类型声明，任何类型数据都可以成为参数，返回值类型无需声明，
  - 函数返回值可以是一个整数、一个字符串、一个列表

1.6 课程安排

- 课程讲授内容：
  - Ch1 Python语言入门概览
  - Ch2 数据结构
  - Ch3 控制流与函数编程
  - Ch4文件操作与数据存取
  - Ch5 模块与面向对象编程
  - Ch6 Python数据科学工具
  - Ch7 Python扩展与性能优化
  - Ch8 正则表达式
  - Ch9 网络爬虫
  - 总复习

1.6-1 教材推荐

- Wesley J. Chun著，孙波翔，李斌，李晗译，Python核心编程（第3版），人民邮电出版社，2016
- Wesley J. Chun 著，宋广吉 译，Python核心编程（第二版），人民邮电出版社，2008
- Magnus L. Hetland著，司维等译，Python基础教程(第2版)，人民邮电出版社，2014

1.6-2 参考教材

- Laura Cassell，Alan Gauld著， 高弘扬、卫莹 译，Python项目开发实战，清华大学出版社，2015
- （美）麦金尼　著，唐学韬　等译，利用Python进行数据分析，机械工业出版社，2014
- 张若愚 著，Python科学计算，清华大学出版社，2012.

1.6-3 成绩

- 期末总成绩构成：
  - 平时成绩40%+期末考试 60%
- 平时成绩构成：
  - 4次上机实验
    - ✓计划安排在第5, 9, 13, 16周，地点：信息楼机房
  - 平时作业（5次左右）
  - 考勤+课堂表现

第1章 Python语言入门概览

主讲人：张 华
浙江工商大学计算机与信息工程学院

提纲

- 1.1 Python语言概述
- 1.2 Python能做什么？
- 1.3 Python的安装、运行与IDE
- 1.4最简单的Python代码
- 1.5 Python语言风格
- 1.6 课程安排

1.1 Python语言概述

- 听说过Python吗？

1.1-1 Python的面世

- 1989年,Guido van Rossum在阿姆斯特丹完成，第一个公开版发行于1991年。
- Guido为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，做为ABC语言的一种继承
  - ABC是由Guido参加设计的一种教学语言非常优美和强大，是专门为非专业程序员设计的
  - 使用Python作为语言的名字，因为是英国幽默剧团:"Monty Python飞行马戏团"的fans

创始人：

吉多·范罗苏姆

Guido van Rossum

1.1-1 Python的面世

- Python 2.0于2000年10月16日发布，主要是实现了完整的垃圾回收，并且支持Unicode。
- Python 3.0于2008年12月3日发布，此版不完全兼容之前的Python源代码。
- 目前Python2使用最广泛的版本是2.7，Python3的最新版本为3.7.2（截止2019.02）
- Python3不兼容Python2。本课程仅考虑Python3.

1.1-2 Python解释器

- Python是一门跨平台的脚本语言，Python规定了一个Python语法规则，实现Python语法的解释程序就成为Python的解释器
  - CPython（ClassicPython, 最原始python的实现，需要区别于其他实现的时候才以CPython称呼;或用C语言实现的的Python）
  - Jython （Java语言实现的Python ）
  - Ironpython (面向.NET和ECMA CLI的Python实现)
  - PyPy (使用Python语言写的Python)
  - Zhpy（支持繁/简中文语句编写程序的Python语言）

1.1-3 Python的特色

- 容易上手
  - 提供交互式环境
  - 语法简洁
    - ✓高级数据结构简洁地表达复杂的操作
    - ✓语句组织依赖于缩进
    - ✓参数或变量不需要声明
- 火力强大
  - 易学但不简单，从桌面程序，到网络互联，图形处理，科学计算，机器学习/深度学习，实时控制，到处都有Python的身影
  - 跨平台的（Windows, Unix, Macantoshi）
  - 面向对象

1.1-3 Python的特色

- 快速开发
  - Python内建的高层次数据结构，以及动态类型和动态邦定，非常适合于快速应用开发
  - Python语法强调可读性，降低了程序的维护费用
  - Python支持模块和包，并鼓励程序模块化和代码重用。

1.1-3 Python的特色

- 丰富的库
  - Python 标准库已经很庞大。可帮你处理各种工作：正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、 FTP、电子邮件、XML文件、GUI和其他与系统有关的操作。
  - Python开源、免费，在“百花齐放”式地发展中，已经产生大量的高质量库，如Pygame、 matplotlib 、scipy、numpy、sklearn、tensorflow、pytorch等等
- 可扩展、可嵌入
  - 对性能有特殊要求的部分代码可用C或C++编写
  - 可以把Python嵌入你的C/C++程序，从而向你的程序用户提供脚本功能
- 解释性
  - python程序不需要编译成二进制代码，可以直接从源代码运行程序。使得Python程序更加易于移植。

1.2 Python能做什么?

- Python既具备高性能且能使编程简单有趣的特点
- Python能做什么呢?
  - 系统编程
  - 用户图形接口
  - Internet通信
  - 数据库编程
  - 数值计算和科学计算
  - 游戏开发、图像处理、人工智能、机器人等
  - 。。。。。。。

Python确实是一种十分精彩又强大的语言。
Python似乎是万能的。

1.2-1 Python可做的事

- Internet通信
  - Python提供了标准Internet模块，可以广泛应用到各种网络任务中，无论服务端还是在客户端
  - 套接字
  - XML-RPC、SOAP、Telnet等通信
  - Twisted
  - 网站开发工具：HTMLGen、mod_python、Django、TurboGears、Zop，使Python能够快速构建功能完善和高质量的网站

- 数据库编程
  - 支持所有主流数据库：Oracle、Sybase、MySQL、PostgreSQL、Informix、SQLite
  - 定义了标准的、可移植的数据库API
  - Pickle模块：永久存储数据
  - 面向对象数据库系统：ZODB
  - 从关系数据库映射到Python类（ORM）： SQLAlchemy 、SQLObject

1.2-1 Python可做的事

- 数据分析与数据科学
  - 科学计算：NumPy，Scipy
  - 数据分析：Pandas
  - 画图：matplotlib
  - 数据挖掘：scikit-learn，nltk，pytorch/tensorflow(深度学习)等
  - 大数据：支持hadoop、spark等大数据平台的python编程

- 游戏、图像处理、机器人等
  - 游戏：Pygame/Bigworld
  - 图像：PIL
  - 机器人控制：PyRO

1.2-2 Python和其他语言的比较

- 比TCL强大，支持“大规模编程”，适宜于开发大型系统
- 比Perl语法简洁，更具可读性、更易于维护，有助于减少Bug
- 比Java更简单、更易于使用
- 比C++更简单、更易于使用，不与之竞争
- 比VB更强大也更具备跨平台特性。(中学开始推广Python课程)
- 比Ruby更成熟、语法更具可读性

1.2-3 谁在用Python?

典型几个国外公司

1.2-3 谁在用Python?

国内谁在用Python

豆瓣douban 知乎 易度云办公 everydo.com 搜狐 SOHU.com

网易游戏 NIE.163.COM Sina App Engine 腾讯网 QQ.com

42 果壳 科技有意思 Guokr.com 快玩游戏 kuaiwan.com

典型几个国内公司

1.3 Python的安装、运行与IDE

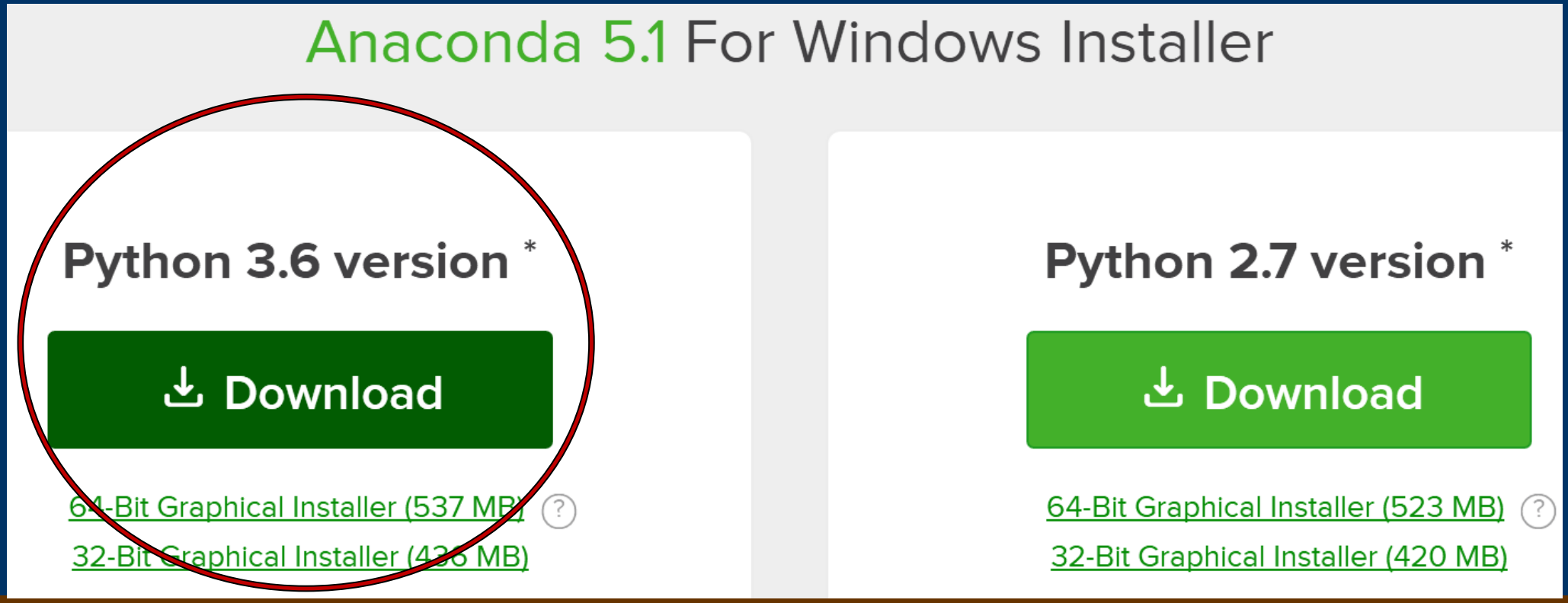
- Python官网(https://www.python.org/downloads/)下载界面

1.3-1 Python的安装

- 安装方式(一)：较为繁琐不推荐！
  - 官网下载Python核心
    - ✓python-2.7.14.msi
    - ✓python-3.6.4.msi　推荐！
  - 常用第三方库下载
    - ✓Python package index (pypi):
      https://pypi.python.org/pypi
    - ✓numpy、 scipy　科学计算
    - ✓pandas　数据分析
    - ✓matplotlib　二维、三维画图
    - ✓pygame　游戏开发
    - ✓wxpython　图形用户界面开发
    - ✓django　web开发
    - ✓scikit-learn　数据挖掘

1.3-1 Python的安装

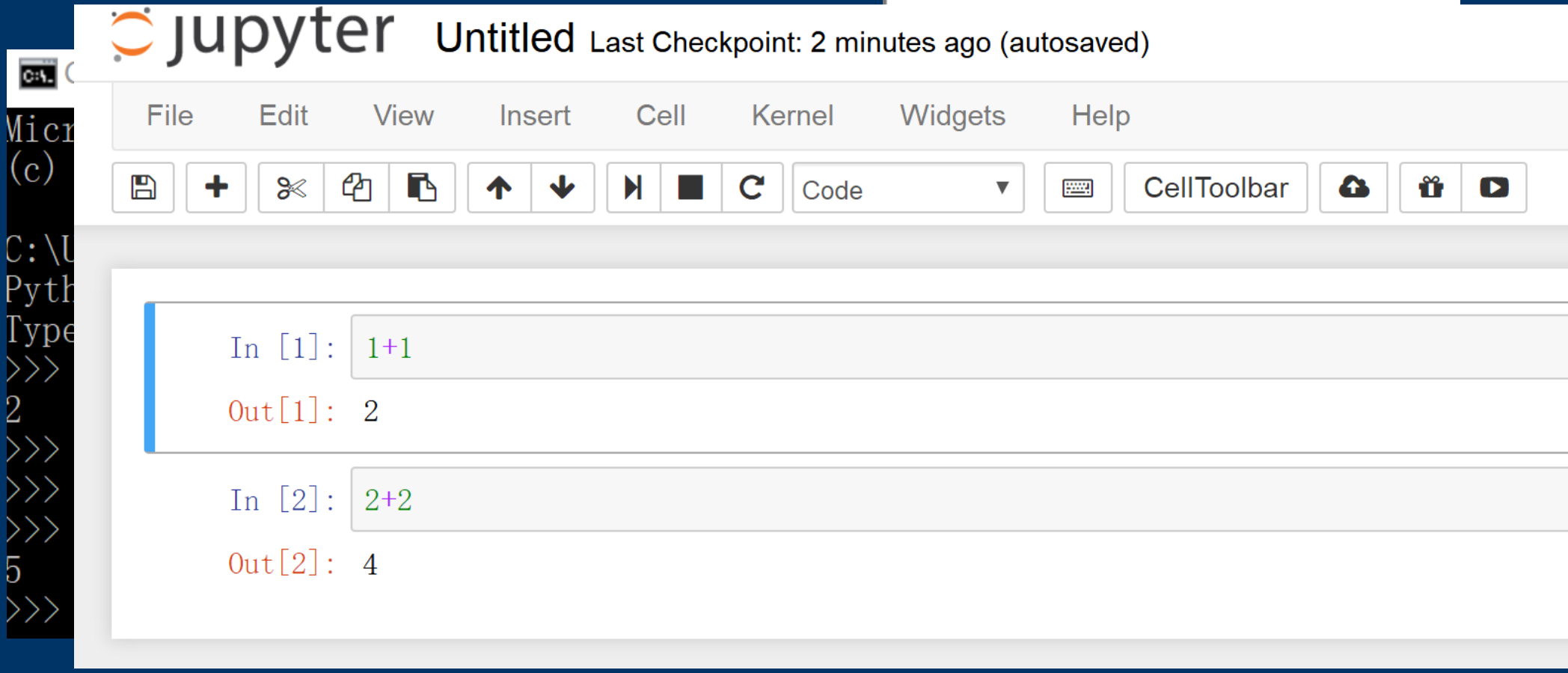
- 安装方式(二)---强烈推荐！
  - 安装Anaconda
  - 官网：https://www.continuum.io/why-anaconda
  - 集成1000多个Python科学计算包
  - 安装完Anaconda，就相当于安装了Python、IPython、集成开发环境Spyder、交互式笔记本Jupyter Notebook、科学计算numpy和scipy、可视化matplotlib、数据分析Pandas等等



1.3-2 Python的运行

交互式shell

- 方式1：打开cmd，敲入python，或使用pyhton自带的ipython
- 方式2：IDE自带交互式shell，如spyder中的shell
- 方式3：Jupyter notebook(功能强大的交互式笔记本)



1.3-2 Python的运行

- 编写源文件
  - Python源文件是以“.py”为后缀的文本文件
  - 可以用任何编辑文本的软件进行编写，如记事本，UltraEdit，EditPlus等
  - 也可以使用专业的python开发环境编写，如IDLE，Spyder，PyCharm等

- 运行源文件
  - 如编写名为“hi.py”的python源文件，内容如下：
    print(“hello world!”)
  - 直接在编辑器（开发环境）中执行，如spyder（anaconda自带）
  - 命令行形式：python hi.py

1.3-3 Python开发环境IDE

- IDLE：Python内置IDE (随python安装包提供)

- PyCharm：由著名的JetBrains公司开发，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工 具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发，推荐！

- Spyder：Spyder是Python(x,y)的作者为它开发的一个简单的集成开发环境。和其他的Python开发环境相比，它最大的优点就是模仿MATLAB的“工作空间”的功能，可以很方便地观察和修改数组的值。安装Anaconda时自带，强烈推荐！

  - 注意，PyCharm community版本免费，Spyder在安装Anaconda时自带。

1.4最简单的Python代码

- Python的设计哲学：

- Python开发者的哲学：

“用一种方法，最好是只有一种方法来做一件事”。

在设计Python语言时，如果面临多种选择，Python开发者总会拒绝花哨的语法，而选择明确的没有或者很少有歧义的语法。

1.4-1 语法入门

- 变量
  - 使用前不用类型声明
  - 在使用过程中类型可以动态变化

- 标识符命名规则：
  - 必须有字母（大小写）、数字、下划线组成，但首字符必须是字母或下划线
    例如，_abc，abc， abc_1为合法标识符，而 1_abc， abc$， @#!为不合法标识符
  - 标识符大小写敏感
    例如，Abc 与 abc 不是一个标识符
  - 标识符不能为 Python 保留字

1.4-1 语法入门

- 表达式
  - 2+3, 100000000000000000000+1
  - 2+3*5，3**5
  - 2/3, 2//3, 5//4.5
  - “hello”+‘world’

- 逻辑表达式
  - C语言：　&&　||　!
  - Python：　and　or　not
  - Python例子：a==1 and b==5

```
In [8]: 3**5
Out[8]: 243

In [9]: 2/3
Out[9]: 0.6666666666666666

In [10]: 2//3
Out[10]: 0

In [11]: "hello"+"world"
Out[11]: 'helloworld'

In [12]:
```

1.4-1 语法入门

- 懒人的python：能少敲就少敲点
  - 句子一般不用分号结束
  - 缩进代替C语言中的花括号{ }（有点霸道！但对养成良好的编程习惯有很大帮助，程序结构变得很清晰），因此行首的空白是有意义的
  - 用“#”开始的行作为注释

```
# test.py using python
a=1
if a==1:
    print(a)
```

```
/* test.c using C*/
#include<stdio.h>
void main (){
    int a;
    a=1;
    if (a==1)
        printf(“%d”,a);
}
```

1.4-1 语法入门

- 强悍的数据结构之列表(list): 最常用，灵活，功能强
  - 用中括号[ ]表示
    - ✓ a=[1,2,3,4]
    - ✓ a=[1,2,3.4,'a','hello']
    - ✓ a=[[1,2],[3,4]]
    - ✓ [1,2]+[3,4]+['a','b']
  - 用下标法引用一个元素
    - ✓ 若a=[1,2,3,4]，与C语言相同，下标从0开始，a[0]的值为1，a[3]为4，a[-1]为4
  - 用切片法引用子列
    - ✓ 若a=[1,2,3,4]，那么a[0:2]为列表[1,2]，a[1:-1]为列表[2,3]

1.4-1 语法入门

- 强悍的数据结构之字符串
  - 用单引号或双引号括起来的一组字符
    - ✓ ‘a’，“a”，’ helo‘
    - ✓ “Hello”与‘Hello’等价
    - ✓ ‘hello’*3，‘hello’+‘hi’
  - 用下标法获取其中字符
    - ✓若a=“string”，a[0]为字符’s’，a[5]为字符’g’，a[-1]为字符’g’
  - 用切片法截取子串
    - ✓若a=“string”, a[0:3]为子串“str”，a[1:-1]为子串”trin”

1.4-1 语法入门

- 选择语句
  - if ***:
    ******
  - if ***:
    ******
    else:
    ******
  - if ***:
    ******
    elif ***:
    ******
    else:
    ******
  - 条件表达式

```
a=1
if a==1:
```

```
a=1
if a==1:
    print("hello")
else:
    print("byebye")
```

```
elif a>1:
    print("hi")
```

C语言版本：x = A ? B : C

python版本：x = B if A else C

1.4-1 语法入门

- 循环结构
  - for x in ***:
    ****
  - while ***:
    ****

```
for x in [1,2,3]:
    print(x)
```

```
n=5
print(range(n))
for x in range(n):
    print(x)
```

```
i=10
while i>=1:
    print(i**2)
    i-=1
```

```
for x in [1,2,3]:
    for y in “abcd”:
        print(y*x)
```

1.4-1 语法入门

函数

- def语句定义

```
def 函数名(形式参数表):
    函数体
```

- 调用

```
函数名(实际参数表)
```

1.4-1 语法入门

- 函数分类
  - 有参函数
  - 无参函数
  - 有返回值函数
  - 无返回值函数
  - 什么都不做的函数

```
#有参有返回值函数sum：
def sum(x,y):
    z=x*x+y*y
    return z
```

```
#有参有返回值函数num：
def num(n):
    return range(1,n)
```

```
#无参无返回值函数myprint：
def myprint():
    print("hello world")
```

```
#什么都不做函数fun：
def fun():
    pass
```

1.4-2 自学Python

- 恭喜! 你已经入门python；接下去，完全可以自学成为一名优秀的pythoner
  - 啄木鸟社区

    http://woodpecker.org.cn/
  - 简明python教程

    http://woodpecker.org.cn/abyteofpython_cn/chinese/
  - Dive Into Python 中文版

    http://woodpecker.org.cn/diveintopython/toc/index.html
  - Elias的python学习资料http://www.elias.cn/Python/HomePage
  - Python自带手册python manuals以及第三方库(pypi)

指定教材？！
还不如上网search…

1.4-3 简单例子

- 例：计算在小于n的正整数中，能被3或5整除的所有数之和。例如，n=10，结果是3,5,6,9满足要求，和值为23。若取n=1000，和值为多少？

C语言代码？

头脑会变大！

```
#ch1: sum_divide_3_5.py
s=0
for i in range(1,1000):
    if i%3==0 or i%5==0:
        s+=i
print(s)

ls=[i for i in range(1,1000) if i%3==0 or i%5==0]
print(sum(ls))
```

方法1

方法2

1.5 Python语言风格

- Python语法的几个显著特征
  - 变量类型动态、无需声明、随时定义
  - 行首缩进表示语句块
  - 没有指针，数据结构功能强大
  - 没有switch选择语句, 没有do *** while循环结构
  - 函数随时随地可以定义，定义中形参无需类型声明，任何类型数据都可以成为参数，返回值类型无需声明，
  - 函数返回值可以是一个整数、一个字符串、一个列表

1.6 课程安排

- 课程讲授内容：
  - Ch1 Python语言入门概览
  - Ch2 数据结构
  - Ch3 控制流与函数编程
  - Ch4文件操作与数据存取
  - Ch5 模块与面向对象编程
  - Ch6 Python数据科学工具
  - Ch7 Python文本分析
  - Ch8 正则表达式
  - Ch9 网络爬虫
  - Ch10 Python扩展与性能优化
  - 总复习

1.6-1 教材推荐

- Wesley J. Chun著，孙波翔，李斌，李晗译，Python核心编程（第3版），人民邮电出版社，2016
- Wesley J. Chun 著，宋广吉 译，Python核心编程（第二版），人民邮电出版社，2008
- Magnus L. Hetland著，司维等译，Python基础教程(第2版)，人民邮电出版社，2014

1.6-2 参考教材

- Laura Cassell，Alan Gauld著， 高弘扬、卫莹 译，Python项目开发实战，清华大学出版社，2015
- （美）麦金尼　著，唐学韬　等译，利用Python进行数据分析，机械工业出版社，2014
- 张若愚 著，Python科学计算，清华大学出版社，2012.

1.6-3 成绩

- 理论课期末总成绩构成：
  - 平时成绩40%+期末闭卷考试 60%
- 实验课期末总成绩构成：
  - 平时成绩40%+期末大报告60%
- 平时成绩构成：
  - 上机实验（35%）
  - 作业（35%）
  - 考勤+课堂表现(30%)

第2章 数据结构

主讲人：张 华
浙江工商大学计算机与信息工程学院

提纲

- 2.1 数据类型与运算符
- 2.2 列表
- 2.3 字符串
- 2.4 元组
- 2.5 字典
- 2.6 集合

2.1 数据类型与运算符

- 仅仅打印“Hello World”就足够了吗？我们应该想要做更多的事——想要得到一些输入，然后做操作，再从中得到一些输出。在Python中，我们可以使用常量和变量，考虑数据类型与结构，利用运算符来完成这些工作。

2.1-1 标识符与变量

- 变量是标识符的例子。命名规则基本与C语言相同，不再赘述。
- 关键字

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

2.1-2 数据类型

- 数字
  - 整数　integer, long,boolean
  - 浮点数　float, complex
- 基本数据结构
  - 序列　sequence
    - ✓列表　list
    - ✓字符串　string
    - ✓元组　tuple
  - 映射
    - ✓字典　dictionary
  - 集合　set

2.1-3 运算符

基本运算符

运算符	名称	说明	例子
+	加	两个对象相加	3 + 5得到8。'a' + 'b'得到'ab'。
-	减	得到负数或是一个数减去另一个数	-5.2得到一个负数。50 - 24得到26。
*	乘	两个数相乘或是返回一个被重复若干次的字符串	2 * 3得到6。'la' * 3得到'lalala'。
**	幂	返回x的y次幂	3 ** 4得到81（即3 * 3 * 3 * 3）
/	除	x除以y	4/3得到1.3333333333333333。而在Python2中4/3得到整数1
//	取整除	返回商的整数部分	4 // 3.0得到1.0
%	取模	返回除法的余数	8%3得到2。-25.5%2.25得到1.5

2.1-3 运算符

比较运算符

运算符	名称	说明	例子
<	小于	返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意变量名的大写。	5 < 3返回0（即False）而3 < 5返回1（即True）。比较可以被任意连接：3 < 5 < 7返回True。
>	大于	返回x是否大于y	5 > 3返回True。如果两个操作数都是数字，它们首先被转换为一个共同的类型。否则，它总是返回False。
<=	小于等于	返回x是否小于等于y	x = 3; y = 6; x <= y返回True。
>=	大于等于	返回x是否大于等于y	x = 4; y = 3; x >= y返回True。
==	等于	比较对象是否相等	x = 2; y = 2; x == y返回True。x = 'str'; y = 'stR'; x == y返回False。x = 'str'; y = 'str'; x == y返回True。
!=	不等于	比较两个对象是否不相等	x = 2; y = 3; x != y返回True。

2.1-3 运算符

逻辑运算符

运算符	名称	说明	例子
not	布尔“非”	如果x为True，返回False。如果x为False，它返回True。	x = True; not y返回False。
and	布尔“与”	如果x为False，x and y返回False，否则它返回y的计算值。	x = False; y = True; x and y，由于x是False，返回False。在这里，Python不会计算y，因为它知道这个表达式的值肯定是False（因为x是False）。这个现象 称为短路计算。
or	布尔“或”	如果x是True，它返回True，否则它返回y的计算值。	x = True; y = False; x or y返回True。

2.2 列表

- 定义
  - 是Python中最具灵活性的有序集合对象类型。列表可以包含任何种类的对象：数字、字符串、自定义对象甚至其他列表
- 特点
  - 是可变对象（动态数组），支持在原处修改，可以通过指定的偏移值和切片、列表方法调用、删除语句等方法实现

2.2-1 列表常量与基本操作

操作	解释
L1=[]	一个空的列表
L2 = [0, 1, 2, 3]	四元素列表
L3 = [‘abc’,10,[‘def’, ‘ghi’]]	嵌套列表
L2[i]	索引
L3[i][j]	索引的索引
L2[i:j]	切片
len(L2)	求长度
L1 + L2	合并
L2 * 3	重复

2.2-2 列表的方法

- **append(x) 方法**
  - 功能：向列表的尾部添加一个元素

```
>>> a=[]
>>> a.append(6)
>>> a.append(7)
>>> a
[6, 7]
>>> a.append([12,15])
>>> a
[6, 7, [12, 15]]
>>>
```

2.2-2 列表的方法

- extend(ls)
  - 功能：列表扩充

```
>>> a=[1,2,3]
>>> b=[5,6,7,[1,2]]
>>> a.extend(b)
>>> a
[1, 2, 3, 5, 6, 7, [1, 2]]
>>>
```

2.2-2 列表的方法

- **insert(i,x)**
  - 向列表指定位置插入一个元素

注意，这是Python2中的range

```
>>> x=range(10)
>>> x
[0, 1, 2, 3, 4, 5,
>>> x.insert(2,"a"
>>> x
[0, 1, 'a', 2, 3,
>>> x.insert(5,199
>>> x
[0, 1, 'a', 2, 3,
>>> x.insert(0,[33
>>> x
[[33, 44], 0, 1, '
>>>
```

Python3中的range，返回一个可迭代的对象

```
In [1]: a=range(10)

In [2]: a
Out[2]: range(0, 10)

In [3]: x=list(a)

In [4]: x
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [5]: y=[i for i in a]

In [6]: y
Out[6]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.2-2 列表的方法

- remove(x)
  - 功能：删除列表中值为x的第一个元素
- pop([i])
  - 将列表中指定位置元素删除，并将其返回。如果没有指定索引(下标)，a.pop()返回最后一个元素。

```
>>> x=[1,2,3,4,5,6,1,2,3]
>>> x.remove(2)
>>> x
[1, 3, 4, 5, 6, 1, 2, 3]
>>> x.pop(2)
4
>>> x
[1, 3, 5, 6, 1, 2, 3]
>>>
```

2.2-2 列表的方法

- index(x)
  - 返回列表中第一个值为x的元素索引。如果没有匹配的元素就会返回一个错误

```
>>> a=[1,2,5,6,2,3]
>>> a.index(2)
1
>>> a.index(10)

Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    a.index(10)
ValueError: 10 is not in list
>>>
```

2.2-2 列表的方法

- count(x)
  - 计算元素x在列表中出现的次数

```
>>> a=[1,2,5,2,3,5,5]
>>> a.count(5)
3
>>> b=a.count(10)
>>> b
0
>>> a[:4].count(2)
2
>>>
```

2.2-2 列表的方法

- sort()
  - 对列表中的元素进行适当的排序。
- reverse()
  - 倒排列表中的元素。

```
>>> a=[5,8,12,9,24,56,10]
>>> a.sort()
>>> a
[5, 8, 9, 10, 12, 24, 56]
>>> a.reverse()
>>> a
[56, 24, 12, 10, 9, 8, 5]
>>>
```

2.2-3 列表的性质

- 列表是一种序列
- 列表中元素类型任意，可以嵌套列表
- 列表是可变的
- 别名问题
  - a引用一个对象，而b=a，两个变量会引用同一个变量

```
>>> a=[58,12,23,5]
>>> b=a
>>> b is a
True
>>> b[0]=1
>>> a
[1, 12, 23, 5]
>>>
```

2.3 字符串

- 单引号
- 双引号：功能与单引号一样，如’c’与”c”是相同的
- 三引号
  - 可以表示多行

```
>>> s='''How are you?
         I am fine. And you? '''
>>> s
'How are you?\n         I am fine. And you? '
>>> x="How are you?
SyntaxError: EOL while scanning string literal
>>>
```

2.3-1 转义字符

- 与C语言类似

转义	意义
\ newline	行连续
\\	反斜线
\’	单引号
\”	双引号
\a	响铃
\b	倒退
\f	换页
\n	新行

2.3-2 字符串基本操作

- 字符串合并：”how”+”what”
- 字符串重复：”how”*3
- 索引、切片
  - X[I]
  - X[I:J]
  - X[I:J:K], K表示步长

```
>>> x="how are you?"
>>> x[1:10:2]
'o r o'
>>>
```

2.3-3 字符串转化

- 数字字符串转化成数字
  - 情形1：整数
  - 情形2：浮点数

```
>>> s='42'
>>> s+1

>>> ss="3.14"
>>> int(ss)

Traceback (most recent call last):
  File "<pyshell#86>", line 1, in <module>
    int(ss)
ValueError: invalid literal for int() with base 10: '3.14'
>>> float(ss)
3.14
>>>
>>> int(s)+10
52
```

2.3-4 字符串格式化

1.占位符%

字符串格式化：占位符%

```
s1="There are %d %s"%(10,'books')
s2="There are %d birds"
a=20
print(s1)
print(s2%(a))
s3='''My name is %s.
    Your name is %s'''
print(s3%('Zhang','Li'))
pi=3.1415926
print("PI:%.2f,PI:%10.3f"%(pi,pi))
```

```
There are 10 books
There are 20 birds
My name is Zhang.
    Your name is Li
PI:3.14,PI:     3.142
```

符号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%f 和 %E 的简写
%p	用十六进制数格式化变量的地址

2.3-4 字符串格式化

- 1.占位符%
  - 格式化操作符辅助指令

符号	功能
*	定义宽度或者小数点精度
-	用做左对齐
+	在正数前面显示加号(+)
<sp>	在正数前面显示空格
#	在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话)

2.3-4 字符串格式化

2. format函数

- Python2.6 开始，新增函数 **str.format()**，增强格式化功能。
- 基本语法是通过 **{}** 和 **:** 来代替以前的 **%** 。
- format 函数可以接受任意多个参数，位置可以不按顺序。

```
>>>"{} {}".format("hello", "world")    # 不设置指定位置，按默认顺序
'hello world'

>>> "{0} {1}".format("hello", "world")  # 设置指定位置
'hello world'

>>> "{1} {0} {1}".format("hello", "world")  # 设置指定位置
'world hello world'
```

2.3-4 字符串格式化

2. format函数

- 字符串的参数使用{NUM}进行表示, 0表示第一个参数, 1表示第二个参数, 以后顺次递加;
- 使用":", 指定代表元素需要的操作, 如":.3"小数点三位, ":8"占8个字符空间等;
- 数字(0, 1, ...)即代表format()里面的元素, 所以可以使用"."调用元素的方法;

字符串格式化：**format**

```
age,name = 25,'Caroline'
print('{0} is {1} years old. '.format(name, age)) #输出参数
print('{0} is a girl. '.format(name))
print('{0:.3} is a decimal. '.format(1/3)) #小数点后三位
print('{0:_^11} is a 11 length. '.format(name)) #使用_补齐空位
print('{first} is as {second}. '.format(first=name, second='Wendy')) #别名替换
print('My name is {0.name}'.format(open('out.txt', 'w'))) #调用方法
print('My name is {0:8}.'.format('Fred')) #指定宽度
```

```
Caroline is 25 years old.
Caroline is a girl.
0.333 is a decimal.
__Caroline__ is a 11 length.
Caroline is as Wendy.
My name is out.txt
My name is Fred    .
```

2.3-4 字符串格式化

数字	格式	输出	描述
3.1415926	{:.2f}	3.14	保留小数点后两位
3.1415926	{:+.2f}	+3.14	带符号保留小数点后两位
-1	{:+.2f}	-1.00	带符号保留小数点后两位
2.71828	{:.0f}	3	不带小数
5	{:0>2d}	05	数字补零 (填充左边, 宽度为2)
5	{:x<4d}	5xxx	数字补x (填充右边, 宽度为4)
10	{:x<4d}	10xx	数字补x (填充右边, 宽度为4)
1000000	{:,}	1,000,000	以逗号分隔的数字格式
0.25	{:.2%}	25.00%	百分比格式
1000000000	{:.2e}	1.00e+09	指数记法
13	{:>10d}	13	右对齐 (默认, 宽度为10)
13	{:<10d}	13	左对齐 (宽度为10)
13	{:^10d}	13	中间对齐 (宽度为10)
11	'{:b}'.format(11) '{:d}'.format(11) '{:o}'.format(11) '{:x}'.format(11) '{:#x}'.format(11) '{:#X}'.format(11)	1011 11 13 b 0xb 0XB	进制

2.3-5 字符串的方法

- 字母大小写变换
  - capitalize()：将首字母大写，其余均为小写
  - title()：将单词首字母大写，其余均为小写

```
>>> s="pRediction of Protein"
>>> s.capitalize()
'Prediction of protein'
>>> s.title()
'Prediction Of Protein'
>>> s.lower()
'prediction of protein'
>>> s.upper()
'PREDICTION OF PROTEIN'
>>> s.swapcase()
'PrEDICTION OF pROTEIN'
>>> s
'pRediction of Protein'
>>>
```

2.3-5 字符串的方法

- 字符串对齐填充

```
>>> s='str'
>>> s.center(7,"*")
'**str**'
>>> s.ljust(7,"*")
'str****'
>>> s.rjust(7,"*")
'****str'
>>> s.zfill(7)
'0000str'
>>> s
'str'
>>> s.center(7,'hi')

Traceback (most recent call last):
  File "<pyshell#123>", line 1, in <module>
    s.center(7,'hi')
TypeError: must be char, not str
>>> |
```

2.3-5 字符串的方法

- 分割与组合
  - split([sep,[maxsplit]]):以sep为分隔符，分割成list，maxsplit表示分割次数，从左开始
  - rsplit([sep,[maxsplit]]):同上，但分割从右开始
  - splitlines([keepends]):以行分隔符分割成list，keepends取bool值，如果为真，每行分隔符会保留
  - join(seq): 把seq中元素串接成

```
>>> s="prediction for proteins"
>>> s.split()
['prediction', 'for', 'proteins']
>>> s.split('i')
['pred', 'ct', 'on for prote', 'ns']
>>> s.split('i',2)
['pred', 'ct', 'on for proteins']
>>> s.rsplit('i',2)
['predict', 'on for prote', 'ns']
>>> ss='''how are you?
I am fine.'''
>>> ss.splitlines()
['how are you?', 'I am fine.']
>>> ss.splitlines(1)
['how are you?\n', 'I am fine.']
>>> ",".join(s.split())
'prediction,for,proteins'
>>> "****".join(s.split(','))
'prediction for proteins'
>>> "****".join(s.split('i'))
'pred***ct***on for prote***ns'
>>>
```

2.3-5 字符串的方法

查找

- find(substr, [start, [end]])：返回字符串中出现substr的第一个字

```
>>> s="hello world!"
>>> s.find('o')
4
>>> s.rfind('o')
7
>>> s.find('p')
-1
>>> s.index('p')

Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    s.index('p')
ValueError: substring not found
>>> s.count('o')
2
>>>
```

没有

数

2.3-5 字符串的方法

- 替换
  - replac str
    ，cou 特
    殊字符
  - strip([理
    解为把
  - lstrip(
  - rstrip(
  - expan
    tab替换

```
>>> s='*** hello world ***'
>>> s.replace('o','y')
'*** helly wyrld ***'
>>> s.replace('o','y',1)
'*** helly world ***'
>>> s.strip('*')
' hello world '
>>> s.strip('*').strip()
'hello world'
>>> s
'*** hello world ***'
>>> s.strip('* ')
'hello world'
>>> s.lstrip('*')
' hello world ***'
>>> s.rstrip('*')
'*** hello world '
>>> |
```

2.3-5 字符串的方法

- 判断
  - startwith(prefix[,start[,end]])：是否以prefix开头
  - endwith(suffix[,start[,end]])：以suffix结尾
  - isalnum()：是否全是字母和数字，并至少有一个字符
  - isalpha()：是否全是字母，并至少有一个字符、
  - isdigit()：是否全是数字，并至少有一个字符
  - isspace()：是否全是空白字符，并至少有一个字符
  - islower()：字符串中的字母是否全是小写
  - isupper()：字符串中的字母是否便是大写
  - istitle()：字符串是否是首字母大写的

2.3-6 字符串性质

- 字符串是一种序列
- 字符串是不可变的
  - 修改字符串后就会重新产生一个新字符串

```
>>> s='hello'
>>> s[0]='c'

Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    s[0]='c'
TypeError: 'str' object does not support item assignment
>>> |
```

2.4 元组

- 定义：由数个逗号分隔的值组成，如
  - t=(1,2,3)
  - x=(‘c’,’h’,12,’2’,(1,2,3))
- 性质
  - 元组在输出时总是有括号的
  - 元组在输入时可以没有括号
  - 像字符串一样，元组是不可变的
  - 元组可以嵌套，元素可以是任意类型

```
>>> t=1,2,3
>>> t
(1, 2, 3)
>>> t[0:2]
(1, 2)
>>> t[0]=3

Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    t[0]=3
TypeError: 'tuple' object does not support item assignment
>>>
```

2.5 字典

定义

- 通过关键字索引的无序数据集合
- 例如：`d={'age':12, 'name':'bob'}`

性质

- 序列是以整数为索引，与此不同，字典以关键字（键）为索引
- 关键字可以是任何不可变类型，通常用字符串或数字
- 不能用列表作为关键字，因为列表随时可能被改变
- 关键字必须是互不相同的

2.5 字典

键和值

- keys(): 取所有键
- values()：取所有键所对应的值
- items(): 取所有键与值对
- get(key):取得键key所对应的值

```
In [21]: tel={'jack':4089,'shy':4139}

In [22]: tel
Out[22]: {'jack': 4089, 'shy': 4139}

In [23]: tel['gree']=4127

In [24]: tel
Out[24]: {'gree': 4127, 'jack': 4089, 'shy': 4139}

In [25]: tel.keys()
Out[25]: dict_keys(['shy', 'gree', 'jack'])

In [26]: tel.values()
Out[26]: dict_values([4139, 4127, 4089])

In [27]: tel.items()
Out[27]: dict_items([('shy', 4139), ('gree', 4127),
('jack', 4089)])

In [28]: tel['jack']
Out[28]: 4089
```

2.6 集合

集合的定义

- 集合(set)是一个无序的，不重复的数据组合
- 集合类型有可变集合(set())和不可变集合(frozenset)两种
- 主要作用如下：
  - ✓去重：把一个列表变成集合，就自动去重了
  - ✓关系测试：测试两组数据之间的交集、差集、并集等关系
- 集合对象还支持union(联合), intersection(交), difference(差)和sysmmetric difference(对称差集)等数学运算

```
1 # ch2: set.py
2 a={"peace","peace","rong","rong","nick"}
3 print("a:",a)
4 b=set(["peace","peace","rong","rong"])
5 print("b:",b)
6
7 print("并集:",a|b)#演示联合
8 print("交集:",a&b)#演示交
9 print("差集:",a-b) #演示差
10 print("对称差:",a^b)#对称差集
```

第3章 控制流与函数编程

主讲人：张 华
浙江工商大学计算机与信息工程学院

提纲

- 3.1 条件语句if
- 3.2 循环语句
- 3.3 列表解析与生成器表达式
- 3.4 函数
- 3.5 lambda函数
- 3.6 内置函数

3.1 条件语句if

- if语句是选取要执行的操作，是Python主要的选择工具，代表Python程序所拥有的大多数逻辑。
- if语句是复合语句，同其他复合语句一样，if语句可以包含其他语句
- if通用格式：

```
if <test1>:
    <statements1>
elif <test2>:
    <statements2>
else:
    <statements3>
```

3.1-1 if语句

- Python中没有switch、case语句
- elif和else都可省略，elif可有多个，作用等同于C/C++中的switch
- 可以用多个 列表

```
#ch3: if_operator.py
def jia(x,y):
    print(x+y)
def jian(x,y):
    print(x-y)
def cheng(x,y):
    print(x*y)
def chu(x,y):
    print(x/y)
operator = {'+':jia,'-':jian,'*':cheng,'/':chu}
def f(x,o,y):
    operator.get(o)(x,y)
f(3,'+',2)
```

3.1-1 if语句

- 真值测试/条件测试语句
  - 在Python中，与大多数程序设计语言一样，整数0代表假，1代表真。不过，除此之外，Python也把任意的空数据结构视为假。
  - 更一般的，真和假的概念是Python中每个对象的固有属性：每个对象不是真就是假
- 测试语句返回值：
  - 数字如果非零，则为True
  - 对象如果非空，则为True
  - 数字零、空对象以及特殊对象None都被认作是假
  - 比较和相等测试会返回True或False
  - 逻辑运算and和or会返回作为操作对象

```
>>> 2<3
True
```

```
>>> 2 or 3
2
>>> 0 or 2
2
>>> [] or 'hello'
'hello'
>>> [] or {}
{}
```

```
>>> 2 and 3
3
>>> 2 and []
[]
>>> [] and 3
[]
>>> [] and {}
[]
```

3.1-2 三元表达式

- C语言三元表达式：
  - `A = X?Y:Z`
- Python表达：
  - if语句：

```
if X:
    A = Y
else:
    A = Z
```

  - 三元表达式
    - ✓`A = Y if X else Z`

```
>>> a = 't' if 'spam' else 'f'
>>> a
't'
>>> a = 't' if '' else 'f'
>>> a
'f'
```

3.2 循环语句

- while循环
- for循环
- break、continue、pass语句

3.2-1 while循环

while一般格式：

```
while <test>:       #Loop test
   <statements1>  #Loop body
else:          #Optional else
   <statements2>  #Run if didn't exit loop with break
```

```
a = 0
b = 10
while a<b:
   print(a)
   a = a + 1
```

- 循环中的else块：
  只有当循环正常离开（测试条件为假）时才会执行，如果循环被break语句中断，则else块不会执行。

3.2-2 for循环

- for循环在Python中是一个通用的序列迭代器：可以遍历任何有序的序列对象内的元素。
- for一般格式：

```
for <target> in <object>:#Assign object items to target
   <statements>   #Repeated loop body:use target
else:
   <statements>   #If we didn't hit a 'break'
```

- 例子：

```
>>> for x in ['a', 'b', 'c']:
...     print(x)
...
a
b
c
```

```
>>> for i in range(0, 5):
...     print(i)
...
0
1
2
3
4
```

3.2-2 for循环

- 循环嵌套
  - 循环的嵌套是指在一个循环中又包含另外一个完整的循环，即循环体中又包含循环语句。**while**循环和**for**循环可以相互嵌套。

- 循环嵌套执行过程
  - 一次外循环对应着完整的一轮内循环。

3.2-3 break、continue和pass

- break语句会立即退出当前循环，不会执行else块的内容。
- continue语句会忽略后面的语句，强制进入下一次循环。
-

```
#ch3:break_continue.py
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
print('Done')
```

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        print ('length is not sufficient')
        continue
    print('Length of the string is', len(s))
```

3.2-4 try-except语句

```
while 1:
        name=input("Enter name:")
        if name=='stop': break
        age=input("Enter age:")
        print('Hello',name,'=>',int(age)**2)
```

- 当age不输入任何值时会报错。可使用try-except解决（异常处理）

```
#ch3:try_except.py
while 1:
    name=input("Enter name:")
    if name=='stop':  break
    age=input("Enter age:")
    try:
        print('Hello',name,'=>',int(age)**2)
    except:
        print('Please input correct age!')
```

3.3 列表解析与生成器表达式

- 列表解析：在一个序列的值上应用一个任意表达式，将其结果收集到一个新的列表中并返回。基本语法形式：
  - [expr for iter_var in iterable]
  - [expr for iter_var in iterable if cond_expr]
- 列表解析不是在解决一个新问题，而只是提供一种新的语法来解决一个已经存在的问题。

```
>>> L= [(x+1,y+1) for x in range(3) for y in range(5)]
>>> L
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5),
(3, 1), (3, 2), (3, 3), (3, 4), (3, 5)]
>>> N=[x+10 for x in range(10) if x>5]
>>> N
[16, 17, 18, 19]
```

3.3 列表解析与生成器表达式

- 生成器表达式：每次只需要获取一个元素时，考虑使用生成器表达式而不是列表解析。使用语法：
  - (expr for iter_var in iterable) 或
  - (expr for iter_var in iterable if cond_expr)

- 生成器表达式与列表解析语法相似，在处理大数据量时，生成器表达式的优势就体现出来了，因为它的内存使用方式更好，效率更高，它并不创建一个列表，只是返回一个生成器。

例子：

```
>>>[i + 1 for i in range(10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>(i +1 for i in range(10) if i % 2)
<generator object <genexpr> at 0x011DC5D0>
```

```
>>> g = (i + 1 for i in range(10) if i % 2)
>>> l = []
>>> for j in g:
        l.append(j)
>>> l
[2, 4, 6, 8, 10]
```

3.4 函数

- 函数概念
  - 函数是一个能够完成特定功能的代码块，可在程序中重复使用，减少程序的代码量和提高程序的执行效率
- Python定义语法：
  -
```
def function_name([arg1],[arg2],…):
    [expression]
    return [value]
```
  - 说明：
    - ✓函数首部先是def，然后函数名，后面紧跟圆括号，最后是冒号，作为函数体开始标志；
    - ✓圆括号中的一些变量名，为函数提供输入

3.4-1 函数例子

```
2 #ch3:funs.py
3 def numcmp(a,b):
4     if a>b: return 1
5     elif a==b: return 0
6     else: return -1
7 print(numcmp(8,9),numcmp(12,1),numcmp(23,23))
8
9 print(numcmp(a=13,b=78))
10
```

3.4-2 函数参数默认值

- def function(x,y,z=12):
  - 其中z=12定义了函数的默认参数
- 默认参数规则：
  - 带默认参数函数定义时，不允许先声明有默认值的参数，而后声

```
12 def func(a=5,b):
13     if a>b: print(a)
14 func(5,10)
15
```

c(a=5,b)无效。

```
    def func(a=5,b):
            ^
SyntaxError: non-default argument follows default
argument
```

```
17 def func(a,b=10):
18     if a>b: print(a)
19 func(20,10)
20
```

3.4-3 参数定义方式

- f(x1,x2,…)：没有默认值
- f(x1,x2=value,…)：提供默认值

```
22 def f(*x):
23     if len(x)==0: print('None')
24     else: print(x)
25 f(1)
26 f(1,2,3)
27 f()
28 f('t',1,'hello')
29
```

```
(1,)
(1, 2, 3)
None
('t', 1, 'hello')
```

在名为x的字典中

```
31 def fd(**x):
32     if len(x)==0: print('None')
33     else: print(x)
34 fd()
35 fd(x=1,y=2,z='c')
36
37 fd(1,2)
38
```

```
None
{'x': 1, 'y': 2, 'z': 'c'}
```

```
fd(1,2)

TypeError: fd() takes 0 positional
arguments but 2 were given
```

3.4-4 函数调用

- 如果将前面四种方式依次编号为1,2,3,4，那么函数参数在调用过程中被解析的顺序为: 先1，后2，再3，最后4

```
def test(x,y=1,*a,**b):
    print(x,y,a,b)
test(1)
test(1,2)
test(1,2,3,4)
test(x=1,y=2)
test(1,a=2)
test(1,2,3,a=4)
test(1,2,3,c=4)
test(1,2,3,y=4)
```

输出结果:

```
1 1 () {}
1 2 () {}
1 2 (3, 4) {}
1 2 () {}
1 1 () {'a': 2}
1 2 (3,) {'a': 4}
1 2 (3,) {'c': 4}
```

```
    test(1,2,3,y=4)

TypeError: test() got multiple values
for argument 'y'
```

3.5 lambda函数

lambda表达式

- 返回可调用的函数对象
- **lambda**是一个**表达式**，而**def**是一个**语句**
- 如s=lambda x: x*x，等价于

  def s(x):

  return x*x

```
>>> s=lambda x: x*x
>>> s(8)
64
>>> s(20)
400
>>> t=lambda x,y: x*x+y+1
>>> t(3)

Traceback (most recent call last):
  File "<pyshell#63>", line 1, in <module>
    t(3)
TypeError: <lambda>() takes exactly 2 arguments (1 given)
>>> t(3,4)
14
>>> |
```

3.5 lambda函数

应用例子

- 排序sort：利用lambda函数定义顺序规则
  - ✓逆向排序
  - ✓多维数据，排序以谁为准？

r中

```python
#sort_lambda.py
nums = [1, 3, 2, 4]
nums.sort()
print("Sorted numbers1:",nums)
nums.sort(reverse=True)
print("Sorted numbers2:",nums)

```

```
Sorted numbers1: [1, 2, 3, 4]
Sorted numbers2: [4, 3, 2, 1]
```

```python
dat = [['Li',10],['Hu',30],['Yu',20],['Ji',40]]
dat.sort()
print("Sorted list1:")
print(dat)
dat.sort(key=lambda pair: pair[1])
print("Sorted list2:")
print(dat)
dat.sort(key=lambda pair: pair[1],reverse=True)
print("Sorted list3:")
print(dat)

```

```
Sorted list1:
[['Hu', 30], ['Ji', 40],
['Li', 10], ['Yu', 20]]
Sorted list2:
[['Li', 10], ['Yu', 20],
['Hu', 30], ['Ji', 40]]
Sorted list3:
[['Ji', 40], ['Hu', 30],
['Yu', 20], ['Li', 10]]
```

3.6 内置函数

- 输入输出：input( )、print( )
- 计算函数：sum( )、len( )、max( )、min( )
- 映射类函数：map( )、filter( )、reduce( )
- 其它常用函数：range( ), enumerate( ), zip( )
  - zip函数：接受任意多个可迭代对象作为参数,将对象中对应的元素打包成一个tuple, 然后返回一个可迭代的zip对象

```python
#somefuns.py
a=[5,7,9,1]
print(sum(a),len(a))
print()
for x in a:
    print(x)
print()
for i, x in enumerate(a):
    print(i,x)
print()
b=[11,22,33,44]
print(zip(a,b))
for x,y in zip(a,b):
    print(x,y)
print()
c=['Are','you','OK','?']
for x,y,z in zip(a,b,c):
    print(x,y,z)
```

```
>>> a1=[1,2,3]
>>> a2=[4,5,6]
>>> a3=[7,8,9]
>>> a4=["a","b","c","d"]
>>> zip1=zip(a1,a2,a3)
>>> print(zip1)
<zip object at 0x7f5a22651c08>
>>> for i in zip1:
...     print(i)
...
(1, 4, 7)
(2, 5, 8)
(3, 6, 9)
```

```
22 4

5
7
9
1

0 5
1 7
2 9
3 1

<zip object at 0x0000026C1A9B3108>
5 11
7 22
9 33
1 44

5 11 Are
7 22 you
9 33 OK
1 44 ?
```

3.6 内置函数

- map(function, sequence [, sequence])
  - 说明：对sequence中的item依次执行function(item)，执行结果输出一个可迭代对象，或说返回类型为iterators
  - 单个序列
  - 多个序列

```
In [8]: def add(n):
   ...:     return n+n
   ...:

In [9]: b=map(add,range(3,10))

In [10]: list(b)
Out[10]: [6, 8, 10, 12, 14, 16, 18]

In [11]: c=map(lambda x:x+x,range(2,20,3))

In [12]: list(c)
Out[12]: [4, 10, 16, 22, 28, 34]

In [13]: d=map(add,"Python")

In [14]: list(d)
Out[14]: ['PP', 'yy', 'tt', 'hh', 'oo', 'nn']
```

```
In [5]: a=map(str,range(5))

In [6]: a
Out[6]: <map at 0x19cd353a080>

In [7]: list(a)
Out[7]: ['0', '1', '2', '3', '4']
```

3.6 内置函数

- filter(function or None, sequence)
  - 说明：对sequence中的item依次执行function(item)，将执行结果为True(!=0)的item组成一个iterators返回，对False者进行过滤
  - 例子：求出m~n之间的所有素数

```
In [30]: prime=filter(lambda x: not [x%i for i in range(2,x)
if x%i==0],range(3,20))

In [31]: prime
Out[31]: <filter at 0x19cd3531b00>

In [32]: list(prime)
Out[32]: [3, 5, 7, 11, 13, 17, 19]

In [33]:
```

利用not(拓展到补集的思路来求质数:list)

```
In [24]: def div(n):
    ...:     return n%2
    ...:

In [25]: af=filter(div,range(10))

In [26]: af
Out[26]: <filter at 0x19cd3531320>

In [27]: list(af)
Out[27]: [1, 3, 5, 7, 9]

In [28]: bf=filter(lambda x:x%2,range(10))

In [29]: list(bf)
Out[29]: [1, 3, 5, 7, 9]
```

3.6 内置函数

- reduce(function, iterator)
  - 参数function必须有两个参数
  - 函数作用是：把结果继续和序列的下一个元素做累积计算。

```
In [33]: from functools import reduce

In [34]: def add(x,y):
    ...:     return x+y
    ...:

In [35]: reduce(add,[1,2,3,4,6])
Out[35]: 16

In [36]:
```

第4章 文件操作与数据存取

主讲人：张 华
浙江工商大学计算机与信息工程学院

提纲

- 4.1 文件操作
- 4.2 文件系统处理
- 4.3 数据永久存储

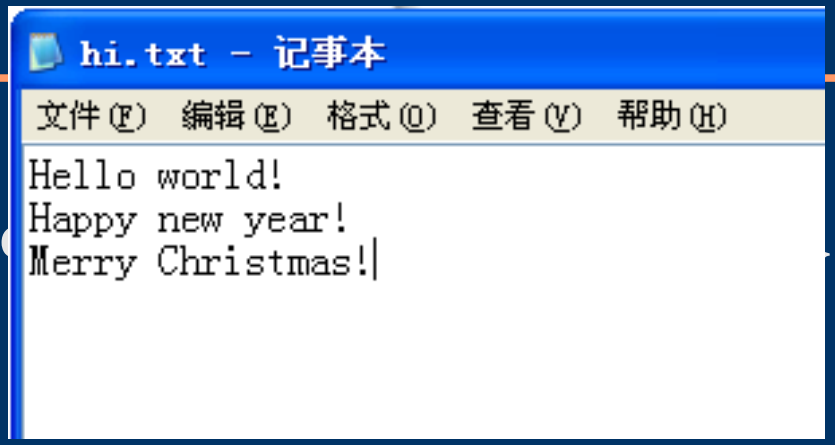
4.1 文件操作

- 输入与输出
  - 程序与用户交互，从用户那里得到输入，然后打印出一些结果。可分别使用input和print语句完成

- 另一种输入输出类型：文件。
  - 读文件：获取数据
  - 写文件：保存数据

4.1-1 读文件

- 打开文件
  - open(“filename”)或open(“fil



- 读取方法:
  - read()：读取全部字符，返回一个字符串
  - readline()：读取一行，返回一个字符串
  - readlines()：读取所有行，返回字符串列表

```
>>> fin=open("D:\\hi.txt")
>>> fin.readline()
'Hello world!\n'
>>> fin.readline()
'Happy new year!\n'
>>> fin.readline()
'Merry Christmas!'
>>> fin.readline()
''
>>> fin=open("D:\\hi.txt")
>>> fin.readlines()
['Hello world!\n', 'Happy new year!\n', 'Merry Christmas!']
>>> fin.readline()
''
>>>
```

```
>>> fin=open("D:\\hi.txt")
>>> xx=fin.read()
>>> xx
'Hello world!\nHappy new year!\nMerry Christmas!'
>>> xx.split()
['Hello', 'world!', 'Happy', 'new', 'year!', 'Merry', 'Christmas!']
>>> xx
'Hello world!\nHappy new year!\nMerry Christmas!'
>>> xx.split('\n')
['Hello world!', 'Happy new year!', 'Merry Christmas!']
>>>
```

4.1-2 写文件

- 打开文件
  - open(‘filename’,’w’)创建新文件，并只允许写
  - open(‘filename’,’a’) 在已经存在的文件中追加内容

- 写入方法
  - write(all_text)
  - writelines(list)

```
>>> fout=open("D:\\a.txt",'w')
>>> fout.write('hello world\nmerry christmas\n')
>>> fout.close()
>>> fout=open("D:\\b.txt",'w')
>>> fout.writelines(['hello world','merry christmas'])
>>> fout.close()
>>>
```

```
>>> fout=open("D:\\c.txt",'w')
>>> for x in range(5,10):
        for y in range(8,20):
                fout.write("%10d%10d\n"%(x,y))


>>> fout.close()
>>>
```

- 关闭文件：文件对象.close( )

4.1-3 with语句

- 处理文件可能发生的两个问题：
  - 可能忘记关闭文件句柄
  - 文件读取数据发生异常，没有进行任何处理

```
file = open("/tmp/foo.txt")
try:
    data = file.read()
finally:
    file.close()
```

太复杂！

with 版本：

```
with open("/tmp/foo.txt") as file:
    data = file.read()
```

4.2 文件系统处理

- 文件系统是操作系统用于明确存储设备(常见的是磁盘，也有基于NAND Flash的固态硬盘)或分区上的文件的方法和数据结构
- Linux、Mac OSX、 Windows等文件系统是最为常见的文件管理工具
- Python标准库则允许我们从Python内部管理文件，虽不如操作系统命令直接，但有其优势：
  - 利用Python语言，并发挥其他Python工具或扩展包，形成组合的文件管理功能
- os包、shutil包

4.2-1 os包

- os包含有各式各样的函数，以实现操作系统的许多功能；这个包非常庞杂，用于文件管理，这里列出最常用的：
  - os.mkdir(*path*)：创建新目录，path为一个字符串，表示新目录路径。相当于$mkdir命令
  - os.rmdir(*path*)：删除空目录，path为一个字符串，表示要删除的目录路径。相当于$rmdir命令
  - os.listdir(*path*)：返回目录中所有文件。相当于$ls命令。
  - os.remove(*path*)：删除path指向的文件。
  - os.rename(*src, dst*)：重命名文件，src和dst为两个路径，分别表示重命名之前和之后的路径。

4.2-1 os包

- os 包常用方法(续)：
  - os.chmod(*path, mode*)：改变path指向的文件的权限。相当于$chmod命令。
  - os.getcwd()：查询当前工作路径 (cwd, current working directory)，相当于$pwd命令。
  - os.chdir(path)：改变当前工作目录
  - os.system()：执行shell命令

```
>>> os.system('cmd') #启动dos
```

4.2-2 os.path模块

- os.path模块主要用于文件属性的获取，在不同的环境中设置文件路径时作用非常大。常用方法如下：
  - os.path.dirname(__file__)：返回当前python执行脚本的执行路径（

```
2 #ch4: ospath.py
3 import os
4 from os.path import join
5 path1 = os.path.dirname(__file__)
6 print('The directory of this .py file is:\n', path1)
7 print()
8
9 newdir=join(path1,'hello')
10 os.mkdir(newdir)
11 for f in os.listdir(path1):
12     print(f)
13
14 print()
15 for f in os.listdir(path1):
16     print(os.path.abspath(f))
17
```

4.2-2 os.path模块

- **os.path模块常用方法(续)：**
  - os.path.split(*path*)：函数返回一个路径的目录名和文件名
  - os.path.isfile(*path*)和os.path.isdir(*path*)函数分别检验给出的路径是一个文件还是目录。
  - os.path.**exists**(*path*)：用来检验给出的路径是否真地存在
  - os.path.basename(path):返回路径path中的文件名
  - os.path.dirname(path):返回path中的文件夹路径
  - os.path.splitext():分离文件名与扩展名
  - os.path.getsize(name):获得文件大小，如果name是目录返回0L

4.3 数据永久存储

- 在运行过程中，python程序会产生一些字符串、列表、字典或其它对象等数据，有时我们需要长久保存下来，方便以后使用，而不是简单的放入内存中关机断电就丢失数据。
- Pickle模块应运而生，它可以将python数据对象转换为一种可以传输或存储的格式。
- python的pickle模块实现了基本的数据序列和反序列化。
  - 序列化操作：能够将程序中运行的对象信息保存到文件中去，永久存储；
  - 反序列化操作：能够从文件中创建上一次程序保存的对象。
- pickle模块使用的数据格式是python专用的，并且不同版本不向后兼容，同时也不能被其他语言所识别。

4.3 数据永久存储

- pickle 模块提供了以下 4 个函数供我们使用：
  - dumps(obj)：将 python 对象序列封装成二进制对象，并返回；
  - loads(bytes_obj)：读取给定的二进制对象数据，并将其转换为Python对象。
  - dump(obj, file, protocol=None)：将 Python 中的对象序列obj转化成二进制对象，并写入文件file。
  - load(file)：读取指定的序列化数据文件file，并返回原先永久存储的python对象
- 以上 4 个函数可以分成两类：
  - dumps 和 loads 实现基于内存的 Python 对象与二进制互转

```
#ch4:pickle_dump_load.py
import pickle
tup0 = ('I love Python', {"HZ":5,"SX":4,"OT":3},[7,8],None)

pick1 = pickle.dumps(tup0) #将tup0转成二进制对象
mytup = pickle.loads(pick1)#将pick1转成Python对象
print(mytup)
```

4.3-1 dump和load

- **pickle.dump(obj, file, protocol=None)**
  - 含义：将对象obj保存到文件file中去
  - **protocol**：pickle 的转码协议，取值为 0、1、2、3、4，其中 0、1、2 对应 Python 早期的版本，3 和 4 则对应 Python 3.x 版本及之后的版本。未指定情况下，默认为 3。
  - file：保存指定的二进制文件。file必须有write()接口，要求该文件必须是以“wb”的打开方式进行操作。

- **pickle.load(file)**
  - load与dump函数相对应，用于将二进制对象文件转换成 Python 对象。
  - file: 参数file表示要转换的二进制对象文件， 且必须以 "rb" 的打开方式操作文件

4.3-1 dump和load

- 例子：可以存储任何类型的Python对象，如函数

```
#ch4:pickle_dump_load.py
import pickle

def fsq(x):
    return x**2+5
tup1 = ('Python', {"HZ":5,"SX":4,"OT":3},fsq, None)

with open ("a.txt", 'wb') as f: #打开文件
    pickle.dump(tup1, f) #用dump将Python对象转成二进制对象文件
with open ("a.txt", 'rb') as f: #打开文件
    tup2 = pickle.load(f) #将二进制文件对象转换成Python对象
print(tup2)
x=10
print("fsq(%d)="%x,tup2[2](x))
```

```
('Python', {'HZ': 5, 'SX': 4, 'OT': 3}, <function fsq at
0x00000274CF515F28>, None)
fsq(10)= 105
```

第5章 模块与面向对象编程

主讲人：张 华
浙江工商大学计算机与信息工程学院

提纲

- 5.1 模块
- 5.2 Package(包)
- 5.3 面向对象编程
- 5.4 类

5.1 模块

代码管理

- 使用交互式环境，当退出python解释器后再次进入，以前创建的一切定义（变量和函数）不复存在。为使程序长久，代码可重复使用，较好办法是用文本编辑器(开发环境)来编写程序。
- 程序变长了，不好管理，有时把长的程序分成多个文件。
- 有时想要在多个程序中使用一个常用函数或变量数据，但不想把定义复制到每一个程序中。
- 为此，python提供一种方法可以从“*.py” 文件中获取变量或函数定义，在脚本或交互式实例中使用。这样的文件被称为模块。

Python模块

- 一般对应于一个程序文件(py文件)，每个py文件都是一个模块

5.1-1 模块导入

- 模块导入的几种方式：
  - import modname 或 import modname as mn
  - from modname import funcname

```
>>> import math
>>> math.sqrt(10)
3.1622776601683795
>>> math.exp(2)
7.38905609893065
>>> math.cos(200)
0.48718767500700594
>>> math.e
```

```
>>> from math import sqrt
>>> sqrt(10)
3.1622776601683795
>>> cos(10)

Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    cos(10)
NameError: name 'cos' is not defined
>>>
```

```
    sqrt(10)
NameError: name 'sqrt' is not defined
>>>
```

5.1-2 标准库模块

random模块

- 用于生成随机数

```
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'Syste
mRandom', 'TWOPI', 'WichmannHill', '_BuiltinMethodType', '_MethodType', '__all_
_', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '_acos',
'_ceil', '_cos', '_e', '_exp', '_hashlib', '_hexlify', '_inst', '_log', '_pi',
'_random', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn', 'b
etavariate', 'choice', 'division', 'expovariate', 'gammavariate', 'gauss', 'get
randbits', 'getstate', 'jumpahead', 'lognormvariate', 'normalvariate', 'paretov
ariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuff
le', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']
>>> random.random()
0.4048024621218488
>>> random.uniform(10,20)
19.305872606276502
>>> random.uniform(10,20)
13.038511787462792
>>> random.randint(10,20)
19
>>> random.randint(10,20)
11
>>> random.choice("python")
'y'
>>> random.choice(["python","java",".net","C#","C++"])
'java'
>>> p=["python","java",".net","C#","C++"]
>>> random.shuffle(p)
>>> p
```

5.1-2 标准库模块

time

- 提供各种操作时间的函数

```
>>> import time
>>> dir(time)
['__doc__', '__name__', '__package__', 'accept2dyear', 'altzone', 'asctime', 'c
lock', 'ctime', 'daylight', 'gmtime', 'localtime', 'mktime', 'sleep', 'strftime
', 'strptime', 'struct_time', 'time', 'timezone', 'tzname']
>>> time.time()
1396239111.718
>>> time.ctime()
'Mon Mar 31 12:12:02 2014'
>>> time.sleep(5)
>>>
```

5.1-2 标准库模块

- string
  - 提供各种处理字符串的函数

```
>>> dir(string)
['Formatter', 'Template', '_TemplateMetaclass', '__builtins__', '__doc__', '__f
ile__', '__name__', '__package__', '_float', '_idmap', '_idmapL', '_int', '_lon
g', '_multimap', '_re', 'ascii_letters', 'ascii_lowercase', 'ascii_uppercase',
'atof', 'atof_error', 'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize',
 'capwords', 'center', 'count', 'digits', 'expandtabs', 'find', 'hexdigits', 'i
ndex', 'index_error', 'join', 'joinfields', 'letters', 'ljust', 'lower', 'lower
case', 'lstrip', 'maketrans', 'octdigits', 'printable', 'punctuation', 'replace
', 'rfind', 'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitfields', 'str
ip', 'swapcase', 'translate', 'upper', 'uppercase', 'whitespace', 'zfill']
>>> s="hello world"
>>> string.capitalize(s)
'Hello world'
>>> string.capwords(s)
'Hello World'
>>> string.count(s,"h")
1
>>> string.find(s,'a')
-1
>>> string.split('hello world')
['hello', 'world']
>>>
```

5.1-2 标准库模块

```
>>> os.getcwd()
'C:\\Python27'
>>> os.chdir("d:\\")
>>> os.mkdir("myhello")
>>> os.chdir("myhello")
>>> os.getcwd()
'd:\\myhello'
>>> os.mkdir("zhang")
>>> os.mkdir("wang")
>>> os.mkdir("li")
```

os

- **os.getcwd()：得到当前工作目录**
- os.chdir(“dirname”)：改变当前脚本工作目录到dirname
- os.listdir(“dirname”)： 列出指定目录下所有文件和子目录，包括隐藏文件
- os.mkdir(“dirname”))：生成单级目录
- os.rmdir(“dirname”))：删除单级空目录，若目录不空则无法删除
- os.walk(“dirname”)：遍历目录
- os.remove()： 删除一个文件
- os.rename(“oldname”,“newname”)： 重命名文件
- os.system(command)： 运行shell命令

```
>>> import os
>>> dir(os)
['F_OK', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM',
 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL', 'O_SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT',
 'O_TRUNC', 'O_WRONLY', 'P_DETACH', 'P_NOWAIT', 'P_NOWAITO', 'P_OVERLAY', 'P_WAI
T', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'UserDict', 'W_OK', '
X_OK', '_Environ', '__all__', '__builtins__', '__doc__', '__file__', '__name__',
 '__package__', '_copy_reg', '_execvpe', '_exists', '_exit', '_get_exports_list'
, '_make_stat_result', '_make_statvfs_result', '_pickle_stat_result', '_pickle_s
tatvfs_result', 'abort', 'access', 'altsep', 'chdir', 'chmod', 'close', 'closera
nge', 'curdir', 'defpath', 'devnull', 'dup', 'dup2', 'environ', 'errno', 'error'
, 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execvpe'
, 'extsep', 'fdopen', 'fstat', 'fsync', 'getcwd', 'getcwdu', 'getenv', 'getpid',
 'isatty', 'kill', 'linesep', 'listdir', 'lseek', 'lstat', 'makedirs', 'mkdir',
'name', 'open', 'pardir', 'path', 'pathsep', 'pipe', 'popen', 'popen2', 'popen3'
, 'popen4', 'putenv', 'read', 'remove', 'removedirs', 'rename', 'renames', 'rmdi
r', 'sep', 'spawnl', 'spawnle', 'spawnv', 'spawnve', 'startfile', 'stat', 'stat_
float_times', 'stat_result', 'statvfs_result', 'strerror', 'sys', 'system', 'tem
pnam', 'times', 'tmpfile', 'tmpnam', 'umask', 'unlink', 'unsetenv', 'urandom', '
utime', 'waitpid', 'walk', 'write']
```

Python3中改为

```
>>> os.listdir("d:\\myhello")
['zhang', 'wang', 'li']
>>> fin=open("1.txt",'w')
>>> fin.write("hello world!\n")
>>> fin.close()
>>> os.listdir("d:\\myhello")
['zhang', 'wang', 'li', '1.txt']
>>> for x,y,z in os.walk("d:\\myhello"):
        print x,y,z


d:\myhello ['zhang', 'wang', 'li'] ['1.txt']
d:\myhello\zhang [] []
d:\myhello\wang [] []
d:\myhello\li [] []
>>>
```

5.1-2 标准库模块

- os.path
  - os.path.basename():输出文件名
  - os.path.dirname()：输出文件所在的路径
  - os.path.split():分割文件名与路径，返回（fpath,fname）元组
  - os.path.splitdrive():
  - os.path.join():将分离部分组成一个文件路径名
  - os.path.exists()　指定路径（文件或者目录）是否存在
  - os.path.isabs()　指定路径是否为绝对路径
  - os.path.isdir()　指定路径是否存在且为一个目录
  - os.path.isfile()

```
>>> from os.path import *
>>> fname="D:\\myhello\\1.txt"
>>> basename(fname)
'1.txt'
>>> dirname(fname)
'D:\\myhello'
>>> split(fname)
('D:\\myhello', '1.txt')
>>> splitdrive(fname)
('D:', '\\myhello\\1.txt')
>>> join('D:','myhello','1.txt')
'D:myhello\\1.txt'
>>> exists(fname)
True
>>> isfile(fname)
True
>>> isdir(fname)
False
>>> isdir("D:\\myhello")
True
>>> isabs("D:\\myhello")
True
>>>
```

5.1-3 自定义模块

- 创建.py文件
  - 例如：模块m1即是python文件m1.py
  - 考虑在fibo.py中使用模块m1

```python
#ch5: m1.py
PI=3.1415
year=2014

def fib(n): #print Fibonacci series up to n-1
    a,b=0,1
    while b<n:
        print(b,)
        a,b=b,a+b

def fib2(n): #return Fibonacci series up to n-1
    result=[]
    a,b=0,1
    while b<n:
        result.append(b)
        a,b=b,a+b
    return result
```

```python
#ch5:fibo.py
import m1

print(m1.PI)
m1.fib(13)
print(m1.fib2(50))
```

5.1-4 模块的功能小结

- 代码重用
- 区分系统命名空间
- 实现数据和服务共享

5.2 包（package）

- 在创建许许多多模块后，我们可能希望将某些功能相近的文件组织在同一文件夹下，这里就需要运用包的概念了。
- 把一堆相关的python模块放在一个目录下，再加上一个__init__.py文件就构成了一个python 包，python包可以嵌套
- 包对应于文件夹，使用包的方式跟模块也类似，唯一需要注意的是，当文件夹当作包使用时，文件夹需要包含__init__.py文件，主要是为了避免将文件夹名当作普通的字符串。
- 使用方法：
  - import Sound.Effects.echo
  - import Sound.Effects.echo as see
  - from Sound.Effects import echo
  - from Sound.Effects.echo import *

```
Sound/                          Top-level package
      __init__.py               Initialize the sound package
      Formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      Effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
```

5.3 面向对象编程

- Python从设计之初就已经是一门面向对象的语言，因此，在Python中创建一个类和对象是很容易的。

- 对象
  - 对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象。
  - Python对象实现数据和操作的结合，使数据和操作封装于对象的统一体中。

- 类
  - 具有相同或相似性质的对象的抽象就是类。因此，对象的抽象是类，类的具体化就是对象，也可以说类的实例是对象。
  - 类具有数据属性和操作。

5.4 类

- 类：一种数据结构，封装了属性和方法

- 类的定义：
  - class ClassName(object):

    ‘class string documentation string’ #文档字符串

    class_suite #类体

- 面向对象编程：数据与方法结合在一起，用对象的方式组织程序

- 面向过程编程：根据操作数据的函数或语句块

5.4-1 类的定义

- 最简单的类定义形式：

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

  - 类的定义语句（class）和函数定义语句（def）一样，执行后才能生效；当然也可以放到if语句的某个分支，或一个函数的内部
  - 类定义完成时（正常退出），就创建了一个类对象。

- 类对象支持两种操作：属性引用和实例化。

5.4-3 类的实例对象

类的实例化

- 类的实例化方式：`x = Class1()`，创建一个新的类实例并将该对象赋给局部变量x.
- 很多类倾向于将对象创建为有初始状态的情况。因此类可定义一个名为__init__()的特殊方法（构造方法），像下面这样：

```
def __init__(self):
    self.data = []
```

- 类的实例化会自动为新创建的类实例调用__init__() 方法。
- __init__() 方法可以有参数，例如：

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
x = Complex(3.0, -4.5)
print(x.r, x.i)
```

类的实例化

5.4-4 例子

```
>>> from random import randrange
>>> class msdie:
        '''Class definition for a multiple-side die'''
        def __init__(self,s):
                self.sides=s
                self.value=1
        def roll(self):
                self.value=randrange(1,self.sides+1)
        def getvalue(self):
                return self.value
        def setvalue(self,v):
                self.value=v


>>> die=msdie(6) #创建一个msdie类对象
>>> die.getvalue() #调用对象die的getvalue()方法
1
>>> die.roll() #调用对象die的roll()方法，模拟掷骰子过程
>>> die.getvalue()
5
>>> die.roll()
>>> die.getvalue()
2
>>> die.value  #获取对象die的变量成员value的取值
2
>>> die.setvalue(3) #通过调用对象die的方法成员setvalue()改变变量成员value的取值
>>> die.value
3
```

5.4-4 例子

- 例子说明
  - class <类名>:
    <方法定义>
  - 例子中类名为**msdie**，方法有4个：**__init__, roll, getvalue，setvalue**
  - 方法的定义与函数类似，第一个参量self是专用的，指向方法的作用对象
  - 话说**对象是数据与操作**的结合，定义中四个方法对应于操作，但数据呢？
  - 在构造函数__ 据成员！

```
>>> class msdie:
        '''Class definition for a multiple-side die'''
        def __init__(self,s):
            self.sides=s
            self.value=1
        def roll(self):
            self.value=randrange(1,self.sides+1)
        def getvalue(self):
            return self.value
        def setvalue(self,v):
            self.value=v
```

5.4-4 例子

```
>>> die=msdie(6) #创建一个msdie类对象
>>> die.getvalue() #调用对象die的getvalue()方法
1
```

- 实例(对象)
  - 程序中创建实例对象形如函数调用一般
  - 实际由构造函数__init__创建，参数通过__init__传递
  - 本例中,__init__构造函数有2个参数，其中self为专用参数，指向实例对象，在创建时不需要写出来
- 实例变量(数据属性调用)
  - 实例变量访问方式：

    <对象>.<变量名>
  - 例如

```
>>> die.value   #获取对象die的变量成员value的取值
2
```

```
>>> die.sides
6
>>>
```

5.4-4 例子

- 方法调用
  - 方式：
    <对象>.<函数名>(参数)
  - 注：同函数调用类似，但需指明作用对象，而参数self不用再写明
  - 例如

```
>>> die.roll() #调用对象die的roll()方法，模拟掷骰子过程
>>> die.getvalue()
5
>>> die.roll()
>>> die.getvalue()
2
```

5.4-5 类的继承与派生

- 一个类可通过继承的方式获得另一个类的属性和方法
  - 子类（派生类 DerivedClassName）会继承父类(基类 BaseClassName)的属性和方法

- 派生类的定义方式：

```
class DerivedClassName(BaseClassName):
    <statement_1>
    ...
    ...
    <statement_N>
```

  - 构造子类对象时，如果请求的属性和方法在子类中找不到，将转向父类中进行查找 。
  - 如果父类本身也派生自其他某个类，则此规则将被递归地应用。
  - 如果子类中有与父类同名的方法，则通过子类实例去调用方法时，会调用子类的方法。称为方法的重写(覆盖，override)

5.4-5 类的继承与派生

类继承举例：

```
#ch5: class_inherit.py
class people: #单继承：定义父类
    name = ''
    age = 0
    def __init__(self,n,a):#定义构造方法
        self.name = n
        self.age = a
    def speak(self):
        print("%s说：我%d岁。"%(self.name,self.age))
print('Initial age:',people.age)
class student(people): #定义子类
    grade = ''
    def __init__(self,n,a,g):
        people.__init__(self,n,a) #调用父类构造函数
        self.grade = g
    def speak(self): #重写(override)父类方法
        print("%s说：我%d岁在读%d年级"%(self.name,self.age,self.grade))
stu = student('Alice',10,3)
stu.speak()
```

```
Initial age: 0
Alice说：我10岁了，我在读3年级
```

5.4-5 类的继承与派生

- Python 也支持多重继承
  - 多继承的类定义方式：

```
class DerivedClassName(Base1, Base2, Base3):
    <statement_1>
    ...
    ...
    <statement_N>
```

  - 与单继承一样，构造子类对象时，如果请求的属性和方法在子类中找不到，将转向父类中进行查找。查找的原则是深度优先、从左至右。
  - 如果某一属性在 DerivedClassName 中未找到，则会到 Base1 中搜索它，然后（递归地）到 Base1 的基类中搜索，如果在那里未找到，再到 Base2 中搜索，依此类推。

5.4-5 类的继承与派生

```
29 #ch5: class_inherit.py
30 class people: # 多继承: 定义父类
31     name = ''
32     def __init__(self, n):
33         self.name = n
34     def speak(self):
35         print("我叫%s." % (self.name))
36 class student(people): #student类单继承people类
37     age = 0
38     def __init__(self, n, a):
39         people.__init__(self, n)
40     def speak(self):
41         print("我叫%s，我今年%d岁" % (self.name, self.age))
42 class speaker():#定义另一个父类
43     topic,name= '',''
44     def __init__(self, n, t):
45         self.name,self.topic = n,t
46     def speak(self):
47         print("我叫%s，我分享的主题是%s" % (self.name, self.topic))
48 class sample(speaker, student):#多重继承
49     def __init__(self, n, a, t):
50         student.__init__(self, n, a)
51         speaker.__init__(self, n, t)
52
53 test = sample("Alice", 25, "Python与数据科学")
54 test.speak()  # 默认调用的是speaker类的方法
55
```

```
我叫Alice，我分享的主题是Python与数据科学
```

5.4-5 类的继承与派生

- 补充说明
  - Python有两个内置函数可被用于继承机制，isinstance()和issubclass()
  - isinstance()检查一个实例的类型。如isinstance(obj, int) 仅会在 obj.__class__ 为 int 或某个派生自 int 的类时返回 True。
  - issubclass()检查类的继承关系。如issubclass(bool, int) 会返回 True，因为 bool 是 int 的子类。 但是，issubclass(float, int) 为 False，因为 float 不是 int 的子类。

第6章 Python数据科学库

提纲

- 6.1 科学计算Numpy和Scipy
- 6.2 数据分析Pandas
- 6.3 数据挖掘sklearn
- 6.4 绘图库Matplotlib

6.1 科学计算numpy和scipy

- Python中用list保存一组值，可以当做数组使用，不过list的元素可以是任何对象，保存的是对象的指针，用list数值计算显然浪费内存和CPU计算时间。

- Python还提供了一个array模块，array对象和列表不同，它直接保存数值，和C语言的一维数组类似，但它不支持多维，也没有各种运算函数，因此不适合数值计算。

- NumPy的诞生弥补了这些不足，NumPy提供了两种基本的对象：ndarray(N-dimensional array object)和 ufunc(universal function object)。
  - ndarray(下文统称为数组)是存储单一数据类型的多维数组
  - 而ufunc则是能够对数组进行处理的函数。

- Scipy是在NumPy库的基础上增加了众多的数学、科学以及工程计算中常用的库函数。

6.1-1 numpy的ndarray对象创建

- 数值计算库的导入

  import numpy as np

- ndarray实例
  - 通过传递Python的列表、元组等序列对象可以创建数组；如果传递的是多层嵌套的序列，将创建多维数组(例中的变量c)。

```
import numpy as np
```

```
a=np.array([1,2,3,4])  #从列表创建
b=np.array((5,6,7,8)) #从元组创建
c=np.array([[1,2,3,4],[4,5,6,7],[7,8,9,10]]) #二维数组
```

```
a
```

```
array([1, 2, 3, 4])
```

```
b
```

```
array([5, 6, 7, 8])
```

```
c
```

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

6.1-1 numpy的ndarray对象创建

ndarray实例的属性

- 数组的大小可以通过其shape属
shape属性，在保持数组元素个
的长度。

属性	说明
.ndim	秩，即轴的数量或维度的数量
.shape	ndarray对象的尺度，对于矩阵，n行m列
.size	ndarray对象元素的个数，相当于.shape中n*m的值
.dtype	ndarray对象的元素类型
.itemsize	ndarray对象中每个元素的大小，以字节为单位

```
import numpy as np

a=np.array([1,2,3,4])  #从列表创建
b=np.array((5,6,7,8)) #从元组创建
c=np.array([[1,2,3,4],[4,5,6,7],[7,8,9,10]]) #二维数组
```

```
print("ndim","a:", a.ndim,", b:",b.ndim,", c:",c.ndim)
print("shape","a:", a.shape,", b:",b.shape,", c:",c.shape)
print("size","a:", a.size,", b:",b.size,", c:",c.size)
print("dtype","a:", a.dtype,", b:",b.dtype,", c:",c.dtype)
print("itemsize","a:", a.itemsize,", b:",b.itemsize,", c:",c.itemsize)
```

```
ndim a: 1 , b: 1 , c: 2
shape a: (4,) , b: (4,) , c: (3, 4)
size a: 4 , b: 4 , c: 12
dtype a: int32 , b: int32 , c: int32
itemsize a: 4 , b: 4 , c: 4
```

6.1-1 numpy的ndarray对象创建

ndarray实例的方法

- tolist方法: 将数组变回list，但原数组不变
- reshape方法：创建一个新尺寸数组，原数组的shape保持不变。

```
import numpy as np
```

```
a=np.array([1,2,3,4])  #从列表创建
b=np.array((5,6,7,8)) #从元组创建
c=np.array([[1,2,3,4],[4,5,6,7],[7,8,9,10]]) #二维数组
```

```
a.tolist()
```

```
[1, 2, 3, 4]
```

```
a
```

```
array([1, 2, 3, 4])
```

```
c.tolist()
```

```
[[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]]
```

```
c
```

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

```
c.shape
```

```
(3, 4)
```

```
c.shape=4,3
```

```
c
```

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

```
d=a.reshape((2,2))
d
```

```
array([[1, 2],
       [3, 4]])
```

```
a
```

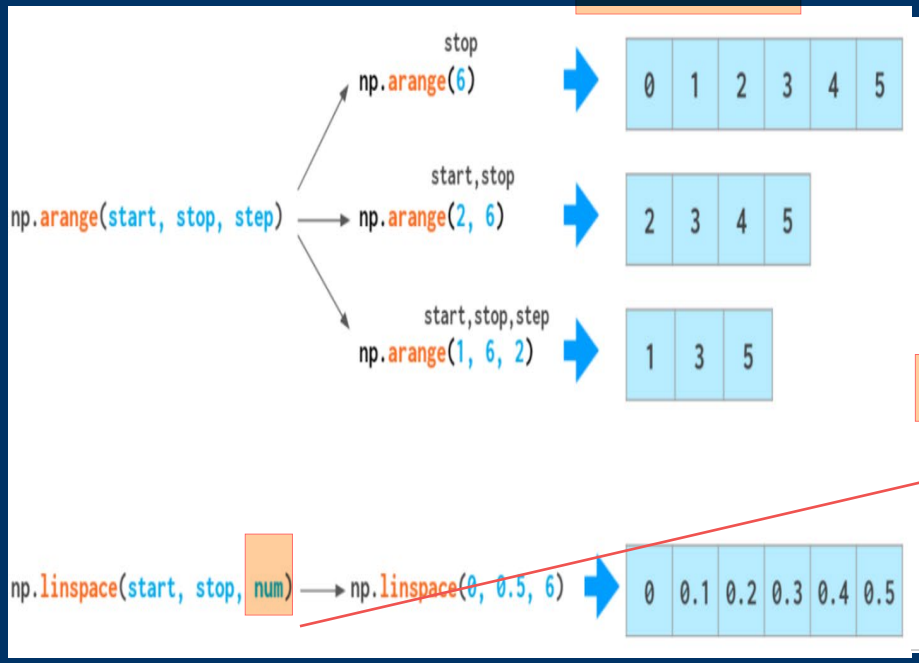
```
array([1, 2, 3, 4])
```

6.1-1 numpy的ndarray对象创建

创建数组

通过人工输入数据创建数组，显然效率不高。因此NumPy提供了很多专门用来创建数组的函数。

- **arange函数类似于python的range函数**，通过指定开始值、终值和步长来创建一维数组，注意数组不包括终值。
- **linspace**函数通过指定开始值、终值和元素个数来创建一维数组，可以通过endpoint关键字指定是否包括终值，缺省设置是包括终值。
- **logspace**函数和**linspace**类似，不过它创建等比数列。




```
np.arange(0,1,0.1)
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])

np.linspace(0,1,10)
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])

np.logspace(0,2,20)
array([  1.        ,   1.27427499,   1.62377674,   2.06913808,
         2.6366509 ,   3.35981829,   4.2813324 ,   5.45559478,
         6.95192796,   8.8586679 ,  11.28837892,  14.38449888,
        18.32980711,  23.35721469,  29.76351442,  37.92690191,
        48.32930239,  61.58482111,  78.47599704, 100.        ])
```

6.1-1 numpy的ndarray对象创建

创建特定数组

- **zeros()、ones()、identity()、empty()**可以创建指定形状和类型的数组；**diag()** 创建对角线为指定元素的二维数组

```
np.zeros((2,3)) #元素全为0的2*3矩阵
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
np.ones((2,3)) #元素全为1的2*3矩阵
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
np.identity(4) #4*4的单位矩阵
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
np.diag([1,2,3,4,5]) #5*5的对角矩阵
```

```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

```
import numpy as np
#从迭代器中生成
ite=(x**2 for x in range(5))
print("ite:",ite)
print("ite_array:\n",np.fromiter(ite,dtype=int))

#从函数中生成
def f(i, j):    return i-j
print("f:",f)
print("f_array:\n",np.fromfunction(f, (4, 4)))
```

```
ite: <generator object <genexpr> at 0x0000012B5BB9ECA8>
ite_array:
 [ 0  1  4  9 16]
f: <function f at 0x0000012B5BC21D90>
f_array:
 [[ 0. -1. -2. -3.]
 [ 1.  0. -1. -2.]
 [ 2.  1.  0. -1.]
 [ 3.  2.  1.  0.]]
```

6.1-1 numpy的ndarray对象创建

np.random模块(随机抽样)

某次运行结果：

- 1. 简单的随机数据生成函数

函数	功能说明
rand(d0, d1, ..., dn)	返回给定形状的0~1随机值样本
randn(d0, d1, ..., dn)	返回给定形状且服从标准正态分布的随机值样本
randint(low[, high, size])	返回随机的整数，位于半开区间 [low, high)
random_integers(low[, high, size])	返回随机的整数，位于闭区间 [low, high]
random_sample([size])	返回随机的浮点数，在半开区间 [0.0, 1.0)
random([size])	返回随机的浮点数，在半开区间 [0.0, 1.0)
ranf([size])	返回随机的浮点数，在半开区间 [0.0, 1.0)
sample([size])	返回随机的浮点数，在半开区间 [0.0, 1.0)
choice(a[, size, replace, p])	生成一个随机样本，从一个给定的一维数组
bytes(length)	返回随机字节

```python
#ch6_numpy_random.py
from numpy import random as nr
rand_x=nr.rand(2,3)
rand_x0=nr.rand()
randn_x=nr.randn(2,3)
randn_x0=nr.randn()

randint_x1=nr.randint(5,10, size=(2,4))
randint_x2=nr.randint(5, size=10)

cho_x1=nr.choice(5,4)#等价于randint(0,5,3),且权值相同
cho_x2=nr.choice(5,4,p=[0.1,0,0.3,0.6,0])#非均匀分布，服从分布p
cho_x3=nr.choice(5,3,replace=False) #无放回抽样
cho_x4=nr.choice(5,3,replace=False,p=[0.1,0,0.3,0.6,0])
words = ['pooh', 'rabbit', 'piglet', 'Christopher']
cho_x5=nr.choice(words,3, p=[0.5,0.1,0.1,0.3])

print("rand_x:\n",rand_x)
print("rand_x0:\n",rand_x0)
print("randn_x:\n",randn_x)
print("randn_x0:\n",randn_x0)
print("randint_x1:\n",randint_x1)
print("randint_x2:\n",randint_x2)
print("cho_x1:\n",cho_x1)
print("cho_x2:\n",cho_x2)
print("cho_x3:\n",cho_x3)
print("cho_x4:\n",cho_x4)
print("cho_x5:\n",cho_x5)
```

```
rand_x:
 [[0.65537735 0.04348519 0.65115586]
 [0.0936306  0.20192958 0.71997289]]
rand_x0:
 0.4771860435670533
randn_x:
 [[ 0.72495484  0.25773218  0.14526712]
 [ 1.47326439 -1.20794305  0.53319647]]
randn_x0:
 0.5983932599957633
randint_x1:
 [[7 5 6 5]
 [6 6 5 9]]
randint_x2:
 [0 1 2 1 4 2 3 4 4 4]
cho_x1:
 [0 0 4 3]
cho_x2:
 [2 3 2 3]
cho_x3:
 [4 3 2]
cho_x4:
 [2 3 0]
cho_x5:
 ['pooh' 'Christopher' 'pooh']
```

6.1-1 numpy的ndarray对象创建

np.random模块(随机抽样)

- 2. 排列

函数	功能说明

shuffle(x)	现场修改序列，改变自身内容。
	>>> arr = np.arange(10)
	>>> np.random.shuffle(arr)
	>>> arr
	[1 7 5 2 9 4 3 6 0 8]

permutation(x)	返回一个随机排列
	>>> np.random.permutation(10)
	array([1, 7, 4, 3, 0, 9, 2, 5, 8, 6])
	
	>>> np.random.permutation([1, 4, 9, 12, 15])
	array([15, 1, 9, 4, 12])

6.1-1 numpy的ndarray对象创建

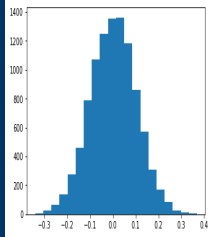
np.random模块(随机抽样)

- 3.概率分布

函数	功能说明
normal([loc, scale, size])	正态(高斯)分布
poisson([lam, size])	泊松分布
uniform([low, high, size])	均匀分布
binomial(n, p[, size])	二项分布
chisquare(df[, size])	卡方分布
exponential([scale, size])	指数分布
geometric(p[, size])	几何分布

```python
#ch6_numpy_random.py
import numpy as np
mu, sigma = 0, 0.1#mean and standard deviation
s = np.random.normal(mu, sigma, 10000)
print("#samples:",len(s))
print(abs(mu-np.mean(s))<0.01)
print(abs(sigma-np.std(s,ddof=1))<0.01)
#ddof=1表示无偏估计，除式分母使用N-ddof

import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(s, 20)
print(len(bins),bins)
print(len(count),count)
```

```
#samples: 10000
True
True
21 [-0.34037631 -0.30490846 -0.2694406  -0.23397275 -0.19850489 -0.16303704
 -0.12756918 -0.09210133 -0.05663348 -0.02116562  0.01430223  0.04977009
  0.08523794  0.1207058   0.15617365  0.1916415   0.22710936  0.26257721
  0.29804507  0.33351292  0.36898078]
20 [   8.   26.   63.  137.  275.  460.  788. 1071. 1251. 1354. 1362. 1180.
  858.  571.  306.  172.   81.   22.   11.    4.]
```



6.1-2 数组元素索引

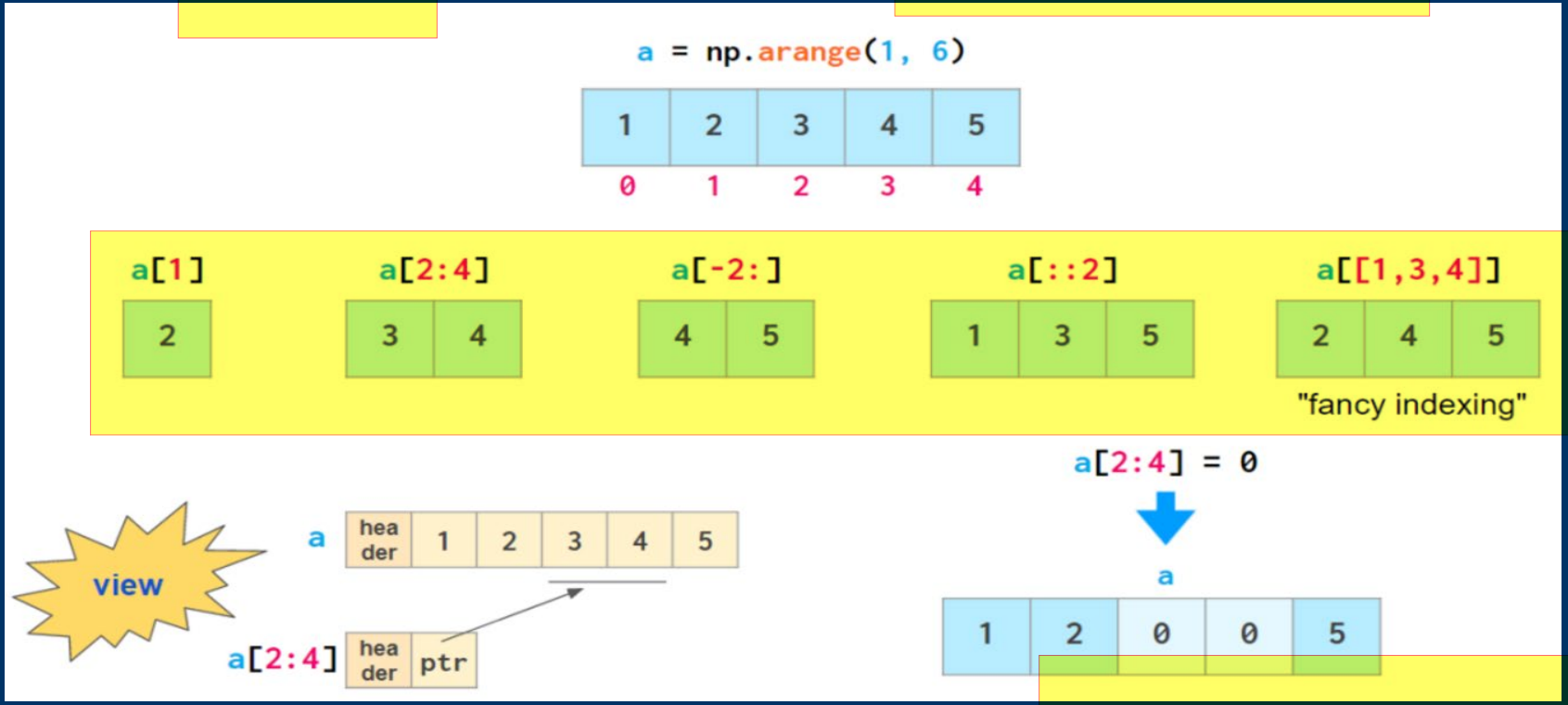
元素访问/索引方法

- 1.使用下标
  - ✓ 与python列表序列有所不同，通过下标范围获取的新数组是原始数组的一个视图，与原始数组共享数据空间。
- 2.使用整数序列
  - ✓ 使用整数序列中的每个元素作为下标，而整数序列可以是列表或者数组。获得的数组不与原始数组共享空间。
- 3.使用布尔数组
  - ✓ 当使用布尔数组b作为下标存取数组x中的元素时，将收集数组x中所有在数组b中对应下标为True的元素。
  - ✓ 使用布尔数组作为下标获得的数组不和原始数组共享数据空间，注意这种方式只对应于布尔数组，不能使用布尔列表。

6.1-2 数组元素索引

元素访问：一维数组（下标或整数序列）

- 下图中下标或切片方式的索引均为view，与原数组共享数据空间。



python list vs numpy array

```
a = [1, 2, 3]
b = a          # no copy
c = a[:]       # copy
d = a.copy()   # copy
```

```
a = np.array([1, 2, 3])
b = a          # no copy
c = a[:]       # no copy!!!
d = a.copy()   # copy
```

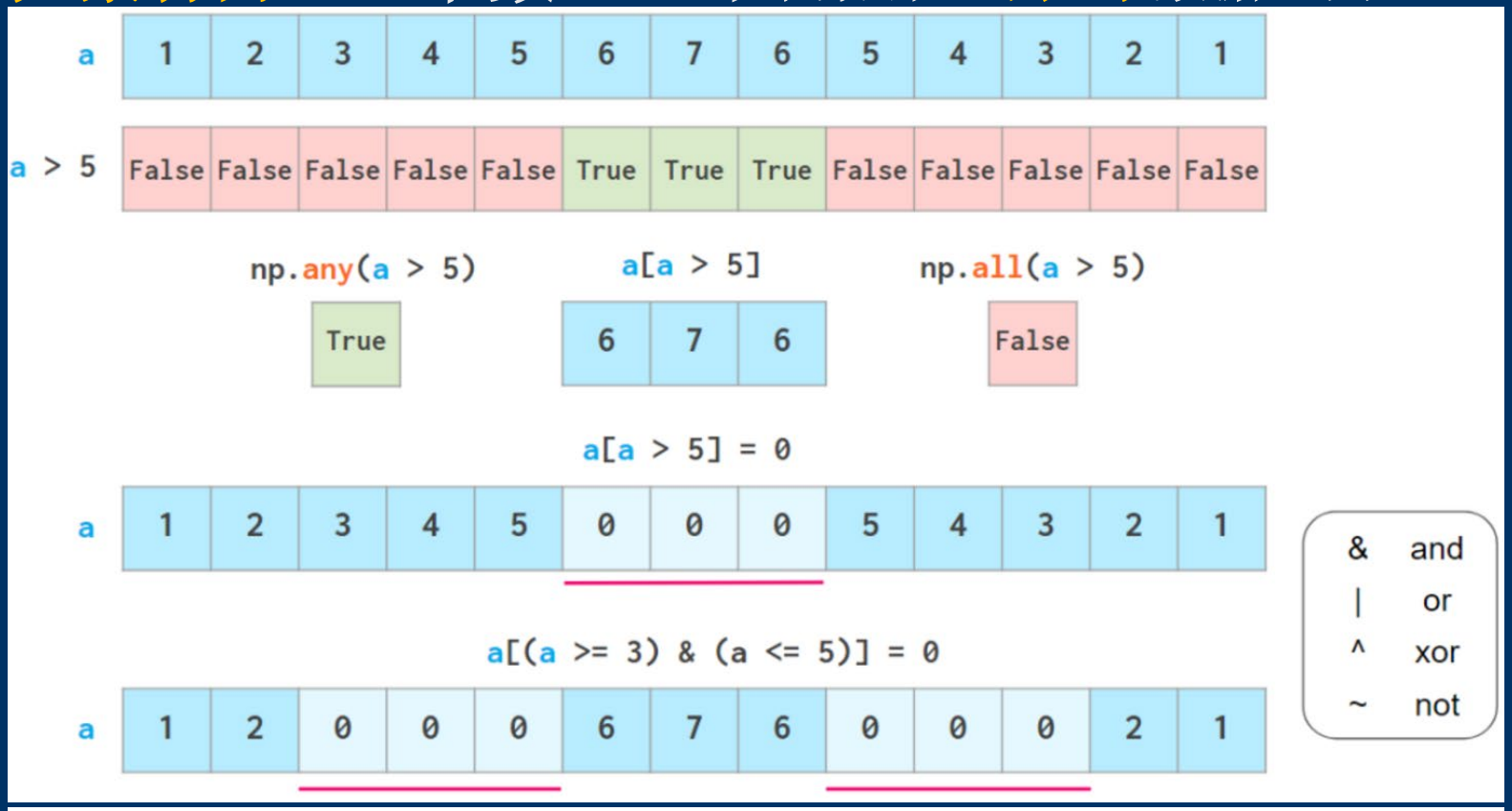
6.1-2 数组元素索引

元素访问：一维数组例子

```
In [3]: import numpy as np

In [11]: ls=[2,3,3]

In [12]: c=a[ls]

In [13]: c
Out[13]: array([8, 7, 7])

In [14]: c[1]=-30

In [15]: c
Out[15]: array([  8, -30,   7])

In [16]: a
Out[16]: array([ 10,   9,   8,   7, -20,   5,   4,   3,   2])

In [17]:
```

6.1-2 数组元素索引

元素访问：一维数组（布尔索引：不共享数据空间）




```
In [6]: a
In [10]: a[a>5]=100

In [11]: a
Out[11]: array([  0,   1,   2,   3,   4,   5, 100, 100, 100, 100])

In [12]: a[(a>=3)&(a<=5)]=-2

In [13]: a
Out[13]: array([  0,   1,   2,  -2,  -2,  -2, 100, 100, 100, 100])
```

6.1-2 数组元素索引

元素访问：一维数组（布尔索引）

- 也可以通过函数**np.where()**和**np.clip()**来改变数组元素的取值

a	1	2	3	4	5	6	7	6	5	4	3	2	1
	0	1	2	3	4	5	6	7	8	9	10	11	12

```
np.where(a > 5)
```

仅访问

5	6	7

```
= np.nonzero(a > 5)
```

```
[49]
np.where(random.rand(10)>0.5,100,-10)

array([-10, 100, -10, 100, 100, 100, -10, 100, -10, 100])
```

```
a[a < 5] = 0; a[a >= 5] = 1
```

a	0	0	0	0	1	1	1	1	1	0	0	0	0

访问加修改 `= np.where(a >= 5, 1, 0)`

```
a[a < 3] = 3; a[a > 5] = 5
```

a	3	3	3	4	5	5	5	5	5	4	3	3	3

```
= np.clip(a, 3, 5)
```

6.1-2 数组元素索引

元素访问：二维数组

- 与一维数组类似，但下标需多个维度来表示。 下图中a为一个6*6的二维数组，用颜色区分各个下标以及其对应的选择区域。

```
>>> a[0,3:5]
array([3,4])
>>> a[4:,4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,::2]
array([[20,22,24],
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

```
In [14]: a=np.array([range(10*i,10*i+6) for i in range(6)])

In [15]: a
Out[15]:
array([[ 0,  1,  2,  3,  4,  5],
       [10, 11, 12, 13, 14, 15],
       [20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35],
       [40, 41, 42, 43, 44, 45],
       [50, 51, 52, 53, 54, 55]])
```

6.1-2 数组元素索引

元素访问：二维数组

- 多维数组可以使用整数序列和布尔数组进行存取。

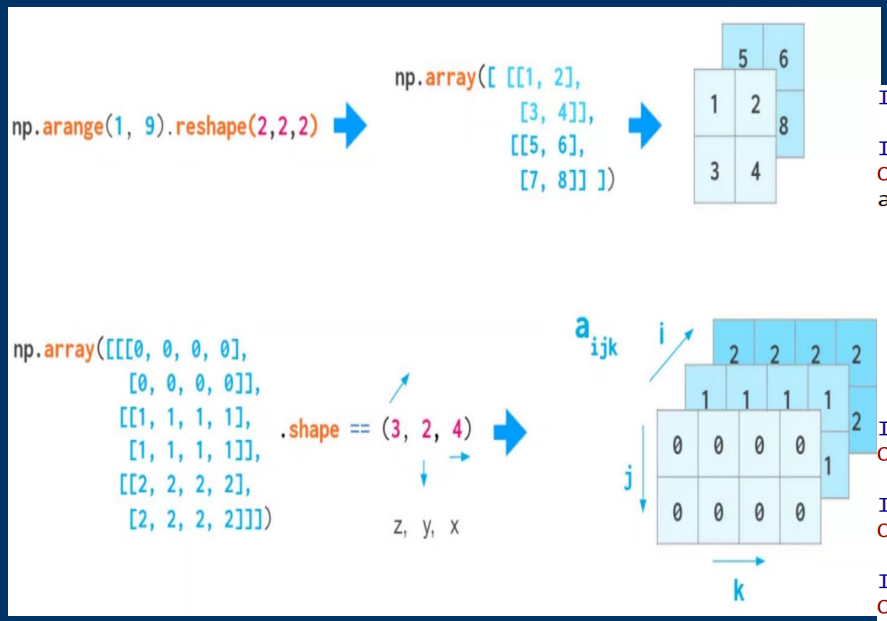
```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([1,12,23,34,45])
>>> a[3:,[0,2,5]]
array([[30,32,35],
       [40,42,45],
       [50,52,55]])
>>> mask=np.array([1,0,1,0,0,1],
                  dtype=np.bool)
>>> a[mask,2]
array([2,22,52])
```

- **a[(0,1,2,3,4),(1,2,3,4,5)]**：用于存取数组的下标仍然是一个有两个元素的组元，组元中的每个元素都是整数序列，分别对应数组的第0轴和第1轴。从两个序列的对应位置取出两个整数组成下标： a[0,1], a[1,2], ..., a[4,5]。
- **a[3:, [0, 2, 5]]**：下标中的第0轴是一个范围，它选取第3行之后的所有行；第1轴是整数序列，它选取第0, 2, 5三列。
- **a[mask, 2]**：下标的第0轴是一个布尔数组，它选取第0，2，5行；第1轴是一个整数，选取第2列。

6.1-2 数组元素索引

元素访问：高维数组

- 对于更高维的数组，只需记住，第一个索引是平面的编号，然后才是在该平面




```
In [21]: a=np.array([[range(10*i+j,10*i+3+j) for i in range(2)] for j in range(3)])

In [22]: a
Out[22]:
array([[[ 0,  1,  2],
        [10, 11, 12]],

       [[ 1,  2,  3],
        [11, 12, 13]],

       [[ 2,  3,  4],
        [12, 13, 14]]])

In [23]: a.shape
Out[23]: (3, 2, 3)

In [24]: a[1,1,2]
Out[24]: 13

In [25]: a[1,1,1:3]
Out[25]: array([12, 13])

In [26]: a[1:,:,:]
Out[26]:
array([[[ 1,  2,  3],
        [11, 12, 13]],

       [[ 2,  3,  4],
        [12, 13, 14]]])
```

6.1-2 数组元素索引

axis参数

- 在许多操作（例如求和np.sum()、最大值np.max()、最小值np.min()、平均值np.mean()、标准差np.std()等）中，我们需要告诉NumPy是否要跨行或跨列进行操作。
- 为使用任意维数的通用表示法，NumPy引入**axis**的概念：axis参数实际上指定索引序：第一个索引是axis=0，第二个索引是axis=1

```python
#ch6_axis_statfuncs.py
import numpy as np
a=np.array([range(10*i,10*i+3) for i in range(2)])
print("array a:\n",a)
print(np.sum(a),np.max(a),np.min(a),np.mean(a),np.std(a))
print(a.sum(),a.max(),a.min(),a.mean(),a.std())

print("sum0:",np.sum(a,axis=0))
print("sum1:",np.sum(a,axis=1))
print("mean0:",a.mean(axis=0))
print("mean1:",a.mean(axis=1))
```

```
array a:
 [[ 0  1  2]
 [10 11 12]]
36 12 0 6.0 5.066228051190222
36 12 0 6.0 5.066228051190222
sum0: [10 12 14]
sum1: [ 3 33]
mean0: [5. 6. 7.]
mean1: [ 1. 11.]
```

6.1-3 numpy的ufunc运算

ufunc运算简介

- ufunc是universal function的缩写，它是一种能对数组的每个元素进行操作的函数。
- NumPy内置的许多ufunc函数都是在C语言级别实现的，因此它们的计算速度非常快。

- 自定义ufunc函数

  通过组合标准的ufunc函数的调用，可以实现各种算式的数组计算。不过有些时候这种算式不易编写，而针对每个元素的计算函数却很容易用Python实现，这时可以用frompyfunc函数将一个计算单个元素的函数转换成ufunc函数。这样就可以方便地用所产生的ufunc函数对数组进行计算了。

6.1-3 numpy的ufunc运算

- NumPy内置的许多ufunc函数都是在C语言级别实现的，它们的计算速度非常快，例子：

```
import numpy as np
x=np.linspace(0,2*np.pi,10)
y=np.sin(x)
y
```

```
array([ 0.00000000e+00,  6.42787610e-01,  9.84807753e-01,  8.66025404e-01,
        3.42020143e-01, -3.42020143e-01, -8.66025404e-01, -9.84807753e-01,
       -6.42787610e-01, -2.44929360e-16])
```

- 以上例子说明
  - 先用linspace产生一个从0到2*PI的等距离的10个数，然后将其传递给sin函数。
  - 由于**np.sin是一个ufunc函数**，因此它对x中的每个元素求正弦值，然后将结果返回，并且赋值给y。
  - 计算之后x中的值并没有改变，而是新创建了一个数组保存结果。

6.1-3 numpy的ufunc运算

ufunc运算的优势在哪里？

```
math.sin: 0.3722636799999748
numpy.sin: 0.01312682666664955
numpy.sin loop: 1.311031039999989
```

```python
# ch6_math_sin_numpy_sin_comparison_ufunc.py
import time,math
import numpy as np

x = [i * 0.001 for i in range(1000000)]
start = time.clock()
for i, t in enumerate(x):
    x[i] = math.sin(t)
print("math.sin:", time.clock() - start)

x = [i * 0.001 for i in range(1000000)]
x = np.array(x)
start = time.clock()
np.sin(x,x) #ufunc运算
print("numpy.sin:", time.clock() - start)

x = [i * 0.001 for i in range(1000000)]
start = time.clock()
for i, t in enumerate(x):
    x[i] = np.sin(t) #迭代逐个计算
print("numpy.sin loop:", time.clock() - start)
```

在我的电脑上计算100万次正弦值，numpy.sin比math.sin快30倍左右。

然而，对单个数的计算math.sin则比numpy.sin快得多。请注意numpy.sin的计算速度只有math.sin的1/5。

这是因为，numpy.sin为了同时支持数组和单个值的计算，其C语言的内部实现要比math.sin复杂很多。此外，返回值类型有所不同，math.sin返回Python的标准float类型，而numpy.sin则返回一个numpy.float64类型。

6.1-3 numpy的ufunc运算

Numpy提供大量的**ufunc函数运算**

a^2	=	2	3	** 2	=	4	9
$\sqrt{a}$	=	np.sqrt(4	9)		=	2.	3.
e^a	=	np.exp(1	2)		=	2.718	7.389
$\ln a$	=	np.log(np.e	np.e**2)		=	1.	2.

np.floor(	1.1	1.5	1.9	2.5)	=	1.	1.	1.	2.
np.ceil(	1.1	1.5	1.9	2.5)	=	2.	2.	2.	3.
np.round(	1.1	1.5	1.9	2.5)	=	1.	2.	2.	2.

三角函数

np.sin(	np.pi	np.pi/2)	=	0.	1.
np.arcsin(	0.	1.)	=	0.	1.57

sin	arcsin	sinh	arcsinh
cos	arccos	cosh	arccosh
tan	arctan	tanh	arctanh

加、减、乘、除

1	2	+	4	8	=	5	10		
1	2	-	4	8	=	-3	-6		
4	8	*	2	5	=	8	40		
4	8	/	2	5	=	2.0	1.6	np.float64	
4	8	//	2	5	=	2	1	np.int32	
3	4	**	2	3	=	9	64		

6.1-3 numpy的ufunc运算

自定义ufunc函数

- **格式：frompyfunc(func,nin,nout)**

  其中，ufunc是计算单个元素的函数,nin是输入参数的个数，nout是func返回值的个数。例：

```
#ch6_frompyfunc.py
import numpy as np

def triangle_func(c,c0,hc):
    def trifunc(x):
        x=x-int(x)
        if x>=c: r=0.0
        elif x<c0: r=x/c0*hc
        else: r=(c-x)/(c-c0)*hc
        return r
    return np.frompyfunc(trifunc,1,1)

trifun=triangle_func(0.6,0.4,1.0)
a=np.linspace(1,10,5)
print(a)
print(trifun(a))
```

- 另外，也可以通过vectorize( )函数实现，请自行查阅相关资料。

6.1-4 向量与矩阵运算

- 线性代数中的向量与矩阵运算
  - NumPy中同时存在**ndarray**和**matrix**对象。
  - 利用**ndarray**对象可以实现向量与矩阵运算，不再推荐使用**matrix**对象。

- 向量表示为一维数组
  - 向量的点乘(内积)：

$\vec{a} \cdot \vec{b}$ = np.dot(1 2 , 3 4)

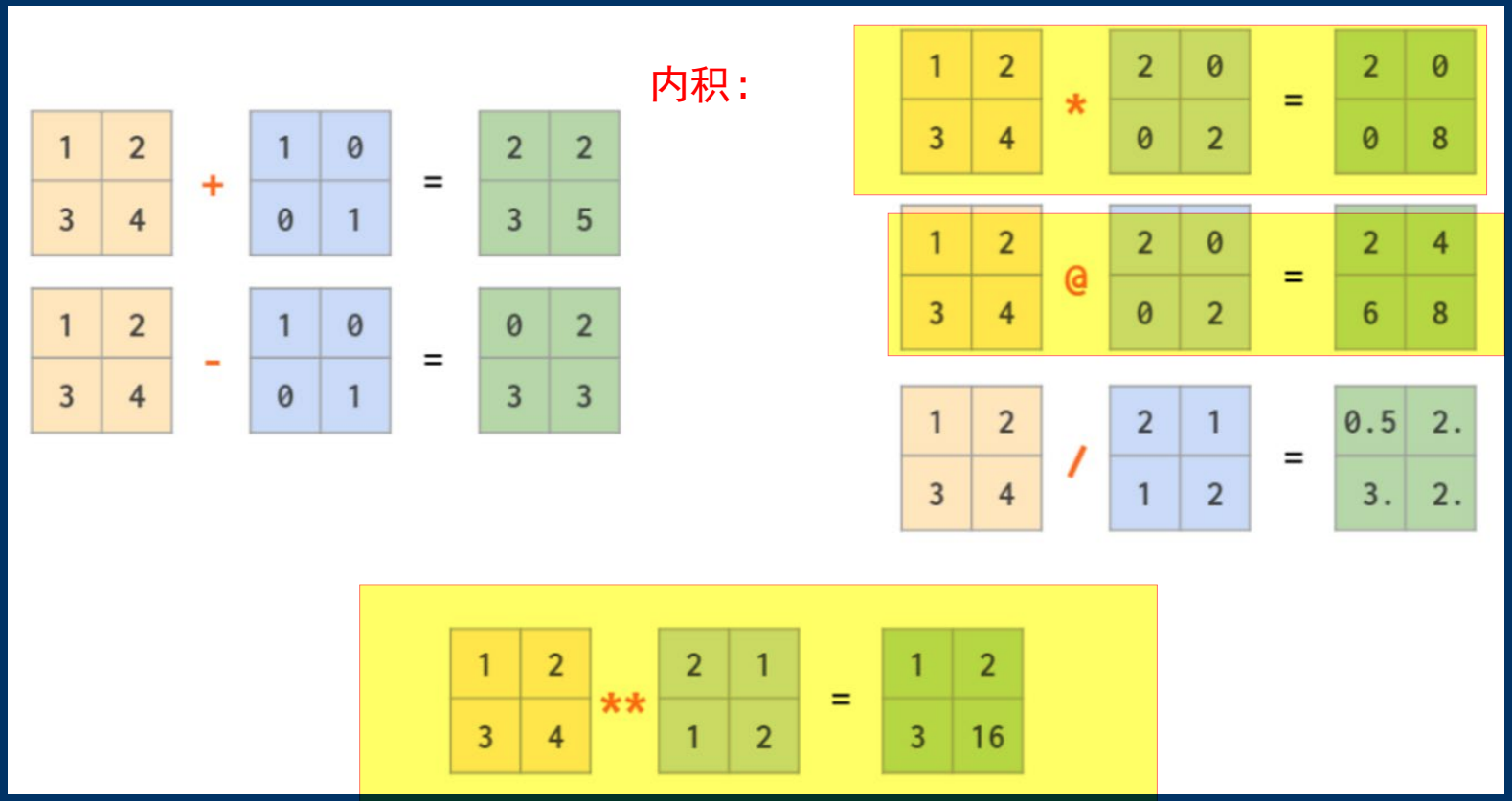
= 1 2 @ 3 4 = 11

  - 向量的叉乘（外积）：

$\vec{a} \times \vec{b}$ = np.cross(2 0 0 , 0 3 0) = 0 0 6

6.1-4 向量与矩阵运算

- 矩阵表示为二维数组
  - 基本矩阵运算（+，-，*，/，//和**，以及矩阵乘积@）



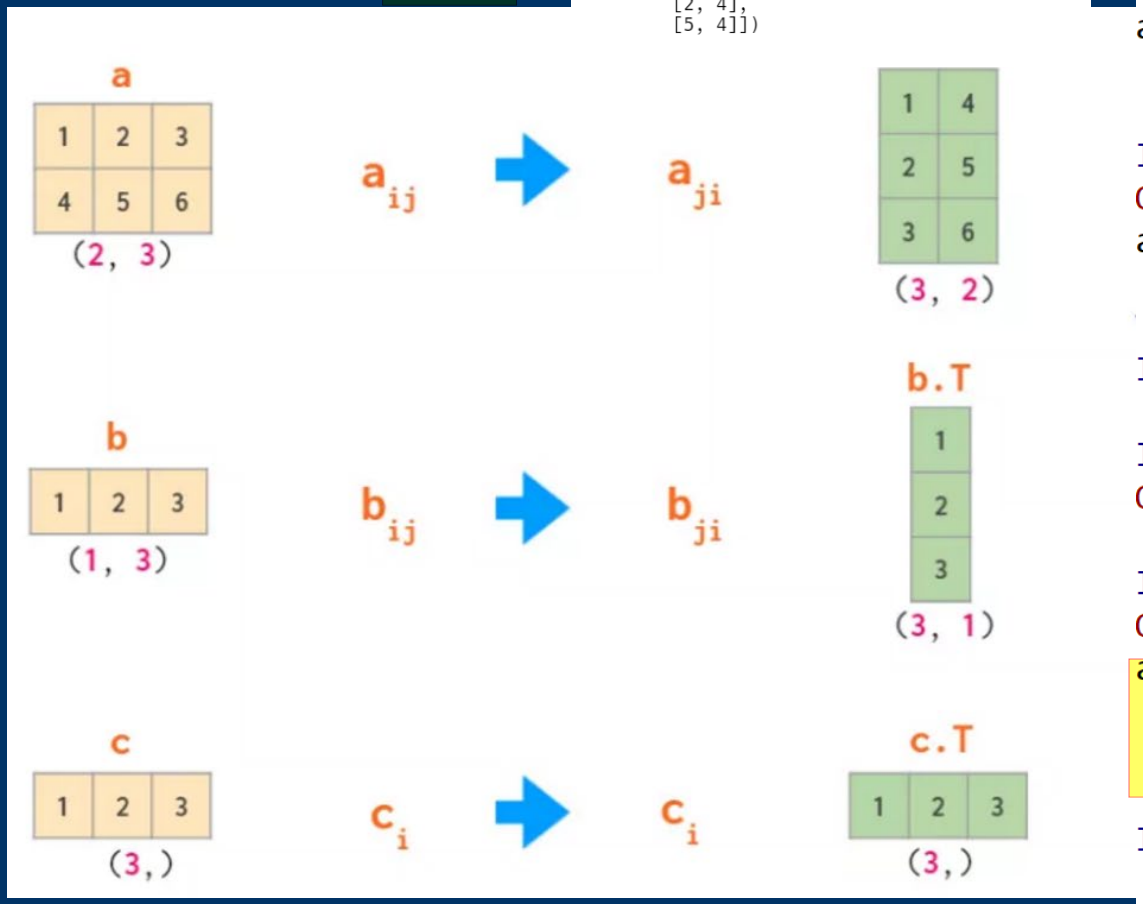

```
In [34]: a
Out[34]:
array([[1, 2],
       [1, 1]])

In [35]: b
Out[35]:
array([[-1, -2],
       [ 0,  3]])

In [36]: a@b
Out[36]:
array([[-1,  4],
       [-1,  1]])

In [37]: a*b
Out[37]:
array([[-1, -4],
       [ 0,  3]])

In [38]: a*b
Out[38]:
array([[-1, -4],
       [ 0,  3]])
```

6.1-4 向量与矩阵运算

- 矩阵表示为二维数组
  - 矩阵的转置(a.T)

```
c=np.array([1,2,5,6,4,4]).reshape(2,3)
c.T
```

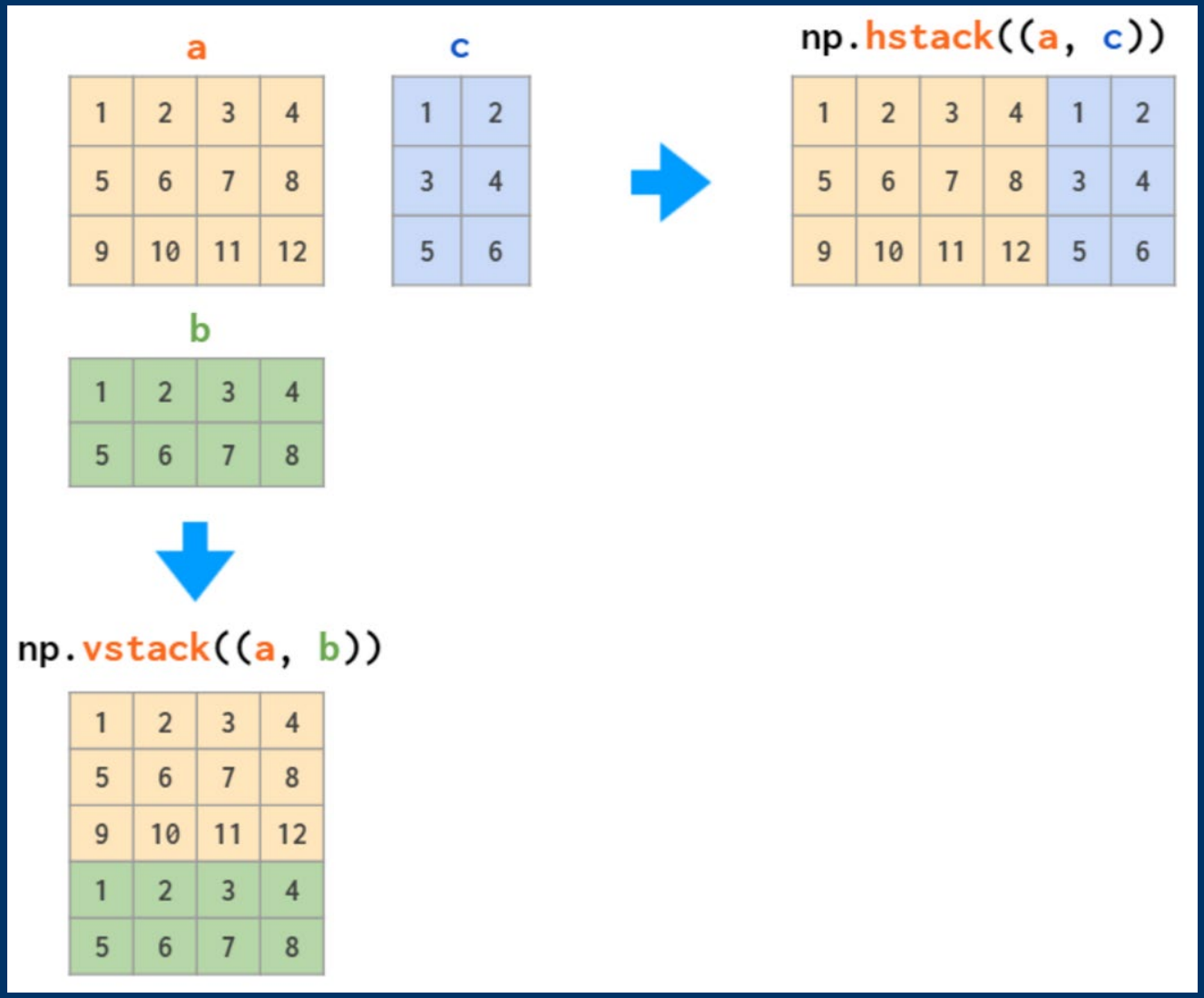
```
array([[1, 6],
       [2, 4],
       [5, 4]])
```




```
In [40]: a
Out[40]:
array([[1, 2],
       [1, 1]])

In [41]: a.T
Out[41]:
array([[1, 1],
       [2, 1]])

In [42]: b=np.array([[1,2,3]])

In [43]: b.shape
Out[43]: (1, 3)

In [44]: b.T
Out[44]:
array([[1],
       [2],
       [3]])

In [45]: c=np.array([1,2,3])

In [46]: c.shape
Out[46]: (3,)

In [47]: c.T
Out[47]: array([1, 2, 3])
```

6.1-4 向量与矩阵运算

- 矩阵表示为二维数组
  - 矩阵的连接操作：vstack()和hstack()



6.1-4 向量与矩阵运算

矩阵表示为二维数组

- **数据排序sort**
  - ✓ np.sort(a, axis=-1, kind='quicksort', order=None)
    - ➢ kind：排序算法，有 ‘quicksort’, ‘mergesort’, ‘heapsort’, ‘stable’

```python
#ch6_sort_argsort.py
import numpy as np
a = np.array([[1,4],[3,1]])
print("Original array a:\n",a)
a1=np.sort(a)  # 默认按最后一维排序
print("Sorted array a1:\n",a1)

a2=np.sort(a, axis=0)# 按第1维排序
print("Sorted array a2:\n",a2)

a3=np.sort(a, axis=None) #所有元素排序
print("Sorted array a3:\n",a3)

dtype=[('name', 'S10'), ('height', float), ('age', int)] #自定义数据类型
values=[('Arthur', 1.8, 41),  ('Lancelot', 1.9, 38), ('Galahad', 1.7, 38)]
b=np.array(values, dtype=dtype) #多级排序
a4=np.sort(b, order='height')
print("Sorted array a4:\n",a4)
```

```
Original array a:
 [[1 4]
 [3 1]]
Sorted array a1:
 [[1 4]
 [1 3]]
Sorted array a2:
 [[1 1]
 [3 4]]
Sorted array a3:
 [1 1 3 4]
Sorted array a4:
 [(b'Galahad', 1.7, 38) (b'Arthur', 1.8, 41) (b'Lancelot', 1.9, 38)]
```

6.1-4 向量与矩阵运算

- 矩阵表示为二维数组
  - 数据排序**argsort**
    - ✓ np.argsort(a, axis=-1, kind='quicksort', order=None)

```python
#ch6_sort_argsort.py
import numpy as np
x = np.array([3, 1, 2])#一维数组
print("index:",np.argsort(x))#argsort返回排序后的下标
   return the index of sorted elements before sorted
x = np.array([[2, 3,5], [8, 1,-3]])#二维数组
print("Original array:\n",x,'\n')
print("Sorted array0:\n",np.sort(x,axis=0))
print("index0:\n",np.argsort(x, axis=0))
print()
print("Sorted array1:\n",np.sort(x,axis=1))
print("index1:\n",np.argsort(x, axis=1))
```

```
index: [1 2 0]
Original array:
 [[ 2  3  5]
 [ 8  1 -3]]

Sorted array0:
 [[ 2  1 -3]
 [ 8  3  5]]
index0:
 [[0 1 1]
 [1 0 0]]

Sorted array1:
 [[ 2  3  5]
 [-3  1  8]]
index1:
 [[0 1 2]
 [2 1 0]]
```

6.1-5 文件数据存取

csv文件存取

- np.savetxt(fname, array, fmt='%.18e', delimiter=None)
- np.loadtxt(fname, dtype=np.float, delimiter=None, unpack=False)
- CSV只能有效存储一维和二维数组，因此np.savetxt() 和np.loadtxt() 只能有效存取一维和二维数组

任意维度文件存取

- a.tofile(fname, sep='', format='%s’)
- np.fromfile(fname, dtype=float, count=-1, sep=“)
- a.tofile()和np.fromfile()需要配合使用

```
#ch6_savetxt_loadtxt.py
import numpy as np
a=np.arange(100).reshape(5,20)
print("array a:\n",a)
np.savetxt('a.csv',a,fmt='%d',delimiter=',')
b=np.arange(30).reshape(3,10)
np.savetxt('b.csv',b,fmt='%.1f',delimiter=',')
c=np.loadtxt('b.csv',delimiter=',')
print("array c:\n",c)
d=np.arange(12).reshape(3,2,2)
print("array d:\n",d)
d.tofile('d.dat',sep=',',format='%d')
e=np.fromfile('d.dat',dtype=np.int,sep=',')
print("array e:\n",e)
```

界定符

```
array a:
 [[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]]
array c:
 [[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]
 [20. 21. 22. 23. 24. 25. 26. 27. 28. 29.]]
array d:
 [[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]]]
array e:
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
```

6.1-5 文件数据存取

便捷文件存取

```python
#ch6_saveload.py
import numpy as np
a=np.arange(30).reshape(2,3,5)
print('array a:\n',a)

np.save('a.npy',a)
b=np.load('a.npy')
print("array b:\n",b)

np.savez('c.npz',a) #存储压缩数据
c=np.load('c.npz')
print("c:\n",c)
print("array c:\n",c['arr_0'])

#压缩多个数组
u = np.array([[1,2,3],[4,5,6]])
v = np.arange(0, 1.0, 0.1)
w = np.sin(v)
np.savez("result.npz", u, v, sin_array = w)
r = np.load("result.npz")
print('array u:\n',r["arr_0"]) # 数组u
print('array v:\n',r["arr_1"]) # 数组v
print('array w:\n',r["sin_array"]) # 数组w
```

```
array a:
 [[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]
array b:
 [[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]
c:
 <numpy.lib.npyio.NpzFile object at
0x00000224A06123C8>
array c:
 [[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]
```

```
array u:
 [[1 2 3]
 [4 5 6]]
array v:
 [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
array w:
 [0.         0.09983342 0.19866933
0.29552021 0.38941834 0.47942554
 0.56464247 0.64421769 0.71735609
0.78332691]
```

6.1-6 SciPy

- **SciPy库**建立在 Numpy 库之上，提供了大量科学算法，主要包括以下主题：
  - 特殊函数 (scipy.**special**)
  - 积分 (scipy.**integrate**)
  - 最优化 (scipy.**optimize**)
  - 插值 (scipy.**interpolate**)
  - 傅立叶变换 (scipy.**fft**)
  - 信号处理 (scipy.**signal**)
  - 线性代数 (scipy.**linalg**)
  - 空间数据 (scipy.**spatial**)
  - 统计 (scipy.**stats**)
  - 多维图像处理 (scipy.**ndimage**)
  - 文件 IO (scipy.**io**)
- 官方文档： https://docs.scipy.org/doc/scipy/reference/

6.1-6 SciPy

数值积分scipy.integrate

- 如何计算半径为1的半圆的面积？其面积应等于PI/2

```python
#ch6_scipy_integrate_area.py
from scipy import integrate

#定义半圆曲线
def half_circle(x):
    return (1-x**2)**0.5
#利用quad函数进行数值积分
pi_half,err = integrate.quad(half_circle, -1, 1)
print("PI value:", pi_half*2)

```

$$\int_{-1}^{1}\int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \sqrt{1-y^2}\,dydx$$

- 官方文档：https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

6.1-6 SciPy

线性代数scipy.linalg

- 官方文档： https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html
- 求解线性方程组

$$\begin{aligned} x+3y+5z&=10\\ 2x+5y+z&=8\\ 2x+3y+8z&=3 \end{aligned}$$

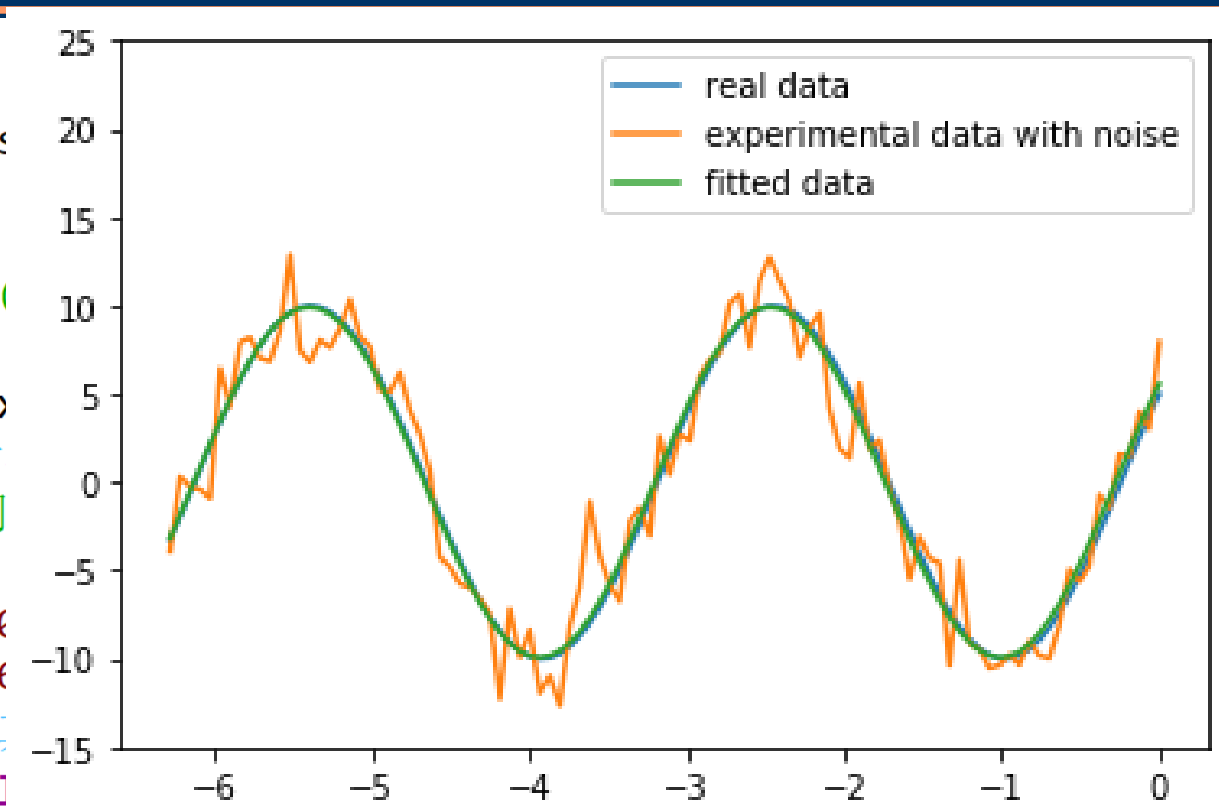
$$\begin{bmatrix} x\\ y\\ z \end{bmatrix}=\begin{bmatrix} 1 & 3 & 5\\ 2 & 5 & 1\\ 2 & 3 & 8 \end{bmatrix}^{-1}\begin{bmatrix} 10\\ 8\\ 3 \end{bmatrix}=\frac{1}{25}\begin{bmatrix} -232\\ 129\\ 19 \end{bmatrix}=\begin{bmatrix} -9.28\\ 5.16\\ 0.76 \end{bmatrix}$$

```python
#ch6_linearSystemOfEquations.py
import numpy as np
from scipy import linalg
A = np.array([[1,3,5],[2,5,1],[2,3,8]])
print("array A:\n",A)
b = np.array([[10],[8],[3]])
print("b:\n",b)

x1=linalg.inv(A).dot(b)  # solution
print("solution x1:\n",x1)
print('check1:\n',A.dot(linalg.inv(A).dot(b))-b)

x2=np.linalg.solve(A, b)  # fast
print("solution x2:\n",x2)
print("check2:",A.dot(np.linalg.solve(A, b))- b)

```

```
array A:
 [[1 3 5]
 [2 5 1]
 [2 3 8]]
b:
 [[10]
 [ 8]
 [ 3]]
solution x1:
 [[-9.28]
 [ 5.16]
 [ 0.76]]
check1:
 [[ 0.00000000e+00]
 [-1.77635684e-15]
 [-8.88178420e-16]]
solution x2:
 [[-9.28]
 [ 5.16]
 [ 0.76]]
check2: [[ 0.00000000e+00]
 [-1.77635684e-15]
 [-1.77635684e-15]]
```

6.1-6 Scipy

```
6 #ch6_scipy_least_square
7 import numpy as np
8 from scipy.optimize import leas
9 import pylab as pl
10 def func(x, p):
11     """数据拟合所用的函数: A*sin(
12     A, k, theta = p
13     return A*np.sin(2*np.pi*k*x
14 def residuals(p, y, x): #定义计
15     """实验数据x, y和拟合函数之间
16     return y - func(x, p)
17 x = np.linspace(0, -2*np.pi, 16
18 A, k, theta = 10, 0.34, np.pi/6
19 y0 = func(x, [A, k, theta]) #
20 y1 = y0 + 2 * np.random.randn(1
21 p0 = [7, 0.2, 0] # p0为拟合参数的初始值
22 # 调用leastsq进行数据拟合，args为需要拟合的实验数据
23 plsq = leastsq(residuals, p0,
24 print("真实参数:", [A, k, theta
25 print("拟合参数", plsq[0]) # 实验
26 pl.plot(x, y0, label="real data")
27 pl.plot(x, y1, label="experimental data with noise")
28 pl.plot(x, func(x, plsq[0]), label="fitted data")
29 pl.ylim(-15,25)
30 pl.legend()
31 pl.show()
```



某次运行结果:

```
真实参数: [10, 0.34, 0.5235987755982988]
拟合参数: [ 9.92134375  0.34162634 -5.68594434]
```

6.2 数据分析pandas包

- pandas基于NumPy构建，使数据分析工作变得更快更简单。
- pandas两个核心数据结构：
  - Series：对应于一维的序列
  - DataFrame：对应于二维的表结构
- pandas 约定俗成的导入方法如下：

```
from pandas import Series, DataFrame
import pandas as pd
```

- 官网

Python Data Analysis Library ¶

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NUMFocus sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project.

A Fiscally Sponsored Project of
NUMFOCUS
OPEN CODE = BETTER SCIENCE

6.2-1 Series

- Series简介
  - Series是一种类似于一维数组的对象，它由一组数据(各种numpy数据类型)以及一组与之相关的数据标签(即索引)组成。
  - Series可以看做一个定长的有序字典
- Series 对象：

```
>>> s = Series([1,2,3.0,'abc'])
>>> s
0      1
1      2
2      3
3    abc
dtype: object
```

  - 注意到，Series 对象包含两个主要的属性：index 和 values，分别为上例中左右两列。

6.2-1 Series

Series 对象（续）：

- 传给构造器的是一个列表，所以 index 的值是从 0 起递增的整数，如果传入的是一个**类字典的键值对结构**，就会生成 index-value 对应的 Series；或者在初始化的时候以关键字参数显式指定一个 index 对象：

```
>>> s = Series(data=[1,3,5,7],index = ['a','b','x','y'])
>>> s
a    1
b    3
x    5
y    7
dtype: int64
>>> s.index
Index(['a', 'b', 'x', 'y'], dtype='object')
>>> s.values
array([1, 3, 5, 7], dtype=int64)
```

- 这

6.2-2 DataFrame

DataFrame简介

- DataFrame是一个表格型的数据结构，由一组有序的列构成，每列可以是不同的值类 型，不像 ndarray 只能有一个 dtype。
- DataFrame既有行索引也有列索引，基本上可以把 DataFrame 看成是共享同一个 index 的 Series 的集合。

DataFrame 的构造方法：

- 与 Series 类似，只不过可以同时接受多条一维数据源，每一条都会成为单独的一列

```
>>> data = {'state':['Ohino','Ohino','Ohino','Nevada','Nevada'],
        'year':[2000,2001,2002,2001,2002],
        'pop':[1.5,1.7,3.6,2.4,2.9]}
>>> df = DataFrame(data)
>>> df
   pop   state  year
0  1.5   Ohino  2000
1  1.7   Ohino  2001
2  3.6   Ohino  2002
3  2.4  Nevada  2001
4  2.9  Nevada  2002

[5 rows x 3 columns]
```

6.2-2 DataFrame

DataFrame 的构造方法（续）：

```
>>> df = DataFrame(data,index=['one','two','three','four','five'],
                   columns=['year','state','pop','debt'])
>>> df
       year   state  pop debt
one    2000   Ohino  1.5  NaN
two    2001   Ohino  1.7  NaN
three  2002   Ohino  3.6  NaN
four   2001  Nevada  2.4  NaN
five   2002  Nevada  2.9  NaN

[5 rows x 4 columns]
```

修改行标签和列表签（列表签的修改包括增减、调序）

```
>>> data = {'state':['Ohino','Ohino','Ohino','Nevada','Nevada'],
            'year':[2000,2001,2002,2001,2002],
            'pop':[1.5,1.7,3.6,2.4,2.9]}
>>> df = DataFrame(data)
>>> df
   pop   state  year
0  1.5   Ohino  2000
1  1.7   Ohino  2001
2  3.6   Ohino  2002
3  2.4  Nevada  2001
4  2.9  Nevada  2002

[5 rows x 3 columns]
```

datase
0

- 缺失值由 NaN 补上。看一下 index、columns 和 索引的类型：

```
>>> df.index
Index(['one', 'two', 'three', 'four', 'five'], dtype='object')
>>> df.columns
Index(['year', 'state', 'pop', 'debt'], dtype='object')
>>> type(df['debt'])
<class 'pandas.core.series.Series'>
```

6.2-3 对象属性

- 索引和切片
  - 就像 Numpy，pandas 也支持通过 **obj[::]** 的方式进行索引和切片，以及通过布尔型数组进行过滤。
  - 不过须要注意，因为 pandas 对象的 index 不限于整数，所以当使用**非整数作为切片索引**时，它是**末端包含**的

```
>>> foo
a     4.5
b     7.2
c    -5.3
d     3.6
dtype: float64
>>> bar
0     4.5
1     7.2
2    -5.3
3     3.6
dtype: float64
```

```
>>> foo[:2]
a     4.5
b     7.2
dtype: float64
>>> bar[:2]
0     4.5
1     7.2
dtype: float64
>>> foo[:'c']
a     4.5
b     7.2
c    -5.3
dtype: float64
```

6.2-3 对象属性

- 索引和切片（续）
  - DataFrame 对象的索引方式较特别，因为有两个轴向（双重索引）
  - DataFrame 对象的标准切片语法为：.ix[::,::]。ix 对象可以接受两套切片，分别为行（axis=0）和列（axis=1）的方向。

```
>>> df['Ohio']
a    0
c    3
d    6
Name: Ohio, dtype: int32
>>> df[:'c']
   Ohio  Texas  California
a     0      1           2
c     3      4           5

[2 rows x 3 columns]
>>> df[:2]
   Ohio  Texas  California
a     0      1           2
c     3      4           5

[2 rows x 3 columns]
```

```
>>> df
   Ohio  Texas  California
a     0      1           2
c     3      4           5
d     6      7           8

[3 rows x 3 columns]
>>> df.ix[:2,:2]
   Ohio  Texas
a     0      1
c     3      4

[2 rows x 2 columns]
>>> df.ix['a','Ohio']
0
```

6.2-3 对象属性

- 索引和切片（续）
  - 使用布尔型数组的情况，注意行与列的不同切法（列切法的：不能省）：

```
>>> df['Texas']>=4
a    False
c     True
d     True
Name: Texas, dtype: bool
>>> df[df['Texas']>=4]
   Ohio  Texas  California
c     3      4           5
d     6      7           8

[2 rows x 3 columns]
>>> df.ix[:,df.ix['c']>=4]
   Texas  California
a      1           2
c      4           5
d      7           8

[3 rows x 2 columns]
```

6.2-3 对象属性

统计方法

- pandas对象拥有一组常用的数学和统计方法。它们大部分都属于约简和汇总统计，用于从Series中提取的个值（如sum或mean)或从DataFrame的行或列中提取一个Series。
- 描述和汇总统计

方法	说明
count	非NA值的数量
describe	针对Series或各DataFrame列计算汇总统计
min,max	计算最小值和最大值
argmin,argmax	计算能够获取到最小值和最大值的索引位置
idxmin,idxmax	计算能够获取到最小值和最大值的索引值
quantile	计算样本的分位数（0到 1）
sum	值的总和

6.2-3 对象属性

- 统计方法
  - 描述和汇总统计（续）

mean	值的平均数
median	值的算术中位数（50%分位数)
mad	根据平均值计算平均绝对离差
var	样本值的方差
std	样本值的标准差
skew	样本值的偏度（三阶矩）
kurt	样本值的峰度（四阶矩）
cumsum	样本值的累计和
cummin,cummax	样本值的累计最大值和累计最小
cumprod	样本值的累计积
diff	计算一阶差分（对时间序列很有用)
pct_change	计算百分数变化

6.2-3 对象属性

- 统计方法
  - 例子1：

```
>>> df
    one  two
a  1.40  NaN
b  7.10 -4.5
c   NaN  NaN
d  0.75 -1.3

[4 rows x 2 columns]
```

```
>>> df.mean()
one    3.083333
two   -2.900000
dtype: float64
>>> df.mean(axis=1)
a    1.400
b    1.300
c      NaN
d   -0.275
dtype: float64
>>> df.mean(axis=1,skipna=False)
a      NaN
b    1.300
c      NaN
d   -0.275
dtype: float64
```

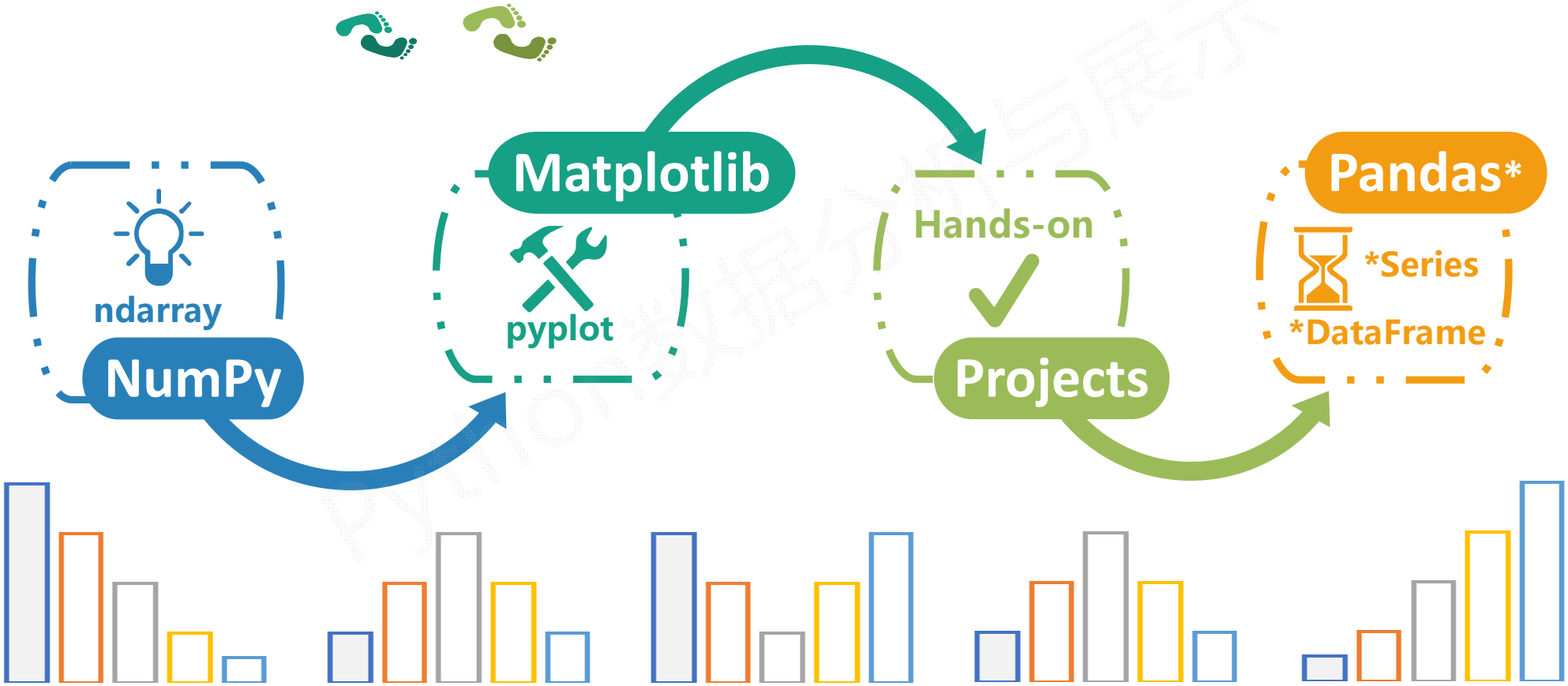
Pandas 补充材料

DV07

python
弹指之间 · 享受创新

Python数据分析与展示

掌握表示、清洗、统计和展示数据的能力



O4X -Tian

Pandas库的介绍

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

http://pandas.pydata.org

overview // get pandas // documentation // community // talks

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NUMFocus sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project.

A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

0.19.2 Final (December 24, 2016)

This is a minor bug-fix release in the 0.19.x series and includes some small regression fixes, bug fixes and performance improvements.

Highlights include:

- Compatibility with Python 3.6
- Added a Pandas Cheat Sheet.

See the v0.19.2 Whatsnew page for an overview of all bugs that have been fixed in 0.19.2.

VERSIONS

Release
0.19.2 - December 2016
download // docs // pdf

Development
0.20.0 - 2017
github // docs

Previous Releases
0.19.1 - download // docs // pdf
0.19.0 - download // docs // pdf
0.18.1 - download // docs // pdf
0.18.0 - download // docs // pdf
0.17.1 - download // docs // pdf
0.17.0 - download // docs // pdf
0.16.2 - download // docs // pdf
0.16.1 - download // docs // pdf
0.16.0 - download // docs // pdf
0.15.2 - download // docs // pdf
0.15.1 - download // docs // pdf
0.15.0 - download // docs // pdf
0.14.1 - download // docs // pdf
0.14.0 - download // docs // pdf
0.13.1 - download // docs // pdf
0.13.0 - download // docs // pdf
0.12.0 - download // docs // pdf

Pandas库的引用

Pandas是Python第三方库，提供高性能易用数据类型和分析工具

```
import pandas as pd
```

Pandas基于NumPy实现，常与NumPy和Matplotlib一同使用

Pandas库小测

```
In [66]: import pandas as pd

In [67]: d = pd.Series(range(20))

In [68]: d
Out[68]:
0      0
1      1
2      2
3      3
4      4
5      5
6      6
7      7
8      8
9      9
10    10
11    11
12    12
13    13
14    14
15    15
16    16
17    17
18    18
19    19
dtype: int32
```

```
In [65]: d.cumsum()
Out[65]:
0       0
1       1
2       3
3       6
4      10
5      15
6      21
7      28
8      36
9      45
10     55
11     66
12     78
13     91
14    105
15    120
16    136
17    153
18    171
19    190
dtype: int32
```

计算前N项累加和

Python数据分析与展示

Pandas库的理解

两个数据类型：Series, DataFrame

基于上述数据类型的各类操作

基本操作、运算操作、特征类操作、关联类操作

Pandas库的理解

NumPy	Pandas
基础数据类型	扩展数据类型
关注数据的结构表达	关注数据的应用表达
维度：数据间关系	数据与索引间关系

Pandas库的Series类型

Series类型

Series类型由一组数据及与之相关的数据索引组成

index_0 → data_a

index_1 → data_b

index_2 → data_c

index_3 → data_d

索引　　数据

Series类型

Series类型由一组数据及与之相关的数据索引组成

```
In [72]: import pandas as pd

In [73]: a = pd.Series([9, 8, 7, 6])

In [74]: a
Out[74]:
0    9
1    8
2    7
3    6
dtype: int64
```

自动索引

NumPy中数据类型

Series类型

Series类型由一组数据及与之相关的数据索引组成

```
In [76]: import pandas as pd

In [77]: b = pd.Series([9, 8, 7, 6], index=['a','b','c','d'])

In [78]: b
Out[78]:
a    9
b    8
c    7
d    6
dtype: int64
```

自定义索引

作为第二个参数，可以省略index=

Series类型

Series类型可以由如下类型创建：

- Python列表
- 标量值
- Python字典
- ndarray
- 其他函数

Python数据分析与展示

Series类型

从标量值创建

不能省略index

```
In [85]: import pandas as pd

In [86]: s = pd.Series(25, index=['a','b','c'])

In [87]: s
Out[87]:
a    25
b    25
c    25
dtype: int64
```

Series类型

从字典类型创建

index从字典中进行选择操作

```
In [88]: import pandas as pd

In [89]: d = pd.Series({'a':9, 'b':8, 'c':7})

In [90]: d
Out[90]:
a    9
b    8
c    7
dtype: int64
```

```
In [91]: e = pd.Series({'a':9, 'b':8, 'c':7}, index=['c', 'a', 'b', 'd'])

In [92]: e
Out[92]:
c    7.0
a    9.0
b    8.0
d    NaN
dtype: float64
```

Series类型

从ndarray类型创建

```
In [98]: import pandas as pd

In [99]: import numpy as np

In [100]: n = pd.Series(np.arange(5))

In [101]: n
Out[101]:
0    0
1    1
2    2
3    3
4    4
dtype: int32
```

```
In [102]: m = pd.Series(np.arange(5), index=np.arange(9,4,-1))

In [103]: m
Out[103]:
9    0
8    1
7    2
6    3
5    4
dtype: int32
```

Series类型

Series类型可以由如下类型创建：

- Python列表，index与列表元素个数一致
- 标量值，index表达Series类型的尺寸
- Python字典，键值对中的“键”是索引，index从字典中进行选择操作
- ndarray，索引和数据都可以通过ndarray类型创建
- 其他函数，range()函数等

Series类型的基本操作

Series类型包括index和values两部分

Series类型的操作类似ndarray类型

Series类型的操作类似Python字典类型

Series类型的基本操作

```
In [104]: import pandas as pd

In [105]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [106]: b
Out[106]:
a    9
b    8
c    7
d    6
dtype: int64

In [107]: b.index
Out[107]: Index(['a', 'b', 'c', 'd'], dtype='object')

In [108]: b.values
Out[108]: array([9, 8, 7, 6], dtype=int64)
```

.index	获得索引
.values	获得数据

Python教程不白费

Series类型的基本操作

```
In [109]: b['b']
Out[109]: 8

In [110]: b[1]
Out[110]: 8

In [111]: b[['c', 'd', 0]]
Out[111]:
c    7.0
d    6.0
0    NaN
dtype: float64

In [112]: b[['c', 'd', 'a']]
Out[112]:
c    7
d    6
a    9
dtype: int64
```

自动索引和自定义索引并存

两套索引并存，但不能混用

Series类型的基本操作

Series类型的操作类似ndarray类型：

- 索引方法相同，采用[]
- NumPy中运算和操作可用于Series类型
- 可以通过自定义索引的列表进行切片
- 可以通过自动索引进行切片，如果存在自定义索引，则一同被切片

Series类型的基本操作

```
In [113]: import pandas as pd

In [114]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [115]: b
Out[115]:
a    9
b    8
c    7
d    6
dtype: int64

In [116]: b[3]
Out[116]: 6

In [117]: b[:3]
Out[117]:
a    9
b    8
c    7
dtype: int64
```

```
In [118]: b[b > b.median()]
Out[118]:
a    9
b    8
dtype: int64

In [119]: np.exp(b)
Out[119]:
a    8103.083928
b    2980.957987
c    1096.633158
d     403.428793
dtype: float64
```

Python数据分析与展示

Series类型的基本操作

Series类型的操作类似Python字典类型：

- 通过自定义索引访问
- 保留字in操作
- 使用.get()方法

Python数据分析与展示

Series类型的基本操作

```
In [120]: import pandas as pd

In [121]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [122]: b['b']
Out[122]: 8

In [123]: 'c' in b
Out[123]: True

In [124]: 0 in b
Out[124]: False

In [125]: b.get('f', 100)
Out[125]: 100
```

Series类型对齐操作

Series + Series

```
In [134]: import pandas as pd

In [135]: a = pd.Series([1, 2, 3], ['c', 'd', 'e'])

In [136]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [137]: a + b
Out[137]:
a    NaN
b    NaN
c    8.0
d    8.0
e    NaN
dtype: float64
```

Series类型在运算中会自动对齐不同索引的数据

Series类型的name属性

Series对象和索引都可以有一个名字，存储在属性.name中

```
In [149]: import pandas as pd

In [150]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [151]: b.name

In [152]: b.name = 'Series对象'

In [153]: b.index.name = '索引列'

In [154]: b
Out[154]:
索引列
a    9
b    8
c    7
d    6
Name: Series对象, dtype: int64
```

Series类型的修改

Series对象可以随时修改并即刻生效

```
In [157]: import pandas as pd

In [158]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [159]: b['a'] = 15

In [160]: b.name = "Series"

In [161]: b
Out[161]:
a    15
b     8
c     7
d     6
Name: Series, dtype: int64
```

```
In [162]: b.name = "New Series"

In [163]: b['b', 'c'] = 20

In [164]: b
Out[164]:
a    15
b    20
c    20
d     6
Name: New Series, dtype: int64
```

Python数据分析与挖掘

Series类型

Series是一维带“标签”数组

index_0 → data_a

Series基本操作类似ndarray和字典，根据索引对齐

Pandas库的DataFrame类型

DataFrame类型

DataFrame类型由共用相同索引的一组列组成

index_0 →	data_a	data_1		data_w
index_1 →	data_b	data_2		data_x
index_2 →	data_c	data_3	……	data_y
index_3 →	data_d	data_4		data_z

索引　　多列数据

DataFrame类型

column　　axis=1

index

index_0	data_a	data_1		data_w
index_1	data_b	data_2		data_x
index_2	data_c	data_3	……	data_y
index_3	data_d	data_4		data_z

axis=0

DataFrame是一个表格型的数据类型，每列值类型可以不同

DataFrame既有行索引、也有列索引

DataFrame常用于表达二维数据，但可以表达多维数据

DataFrame类型

DataFrame类型可以由如下类型创建：

- 二维ndarray对象
- 由一维ndarray、列表、字典、元组或Series构成的字典
- Series类型
- 其他的DataFrame类型

DataFrame类型

从二维ndarray对象创建

```
In [165]: import pandas as pd

In [166]: import numpy as np

In [167]: d = pd.DataFrame(np.arange(10).reshape(2,5))

In [168]: d
Out[168]:
   0  1  2  3  4
0  0  1  2  3  4
1  5  6  7  8  9
```

自动行索引

自动列索引

DataFrame类型

从一维ndarray对象字典创建

```
In [171]: import pandas as pd

In [172]: dt = {'one': pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     ...:       'two': pd.Series([9, 8, 7, 6], index = ['a', 'b', 'c', 'd'])}

In [173]: d = pd.DataFrame(dt)

In [174]: d
Out[174]:
   one  two
a  1.0    9
b  2.0    8
c  3.0    7
d  NaN    6

In [175]: pd.DataFrame(dt, index=['b', 'c', 'd'], columns=['two', 'three'])
Out[175]:
   two three
b    8   NaN
c    7   NaN
d    6   NaN
```

自定义行索引

自定义列索引

数据根据行列索引自动补齐

DataFrame类型

从列表类型的字典创建

```
In [176]: import pandas as pd

In [177]: dl = {'one': [1, 2, 3, 4], 'two': [9, 8, 7, 6]}

In [178]: d = pd.DataFrame(dl, index = ['a', 'b', 'c', 'd'])

In [179]: d
Out[179]:
   one  two
a    1    9
b    2    8
c    3    7
d    4    6
```

2016年7月部分大中城市新建住宅价格指数

城市	环比	同比	定基
北京	101.5	120.7	121.4
上海	101.2	127.3	127.8
广州	101.3	119.4	120.0
深圳	102.0	140.9	145.5
沈阳	100.1	101.4	101.6

```
In [194]: import pandas as pd

In [195]: dl = {'城市': ['北京', '上海', '广州', '深圳', '沈阳'],
     ...:       '环比': [101.5 , 101.2 , 101.3, 102.0 , 100.1 ],
     ...:       '同比': [120.7 , 127.3 , 119.4, 140.9 , 101.4 ],
     ...:       '定基': [121.4 , 127.8 , 120.0, 145.5 , 101.6 ]}

In [196]: d = pd.DataFrame(dl, index=['c1', 'c2', 'c3', 'c4', 'c5'])

In [197]: d
Out[197]:
      同比  城市    定基    环比
c1  120.7  北京  121.4  101.5
c2  127.3  上海  127.8  101.2
c3  119.4  广州  120.0  101.3
c4  140.9  深圳  145.5  102.0
c5  101.4  沈阳  101.6  100.1
```

```
In [200]: d.index
Out[200]: Index(['c1', 'c2', 'c3', 'c4', 'c5'], dtype='object')

In [201]: d.columns
Out[201]: Index(['同比', '城市', '定基', '环比'], dtype='object')

In [202]: d.values
Out[202]:
array([[120.7, '北京', 121.4, 101.5],
       [127.3, '上海', 127.8, 101.2],
       [119.4, '广州', 120.0, 101.3],
       [140.9, '深圳', 145.5, 102.0],
       [101.4, '沈阳', 101.6, 100.1]], dtype=object)
```

```
In [194]: import pandas as pd

In [195]: dl = {'城市': ['北京', '上海', '广州', '深圳', '沈阳'],
     ...:        '环比': [101.5 , 101.2 , 101.3, 102.0 , 100.1 ],
     ...:        '同比': [120.7 , 127.3 , 119.4, 140.9 , 101.4 ],
     ...:        '定基': [121.4 , 127.8 , 120.0, 145.5 , 101.6 ]}

In [196]: d = pd.DataFrame(dl, index=['c1', 'c2', 'c3', 'c4', 'c5'])

In [197]: d
Out[197]:
      同比  城市    定基    环比
c1  120.7  北京  121.4  101.5
c2  127.3  上海  127.8  101.2
c3  119.4  广州  120.0  101.3
c4  140.9  深圳  145.5  102.0
c5  101.4  沈阳  101.6  100.1
```

```
In [209]: d['同比']
Out[209]:
c1    120.7
c2    127.3
c3    119.4
c4    140.9
c5    101.4
Name: 同比, dtype: float64

In [210]: d.ix['c2']
Out[210]:
同比    127.3
城市       上海
定基    127.8
环比    101.2
Name: c2, dtype: object

In [211]: d['同比']['c2']
Out[211]: 127.3
```

DataFrame类型

DataFrame是二维带“标签”数组

	column_0	column_1	column_i
index_0 →	data_a	data_1	data_w

DataFrame基本操作类似Series，依据行列索引

Pandas库的数据类型操作

数据类型操作

如何改变Series和DataFrame对象？

增加或重排：重新索引

删除：drop

Python数据分析

重新索引

.reindex()能够改变或重排Series和DataFrame索引

```
In [194]: import pandas as pd

In [195]: dl = {'城市': ['北京', '上海', '广州', '深圳', '沈阳'],
     ...:        '环比': [101.5 , 101.2 , 101.3, 102.0 , 100.1 ],
     ...:        '同比': [120.7 , 127.3 , 119.4, 140.9 , 101.4 ],
     ...:        '定基': [121.4 , 127.8 , 120.0, 145.5 , 101.6 ]}

In [196]: d = pd.DataFrame(dl, index=['c1', 'c2', 'c3', 'c4', 'c5'])

In [197]: d
Out[197]:
      同比  城市     定基     环比
c1  120.7  北京  121.4  101.5
c2  127.3  上海  127.8  101.2
c3  119.4  广州  120.0  101.3
c4  140.9  深圳  145.5  102.0
c5  101.4  沈阳  101.6  100.1
```

```
In [218]: d = d.reindex(index=['c5','c4','c3','c2','c1'])

In [219]: d
Out[219]:
      同比  城市     定基     环比
c5  101.4  沈阳  101.6  100.1
c4  140.9  深圳  145.5  102.0
c3  119.4  广州  120.0  101.3
c2  127.3  上海  127.8  101.2
c1  120.7  北京  121.4  101.5

In [220]: d = d.reindex(columns=['城市','同比','环比','定基'])

In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

重新索引

.reindex(index=None, columns=None, …)的参数

参数	说明
index, columns	新的行列自定义索引
fill_value	重新索引中，用于填充缺失位置的值
method	填充方法，ffill当前值向前填充，bfill向后填充
limit	最大填充量
copy	默认True，生成新的对象，False时，新旧相等不复制

重新索引

```
In [218]: d = d.reindex(index=['c5','c4','c3','c2','c1'])

In [219]: d
Out[219]:
       同比  城市     定基     环比
c5  101.4  沈阳  101.6  100.1
c4  140.9  深圳  145.5  102.0
c3  119.4  广州  120.0  101.3
c2  127.3  上海  127.8  101.2
c1  120.7  北京  121.4  101.5

In [220]: d = d.reindex(columns=['城市','同比','环比','定基'])

In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

```
In [237]: newc = d.columns.insert(4,'新增')

In [238]: newd = d.reindex(columns=newc, fill_value=200)

In [239]: newd
Out[239]:
    城市     同比     环比     定基   新增
c5  沈阳  101.4  100.1  101.6  200
c4  深圳  140.9  102.0  145.5  200
c3  广州  119.4  101.3  120.0  200
c2  上海  127.3  101.2  127.8  200
c1  北京  120.7  101.5  121.4  200
```

索引类型

```
In [241]: d.index
Out[241]: Index(['c5', 'c4', 'c3', 'c2', 'c1'], dtype='object')

In [242]: d.columns
Out[242]: Index(['城市', '同比', '环比', '定基'], dtype='object')
```

Series和DataFrame的索引是Index类型

Index对象是不可修改类型

索引类型的常用方法

方法	说明
.append(idx)	连接另一个Index对象，产生新的Index对象
.diff(idx)	计算差集，产生新的Index对象
.intersection(idx)	计算交集
.union(idx)	计算并集
.delete(loc)	删除loc位置处的元素
.insert(loc,e)	在loc位置增加一个元素e

索引类型的使用

```
In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

```
In [258]: nc = d.columns.delete(2)

In [259]: ni = d.index.insert(5, 'c0')

In [260]: nd = d.reindex(index=ni, columns=nc, method='ffill')

In [261]: nd
Out[261]:
    城市     同比     定基
c5  沈阳  101.4  101.6
c4  深圳  140.9  145.5
c3  广州  119.4  120.0
c2  上海  127.3  127.8
c1  北京  120.7  121.4
c0  北京  120.7  121.4
```

Python数据分析与展示

删除指定索引对象

.drop()能够删除Series和DataFrame指定行或列索引

```
In [276]: a = pd.Series([9, 8, 7, 6], index=['a','b','c','d'])

In [277]: a
Out[277]:
a    9
b    8
c    7
d    6
dtype: int64

In [278]: a.drop(['b', 'c'])
Out[278]:
a    9
d    6
dtype: int64
```

```
In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

```
In [269]: d.drop('c5')
Out[269]:
    城市     同比     环比     定基
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4

In [270]: d.drop('同比', axis=1)
Out[270]:
    城市     环比     定基
c5  沈阳  100.1  101.6
c4  深圳  102.0  145.5
c3  广州  101.3  120.0
c2  上海  101.2  127.8
c1  北京  101.5  121.4
```

Pandas库的数据类型运算

算术运算法则

算术运算根据行列索引，补齐后运算，运算默认产生浮点数

补齐时缺项填充NaN（空值）

二维和一维、一维和零维间为广播运算

采用+ - * /符号进行的二元运算产生新的对象

数据类型的算术运算

```
In [298]: import pandas as pd

In [299]: import numpy as np

In [300]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [301]: a
Out[301]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [302]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [303]: b
Out[303]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19
```

```
In [304]: a + b
Out[304]:
      0     1     2     3   4
0   0.0   2.0   4.0   6.0 NaN
1   9.0  11.0  13.0  15.0 NaN
2  18.0  20.0  22.0  24.0 NaN
3   NaN   NaN   NaN   NaN NaN

In [305]: a * b
Out[305]:
      0     1      2      3   4
0   0.0   1.0    4.0    9.0 NaN
1  20.0  30.0   42.0   56.0 NaN
2  80.0  99.0  120.0  143.0 NaN
3   NaN   NaN    NaN    NaN NaN
```

自动补齐，缺项补NaN

数据类型的算术运算

方法形式的运算

方法	说明
.add(d, **argws)	类型间加法运算，可选参数
.sub(d, **argws)	类型间减法运算，可选参数
.mul(d, **argws)	类型间乘法运算，可选参数
.div(d, **argws)	类型间除法运算，可选参数

数据类型的算术运算

```
In [298]: import pandas as pd

In [299]: import numpy as np

In [300]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [301]: a
Out[301]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [302]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [303]: b
Out[303]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19
```

```
In [315]: b.add(a, fill_value = 100)
Out[315]:
       0      1      2      3      4
0    0.0    2.0    4.0    6.0  104.0
1    9.0   11.0   13.0   15.0  109.0
2   18.0   20.0   22.0   24.0  114.0
3  115.0  116.0  117.0  118.0  119.0

In [316]: a.mul(b, fill_value = 0)
Out[316]:
      0     1      2      3    4
0   0.0   1.0    4.0    9.0  0.0
1  20.0  30.0   42.0   56.0  0.0
2  80.0  99.0  120.0  143.0  0.0
3   0.0   0.0    0.0    0.0  0.0
```

fill_value参数替代NaN，替代后参与运算

数据类型的算术运算

```
In [306]: import pandas as pd

In [307]: import numpy as np

In [308]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [309]: b
Out[309]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19

In [310]: c = pd.Series(np.arange(4))

In [311]: c
Out[311]:
0    0
1    1
2    2
3    3
dtype: int32
```

```
In [321]: c - 10
Out[321]:
0   -10
1    -9
2    -8
3    -7
dtype: int32

In [322]: b - c
Out[322]:
      0     1     2     3   4
0   0.0   0.0   0.0   0.0 NaN
1   5.0   5.0   5.0   5.0 NaN
2  10.0  10.0  10.0  10.0 NaN
3  15.0  15.0  15.0  15.0 NaN
```

不同维度间为广播运算，一维Series默认在轴1参与运算

数据类型的算术运算

```
In [306]: import pandas as pd

In [307]: import numpy as np

In [308]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [309]: b
Out[309]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19

In [310]: c = pd.Series(np.arange(4))

In [311]: c
Out[311]:
0    0
1    1
2    2
3    3
dtype: int32
```

```
In [323]: b.sub(c, axis=0)
Out[323]:
    0   1   2   3   4
0   0   1   2   3   4
1   4   5   6   7   8
2   8   9  10  11  12
3  12  13  14  15  16
```

使用运算方法可以令一维Series参与轴0运算

比较运算法则

比较运算只能比较相同索引的元素，不进行补齐

二维和一维、一维和零维间为广播运算

采用> < >= <= == !=等符号进行的二元运算产生布尔对象

数据类型的比较运算

```
In [335]: import pandas as pd

In [336]: import numpy as np

In [337]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [338]: a
Out[338]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [339]: d = pd.DataFrame(np.arange(12, 0, -1).reshape(3,4))

In [340]: d
Out[340]:
    0   1   2  3
0  12  11  10  9
1   8   7   6  5
2   4   3   2  1
```

```
In [342]: a > d
Out[342]:
       0      1      2      3
0  False  False  False  False
1  False  False  False   True
2   True   True   True   True

In [343]: a == d
Out[343]:
       0      1      2      3
0  False  False  False  False
1  False  False   True  False
2  False  False  False  False
```

同维度运算，尺寸一致

数据类型的比较运算

```
In [344]: import pandas as pd

In [345]: import numpy as np

In [346]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [347]: a
Out[347]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [348]: c = pd.Series(np.arange(4))

In [349]: c
Out[349]:
0    0
1    1
2    2
3    3
dtype: int32
```

```
In [350]: a > c
Out[350]:
       0      1      2      3
0  False  False  False  False
1   True   True   True   True
2   True   True   True   True

In [351]: c > 0
Out[351]:
0    False
1     True
2     True
3     True
dtype: bool
```

不同维度，广播运算，默认在1轴

单元小结

Pandas库入门

Series = 索引 + 一维数据

DataFrame = 行列索引 + 二维数据

理解数据类型与索引的关系，操作索引即操作数据

重新索引、数据删除、算术运算、比较运算

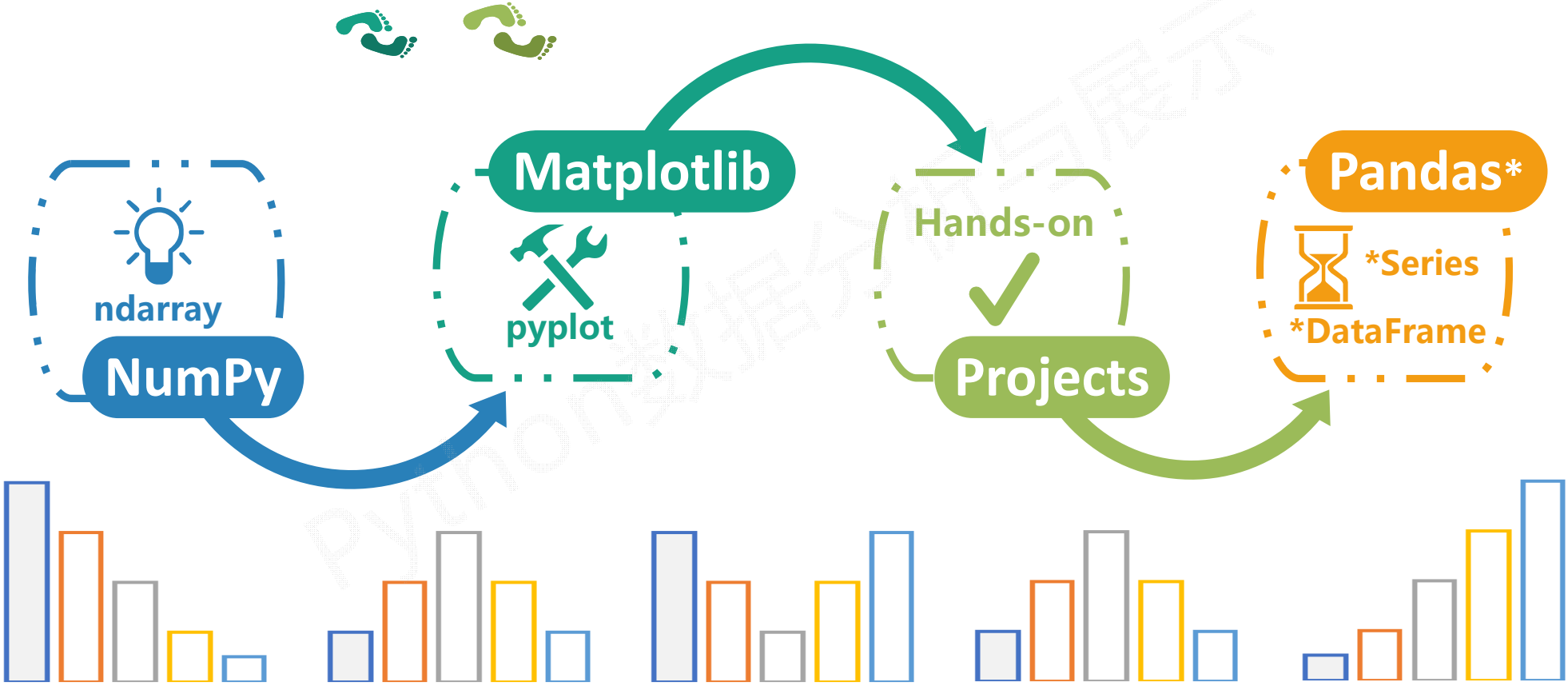
像对待单一数据一样对待Series和DataFrame对象

Pandas数据特征分析

python
弹指之间 · 享受创新

Python数据分析与展示

掌握表示、清洗、统计和展示数据的能力



O4X -Tian

数据的排序

对一组数据的理解

3.1404

3.1413 3.1401 3.1376

3.1398 3.1349

一组数据

表达一个或多个含义

摘要 ⇨

基本统计（含排序）

分布/累计统计

数据特征

相关性、周期性等

数据挖掘（形成知识）

数据形成有损特征的过程

Pandas库的数据排序

.sort_index()方法在指定轴上根据索引进行排序，默认升序

.sort_index(axis=0, ascending=True)

```
In [358]: import pandas as pd

In [359]: import numpy as np

In [360]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [361]: b
Out[361]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19
```

```
In [362]: b.sort_index()
Out[362]:
    0   1   2   3   4
a   5   6   7   8   9
b  15  16  17  18  19
c   0   1   2   3   4
d  10  11  12  13  14
```

```
In [363]: b.sort_index(ascending=False)
Out[363]:
    0   1   2   3   4
d  10  11  12  13  14
c   0   1   2   3   4
b  15  16  17  18  19
a   5   6   7   8   9
```

Pandas库的数据排序

.sort_index(axis=0, ascending=True)

```
In [358]: import pandas as pd

In [359]: import numpy as np

In [360]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [361]: b
Out[361]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19
```

```
In [366]: c = b.sort_index(axis=1, ascending=False)

In [367]: c
Out[367]:
    4   3   2   1   0
c   4   3   2   1   0
a   9   8   7   6   5
d  14  13  12  11  10
b  19  18  17  16  15

In [368]: c = c.sort_index()

In [369]: c
Out[369]:
    4   3   2   1   0
a   9   8   7   6   5
b  19  18  17  16  15
c   4   3   2   1   0
d  14  13  12  11  10
```

Pandas库的数据排序

.sort_values()方法在指定轴上根据数值进行排序，默认升序

```
Series.sort_values(axis=0, ascending=True)
DataFrame.sort_values(by, axis=0, ascending=True)
```

by : axis轴上的某个索引或索引列表

Pandas库的数据排序

.sort_values(by, axis=0, ascending=True)

```
In [358]: import pandas as pd

In [359]: import numpy as np

In [360]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [361]: b
Out[361]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19
```

```
In [386]: c = b.sort_values(2, ascending=False)

In [387]: c
Out[387]:
    0   1   2   3   4
b  15  16  17  18  19
d  10  11  12  13  14
a   5   6   7   8   9
c   0   1   2   3   4

In [388]: c = c.sort_values('a', axis=1, ascending=False)

In [389]: c
Out[389]:
    4   3   2   1   0
b  19  18  17  16  15
d  14  13  12  11  10
a   9   8   7   6   5
c   4   3   2   1   0
```

Pandas库的数据排序

NaN统一放到排序末尾

```
In [395]: import pandas as pd

In [396]: import numpy as np

In [397]: a = pd.DataFrame(np.arange(12).reshape(3,4), index=['a', 'b', 'c'])

In [398]: a
Out[398]:
   0  1   2   3
a  0  1   2   3
b  4  5   6   7
c  8  9  10  11

In [399]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [400]: b
Out[400]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19

In [401]: c = a + b

In [402]: c
Out[402]:
      0     1     2     3   4
a   5.0   7.0   9.0  11.0 NaN
b  19.0  21.0  23.0  25.0 NaN
c   8.0  10.0  12.0  14.0 NaN
d   NaN   NaN   NaN   NaN NaN
```

```
In [403]: c.sort_values(2, ascending = False)
Out[403]:
      0     1     2     3   4
b  19.0  21.0  23.0  25.0 NaN
c   8.0  10.0  12.0  14.0 NaN
a   5.0   7.0   9.0  11.0 NaN
d   NaN   NaN   NaN   NaN NaN

In [404]: c.sort_values(2, ascending = True)
Out[404]:
      0     1     2     3   4
a   5.0   7.0   9.0  11.0 NaN
c   8.0  10.0  12.0  14.0 NaN
b  19.0  21.0  23.0  25.0 NaN
d   NaN   NaN   NaN   NaN NaN
```

数据的基本统计分析

基本的统计分析函数

适用于Series和DataFrame类型

方法	说明
.sum()	计算数据的总和，按0轴计算，下同
.count()	非NaN值的数量
.mean() .median()	计算数据的算术平均值、算术中位数
.var() .std()	计算数据的方差、标准差
.min() .max()	计算数据的最小值、最大值

基本的统计分析函数

适用于Series类型

方法	说明
.argmin()　.argmax()	计算数据最大值、最小值所在位置的索引位置（自动索引）
.idxmin()　.idxmax()	计算数据最大值、最小值所在位置的索引（自定义索引）

基本的统计分析函数

适用于Series和DataFrame类型

方法	说明
.describe()	针对0轴（各列）的统计汇总

基本的统计分析函数

```
In [425]: import pandas as pd

In [426]: a = pd.Series([9, 8, 7, 6], index=['a','b','c','d'])

In [427]: a
Out[427]:
a    9
b    8
c    7
d    6
dtype: int64

In [428]: a.describe()
Out[428]:
count    4.000000
mean     7.500000
std      1.290994
min      6.000000
25%      6.750000
50%      7.500000
75%      8.250000
max      9.000000
dtype: float64
```

```
In [431]: type(a.describe())
Out[431]: pandas.core.series.Series

In [432]: a.describe()['count']
Out[432]: 4.0

In [433]: a.describe()['max']
Out[433]: 9.0
```

Python数据分析与展示

基本的统计分析函数

```
In [434]: import pandas as pd

In [435]: import numpy as np

In [436]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [437]: b.describe()
Out[437]:
              0          1          2          3          4
count   4.000000   4.000000   4.000000   4.000000   4.000000
mean    7.500000   8.500000   9.500000  10.500000  11.500000
std     6.454972   6.454972   6.454972   6.454972   6.454972
min     0.000000   1.000000   2.000000   3.000000   4.000000
25%     3.750000   4.750000   5.750000   6.750000   7.750000
50%     7.500000   8.500000   9.500000  10.500000  11.500000
75%    11.250000  12.250000  13.250000  14.250000  15.250000
max    15.000000  16.000000  17.000000  18.000000  19.000000
```

```
In [443]: type(b.describe())
Out[443]: pandas.core.frame.DataFrame

In [444]: b.describe().ix['max']
Out[444]:
0    15.0
1    16.0
2    17.0
3    18.0
4    19.0
Name: max, dtype: float64

In [445]: b.describe()[2]
Out[445]:
count     4.000000
mean      9.500000
std       6.454972
min       2.000000
25%       5.750000
50%       9.500000
75%      13.250000
max      17.000000
Name: 2, dtype: float64
```

数据的累计统计分析

累计统计分析函数

适用于Series和DataFrame类型，累计计算

方法	说明
.cumsum()	依次给出前1、2、…、n个数的和
.cumprod()	依次给出前1、2、…、n个数的积
.cummax()	依次给出前1、2、…、n个数的最大值
.cummin()	依次给出前1、2、…、n个数的最小值

累计统计分析函数

```
In [448]: import pandas as pd

In [449]: import numpy as np

In [450]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [451]: b
Out[451]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19
```

```
In [452]: b.cumsum()
Out[452]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   7   9  11  13
d  15  18  21  24  27
b  30  34  38  42  46
```

```
In [453]: b.cumprod()
Out[453]:
   0     1     2     3     4
c  0     1     2     3     4
a  0     6    14    24    36
d  0    66   168   312   504
b  0  1056  2856  5616  9576
```

```
In [454]: b.cummin()
Out[454]:
   0  1  2  3  4
c  0  1  2  3  4
a  0  1  2  3  4
d  0  1  2  3  4
b  0  1  2  3  4
```

```
In [455]: b.cummax()
Out[455]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19
```

累计统计分析函数

适用于Series和DataFrame类型，滚动计算（窗口计算）

方法	说明
.rolling(w).sum()	依次计算相邻w个元素的和
.rolling(w).mean()	依次计算相邻w个元素的算术平均值
.rolling(w).var()	依次计算相邻w个元素的方差
.rolling(w).std()	依次计算相邻w个元素的标准差
.rolling(w).min() .max()	依次计算相邻w个元素的最小值和最大值

累计统计分析函数

```
In [448]: import pandas as pd

In [449]: import numpy as np

In [450]: b = pd.DataFrame(np.arange(20).reshape(4,5), index=['c','a','d','b'])

In [451]: b
Out[451]:
    0   1   2   3   4
c   0   1   2   3   4
a   5   6   7   8   9
d  10  11  12  13  14
b  15  16  17  18  19
```

```
In [465]: b.rolling(2).sum()
Out[465]:
      0     1     2     3     4
c   NaN   NaN   NaN   NaN   NaN
a   5.0   7.0   9.0  11.0  13.0
d  15.0  17.0  19.0  21.0  23.0
b  25.0  27.0  29.0  31.0  33.0

In [466]: b.rolling(3).sum()
Out[466]:
      0     1     2     3     4
c   NaN   NaN   NaN   NaN   NaN
a   NaN   NaN   NaN   NaN   NaN
d  15.0  18.0  21.0  24.0  27.0
b  30.0  33.0  36.0  39.0  42.0
```

数据的相关分析

相关分析

两个事物，表示为X和Y，如何判断它们之间的存在相关性？

相关性

- X增大，Y增大，两个变量正相关
- X增大，Y减小，两个变量负相关
- X增大，Y无视，两个变量不相关

协方差

两个事物，表示为X和Y，如何判断它们之间的存在相关性？

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

- 协方差>0，X和Y正相关
- 协方差<0，X和Y负相关
- 协方差=0，X和Y独立无关

Pearson相关系数

两个事物，表示为X和Y，如何判断它们之间的存在相关性？

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

- 0.8-1.0 极强相关
- 0.6-0.8 强相关
- 0.4-0.6 中等程度相关
- 0.2-0.4 弱相关
- 0.0-0.2 极弱相关或无相关

r取值范围[-1,1]

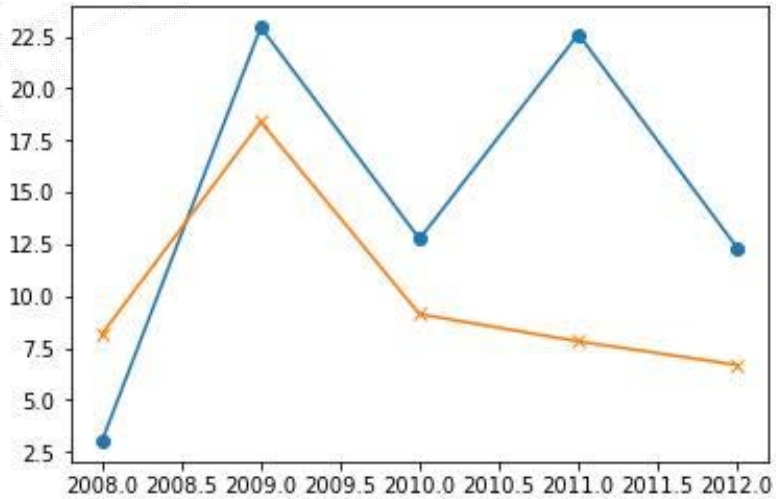
相关分析函数

适用于Series和DataFrame类型

方法	说明
.cov()	计算协方差矩阵
.corr()	计算相关系数矩阵，Pearson、Spearman、Kendall等系数

实例：房价增幅与M2增幅的相关性

```
In [484]: import pandas as pd

In [485]: hprice = pd.Series([3.04, 22.93, 12.75, 22.6, 12.33], index=['2008',
'2009', '2010', '2011', '2012'])

In [486]: m2 = pd.Series([8.18, 18.38, 9.13, 7.82, 6.69], index=['2008', '2009',
'2010', '2011', '2012'])

In [487]: hprice.corr(m2)

Out[487]: 0.5239439145220387
```



单元小结

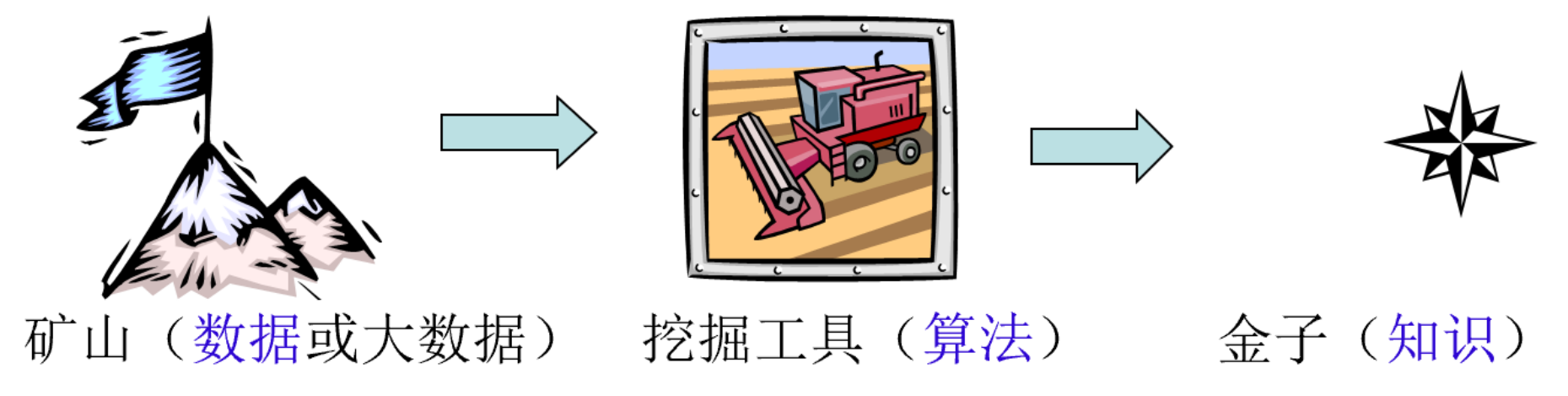
Pandas数据特征分析

一组数据的摘要

排序	.sort_index()　.sort_values()
基本统计函数	.describe()
累计统计函数	.cum*()　.rolling().*()
相关性分析	.corr()　.cov()

6.3 数据挖掘sklearn

- 数据挖掘的定义
  - 简单定义：即从数据中发现知识
  - 略微具体的定义：从大量的数据中挖掘哪些令人感兴趣的、有用的、隐含的、先前未知的和可能有用的模式或知识



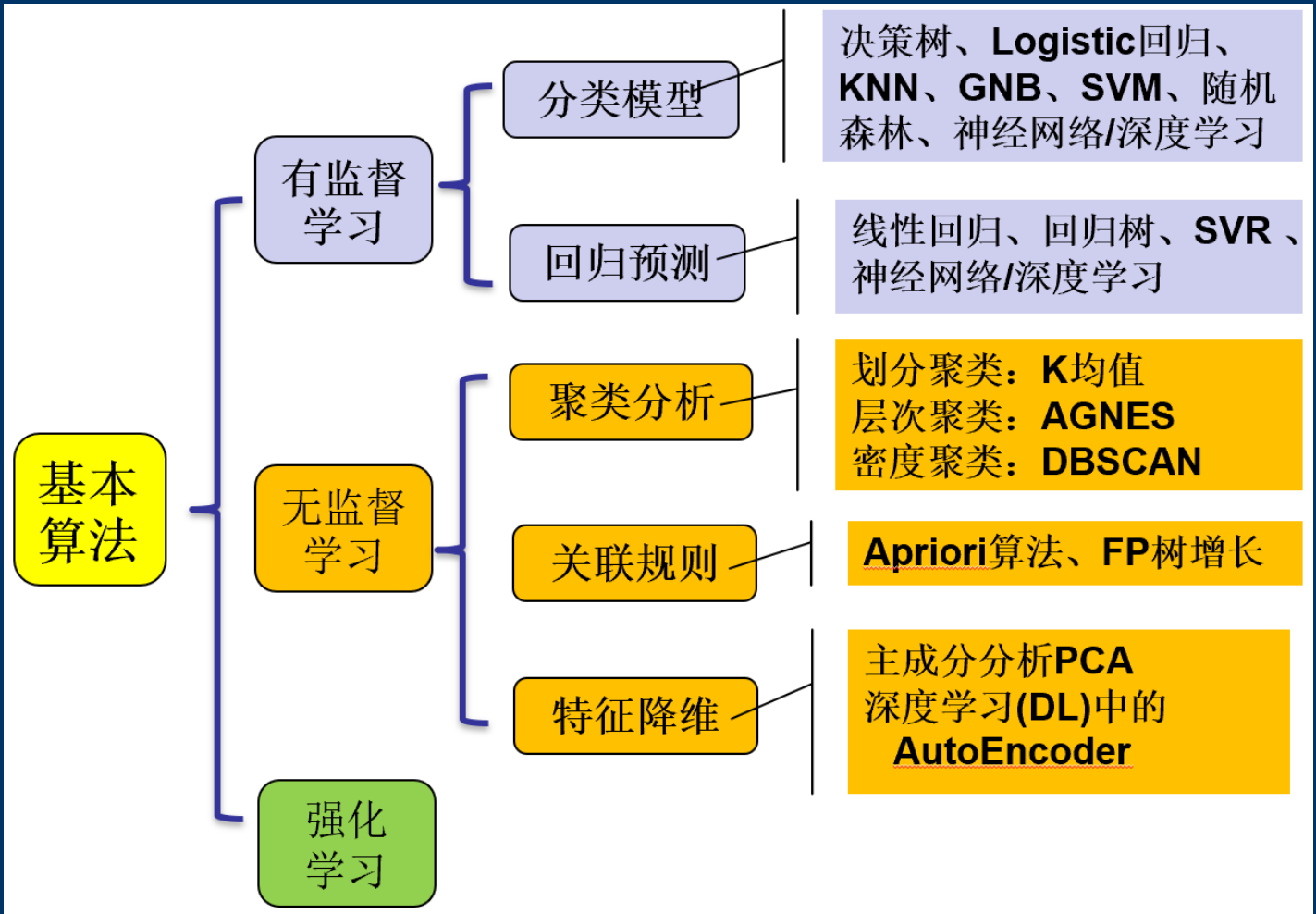
6.3 数据挖掘sklearn

- 数据挖掘三大基本任务
  - 分类预测(Classification and Prediction)：KNN、GNB、Logistic回归、决策树、SVM、线性回归、神经网络等
  - 聚类分析(Clustering Analysis)：k-均值等
  - 关联规则(Association Rule)：Apriori算法等

- 机器学习是数据挖掘的算法支撑，主要包括三种学习方式：
  - 有监督学习(supervised learning)
  - 无监督学习(unsupervised learning)
  - 强化学习(reinforcement learning)

6.3-1 数据挖掘算法

数据挖掘算法一览表



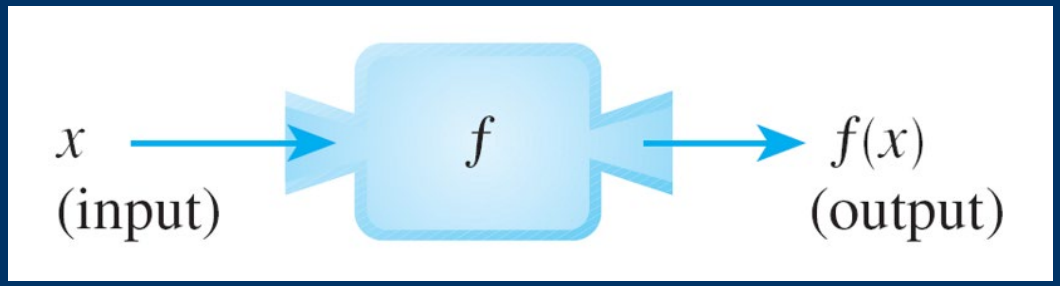
- 仅简要介绍**KNN、GNB、线性回归以及Logistic回归算法**，其余算法自学！

6.3-1 数据挖掘算法

分类器(classifier)的目标

- 数据挖掘中的分类(classification)任务，是在若干样本数据上，学习到一个模型，然后用这个模型对新数据进行预测（分类）。

- 分类的目的是获得一个分类函数或分类模型(也常常称作分类器)，该模型能把数据样本映射到某一个给定类别。
  - ✓如预测明天的天气是晴天、雨天还是阴天？(三分类问题)



f的结构可以很丰富

- 每个样本一般由多个元素(数字)表示，则称为多维数据或者多变量数据(Multi-Dimensional/Multivariate Data)，每个变量称为一个特征(Feature)，从而样本一般由一个特征向量(feature vector)表示。

6.3-1 数据挖掘算法

K最近邻(KNN)

- 就是针对未知类标号的新样例，在已知类标号的训练集中，找出*k*个最相似实例，进行即时分类。

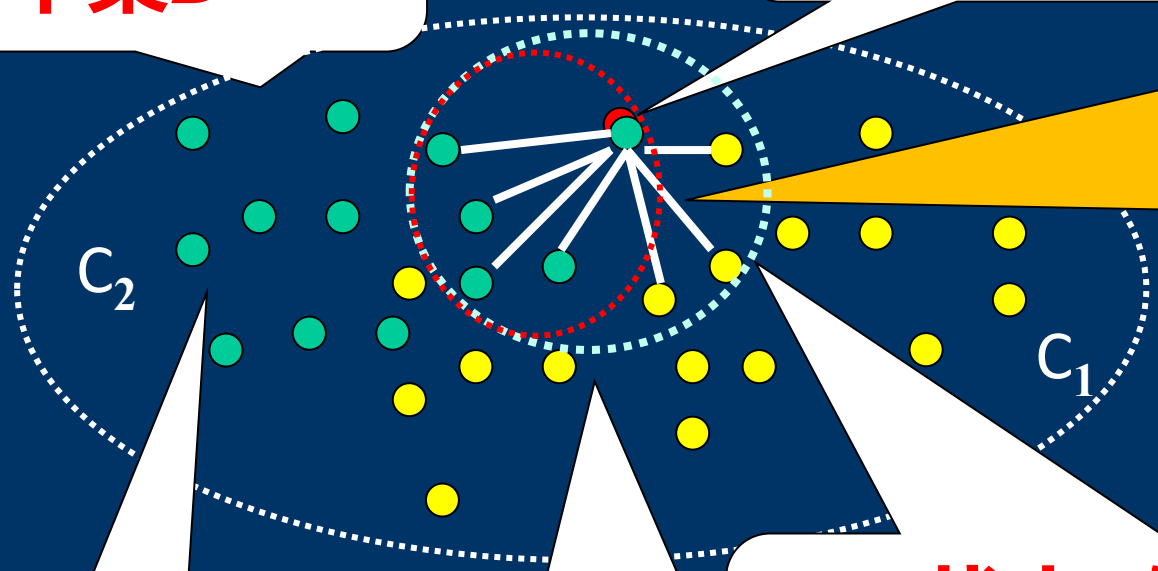
- 算法特点：
  - ✓这种方法为局部逼近，复杂度低，不失为一个方法。
  - ✓这是一种基于示例的学习方法
  - ✓事先不给出学习模型，所以把它称作是一种懒惰的(lazy)或基于要求的学习方法。

6.3-1 数据挖掘算法　　k-最近邻算法

(1) 给定 N个已知类别样本集D

(2) 输入未知类别样本x

(3) 计算x到$x_i \in D$，$(i = 1,2,\ldots,N)$的距离$d_i(x)$



(4) 找出x的k个最近邻元 $X_k = \{x_i, i = 1, 2, \ldots, k\}$

(5) 若$k = 7$，看X_k中属于哪一类的样本最多？$k_1 = 3 < k_2 = 4$

(6) 判$x \in C_2$

6.3-1 数据挖掘算法

朴素贝叶斯方法

- 贝叶斯公式：

$$P(H|X) = \frac{P(XH)}{P(X)} = \frac{P(X|H)P(H)}{P(X)}$$

✓ H：是一个假设，如假设X属于类C。

✓ X：为未标注测试样本，表示为一个特征向量，但不知属于哪类，要确定其类。

- 什么是朴素贝叶斯？

✓ 给定样本的类标号，假定属性值相互条件独立，即属性间不存在依赖关系，即

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

✓ 例如，$\boldsymbol{P}$(收入低, 是学生, 信用良好|买电脑) = $\boldsymbol{P}$(收入低|买电脑) * $\boldsymbol{P}$(是学生|买电脑) * $\boldsymbol{P}$(信用良好|买电脑)

例子：朴素贝叶斯分类

- 未标注（待预测）样本具有属性：收入低、是学生、信用良好，属于哪个类？
- 记$X = (\text{收入低}, \text{是学生}, \text{信用良好})$
- 根据贝叶斯公式，

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

问题相当于比较下列两个式子的大小：

$P(X|\text{买电脑}) * P(\text{买电脑})$？
$P(X|\text{不买电脑}) * P(\text{不买电脑})$？

收入	是否学生	信用状况	买电脑?
高	否	良好	否
高	是	良好	是
中	是	良好	是
低	是	良好	是
低	否	良好	否
低	否	良好	否
中	否	良好	否
低	是	良好	?

例子：朴素贝叶斯分类

- 先计算类别的先验概率：
  - $P(\text{买电脑}) = 3/7$
  - $P(\text{不买电脑}) = 4/7$

- 类条件下X的后验概率：
  - $P(X|\text{买电脑}) = P(\text{收入低}|\text{买电脑}) * P(\text{是学生}|\text{买电脑}) * P(\text{信用良好}|\text{买电脑}) = (1/3) * (3/3) * (3/3) = 1/3$
  - $P(X|\text{不买电脑}) = P(\text{收入低}|\text{不买电脑}) * P(\text{是学生}|\text{不买电脑}) * P(\text{信用良好}|\text{不买电脑}) = (2/4) * (0/4) * (4/4) = 0$

收入	是否学生	信用状况	买电脑？
高	否	良好	否

- 后验概率的比较：
  - P(X|买电脑) *P (买电脑) =3/7*1/3=1/7
  - P(X|不买电脑)* P(不买电脑) =4/7*0=0
  - 因此P(买电脑|X)> P(不买电脑|X)，预测X是会买电脑的

6.3-1 数据挖掘算法

- 高斯朴素贝叶斯方法GNB
  - 如果一个特征取离散值，则$P(x_k|C_i) = |S_{ik}|/|S_i|$，$S_i$为$C_i$类的样本集，$S_{ik}$为$S_i$中特征取$x_k$的样本集
  - 如果是连续值特征，则通常假定该属性服从高斯分布，即

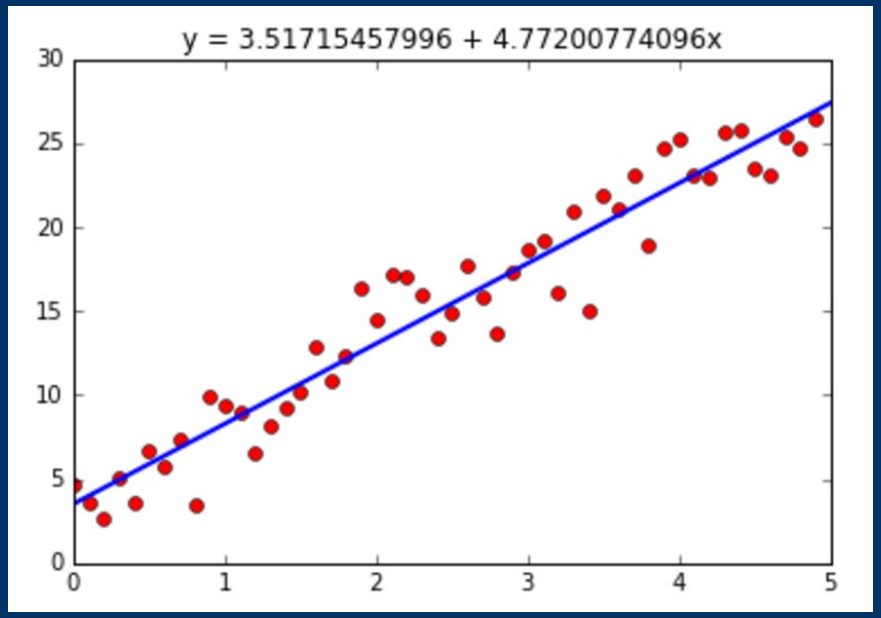
$$P(x_k|C_i) = g(x_k, \mu_{c_i}, \sigma_{c_i}) = \frac{1}{\sqrt{2\pi\sigma_{c_i}}} e \frac{(x_k - \mu_{c_i})}{2\sigma_{c_i}^2}$$

其中 μ_{c_i}, σ_{c_i} 分别为C_i类的样本集S_i中特征的平均值与标准差。

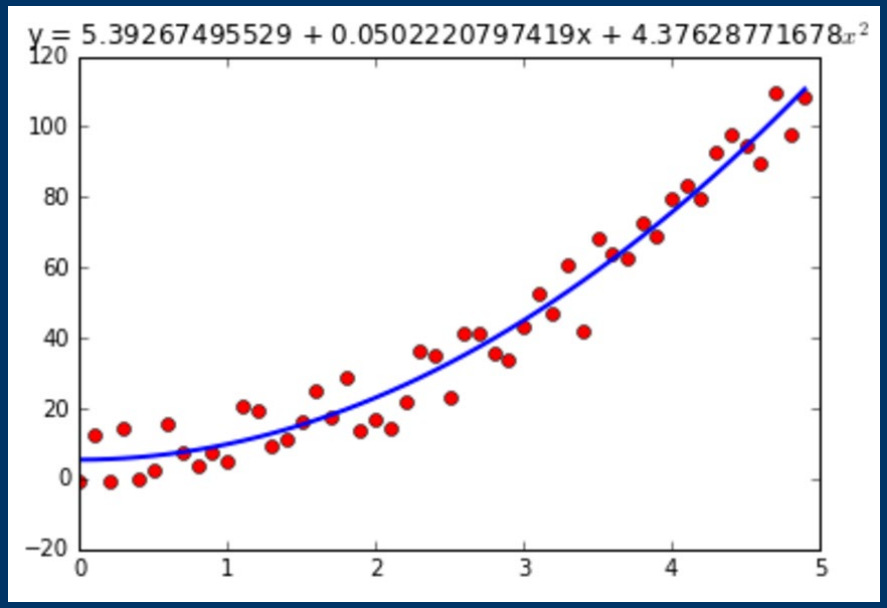
Gaussian naïve Bayes

6.3-1 数据挖掘算法

- 分类(classification)：把样本分配到若干类之一(离散标记)
- 回归(regression)：研究分析因变量随自变量变化的关系形式，建立变量之间的方程
- (实值)预测：将回归方程作为模型，根据自变量在预测期的数量变化来预测因变量值



线性回归



非线性回归

6.3-1 数据挖掘算法

多元线性回归模型

- 回归模型：$Y = w_1x_1 + w_2x_2 + \cdots + w_mx_m + b + e$
- 记$\boldsymbol{x}_i = (x_1^{(i)}, x_2^{(i)}, \ldots, x_m^{(i)}, 1)^T$为数据集中第$i$条数据。
  设$\boldsymbol{\omega} = (w_1, w_2, \ldots, w_m, b)^{\boldsymbol{T}}$ 为拟合函数的待定参数(权重系数+截距)
- 拟合函数可以简记为：$f(\boldsymbol{x}_i; \boldsymbol{\omega}) = \boldsymbol{\omega}^T\boldsymbol{x}_i$

代价函数（cost function）

$$\boldsymbol{J}(\boldsymbol{\omega}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{\omega}^{\boldsymbol{T}}\boldsymbol{x}_i)^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega})$$

其中， $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix}$ $\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ $N = n$为样本数

6.3-1 数据挖掘算法

代价函数(续)

$$
\begin{aligned}
J(\boldsymbol{\omega}) &= \frac{1}{N}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\omega})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\omega}) \\
&= \frac{1}{N}(\boldsymbol{y}^T-\boldsymbol{\omega}^T\boldsymbol{X}^T)(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\omega}) \\
&= \frac{1}{N}(\boldsymbol{y}^T\boldsymbol{y}-\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{\omega}-\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{y}+\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega}) \\
&= \frac{1}{N}(\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega}-2\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{y}+\boldsymbol{y}^T\boldsymbol{y})
\end{aligned}
$$

优化目标：让代价函数最小

$$
\frac{\partial J}{\partial \boldsymbol{\omega}} = \frac{1}{N}(2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega}-2\boldsymbol{X}^T\boldsymbol{y}) = 0
$$

$$
\Rightarrow
$$

$$
\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega} = \boldsymbol{X}^T\boldsymbol{y}
$$

$$
\Rightarrow
$$

$$
\boldsymbol{\omega} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}
$$

最小二乘法

6.3-1 数据挖掘算法

线性回归

- $z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta^T x$
- 其中$\theta = (\theta_0, \theta_1, \theta_2, \ldots, \theta_n)^T$, $x = (1, x_1, x_2, \ldots, x_n)^T$

logistic回归

- 基于线性回归，是一种广义线性回归模型
- 正样本概率：$P(y = 1|x;\theta) = h_\theta\ (x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}}$
- 负样本概率：$P(y = \mathbf{0}|x;\theta)$=1 − $P(y = 1|x;\theta)$
- 其中$f(z) = \frac{1}{1+e^{-z}}$称为sigmoid函数
- logistic回归是一种分类算法，而线性回归是一种实值预测(回归)方法。
- 可以看出， logistic regression算法是将线性函数的结果映射到了sigmoid函数中。
- 参数学习：极大似然估计

6.3-2 模型评估

- 建立的分类模型好不好？该如何评估？
  - 二分类问题中的混淆矩阵(Confusion matrix)

- 二分类问题
  - 真阳性(True Positive, TP)：实际为阳性 预测为阳性
  - 真阴性(True Negative, TN)：实际为阴性 预测为阴性
  - 假阳性(False Positive, FP)：实际为阴性 预测为阳性
  - 假阴性(False Negative, FN)：实际为阳性 预测为阴性

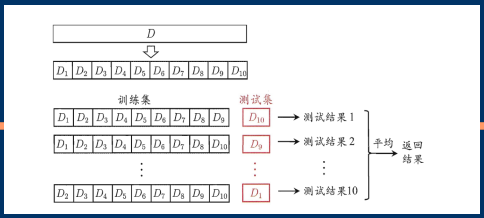
- 混淆矩阵

Confusion Matrix		Predict	
		1	0
Real	1	TP	FN
	0	FP	TN

6.3-2 模型评估

- 评估分类模型的性能指标
  - 灵敏度(Sensitivity)：也称查全率(Recall)
    Sn $= TP/(TP+FN)$
  - 特异度(Specificity):
    Sp=$TN/(TN+FP)$
  - 精度(Precision):
    Precisioin $= TP/(TP+FP)$
  - 准确率(Accuracy):
    Acc $= (TP+TN)/(TP+TN+FN+FP)$
  - F1 score=$2 * Precision * Recall/(Precision+Recall)$
  - ROC曲线下方的面积AUC(area under curve)也可以用来评估分类模型的预测性能

6.3-2 模型评估



评估方法

- 留出法(**hold out**): 直接将数据集D划分为两个互斥的集合，其中一个集合作为训练集S,另一个作为测试集T。

- 交叉验证法(**cross validation**): 也称为K折交叉验证(**K-fold cross validation**)
  - ✓准备好训练集，随机分成K等份，即K个子集；一般情况，**K=5或10.**
  - ✓将其中1个子集作为测试集，其余K-1个子集合并作为训练集。**每个子集均作为测试集一次**，如此**交叉验证重复K次，平均K次结果**最终得到一个单一估测。

- 留一法(**leave-one-out**): 特殊的交叉验证法，即**K=N**(样本总量)

- 自助法(**boostrap**)
  - ✓给定N个样本的数据集D, 每次从D中随机挑选一个样本，将其放入D’,然后再放回数据集D，使得样本在下次采样时仍有可能被采到,这个过程重复执行N次后，就会得到包含N个数据样本的集合D’。
  - ✓此过程称为**自助采样**，且初始数据集D中约有36.8%的样本未出现在样本数据集D’中，于是可用D’用作训练集，D\D’用作测试集（36.8%的样本）。

6.3-3 数据挖掘平台sklearn

sklearn简介

- scikit-learn(简称sklearn)是机器学习中常用的第三方模块，封装了常用的机器学习方法，包括：
  - ✓回归(Regression)
  - ✓分类(Classfication)
  - ✓聚类(Clustering)
  - ✓降维(Dimensionality Reduction)

scikit-learn

Machine Learning in Python

Getting Started | Release Highlights for 0.24 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

```python
#ch6_SimpleClassifiers.py

#K-nearest neighbors
X = [[0.2,1.,1.5], [1.2,3.1,0.8], [2.,3.2,1.], [3.,2.,0.3]]
y = [0, 0, 1, 1]#三分类：训练集含4个样本,类标号为0或1
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
newx=[[1.1,-1.2,2.3]]
print("Pred_KNN:",neigh.predict(newx))
print("Prob_KNN:",neigh.predict_proba(newx))
```

```
Pred_KNN: [0]
Prob_KNN: [[0.66666667 0.33333333]]
```

```python
#Gaussian naive bayes
import numpy as np
X = np.array([[-1,-1],[-2,-1],[-3,-2],[1,1],[2,1],[3,2]])
Y = np.array([1, 1, 1, 2, 2, 2])
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X, Y)
newx=[[0.8, -1],[1,1.2]]
print("Pred_GNB:",clf.predict(newx))
print("Prob_GNB:",clf.predict_proba(newx))
```

```
Pred_GNB: [1 2]
Prob_GNB: [[9.99253971e-01 7.46028997e-04]
 [1.38163307e-09 9.99999999e-01]]
```

```python
#logistic regression
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
print("X shape:",X.shape,type(X)) #150个样本，4个特征
print("label:",set(y)) #3分类问题
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression().fit(X, y)
testX=X[:2, :]
print("Pred_logR:",clf.predict(testX))
print("Prob_logR:",clf.predict_proba(testX))
```

```
X shape: (150, 4) <class 'numpy.ndarray'>
label: {0, 1, 2}
Pred_logR: [0 0]
Prob_logR: [[8.79681649e-01 1.20307538e-01 1.08131372e-05]
 [7.99706325e-01 2.00263292e-01 3.03825365e-05]]
```

6.3-4 sklearn中的分类器

sklearn分类器实战通用步骤

- 1.准备好训练数据集，包括样本特征数据X及其对应的标签数据y
- 2.导入分类算法对应的Python类，创建一个分类器实例clf
- 3.训练模型：clf.fit(X,y)
- 4.预测新样本数据testX：clf.predict(testX) 或 clf.predict_prob(testX)
- 5.计算各种性能指标(需要知道testX的标签)，评估模型好坏。
  - ✓ Sensitivity or Recall
  - ✓ Specificity
  - ✓ Precision
  - ✓ Accuracy
  - ✓ F1 score
  - ✓ AUC

6.3-4 sklearn中的分类器

- sklearn中分类器的Python类
  - K最近邻KNN
    - ✓ from sklearn.neighbors import KNeighborsClassifier
  - 高斯朴素贝叶斯GNB
    - ✓ from sklearn.naive_bayes import GaussianNB
  - Logistic回归LogR
    - ✓ from sklearn.linear_model import LogisticRegression
  - 决策树DT
    - ✓ from sklearn.tree import DecisionTreeClassifier
  - 随机森林RF
    - ✓ from sklearn.ensemble import RandomForestClassifier
  - 梯度提升决策树GBDT
    - ✓ from sklearn.ensemble import GradientBoostingClassifier
  - 支持向量机SVM
    - ✓ from sklearn.svm import SVC
  - 神经网络NN：多层感知器MLP
    - ✓ from sklearn.neural_network import MLPClassifier

- 详情参考sklearn API：https://scikit-learn.org/stable/modules/classes.html

6.3-4 sklearn中的分类器

- 分类案例：澳大利亚银行客户的信用评级
  - 数据集：包括690位客户的信用数据，最后1列为标签0或1
  - hold out评估方式：随机选择600位客户数据(AustralianCredit_train600.txt)作为训练集，其余90位客户数据(AustralianCredit_test90.txt)为测试集

```python
#ch6_creditscoring.py
import numpy as np
from sklearn import linear_model
from math import sqrt

data1=np.loadtxt('AustralianCredit_train600.txt') # trainset
train_features=data1[:,:-1] # features
train_labels=data1[:,-1].astype(int) # labels
data2=np.loadtxt('AustralianCredit_test90.txt') # testset
test_features=data2[:,:-1]  # features
test_labels=data2[:,-1].astype(int) # labels

clf = linear_model.LogisticRegression() #classifier
clf.fit(train_features, train_labels)
pred_labels=clf.predict(test_features) # predicted labels
print("真实类别:\n",test_labels)
print("预测类别:\n",pred_labels)

TP,TN,FP,FN=0,0,0,0   # 1: positive, 0: negative
for x,y in zip(test_labels,pred_labels):
    if x==1 and y==1: TP+=1
    if x==0 and y==0: TN+=1
    if x==1 and y==0: FN+=1
    if x==0 and y==1: FP+=1
print("混淆矩阵: ","TP=",TP,"TN=",TN,"FP=",FP,"FN=",FN)
sen,spe,pre=TP/(TP+FN),TN/(TN+FP),TP/(TP+FP)  #sensitivity,specificity,precision
acc=(TP+TN)/(TP+TN+FN+FP) #accuracy
MCC=(TP*TN-FP*FN)/sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))  #MCC
```

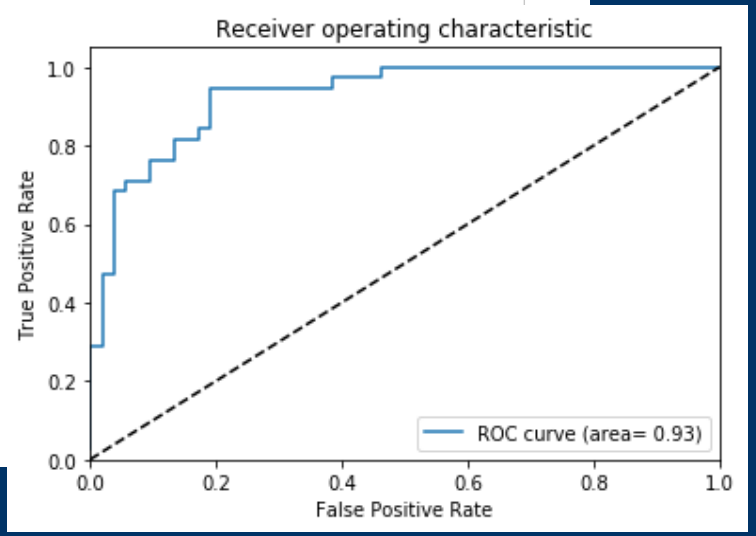
```
真实类别:
 [0 1 1 1 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 1 0
0 0 0 0 1 0 1 0 0 0 0
 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1 0 1 1
0 0 0 0 0 0 1 1 0 1
 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 1]
预测类别:
 [0 1 1 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1
1 0 1 0 0 0 1 0 0 1 0
 0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1
0 0 0 1 0 0 1 1 0 1
 0 0 0 1 0 1 0 1 1 0 1 1 0 1 1 0]
混淆矩阵:  TP= 33 TN= 42 FP= 10 FN= 5
sen:0.8684 spe:0.8077 pre:0.7674 acc:0.8333 mcc:0.6685
```

6.3-4 sklearn中的分类器

分类案例 (续)

```
混淆矩阵:
 [[42 10]
 [ 5 33]]
准确率: 0.8333
精度: 0.7674
召回率: 0.8684
F1 score:0.8148
MCC:0.6685
AUC:0.9276
```

```python
#调用sklearn自带的性能指标计算方法
from sklearn import metrics
print("混淆矩阵: \n",metrics.confusion_matrix(test_labels,pred_labels))
print("准确率: %.4f"%metrics.accuracy_score(test_labels,pred_labels))
print("精度: %.4f"%metrics.precision_score(test_labels,pred_labels,pos_label=1))
print("召回率: %.4f"%metrics.recall_score(test_labels,pred_labels,pos_label=1))
print("F1 score:%.4f"%metrics.f1_score(test_labels,pred_labels))
print("MCC:%.4f"%metrics.matthews_corrcoef(test_labels,pred_labels))

pred_prob=clf.predict_proba(test_features)
auc=metrics.roc_auc_score(test_labels,pred_prob[:,1])
print('AUC:%.4f'%auc)
fpr, tpr, thresholds = metrics.roc_curve(test_labels, pred_prob[:,1], pos_label=1)
#print(fpr,tpr,thresholds)

import matplotlib.pyplot as plt
plt.figure()
plt.plot(fpr,tpr,label='ROC curve (area= %0.2f)'%auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

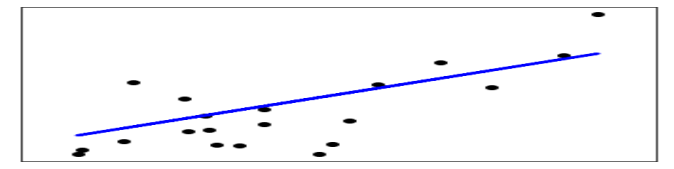


6.3-5 sklearn中的回归方法

线性回归

```python
#ch6_linearRegression.py
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3 #真实的y值
reg = LinearRegression()
reg.fit(X, y)
print("weights:",reg.coef_)
print("intercept:",reg.intercept_)
print("prediction:",reg.predict(np.array([[3, 5]])))
```

6.3-5 sklearn中的回归方法



- 线性回归实例
  - 仅考虑1个特征

```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```

```python
#ch6_linReg_example.py
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
#Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
diabetes_X = diabetes_X[:, np.newaxis, 2] #Use only one feature
#Split the features and targets into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

regr = linear_model.LinearRegression()#Linear regression object
regr.fit(diabetes_X_train, diabetes_y_train)#Train the model
diabetes_y_pred = regr.predict(diabetes_X_test)# Make predictions

print('Coefficients: \n', regr.coef_)#The coefficients
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))
plt.scatter(diabetes_X_test,diabetes_y_test,color='black')#Plot outputs
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

6.3-5 sklearn中的回归方法

线性回归实例

- 考虑全部10个特征

```python
#ch6_linReg_example1.py
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
#Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
#Split the features and targets into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

regr = linear_model.LinearRegression()#Linear regression object
regr.fit(diabetes_X_train, diabetes_y_train)#Train the model
diabetes_y_pred = regr.predict(diabetes_X_test)# Make predictions

print("训练集特征个数: ",diabetes_X_train.shape[1])
print('Coefficients: \n', regr.coef_)#The coefficients
print("Intercept:\n",regr.intercept_)
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))
print("Actual labels of the test set:\n",diabetes_y_test)
```

```
训练集特征个数:  10
Coefficients:
 [ 3.03499549e-01 -2.37639315e+02
5.10530605e+02  3.27736980e+02
 -8.14131709e+02  4.92814588e+02
1.02848452e+02  1.84606489e+02
  7.43519617e+02  7.60951722e+01]
Intercept:
 152.76430691633442
Mean squared error: 2004.57
Coefficient of determination: 0.59
Actual labels of the test set:
 [233.  91. 111. 152. 120.  67. 310.
94. 183.  66. 173.  72.  49.  64.
  48. 178. 104. 132. 220.  57.]
```

6.3-5 sklearn中的回归方法

sklearn回归方法一览表

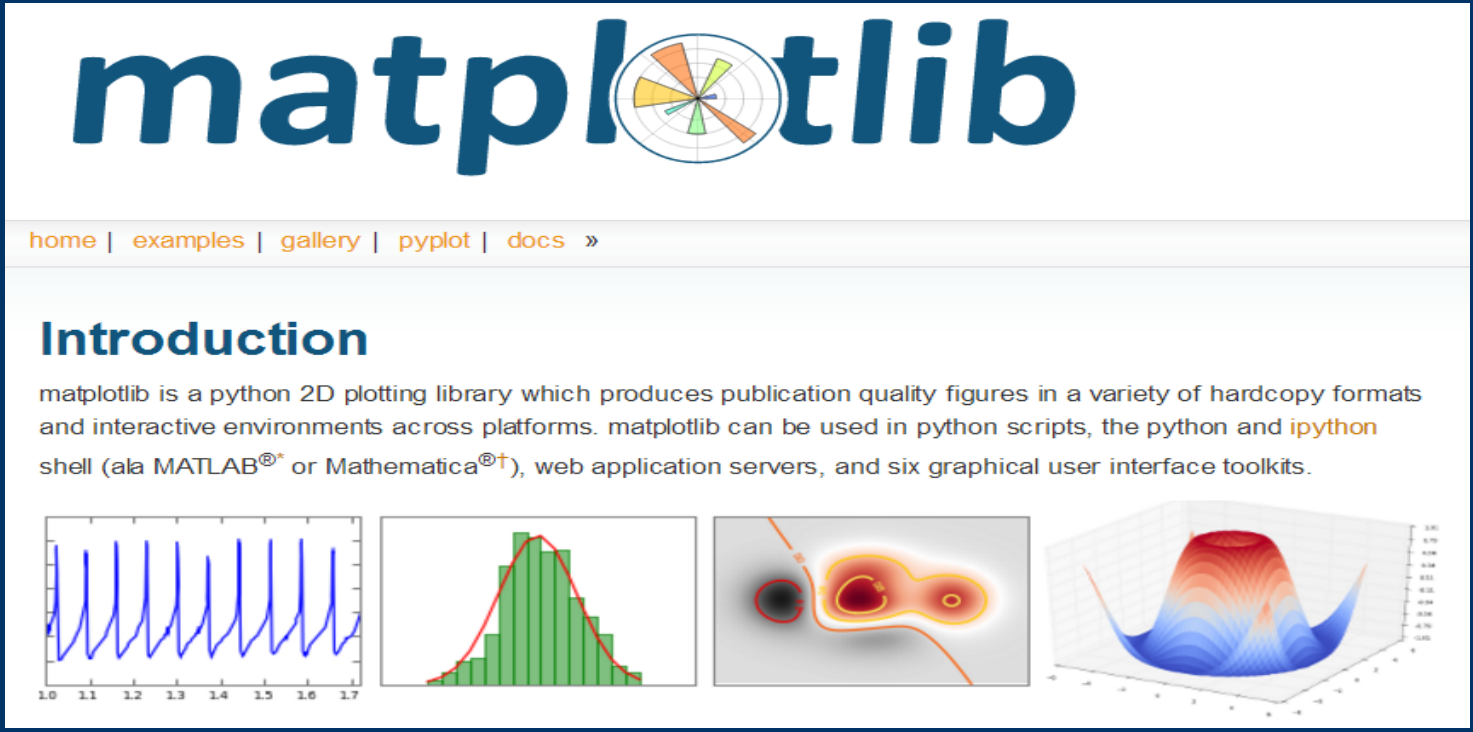
```
#ch6_sklearn_regressors.py
from sklearn import linear_model #线性回归
model_LinearRegression = linear_model.LinearRegression()
from sklearn import linear_model #岭回归
model_Ridge = linear_model.Ridge()

from sklearn import svm #SVM回归
model_SVR = svm.SVR()

from sklearn import neighbors #KNN回归
model_KNeighborsRegressor = neighbors.KNeighborsRegressor()

from sklearn import tree #决策树回归
model_DecisionTreeRegressor = tree.DecisionTreeRegressor()

from sklearn import ensemble # 随机森林回归，用20个决策树
model_RandomForestRegressor = ensemble.RandomForestRegressor(n_estimators=20)

from sklearn import ensemble #GBRT回归，用100个决策树
model_GradientBoostingRegressor = ensemble.GradientBoostingRegressor(n_estimators=100)

from sklearn.tree import ExtraTreeRegressor #ExtraTree极端随机树回归
model_ExtraTreeRegressor = ExtraTreeRegressor()

from sklearn.ensemble import BaggingRegressor #Bagging回归
model_BaggingRegressor = BaggingRegressor()
```

6.3-6 模型的保存与应用

- Pickle模块应用

6.4 绘图库Matplotlib

- matplotlib 是python最著名的绘图库，它提供了一整套和matlab相似的命令API，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入GUI应用程序中。
- 文档相当完备，见官网：http://matplotlib.org/

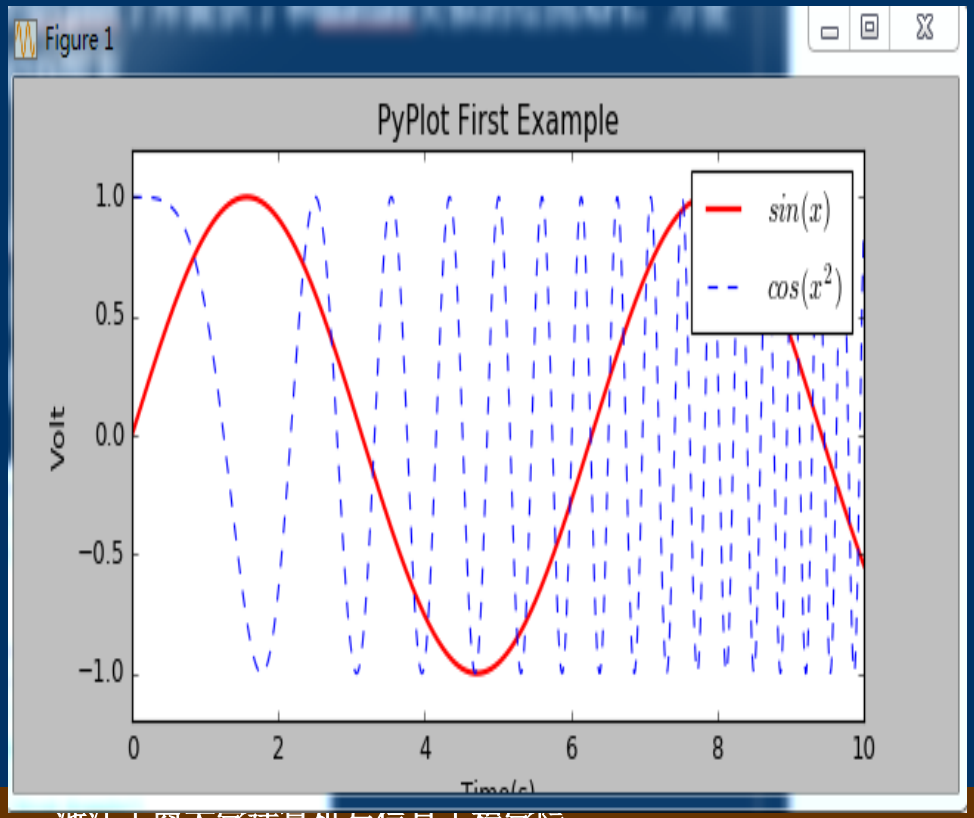


6.4-1 快速绘图

快速绘图

- matplotlib的pyplot子库提供了和matlab类似的绘图API，方便用户快速绘制2D图表。
- matplotlib还提供pylab模块，包括许多numpy和pyplot中常用函数，方便快速进行计算绘图，可用于IPython中的快速交互式使用。
- 实例：

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 1000)
y = np.sin(x)
z = np.cos(x**2)
plt.figure(figsize=(8,4))
plt.plot(x,y,label="$sin(x)$",color="red",linewidth=2)
plt.plot(x,z,"b--",label="$cos(x^2)$")
plt.xlabel("Time(s)")
plt.ylabel("Volt")
plt.title("PyPlot First Example")
plt.ylim(-1.2,1.2)
plt.legend()
plt.show()
```



6.4-1 快速绘图

- plot函数的调用方式很灵活，用关键字参数指定各种属性：
  - label : 给所绘制的曲线一个名字，此名字在图示(legend)中显示。只要在字符串前后添加"$"符号，matplotlib就会使用其内嵌的latex引擎绘制的数学公式。
  - color : 指定曲线的颜色
  - linewidth : 指定曲线的宽度
  - xlabel / ylabel : 设置X轴/Y轴的文字
  - title : 设置图表的标题
  - xlim/ylim : 设置X轴/Y轴的范围
  - legend : 显示图例

6.4-2 绘图函数

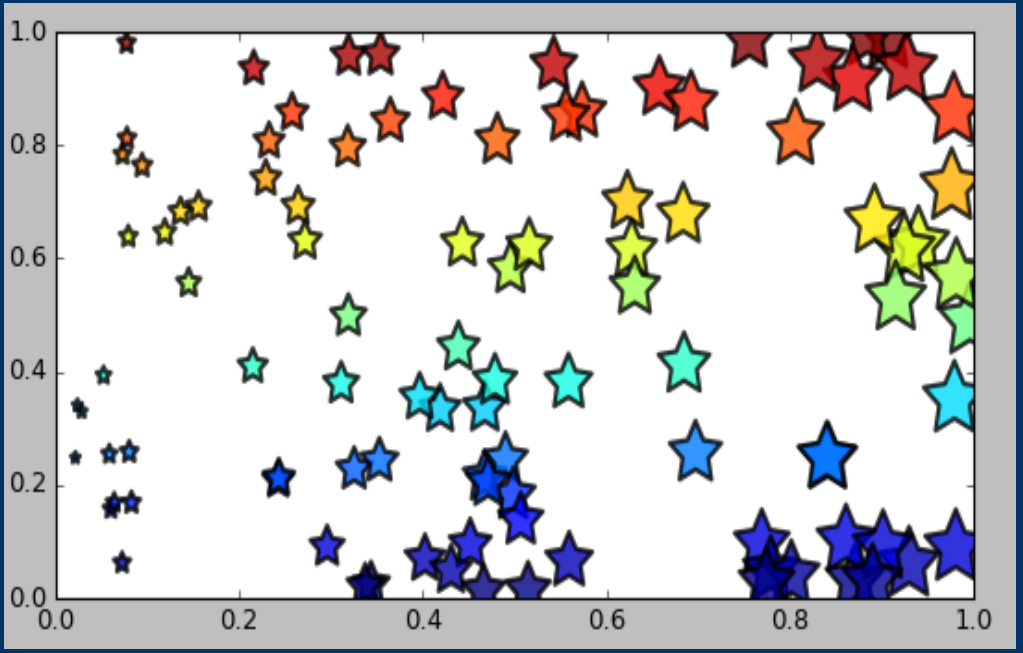
- 常见绘图函数:
  - plot( )：折线图
  - scatter( ): 散列图
  - bar( )：柱状图
  - hist( ): 直方图
  - pie( ): 饼图
  - contour( )：等值线图

6.4-2 绘图函数

散列图

- scatter()的前两个参数是数组，分别指定每个点的X轴和Y轴的坐标。s参数指定点的大 小，值和点的面积成正比。c参数指定每个点的颜色，可以是数值或数组。marker参数设置点的形状。通过alpha参数设置点的透明度，通过lw参数设置线宽。 facecolors参数为“none”时，表示散列点没有填充色。

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
x = np.random.random(100)
y = np.random.random(100)
plt.scatter(x, y, s=x*1000, c=y,
            marker=(5, 1), alpha=0.8,
            lw=2, facecolors="none")
plt.xlim(0,1)
plt.ylim(0,1)
plt.show()
```

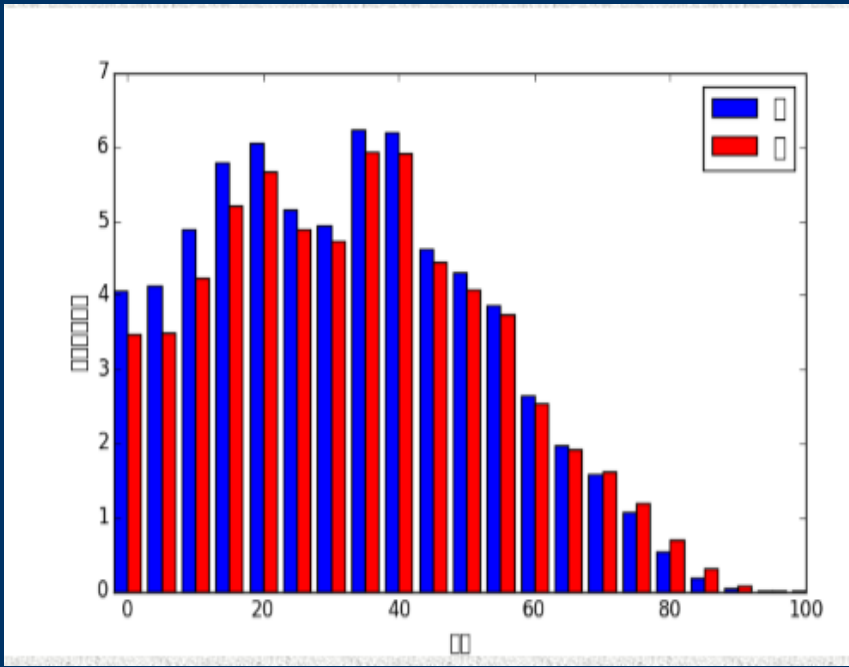


6.4-2 绘图函数

柱状图

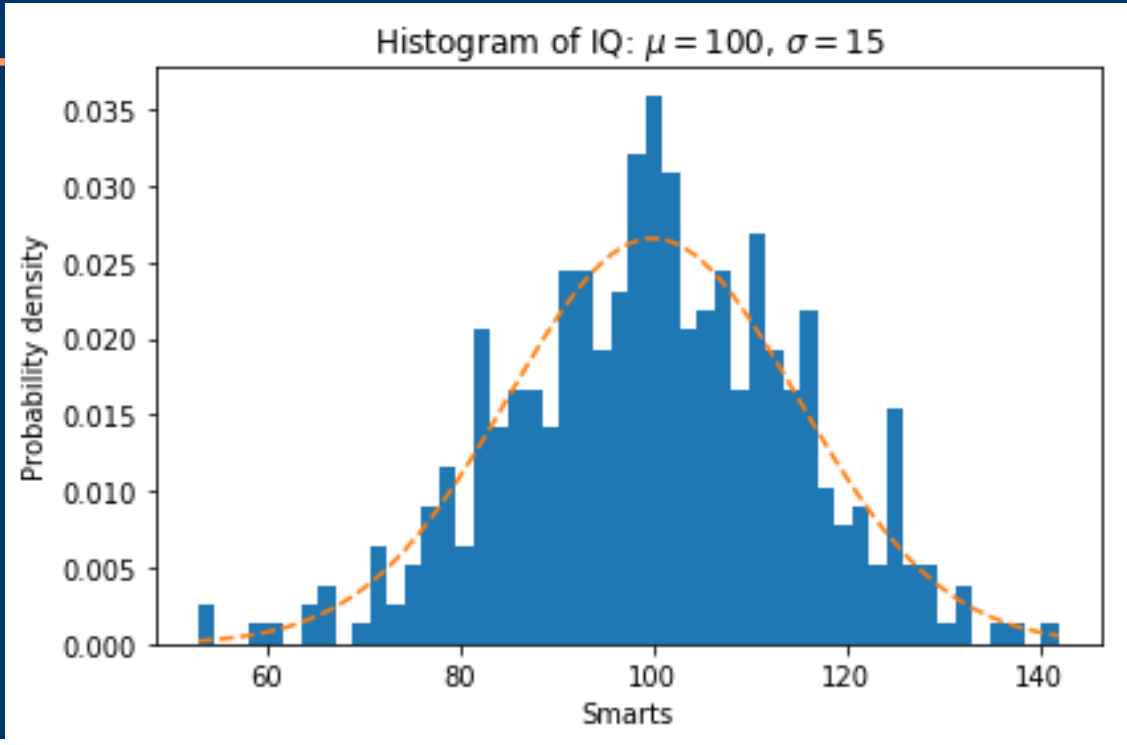
- 柱状图用其每根柱子的长度表示值的大小，它们通常用来比较两组或多组值。下
- 面的程序从文件中读入中国人口的年龄分布数据,并使用柱状图比较男性和女性的年龄分布。
- 调用bar()绘制男性人口分布的柱状图。bar()的第二个参数为每根柱子的高度，第三个参数指定所有柱子的宽度。当第三个参数为序列时，可以为每根柱子指定宽度。

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

data = np.loadtxt("china_population.txt")
width = (data[1,0] - data[0,0])*0.4

plt.figure(figsize=(8,5))
plt.bar(data[:,0]-width, data[:,1]/1e7, width, color="b", label=u"男")
plt.bar(data[:,0], data[:,2]/1e7, width, color="r", label=u"女")
plt.xlim(-width, 100)
plt.xlabel(u"年龄")
plt.ylabel(u"人口（千万）")
plt.legend()
plt.show()
```



6.4-2 绘图函数

- 直方图：**hist( )**



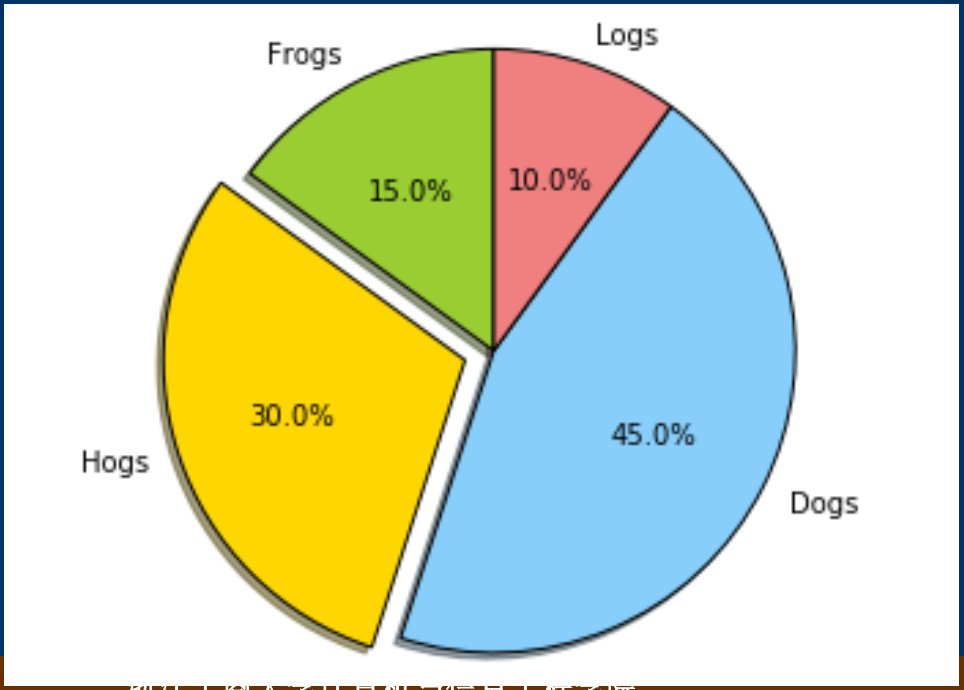

```python
#ch6_hist.py
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(19680801)
mu = 100  # mean of distribution
sigma = 15  # standard deviation of distribution
x = mu + sigma * np.random.randn(437)# example data
num_bins = 50
fig, ax = plt.subplots()

# the histogram of the data
n, bins, patches = ax.hist(x, num_bins, density=True)
# add a 'best fit' line
a=1/(np.sqrt(2*np.pi)*sigma)
y = a*np.exp(-0.5*(1/sigma*(bins-mu))**2)
ax.plot(bins, y, '--')
ax.set_xlabel('Smarts')
ax.set_ylabel('Probability density')
ax.set_title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')

fig.tight_layout()
plt.show()
```

6.4-2 绘图函数

- 饼图：pie()

```
#ch6_pie_plot.py
import matplotlib.pyplot as plt

# The slices will be ordered and plotted counter-clockwise.
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
colors = ['yellowgreen','gold','lightskyblue','lightcoral']
explode = (0,0.1,0,0)#only "explode" the 2nd slice('Hogs')
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=90)
#Set aspect ratio to be equal so that pie is drawn as a circle.
plt.axis('equal')
plt.show()
```

参数explode表示将某部分暴突出来



6.4-3 绘制多个子图

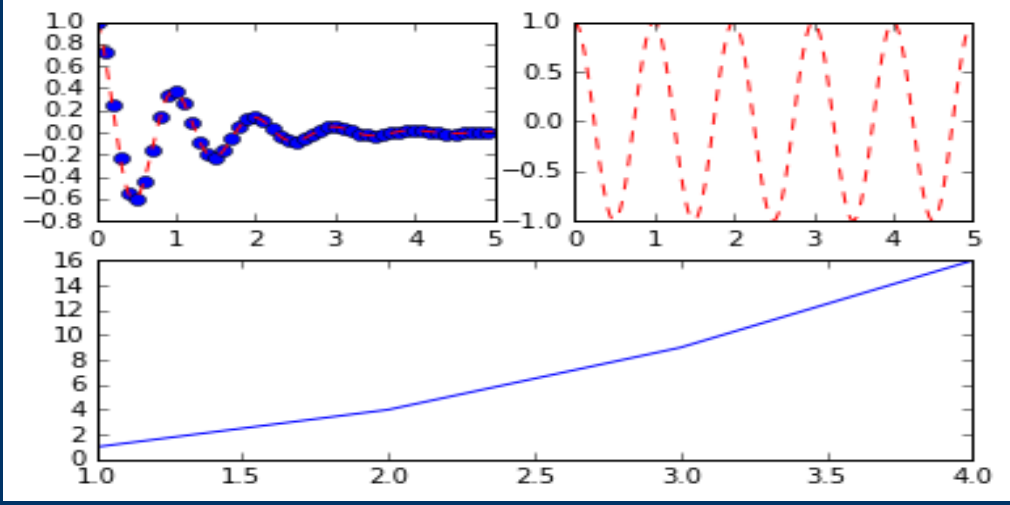
subplot绘制多个子图：

- 在matplotlib下，一个Figure对象可以包含多个子图（Axes），可以使用subplot()快速绘制，格式如下：

  subplot(numRows, numCols, plotNum)

- 图表的整个绘图区域被分成numRows行和numCols列，plotNum参数指定创建的Axes对象所在的区域。
  - ✓如果numRows ＝ 3，numCols ＝ 2，那整个绘制图表样式为3X2的图片区域。这时，当plotNum ＝ 1时，第一行第一列的子图

```python
#ch6_multiple_subplots.py
import matplotlib.pyplot as plt
import numpy as np
def f(t):
    return np.exp(-t) * np.cos(2 * np.pi * t)
t1 = np.arange(0, 5, 0.1)
t2 = np.arange(0, 5, 0.02)
plt.figure(12)
plt.subplot(221)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'r--')

plt.subplot(222)
plt.plot(t2, np.cos(2 * np.pi * t2), 'r--')

plt.subplot(212)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])

plt.show()
```



第7章 文本分析

提纲

- 7.1 文本分析任务
- 7.2 Python文本分析库

7.1 文本分析任务

文本分析意义

- 根据估算，各类组织(包括企业、政府)拥有的数据里，80%是非结构化的数据，其中大部分是文本的形式。
- 非结构化的文本数据，包括电子邮件、博客、微博、客户反馈、医疗记录、合同文本等。
- 这些文本里，隐藏着潜在的价值。只有通过适当的分析方法，才能从中提取有价值的信息。

7.1 文本分析任务

- 文本分析或自然语言处理NLP的主要任务：
  - 文本索引与检索(Indexing and Search)
  - 文本分类(Text Classification)
  - 文本摘要(Document Summarization)
  - 文本信息抽取(Text Information Extraction)
  - 情感分析(Sentiment Analysis)

7.1-1 句子预处理

- 句子切分与分词
  - 进行文本分析前，首先需要把文本切分成单个的句子。完成该功能的软件，称为语句切分器。
  - 接着，需要对句子进行分词，完成该功能的软件，称为分词器。

- 词性标注
  - 词性是词汇最基本的语法属性，词性标注就是在给定句子中判定每个词的词性并加以标注的过程，比如名词(Noun)、动词(Verb)、形容词(Adjective)等。

- 诸如Jieba、NLTK等自然语言处理工具可以进行句子的切分和分词。

7.1-2 文本分类

- 文本分类的定义
  - 文本分类是把文档集合中的每个文档，划分到一个预先定义的主题类别。
    - ✓判断Email是正常邮件和垃圾邮件，二分类问题。
    - ✓新闻主题分类：判断新闻属于哪个类别，如财经、体育、娱乐等
    - ✓自动问答系统中的问句分类：问句类型
    - ✓让AI当法官：基于案件事实描述文本的罚金等级分类（多分类）
  - 文本分类系统的一般步骤：
    - ✓1.预处理输入文本
    - ✓2.特征提取，得到文本表示
    - ✓3.分类器分类
  - 对于分类器的设计，有包括k最近邻、朴素贝叶斯、支持向量机和决策树在内的许多算法。

7.1-3 信息抽取

信息抽取(Information Extraction)

- 简称IE，即从自然语言文本中，抽取出特定的事件或事实信息，帮助我们将海量内容自动分类、提取和重构。这些信息通常包括指定类型的实体(entity)、关系(relation)、事件(event) 。
  - ✓ 从新闻中抽取时间、地点、关键人物
  - ✓ 从技术文档中抽取产品名称、开发时间、性能指标等
- 从自然语言中抽取用户感兴趣的事实信息，无论是在知识图谱、信息检索、问答系统还是在情感分析、文本挖掘中，信息抽取都有广泛应用。

信息抽取主要包含三大子任务

- 关系抽取：通常我们说的三元组(triple)抽取，主要用于抽取实体间的关系。
- 实体抽取与链指：也就是命名实体识别(named entity recognition, NER)。
- 事件抽取：相当于一种多元关系的抽取。

7.1-3 信息抽取

关系信息抽取常用数据集

- **ACE关系抽取任务数据集：**ACE2005关系抽取数据集包含599篇与新闻和邮件相关的文档，其数据集内包含7大类25小类关系。

- SemEval2010 Task 8数据集：该数据集包含 9 种关系类型，分别是Compoent-Whole、Instrument-Agency、Member-Collection、Cause-Effect、Entity-Destination、Content-Container、Message-Topic、Product-Producer和Entity-Origin。考虑到实体之间关系的方向以及不属于前面9种关系的Other关系，共生成19类实体关系。其中训练数据 8000个，测试数据2717个。

- NYT2010数据集：是Riedel等人在2010年将Freebase知识库中的知识“三元组”对齐到“纽约时报”新闻中得到的训练数据。该数据集中，数据的单位是句包，一个句包由包含该实体对的若干句子构成。其中，训练数据集从《纽约时报》2005-2006年语料库中获取，测试集从2007年语料库中获取。

7.1-4 情感分析

情感分析的定义

- 情感分析，是NLP领域的主要问题之一。主要研究人们针对实体（包括产品、服务、组织、机构、事件、话题等）表达出来的主观观点、情感、评价、态度和情绪。

情感分析任务

- 包括情感分类、观点抽取、观点问答以及观点摘要等。其中情感分类是情感分析领域研究最广的问题。

- 情感分类是指根据文本所表达的含义和情感信息将文本划分成褒扬、贬义、中性或具体的类型。例如，有三句话“品牌M汽车的油耗很高”、“品牌J汽车的性价比相当的高”、“品牌Z汽车有1m多高”。
  - ✓第一句表达了贬义的情感倾向
  - ✓第二句表达了褒扬的情感倾向
  - ✓第三句则表述了一个事实，没有情感倾向。

7.1-5 其他文本分析任务

- 文本摘要
  - 指通过各种技术，对文本或者是文本集合，抽取、总结或精炼其中的要点信息，用以概括和展示原始文本(集合)的主要内容或大意
- 机器翻译
  - 机器翻译是将一种语言的文本，转换为另一种语言的文本。
- 话题检测与跟踪(Topic Detection and Tracking, TDT)
  - 指从新闻等数据流中，识别不同的事件，并且对其进行跟踪的自动化技术。在话题检测和跟踪任务中，“话题”也称为“事件”。
- 问答系统
  - 问答系统是信息检索系统的一种高级形式。它能用准确、简洁的自然语言回答用户用自然语言提出的问题。
- ……

7.2 Python文本分析库

Jieba库

- Jieba是一款优秀的 Python 第三方中文分词库，可以对中文文本进行分词、词性标注、关键词抽取等功能，并支持自定义词典。

NLTK库

- Natural Language Toolkit，自然语言处理工具包。大概是NLP领域最为著名的一个Python库。

其它NLP库

- HanLP：Han Language Processing，汉语言处理包。功能包括：中文分词、词性标注、命名实体识别、依存句法分析、关键词提取、新词发现、短语提取、自动摘要、文本分类、拼音简繁。地址：https://github.com/hankcs/HanLP
- Stanford NLP：提供一系列自然语言分析的工具，支持中文、英文、阿拉伯语、法语、德语、西班牙语等多种语言。地址: https://stanfordnlp.github.io/CoreNLP/
- Gensim：可以用来从文档中自动提取语义主题，含众多非监督学习算法：TF/IDF，潜在语义分析(Latent Semantic Analysis，LSA) 等。还支持Word2Vec，Doc2Vec等模型。 地址: https://radimrehurek.com/gensim/

7.2-1 Jieba分词库

Jieba库的安装

- Jieba库为优秀的中文分词第三方库，没有被集成在Anaconda中，需额外安装，安装方式：**pip install jieba**

Jieba分词

- Jieba分词有三种不同的分词模式：精确模式、全模式和搜索引擎模式。
  - ✓精确模式是最常用的分词方法。
  - ✓全模式将句子中所有可能的词都列举出来。
  - ✓搜索引擎模式则适用于搜索引擎使用。

7.2-1 Jieba分词库

常用分词方法

函数	描述
jieba.cut(s)	精确模式，返回一个可迭代的数据类型
jieba.cut(s,cut_all=True)	全模式，输出文本s中所有可能单词
jieba.cut_for_search(s)	搜索引擎模式，适合搜索引擎建立索引的分词结果
jieba.lcut(s)	精确模式，返回一个 列表类型，建议使用
jieba.lcut(s,cut_all=True)	全模式，返回一个列表类型，建议使用
jieba.lcut_for_search(s)	搜索引擎模式，返回一个列表类型，建议使用
jieba.add_word(w)	向分词词典中增加新词w
jieba.del_word(w)	从分词词典中删除词汇w

7.2-1 Jieba分词库

Jieba分词例子

```python
#ch7: jieba_cut_pos_KWextract.py
# ============ 1.分词 =======================
import jieba
text = "文本分析涵盖了许多研究方向。"
# 精确模式
seg1 = jieba.cut(text, cut_all=False) #返回迭代器
print("seg1:",seg1)
print("精确模式:","/ ".join(seg1))
seg2 = jieba.lcut(text, cut_all=False)#返回列表
print("精确模式list:",seg2)

# 全模式：将句子中所有可能的词都列举出来
seg3 = jieba.cut(text, cut_all=True)
print("seg3:",seg3)
print("全模式:","/ ".join(seg3))
seg4 = jieba.lcut(text, cut_all=True)
print("全模式list:",seg4)
```

```
seg1: <generator object Tokenizer.cut at 0x0000021C08DD1678>
精确模式: 文本/ 分析/ 涵盖/ 了/ 许多/ 研究/ 方向/ 。
精确模式list: ['文本', '分析', '涵盖', '了', '许多', '研究', '方向', '。']
seg3: <generator object Tokenizer.cut at 0x0000021C03BD4888>
全模式: 文本/ 本分/ 分析/ 涵盖/ 了/ 许多/ 研究/ 方向/ /
全模式list: ['文本', '本分', '分析', '涵盖', '了', '许多', '研究', '方向', '', '']
```

7.2-1 Jieba分词库

词性标注

- Jieba分词采用基于统计模型的标注方法，使用北大计算所的词性标注集。

例子

```python
# ============= 2.词性标注 ================
import jieba.posseg as pseg

text = "文本分析涵盖了许多研究方向。"
seg = pseg.cut(text)
print("seg:",seg)
print([word for word in seg])

```

```
seg: <generator object cut at 0x0000021C03BD4888>
[pair('文本', 'n'), pair('分析', 'vn'), pair('涵盖', 'v'),
pair('了', 'ul'), pair('许多', 'm'), pair('研究', 'vn'),
pair('方向', 'n'), pair('。', 'x')]
```

7.2-1 Jieba分词库

关键词抽取

- Jieba中与关键词抽取相关的方法有两个，分别是基于 **TF-IDF** 算法的关键词抽取以及基于 **TextRank** 算法的关键词抽取。

例子

```python
# ============= 3.关键词抽取 ==========
import jieba.analyse as an

# 基于TF-IDF算法
result = an.extract_tags(text, topK=3)  #指定抽取topK个关键词
print(result)

# 基于TextRank算法
result = an.textrank(text, topK=3)
print(result)
```

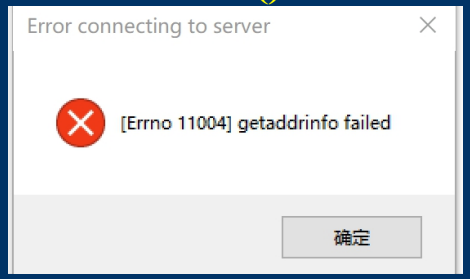
```
['文本', '涵盖', '方向']
['涵盖', '研究', '分析']
```

7.2-2 NLP工具包NLTK

- **NLTK** (Natural Language Toolkit)
  - 包含分词、词干提取、语法树分析、词性标注、命名实体识别、短语识别以及文本分类等文本处理功能。主要应用于英文数据，历史悠久。

  - 语料库安装：Anaconda已集成nltk。进行“import nltk”测试，是否会报错？
    - ✓ 然而，nltk需配置大量语料库，尝试nltk.download()下载至本地，但很可能报错！

```python
import nltk
nltk.download()
```



    - ✓ 若无法通过nltk.download()自动下载，可尝试从github下载至本地： https://github.com/nltk/nltk_data/tree/gh-pages 获得nltk_data-gh-pages.zip 文件，解压后将文件中packages文件夹的全部子目录放置在nltk搜索的任意一个路径下，如‘E:\\nltk_data’ (若还不存在，则创建)：

```
Searched in:
  - 'C:\\Users\\zero/nltk_data'
  - 'C:\\nltk_data'
  - 'D:\\nltk_data'
  - 'E:\\nltk_data'
  - 'C:\\ProgramData\\Anaconda3\\nltk_data'
  - 'C:\\ProgramData\\Anaconda3\\share\\nltk_data'
  - 'C:\\ProgramData\\Anaconda3\\lib\\nltk_data'
  - 'C:\\Users\\zero\\AppData\\Roaming\\nltk_data'
```

7.2-2 NLP工具包NLTK

Case 1: 分句与分词

```python
#ch7: nltk_cases.py
# ========== 1.分句与分词 =================
# 分句
from nltk.tokenize import sent_tokenize
text = "Hello Adam, how are you? \
          I hope everything is going well.\
          Today is a good day, see you dude."
print("分句:", sent_tokenize(text))
# 分词
from nltk.tokenize import word_tokenize
text = "Hello Mr. Adam, how are you?\
          I hope everything is going well.\
          Today is a good day, see you dude."
print("分词:", word_tokenize(text))
```

运行结果:

```
分句: ['Hello Adam, how are you?', 'I hope everything is going well.', 'Today is a good day, see you dude.']
分词: ['Hello', 'Mr.', 'Adam', ',', 'how', 'are', 'you', '?', 'I', 'hope', 'everything', 'is', 'going', 'well', '.', 'Today', 'is', 'a', 'good', 'day', ',', 'see', 'you', 'dude', '.']
```

- 注：运行以上程序可能还会出现右图所示的报错提醒。
  解决办法: 解压语料库文件夹“nltk_data\tokenizers”中的punkt.zip文件即可。

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\nltk\data.py", line 675, in find
    raise LookupError(resource_not_found)

LookupError:
**********************************************************************
  Resource punkt not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt')
```

7.2-2 NLP工具包NLTK

Case 2: 词性标注

- 进行词性标注，需要使用**nltk.pos_tag(tokens)**函数，其中**tokens**是句子**分词后的结果**。词性标注是句子级的标注。

```
19 # ============ 2.词性标注 ==================
20 import nltk
21 text=nltk.word_tokenize('what does the fox say')
22 print("词性标注:",nltk.pos_tag(text))
```

```
词性标注: [('what', 'WDT'), ('does', 'VBZ'), ('the', 'DT'),
('fox', 'NNS'), ('say', 'VBP')]
```

7.2-2 NLP工具包NLTK

- Case 3: 命名实体识别(Named Entity Recognition，简称NER)
  - NER任务的目标是识别出待处理文本中指定的实体类别：
    - ✓三大类（实体类、时间类和数字类）
    - ✓七小类（人名、机构名、地名、时间、日期、货币和百分比）命名实体。
    - ✓九小类：
      - PERSON: 人名，如President Obama
      - ORGANIZATION：机构名，如WHO、ISRO
      - LOCATION：地点，如Murray River
      - DATE: 日期，如June、2008-06-29
      - TIME：时间，如12:30:00 AM
      - MONEY：货币，如Twenty dollars，100 GBP
      - PERCENT：百分比，如20%， twenty pct.
      - FACILITY：设施，如Washington Monument
      - GPE: 地缘政治实体，如 South East Asia, North America

7.2-2 NLP工具包NLTK

Case 3: 命名实体识别(Named Entity Recognition，简称NER)

- 通过调用nltk.ne_chunk函数，可以进行命名实体识别。需要把参数tags，即句子词性标注的结果，传递给ne_chunk函数。

```
# ============ 3.命名实体识别 =======================
#sentence = "I am very excited about the next generation of Apple products."
sentence = "Edison went to Tsinghua University today."
tokens = nltk.word_tokenize(sentence)  # 分词
tags = nltk.pos_tag(tokens)  # 词性标注
ners = nltk.ne_chunk(tags)  # 利用词性标注结果，进行NER,返回一种树结构
print(ners)
for i,tree in enumerate(ners.subtrees()):
    if tree.label()=="S":
        continue
    print("NE%d:"%(i),tree)
```

```
(S
  (PERSON Edison/NNP)
  went/VBD
  to/TO
  (ORGANIZATION Tsinghua/NNP University/NNP)
  today/NN
  ./.)
NE1: (PERSON Edison/NNP)
NE2: (ORGANIZATION Tsinghua/NNP University/NNP)
```

7.2-2 NLP工具包NLTK

Case 3: 命名实体识别（多个句子情形）

```python
#ch7: ner.py
import nltk
import pandas as pd
text = """
FIFA was founded in 1904 to oversee international compe
associations of Belgium, Denmark, France, Germany, the
Sweden, and Switzerland. Headquartered in Zürich, its r
211 national associations. Member countries must each a
the six regional confederations into which the world is
Europe, North & Central America and the Caribbean, Ocea
"""
sentences = nltk.sent_tokenize(text) # 分句
tokenized_sentences=[nltk.word_tokenize(sentence) for s
tagged_sentences=[nltk.pos_tag(sentence) for sentence i
ne_chunked_sents=[nltk.ne_chunk(tagged) for tagged in t
named_entities = []
for ne_taged_sentence in ne_chunked_sents:
    for tagged_tree in ne_taged_sentence:
        if hasattr(tagged_tree, 'label'):
            name = tagged_tree[0][0]
            C = tagged_tree.label()
            named_entities.append((name,C))
entity_frame = pd.DataFrame(named_entities, columns=['Entity Name', 'Entity Type'])
print(entity_frame)
```

```
     Entity Name   Entity Type
0           FIFA  ORGANIZATION
1        Belgium           GPE
2        Denmark           GPE
3         France           GPE
4        Germany           GPE
5    Netherlands           GPE
6          Spain           GPE
7         Sweden           GPE
8    Switzerland           GPE
9         Zürich           GPE
10        Africa        PERSON
11          Asia           GPE
12        Europe           GPE
13         North           GPE
14       Central  ORGANIZATION
15     Caribbean      LOCATION
16       Oceania           GPE
17         South           GPE
```

7.2-2 NLP工具包NLTK

Case 4: 去除停用词(Stopwords)

```python
# ============= 4.去除停用词 =======================
from nltk.corpus import stopwords

tokens = ['my', 'dog', 'has', 'flea', 'problems', 'help', 'please',
          'maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid',
          'my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him']
clean_tokens = tokens[:]
stwords = stopwords.words('english')
for token in tokens:
    if token in stwords:
        clean_tokens.remove(token)
print("去除停用词后:",clean_tokens)
print("全部停用词:",stwords)
```

效率，在处理自
字或词，这些字或

```
去除停用词后: ['dog', 'flea', 'problems', 'help', 'please', 'maybe', 'take', 'dog', 'park', 'stupid',
'dalmation', 'cute', 'I', 'love']
全部停用词: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she',
"she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did',
'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at',
'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again',
'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',
'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',
'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

7.2-2 NLP工具包NLTK

Case 5: 情感分析

- 使用NLTK，可以对文本进行情感分析，确定其情感倾向。

```python
# ============= 5.情感分析 =======================
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sentences = ['this is a good book',  # 正面情感
             'this is a awesome book']  # 正面情感
sid = SentimentIntensityAnalyzer()  # 创建分类器
for sentence in sentences:
    print(sentence)
    ss = sid.polarity_scores(sentence)  # 对句子进行情感分析
    print("情感得分: ", ss)  # 其中，compound表示复杂程度，neu表示中性，
                             #      neg表示负面情绪，pos表示正面情绪
    print('-----------------------------')
```

```
this is a good book
情感得分:  {'neg': 0.0, 'neu': 0.508, 'pos': 0.492,
'compound': 0.4404}
-----------------------------
this is a awesome book
情感得分:  {'neg': 0.0, 'neu': 0.423, 'pos': 0.577,
'compound': 0.6249}
-----------------------------
```

第8章 正则表达式

提纲

- 8.1 正则表达式概述
- 8.2 特殊符号与字符
- 8.3 正则表达式模块re

8.1 正则表达式概述

- 在信息化时代的今天，计算机需要做大量的文本与数据处理等工作。
- 我们可能不清楚这些文本或数据的具体内容，因此以某种可被计算机自动识别和处理的模式表达出来是非常有用的。
- 正则表达式提供了这样的模式，是一个特殊的字符序列，帮助你方便的检查一个字符串是否与某种模式匹配。
- 在Python中内置了一个re模块以支持正则表达式。
- 正则表达式有两种基本的操作，分别是匹配和替换。
  - 匹配就是在一个文本字符串中搜索匹配一特殊表达式
  - 替换就是在一个字符串中查找并替换匹配一特殊表达式的字符串

8.2 特殊符号和字符

基本字符

字符	描述
text	匹配text字符串
.	匹配除换行符之外的任意一个单个字符
^	匹配一个字符串的开头
$	匹配一个字符串的末尾

8.2 特殊符号和字符

- 匹配限定符
  - 用来约束匹配的次数

限定符	描述
*	重复匹配前表达式零次或多次
+	重复匹配前表达式一次或多次
?	重复匹配前表达式零次或一次
+或*后跟?	表示非贪婪匹配，即尽可能少的匹配，如*?重复任意次，但尽可能少重复；
.*?	表示匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。
{m}	精确重复匹配前表达式m次
{m,}	至少重复匹配前表达式m次
{m,n}	至少重复匹配前表达式m次，至多重复匹配前表达式n次

8.2 特殊符号和字符

组和运算符

组	描述
[...]	匹配集合内的字符，如[a-z],[1-9]或[,./;']
[^...]	匹配除集合外的所有字符，相当于取反操作
A|B	匹配表达式A或B，相当于OR操作
(...)	表达式分组，每对括号为一组，如([a-b]+)([A-Z]+)([1-9]+)
(?P<name>...)	分组的命名模式，取此分组的内容时可以使用索引，也可以使用name

8.2 特殊符号和字符

例如\b匹配字符边界，food\b匹配"food"、"zoofood"，而和"foodies"不匹配。
又譬如，\bthe\b仅匹配单词the

- 特殊字符序列

字符	描述
\A	只匹配字符串的开始
\b	匹配一个单词边界
\B	匹配一个单词的非边界
\d	匹配任意十进制数字字符，等价于r'[0-9]'
\D	匹配任意非十进制数字字符，等价于r'[^0-9]'
\s	匹配任意空格字符（空格符、tab制表符、换行符、回车、换页符、垂直线符号）
\S	匹配任意非空格字符
\w	匹配任意字母数字及下划线
\W	匹配任意非字母数字及下划线
\Z	仅匹配字符串的尾部
\\	匹配反斜线字符

8.2 特殊符号和字符

正则表达式例子

正则表达式模式	匹配的字符串
b[aeiu]t	bat,bet,bit,but
[cr][23]	一个包含2个字符的字符串：第一个是“r”或“c”,第二个是“2”或“3”，例如：c2、r3、r2等

正则表达式模式	匹配的字符串
z.[0-9]	字符“z”，后面跟任意一个字符，然后是一个十进制数字
[r-u][env-y][us]	“r”到“u”中的任意一个字符，后面跟的是“e”到“y”中的任意一个字符，在后面是字符“u”或“s”
[^aeiou]	一个非元音字符
[^\t\n]	除TAB制表符和换行符以外的任意一个字符

8.2 特殊符号和字符

正则表达式例子

正则表达式模式	匹配的字符串
[dn]ot?	字符“d”或“o”,后面是一个“o”，最后是最多一个字符“t”，即do、no、dot、not
0?[1-9]	1~9中的任意一位数字，前面可能还有一个“0”
[0-9]{15,16}	15位或16位数字表示
[KQRBNP][a-h][1-8]-[a-h][1-8]	表示国际象棋合法的棋盘移动。即“K”、“Q”、“R”、“B”、“N”或“P”等字母后面加上两个用连字符连在一起的“a1”到“h8”之间的棋盘坐标

正则表达式模式	匹配的字符串
\w+-\d+	一个由字母或数字组成的字符串和至少一个数字，两部分中间由连字符连接
[A-Za-z]\w*	第一个字符是字母，其余字符是字母或数字
\d{3}-\d{3}-\d{4}	电话号码，前面带区号前缀，例如800-555-1212
\w+@\w+\.com	简单的XXX@YYY.com格式的电子邮件地址

8.3 正则表达式模块re

- re模块
  - 通过re模块，可在python中利用正则表达式对字符串进行搜索、抽取和替换操作。
  - re 模块使 Python 语言拥有全部的正则表达式功能

- 核心函数和方法
  - 模块的函数或正则表达式对象的方法compile

函数	描述
compile(pattern,flags=0)	对正则表达式pattern进行编译，flags是可选表示符，并返回一个regex对象

8.3 正则表达式模块re

- 两种调用方式
  - 利用正则表达式对象的方法
  - 直接调用模块re的函数
  - 以findall为例

```
2 #regexobj.py
3 import re
4 text="Joodie is a handsome boy. He is cool, clever.."
5 rex=re.compile('\w*oo\w*')#查找所有包含'oo'的单词
6 #1.正则表达式对象的方法findall
7 a=rex.findall(text)
8 print("正则表达式对象的方法:",a)
9 #2.模块re的函数findall
10 b=re.findall('\w*oo\w*',text)
11 print("模块re的函数:",b)
12
```

```
正则表达式对象的方法: ['Joodie', 'cool']
模块re的函数: ['Joodie', 'cool']
```

8.3 正则表达式模块re

- 核心函数和方法(续)
  - re模块的函数或regex(正则表达式)对象的方法(函数与方法同名)
  - 返回一个匹配对象或列表

函数/方法	说明
match(pattern, string, flags=0)	尝试用正则表达式模式pattern匹配字符串string,flags是可选标识符，如果匹配成功，则返回一个匹配对象；否则返回None
search(pattern, string, flags=0)	在字符串string中搜索正则模式pattern的第一次出现，flags是可选标识符，如果匹配成功，则返回一个匹配对象；否则返回None
findall(pattern, string[, flags])	在字符串string中搜索正则表达式模式pattern的所有（非重复）出现；返回一个匹配对象的列表
finditer(pattern, string[, falgs])	和findall()相同，但返回的不是列表而是迭代器；对于每个匹配，该迭代器返回一个匹配对象
split(pattern, string, max=0)	根据pattern中的分隔符把字符string分割为一个列表，返回成功匹配的列表，最多分割max次（默认是分割所有匹配的地方）
sub(pattern, repl, string, max=0)	把字符串string中所有匹配正则表达式pattern的地方替换成字符串repl，如果max的值没有给出，则对所有匹配的地方进行替换

8.3 正则表达式模块re

核心函数和方法(续)

- 匹配对象的方法

方法	描述
group(num=0)	返回全部匹配对象
groups(default=None)	返回一个包含全部匹配的子组的元组

- 常用属性或处理标志

标志	描述
re.I或re.IGNORECASE	忽略表达式的大小写来匹配文本。

8.3-1 compile()编译正则表达式

- re.compile( )可以把正则表达式编译成一个regex对象。
- 对于那些经常使用的正则表达式，这样的预编译可以提高一定的效率。
  - regex对象的一个例子：

```
2 #regexobj.py
3 import re
4 text = "JGood is a handsome boy, he is cool, clever, and so on..."
5 regex = re.compile(r'\w*oo\w*')
6 #查找所有包含'oo'的单词
7 print(regex.findall(text))
8
9
```

```
['JGood', 'cool']
```

8.3-2 match()

- 在处理正则表达式
  匹配对象。
- match()函数尝试
- 如果匹配成功，

```python
#rematch_search.py
import re
pattern=re.compile('h.llo')
result1=re.match(pattern,'hello C') #1
result2=re.match(pattern,'C hello') #2
result3=re.match(pattern,'helo') #3
result4=re.match(pattern,'hello world') #4
print(result1)
if result1: print('1匹配:',result1.group())
else: print('1匹配:失败')
if result2: print('2匹配:',result2.group())
else: print('2匹配:失败')
if result3: print('3匹配:',result3.group())
else: print('3匹配:失败')
if result4: print('4匹配:',result4.group())
else: print('4匹配:失败')
```

```
<_sre.SRE_Match object; span=(0, 5), match='hello'>
1匹配: hello
2匹配:失败
3匹配:失败
4匹配: hello
```

8.3-3 search()

- search和match工作方式一样，区别在于：
  - re.match(pattern, string, flags=0)从开头开始匹配string
  - re.search(pattern, string, flags=0)从anywhere 开始匹配string
- search和match的相同之处：
  - 均返回_sre.SRE_Match对象，如果不能匹配，返回None。

```
19 #rematch_search.py
20 import re
21 res1=re.match('[a-z]oo','seafood')
22 print('match匹配:',res1)
23 if res1: print(res1.group())
24 res2=re.search('[a-z]oo','seafood')
25 print('search匹配:',res2)
26 if res2: print(res2.group())
27
```

```
match匹配: None
search匹配: <_sre.SRE_Match object; span=(3, 6), match='foo'>
foo
```

8.3-4 匹配对象的group方法

- **group**：获取子模式(组)的匹配项

```
3  import re
4  pat=re.compile(r'www\.(.*)\.(.*)') #用()表示组1，组2,...
5  m=pat.match('www.dxy.com')
6  m0=m.group()          #默认为0，表示匹配整个字符串
7  m1=m.group(1)         #返回给定组1匹配的子字符串
8  m2=m.group(2)         #返回给定组2匹配的子字符串
9  print(m0)
10 print(m1)
11 print(m2)
12
```

```
www.dxy.com
dxy
com
```

8.3-5 findall()和finditer()

- findall( )搜索string，以列表形式返回全部能匹配的子串。
  - findall(string[,flags]) 或re.findall(pattern, string[, flags])
- finditer()搜索string，返回顺序访问每个匹配结果的迭代器。
  - finditer(string[,flags]) 或 re.finditer(pattern, string[, flags])

```python
#refind.py
import re
p=re.compile('\d+')
f1=re.findall(p,'one1two2three3four4')
print('findall:',f1)
f2=re.finditer(p,'one1two2three3four4')
print('finditer:',f2)
for m in f2: print(m,':',m.group())
```

```
findall: ['1', '2', '3', '4']
finditer: <callable_iterator object at 0x0000021098F1BE10>
<_sre.SRE_Match object; span=(3, 4), match='1'> : 1
<_sre.SRE_Match object; span=(7, 8), match='2'> : 2
<_sre.SRE_Match object; span=(13, 14), match='3'> : 3
<_sre.SRE_Match object; span=(18, 19), match='4'> : 4
```

8.3-6 sub()

- sub：substitute的缩写，表示替换
  - p.sub(repl, string[, count, flags])(p为正则表达式对象) 或 re.sub(pattern, repl, string[, count, flags])
  - 通过正则表达式，可以实现比普通字符串的replace更加强大的替换功能
  - re.sub()的各参数详解
    - ✓pattern：表示正则中的模式字符串；需要知道的是：若出现反斜杠加数字（\d），则用对应的匹配组（matched group） 去替换。
    - ✓repl：就是replacement，用来替换的字符串； repl可以是字符串，也可以是函数。
    - ✓string：即表示要被处理，要被替换的那个string字符串。
    - ✓count：指定要处理的部分；默认全部处理。

8.3-6 sub()

- sub方法的例子1： repl是字符串，不含'\d'

```
15 #resub.py
16 import re
17 r1=re.sub('a','A','abcasd') #找到a用A替换
18 print(r1)
19 pat = re.compile('a')
20 r2=pat.sub('A','abcasd')
21 print(r2)
22 pat=re.compile('(blue|white|red)')
23 r3=pat.sub('colour','blue socks and red shoes')
24 print(r3)
25
```

```
AbcAsd
AbcAsd
colour socks and colour shoes
```

8.3-6 sub()

- sub方法的例子2： repl是字符串，含'\d'

```
pat=re.compile(r'www\.(.*)\..{3}') #正则表达式
r4=pat.match('www.dxy.com').group(1)
print(r4)
r5=re.sub(r'www\.(.*)\..{3}',r'\1','hello,www.dxy.com')
print(r5)
r6=pat.sub(r'\1','hello,www.dxy.com')
print(r6)
#r'1'是第一组的意思
#匹配找到符合规则的"www.dxy.com",用组1字符串去替换整个匹配。
```

```
dxy
hello,dxy
hello,dxy
```

8.3-6 sub()

- sub方法的例子3: repl是字符串，含多个'\d'

```
36 pat=re.compile(r'(\w+) (\w+)')      #正则表达式
37 s='hello world ! hello hz !'
38 r7=pat.findall('hello world ! hello hz !')
39 print(r7)
40 r8=pat.sub(r'\2 \1',s)
41 #通过正则得到组1(hello)，组2(world)
42 #再通过sub去替换。即组1替换组2，组2替换组1，调换位置。
43 print(r8)
44
```

```
[('hello', 'world'), ('hello', 'hz')]
world hello ! hz hello !
```

8.3-6 sub()

- sub方法的例子4: repl是函数
  - 字符串中的数字+111

```
46 def _add111(matched):
47     intStr = matched.group("number") #group(name),由?P<name>定义
48     intValue = int(intStr)
49     addedValue = intValue + 111
50     addedValueStr = str(addedValue)
51     return addedValueStr
52 inputStr = "hello 123 world 456"
53 replacedStr = re.sub("(?P<number>\d+)", _add111, inputStr)
54 #模式字符串中?P<name>是给正则匹配中的组起一个名字,
55 #无论它出现在第几个括号中，都是这个名字，与圆括号的顺序无关,
56 #它出现在的那对括号内，就表示这个括号内匹配的内容就是group(name)
57 print("replacedStr=",replacedStr) #结果: hello 234 world 567
58
```

```
replacedStr= hello 234 world 567
```

8.3-6 split()

- split( )用于字符串分割：
  - split(string[, maxsplit]) 或 re.split(pattern, string[, maxsplit])
  - 按照能够匹配的子串将string分割后返回列表。maxsplit用于指定最大分割次数，若不指定，则将全部分割。

```
#resplit.py
import re
p=re.compile(r'\d+')
print(p.split('one1two2three3four4'))

```

```
['one', 'two', 'three', 'four', '']
```

  - 如果分隔符没有使用由特殊符号表示的正则表达式来匹配多个模式，那re.split()和string.split()的执行过程是一样的：

```
s="str1:str2:str3"
print(re.split(':',s))
print(s.split(':'))
```

```
['str1', 'str2', 'str3']
['str1', 'str2', 'str3']
```

8.3-6 split()

- re.split()例子：比string.split()功能更为强大

```
#resplit.py
import re
print(re.split(',','a,s,d,**asd'))#返回列表
pat = re.compile(',')
print(pat.split('a,s,d,**asd'))
print(re.split('[, *]+','a ,  s  ,d ,,**asd'))
#正则匹配: [, *]+,1个或多个逗号、空格、*等符号均可作为分隔符。
print(re.split('[, *]+','a ,  s  ,d ,,**asd',maxsplit=2))
# maxsplit 最多分割次数
```

```
['a', 's', 'd', '**asd']
['a', 's', 'd', '**asd']
['a', 's', 'd', 'asd']
['a', 's', 'd ,,**asd']
```

第9章 网络爬虫

提纲

- 9.1 网络爬虫概述
- 9.2 urllib模块
- 9.3 Beautiful Soup

9.1 网络爬虫概述

- 什么是爬虫？
  - 网络爬虫又被称为网页蜘蛛(Web spider)，网络机器人。把互联网比喻成一个蜘蛛网，那么Spider就是在网上爬来爬去的蜘蛛。
  - 爬虫过程：从网站某一个页面（通常是首页）开始，读取网页的内容，找到在网页中的其它链接地址，然后通过这些链接地址寻找下一个网页，这样一直循环下去，直到把这个网站所有的网页都抓取完为止。
- 定义
  - 网络爬虫是一个自动抓取网页的程序或脚本。
    - ✓网络爬虫的基本操作是抓取网页
    - ✓网络爬虫是搜索引擎的重要组成部分。

9.1-1统一资源定位符URL

- 如何才能随心所欲地获得自己想要的页面？
  - 先从URL开始。

- 浏览网页
  - 抓取网页的过程其实和读者平时使用IE、chrome等浏览器浏览网页的道理是一样的。
    - ✓比如，浏览器的地址栏中输入www.baidu.com 这个地址
    - ✓打开网页的过程其实就是浏览器作为一个浏览的“客户端”，向服务器端发送了 一次请求，把服务器端的文件“抓”到本地，再进行解释、展现。
    - ✓HTML是一种标记语言，用标签标记内容并加以解析和区分。
    - ✓浏览器的功能是将获取到的HTML代码进行解析，然后将原始的代码转变成我们直接看到的网站页面。

9.1-1统一资源定位符URL

- 统一资源定位符：Uniform Resource Locator
  - 通俗地说，URL是Internet上描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上。
  - 采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。
- URL的格式由三部分组成：
  - 1. 协议(或称为服务方式)
  - 2. 存有该资源的主机IP地址(有时也包括端口号)
  - 3. 主机资源的具体地址，如目录和文件名等
    - ✓第1部分和第2部分用“://”符号隔开，
    - ✓第2部分和第3部分用“/”符号隔开。

9.1-1统一资源定位符URL

- URL例子
  - http协议

    http://www.zjgsu.edu.cn/Channel_1/index.html

    ✓1. 使用超级文本传输协议HTTP，提供超级文本信息服务的资源。

    ✓2. 计算机域名为www.zjgsu.edu.cn

    ✓3. 超级文本文件(文件类型为.html)是在目录 / Channel_1下的index.html
  - ftp的URL

    ftp://test:test@192.168.0.1:21/profile

    ✓1. ftp协议，用户名和密码为：test、test

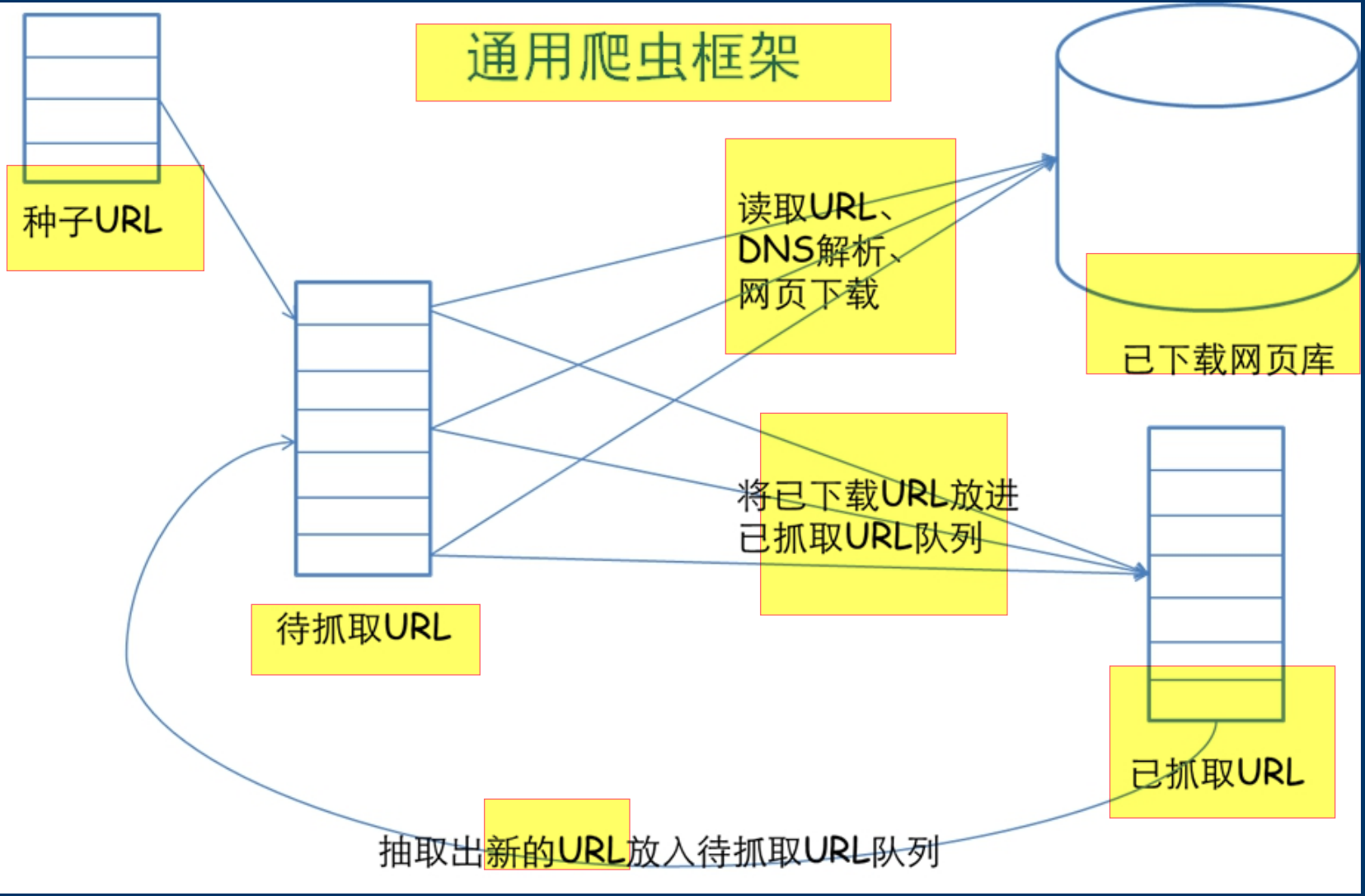
    ✓2. 服务器IP和port为：192.168.0.1:21

    ✓3. 登录后转到profile文件夹

9.1-2 爬虫的工作流程

- 基本工作流程
  - 1. 首先选取一部分精心挑选的种子URL；
  - 2. 将这些URL放入待抓取URL队列；
  - 3. 从队列中取出待抓取URL，解析DNS，并且得到主机的IP，并将URL对应的网页下载下来，存储进已下载网页库中。此外，将这些URL放进已抓取URL队列。
  - 4. 分析已抓取URL队列中的URL，分析其中的其他URL，并且将URL放入待抓取URL队列，从而进入下一个循环。

9.1-2 爬虫的工作流程


通用爬虫框架
种子URL
读取URL、
DNS解析、
网页下载
已下载网页库
将已下载URL放进
已抓取URL队列
待抓取URL
已抓取URL
抽取出新的URL放入待抓取URL队列

9.1-2 爬虫的工作流程

- Python爬虫工具分类：
  - 简单爬虫设计
    - ✓下载网页：urllib（入门简单爬虫库），Requests
    - ✓解析网页：Beautiful Soup
    - ✓模拟交互，处理JS动态网页：Selenium
  - 高级爬虫框架Scrapy
  - 分布式爬虫设计
    - ✓分布式队列
    - ✓布隆过滤器 (Bloom Filter)

9.2 urllib模块

urllib 库

- Urllib库是Python中一个功能强大、用于操作URL，在做网络爬虫时经常用到的库。
- 在Python2.x中，分为Urllib库和Urllib2库，Python3.x之后都合并到Urllib库中，使用方法稍有不同。
  - ✓在Pytho2.x中使用import urllib2——-对应的，在Python3.x中会使用

    import urllib.request，urllib.error
  - ✓在Pytho2.x中使用import urlopen——-对应的，在Python3.x中会使用

    import urllib.request.urlopen
  - ✓在Pytho2.x中使用import urlparse——-对应的，在Python3.x中会使用

    import urllib.parse

9.2 urllib模块

- urllib 库的四个基本模块
  - urllib.request：可以用来发送request和获取request的结果
  - urllib.error：包含urllib.request产生的异常
  - urllib.parse：用来解析和处理URL
  - urllib.robotparse：用来解析页面的robots.txt文件

9.2-1 基本操作

1.使用urlopen获取网页

- urllib.request.urlopen(url, data=None, [timeout,]*, cafile=None, capath=None, cadefault=False, context=None)
  - ✓url：需要打开的网址，可以是一个string，或者一个Request对象
  - ✓data：Post方式提交的数据
  - ✓timeout：设置网站的访问超时时间

```python
#simplest_crawler.py
import urllib.request as ur
response = ur.urlopen('http://python.org/')
print(response)
html = response.read()
print("Python主页：")
print(html)
```

HTTPResponse 对象

```
<http.client.HTTPResponse object at 0x00000232308686D8>
Python主页：
b'<!doctype html>\n<!--[if lt IE 7]>   <html class="no-js ie6
lt-ie7 lt-ie8 lt-ie9">   <![endif]-->\n<!--[if IE 7]>
<html class="no-js ie7 lt-ie8 lt-ie9">          <![endif]-->
\n<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">
<![endif]-->\n<!--[if gt IE 8]><!--><html class="no-js"
lang="en" dir="ltr">  <!--<![endif]-->\n\n<head>\n    <meta
charset="utf-8">\n    <meta http-equiv="X-UA-Compatible"
content="IE=edge">\n\n    <link rel="prefetch" href="//
```

9.2-1 基本操作

- urlopen返回的对象提供方法：
  - read() , readline() ,readlines() , fileno() , close() ：对HTTPResponse类型数据进行操作
  - info()：返回HTTPMessage对象，表示远程服务器返回的头信息
  - getcode()：返回Http状态码。如果是http请求，200请求成功完成;404网址未找到
  - geturl()：返回请求的url

9.2-1 基本操作

- 2. 使用urllib.request.Request包装请求，再通过urlopen获取页面

```
2 #request_urlopen.py
3 import urllib.request as ur
4 url = r'http://www.lagou.com/zhaopin/Python/?labelWords=label'
5 headers = {        #伪装成浏览器
6     'User-Agent': r'Mozilla/5.0 (Windows NT 6.1; WOW64) \
7                     AppleWebKit/537.36(KHTML, like Gecko) '
8                    r'Chrome/45.0.2454.85 Safari/537.36 115Browser/6.0.3',
9     'Referer': r'http://www.lagou.com/zhaopin/Python/?labelWords=label',
10    'Connection': 'keep-alive' }
11 req = ur.Request(url, headers=headers)
12 print(req)
13 page = ur.urlopen(req).read()
14 page = page.decode('utf-8')
15 print(page)
16
```

```
<urllib.request.Request object at 0x0000012C875A0048>
<!DOCTYPE html>
<html>
<head>
    <!-- meta -->
    <meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible"
```

9.2-1 基本操作

Post数据

- urllib.request.Request()中的data参数默认为None，当data参数不为空的时候，urlopen()提交方式为Post

```
#post_urlopen.py
from urllib import request, parse
url = r'http://www.lagou.com/jobs/positionAjax.json?'
headers = {          #伪装成浏览器
    'User-Agent': r'Mozilla/5.0 (Windows NT 6.1; WOW64) \
                    AppleWebKit/537.36 (KHTML, like Gecko) '
                   r'Chrome/45.0.2454.85 Safari/537.36 115Browser/6.0.3',
    'Referer': r'http://www.lagou.com/zhaopin/Python/?labelWords=label',
    'Connection': 'keep-alive' }
data = {
    'first': 'true',
    'pn': 1,
    'kd': 'Python' }
data = parse.urlencode(data).encode('utf-8')
req = request.Request(url, headers=headers, data=data)
page = request.urlopen(req).read()
page = page.decode('utf-8')

```

9.2-1 基本操作

- Post数据(续)
  - urllib.parse.urlencode(query, doseq=False, safe='', encoding=None, errors=None)
    urlencode( )主要作用就是将url附上要提交的数据

```
data = {
    'first': 'true',
    'pn': 1,
    'kd': 'Python' }
data = parse.urlencode(data).encode('utf-8')
req = request.Request(url, headers=headers, data=data)
```

  - 经过urlencode( )转换后的data数据为?first=true?pn=1?kd=Python，最后提交的url为:
    http://www.lagou.com/jobs/positionAjax.json?first=true?pn=1?kd=Python

9.2-2 简单抓取网页

不同网页爬取

```
3 #tieba_crawler.py
4 import urllib.request     #主要用于打开和阅读url
5 from os.path import join #用于串联完整路径
6
7 wp=r"D:\Teaching\@LessonCourse\Python Programming and Practice\MyLecture\pyegs\ch9\html"
8 print("模拟抓取百度贴吧python、java、C#页面,并写入指定路径文件")
9 def tieba_baidu(url,s):
10     header={'User-Agent': r'Mozilla/5.0 (Windows NT 6.3; WOW64) \
11     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.235'}#伪装成浏览器
12     for i in range(len(s)):
13         file_name=join(wp,s[i]+".html")
14         print("正在下载"+s[i]+"页面，并保存为"+file_name)
15         req=urllib.request.Request(url+s[i],headers=header)
16         m=urllib.request.urlopen(req).read()
17         with open(file_name,"wb") as file:
18             file.write(m)
19
20 if __name__=="__main__":
21     url="http://tieba.baidu.com/f?kw="
22     s_tieba=["python","java","c#"]
23     tieba_baidu(url,s_tieba)
24
```

9.2-2 简单抓取网页

下载网页中的图片

```python
#picture_crawler.py
import urllib.request     #主要用于打开和阅读url
import os,re
import urllib.error       #用于错误处理

print("爬取指定页面的jpg格式的图片")
dirpath="D:\Teaching\@LessonCourse\Python Programming and Practice\MyLecture\pyegs\ch9\pic"
def baidu_tieba(url,s):
    '''根据传入的地址和关键字列表进行图片抓取'''
    for i in range(len(s)):
        count=1
        file_name=os.path.join(dirpath,s[i]+".html")
        print("正在下载"+s[i]+"页面，并保存为"+file_name)
        m=urllib.request.urlopen(url+s[i]).read()
        dirname=s[i]
        new_path=os.path.join(dirpath,dirname)
        if not os.path.isdir(new_path):
            os.makedirs(new_path) #创建目录保存每个网页上的图片
        page_data=m.decode()
        page_image=re.compile('<img src=\"(.+?)\"')     #匹配图片的pattern
        for image in page_image.findall(page_data):#用正则表达式匹配所有的图片
            pattern=re.compile(r'http://.*.jpg$')    #匹配jpg格式的文件
            if pattern.match(image): #如果匹配 则获取图片信息，若不匹配，进行下一个页面的匹配
                try:
                    image_data=urllib.request.urlopen(image).read()     #获取图片信息
                    image_path=os.path.join(dirpath,dirname,str(count)+".jpg")    #给图片命名
                    count+=1
                    with open(image_path,"wb") as image_file:
                        image_file.write(image_data)                     #将图片写入文件
                except urllib.error.URLError as e:
                    print("Download failed")
            with open(file_name,"wb") as file:     #将页面写入文件
                file.write(m)

if __name__=="__main__":
    url="http://tieba.baidu.com/f?kw="
    s_tieba=["python","java","c#"]
    baidu_tieba(url,s_tieba)
```

9.3 Beautiful Soup

- Beautiful Soup为python抓取网页数据的一个库
  - Beautiful Soup提供一些简单的、python式的函数用来处理导航、搜索、修改分析树等功能。
  - Beautiful Soup是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。
  - Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为utf-8编码。
  - Beautiful Soup已成为和lxml、html6lib一样出色的python解释器，为用户灵活地提供不同的解析策略或强劲的速度。
  - Python正则表达式的使用较为繁琐，而 Beautiful Soup在数据提取方面友好易用。

9.3 Beautiful Soup

一个简单例子：抓取文章标题

```
3 #bs4_title_crawler.py
4 from urllib import request
5 from bs4 import BeautifulSoup
6
7 #获取简书首页的全部文章标题
8 url = r'http://www.jianshu.com'
9 # 模拟真实浏览器进行访问
10 headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) \
11     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36'}
12 page = request.Request(url, headers=headers)
13 page_info = request.urlopen(page).read()
14 page_info = page_info.decode('utf-8')
15 # 将获取到的内容转换成BeautifulSoup格式，并将html.parser作为解析器
16 soup = BeautifulSoup(page_info, 'html.parser')
17 # 以格式化的形式打印html
18 # print(soup.prettify())
19 titles = soup.find_all('a', 'title')  # 查找所有a标签中class='title'的语句
20 # 打印查找到的每一个a标签的string
21 #print(titles)
22 for title in titles:
23     print(title.string)
```