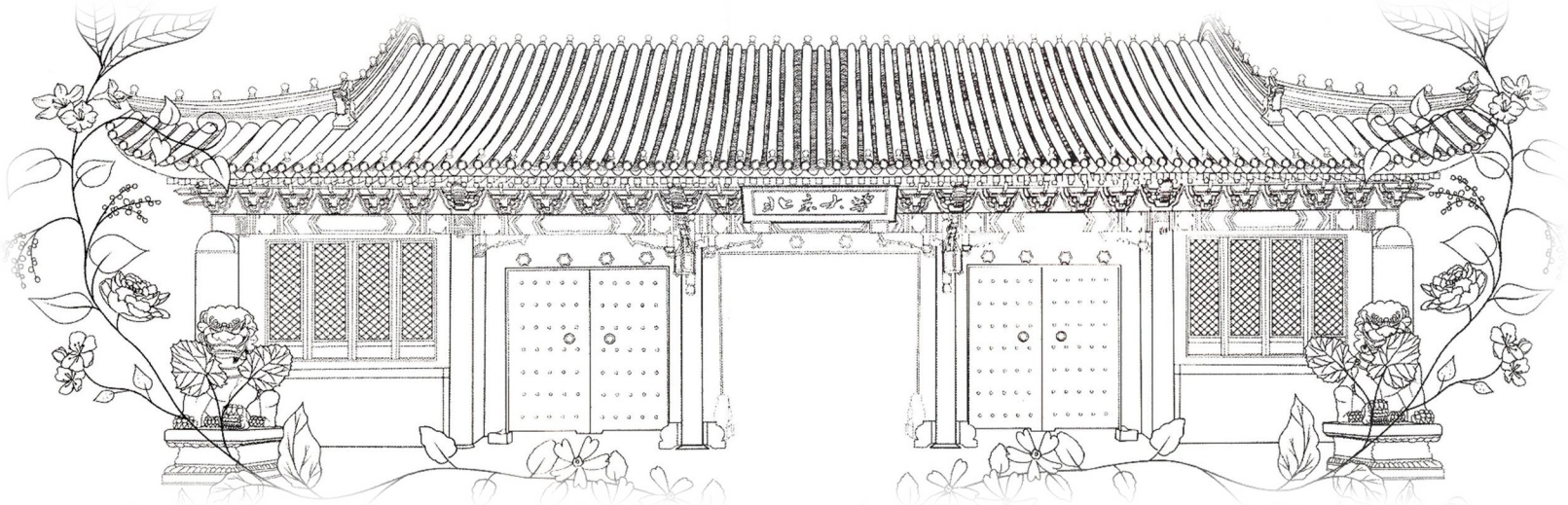
第 1 章 Python数据分析基础 I

北京大学信息管理系

2021/3/7



本章内容

01 Python语法回顾

02 常用工具包

03 NumPy基础

04 Pandas基础

本章内容

01 Python语法回顾

02 常用工具包

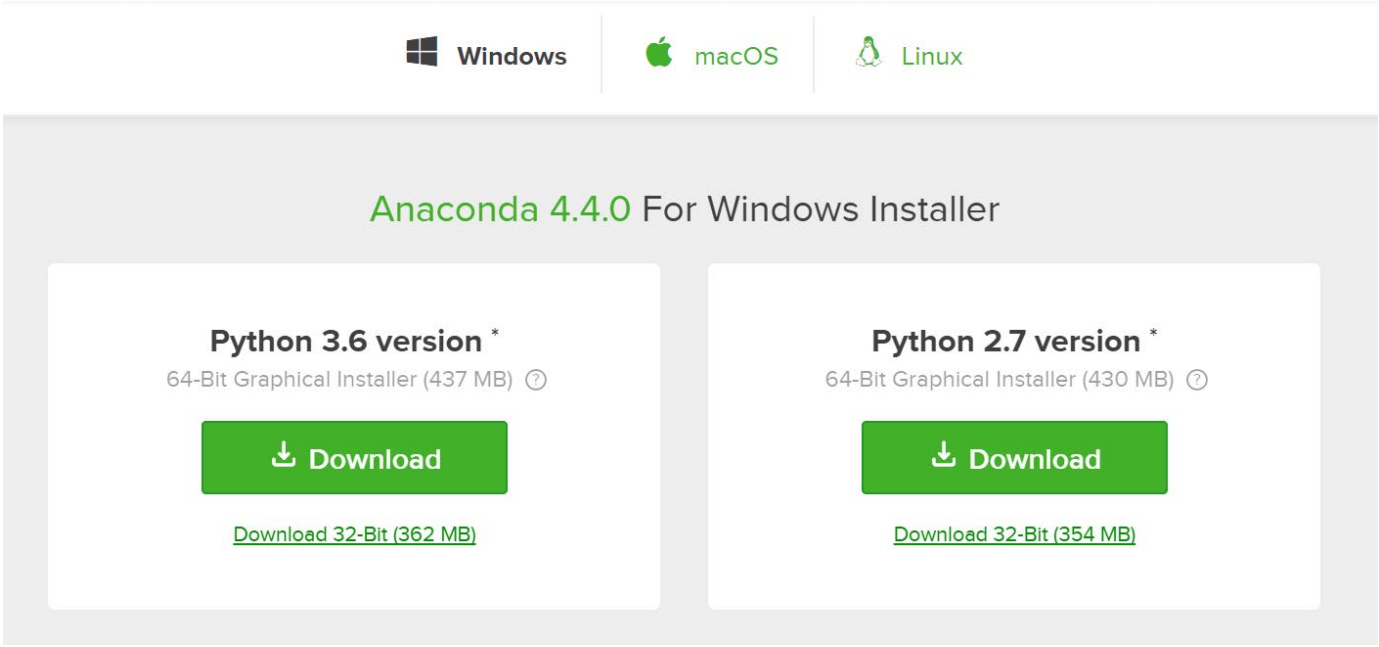
03 NumPy基础

04 Pandas基础

选择Python

- 对于数据分析和交互式、探索性计算和数据可视化，Python将不可避免地与广泛使用的其他领域特定的开放源码和商业编程语言和工具进行比较，例如R、MATLAB、SAS、Stata等。
- 近年来，Python改进了的库支持(主要是Pandas)使其成为数据处理任务的强大工具。结合Python在通用编程中的优势，它是构建以数据为中心的应用程序的单一语言的最佳选择。

利用Anaconda安装Python



Anacnoda3 5.2.0 版本可通过以下链接下载
https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/

Jupyter Notebook

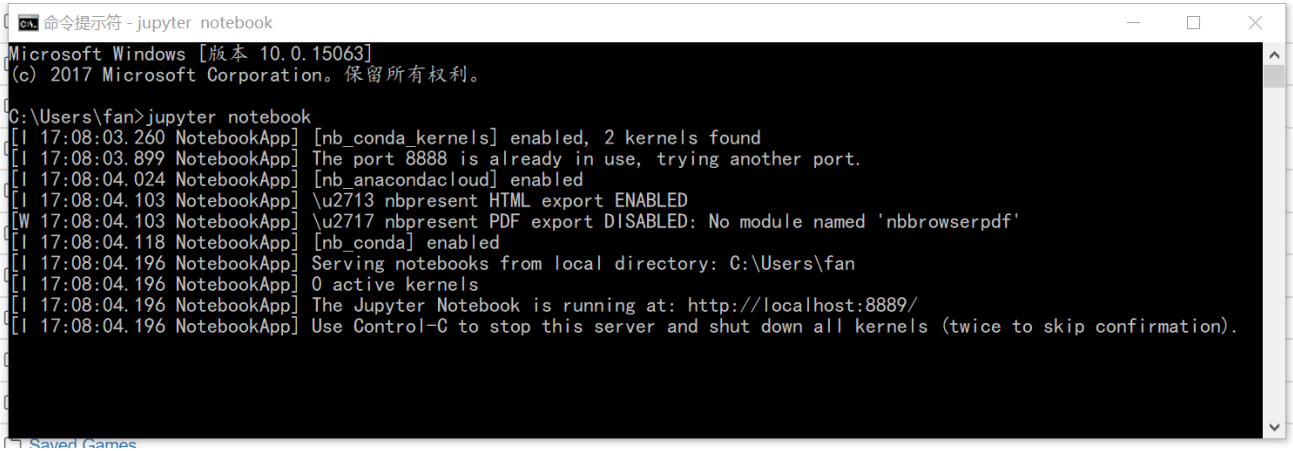
Jupyter Notebook是一种交互式笔记本。

- Jupyter Notebook 与 IPython终端 共享同一个内核。
- 继承了IPython的诸多优点，此外：
- 基于过程：作为笔记本，可以交互式地做计算，并且画图，按顺序保留所有脚本和输出。

Jupyter Notebook

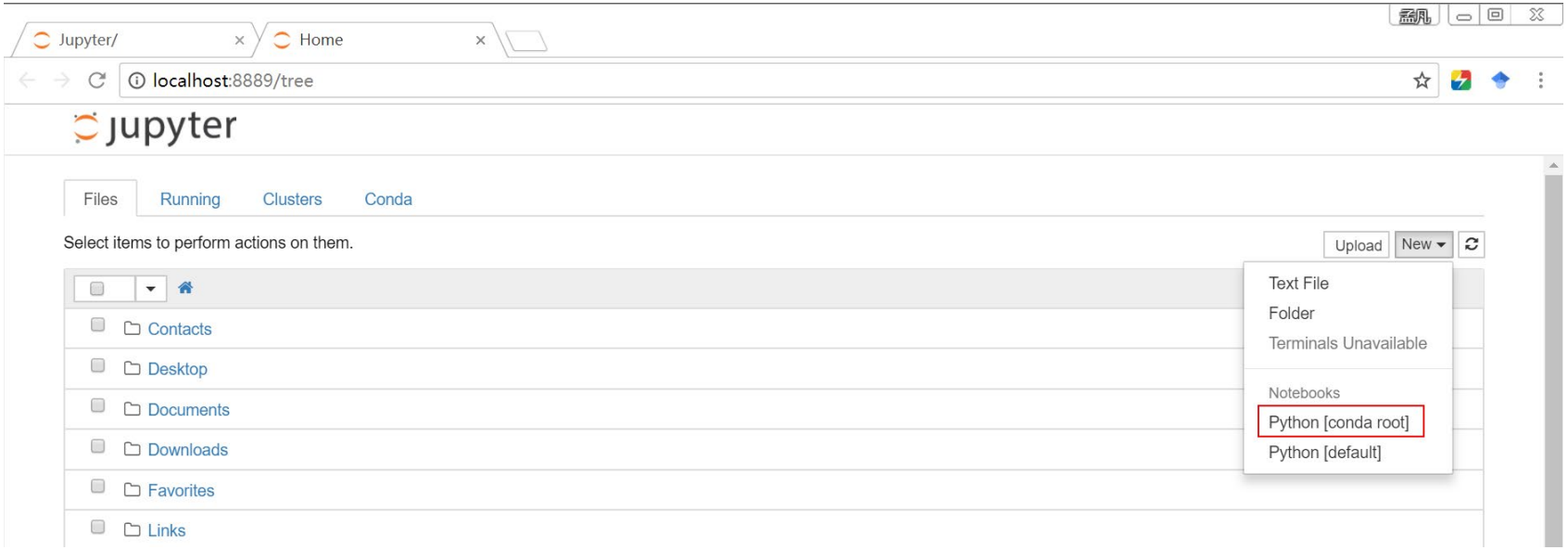
1. 安装：随Anaconda自动安装
2. 打开：在命令提示行或搜索框或运行中运行：

   jupyter notebook



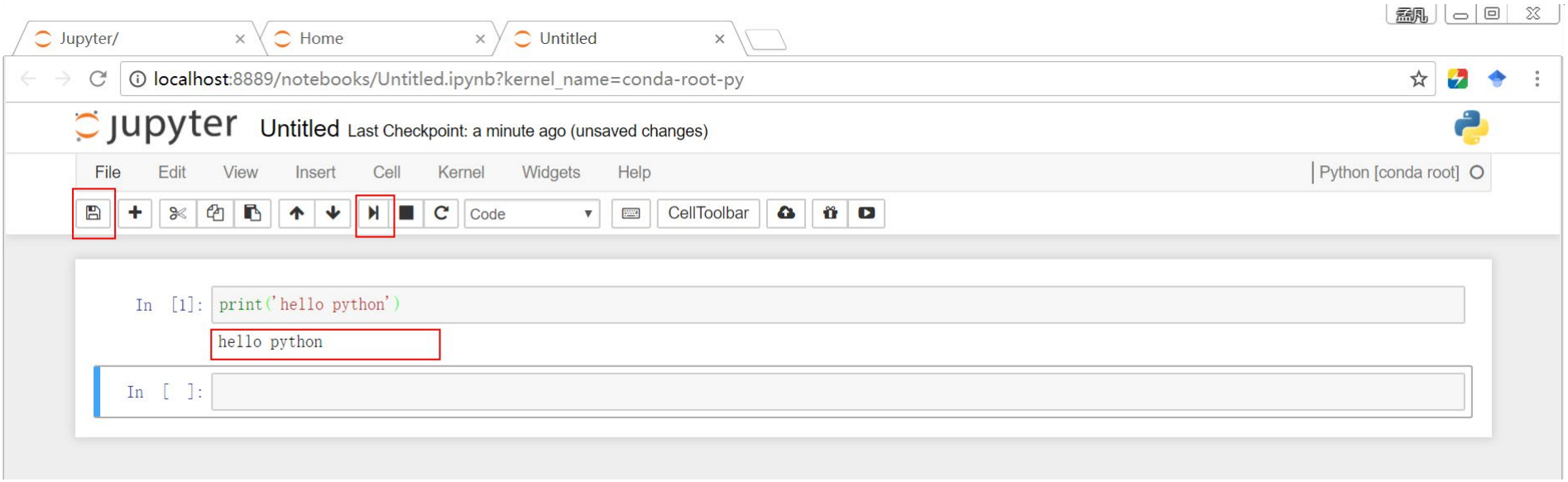
Jupyter Notebook

3. 新建notebook：New->Python



Jupyter Notebook

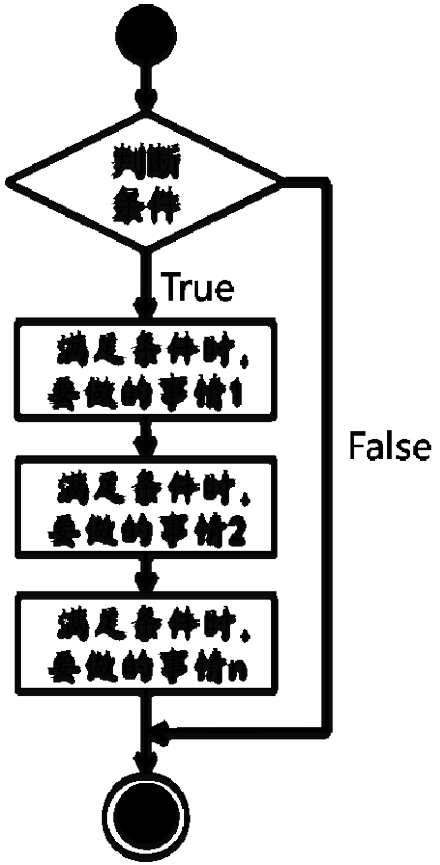
4. 输入、执行、保存



if 语句

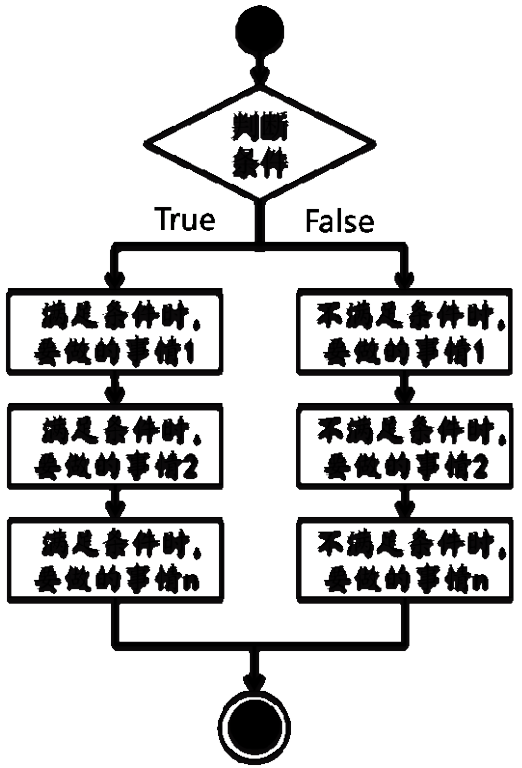
if语句是最简单的条件判断语句，它可以控制程序的执行流程。

```
if 判断条件:
    满足条件时要做的事情1…
    满足条件时要做的事情2…
    ...(省略)...
    满足条件时要做的事情n…
```



if-else语句

```
if 条件:
    满足条件时要做的事情1
    满足条件时要做的事情2
    ...(省略)...
    满足条件时要做的事情3
else:
    不满足条件时要做的事情1
    不满足条件时要做的事情2
    ...(省略)...
    不满足条件时要做的事情n
```



if-elif语句

if-elif判断语句，该语句可以判断多种情况。

```
if 判断条件1:
        满足条件1时要做的事情
elif 判断条件2:
        满足条件2时要做的事情
elif 判断条件3:
        满足条件3时要做的事情
```

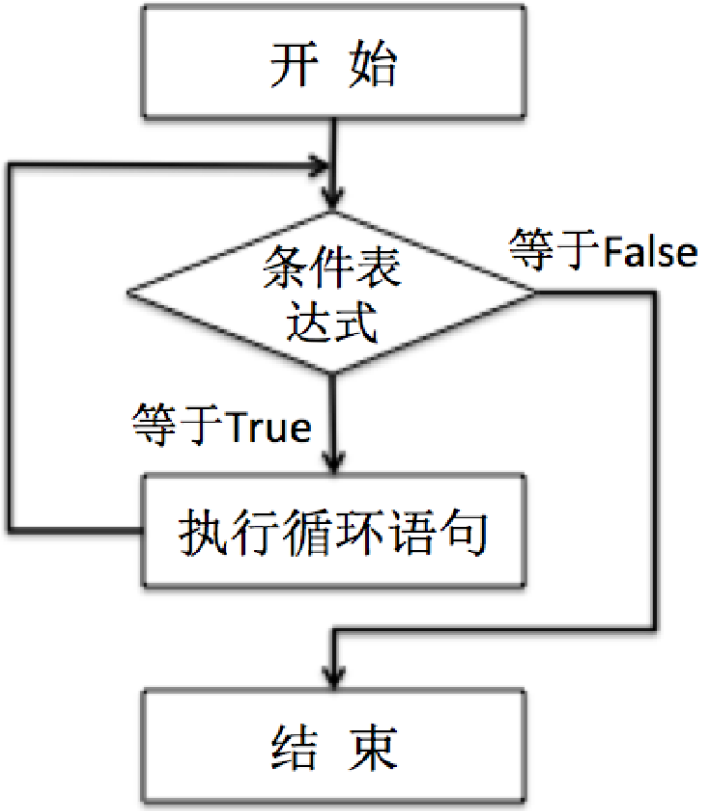
- 当满足判断条件1时，执行满足条件1时要做的事情，然后整个if结束；
- 如果不满足判断条件1，那么判断是都满足条件2，如果满足判断条件2，就执行满足条件2时要做的事情，然后整个if结束
- 当不满足判断条件1和判断条件2，如果满足判断条件3，则执行满足判断条件3时要做的事情，然后整个if结束。

while循环

while循环语法格式

while 条件表达式：
    条件满足，执行循环语句

条件表达式永远为true，无限循环。



for循环

Python中的for循环可以遍历任何序列的项目：

语法格式

```
for 变量 in 序列:
    循环语句
```

例如：

```
for i in [0,1,2]:
    print(i)
```

for循环

考虑到我们使用的数值范围经常变化，Python提供了一个内置range()函数，它可以生成一个数字序列。

语法格式

```
for i in range(start,end):
    执行循环语句
```

程序在执行for循环时：

- 循环计时器变量i被设置为start；
- 执行循环语句；
- i递增
- 每设置一个新值都会执行一次循环
- 当i等于end时，循环结束。

break语句

break语句用于结束整个循环。

```
for i in range(5):
    print("-------")
    print(i)
```

```
for i in range(5):
    print("-------")
    if i==3:
        break
    print(i)
```

这两个结果有什么不同？

continue语句

continue的作用是用来结束本次循环，紧接着执行下一次的循环。

```
for i in range(5):
    print("-------")
    print(i)
```

```
for i in range(5):
    print("-------")
    if i==3:
        continue
    print(i)
```

这两个结果有什么不同？

pass语句

pass是空语句，它是为了保持程序结构完整性。

```
for letter in 'Runoob':
   if letter == 'o':
      pass
      print ('执行 pass 块')
   print ('当前字母 :', letter)
print ("Good bye!")
```

pass语句不做任何事情，用作占位。

else语句

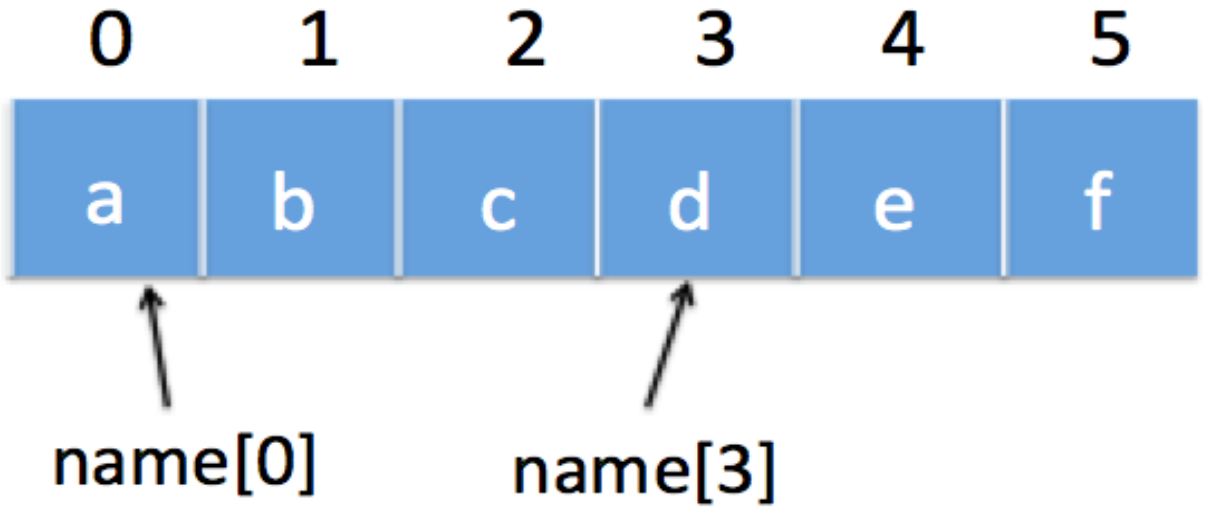
else语句可以和循环语句结合使用，并且else语句旨在循环完成后执行。

```
count = 0
while count < 5:
        print(count, " is less than 5")
        count = count + 1
else:
        print(count, " is not less than 5")
```

运行结果是什么？

字符串的存储方式

字符串中的每个字符都对应一个下标，下标编号是从0开始的。



什么是切片

切片的语法格式如下所示：

[起始:结束:步长]

切片选取的区间属于**左闭右开**型，即从"起始"位开始，到"结束"位的前一位结束（不包含结束位本身）

使用切片截取字符串

假设有字符串 name="abcdef"，则：

name[0:3]	→	abc
name[3:5]	→	de
name[1:-1]	→	bcde
name[2:])	→	cdef
name[::-2]	→	fdb

列表

- 列表是Python中内置可变序列，是一个元素的有序集合，列表中的每一个数据称为元素，列表的所有元素放在一对中括号“[”和“]”中，并使用逗号分隔开；
- 当列表元素增加或删除时，列表对象自动进行扩展或收缩内存，保证元素之间没有缝隙；
- 在Python中，一个列表中的数据类型可以各不相同，可以同时分别为整数、实数、字符串等基本类型，甚至是列表、元素、字典、集合以及其他自定义类型的对象。例如：

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
['spam', 2.0, 5, [10, 20]]
[['file1', 200,7], ['file2', 260,9]]
```

列表方法

方法	说明
list.append(x)	将元素x添加至列表尾部
list.extend(L)	将列表L中所有元素添加至列表尾部
list.insert(index, x)	在列表指定位置index处添加元素x
list.remove(x)	在列表中删除首次出现的指定元素
list.pop([index])	删除并返回列表对象指定位置的元素
list.clear()	删除列表中所有元素，但保留列表对象，该方法在Python2中没有
list.index(x)	返回值为x的首个元素的下标
list.count(x)	返回指定元素x在列表中的出现次数
list.reverse()	对列表元素进行原地逆序
list.sort()	对列表元素进行原地排序
list.copy()	返回列表对象的**浅拷贝**，该方法在Python2中没有

列表推导式

- 列表推导式可以说是Python程序开发时应用最多的技术之一。列表推导式使用非常简洁的方式来快速生成满足特定需求的列表，代码具有非常强的可读性。例如：

```
>>> aList = [x*x for x in range(10)]
# 相当于
>>> aList = []
>>> for x in range(10):
        aList.append(x*x)
```

列表推导式

- 列表推导能非常简洁的构造一个新列表:只用一条简洁的表达式即可对得到的元素进行转换变形
- 其基本格式如下:

```
[expr for value in collection ifcondition]
```

- 相当于

```
>>> result = []
>>> for value in collection:
        if condition:
            result.append(expression)
```

列表推导式

- 使用列表推导式实现嵌套列表的平铺

```
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 列出当前文件夹下所有Python源文件

```
>>> [filename for filename in os.listdir('.') if filename.endswith('.py')]
```

- 列出当前列表中符合某条件的元素

```
>>> aList = [-1,-4,6,7.5,-2.3,9,-11]
>>> [i for i in aList if i>0]
```

[6, 7.5, 9]

列表推导式

- 在列表推导式中使用多个循环，实现多序列元素的任意组合，并且可以结合条件语句过滤特定元素

```
>>> [(x, y) for x in range(3) for y in range(3)]
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

元组

- 元组和列表类似，但属于不可变序列，元组一旦创建，用任何方法都不可以修改其元素。
- 元组的定义方式和列表相同，但定义时所有元素是放在一对圆括号“(”和“)”中，而不是方括号中。

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

元组创建与删除

- 使用“=”将一个元组赋值给变量

```
>>> a_tuple = ('a', )
>>> a_tuple
('a')
>>> a_tuple = ('a', 'b', 'mpilgrim',
'z', 'example')
>>> a_tuple
('a', 'b', 'mpilgrim', 'z', 'example')
```

```
>>> a = 3
>>> a
3
>>> a = 3,
>>> a
(3,)
>>> x = () #空元组
>>> x
()
```

元组与列表的区别

- 元组中的数据一旦定义就不允许更改。
- 元组没有append()、extend()和insert()等方法，无法向元组中添加元素；
- 元组没有remove()或pop()方法，也无法对元组元素进行del操作，不能从元组中删除元素。
- 内建的tuple( )函数接受一个列表参数，并返回一个包含同样元素的元组，而list( )函数接受一个元组参数并返回一个列表。从效果上看，tuple( )冻结列表，而list( )融化元组。

元组的优点

- 元组的速度比列表更快。如果定义了一系列常量值，而所需做的仅是对它进行遍历，那么一般使用元组而不用列表。
- 元组对不需要改变的数据进行“写保护”将使得代码更加安全。
- 一些元组可用作字典键（特别是包含字符串、数值和其它元组这样的不可变数据的元组）。列表永远不能当做字典键使用，因为列表不是不可变的。

生成器表达式

- 生成器推导式与列表推导式非常接近，只是生成器推导式使用圆括号而不是列表推导式所使用的方括号。
- 与列表推导式不同的是，生成器推导式的结果是一个生成器对象，而不是列表，也不是元组。
- 使用生成器对象的元素时，可以根据需要将其转化为列表或元组，也可以使用生成器对象的__next__()方法（Python 3.x）进行遍历，或者直接将其作为迭代器对象来使用。
- 不管用哪种方法访问其元素，当所有元素访问结束以后，如果需要重新访问其中的元素，必须重新创建该生成器对象。

生成器表达式

```
>>> g=((i+2)**2 for i in range(10))
>>> g
<generator object <genexpr> at
0x02B15C60>
>>>tuple(g)
(4, 9, 16, 25, 36, 49, 64, 81, 100, 121)
>>> tuple(g)
()
```

```
>>> g=((i+2)**2 for i in range(10))
>>> g.__next__()
4
>>> g.__next__()
9
>>> g.__next__()
16
>>> g.__next__()
25
```

生成器表达式

- 列表推导式采用方括号[]表示，并且用到了一个简写版的for循环，第一部分是一个生成结果列表元素的表达式，第二部分是一个输入表达式上的循环。阅读理解列表表达式的推荐做法是先从里面的for循环开始，向右查看是否有if条件，然后将推导式开始的那个表达式映射到每一个匹配的元素上去。
- 生成器表达式(generator expression)，除了它有一种称之为“惰性计算”的特点以外，它和列表推导式的用法基本一致。它的工作方式是每次处理一个对象，而不是一口气处理和构造整个数据结构，这样做的潜在优点是可以节省大量的内存。

字典

- 字典是键值对的无序可变集合。
- 定义字典时，每个元素的键和值用冒号分隔，元素之间用逗号分隔，所有的元素放在一对大括号“{”和“}”中。
- 字典中的每个元素包含两部分：键和值，向字典添加一个键的同时，必须为该键增添一个值。
- 字典中的键可以为任意不可变数据，比如整数、实数、复数、字符串、元组等等。
- 字典中的键不允许重复。
- globals()返回包含当前作用域内所有全局变量和值的字典
- locals()返回包含当前作用域内所有局部变量和值的字典

字典创建与删除

- 使用”=“将一个字典赋值给一个变量

```
>>> a_dict = {'server': 'db.diveintopython3.org', 'database':
'mysql'}
>>> a_dict
{'database': 'mysql', 'server': 'db.diveintopython3.org'}
>>> x = {} #空字典
>>> x
{}
```

集合的创建与删除

- 集合是无序可变集合，使用一对大括号界定，元素不可重复。
- 直接将集合赋值给变量

```
>>> a={3,5}
>>> a.add(7)
>>> a
set([3, 5, 7])
```

- 使用del删除整个集合

```
>>> del a
```

- 使用set将其他类型数据转换为集合

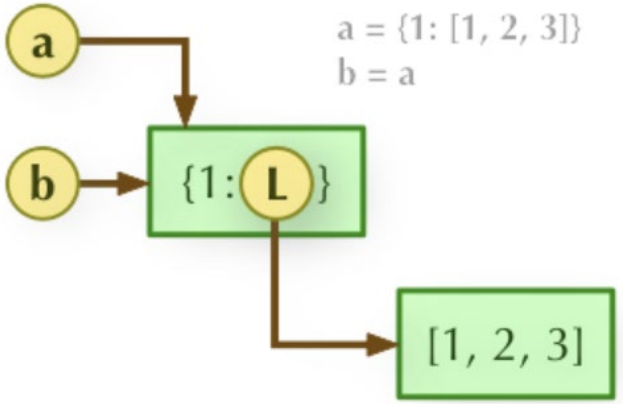
```
>>> a_set=set(range(8,14))
>>> a_set
set([8, 9, 10, 11, 12, 13])
>>> b_set=set([0,1,2,3,0,1,2,3,7,8])
>>> b_set
set([0, 1, 2, 3, 7, 8])
>>> c_set = set() #空集合
>>> c_set
set()
```

赋值和拷贝

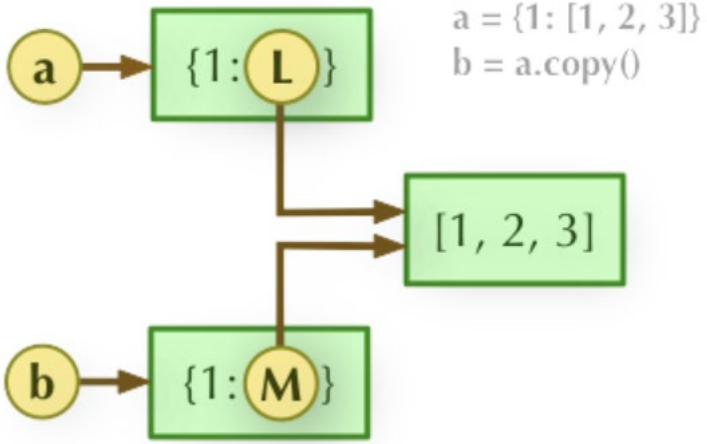
- 直接赋值：其实就是对象的引用（别名）。
- 浅拷贝(copy)：拷贝父对象，不会拷贝对象的内部的子对象。
- 深拷贝(deepcopy)： copy 模块的 deepcopy 方法，完全拷贝了父对象及其子对象。

赋值和拷贝

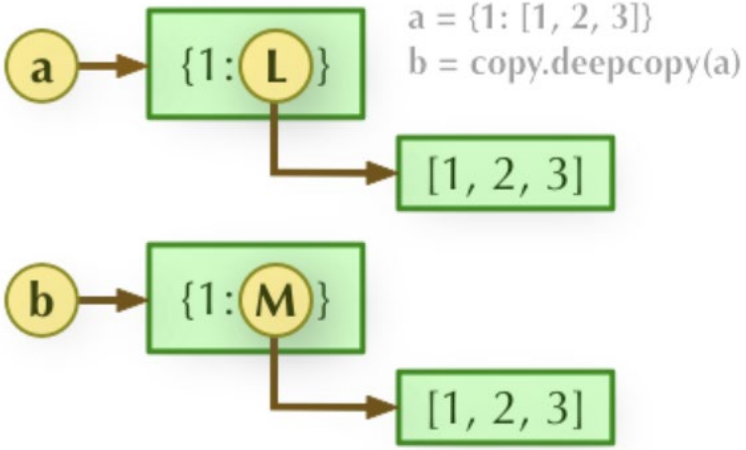
b = a: 赋值引用，a 和 b 都指向同一个对象。



b = a.copy(): 浅拷贝, a 和 b 是一个独立的对象，但他们的子对象还是指向统一对象（是引用）。



b = copy.deepcopy(a): 深度拷贝, a 和 b 完全拷贝了父对象及其子对象，两者是完全独立的。



赋值和拷贝

```
#!/usr/bin/python
# -*-coding:utf-8 -*-

import copy
a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a                        #赋值，传对象的引用
c = copy.copy(a)             #对象拷贝，浅拷贝
d = copy.deepcopy(a)         #对象拷贝，深拷贝

a.append(5)                  #修改对象a
a[4].append('c')             #修改对象a中的['a', 'b']数组对象

print( 'a = ', a )
print( 'b = ', b )
print( 'c = ', c )
print( 'd = ', d )
```

```
('a = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5])
('b = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5])
('c = ', [1, 2, 3, 4, ['a', 'b', 'c']])
('d = ', [1, 2, 3, 4, ['a', 'b']])
```

函数的定义和调用

Python定义函数使用def关键字，格式如下：

```
def 函数名（参数列表）：
    函数体
```

函数的定义和调用

这是一个自定义的函数：

```
def printInfo():
    print('----------------------------------')
    print('        生命苦短，我用Python          ')
    print('----------------------------------')
```

函数的定义和调用

定义了函数之后，想要让这些代码能够执行，需要调用函数。通过“函数名()” 即可完成调用。

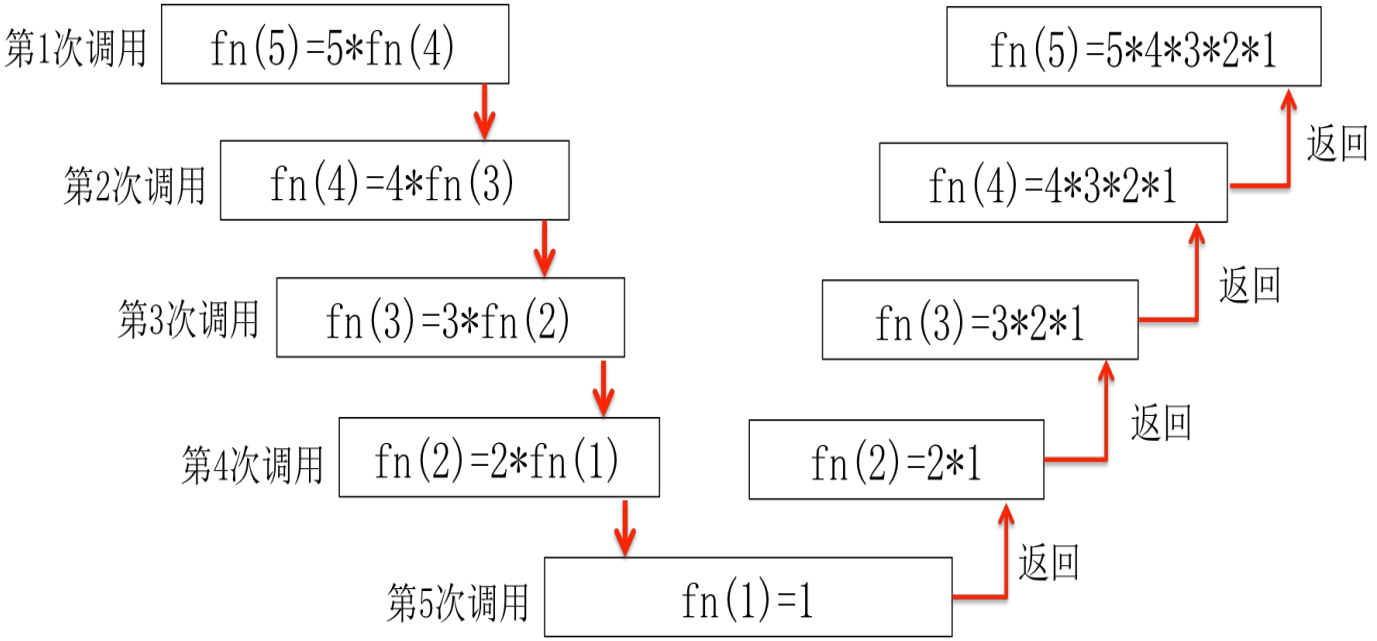
```
# 调用刚才定义的函数
printInfo()
```

递归函数

一个函数的内部可以调用其他函数。但是，如果一个函数在内部不调用其它的函数，而是自己本身的话，这个函数就是递归函数。

递归函数

使用递归，实现阶乘 n! = 1 * 2 * 3 * ... * n的计算。



匿名函数

匿名函数就是没有名称的函数，也就是不再使用def语句定义的函数。如果要声明匿名函数，则需要使用lambda关键字，匿名函数的声明格式如下所示：

```
lambda [arg1 [,arg2,.....argn]]:expression
```

匿名函数

匿名函数示例代码：

```
sum = lambda arg1, arg2: arg1 + arg2
print("运行结果： ", sum( 10, 20 ))
print("运行结果： ", sum( 20, 20 ))
```

匿名函数

```
salaries={
    'egon':3000,
    'alex':100000000,
    'wupeiqi':10000,
    'yuanhao':2000
}
def get(k):
    return salaries[k]
print(max(salaries,key=get)) #'alex'
print(max(salaries,key=lambda x:salaries[x]))
```

实例：求工资最高的人

匿名函数

注意：

使用Lambda声明的匿名函数能接收任何数量的参数，但只能返回一个表达式的值。匿名函数不能直接调用print，因为lambda需要一个表达式。

模块的制作

test.py

```
def add(a,b):
    return a+b
# 用来进行测试
ret = add(12,22)
print('int test.py file,12+22=%d'%ret)
```

main.py

```
import test
result = test.add(11,22)
print(result)
```

运行main文件时，发现运行结果输出了test.py中的测试代码。

模块的制作

- python提供了一个__name__属性，每个模块都有一个__name__属性，当其值为'__main__'时，表明该模块自身在运行，否则是被引用。

```
if __name__=='__main__':
        ret=add(12,22)
        print('int test.py file,__name__is %s'%__name__)
```

本章内容

01 Python语法回顾

02 常用工具包

03 NumPy基础

04 Pandas基础

数据分析常用工具包

Python进行数据分析的常用工具包主要有：

- **NumPy**
- **Pandas**
- **Matplotlib**
- **Seaborn**
- **Statsmodels**
- **SciPy**
- **scikit-learn**
- **……**

NumPy

NumPy(全称为Numerical Python)是Python中科学计算的基本包。Python数据分析的绝大多数内容基于NumPy和在NumPy之上构建的库。其提供了一下功能：

- 一个快速高效的多维数组对象ndarray
- 用数组或数组之间的数学运算来执行元素计算的函数
- 用于读取和写入基于数组的数据集到磁盘的工具
- 线性代数运算，傅里叶变换，和随机数生成
- 集成连接C、C++和Fortran代码到Python的工具

NumPy

除了NumPy在Python中添加的快速数组处理功能之外，它在数据分析方面的主要目的之一是作为**数据在算法之间传递的主要容器**。

- 对于数字数据，NumPy数组比其他内置的Python数据结构**更有效地存储和操作数据**。
- 另外，以较低级别的语言(如C或Fortran)编写的库可以对存储在NumPy数组中的数据进行操作，而无需复制任何数据。

Pandas

Pandas提供丰富的数据结构和功能，旨在使结构化数据快速、简单、富有表现力。它是使Python成为一个强大且高效的数据分析环境的关键因素之一。Pandas基于两种数据类型：Series与DataFrame。

- Series是一个一维的数据类型，其中每一个元素都有一个标签。 Series类似于Numpy中元素带标签的数组。其中，标签可以是数字或者字符串。
- DataFrame是一个二维的表结构。Pandas的DataFrame可以存储许多种不同的数据类型，并且每一个坐标轴都有自己的标签。

Pandas

Pandas.Series

```
new_player
```

```
Fabregas        40000
Messi           75000
Ronaldo         85000
Rooney          50000
Van persie      67000
dtype: int64
```

Pandas. DataFrame

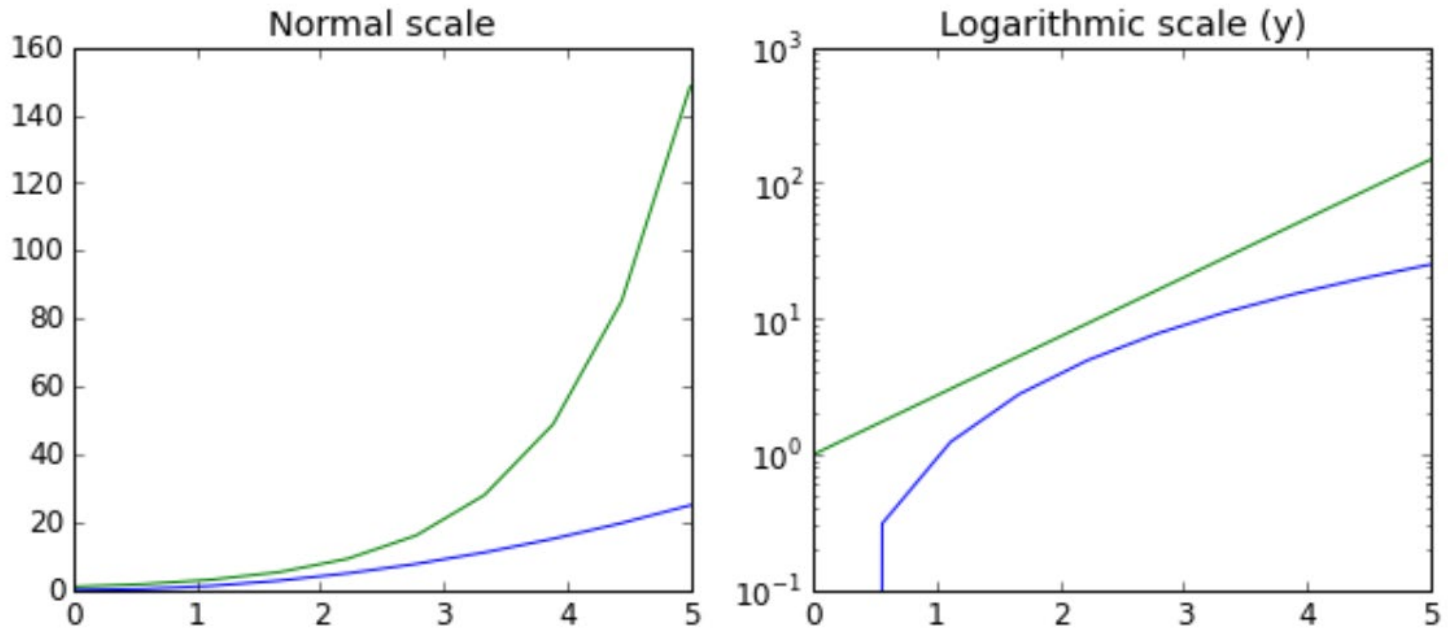
```
>>> frame
    total_bill  tip   sex     smoker  day  time    size
1   16.99       1.01  Female  No      Sun  Dinner  2
2   10.34       1.66  Male    No      Sun  Dinner  3
3   21.01       3.5   Male    No      Sun  Dinner  3
4   23.68       3.31  Male    No      Sun  Dinner  2
5   24.59       3.61  Female  No      Sun  Dinner  4
6   25.29       4.71  Male    No      Sun  Dinner  4
7   8.77        2     Male    No      Sun  Dinner  2
8   26.88       3.12  Male    No      Sun  Dinner  4
9   15.04       1.96  Male    No      Sun  Dinner  2
10  14.78       3.23  Male    No      Sun  Dinner  2
```

matplotlib

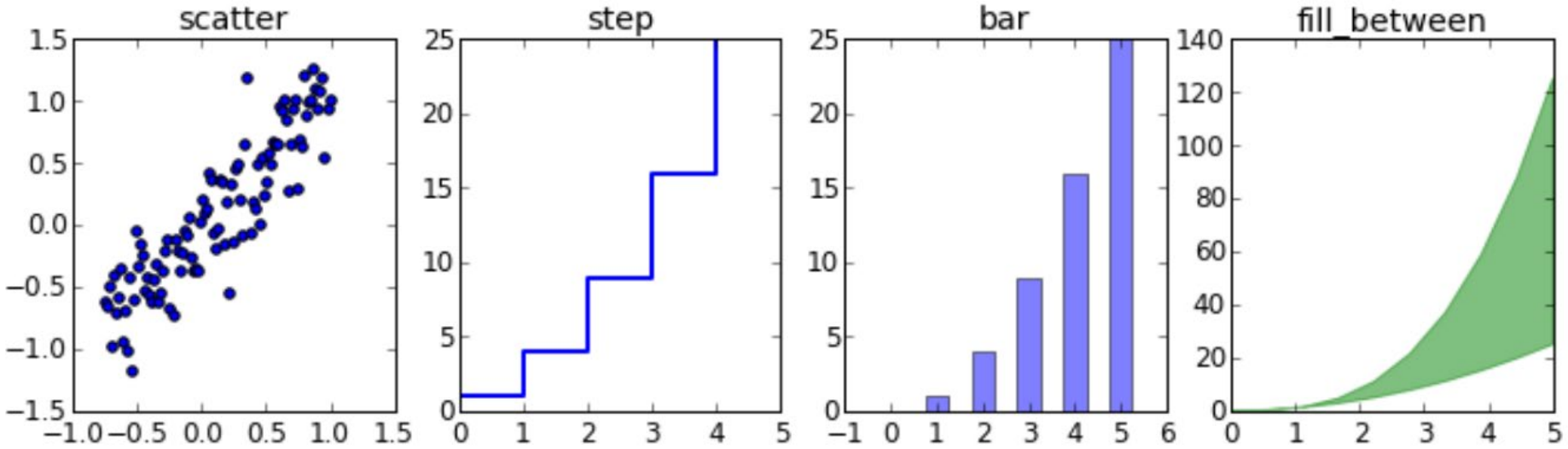
matplotlib是最流行的用于生成绘图和其他2D和3D数据可视化的Python库。它最初是由John D. Hunter (JDH)创建的，现在由一个大型开发团队维护。它非常适合于创建用于发布和展示的图形。

它与IPython集成得很好，从而为绘图和探测数据提供了一个舒适的交互式环境。这些图也具有互动性；可以在plot窗口中使用工具栏来放大图的一部分。

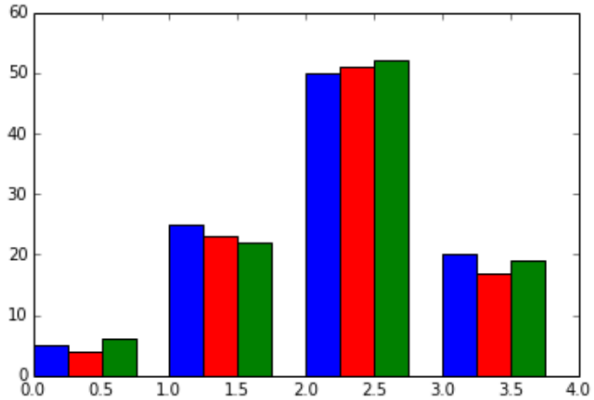
matplotlib

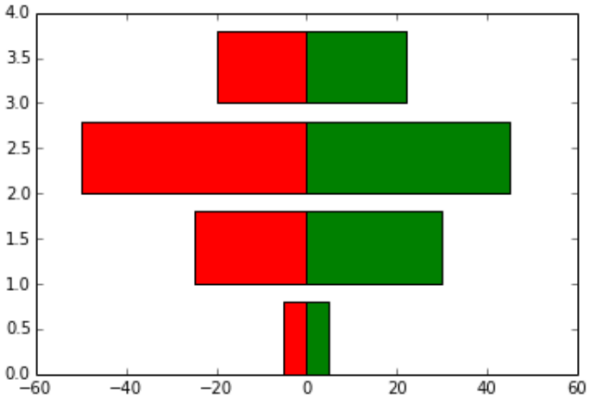

Normal scale
160
140
120
100
80
60
40
20
0
0
1
2
3
4
5
Logarithmic scale (y)
10^3
10^2
10^1
10^0
10^{-1}
0
1
2
3
4
5

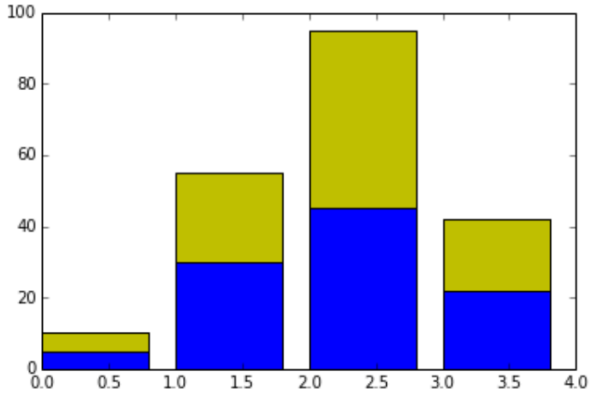
matplotlib

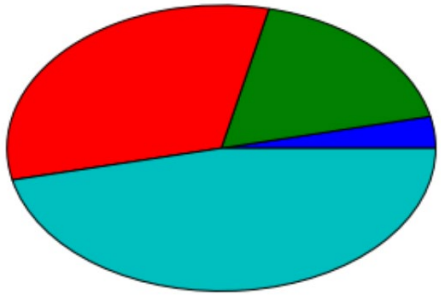


matplotlib

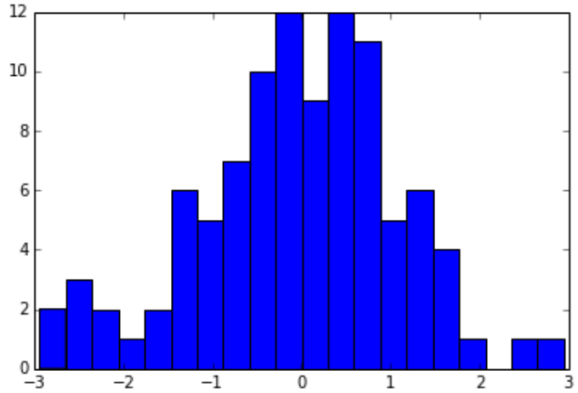

60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

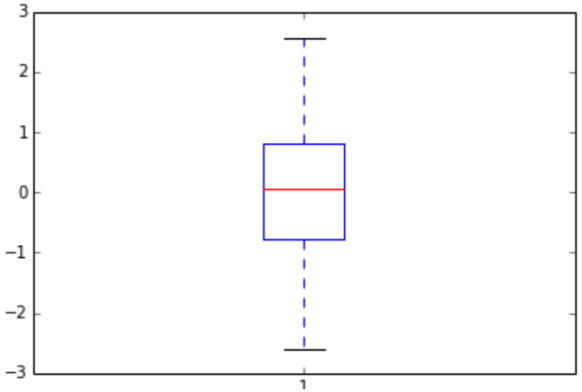

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−60
−40
−20
0
20
40
60

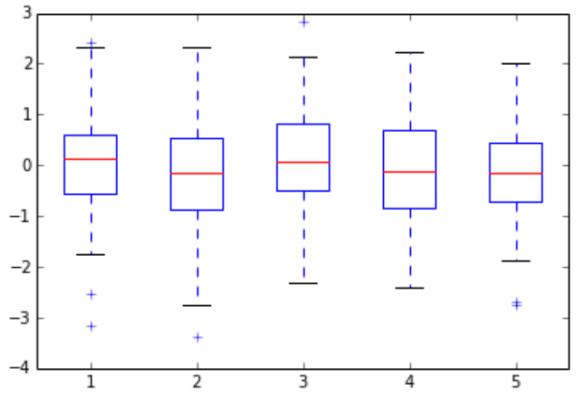

100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

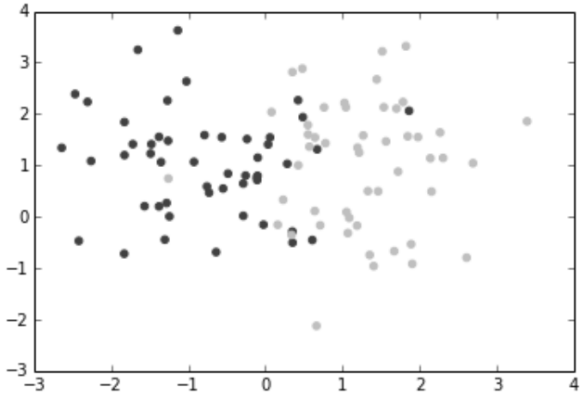


matplotlib

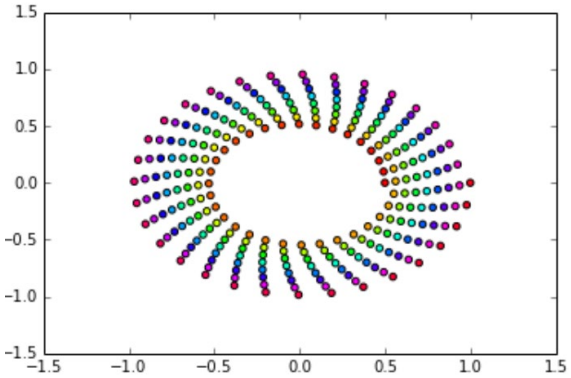

12
10
8
6
4
2
0
−3
−2
−1
0
1
2
3

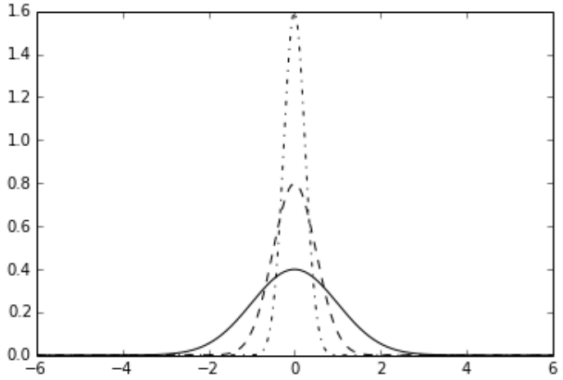

3
2
1
0
−1
−2
−3
1

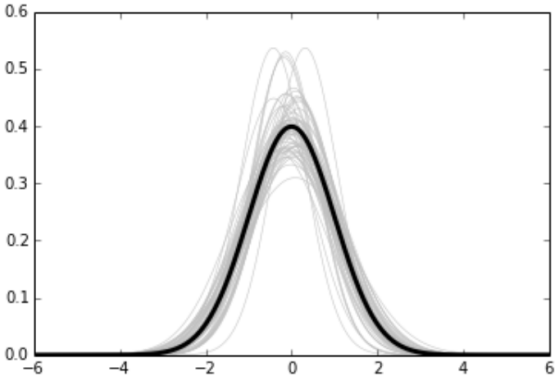

3
2
1
0
−1
−2
−3
−4
1
2
3
4
5

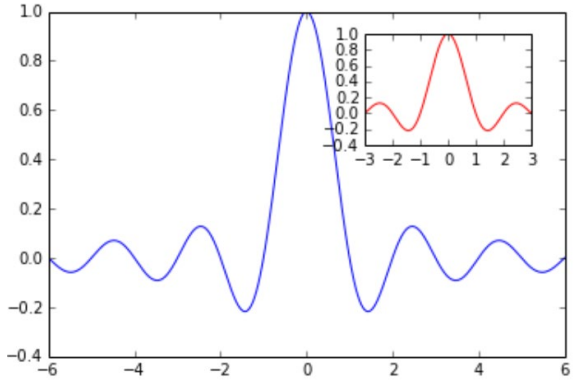

4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4

matplotlib


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

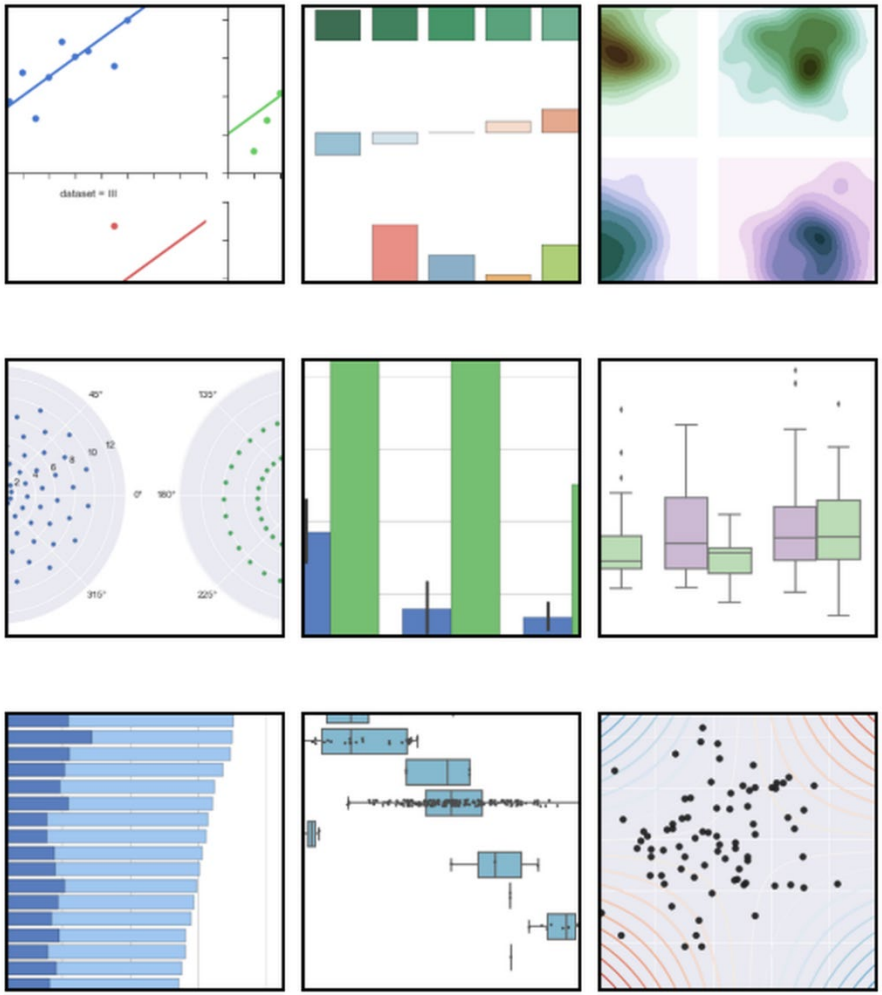

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6


0.6
0.5
0.4
0.3
0.2
0.1
0.0
−6
−4
−2
0
2
4
6


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−6
−4
−2
0
2
4
6
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−3
−2
−1
0
1
2
3

seaborn

- 另一种可视化常用的工具包
- 没有matplotlib灵活和自主，但封装得好，有时一行代码即可画出好看的图



statsmodels

Statsmodels是Python的统计建模和计量经济学工具包，包括一些描述统计、统计模型估计和推断。主要功能包括：

- Liner regression models：线性回归模型
- Generalized linear models：一般线型模型
- Robust linear models：鲁棒线性模型
- Discrete choice models：离散选择模型
- ANOVA：方差分析模型
- Time series analysis：时间序列分析
- Nonparametric estimators：非参检验
- a wide range of statistical tests：各种统计检验
- 以各种方式输出表格：text，latex，html；读取各种格式的数据
- 绘图功能

SciPy

SciPy 是基于Numpy构建在科学计算中处理多个不同标准问题域的包的集合。主要包括以下模块包括：

- scipy.integrate: 数值积分和微分方程求解器
- scipy.linalg: 拓展了numpy.linalg中的线性代数和矩阵分解功能
- scipy.optimize:函数优化器(最小化器)和根查找算法
- scipy.signal: 信号处理工具
- scipy.sparse: 系数矩阵和线性系统求解器
- scipy.special: 对于SPECFUN的封装， SPECFUN库实现了许多常见的数学函数
- scipy.stats: 标准连续和离散概率分布(密度函数，采样器，连续分布函数)，各种统计检验，和更多的描述性统计
- scipy.weave: 使用内联c++代码来加速数组计算的工具

通过结合使用NumPy和SciPy能够实现绝大部分matlab及其工具包的功能。

scikit-learn

scikit-learn是Python的一个开源机器学习模块，它建立在NumPy，SciPy和matplotlib模块之上，实现了大量的机器学习算法。包括：

- **Classification**：分类 - SVM, nearest neighbors, random forest, logistic regression, etc.
- **Regression**：回归 - Lasso, ridge regression, etc.
- **Clustering**：聚类 - *k*-means, spectral clustering, etc.
- **Dimensionality** reduction：降维 - PCA, feature selection, matrix factorization, etc.
- **Model selection**：模型选择 - Grid search, cross-validation, metrics
- **Preprocessing**：预处理 - Feature extraction, normalization

scikit-learn

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

Import convention

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

Import convention

```
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.ensemble import GradientBoostingRegressor as GBR
from sklearn.ensemble import AdaBoostRegressor as ABR
from sklearn.neighbors import KNeighborsRegressor as KNNR
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.externals import joblib
```

数据分析示例

参考《Python for Data Analysis》(2nd Edition)

Ch14

14.1. 美国政府网站访问数据

文件：'datasets/bitly_usagov/example.txt'

格式：json

14.2. 电影评分及用户数据

文件：'users.dat' 'ratings.dat' 'movies.dat'

本章内容

01 Python语法回顾

02 常用工具包

03 NumPy基础

04 Pandas基础

ndarray

NumPy的主要对象是同种元素的多维数组。这是一个所有的元素都是一种类型、通过一个正整数元组索引的元素表格(通常是元素是数字)。在NumPy中维度(dimensions)叫做轴(axes)，轴的个数叫做秩(rank)。

例如，在3D空间一个点的坐标[1, 2, 3]是一个秩为1的数组，因为它只有一个轴。那个轴长度为3。又例如，在以下例子中，数组的秩为2(它有两个维度)，第一个维度长度为2，第二个维度长度为3.

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

NumPy的数组类被称作**ndarray**。通常被称作数组。注意numpy.array和标准Python库类array.array并不相同，后者只处理一维数组和提供少量功能。

对象属性

重要ndarray**对象属性**有：

- **ndarray.ndim**：数组轴的个数，在python的世界中，轴的个数被称作秩
- **ndarray.shape**：数组的维度。这是一个指示数组在每个维度上大小的整数元组。例如一个n排m列的矩阵，它的shape属性将是(2,3)，这个元组的长度显然是秩，即维度或者ndim属性
- **ndarray.size**：数组元素的总个数，等于shape属性中元组元素的乘积。
- **ndarray.dtype**：一个用来描述数组中元素类型的对象，可以通过创造或指定dtype使用标准Python类型。另外NumPy提供它自己的数据类型。
- **ndarray.itemsize**：数组中每个元素的字节大小。例如，一个元素类型为float64的数组itemsize属性值为8(=64/8),又如，一个元素类型为complex32的数组item属性为4(=32/8).
- **ndarray.data**：包含实际数组元素的缓冲区，通常我们不需要使用这个属性，因为我们总是通过索引来使用数组中的元素

对象属性

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
```

```
>>> a.itemsize
4
>>> a.size
15
>>> type(a)
numpy.ndarray
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
numpy.ndarray
```

创建数组

- 可以使用array函数从常规的Python列表和元组创造数组。所创建的数组类型由原序列中的元素类型推导而来

```
>>> a = np.array( [2,3,4] )
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int32')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

```
>>> a = np.array(1,2,3,4)    # WRONG
```

```
>>> a = np.array([1,2,3,4])  # RIGHT
```

创建数组

- 数组将序列包含序列转化成二维的数组，序列包含序列包含序列转化成三维数组等等。

```
>>> b = np.array( [ (1.5,2,3), (4,5,6) ] )
>>> b
array([[ 1.5,  2. ,  3. ],
       [  4. ,  5. ,  6. ]])
```

- 数组类型可以在创建时显示指定

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

创建数组

- 通常，数组的元素开始都是未知的，但是它的大小已知。因此，NumPy提供了一些使用占位符创建数组的函数。这最小化了扩展数组的需要和高昂的运算代价。
- 函数zeros创建一个全是0的数组，函数ones创建一个全1的数组，函数empty创建一个内容随机并且依赖与内存状态的数组。默认创建的数组类型(dtype)都是float64。

创建数组

```
>>> np.zeros( (3,4) )
array([[0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )                # dtype can also be
specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]] , dtype=int16)
>>> np.empty( (2,3) )
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

创建数组

- 为了创建一个数列，NumPy提供一个类似range的函数返回数组

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )                 # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

当arange使用浮点数参数时，由于有限的浮点数精度，通常无法预测获得的元素个数。因此，最好使用函数linspace去接收我们想要的元素个数来代替用range来指定步长。

np.linspace(start, stop, num=50)

```
>>> np.linspace(0,20,6)
array([  0.,   4.,   8.,  12.,  16.,  20.])
```

打印数组

- 打印一个数组时，NumPy以类似嵌套列表的形式显示它，但是呈以下布局：
  - 最后的轴从左到右打印
  - 次后的轴从顶向下打印
  - 剩下的轴从顶向下打印，每个切片通过一个空行与下一个隔开
- 一维数组被打印成行，二维数组成矩阵，三维数组成矩阵列表。

打印数组

```
>>> # 1d array
>>> a = np.arange(6)
>>> print(a)
[0 1 2 3 4 5]

>>> # 2d array
>>> b = np.arange(12).reshape(4,3)
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
>>> # 3d array
>>> c = np.arange(24).reshape(2,3,4 )
>>> print(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]]]
```

打印数组

- 如果一个数组用来打印太大了，NumPy自动省略中间部分而只打印角落

```
>>> print(np.arange(10000))
[   0    1    2 ..., 9997 9998 9999]
>>>
>>> print(np.arange(10000).reshape(100,100))
[[   0    1    2 ...,   97   98   99]
 [ 100  101  102 ...,  197  198  199]
 [ 200  201  202 ...,  297  298  299]
 ...,
 [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
 [9900 9901 9902 ..., 9997 9998 9999]]
```

如果想禁用NumPy的这种行为并强制打印整个数组，可以设置printoptions参数来更改打印选项。

```
>>> np.set_printoptions(threshold=np.float('inf'))
```

基本运算

- 数组的算术运算是按元素的。新的数组被创建并且被结果填充。

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 ,
-2.62374854])
>>> a<35
array([True, True, False, False], dtype=bool)
```

基本运算

- 不像许多矩阵语言，NumPy中的乘法运算符*指示按元素计算，矩阵乘法可以使用dot函数或创建矩阵对象实现。

```
>>> A = np.array( [[1,1], [0,1]] )  ⟶  array([[1, 1],
                                            [0, 1]])

>>> B = np.array( [[2,0], [3,4]] )  ⟶  array([[2, 0],
                                            [3, 4]])

>>> A*B                         # elementwise
product
array([[2, 0],
       [0, 4]])

>>> np.dot(A,B)                 # matrix product
array([[5, 4],
       [3, 4]])
```

基本运算

- 有些操作符像+=和*=被用来更改已存在数组而不创建一个新的数组。

```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[3.07931765, 3.48256859, 3.91269811],
       [3.54427641, 3.01035951, 3.71245452]])
>>> a = a + b
>>> a
array([[6.07931765, 6.48256859, 6.91269811],
       [6.54427641, 6.01035951, 6.71245452]])
```

基本运算

- 当运算的是不同类型的数组时，结果数组**和更普遍和精确的一致**(这种行为叫做upcast)。

```
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0,np.pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
```

基本运算

- **运算默认应用到数组**好像它就是一个数字组成的列表，无关数组的形状。然而，指定axis参数你可以把运算应用到数组指定的轴上：

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>> b.sum()
66

>>> b.sum(axis=0)                    # sum of each column
array([12, 15, 18, 21])

>>> b.min(axis=1)                    # min of each row
array([0, 4, 8])

>>> b.cumsum(axis=1)                 # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

通用函数(ufunc)

- NumPy提供常见的数学函数如sin,cos和exp等。在NumPy中，这些叫作“通用函数”(ufunc)。在NumPy里这些函数作用按数组的元素运算，产生一个数组作为输出。

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> np.sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([ 2.,  0.,  6.])
```

通用函数(ufunc)

- 更多函数 all, alltrue, any, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, conjugate, corrcoef, cov, cross, cumprod, cumsum, diff, dot, floor, inner, inv, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sometrue, sort, std, sum, trace, transpose, var, vdot, vectorize, where

更多通用函数可参见：https://docs.scipy.org/doc/numpy/reference/routines.html

索引、切片和迭代

- 一维数组可以被索引、切片和迭代，就像列表和其它Python序列。

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000
>>> a
array([-1000,     1, -1000,    27, -1000,   125,   216,   343,   512,   729])
>>> a[ : :-1]                                 # reversed a
array([  729,   512,   343,   216,   125, -1000,    27, -1000,     1, -1000])
>>> for i in a:
        print(i**(1/3),end=',')
```

索引、切片和迭代

- 多维数组可以每个轴有一个索引。这些索引由一个逗号分割的元组给出。

```
>>> def f(x,y):
        return(10*x+y)
>>> b = np.fromfunction(f,(5,4),dtype=np.int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]                    # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]                   # each column in the second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

```
>>> b = np.fromfunction( lambda i, j: 10*i + j, (5,4),dtype=np.int)
```

索引、切片和迭代

```
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
```

- 当少于轴数的索引被提供时，缺失的索引被认为是整个切片：

```
>>> b[-1]                                  # the last row. equivalent to b[-1,:]
array([40, 41, 42, 43])
```

- b[i]中括号中的表达式被当作‘i’和一系列‘:’来代表剩下的轴。NumPy也允许使用“点”像b[i,...]。

- 点(...)代表许多产生一个完整的索引元组必要的分号。**如果x是秩为5的数组(即它有5个轴)**，那么：

- x[1,2, ...] 等同于 x[1,2,:,:,:],
- x[...,3] 等同于 x[:,:,:,:,3]
- x[4, ...,5,:] 等同 x[4,:,:,5,:].

索引、切片和迭代

```
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
```

- 迭代多维数组是就第一个轴而言的：

```
>>> for row in b:
...         print(row)
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

- 然而，如果对每个数组中元素进行运算，我们可以使用**flat属性**，该属性是数组元素的一个迭代器：

```
>>> for element in b.flat:
...         print(element,end=' ')
...
0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43
```

形状操作

- **更改数组的形状**
- 一个数组的形状由它每个轴上的元素个数给出：

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> a
array([[ 5.,  5.,  8.,  4.],
       [ 2.,  9.,  0.,  2.],
       [ 1.,  9.,  9.,  7.]])
>>> a.shape
(3, 4)
```

形状操作

- 一个数组的形状可以被多种命令修改：

```
>>> a.ravel() # flatten the array
array([ 5.,  5.,  8.,  4.,  2.,  9.,  0.,  2.,  1.,  9.,  9.,  7.])
>>> a.shape = (6, 2)
>>> a.transpose()
array([[ 5.,  8.,  2.,  0.,  1.,  9.],
       [ 5.,  4.,  9.,  2.,  9.,  7.]])
>>> a.reshape(4, 3)
array([[ 5.,  5.,  8.],
       [ 4.,  2.,  9.],
       [ 0.,  2.,  1.],
       [ 9.,  9.,  7.]])
```

由ravel()展平的数组元素的顺序通常是“C风格”的，就是说，最右边的索引变化得最快，所以元素a[0,0]之后是a[0,1]。通过reshape()改变形状，也采用同样的顺序。

形状操作

- reshape, ravel, transpose 函数改变参数形状并返回结果
- resize, shape函数改变数组自身

```
>>> a
array([[ 5.,  5.],
       [ 8.,  4.],
       [ 2.,  9.],
       [ 0.,  2.],
       [ 1.,  9.],
       [ 9.,  7.]])
>>> a.resize((2,6))
>>> a
array([[ 7.,  5.,  9.,  3.,  7.,  2.],
       [ 7.,  8.,  6.,  8.,  3.,  2.]])
```

组合(stack)不同的数组

- 几种方法可以沿不同轴将数组堆叠在一起：

```
>>> a = np.floor(10*np.random.random((2,2)))
>>> a
array([[ 1.,  1.],
       [ 5.,  8.]])
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 3.,  3.],
       [ 6.,  0.]])
```

```
>>> np.vstack((a,b))
array([[ 1.,  1.],
       [ 5.,  8.],
       [ 3.,  3.],
       [ 6.,  0.]])
>>> np.hstack((a,b))
array([[ 1.,  1.,  3.,  3.],
       [ 5.,  8.,  6.,  0.]])
```

column_stack和row_stack详见
https://docs.scipy.org/doc/numpy/reference/routines.html

数组分割(split)

- 使用hsplit能将数组沿着它的水平轴分割，或者指定返回相同形状数组的个数，或者指定在哪些列后发生分割:

```
>>> a = np.floor(10*np.random.random((2,12)))
>>> a
array([[ 8.,  8.,  3.,  9.,  0.,  4.,  3.,  0.,  0.,  6.,  4.,  4.],
       [ 0.,  3.,  2.,  9.,  6.,  0.,  4.,  5.,  7.,  5.,  1.,  4.]])
>>> np.hsplit(a,3)   # Split a into 3
[array([[ 8.,  8.,  3.,  9.],
       [ 0.,  3.,  2.,  9.]]),
              array([[ 0.,  4.,  3.,  0.],
       [ 6.,  0.,  4.,  5.]]), array([[ 0.,  6.,  4.,  4.],
       [ 7.,  5.,  1.,  4.]])]
>>> np.hsplit(a,(3,4))   # Split a after the third and the fourth column
[array([[ 8.,  8.,  3.],
       [ 0.,  3.,  2.]]), array([[ 9.],
       [ 9.]]), array([[ 0.,  4.,  3.,  0.,  0.,  6.,  4.,  4.],
       [ 6.,  0.,  4.,  5.,  7.,  5.,  1.,  4.]])]
```

复制和视图

- 当运算和处理数组时，它们的数据有时被拷贝到新的数组，有时不是。具体来看，这有三种情况:
- 1. 完全不拷贝

简单的赋值不拷贝数组对象或它们的数据。

Python 传递不定对象作为参考，所以函数调用不拷贝数组。

```
>>> a = np.arange(12)
>>> b = a            # no new object is created
>>> b is a           # a and b are two names for the same ndarray object
True
>>> b.shape = 3,4    # changes the shape of a
>>> a.shape
(3, 4)
```

复制和视图

- 2.视图(view)和浅复制
- 不同的数组对象分享同一个数据。视图方法创造一个新的数组对象指向同一数据。

```
>>> c = a.view()
>>> c is a
False
>>> c.base is a          # c is a view of the data owned by a
True
>>> c.flags.owndata
False
>>>
>>> c.shape = 2,6        # a's shape doesn't change
>>> a.shape
(3, 4)
>>> c[0,4] = 1234        # a's data changes
```

```
>>> a
array([[   0,    1,    2,    3],
       [1234,    5,    6,    7],
       [   8,    9,   10,   11]])
```


复制和视图

- 2.视图(view)和浅复制

```
>>> a
array([[  0,   1,   2,   3],
       [1234,   5,   6,   7],
       [  8,   9,  10,  11]])
```

- 切片数组返回它的一个视图：

```
>>> s = a[ : , 1:3]     # spaces added for clarity; could also be written "s = a[:,1:3]"
>>> s[:] = 10           # s[:] is a view of s. Note the difference between s=10 and
s[:]=10
>>> a
array([[  0,  10,  10,   3],
       [1234,  10,  10,   7],
       [  8,  10,  10,  11]])
```

复制和视图

- 3.深复制
- 这个复制方法完全复制数组和它的数据。

```
>>> d = a.copy()                          # a new array object with new data is created
>>> d is a
False
>>> d.base is a                           # d doesn't share anything with a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

函数和方法(method)总览

- 创建数组
  - arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, r , zeros, zeros_like
- 转化
  - astype, atleast_1d, atleast_2d, atleast_3d, mat
- 操作
  - array split, column stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack

函数和方法(method)总览

- 询问
  - all, any, nonzero, where
- 排序
  - argmax, argmin, argsort, max, min, ptp, searchsorted, sort
- 运算
  - choose, compress, cumprod, cumsum, inner, fill, imag, prod, put, putmask, real, sum
- 基本统计
  - cov, mean, std, var
- 基本线性代数
  - cross, dot, outer, svd, vdot