易汉博 | 领先的大数据与健康解决方案
Leading solutions for big data and health

Python3 学习教程

易生信培训团队

联系我们: train@ehbio.com

抱歉，转成 PDF 后部分格式问题还未解决，会继续更新调整

2018-05-09

Contents

1 Python 基础 **9**

1.1 交互模式下表达式 9

1.2 Python 中的数据类型：整数（int）、浮点（float）和字符串（str） 10

1.3 字符串的连接和复制 10

1.4 变量 11

1.5 第一小程序 HelloWorld.py 12

1.6 逻辑和比较操作 15

1.6.1 布尔逻辑值 15

1.6.2 比较操作符 15

1.6.3 布尔操作符 and or not 16

1.6.4 布尔操作和比较操作符 16

1.7 控制流 17

1.7.1 if 语句 17

1.7.2 elif 否则如果 18

1.7.3 while 循环 18

1.7.4 break 和 continue 19

1.7.5 for 和 range () 函数 20

1.7.6 range() 函数（开始，停止，步长） 21

1.8 导入模块 . . . 23
1.9 函数：内置函数、自定义函数 . . . 25
1.9.1 函数 print ()，len ()，input ()，int ()，str () 均为内置函数 . . . 25
1.10 局部和全局作用域 . . . 27
1.11 声明为全局变量 global . . . 29
1.12 异常的处理 . . . 30
1.13 列表 . . . 33
1.14 字符串和元组 . . . 42
1.15 引用 . . . 44
1.16 字典键：值对 . . . 46
1.17 字典与列表 . . . 47
1.17.1 keys()、values () 和 items( ) . . . 48
1.18 字符串操作 . . . 51

2 Python 教程 **59**

2.1 背景介绍 . . . 59
2.1.1 编程开篇 . . . 59
2.1.2 为什么学习 Python . . . 60
2.1.3 Python 常用包 . . . 60
2.1.4 怎么学习 Python . . . 60
2.1.5 Python 学习的几个阶段 . . . 61

2.1.6 如何安装 Python 61
2.1.7 如何运行 Python 命令和脚本 62
2.1.8 使用什么编辑器写 Python 脚本 63
2.2 Python 程序事例 63
2.3 Python 语法 65
2.3.1 层级缩进 65
2.3.2 Python 作为计算器的使用 66
2.3.3 变量、数据结构、流程控制 67
2.4 输入输出 95
2.4.1 交互式输入输出 95
2.4.2 文件读写 96
2.5 实战练习（一） 98
2.5.1 背景知识 98
2.5.2 作业 (一) 99
2.6 函数操作 102
2.6.1 作业（二） 107
2.7 模块 107
2.8 命令行参数 108
2.8.1 作业（三） 111
2.9 更多 Python 内容 112

2.9.1 单语句块 . . . 112
2.9.2 列表解析 . . . 112
2.9.3 字典解析 . . . 113
2.9.4 断言 . . . 113
2.9.5 更多字符串方法 . . . 114
2.9.6 lambda, map, filer, reduce (保留节目) . . . 117
2.9.7 **exec, eval (执行字符串 python 语句, 保留节目)** . . . 119
2.9.8 正则表达式 . . . 119
2.10 Python 画图 . . . 123
2.11 Reference . . . 149

3 Python 作图 **150**

3.1 绘图基础 . . . 150
3.1.1 Figure 和 Subplot . . . 150
3.1.2 调整 subplot 周围间距 . . . 152
3.1.3 颜色标记和线型 . . . 156
3.1.4 刻度、标签和图例 . . . 158
3.1.5 添加图例 legend . . . 160
3.1.6 注解 . . . 161
3.1.7 图片保存 . . . 163
3.2 绘图实例 . . . 164

3.2.1 绘制散点图 . . . 164
3.2.2 折线图 . . . 165
3.2.3 直方图 . . . 168
3.2.4 直条图 . . . 169
3.2.5 箱线图 . . . 171
3.2.6 饼图 . . . 175
3.2.7 绘制基因矩阵的热图 . . . 177

4 Python 实战 189

4.1 Python 实战 . . . 189
4.1.1 ID 转换 . . . 189
4.1.2 每条染色体基因数目统计 . . . 192
4.1.3 所有外显子总长度统计 . . . 194
4.2 Python 小技巧 . . . 196

5 Pandas 学习教程 208

5.1 What is pandas . . . 208
5.2 Pandas 读取文件 . . . 209
5.2.1 获取目标文件 . . . 209
5.2.2 查看目标文件内容和格式 . . . 209
5.2.3 读取两列文件 . . . 210

5.2.4 数据表的索引 . . . 211
5.2.5 读取多列文件 . . . 213
5.2.6 选取多列数据 . . . 215
5.2.7 重命名列名字 . . . 216
5.2.8 合并矩阵 . . . 217
5.2.9 矩阵数据提取 . . . 229
5.2.10 读取 META data 文件 . . . 230
5.3 Pandas 写入文件 . . . 238
5.3.1 写入文本文件 . . . 238
5.4 PANDAS 矩阵的小应用 . . . 238
5.5 Seaborn 绘图 . . . 249

6 Python 科学计算 258

6.1 NumPy . . . 258
6.1.1 NumPy 数组 . . . 258
6.1.2 布尔语句和 NumPy 数组 . . . 266
6.1.3 NumPy 读写文件 . . . 268
6.1.4 NumPy 的 Math 模块 . . . 269
6.2 SciPy . . . 269
6.2.1 最优化和最小化 . . . 269
6.2.2 插值 . . . 276

7 易生信 Python 培训练习和考核题目 281

8 生信教程文章集锦 294

8.1 生信宝典 294

8.1.1 系列教程 294

8.1.2 NGS 分析工具评估 295

8.1.3 宏基因组教程 295

8.1.4 系列宣传 295

8.1.5 生信生物知识 296

8.1.6 文献精读 296

8.1.7 Linux 296

8.1.8 CIRCOS 系列 297

8.1.9 R 统计和作图 297

8.1.10 扩增子三步曲 297

8.1.11 宏基因组分析专题 298

8.1.12 NGS 基础 298

8.1.13 癌症数据库 298

8.1.14 Python 299

8.1.15 NGS 软件 299

8.1.16 Cytoscape 网络图 299

8.1.17 分子对接 299

8.1.18 生信宝典之傻瓜式 . . . 300
8.1.19 生信人写程序 . . . 300
8.1.20 小技巧系列 . . . 300
8.1.21 招聘 . . . 300
8.2 宏基因组 . . . 300
8.2.1 精选文章推荐 . . . 301
8.2.2 培训、会议、征稿、招聘 . . . 302
8.2.3 科研经验 . . . 302
8.2.4 软件和数据库使用 . . . 302
8.2.5 扩增子学习三步曲 . . . 303
8.2.6 宏基因组分析专题 . . . 303
8.2.7 R 统计绘图 . . . 303
8.2.8 实验设计与技术 . . . 304
8.2.9 基础知识 . . . 304
8.2.10 必读综述 . . . 304
8.2.11 高分文章套路解读 . . . 305
8.2.12 科普视频-寓教于乐 . . . 306
8.2.13 友军文章汇总推荐 . . . 306

9 公司简介 **308**

1 Python 基础

王绪宁制作

欢迎访问我们的视频课程 https://bioinfo.ke.qq.com。

1.1 交互模式下表达式

```
# 表达式： 值、操作符优先级
2 + 2
```

```
4
```

```
2 + 3 * 5
```

```
17
```

```
( 2 + 3 ) * 5
```

```
25
```

```
23 % 7 # 取余数，判断数据是奇数还是偶数时应用
```

```
2
```

```
23 // 7 #
```

```
3
```

```
# 示例报错，语法要符合规则，像英语语法规则一样。
2+
```

```
  File "<ipython-input-9-d066253e063d>", line 2
    2+
     ^
SyntaxError: invalid syntax
```

1.2 Python 中的数据类型：整数（int）、浮点（float）和字符串（str）

```
type(2)
```

```
int
```

```
type(2.5)
```

```
float
```

```
int(2.4)
```

```
2
```

```
type('Hello, world!') # 字符串用成对的引号（单引号、双引号、三引号）
```

```
str
```

```
# 如果字符串没有引号，则报错
‘hello , world
```

```
  File "<ipython-input-16-874a8e2fccea>", line 2
    ‘hello , world
                ^
SyntaxError: invalid character in identifier
```

1.3 字符串的连接和复制

```
'Hello' + 'world!'
```

```
'helloworld!'
```

```
'hello' + 21 # 错误示例，两个字符串才能相加
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-18-2f37c7cd8411> in <module>()
----> 1 'hello'+21


TypeError: must be str, not int
```

```
'hello' + '21'
```

```
'hello21'
```

```
'Hello' * 5 # 必须是整数，才可复制
```

```
'HelloHelloHelloHelloHello'
```

```
'hello'*5.0 # 错误示例
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-21-8dc053398ad8> in <module>()
----> 1 'hello'*5.0


TypeError: can't multiply sequence by non-int of type 'float'
```

多个字符串的合并使用 + 不是最好的方式，大家想想为什么？有什么更好的方式呢？参见文章为啥我的 Python 这么慢(一)。

1.4 变量

```
# 变量就像一个盒子一样
a = 5
a
```

```
5
```

```
b = 3
b
```

```
3
```

```
a + b
```

```
8
```

```
# 第一次赋值为变量的初始化，再次赋值则被覆写
a = 30
a
```

```
30
```

```
c = 'hello'
c
```

```
'hello'
```

```
c = 'goodbye'
c
```

```
'goodbye'
```

```
# 变量名命名：为清晰表达，驼峰式，下划线式
LookLikeThis = 1
look_like_this = 'a'
```

1.5 第一小程序 HelloWorld.py

```
# 第一小程序的准备
# 注释用 #
# print() # 打印内容显示在屏幕上
print('ehbio')
a = 5
print(a)  # 打印变量
```

```
ehbio
```

```
myName = input()
```

```
a
```

```
myName = input('please type your name！\n')
```

```
please type your name！
ehbio
```

```
#len() 查看字符串的个数
len('hello')
len('hell o')
```

```
6
```

```
## 类型转换的函数
str()
int()
float()
```

```
0.0
```

```
str(8)
```

```
'8'
```

```
int(2.5)
```

```
2
```

```
float(2)
```

```
2.0
```

```
42 == '42' # 字符和数字不同
```

```
False
```

```
# 特殊的值 None, 函数部分会再次提及此值
type(None)
```

```
NoneType
```

```
None == 42
```

```
False
```

```
#This is my second python pargram!

myName=input('What is your name?')
print ('It is  good to meet you,' + myName)
print ('The length of your name is '+ str(len(myName)))

print('What is your age?')
myAge=input()
print ('You will be ' + str( int( myAge)+1 )+ ' in a year.')
```

```
What is your name?wang
It is  good to meet you,wang
The length of your name is 4
What is your age?
2
You will be 3 in a year.
```

```
# 如上是在交互模式下的运行，那写好脚本如何运行呢？
import os
os.getcwd()
```

```
'F:\\mega\\ehbio\\python'
```

```
os.chdir('F:\\mega\\ehbio\\python')
os.getcwd()
```

```
'F:\\mega\\ehbio\\python'
```

```
# 同时展示 IPython 模式下运行程序
%run helloworld.py
```

1.6 逻辑和比较操作

1.6.1 布尔逻辑值

```
a = False
a
```

```
False
```

```
b = True
b
```

```
True
```

1.6.2 比较操作符

```
42 == 21
```

```
False
```

```
42 == '42'
```

```
False
```

```
2 != 1
```

```
True
```

```
a = 'hello'
a == 'hello'
```

```
True
```

```
a == 'goodbye'
```

```
False
```

1.6.3 布尔操作符 and or not

```
True and True
```

```
True
```

```
True & False
```

```
False
```

```
not False
```

```
True
```

```
not not True
```

```
True
```

1.6.4 布尔操作和比较操作符

```
( 3 > 2 ) and ( 5 < 6 )
```

```
True
```

```
( 3 < 2 ) or ( 5 < 6 )
```

```
True
```

1.7 控制流

```
# 条件
name = input('Please enter a name\n')
if name == 'ehbio':
    print('hello ehbio')
else:
    print('You are not ehbio')
```

```
Please enter a name
ehbio
hello ehbio
```

1.7.1 if 语句

if 条件（True or False）：

代码块1

else ：

代码块2

1.7.2 elif 否则如果

elif 条件：

代码块

```
age = int(input('please enter your age!\n'))  # input() 从屏幕中录入的内容，属于字符串类型
if age < 18:
    print('Hi, kids')
elif 40 >= age >= 12:
    print('Hi, young man')
elif age > 40:
    print('Hi, old man')
```

```
please enter your age!
34
Hi, young man
```

1.7.3 while 循环

```
a = 0
if a < 5:
    print('Hello, world')
    a = a + 1
```

```
Hello, world
```

```
a = 0
while a < 5:
    print('Hello, world')
    a = a + 1
```

```
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

```
name = ''
while name != 'Your name':
    name = input('Enter your name\n')
print('Thank you')
```

```
Enter your name
your name
Thank you
```

1.7.4 break 和 continue

```
while True:
    name = input('Please enter your name!\n')
    if name == 'Your name':
        break  # 跳出循环
print('You are right!')
```

```
Please enter your name!
Your name
You are right!
```

```
# 模拟登陆账号
while True:
    name = input('Who are you?\n')
    if name != 'Bob':
        continue  # 将程序跳转到开头
    print('Hello, Bob. What is your password?')
    password = input()
    if password == 'fish':
        break
print('Access granted!')
```

```
Who are you?
Bob
Hello, Bob. What is your password?
fish
Acess granted!
```

1.7.5 for 和 range () 函数

```
# for 和 range () 实现固定的循环次数
for i in range(5):
    print(i)
    print('Hello world')
```

```
0
Hello world
1
Hello world
2
Hello world
3
Hello world
4
Hello world
```

```
# 等价的 while 循环
i = 0
while i < 5:
    print(i)
    print('Hello world')
    i = i + 1
```

```
0
Hello world
1
Hello world
2
Hello world
3
Hello world
4
Hello world
```

```
# 高斯的 1+2+3+...+100=?
total = 0
for i in range(101):
    total = total + i
print(total)
```

```
5050
```

```
# 高斯的 1+2+3+...+100=?
total = 0
for i in range(101):
    total += i
print(total)
```

```
5050
```

1.7.6 range() 函数（开始，停止，步长）

```
for i in range(1, 10, 2):
    print(i)
```

```
1
3
5
7
9
```

```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
```

```
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  __reduce__(...)
|      helper for pickle
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(...)
|      Return a reverse iterator.
|
|  count(...)
|      rangeobject.count(value) -> integer -- return number of occurrences of value
|
|  index(...)
```

```
 |      rangeobject.index(value, [start, [stop]]) -> integer -- return index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

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

```
5
4
3
2
1
0
```

```
# 脑筋急转弯，题目如下
# 100 元钱，5 元铅笔盒，3 元一只笔，0.5 元一块橡皮，将 100 元花完，同时三种物品的个数和为 100，请用编程解决

# 假设铅笔盒为 x 个，笔为 y 只，橡皮 z 个
print("x y z")
for x in range(1, 101):
    for y in range(1, 101):
        for z in range(1, 101):
            if x + y + z == 100 and 5 * x + 3 * y + 0.5 * z == 100:
                print(x, y, z)
```

```
x y z
5 11 84
10 2 88
```

1.8 导入模块

```
import random  # 当多个模块时，用逗号隔开
for i in range(5):
    print(random.randint(1, 10))
```

```
2
9
2
4
10
```

```
import random as rd
for i in range(5):
    print(rd.randint(1, 10))
```

```
2
8
5
7
7
```

```
from random import randint
for i in range(5):
    print(randint(1, 10))
```

```
6
8
5
6
2
```

```
import sys
import os
import math
```

```
# 综合小例子，控制流的结束，执行完成本身即可结束
import sys
while True:
    print('type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()  # 退出 Python 程序
    print('Let us have a short break!')
```

```
type exit to exit.
exit
```

```
An exception has occurred, use %tb to see the full traceback.


SystemExit



C:  \Users\Administrator\Anaconda3\lib\site-  packages\IPython\core\interactiveshell.py:
2870: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

1.9 函数：内置函数、自定义函数

1.9.1 函数 print（），len （） ,input （），int （），str （） 均为内置函数

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
print('a', 'b')
```

```
a b
```

```
# 自定义函数
def sum(a, b):
    c = a + b
    print(c)


sum(3, 4)
```

7

```
def sum(a, b):  # a,b 为自定义函数中的两个参数
    c = a + b
    return c


sum(1, 3)
```

4

```
import random


def getAnswer(answerNumber):
    if answerNumber == 1:
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidely so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'


r = random.randint(1, 9)
fortune = getAnswer(r)
print(r, fortune)
```

8 Outlook not so good

```
# 关于 None 的值
print()  # 打印内容为空，所以为 None，其他编程语言可能为 NA，null 等
```

```
print('cat', 'dog', 'mice')  # 位置参数
```

```
cat dog mice
```

```
print('cat', 'dog', 'mice', sep='*', end='88888')  # end，sep 就是关键字参数
```

```
cat*dog*mice88888
```

1.10 局部和全局作用域

全局作用：函数之外。

局部作用：函数之内。

局部可以访问全局变量，而全局不能使用局部变量。

```
def spam():
    eggs = 1000
    return eggs


spam()
```

```
1000
```

```
eggs # 报错：eggs 未被定义，因为 eggs 是自定义函数 spam() 中的局部变量
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-90-6eaa7316e4ae> in <module>()
----> 1 eggs

NameError: name 'eggs' is not defined
```

```
# 同理局部变量不能使用其他局部变量
def ehbio():
    eggs = 99
    spam()
    print(eggs)


def spam():
    eggs = 0
    return eggs


ehbio()
```

```
99
```

```
spam()
```

```
0
```

```
# 全局变量在局部作用域中使用
def spam():
    print(eggs)


eggs = 28
spam()
```

```
28
```

```
print(eggs)
```

```
28
```

```
# 尽量避免名称相同的局部变量和全局变量
def spam():
    eggs = 'spam local'
    print(eggs)  # 输出 spam local


def bacon():
```

```
    eggs = 'bacon local'
    print(eggs)  # 输出 bacon local
    spam()
    print(eggs)  # 输出 bacon local


eggs = 'global'
bacon()
```

```
bacon local
spam local
bacon local
```

```
print(eggs)
```

```
global
```

1.11 声明为全局变量 global

```
# 在函数内修改全局变量
def spam():
    global eggs  # 声明为全局变量
    eggs = 5


eggs = 7
spam()  # 调用自定义函数 spam()
print(eggs)
```

```
5
```

```
def spam():
    global eggs  # this is the global
    eggs = 'spam'


def bacon():
    eggs = 'bacon'  # this is a local
```

```
def ham():
    print(eggs)  # this is the global


eggs = 42  # this is the global
spam()
print(eggs)
```

```
spam
```

1.12 异常的处理

自定义函数可能报错，报错的处理

```
def spam(divideBy):
    return 21 / divideBy


spam(3)
spam(7)
```

```
3.0
```

```
spam(0)  # 如果是 0，则会报错
```

```
---------------------------------------------------------------------------

ZeroDivisionError                         Traceback (most recent call last)

<ipython-input-107-b3fca5f67bd4> in <module>()
----> 1 spam(0)


<ipython-input-106-f3d08c8e42a5> in spam(divideBy)
      1 def spam(divideBy):
----> 2     return 21/divideBy
      3 spam(3)
      4 spam(7)


ZeroDivisionError: division by zero
```

```
# 此时，我们可以借助 try 和 except 解决
def spam(divideBy):
    try:
        return 21 / divideBy
    except ZeroDivisionError:
        print('Error: invalid argument')


spam(0)
spam(3)
```

```
Error: invalid argument
```

```
7.0
```

```
# 以上为知道报错内容，如果不知道报错内容呢？
def spam(divideBy):
    try:
        return 21 / divideBy
    except:
        print('Error: invalid argument')


spam(0)
```

```
Error: invalid argument
```

```
import traceback


def spam(divideBy):
    try:
        return 21 / divideBy
    except:
        traceback.print_exc()


spam(0)
```

```
Traceback (most recent call last):
```

```
  File "<ipython-input-115-9238a8901fff>", line 4, in spam
    return 21/ divideBy
ZeroDivisionError: division by zero
```

```
# 用 sys 模块
import sys


def spam(divideBy):
    try:
        return 21 / divideBy
    except:
        info = sys.exc_info()
        #print (info)
        print(info[0], ":", info[1])


spam(0)
```

```
<class 'ZeroDivisionError'> : division by zero
```

```
# 只有 6 次机会的猜字游戏
# This is a guess the number game.
import random
secretNumber = random.randint(1, 20)
print('I am thinking of a number between 1 and 20.')

# Ask the player to guess 6 times.
for guessesTaken in range(1, 7):
    print('Take a guess.')
    guess = int(input())

    if guess < secretNumber:
        print('Your guess is too low.')
    elif guess > secretNumber:
        print('Your guess is too high.')
    else:
        break  # This condition is the correct guess!

if guess == secretNumber:
    print('Good job! You guessed my number in ' +
          str(guessesTaken) + ' guesses!')
else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

```
I am thinking of a number between 1 and 20.
Take a guess.
1
Your guess is too low.
Take a guess.
6
Your guess is too low.
Take a guess.
5
Your guess is too low.
Take a guess.
3
Your guess is too low.
Take a guess.
4
Your guess is too low.
Take a guess.
6
Your guess is too low.
Nope. The number I was thinking of was 7
```

```python
# collatz 序列，如果是偶数则打印 n//2， 如果是奇数则打印 3*n+1
def collatz(n):
    print(n)
    if n % 2 == 1 and n > 1:
        collatz(3 * n + 1)
    elif n % 2 == 0:
        collatz(n // 2)


n = input('enter a number:')
n = int(n)
collatz(n)
```

```
enter a number:4
4
2
1
```

1.13 列表

“列表” 是一个值，比如 [1,3,3,‘cat’]

```
type([1, 2, 'a'])
```

```
list
```

```
a = [1, 2, 3, 4, 'a', 'b']
a
```

```
[1, 2, 3, 4, 'a', 'b']
```

```
b = [a, 3, 4]  # 列表可以嵌套列表，字典等
b
```

```
[[1, 2, 3, 4, 'a', 'b'], 3, 4]
```

取列表中单个值

```
a = [i for i in range(5)]
a
```

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

```
a[0]  # 用元素的下标取值
```

```
0
```

```
a[3]
```

```
3
```

```
zoo = ['cat', 'dog', 'rat', 'elephant']
zoo[3]
```

```
'elephant'
```

```
zoo[0:2]
```

```
['cat', 'dog']
```

```
'hello ' + zoo[3]
```

```
'hello elephant'
```

负数下标

```
zoo[-1]
```

```
'elephant'
```

```
a = ['b', 'c', 'd']
zoo = ['cat', 'dog', 'rat', 'elephant']
total = [a, zoo]
```

```
total
```

```
[['b', 'c', 'd'], ['cat', 'dog', 'rat', 'elephant']]
```

```
total[0][1]
```

```
'c'
```

```
total[1][0]
```

```
'cat'
```

zoo[0] 列表和下标 zoo[0:2] 列表和切片

```
zoo[0:2]
```

```
['cat', 'dog']
```

```
zoo[100] # 超出后报错
```

```
IndexError                                Traceback (most recent call last)

<ipython-input-13-034b9400bdc5> in <module>()
----> 1 zoo[100]


IndexError: list index out of range
```

查看列表长度

```
len(zoo)
```

```
4
```

```
# 用下标更改列表的值
zoo = ['cat', 'dog', 'rat', 'elephant']
zoo
```

```
['cat', 'dog', 'rat', 'elephant']
```

```
zoo[0] = 'five'
```

```
zoo
```

```
['five', 'dog', 'rat', 'elephant']
```

列表的连接和复制

```
a = [i for i in range(5)]
a
```

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

```
b = [i for i in range(3, 10, 2)]
b
```

```
[3, 5, 7, 9]
```

```
a + b
```

```
[0, 1, 2, 3, 4, 3, 5, 7, 9]
```

```
a * 3  # 列表的复制
```

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

删除某个元素

```
del a[0] # 删除运行中的某个变量
```

```
a
```

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

```
del a[2]
```

```
a
```

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

```
# 应用列表的实例
bookNames = []
while True:
    print('Enter the name of books' +
          str(len(bookNames) + 1) + 'Or enter nothing to stop')
    name = input()
    if name == '':
        break
    bookNames = bookNames + [name]
print('The books are:\n')
for name in bookNames:
    print(' ' + name)
```

```
enter the name of books1Or enter nothing to stop
lives
enter the name of books2Or enter nothing to stop
python
enter the name of books3Or enter nothing to stop
```

```
linux
enter the name of books4Or enter nothing to stop

The books are:

 lives
 python
 linux
```

列表在循环中的应用

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

```
0
1
2
3
4
```

```
for i in [0, 1, 2, 3, 4]:
    print(i)
```

```
0
1
2
3
4
```

in 和 not in 操作

```
tool = ['python', 'R', 'perl', 'java']
'python' in tool
```

```
True
```

```
'C' not in tool
```

```
True
```

应用列表多重赋值

```
tool = ['python', 'R', 'perl', 'java']
first = tool[0]
second = tool[1]
third = tool[2]
forth = tool[3]
print(first, second, third, forth)
```

```
python R perl java
```

```
# 另一种方式
tool = ['python', 'R', 'perl', 'java']
first, second, third, forth = tool
print(first, second, third, forth)
```

```
python R perl java
```

增强的赋值操作

```
### +=     -=     *=     %=  注意：+= *=可以完成字符串连接和复制
```

```
a = 32
a += 1  # 等价于 a = a + 1
a
```

```
33
```

方法和函数是一回事，每个数据类型都有一些自己的一组方法。列表数据：查找、添加、删除或操作列表值

```
tool = ['python', 'R', 'perl', 'java']
tool.index('R')
```

```
1
```

```
tool.index('perl')
```

```
2
```

```
tool.index('C')  # 没有在 list 中的情况
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-44-385ebd06ddfe> in <module>()
----> 1 tool.index('C')


ValueError: 'C' is not in list
```

```
# 添加
tool = ['python', 'R', 'perl', 'java']
tool.append('C')  # 没执行一次就多一个
tool
```

```
['python', 'R', 'perl', 'java', 'C']
```

```
# 插入
tool = ['python', 'R', 'perl', 'java']
tool.insert(1, 'C')
tool
```

```
['python', 'C', 'R', 'perl', 'java']
```

```
# 插入和添加，只能在列表上用 'str' object has no attribute 'append'
eggs = 'hello'
eggs.append('o')
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-51-f76c80edd988> in <module>()
      1 #插入和添加，只能在列表上用
      2 eggs='hello'
----> 3 eggs.append('o')


AttributeError: 'str' object has no attribute 'append'
```

```
# 插入和添加，只能在列表上用 'str' object has no attribute 'insert'
eggs = 'hello'
eggs.insert(1, 'o')
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-52-21aceb5fda09> in <module>()
      1 #插入和添加，只能在列表上用 'str' object has no attribute 'append'
      2 eggs='hello'
----> 3 eggs.insert(1,'o')


AttributeError: 'str' object has no attribute 'insert'
```

```
# 删除
tool = ['python', 'R', 'perl', 'java']
tool.remove('java')
tool
```

```
['python', 'R', 'perl']
```

```
# 删除不存在的值将报错
tool = ['python', 'R', 'perl', 'java']
tool.remove("C")
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-54-c1e62b8d0701> in <module>()
      1 #删除不存在的值将报错
      2 tool = ['python', 'R', 'perl', 'java' ]
----> 3 tool.remove("C")


ValueError: list.remove(x): x not in list
```

```
# 对列表的值进行排序
lt = [1, 32, 4, 7, 5]
lt.sort()
lt
```

```
[1, 4, 5, 7, 32]
```

```
lt.sort(reverse=True)
lt
```

```
[32, 7, 5, 4, 1]
```

```
lt = [1, 9, 2, 'a']  # 不同类型不可排序,sort() 使用 ASCII 码进行排序，因此小写 a 在 Z 之前
lt.sort()
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-89-69cf80e7b764> in <module>()
      1 lt=[1,9,2,'a']#不同类型不可排序,sort()使用ASCII码进行排序，因此小写a在Z之前
----> 2 lt.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

1.14 字符串和元组

```
name = 'python'
name[0:2]
```

```
'py'
```

```
name[-1]
```

```
'n'
```

```
'p' in name
```

```
True
```

```
'p' not in name
```

```
False
```

```
for i in name:
    print (i)
```

```
p
y
t
h
o
n
```

```
# 元组 tuple, 不可修改
eggs = (1, 2, 'a')
type(eggs)
```

```
tuple
```

```
# 不可修改
eggs[1] = 'python'
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-71-d14153220c16> in <module>()
----> 1 eggs[1]='python'


TypeError: 'tuple' object does not support item assignment
```

```
eggs[1]
```

```
2
```

```
# 元组用 ()，列表 []，注意区分如下情况
type(('hello'))
```

str

```
type(('hello',)) # 逗号在元组中其“重要作用”
```

tuple

```
tool = ['python', 'R', 'perl', 'java']
tuple(tool)
```

```
('python', 'R', 'perl', 'java')
```

```
a = (1, 3, 4)
list(a)
```

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

1.15 引用

```
a = 42
b = a
a = 100
a
```

```
100
```

```
b
```

```
42
```

列表的引用

```
a = [i for i in range(5)]
a
```

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

```
b = a
b
```

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

```
a[0] = 'hello'
a
```

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

```
b
```

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

```
import copy  # 列表的复制
a = ['b', 'c', 'd', 'e']
a
```

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

```
f = copy.copy(a)
f
```

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

```
f[0] = 'python'
```

```
f
```

```
['python', 'c', 'd', 'e']
```

```
a
```

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

deep.copy() 如果列表中包含了列表，使用 deepcopy()

```
a = [1, 2, 4, 6, [7, 9]]
b = copy.copy(a)
b
```

```
[1, 2, 4, 6, [7, 9]]
```

```
a[-1][0] = 70
b
```

```
[1, 2, 4, 6, [70, 9]]
```

```
a = [1, 2, 4, 6, [7, 9]]
b = copy.deepcopy(a)
b
```

```
[1, 2, 4, 6, [7, 9]]
```

```
a[-1][0] = 111
a
```

```
[1, 2, 4, 6, [111, 9]]
```

```
b
```

```
[1, 2, 4, 6, [7, 9]]
```

1.16 字典键：值对

```
myTool = {'first': 'python', 'second': 'R', 'third': 'linux', 'forth': 'perl'}
myTool
```

```
{'first': 'python', 'forth': 'perl', 'second': 'R', 'third': 'linux'}
```

```
myTool['first']
```

```
'python'
```

1.17 字典与列表

```
# 默认字典是不排序的，不能应用下标或者切片取出，而列表是有序列的
a = [1, 2, 3]
b = [2, 1, 3]
a == b
```

```
False
```

```
myTool1 = {'first': 'python', 'second': 'R', 'third': 'linux', 'forth': 'perl'}
myTool2 = {'first': 'python', 'third': 'linux', 'second': 'R', 'forth': 'perl'}
myTool1 == myTool2
```

```
True
```

```
# 取值通过键
myTool['second']
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-27-ac22a2f1a4f0> in <module>()
      1 #取值通过键
----> 2 myTool['second']


NameError: name 'myTool' is not defined
```

```
# 综合小程序
birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4'}

while True:
    print('Enter a name: (blank to quit)')
    name = input()
    if name == '':
        break

    if name in birthdays:
        print(birthdays[name] + ' is the birthday of ' + name)
    else:
```

```
        print('I do not have birthday information for ' + name)
        print('What is their birthday?')
        bday = input()
        birthdays[name] = bday
        print('Birthday database updated.')
```

```
Enter a name: (blank to quit)
Alice
Apr 1 is the birthday of Alice
Enter a name: (blank to quit)
wang
I do not have birthday information for wang
What is their birthday?
Oct 1
Birthday database updated.
Enter a name: (blank to quit)
wang
Oct 1 is the birthday of wang
Enter a name: (blank to quit)
```

1.17.1 keys()、values () 和 items( )

```
myTool = {'first': 'python', 'second': 'R', 'third': 'linux', 'forth': 'perl'}
for k in myTool.keys():
    print(k)
```

```
first
second
third
forth
```

```
for v in myTool.values():
    print(v)
```

```
python
R
linux
perl
```

```
myTool.items()
```

```
dict_items([('first', 'python'), ('second', 'R'), ('third', 'linux'), ('forth', 'perl')])
```

```
for k, v in myTool.items():
    print('Key: ' + k + ' Value: ' + v)
```

```
Key: first Value: python
Key: second Value: R
Key: third Value: linux
Key: forth Value: perl
```

```
# 检查字典中是否存在键
'first' in  myTool
```

```
True
```

```
'hello' in myTool
```

```
False
```

```
# get()，两个参数分别为“键”，如果没有该键，返回的值
myTool = {'first': 'python', 'second': 'R', 'third': 'linux', 'forth': 'perl'}
myTool.get('first', 0)
```

```
'python'
```

```
myTool.get('hello', 0)
```

```
0
```

```
# 不使用 get() 时，如果该键不存在，则报错
myTool['hello']
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)
```

```
<ipython-input-14-72a3aef6d23a> in <module>()
      1 #不使用get()时
----> 2 myTool['hello']


KeyError: 'hello'
```

setdefault() 如果没有该键，则添加

```
# 写循环实现
myTool = {'first': 'python', 'second': 'R', 'third': 'linux', 'forth': 'perl'}
if 'fifth' not in myTool:
    myTool['fifth'] = 'C'
myTool
```

```
{'fifth': 'C',
 'first': 'python',
 'forth': 'perl',
 'second': 'R',
 'third': 'linux'}
```

```
# setdefault()
myTool = {'first': 'python', 'second': 'R', 'third': 'linux', 'forth': 'perl'}
myTool.setdefault('fifth', 'C')
```

```
'C'
```

```
myTool
```

```
{'fifth': 'C',
 'first': 'python',
 'forth': 'perl',
 'second': 'R',
 'third': 'linux'}
```

```
sequece = 'life is short, I use python'
count = {}

for character in sequece:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

print(count)
```

```
{'l': 1, 'i': 3, 'f': 1, 'e': 2, ' ': 5, 's': 3, 'h': 2, 'o': 2, 'r': 1, 't': 2, ',': 1, 'u': 1
```

```
sequece = 'life is short, I use python'
count = {}

for character in sequece:
    count[character] = count.get(character,0) + 1

print(count)
```

```
{'l': 1, 'i': 2, 'f': 1, 'e': 2, ' ': 5, 's': 3, 'h': 2, 'o': 2, 'r': 1, 't': 2, ',': 1, 'I': 1
```

```
# 简单序列计数
sequece = 'TCGAAATTCGGGCATGGAG'
count = {}

for character in sequece:
    count.setdefault(character, 0)
    count[character] = count[character] + 1
for k, v in count.items():
    print(k + ' ' + str(v))
```

```
T 4
C 3
G 7
A 5
```

1.18 字符串操作

```
# 一个错误开始
'This is Bob's book'
```

```
  File "<ipython-input-26-9caaf6a74585>", line 2
    'This is Bob's book'
                 ^
SyntaxError: invalid syntax
```

双引号字符串可以使用双引号，双引号之间可以使用单引号

```
"This is Bob's book"
```

```
"This is Bob's book"
```

转义符 \n 换行 \' 单引号 \" 双引号 \\倒斜杠 \t 制表符

```
print("Life is short,\n I use python")
```

```
Life is short,
 I use python
```

原始字符串：忽略转义符号

```
print(r"Life is short,\n I use python")
```

```
Life is short,\n I use python
```

三重引号多行字符串

```
print('''Life is short,\n I use python!
do you like python
yes''')
```

```
Life is short,
 I use python!
do you like python
yes
```

多行注释 “”” “””

```
"""
简单序列计数
多行注释

"""
sequece = 'TCGAAATTCGGGCATGGAG'
count = {}
```

```
for character in sequece:
    count.setdefault(character, 0)
    count[character] = count[character] + 1
for k, v in count.items():
    print(k + ' ' + str(v))
```

```
T 4
C 3
G 7
A 5
```

字符串切片

```
a = "life is short, i use python"
a
```

```
'life is short, i use python'
```

```
a[1:5]
```

```
'ife '
```

```
a[-5:-3]
```

```
'yt'
```

```
a[-3:]
```

```
'hon'
```

```
'python' in a
```

```
True
```

字符串方法

```
a = 'Hello world'
a.upper()
```

```
'HELLO WORLD'
```

```
b = 'NI HAO'
b.lower()
```

```
'ni hao'
```

```
b.isupper()
```

```
True
```

```
a.isupper()
```

```
False
```

```
# 简单介绍了验证码为什么不用区分大小写
feeling = input('How are you?\n')
if feeling.lower == 'great':
    print('I feel great too.')
else:
    print('I hope you happy')
```

```
How are you?
Great
I hope you happy
```

is.X 字符串方法 isalpha() 是否只含字母 isalnum() 是否只是字母或数字 isdecimal() 是否只有数字 isspace() 是否只有空格制表符换行 istitle() 是否字符串为大写开头，后面均为小写字母

```
a = 'b1'
a.isalpha()
```

```
False
```

```
a = 'b c'
a.isalpha()
```

```
False
```

```
a = 'bc1'
a.isalnum()
```

```
True
```

```
a = '1a'
a.isalnum()
```

```
True
```

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters or numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters or numbers.')
```

```
Enter your age:
2
Select a new password (letters or numbers only):
+
Passwords can only have letters or numbers.
Select a new password (letters or numbers only):
2e
```

startswith() 和 endswith()

```
seq = '>0001\nTCGATTACGG'
if seq.startswith('>'):
    print(seq)
```

```
>0001
TCGATTACGG
```

join() 和 split()

```
','.join(['i', 'love', 'python'])
```

```
'i,love,python'
```

```
'***'.join(['i', 'love', 'python'])
```

```
'i***love***python'
```

```
"linux R perl C python".split()
```

```
['linux', 'R', 'perl', 'C', 'python']
```

```
'i***love***python'.split('***')
```

```
['i', 'love', 'python']
```

文本对齐 rjust() ljust() center()

```
'hello'.rjust(10)
```

```
'     hello'
```

```
'hello'.rjust(20,'*')
```

```
'***************hello'
```

```
'hello'.center(20,'-')
```

```
'-------hello--------'
```

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))


picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

```
---PICNIC ITEMS--
sandwiches..    4
apples......   12
cups........    4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches..........     4
apples..............    12
cups................     4
cookies.............  8000
```

strip(),rstrip(),lstrip() 删除空白字符

```
spam = '    hello    '
spam
```

```
'    hello    '
```

```
spam.strip()
```

```
'hello'
```

```
spam.rstrip()
```

```
'    hello'
```

```
spam.lstrip()
```

```
'hello    '
```

```
a = 'Hello world, welcome to python world'
a.strip('d')
```

```
'Hello world, welcome to python worl'
```

FINISH

2 Python 教程

陈同 (chentong_biology@163.com)

欢迎来到 Python 的世界，本教程将带你遨游 `Python`，领悟 `Python` 的魅力。本教程专注于帮助初学者，尤其是生物信息分析人员快速学会 `Python` 的常用功能和使用方式，因此只精选了部分 `Python` 的功能，请额外参考 Python 经典教程A byte of python和它的中文版 来更好的理解 *Python*. 本文档的概念和文字描述参考了 A byte of python(中文版)，特此感谢。

本教程在 `2017` 年 `12` 月 `25` 日更新到 `Python3` 版本。

欢迎访问我们的视频课程 https://bioinfo.ke.qq.com。

2.1 背景介绍

2.1.1 编程开篇

A：最近在看什么书？

B：编程。

A：沈从文的那本啊。

B：……

C：最近在学一门新语言，Python。

D：那是哪个国家的语言？

C：……

2.1.2 为什么学习 Python

- 语法简单

  Python 语言写作的程序就像自然语言构建的伪代码一样，“所见即所想”。读 `Python` 代码就像读最简单的英文短文一样，写 `Python` 代码比写英文文章都要简单，“所想即所写”。很多刚学习 `Python` 的朋友都觉得很不可思议，原来怎么想怎么写出来就对了。

- 功能强大

  现在程序语言的发展已经很成熟，每一种程序语言都能实现其它程序语言的全部功能。因此就程序语言本身来讲，功能都相差不大。`Python` 语言的功能强大在于其活跃的社区和强大的第三方模块支持，使其作为科学计算的能力越来越强。

- 可扩展性好

  能与 C 完美的融合，加快运行速度。可用加速模块有 `Cython`, `PyPy`, `Pyrex`, `Psyco` 等.

2.1.3 Python 常用包

1. 科学计算 `Numpy`, `SciPy` (也是安装 python 包的拦路虎直到有了conda)
2. 类比于 R 的数据框操作包 Pandas
3. 可视化工具 `Seaborn` (配合 pandas), `matplotlib` (类比 MATLAB), `plotly` (交互式绘图), `ggplot` (类比 ggplot2)
4. 网站开发 `web.py`, `Django`, `Flask`
5. 任务调度和流程管理 `Airflow` (pipeline 首选)
6. 机器学习 `scikit-learn` (经典), `PyML`, `Tensorflow` (谷歌释放), `pylearn2`, `Orange` (图形界面的机器学习包)
7. 网页抓取 `Beautiful Soup`，`requests`,
8. 可重复编程 `Jupyter`
9. 正则表达式 `re`

2.1.4 怎么学习 Python

编程就像拼乐高，需要我们知道每个组分的特征以便在需要时可以使用，也需要脑袋中有个蓝图知道每一步要做什么，二者结合，就可以拼出你想要的世界。

在我看来，学习编程是学以致用，学习方式是硬着头皮去读，去写。

- 多读经典书籍

  首先从概念和理论上了解程序设计或 Python 程序设计，多读。书读百遍其义自见。

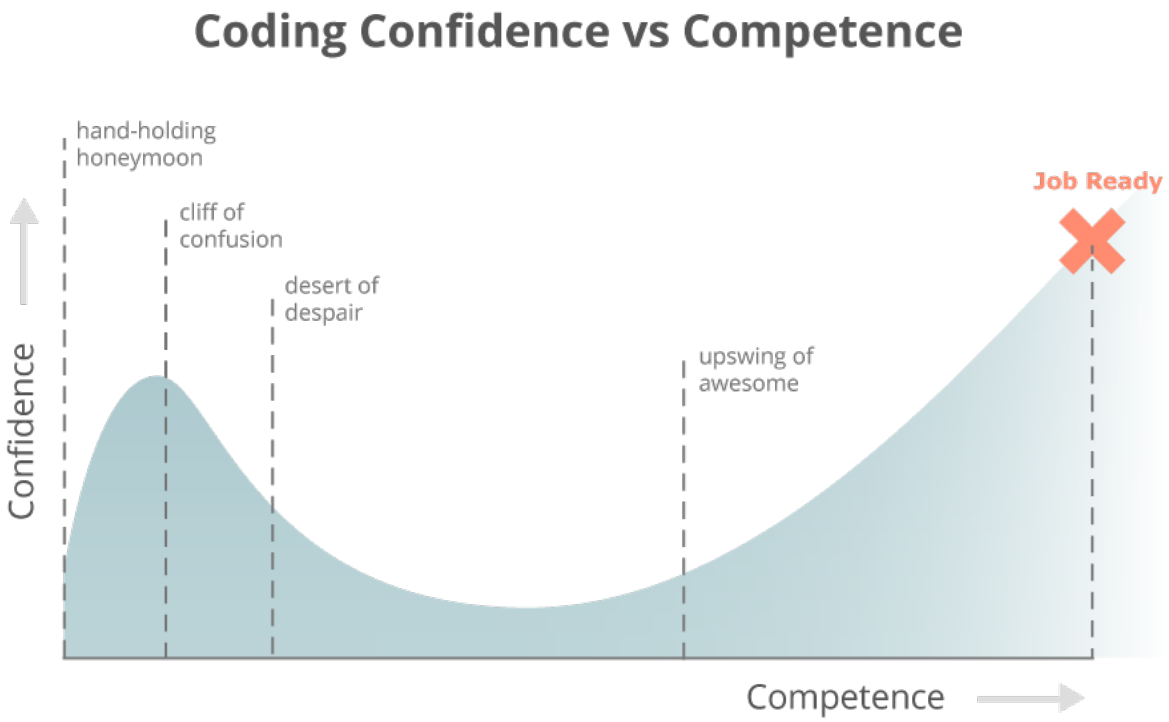
- 多做练习

  任何练习题都可以，先易后难。如果能找到专业相关的，更好。

- 多读代码

  多读优秀的代码，矫正自己的习惯和思维。

2.1.5 Python 学习的几个阶段



(图例 “编程信心与能力”：纵轴为信心值，横轴为能力水平，虚线从左至右依次分割出手牵手蜜月期、混沌悬崖、绝望沙漠、令人兴奋的上升期四个阶段，第 5 条虚线标志着工作准备就绪)

- 读文档是蜜月期，读读就过去，谁都会。
- 写程序是混沌悬崖，知道是这么回事，就是写不出来；
- 调程序是绝望沙漠，怎么看自己写的都对，就是编译器不开眼；
- 程序正确了就是兴奋期，万里长征迈出又一步。

2.1.6 如何安装 Python

Python 社区有很多功能很好的包，但逐个安装需要解决繁杂的依赖关系。通常我会推荐安装已经做好的集成包，一劳永逸的解决后续问题。Anaconda是最优先推荐的分发包，集成了常用的数值计算、图形处理、可视化等工具包如 `IPython`, `matplotlib`, `numpy`, `scipy`, 而且设定了更简单的安装 Python 模块的方法，可以节省大量的安装时间。

2.1.6.1 Python 常用包

1. 科学计算 `Numpy`, `SciPy` (也是安装 python 包的拦路虎直到有了conda)
2. 类比于 R 的数据框操作包 Pandas
3. 可视化工具 `Seaborn` (配合 pandas), `matplotlib` (类比 MATLAB), `plotly` (交互式绘图), `ggplot` (类比 ggplot2)
4. 网站开发 `web.py`, `Django`, `Flask`
5. 任务调度和流程管理 `Airflow` (pipeline 首选)
6. 机器学习 `scikit-learn` (经典), `PyML`, `Tensorflow` (谷歌释放), `pylearn2`, `Orange` (图形界面的机器学习包)
7. 网页抓取 `Beautiful Soup`，`requests`,
8. 可重复编程 `Jupyter`
9. 正则表达式 `re`

2.1.7 如何运行 Python 命令和脚本

- 对于初学者，本手册推荐直接在 `Jupyter Notebook` 下学习 `Python` 命令和脚本。我们这套教程也是用 `Jupyter Notebook` 写作而成，里面的代码可以随时修改和运行，并能同时记录你的脚本和输出，符合现在流行的 “可重复性计算” 的概念。
  - Linux/Unix 用户直接在终端 (Terminal) 进入你的目标文件夹 `cd /working_dir`[回车]，然后在终端输入 `Jupyter notebook`[回车] 即可启动 `Jupyter notebook`。
  - Windows 用户可以新建一个 `Jupyter_notebook.bat` 文件 (新建一个 txt 文件，写入内容后修改后缀为`.bat`。若不能修改后缀，请 Google 搜索 “Window 是如何显示文件扩展名”)，并写入以下内容（注意把前两行的盘符和路径替换为你的工作目录），双击即可运行。

    ```
    D:
    cd PBR_training
    jupyter notebook
    pause
    ```

  - `Jupyter notebook` 启动后会打开默认的浏览器（需要在图形用户界面下工作），这时可以新建或打开相应路径下的 ipynb 文件。
- 对于 LInux 或 Unix 用户，直接在终端输入 `python` 然后回车即可打开交互式 `python` 解释器，如下图所示。在这个解释器了敲入任何合法的 `python` 语句即可执行。此外，所有的命令还可以存储到一个文件一起执行，如下图所示。我们有一个包含 `python` 程序的文件 `test.py`，我们只要在终端输入 `python test.py` 并回车就可以运行这个文件。同时我们也可在终端通过输入 `chmod 755 test.py` 赋予程序 `test.py` 可执行权限，并在终端输入`./test.py` 运行 `Python` 脚本。更多 Linux 下的高级使用和 Linux 命令使用请见教程 Bash_training-chinese.ipynb。

```
@diamond:~$ python
Python 2.6.7 (r267:88850, Mar  1 2012, 14:28:44)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello, you"
Hello, you
>>>
>>>
@diamond:~$ cat test.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
print "Hello, you"
chentong@diamond:~$ python test.py
Hello, you
diamond:~$
```

- 对于 Windows 用户，可以通过 “Windows 键 +R” 调出 “Run” 窗口并输入 “cmd” 打开 Windows 命令解释器，输入 `python` 即可打开交互式 `python` 解释器。同时也可以双击安装后的软件的快捷方式打开图形界面的 `Python` 解释器，可以处理交互式命令和导入 Python 文件并执行。
- 对于交互式 `Python` 解释器，在使用结束后，通过键盘组合键 `Ctrl-d` (Linux/Unix) 或 `Ctrl-z` (Windows) 关闭。

2.1.8 使用什么编辑器写 Python 脚本

在你学成之后，可能主要操作都在服务器完成，而且日常工作一般会以脚本的形式解决。我个人推荐使用Vim来写作 Python 脚本。

Linux 下 `vim` 的配置文件可从我的 github下载，Windows 版可从我的百度云 下载。

2.2 Python 程序事例

```
## 假如我们有如下 FASTA 格式的文件，我们想把多行序列合并为一行，怎么做？
for line in open("data/test2.fa"):
    print(line.strip())
```

```
>NM_001011874 gene=Xkr4 CDS=151-2091
gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccggggaaggaagccagggcg
aggcgaggaggtggcgggaggaggagacagcagggacaggTGTCAGATAAAGGAGTGCTCTCCTCCGCTG
CCGAGGCATCATGGCCGCTAAGTCAGACGGGAGGCTGAAGATGAAGAAGAGCAGCGACGTGGCGTTCACC
CCGCTGCAGAACTCGGACAATTCGGGCTCTGTGCAAGGACTGGCTCCAGGCTTGCCGTCGGGGTCCGGAG
```

```
>NM_001195662 gene=Rp1 CDS=55-909
AAGCTCAGCCTTTGCTCAGATTCTCCTCTTGATGAAACAAAGGGATTTCTGCACATGCTTGAGAAATTGC
AGGTCTCACCCAAAATGAGTGACACACCTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCA
AGTTCCTTCCCCTCGCCATTCAAATATCACTCATCCTGTAGTGGCTAAACGCATCAGTTTCTATAAGAGT
GGAGACCCACAGTTTGGCGGCGTTCGGGTGGTGGTCAACCCTCGTTCCTTTAAGACTTTTGACGCTCTGC
TGGACAGTTTATCCAGGAAGGTACCCCTGCCCTTTGGGGTAAGGAACATCAGCACGCCCCGTGGACGACA
CAGCATCACCAGGCTGGAGGAGCTAGAGGACGGCAAGTCTTATGTGTGCTCCCACAATAAGAAGGTGCTG
>NM_011283 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCAC
ACAAACTATTTACTCTTTTCTTCGTAAGGAAAGGTTCAACTTCTGGTCTCACCCAAAATGAGTGACACAC
CTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCAAGTTCCTTCCCCTCGCCATTCAAATAT
CACTCATCCTGTAGTGGCTAAACGCATCAGTTTCTATAAGAGTGGAGACCCACAGTTTGGCGGCGTTCGG
GTGGTGGTCAACCCTCGTTCCTTTAAGACTTTTGACGCTCTGCTGGACAGTTTATCCAGGAAGGTACCCC
TGCCCTTTGGGGTAAGGAACATCAGCACGCCCCGTGGACGACACAGCATCACCAGGCTGGAGGAGCTAGA
GGACGGCAAGTCTTATGTGTGCTCCCACAATAAGAAGGTGCTGCCAGTTGACCTGGACAAGGCCCGCAGG
CGCCCTCGGCCCTGGCTGAGTAGTCGCTCCATAAGCACGCATGTGCAGCTCTGTCCTGCAACTGCCAATA
TGTCCACCATGGCACCTGGCATGCTCCGTGCCCCAAGGAGGCTCGTGGTCTTCCGGAATGGTGACCCGAA
>NM_0112835 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCAC
ACAAACTATTTACTCTTTTCTTCGTAAGGAAAGGTTCAACTTCTGGTCTCACCCAAAATGAGTGACACAC
CTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCAAGTTCCTTCCCCTCGCCATTCAAATAT
CACTCATCCTGTAGTGGCTAAACGCATCAGTTTCTATAAGAGTGGAGACCCACAGTTTGGCGGCGTTCGG
GTGGTGGTCAACCCTCGTTCCTTTAAGACTTTTGACGCTCTGCTGGACAGTTTATCCAGGAAGGTACCCC
TGCCCTTTGGGGTAAGGAACATCAGCACGCCCCGTGGACGACACAGCATCACCAGGCTGGAGGAGCTAGA
GGACGGCAAGTCTTATGTGTGCTCCCACAATAAGAAGGTGCTGCCAGTTGACCTGGACAAGGCCCGCAGG
CGCCCTCGGCCCTGGCTGAGTAGTCGCTCCATAAGCACGCATGTGCAGCTCTGTCCTGCAACTGCCAATA
TGTCCACCATGGCACCTGGCATGCTCCGTGCCCCAAGGAGGCTCGTGGTCTTCCGGAATGGTGACCCGAA
```

```python
aDict = {}
for line in open('data/test2.fa'):
    if line[0] == '>':
        key = line.strip()
        aDict[key] = []
    else:
        aDict[key].append(line.strip())
##----------------------------------------
for key, valueL in list(aDict.items()):
    print(key)
    print(''.join(valueL))
```

```
>NM_001011874 gene=Xkr4 CDS=151-2091
gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccggggaaggaagccagggcgaggcgaggaggtggcgggaggagga
>NM_001195662 gene=Rp1 CDS=55-909
AAGCTCAGCCTTTGCTCAGATTCTCCTCTTGATGAAACAAAGGGATTTCTGCACATGCTTGAGAAATTGCAGGTCTCACCCAAAATGAGTGACAC
>NM_011283 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCACACAAACTATTTACTCTTTTCTTCGTA
```

```
>NM_0112835 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCACACAAACTATTTACTCTTTTCTTCGT
```

2.3 Python 语法

2.3.1 层级缩进

- 合适的缩进。空白在 Python 中是很重要的，它称为缩进。在逻辑行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组。这意味着同一层次的语句必须有相同的缩进。每一组这样的语句称为一个块。通常的缩进为 4 个空格, 在 `Jupyter Notebook` 中为一个 `Tab` 键。

从下面这两个例子可以看出错误的缩进类型和对应的提示。* “unexpected indent” 表示在不该出现空白的地方多了空白，并且指出问题出在第三行 (line 3)。* “expected an indented block” 表示应该有缩进的地方未缩进，也指出了问题所在行。* “unindent does not match any outer indentation level” 表示缩进出现了不一致，问题通常会在指定行及其前面的行。

```
print(" 不合适的缩进会引发错误，b 前不该有的缩进")
a = 'No indent'
 b = ' 我前面有个空格……'
```

```
  File "<ipython-input-2-ff110daefedf>", line 3
    b = '我前面有个空格……'
    ^
IndentationError: unexpected indent
```

```
print(" 不合适的缩进，print 是 for 的子语句，应该有缩进，却漏掉了")
a = [1,2,3]

for i in a:
print " 我应该被缩进，我从属于 for 循环!!!\n"
```

```
  File "<ipython-input-3-84bbc644895e>", line 5
    print "我应该被缩进，我从属于for循环!!!\n"
        ^
IndentationError: expected an indented block
```

```
a = [1,2, 3]
if a:
    for i in a:
        print(i)
       print(i + 1,
```

```
            " 为什么我的缩进跟其它行不一样呢，我的空格被谁吃了？")
        print(i + 1,
              " 为什么我的缩进跟其它行不一样呢，谁给了我个空格？")
```

```
  File "<tokenize>", line 5
    print(i + 1,
    ^
IndentationError: unindent does not match any outer indentation level
```

2.3.2 Python 作为计算器的使用

Python 中可以进行基本的数学运算，与小学中学过的一样，加减乘除取余数等，需要注意的是运算符的优先级。

```
2 + 2
```

```
4
```

```
2 + 3 +5
```

```
10
```

```
2 + 3 * 5
```

```
17
```

```
(2 + 3) * 5
```

```
25
```

```
## 整除
23 // 7
```

```
3
```

```
## 取余数
23 % 7
```

```
2
```

```
print("Hello, Python!")
```

2.3.2.1 第一个小程序

```
Hello, Python!
```

```
myname = input("Your name: ")
print("Hello", myname)
```

```
Your name: CT
Hello CT
```

2.3.3 变量、数据结构、流程控制

我们先看一个动图展示内存中变量的赋值、存储、值的改变和程序的运行。

点击访问动图 http://www.ehbio.com/ehbio_resource/python_tutor.gif

- 常量，指固定的数字或字符串，如 `2`, `2.9`, `"Hello world"` 等。
- 变量，存储了数字或字符串的事物称为变量，它可以被赋值或被修改。简单的可以理解为变量是一个盒子，你可以把任何东西放在里面，通过盒子的名字来取出盒子内的东西。
  - 数值变量：存储了数的变量。
  - 字符串变量：存储了字符串的变量。字符串变量的名字最好不为 `str`，可以使用 `aStr`。
  - 列表 (list): list 是处理一组有序项目的数据结构，即你可以在一个列表中存储一个 序列的项目。假想你有一个购物列表，上面记载着你要买的东西，你就容易理解列表了。只不过在你的购物表上，可能每样东西都独自占有一行，而在 Python 中，你在每个项目之间用逗号分割。列表中的项目应该包括在方括号中，这样 Python 就知道你是在指明一个列表。一旦你创建了一个列表，你可以添加、删除或是搜索列表中的项目。由于你可以增加或删除项目，我们说列表是 可变的数据类型，即这种类型是可以被改变的。列表变量的名字最好不为 `list`，可以使用 `aList`。
  - 元祖 (tuple)：元组和列表十分类似，但是不可修改。元组通过圆括号中用逗号分割的项目定义。元组通常用在使语句或用户定义的函数能够安全地采用一组值的时候，即被使用的元组的值不会改变。元组变量的名字最好不为 `tuple`，可以使用 `aTuple`。
  - 集合 (Set)：也与列表类似，但是元素不可重复。通常用来去除重复、求交集、并集等。而且集合的查询速度远远快于列表，可以用来提高运算速度。
  - 字典 (dict): 字典类似于你通过联系人名字查找地址和联系人详细情况的地址簿，即，我们把键（名字）和值（详细情况）联系在一起。注意，键必须是唯一的，就像如果有两个人恰巧同名的话，你无法找到正确的信息。多个键可以

指向同一个值。当一个键需要指向多个值时，这些值需要放在列表、元组或字典里面。注意，你只能使用不可变的对象（字符串，数字，元组）来作为字典的键，但是可以用不可变或可变的对象作为字典的值。键值对在字典中以这样的方式标记：d = {key1：value1, key2：value2 }。注意它们的键/值对用冒号分割，而各个对用逗号分割，所有这些都包括在花括号中。记住字典中的键/值对是没有顺序的。如果你想要一个特定的顺序，那么你应该在使用前自己对它们排序。列表变量的名字最好不为 `dict`，可以使用 `aDict`。

- 序列：列表、元组、字符串都是一种序列格式。同时还可以使用 range 来产生序列。序列的两个主要操作时索引操作和切片操作。

- 标示符

  - 变量的名字被称为标示符。标识符对大小写敏感，第一个字符必须是字母表中的字母（大写或小写）或者一个下划线(_)，其它部分额外包含数字。有效的标示符有: `abc`, `_abc`, `a_b_2`, `__23` 等。无效的标示符有: `2a`, `3b`。
  - 标示符最好不使用 Python 内置的关键字，如 `str`, `list`, `int`, `def`, `split`, `dict` 等。
  - 标示符最好能言词达意，即展示变量的类型，又带有变量的实际含义。如 `line` 表示文件的一行，`lineL` 表示存有从文件读入的每一行的列表。

- 控制流

  - `if` 语句

  `if` 语句用来检验一个条件，如果条件为真，我们运行一块语句（称为 `if`-块），否则我们处理另外一块语句（称为 `else`-块）。`else` 从句是可选的。如果有多个条件，中间使用 `elif`。

  举个例子："买五个包子，如果看到卖西瓜的，买一个"——最后程序猿买了一个包子" 买包子 `= 5` `if` 看到卖西瓜的： 买包子 `= 1`

  - `For` 语句

  for..in 是一个循环语句，它在一序列的对象上递归, 即逐一使用队列中的每个项目。

  - `While` 语句

  只要在一个条件为真的情况下，`while` 语句允许你重复执行一块语句。`while` 语句是所谓 循环语句的一个例子。`while` 语句有一个可选的 `else` 从句。

  - `break` 语句是用来 终止循环语句的，即哪怕循环条件没有成为 False 或序列还没有被完全递归，也停止执行循环语句。

  一个重要的注释是，如果你从 `for` 或 `while` 循环中 终止，任何对应的循环 else 块将不执行。

  - `continue` 语句被用来告诉 `Python` 跳过当前循环块中的剩余语句，然后 继续进行下一轮循环。
  - 逻辑运算符 `and`, `or`, `not`。

```
## 变量名命名：为清晰表达，驼峰式，下划线式
LookLikeThis = 1
look_like_this = 'a'
```

```
type(2)
```

2.3.3.1 Python 中的数据类型：整数（int）、浮点（float）和字符串（str）

```
int
```

```
type(2.5)
```

```
float
```

```
type("Hello, everyone")
```

```
str
```

```
type([1,2,3])
```

```
list
```

```
## 类型转换函数
str()
int()
float()
```

```
str(2)
```

```
'2'
```

```
## 字符和数字不同
42 != "42"
```

```
True
```

```
42 == int("42")
```

```
True
```

```
## This is my first python program!

myName = input("Hello, what is your name?")
print('It is  good to meet you,' + myName)
print('The length of your name is', str(len(myName)))

myAge = input('What is your age?')
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

```
Hello, what is your name?CT
It is  good to meet you,CT
The length of your name is 2
What is your age?20
You will be 21 in a year.
```

```
a = False
a
```

2.3.3.2 逻辑值和比较操作

```
False
```

```
b = True
b
```

```
True
```

```
a = "False"
a
```

```
'False'
```

```
42 == 40
```

```
False
```

```
42 == "42"
```

```
False
```

```
30 == 30
```

```
True
```

```
## 注意赋值 (一个 =) 和比较 (==) 的区别
a = 'hello'

a == 'hello'
```

```
True
```

2.3.3.2.1 布尔操作符 and, or, not

- 逻辑与 and：所有为真才为真
- 逻辑或 or：所有为假才为假
- 逻辑非 not: 真变假，假变真

```
(3 > 2) and (5 < 6)
```

```
True
```

```
(3 < 2) or (5 < 6)
```

```
True
```

```
not True
```

```
False
```

2.3.3.3 控制流 if 条件（True or False）：

代码块1

elif 条件：

代码块

else ：

代码块2

```
## 条件
name = input('Please enter a name and click Enter\n')
if name == 'ehbio':
    print ('hello ehbio')
else :
    print ('You are not ehbio')
```

```
Please enter a name and click Enter
CT
You are not ehbio
```

```
a = 0
while a < 5:
    print('Hello, world')
    a = a + 1
```

2.3.3.4 While 循环

```
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

```
print(" 数值变量")
a = 5    # 注意等号两边的空格，为了易于辨识，操作符两侧最好有空格，数量不限
```

```
print(a)

print()
print("The type of a is", type(a))

##print " 这是保留节目，通常判断变量的类型使用的不是 type 是 isinstance."
##print "a is an int, ", isinstance(a,int)

## 再次赋值就是覆盖
a = 6
print(a)
```

2.3.3.5 数值变量操作

数值变量

```
5

The type of a is <class 'int'>
6
```

```
## 判断
print(" 比较数值的大小")
a = 5

## 注意大于号两边的空格，为了易于辨识，操作符两侧最好有空格，数量不限
if a > 4:
    print("a is larger than 4.")
elif a == 4:
    print("a is equal to 4.")
else:
    print("a is less than 4")
```

比较数值的大小

```
a is larger than 4.
```

```
print(" 给定数值变量 a 和 b 的值，通过判断和重新赋值使得 a 的值小，b 的值大")
a = 5
b = 3

if a > b:
    a,b = b,a
##------------------
print(a)
print(b)
```

给定数值变量a和b的值，通过判断和重新赋值使得a的值小，b的值大

```
3
5
```

```
print('''# 数值运算，符合传统的优先级，需要使用括号来改变优先级，
和小学学的数学一模一样!!''')
a = 5
b = 3

print("a + b =", a + b)
print("a * b =", a * b)
print("a / b =", a / b)  # 1
print("2 * (a+b) =", 2 * (a + b))
print(" 取余数： a % b =", a % b)
print(" 取余数是很好的判断循环的地方，因为每个固定的周期余数就会循环一次")
```

```
#数值运算，符合传统的优先级，需要使用括号来改变优先级，
和小学学的数学一模一样！！
a + b = 8
a * b = 15
a / b = 1.6666666666666667
2 * (a+b) = 16
取余数： a % b = 2
取余数是很好的判断循环的地方，因为每个固定的周期余数就会循环一次
```

```
3 / 0
```

```
---------------------------------------------------------------------------

ZeroDivisionError                         Traceback (most recent call last)

<ipython-input-17-2b706ee9dd8e> in <module>()
----> 1 3 / 0


ZeroDivisionError: division by zero
```

```
print(" 字符串变量")

## 注意引号的配对
a = "Hello, welcome to Python"
```

```
##a = 123
##a = str(a)

print("The string a is:", a)
print()

## 占位符
print("The length of this string <%s> is %d" % (a, len(a)))
print()

print("The type of a is", type(a))
```

2.3.3.6 字符串变量操作

字符串变量

```
The string a is: Hello, welcome to Python

The length of this string <Hello, welcome to Python> is 24

The type of a is <class 'str'>
```

```
a = " 大事赖独断而不赖众谋"
print("The string a is:", a)
print()

## len 函数：获得字符串长度
print("The length of this string <%s> is %d" % (a, len(a)))
print()
```

```
The string a is: 大事赖独断而不赖众谋

The length of this string <大事赖独断而不赖众谋> is 10
```

```
a = "Hello, welcome to Python"

print(" 取出字符串的第一个字符、最后一个字符、中间部分字符")
print("The first character of a is %s\n" % a[0])

print("The first five characters of a are %s\n" % a[0:5])

print("The last character of a is %s\n" % a[-1])
print("The last two characters of a are %s\n" % a[-2:])
print("The last character of a is %s\n" % a[len(a) - 1])
print("\n这部分很重要啊，字符串的索引和切片操作是及其常用的。")
```

取出字符串的第一个字符、最后一个字符、中间部分字符

```
The first character of a is H

The first five characters of a are Hello

The last character of a is n

The last two characters of a are on

The last character of a is n
```

这部分很重要啊，字符串的索引和切片操作是及其常用的。

```
a = "oaoaoaoa"

print(" 遍历字符串")
for i in a:
    print(i)

print()
print("The index of first <o> is: ",a.find('o'))
print("The index of first <c> is: ",a.find('c'))

print(" 输出符合特定要求的字符的位置")
print()
pos = 0
for i in a:
    pos += 1
    if i == 'o':
        print(pos)
    #------------------
##-----------------------
print('''\n知道吗？不经意间我们写出了 Python 的
一个内置的标准函数 find 或者 index, 而且功能还更强大''')

print('''\n自己尝试实现程序语言内建的函数是学习程序语言
的很好方法。''')
```

遍历字符串

```
o
a
o
a
o
a
```

```
o
a

The index of first <o> is:  0
The index of first <c> is:  -1
```

输出符合特定要求的字符的位置

```
1
3
5
7
```

知道吗？不经意间我们写出了Python的
一个内置的标准函数find或者index,而且功能还更强大

自己尝试实现程序语言内建的函数是学习程序语言
的很好方法。

```
print(" 我们看看用内置函数如何找到所有 o 的位置\n")
a = "oaoaoaoa"

print(" 内置函数 find 只能确定最先出现的 o 的位置")
pos = a.find('o')

print(" 因此，我们要在发现 o 之后，截取其后的字符串，再执行 find 操作")
while 1:
    print(pos + 1)
    new = a[pos + 1:].find('o')
    if new == -1:
        break
    pos = new + pos + 1
## help(str)
```

我们看看用内置函数如何找到所有 o 的位置

内置函数find只能确定最先出现的 o 的位置
因此，我们要在发现 o 之后，截取其后的字符串，再执行find操作

```
1
3
5
7
```

```
print()
print(" 利用 split 分割字符串\n")
str1 = "a b c d e f g"
strL = str1.split(' ')
```

```
print(strL)
print("\n使用 split 命令就可以把字符串分成列表了，想取用哪一列都随便你了。")
## 使用下面的命令查看可以对字符串进行的操作
## help(str)
```

利用split分割字符串

```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

使用split命令就可以把字符串分成列表了，想取用哪一列都随便你了。

```
print(" 字符串的连接\n")

a = "Hello"
b = "Python"
c = a + ', ' + b
print(c)
print("\n原来字符串相加就可以连起来啊！\n")
print(''' 注意，这不是连接字符串最好的方式。
考虑到字符串是不可修改的，每次连接操作都是新开辟一个内存空间，
把字符串存到里面，这样的连接操作执行几十万次会很影响运行速度。''')
```

字符串的连接

```
Hello, Python
```

原来字符串相加就可以连起来啊！

注意，这不是连接字符串最好的方式。
考虑到字符串是不可修改的，每次连接操作都是新开辟一个内存空间，
把字符串存到里面，这样的连接操作执行几十万次会很影响运行速度。

关于字符串链接为什么不推荐使用 +，文章为啥我的 Python 这么慢 (一)给出了一个很好的演示例子。

```
print(''' 去除字符串中特定的字符。通常我们在文件中读取的一行都包含换行符，
linux 下为\\n \n''')  # \\转义字符

a = "oneline\n"
print("Currently, the string <a> is **", a, "**. \
\n The length of string <a> is **", len(a), "**. \
\n 我为什么换到下一行了？\n")

a = a.strip()
```

```
print("Currently, the string <a> is **", a, "**. \
\n The length of string <a> is **", len(a), "**. \
\n 删掉了换行符后，少了个字符，而且没换行！\n")

a = a.strip('o')
print("Currently, the string <a> is **", a, "**. \
\n The length of string <a> is **", len(a), "**. \
又少了个字符!! \n")

a = a.strip('one')
print("Currently, the string <a> is **", a, "**. \
\n The length of string <a> is **", len(a), "**. \
又少字符了!! \n")
```

去除字符串中特定的字符。通常我们在文件中读取的一行都包含换行符，
linux下为\n

```
Currently, the string <a> is ** oneline
 **.
 The length of string <a> is ** 8 **.
 我为什么换到下一行了？

Currently, the string <a> is ** oneline **.
 The length of string <a> is ** 7 **.
 删掉了换行符后，少了个字符，而且没换行！

Currently, the string <a> is ** neline **.
 The length of string <a> is ** 6 **. 又少了个字符！！

Currently, the string <a> is ** li **.
 The length of string <a> is ** 2 **. 又少字符了！！
```

```
print(" 字符串的替换\n")

a = "Hello, Python"
b = a.replace("Hello", "Welcome")
print(" 原始字符串是:", a)
print()
print(" 替换后的字符串是:", b)
print()

c = a.replace("o", "O")
print(c)
print(" 所有的 o 都被替换了！\n")

print(" 如果我只替换第一个 o 怎么办呢？\n")
```

```
c = a.replace("o", "O", 1)
print(c)
```

字符串的替换

原始字符串是: Hello, Python

替换后的字符串是: Welcome, Python

```
HellO, PythOn
所有的o都被替换了！
```

如果我只替换第一个o怎么办呢？

```
HellO, Python
```

```
## 中文替换也可以
a = " 忙处事为，常向闲中先检点，过举自稀；动时念想，预从静里密操持，非心自息。"
print(a.replace('；', '\n'))
```

```
忙处事为，常向闲中先检点，过举自稀
动时念想，预从静里密操持，非心自息。
```

```
print(" 字符串帮助，查看字符串可用方法")
help(str)
```

```
print(" 大小写判断和转换")
a = 'Sdsdsd'
print("All elements in <%s> is lowercase: %s" % (a, a.islower()))
print("Transfer all elments in <%s> to lowerse <%s>" % (a, a.lower()))
print("Transfer all elments in <%s> to upperse <%s>" % (a, a.upper()))
```

大小写判断和转换

```
All elements in <Sdsdsd> is lowercase: False
Transfer all elments in <Sdsdsd> to lowerse <sdsdsd>
Transfer all elments in <Sdsdsd> to upperse <SDSDSD>
```

```
print(" 这个是个保留节目，有兴趣的看，无兴趣的跳过不影响学习")
print(''' 字符串是不可修改的，
同一个变量名字赋不同的只实际是产生了多个不同的变量。
不同的变量名字赋同样的值，用于比较时相等，但引用不同的区域''')
```

```
b = "123456"
## print b
print("The memory index of b is", id(b))
for i in range(1, 15, 2):
    b = b + '123456'
    # print b
    print("The memory index of b is", id(b))
```

这个是个保留节目，有兴趣的看，无兴趣的跳过不影响学习
字符串是不可修改的，
同一个变量名字赋不同的只实际是产生了多个不同的变量。
不同的变量名字赋同样的值，用于比较时相等，但引用不同的区域

```
The memory index of b is 139844870936200
The memory index of b is 139844868463728
The memory index of b is 139844870954056
The memory index of b is 139844863857088
The memory index of b is 139844863857088
The memory index of b is 139845221506544
The memory index of b is 139844869671408
The memory index of b is 139844868660840
```

```
print(" 字符串转数组")
print()
str1 = "ACTG"
print(list(str1))
a = list(str1)

print()
print(" 字符串转数组之后就可以逆序了，得到其反向序列")
print()
a.reverse()
print(''.join(a))
```

字符串转数组

```
['A', 'C', 'T', 'G']
```

字符串转数组之后就可以逆序了，得到其反向序列

```
GTCA
```

```
print(" 数字字符串转数值")
a = '123'
print(a + '1', int(a) + 1)
```

```
a = '123.5'

print()
## print a + 1
print(float(a) + 1)
print(''' 从文件或命令行参数中取出的数字都是字符串形式出现，
做四则运算时要先用 int 或 float 转换。''')
```

数字字符串转数值
1231 124

124.5
从文件或命令行参数中取出的数字都是字符串形式出现，
做四则运算时要先用int 或 float转换。

```
print(" 字符串倍增")
a = "ehbio "
a * 4
```

字符串倍增

'ehbio ehbio ehbio ehbio '

```
a * 0
```

''

```
## 倍增不可以是小数
a * 3.1
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-21-c65dd4dac397> in <module>()
----> 1 a * 3.1


TypeError: can't multiply sequence by non-int of type 'float'
```

```
## 模拟登陆账号，用户名 Bob，密码 fish
while True:
    name = input('Who are you?\n> ')
    if name != 'Bob':
        continue  # 将程序跳转到开头
    password = input('Hello, Bob. What is your password? (password: fish)\n> ')
    if password == 'fish':
        break # 跳出所在循环或最内层循环
print('Acess granted!')
```

2.3.3.7 break 和 continue

```
Who are you?
> CT
Who are you?
> Bob
Hello, Bob. What is your password? (password: fish)
> sdsds
Who are you?
> Bob
Hello, Bob. What is your password? (password: fish)
> fish
Acess granted!
```

```
## 如果只给一个参数，则是从 0-给定参数 (不包括)，步长为 1
for i in range(4):
    print(i)
```

2.3.3.8 for range (获取一系列数)

```
0
1
2
3
```

```
## 1: start; 10: end (不包括); 2: step
for i in range(1,10,2):
    print(i)
```

```
1
3
5
7
9
```

```
## 步长也可以为负值，从大到小
for i in range(10,1,-2):
    print(i)
```

```
10
8
6
4
2
```

高斯计算 1-100 的加和。

```
## 高斯的 1+2+3+...+100=?
total = 0

## 参数是 101，为什么呢？
for i in range(1,101):
    total = total + i
print(total)
```

```
5050
```

```
## 高斯优化后的
end = 100
sum_all = int((1+end) * end / 2)
##else:
##    sum_all = end * (end -1 ) / 2 + end
print(sum_all)
```

```
5050
```

脑筋急转弯，题目如下:

现有 100 元钱，需要买 100 个物品，其中铅笔盒单价 5 元，笔单价 3 元，橡皮单价 0.5 元，怎么组合可以把 100 元花完，同时三种物品的个数和为 100，请用编程解决。

```
## 纯暴力解法
for x in range(0, 101):
    for y in range(0, 101):
        for z in range(0, 101):
            if x + y + z == 100 and 5 * x + 3 * y + 0.5 * z == 100:
                print(x, y, z)
```

```
0 20 80
5 11 84
10 2 88
```

```
## 优化后的暴力解法
## 限定 box 和 pen 的最大数目，也就是全部钱只买他们，最多能买多少个？
max_box = int(100 / 5) + 1
max_pen = int(100 / 3) + 1
for box_num in range(max_box):
    # 需要买的物品总数是固定的，
    for pen_num in range(max_pen - box_num):
        eraser_num = 100 - box_num - pen_num
        if 5 * box_num + 3 * pen_num + 0.5 * eraser_num == 100:
            print((box_num, pen_num, eraser_num))
```

```
(0, 20, 80)
(5, 11, 84)
(10, 2, 88)
```

```
print("# 构建一个数组")
aList = [1, 2, 3, 4, 5]
print(aList)
print("\n数组可以用下标或区域进行索引\n")
print("The first element is %d." % aList[0])
print()
print("The last element is %d." % aList[-1])
print()
print("The first two elements are", aList[:2])
print("\n数组索引和切片操作与字符串是一样一样的，而且都很重要。")
```

2.3.3.9 列表操作

```
#构建一个数组
[1, 2, 3, 4, 5]
```

数组可以用下标或区域进行索引

```
The first element is 1.

The last element is 5.

The first two elements are [1, 2]
```

数组索引和切片操作与字符串是一样一样的，而且都很重要。

```
aList = []
print("#append: 向数组中增加元素")
aList.append(6)
print(aList)

print("\n#extend: 向数组中增加一个数组")
print()
bList = ['a', 'b', 'c']
aList.extend(bList)
print(aList)

aList + bList
```

```
#append: 向数组中增加元素
[6]

#extend: 向数组中增加一个数组

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

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

```
aList = [1, 2, 3, 4, 3, 5]
print(" 在数组中删除元素")
aList.remove(3)   # 只删除第一个匹配的 3
print()
```

```
print(aList)

aList.pop(3)   # 移除元素的下标为 3 的字符
print()
print(aList)
print('''\npop 和 remove 是不一样的，remove 是移除等于给定值的元素，
pop 是移除给定位置的元素\n''')
```

在数组中删除元素

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

[1, 2, 4, 5]
```

pop和remove是不一样的，remove是移除等于给定值的元素，
pop是移除给定位置的元素

```
aList = [1, 2, 3, 4, 5]

print("# 遍历数组的每个元素")
print()
for ele in aList:
    print(ele)

print("\n# 输出数组，并输出大于 3 的元素")
print()

for ele in aList:
    if ele > 3:
        print(ele)
```

#遍历数组的每个元素

```
1
2
3
4
5
```

#输出数组中大于3的元素

```
4
5
```

```
aList = [i for i in range(30)]
aList
aList = [i for i in range(30) if i % 2 ==0]
aList
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

```
aList = [i for i in range(30)]
print("# 输出数组中大于 3, 且小于 10 的元素")
print()

for ele in aList:
    if ele > 3 and ele < 10:  # 逻辑与，当两个条件都符合时才输出
        print(ele)
```

```
#输出数组中大于3,且小于10的元素

4
5
6
7
8
9
```

```
aList = [i for i in range(30)]
print("# 输出数组中大于 25, 或小于 5 的元素")
print()

for ele in aList:
    if ele > 25 or ele < 5:  # 逻辑或，当两个条件满足一个时就输出
        print(ele)
```

```
#输出数组中大于25,或小于5的元素

0
1
2
3
4
26
27
28
29
```

```
aList = [i for i in range(30)]
print("# 输出数组中不大于 3 的元素")
print()

for ele in aList:
    # 逻辑非，当不符合给定条件时才输出。
    # 对于这个例子就是 ele 不大于 3 时才输出，相当于 if ele <= 3:
    if not ele > 3:
        print(ele)
```

#输出数组中大于3,且小于10的元素

```
0
1
2
3
```

```
aList = [1, 2, 3, 4, 5]
print('\t'.join(aList)) #wrong
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-33-193d4492669b> in <module>()
      1 aList = [1, 2, 3, 4, 5]
----> 2 print('\t'.join(aList)) #wrong


TypeError: sequence item 0: expected str instance, int found
```

```
print(" 连接数组的每个元素（每个元素必须为字符串）")
aList = [1, 2, 3, 4, 5]
## print '\t'.join(aList) #wrong

print(aList)
aList = [str(i) for i in aList]
print(aList)
print('\t'.join(aList))

print(':'.join(aList))
print(''.join(aList))

print('''\n先把字符串存到列表，再使用 join 连接，
是最合适的连接大量字符串的方式''')
```

连接数组的每个元素（每个元素必须为字符串）

```
[1, 2, 3, 4, 5]
['1', '2', '3', '4', '5']
1   2   3   4   5
1:2:3:4:5
12345
```

先把字符串存到列表，再使用join连接，
是最合适的连接大量字符串的方式

```
aList = [1, 2, 3, 4, 5]

print(" 数组反序")
aList.reverse()
print(aList)

print(" 数组元素排序")
aList.sort()
print(aList)

## print "lambda 排序，保留节目"
##aList.sort(key=lambda x: x*(-1))
## print aList
```

数组反序

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

数组元素排序

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

```
aList = [[1,2], [3,4], [2,1]]
aList.sort()
aList
```

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

```
aList.sort(key=lambda x: x[1])
aList
```

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

```
print(" 构建一个集合")
aSet = set([])
```

```
aSet = set([1, 2, 3])
print(aSet)

print(" 增加一个元素")
aSet.add(4)
print(aSet)
aSet.add(3)
print(aSet)
```

2.3.3.10 集合操作

构建一个集合
{1, 2, 3}
增加一个元素
{1, 2, 3, 4}
{1, 2, 3, 4}

```
print(" 采用转换为集合去除列表中的重复元素")
aList = [1, 2, 1, 3, 1, 5, 2, 4, 3, 3, 6]
print(aList)
print(set(aList))
print(list(set(aList)))
```

采用转换为集合去除列表中的重复元素
[1, 2, 1, 3, 1, 5, 2, 4, 3, 3, 6]
{1, 2, 3, 4, 5, 6}
[1, 2, 3, 4, 5, 6]

```
print("\n使用 range，产生一系列的字符串\n")
for i in range(16):
    if i % 4 == 0:
        print(i)
print("\n通过指定步长产生 4 的倍数的数\n")
for i in range(0, 16, 4):
    print(i)
```

2.3.3.11 Range 使用

使用range，产生一系列的字符串

```
0
4
8
12
```

通过指定步长产生4的倍数的数

```
0
4
8
12
```

```
print("# 构建一个字典")
aDict = {}
aDict = {1: 2, 3: 4, 'a': ['b','d'], 'd': 'c',
         'e': {1:2}}

print(" 打印字典")
print(aDict)

print(" 向字典中添加键值对")
aDict[5] = 6
aDict['e'] = 'f'
print(aDict)
```

2.3.3.12 字典操作

```
#构建一个字典
打印字典
{1: 2, 3: 4, 'a': ['b', 'd'], 'd': 'c', 'e': {1: 2}}
向字典中添加键值对
{1: 2, 3: 4, 'a': ['b', 'd'], 'd': 'c', 'e': 'f', 5: 6}
```

```
print()
aDict = {1: 2, 3: 4, 'a': 'b', 'd': 'c'}
print(" 输出字典的键值对 (key-value)")
for key, value in list(aDict.items()):
    print(key, value)
```

```
输出字典的键值对(key-value)
```

```
1 2
3 4
a b
d c
```

```
print(" 有序输出字典的键值对 (key-value)")
aDict = {'1': 2, '3': 4, 'a': 'b', 'd': 'c'}
keyL = list(aDict.keys())
print(keyL)

## Python3 中不支持 int 型和 str 型的比较
## 需要先都转换为 str 型
## keyL = [str(i) for i in keyL]
keyL.sort()
print(keyL)
for key in keyL:
    print(key, aDict[key])
```

```
有序输出字典的键值对(key-value)
['1', '3', 'a', 'd']
['1', '3', 'a', 'd']
1 2
3 4
a b
d c
```

```
print(" 字典的 value 可以是一个列表")
a = 'key'
b = 'key2'
aDict = {}
print(aDict)
aDict[a] = []
print(aDict)
aDict[a].append(1)
aDict[a].append(2)
print(aDict)
aDict[b] = [3, 4, 5]

print()
for key, subL in list(aDict.items()):
    print(key)
    for item in subL:
        print("\t%s" % item)

print(" 这个在存取读入的文件时会很有用的，下面的实战练习会用到这个。")
```

字典的value可以是一个列表

```
{}
{'key': []}
{'key': [1, 2]}

key
    1
    2
key2
    3
    4
    5
```

这个在存取读入的文件时会很有用的，下面的实战练习会用到这个。

字典可以用中文做 Key，中文做 value。

```
aDict = {' 菜根谭': ' 事理因人言而悟者，有悟还有迷，总不如自悟之了了；意兴从外境而得者，有得还有失，总不如自得之
print(aDict[' 菜根谭'])
```

事理因人言而悟者，有悟还有迷，总不如自悟之了了；意兴从外境而得者，有得还有失，总不如自得之休休。

```
print(" 字典的 value 也可以是字典")
a = 'key'
b = 'key2'
aDict = {}
print(aDict)
aDict[a] = {}
print(aDict)
aDict[a]['subkey'] = 'subvalue'
print(aDict)
aDict[b] = {1: 2, 3: 4}

##aDict[(a,b)] = 2
##aDict['a'] = 2
##aDict['b'] = 2

print()
for key, subD in list(aDict.items()):
    print(key)
    for subKey, subV in list(subD.items()):
        print("\t%s\t%s" % (subKey, subV))

print("\n这个在存取读入的文件时会很有用的，下面的实战练习会用到这个。")
```

字典的value也可以是字典

```
{}
{'key': {}}
{'key': {'subkey': 'subvalue'}}

key
    subkey  subvalue
key2
    1   2
    3   4
```

这个在存取读入的文件时会很有用的，下面的实战练习会用到这个。

2.4 输入输出

2.4.1 交互式输入输出

在很多时候，你会想要让你的程序与用户（可能是你自己）交互。你会从用户那里得到输入，然后打印一些结果。我们可以分别使用 raw_input 和 print 语句来完成这些功能。

```
a = input("Please input a string and type enter\n> ")

print("The string you typed in is: ", a)
```

```
Please input a string
> sheng xin bao dian
The string you typed in is:  sheng xin bao dian
```

```
print(" 这是一个保留例子，仅供玩耍\n")
lucky_num = 5
c = 0

while True:
    b = int(input("Please input a number to check if you are \
lucky enough to guess right: \n"))
    if b == lucky_num:
        print("\nYour are so smart!!! ^_^ ^_^")
        #-----------------------------------------------------
    #---------------------------------------------------------
    else:
        print("\nSorry, but you are not right. %>_<%")
        while 1:
            c = input("Do you want to try again? [Y/N] \n")
            if c == 'Y':
```

```
                try_again = 1
                break
            elif c == 'N':
                try_again = 0
                break
            else:
                print("I can not understand you, please check your input. \n")
                continue
        #------------------------------------------------------
        if try_again:
            print("\nHere comes another run. Enjoy!\n")
            continue
        else:
            print("\nBye-bye\n")
            break
```

这是一个保留例子，仅供玩耍

```
Please input a number to check if you are lucky enough to guess right:
4

Sorry, but you are not right. %>_<%
Do you want to try again? [Y/N]
Y

Here comes another run. Enjoy!

Please input a number to check if you are lucky enough to guess right:
3

Sorry, but you are not right. %>_<%
Do you want to try again? [Y/N]
N

Bye-bye
```

2.4.2 文件读写

文件读写是最常见的输入和输出操作。你可以实用 `file` 或 `open` 来实现。

2.4.2.1 文件路径 读写文件时若没有指定文件路径，默认为当前目录。因此需要知道当前所在的目录，然后判断要读取的文件是否在当前目录。

```
import os
os.getcwd()
## os.chdir("path")
```

```
'/MPATHB/ct/ipython/notebook'
```

```
print(" 新建一个文件")

context = '''The best way to learn python contains two steps:
1. Rember basic things mentionded here masterly.

2. Practise with real demands.
'''

print(" 以写入模式 (w) 打开一个文件并命名为 (Test_file.txt)")
fh = open("Test_file.txt", "w")
print(context, file=fh)
## fh.write(context)
fh.close()  # 文件操作完成后必须关闭文件句柄
```

新建一个文件
以写入模式(w)打开一个文件并命名为(Test_file.txt)

```
with open("Test_file.txt", "w") as fh:
    print(context, file=fh)
```

```
print(" 以只读模式 (r) 读入一个名为 (Test_file.txt) 的文件")

print()

for line in open("Test_file.txt"):
    print(line)
```

以只读模式(r)读入一个名为(Test_file.txt)的文件

```
The best way to learn python contains two steps:

1. Rember basic things mentionded here masterly.



2. Practise with real demands.
```

仔细看看上面的输出结果，看上去很空，空行比较多。

```
print(''' 避免中间空行的输出。
从文件中读取的每一行都带有一个换行符，
而 Python 的 print 默认会在输出结束时加上换行符，
因此打印一行会空出一行。为了解决这个问题，有下面两套方案。''')

print("end=''")
print()

for line in open("Test_file.txt"):
    print(line, end='')

print()

print(" 去掉每行自身的换行符")
for line in open("Test_file.txt"):
    print(line.strip())
```

```
避免中间空行的输出。
从文件中读取的每一行都带有一个换行符，
而Python的print默认会在输出结束时加上换行符，
因此打印一行会空出一行。为了解决这个问题，有下面两套方案。
end=''

The best way to learn python contains two steps:
1. Rember basic things mentionded here masterly.

2. Practise with real demands.


去掉每行自身的换行符
The best way to learn python contains two steps:
1. Rember basic things mentionded here masterly.

2. Practise with real demands.
```

2.5 实战练习（一）

2.5.1 背景知识

1. FASTA 文件格式

>seq_name_1

sequence1

>seq_name_2

sequence2

2. FASTQ 文件格式

? 1:N:0:AGTCAA

TCTGTGTAGCCNTGGCTGTCCTGGAACTCACTTTGTAGACCAGGCTGGCATGCA

•

BCCFFFFFFHH#4AFHIJJJJJJJJJJJJJJJJJIJIJJJJJGHIJJJJJJJJJ

2.5.2 作业 (一)

1. 给定 FASTA 格式的文件 (test1.fa 和 test2.fa)，写一个程序 `cat.py` 读入文件，并输出到屏幕
   - 用到的知识点
     - open(file)
     - for .. in loop
     - print
     - the amazng , or strip() function
2. 给定 FASTQ 格式的文件 (test1.fq), 写一个程序 `cat.py` 读入文件，并输出到屏幕
   - 用到的知识点
     - 同上
3. 写程序 `splitName.py`, 读入 test2.fa, 并取原始序列名字第一个空格前的名字为处理后的序列名字，输出到屏幕
   - 用到的知识点
     - split
     - 字符串的索引
   - 输出格式为：

```
>NM_001011874
gcggcggcgggcgagcgggcgctggagtaggagctg.......
```

4. 写程序 `formatFasta.py`, 读入 test2.fa，把每条 FASTA 序列连成一行然后输出
   - 用到的知识点
     - join
     - strip
   - 输出格式为:

```
>NM_001011874
gcggcggcgggc......TCCGCTG......GCGTTCACC......CGGGGTCCGGAG
```

5. 写程序 `formatFasta-2.py`, 读入 test2.fa，把每条 FASTA 序列分割成 80 个字母一行的序列
   - 用到的知识点
     - 字符串切片操作
     - range
   - 输出格式为

```
>NM_001011874
gcggcggcgc.(60个字母).TCCGCTGACG #(每行80个字母)
acgtgctacg.(60个字母).GCGTTCACCC
ACGTACGATG(最后一行可不足80个字母)
```

6. 写程序 `sortFasta.py`, 读入 test2.fa, 并取原始序列名字第一个空格前的名字为处理后的序列名字，排序后输出
   - 用到的知识点
     - sort
     - dict
     - aDict[key] = []
     - aDict[key].append(value)
7. 提取给定名字的序列
   - 写程序 `grepFasta.py`, 提取 fasta.name 中名字对应的 test2.fa 的序列，并输出到屏幕。
   - 写程序 `grepFastq.py`, 提取 fastq.name 中名字对应的 test1.fq 的序列，并输出到文件。
     - 用到的知识点
       * print >>fh, or fh.write()
       * 取模运算，4 % 2 == 0
8. 写程序 `screenResult.py`, 筛选 test.expr 中 foldChange 大于 2 的基因并且 padj 小于 0.05 的基，可以输出整行或只输出基因名字
   - 用到的知识点
     - 逻辑与操作符 and
     - 文件中读取的内容都为字符串，需要用 int 转换为整数，float 转换为浮点数
9. 写程序 `transferMultipleColumToMatrix.py` 将文件 (multipleColExpr.txt) 中基因在多个组织中的表达数据转换为矩阵形式
   - 用到的知识点

  - aDict['key'] = {}
  - aDict['key']['key2'] = value
  - if key not in aDict
  - aDict = {'ENSG00000000003': {"A-431": 21.3, "A-549", 32.5,…},"ENSG00000000003":{},}

- 输入格式 (只需要前 3 列就可以)

```
Gene    Sample  Value   Unit    Abundance
ENSG00000000003 A-431   21.3    FPKM    Medium
ENSG00000000003 A-549   32.5    FPKM    Medium
ENSG00000000003 AN3-CA  38.2    FPKM    Medium
ENSG00000000003 BEWO    31.4    FPKM    Medium
ENSG00000000003 CACO-2  63.9    FPKM    High
ENSG00000000005 A-431   0.0     FPKM    Not detected
ENSG00000000005 A-549   0.0     FPKM    Not detected
ENSG00000000005 AN3-CA  0.0     FPKM    Not detected
ENSG00000000005 BEWO    0.0     FPKM    Not detected
ENSG00000000005 CACO-2  0.0     FPKM    Not detected
```

- 输出格式

```
Name    A-431   A-549   AN3-CA  BEWO    CACO-2
ENSG00000000460 25.2    14.2    10.6    24.4    14.2
ENSG00000000938 0.0 0.0 0.0 0.0 0.0
ENSG00000001084 19.1    155.1   24.4    12.6    23.5
ENSG00000000457 2.8 3.4 3.8 5.8 2.9
```

10. 写程序 `reverseComplementary.py` 计算序列 `ACGTACGTACGTCACGTCAGCTAGAC` 的反向互补序列

- 用到的知识点
  - reverse
  - list(seq)

11. 写程序 `collapsemiRNAreads.py` 转换 smRNA-Seq 的测序数据

- 输入文件格式 (mir.collapse, tab-分割的两列文件，第一列为序列，第二列为序列被测到的次数)

```
ID_REF          VALUE
ACTGCCCTAAGTGCTCCTTCTGGC          2
ATAAGGTGCATCTAGTGCAGATA          25
TGAGGTAGTAGTTTGTGCTGTTT          100
TCCTACGAGTTGCATGGATTC          4
```

- 输出文件格式 (mir.collapse.fa, 名字的前 3 个字母为样品的特异标示，中间的数字表示第几条序列，是序列名字的唯一标示，第三部分是 x 加每个 reads 被测到的次数。三部分用下划线连起来作为 fasta 序列的名字。)

```
>ESB_1_x2
ACTGCCCTAAGTGCTCCTTCTGGC
>ESB_2_x25
ATAAGGTGCATCTAGTGCAGATA
>ESB_3_x100
TGAGGTAGTAGTTTGTGCTGTTT
>ESB_4_x4
TCCTACGAGTTGCATGGATTC
```

12. 简化的短序列匹配程序 (map.py) 把 short.fa 中的序列比对到 ref.fa, 输出短序列匹配到 ref.fa 文件中哪些序列的哪些位置

    - 用到的知识点
      - find
    - 输出格式 (输出格式为 bed 格式，第一列为匹配到的染色体，第二列和第三列为匹配到染色体序列的起始终止位置 (位置标记以 0 为起始，代表第一个位置；终止位置不包含在内，第一个例子中所示序列的位置是 (199,208](前闭后开，实际是 chr1 染色体第 199-206 的序列，0 起始). 第 4 列为短序列自身的序列.)。
    - 附加要求：可以只匹配到给定的模板链，也可以考虑匹配到模板链的互补链。这时第 5 列可以为短序列的名字，第六列为链的信息，匹配到模板链为'+'，匹配到互补链为'-'。注意匹配到互补链时起始位置也是从模板链的 5' 端算起的。

    ```
    chr1    199 208 TGGCGTTCA
    chr1    207 216 ACCCCGCTG
    chr2    63  70  AAATTGC
    chr3    0   7   AATAAAT
    ```

13. 备注：

    - 每个提到提到的 "用到的知识点" 为相对于前面的题目新增的知识点，请综合考虑。此外，对于不同的思路并不是所有提到的知识点都会用着，而且也可能会用到未提到的知识点。但是所有知识点都在前面的讲义部分有介绍。
    - 每个程序对于你身边会写的人来说都很简单，因此你一定要克制住，独立去把答案做出，多看错误提示，多比对程序输出结果和预期结果的差异。
    - 学习锻炼 "读程序"，即对着文件模拟整个的读入、处理过程来发现可能的逻辑问题。
    - 程序运行没有错误不代表你写的程序完成了你的需求，你要去查验输出结果是不是你想要的。

14. 关于程序调试

    - 在初写程序时，可能会出现各种各样的错误，常见的有缩进不一致，变量名字拼写错误，丢失冒号，文件名未加引号等，这时要根据错误提示查看错误类型是什么，出错的是哪一行来定位错误。当然，有的时候报错的行自身不一定有错，可能是其前面或后面的行出现了错误。
    - 用脑袋运行程序：当程序写作完成后，自己尝试对着数据文件，一行一行的执行程序，来看程序的运行是否与自己想干的活一致，有没有纰漏。
    - 当结果不符合预期时，要学会使用 **print** 来查看每步的操作是否正确，比如我读入了字典，我就打印下字典，看看读入的是不是我想要的，是否含有不该存在的字符；或者在每个判断句、函数调入的情况下打印个字符，来跟踪程序的运行轨迹。

2.6 函数操作

函数是重用的程序段。它们允许你给一块语句一个名称，然后你可以在你的程序的任何地方使用这个名称任意多次地运行这个语句块。这被称为 调用函数。我们已经使用了许多内建的函数，比如 `len`, `range`, `input`, `int`, `str`。

也可以导入特定包里面的函数，比如 `os.getcwd`, `sys.exit`。

函数通过 `def` 关键字定义。`def` 关键字后跟一个函数的 标识符名称，然后跟一对圆括号。圆括号之中可以包括一些变量名，该行以冒号结尾。接下来是一块语句，它们是函数体。

```
## 自定义函数
def print_hello():
    print("Hello, you!")


print_hello()
```

```
Hello, you!
```

```
def hello(who):
    print("Hello, %s!" % who)


hello('you')
hello('me')
```

```
Hello, you!
Hello, me!
```

```
## 自定义函数

def sum_num(a, b):
    c = a + b
    return c

d = sum_num(3, 4)
d
```

```
7
```

局部变量和全局变量

全局作用：函数之外，从定义位置起，所有逻辑下游的语句都可以使用

局部作用：函数之内，出了函数就不可以被识别

局部可以使用 `global` 访问全局变量，而全局不能使用局部变量。

```
var = 1

def variable_test():
```

```
    var = 2
    print("var in variable_test is",var)

print("var before running variable_test()", var)
variable_test()
print("var after running variable_test()", var)
```

```
var before running variable_test() 1
var in variable_test is 2
var after running variable_test() 1
```

```
var = 1

def variable_test():
    global var
    var = 2

print("var before running variable_test()", var)
variable_test()
print("var after running variable_test()", var)
```

```
var before running variable_test() 1
var after running variable_test() 2
```

```
## 全局变量在局部作用域中使用
def spam():
    print("eggs in spam",eggs)

eggs = 28
spam()
```

```
eggs in spam 28
```

```
## 全局变量在局部作用域中使用
## 但若局部也有其定义，容易引发未定义冲突
def spam():
    print(eggs)
    eggs = 29


eggs = 28
spam()
print(eggs)
```

```
---------------------------------------------------------------------------

UnboundLocalError                         Traceback (most recent call last)

<ipython-input-22-61f9fdfeb6fe> in <module>()
      6
      7 eggs = 28
----> 8 spam()
      9 print(eggs)


<ipython-input-22-61f9fdfeb6fe> in spam()
      1 # 全局变量在局部作用域中使用
      2 def spam():
----> 3     print(eggs)
      4     eggs = 29
      5


UnboundLocalError: local variable 'eggs' referenced before assignment
```

```
## 尽量避免名称相同的局部变量和全局变量
def spam():
    eggs = 'spam local'
    print("eggs in spam",eggs)  # 输出 spam local


def bacon():
    eggs = 'bacon local'
    print("eggs in bacon", eggs)  # 输出 bacon local
    spam()
    print("eggs in bacon after running spam", eggs)  # 输出 bacon local


eggs = 'global'
bacon()
print("Global eggs", eggs)
```

```
eggs in bacon bacon local
eggs in spam spam local
eggs in bacon after running spam bacon local
Global eggs global
```

```
print(" 把之前写过的语句块稍微包装下就是函数了\n")
```

```
def findAll(string, pattern):
    posL = []
    pos = 0
    for i in string:
        pos += 1
        if i == pattern:
            posL.append(pos)
    #-------------------
    return posL


##------END of findAll-------
a = findAll("ABCDEFDEACFBACACA", "A")
print(a)
print(findAll("ABCDEFDEACFBACACA", "B"))
```

把之前写过的语句块稍微包装下就是函数了

```
[1, 9, 13, 15, 17]
[2, 12]
```

```
def readFasta(file):
    aDict = {}
    for line in open(file):
        if line[0] == '>':
            name = line.strip()
            aDict[name] = []
        else:
            aDict[name].append(line.strip())
    #---------------------------------
    for name, lineL in list(aDict.items()):
        aDict[name] = ''.join(lineL)
    return aDict


print(readFasta("data/test1.fa"))
seqD = readFasta("data/test2.fa")
```

```
{'>NM_001011874 gene=Xkr4 CDS=151-2091': 'gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccgg
```

```
{'>NM_001011874 gene=Xkr4 CDS=151-2091': 'gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccgg
```

```
'>NM_001195662 gene=Rp1 CDS=55-909': 'AAGCTCAGCCTTTGCTCAGATTCTCCTCTTGATGAAACAAAGGGATTTCTGCACAT
'>NM_011283 gene=Rp1 CDS=128-6412': 'AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGT
'>NM_0112835 gene=Rp1 CDS=128-6412': 'AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACG
```

2.6.1 作业（二）

6. 将 “作业（一）” 中的程序块用函数的方式重写，并调用执行
   - 用到的知识点
     - def func(para1,para2,…):
     - func(para1,para2,…)
7. 备注：
   - 每个提到提到的 “用到的知识点” 为相对于前面的题目新增的知识点，请综合考虑。此外，对于不同的思路并不是所有提到的知识点都会用着，而且也可能会用到未提到的知识点。但是所有知识点都在前面的讲义部分有介绍。
   - 每个程序对于你身边会写的人来说都很简单，因此你一定要克制住，独立去把答案做出，多看错误提示，多比对程序输出结果和预期结果的差异。
   - 学习锻炼 “读程序”，即对着文件模拟整个的读入、处理过程来发现可能的逻辑问题。
   - 程序运行没有错误不代表你写的程序完成了你的需求，你要去插眼输出结果是不是你想要的。
8. 关于程序调试
   - 在初写程序时，可能会出现各种各样的错误，常见的有缩进不一致，变量名字拼写错误，丢失冒号，文件名未加引号等，这时要根据错误提示查看错误类型是什么，出错的是哪一行来定位错误。当然，有的时候报错的行自身不一定有错，可能是其前面或后面的行出现了错误。
   - 当结果不符合预期时，要学会使用 print 来查看每步的操作是否正确，比如我读入了字典，我就打印下字典，看看读入的是不是我想要的，是否含有不该存在的字符；或者在每个判断句、函数调入的情况下打印个字符，来跟踪程序的运行轨迹。

2.7 模块

Python 内置了很多标准库，如做数学运算的 `math`, 调用系统功能的 `sys`, 处理正则表达式的 `re`, 操作系统相关功能的 `os` 等。我们主要关注两个库: * sys * sys.argv 处理命令行参数 * sys.exit() 退出函数 * sys.stdin 标准输入 * sys.stderr 标准错误 * os * os.system() 或 os.popen() 执行系统命令 * os.getcwd() 获取当前目录 * os.remove() 删除文件

```
import os
os.getcwd()
##help(os.getcwd)
##os.remove(r'D:\project\github\PBR_training\script\splitName.py')
##os.system('rm file')
```

```
'/MPATHB/ct/ipython/notebook'
```

```
from os import getcwd
getcwd()
```

```
'/MPATHB/ct/ipython/notebook'
```

2.8 命令行参数

sys.argv 是一个列表，存储了包含程序名字在内的传给程序的命令行参数。

```
%%writefile testSys.py
import sys
print(sys.argv)
```

```
Writing testSys.py
```

```
%run testSys 'abc' 1
```

```
['testSys.py', 'abc', '1']
```

```
%%writefile cat.py
import sys

def read_print_file(filename):
    for line in open(filename):
        print(line, end="")
##------END read_print_file--------------------------

##main 函数及其调用部分是我个人写程序的固定格式，照搬就可以
def main(): # 一般主程序会包含在 main 函数中，在文件的最后调用 main 函数即可运行程序
    if len(sys.argv) < 2:  # 如果命令行参数不足两个，则提示操作
        # 一般提示信息输出到标准错误
        print("Usage: python %s filename" % sys.argv[0], file=sys.stderr)
        sys.exit(0)
    file = sys.argv[1]
    read_print_file(file)
##--------END main------------------
## 这句话是说只有在文件被执行时才调用 main 函数。如果这个文件被其它文件调用，则不执行 main 函数。
if __name__ == '__main__':
    main()
```

```
Writing cat.py
```

关于 __main__ 的解释见关于 Python 中的 __main__ 和编程模板。

```
%run cat
```

```
Usage: python cat.py filename
```

```
%run cat data/test1.fa
```

```
>NM_001011874 gene=Xkr4 CDS=151-2091
gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccggggaaggaagccagggcg
>NM_001195662 gene=Rp1 CDS=55-909
AGGTCTCACCCAAAATGAGTGACACACCTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCA
>NM_0112835 gene=Rp15 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCAC
>NM_011283 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCAC
```

使用 optparse，功能更强大 (保留内容)

```
%%writefile skeleton.py
##!/usr/bin/env python

desc = '''
Functional description:

'''

import sys
import os
from time import localtime, strftime
timeformat = "%Y-%m-%d %H:%M:%S"
from optparse import OptionParser as OP

def cmdparameter(argv):
    if len(argv) == 1:
        global desc
        print >>sys.stderr, desc
        cmd = 'python ' + argv[0] + ' -h'
        os.system(cmd)
        sys.exit(0)
    usages = "%prog -i file"
    parser = OP(usage=usages)
    parser.add_option("-i", "--input-file", dest="filein",
        metavar="FILEIN", help="The name of input file. \
```

```
Standard input is accepted.")
    parser.add_option("-v", "--verbose", dest="verbose",
        default=0, help="Show process information")
    parser.add_option("-d", "--debug", dest="debug",
        default=False, help="Debug the program")
    (options, args) = parser.parse_args(argv[1:])
    assert options.filein != None, "A filename needed for -i"
    return (options, args)
##--------------------------------------------------------------------


def main():
    options, args = cmdparameter(sys.argv)
    #----------------------------------
    file = options.filein
    verbose = options.verbose
    debug = options.debug
    #----------------------------------
    if file == '-':
        fh = sys.stdin
    else:
        fh = open(file)
    #--------------------------------
    for line in fh:
        pass
    #-------------END reading file----------
    #----close file handle for files-----
    if file != '-':
        fh.close()
    #-----------end close fh-----------
    if verbose:
        print("--Successful %s" % strftime(timeformat, localtime()),
              file=sys.stderr)
if __name__ == '__main__':
    startTime = strftime(timeformat, localtime())
    main()
    endTime = strftime(timeformat, localtime())
    fh = open('python.log', 'a')
    print("%s\n\tRun time : %s - %s " % \
        (' '.join(sys.argv), startTime, endTime), file=sys.stderr)
    fh.close()
```

```
Writing skeleton.py
```

```
%run skeleton -h
```

```
Usage: skeleton.py -i file

Options:
  -h, --help            show this help message and exit
  -i FILEIN, --input-file=FILEIN
                        The name of input file. Standard input is accepted.
  -v VERBOSE, --verbose=VERBOSE
                        Show process information
  -d DEBUG, --debug=DEBUG
                        Debug the program
```

2.8.1 作业（三）

7. 使 “作业（二）” 中的程序都能接受命令行参数
   - 用到的知识点
     - import sys
     - sys.argv
     - import optparse
8. 备注
   - 每个提到提到的 “用到的知识点” 为相对于前面的题目新增的知识点，请综合考虑。此外，对于不同的思路并不是所有提到的知识点都会用着，而且也可能会用到未提到的知识点。但是所有知识点都在前面的讲义部分有介绍。
   - 每个程序对于你身边会写的人来说都很简单，因此你一定要克制住，独立去把答案做出，多看错误提示，多比对程序输出结果和预期结果的差异。
   - 学习锻炼 “读程序”，即对着文件模拟整个的读入、处理过程来发现可能的逻辑问题。
   - 程序运行没有错误不代表你写的程序完成了你的需求，你要去插眼输出结果是不是你想要的。
9. 关于程序调试
   - 在初写程序时，可能会出现各种各样的错误，常见的有缩进不一致，变量名字拼写错误，丢失冒号，文件名未加引号等，这时要根据错误提示查看错误类型是什么，出错的是哪一行来定位错误。当然，有的时候报错的行自身不一定有错，可能是其前面或后面的行出现了错误。
   - 当结果不符合预期时，要学会使用 print 来查看每步的操作是否正确，比如我读入了字典，我就打印下字典，看看读入的是不是我想要的，是否含有不该存在的字符；或者在每个判断句、函数调入的情况下打印个字符，来跟踪程序的运行轨迹。

2.9 更多 Python 内容

2.9.1 单语句块

```
if True:
    print('yes')

if True: print('yes')

x = 5
y = 3

if x > y:
    print(y)
else:
    print(x)
##-------------
print((y if y < x else x))
print(x)
```

```
yes
yes
3
3
5
```

2.9.2 列表解析

生成新列表的简化的 for 循环

```
aList = [1, 2, 3, 4, 5]
bList = []
for i in aList:
    bList.append(i * 2)
##-----------------------------------
##nameL = [line.strip() for line in open(file)]

bList = [i * 2 for i in aList]
print(bList)
```

```
[2, 4, 6, 8, 10]
```

```
print(" 列表解析可以做判断的")
aList = [1, 2, 3, 4, 5]
bList = [i * 2 for i in aList if i % 2 != 0]
print(bList)
```

列表综合可以做判断的
[2, 6, 10]

```
print(" 列表解析也可以嵌套的")
aList = [1, 2, 3, 4, 5]
bList = [5, 4, 3, 2, 1]
bList = [i * j for i in aList for j in bList]

## for i in aList:
##     for j in bList:
##         print i * j
print(bList)
```

列表综合也可以嵌套的
[5, 4, 3, 2, 1, 10, 8, 6, 4, 2, 15, 12, 9, 6, 3, 20, 16, 12, 8, 4, 25, 20, 15, 10, 5]

2.9.3 字典解析

```
aList = ['a', 'b', 'a', 'c']
aDict = {i:aList.count(i) for i in aList}
aDict
```

```
{'a': 2, 'b': 1, 'c': 1}
```

```
bDict = {i:j*2 for i,j in aDict.items()}
bDict
```

```
{'a': 4, 'b': 2, 'c': 2}
```

2.9.4 断言

设定运行过程中必须满足的条件，当情况超出预期时报错。常用于文件读入或格式判断时，有助于预防异常的读入或操作。

```
a = 1
b = 2
assert a == b, "a is %s, b is %s" % (a, b)

if a == b:
    pass
else:
    print("a is %s, b is %s" % (a, b))
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

<ipython-input-75-9c43179b4557> in <module>()
      1 a = 1
      2 b = 2
----> 3 assert a == b, "a is %s, b is %s" % (a, b)
      4
      5 if a == b:


AssertionError: a is 1, b is 2
```

2.9.5 更多字符串方法

is.X 字符串方法

isalpha() 是否只含字母

isalnum() 是否只是字母或数字

isdecimal() 是否只有数字

isspace() 是否只有空格制表符换行

istitle() 是否字符串为大写开头，后面均为小写字母

```
a = 'b1'
a.isalpha()
```

```
False
```

```
a = 'b c'
a.isalpha()
```

```
False
```

```
a = 'bc1'
a.isalnum()
```

```
True
```

```
a = '1a'
a.isalnum()
```

```
True
```

```
','.join(['i', 'love', 'python'])
```

```
'i,love,python'
```

```
'***'.join(['i', 'love', 'python'])
```

```
'i***love***python'
```

```
"linux R perl C python".split()
```

```
['linux', 'R', 'perl', 'C', 'python']
```

文本对齐 rjust() ljust() center()

```
'hello'.rjust(10)
```

```
'     hello'
```

```
'hello'.rjust(20,'*')
```

```
'***************hello'
```

```
'hello'.center(20,'-')
```

```
'-------hello--------'
```

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))


picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

```
---PICNIC ITEMS--
sandwiches..    4
apples......   12
cups........    4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches..........     4
apples..............    12
cups................     4
cookies.............  8000
```

strip(),rstrip(),lstrip() 删除空白字符

```
spam = '    hello    '
spam
```

```
'    hello    '
```

```
spam.strip()
```

```
'hello'
```

```
spam.rstrip()
```

```
'    hello'
```

```
spam.lstrip()
```

```
'hello    '
```

```
a = 'Hello world, welcome to python world'
a.strip('d')
```

```
'Hello world, welcome to python worl'
```

2.9.6 lambda, map, filer, reduce (保留节目)

- lambda 产生一个没有名字的函数，通常为了满足一次使用，其使用语法为 `lambda argument_list: expression`。参数列表是用逗号分隔开的一个列表，表达式是这些参数的组合操作。
- map 执行一个循环操作，使用语法为 `map(func, seq)`。第一个参数是要调用的函数或函数的名字，第二个参数是一个序列（如列表、字符串、字典）。map 会以序列的每个元素为参数调用 func，并新建一个输出列表。
- filter 用于过滤列表，使用语法为 `filter(func, list)`。以第二个参数的每个元素调用 func，返回值为 True 则保留，否则舍弃。
- reduce 连续对列表的元素应用函数，使用语法为 `reduce(func, list)`。如果我们有一个列表 aList = [1,2,3, … ,n], 调用 `reduce(func, aList)` 后进行的操作为: 首先前两个元素会传入函数 func 做运算，返回值替换这两个元素，成为数组第一个元素 aList = [func(1,2),3, … , n]; 然后当前的前两个元素再传图 func 函数做运算，返回值返回值替换这两个元素，成为数组第一个元素 aList = [func(func(1,2),3), … , n]，直到列表只有一个元素。

```
print(" 求和函数")


def f(x, y): return x + y


print(f([1, 2, 3], [4, 5, 6]))
print(f(10, 15))
```

```
求和函数
[1, 2, 3, 4, 5, 6]
25
```

```
print(" 单个参数的 map, lambda 调用")
aList = [1, 2, 3, 4, 5]
print([x**2 for x in aList])

print(" 多个参数的 map, lambda 调用")
```

```
def f(x, y): return x + y


print(list(map(f, [1, 2, 3], [4, 5, 6])))

print(" 参数为字符串")
print([x.upper() for x in 'acdf'])
```

单个参数的map, lambda调用
[1, 4, 9, 16, 25]
多个参数的map, lambda调用
[5, 7, 9]
参数为字符串
['A', 'C', 'D', 'F']

```
print(" 输出所有的奇数")
aList = [1, 2, 3, 4, 5]
print([x for x in aList if x % 2])
```

输出所有的奇数
[1, 3, 5]

```
from functools import reduce
print(" 列表求和")
aList = [1, 2, 3, 4, 5]
print(reduce(lambda a, b: a + b, aList))
```

列表求和
15

```
from functools import reduce
print(" 列表取最大值")
aList = [1, 2, 3, 4, 5]
print(reduce(lambda a, b: a if a > b else b, aList))
```

列表取最大值
5

2.9.7 exec, eval (执行字符串 python 语句, 保留节目)

```
a = 'print("Executing a string as a command")'
exec(a)
```

```
Executing a string as a command
```

```
a = '(2 + 3) * 5'
eval(a)
```

```
25
```

2.9.8 正则表达式

正则表达式通俗来讲就是模式匹配，给定一个模式，寻找可以配对的子串。在 Python 中，是使用 `re` 模块来实现的。

`re.compile`: 转换正则表达式为模式对象

`re.match`: 在待匹配字符串的起始匹配模式对象

`re.search`: 在待匹配字符串内部搜索第一个匹配对象

```
## 示例，寻找起始密码子
import re
cds = "ATGACGCTCGGACGACTAATG"
start_codon = re.compile('ATG')
start_codon_match = start_codon.match(cds)
start_codon_match.group()
```

```
'ATG'
```

```
## 如果前面有 UTR，起始密码子不在第一位，则 match 找不到
mRNA = "GTCAATGACGCTCGGACGACTAATG"
start_codon_match = start_codon.match(mRNA)
if start_codon_match:
    print(start_codon_match.group())
else:
    print("No start codon found at the beginning of the given sequence.")
```

```
No start codon at the beginning of the given sequence.
```

```
## 如果前面有 UTR，起始密码子不在第一位，则需要使用 search
mRNA = "GTCAATGACGCTCGGACGACTAATG"
start_codon_match = start_codon.search(mRNA)
if start_codon_match:
    print(start_codon_match.group())
else:
    print("No start codon found in the given sequence.")
```

```
ATG
```

```
## 如果想找出所有的终止子，则使用 findall
mRNA = "ATGATGTAAUAATAGUGA"
stop_codon = re.compile('[TU]AA|[TU]AG|[TU]GA')
##stop_codon = re.compile('UAA|UAG|UGA')
stop_codon.findall(mRNA)
```

```
['TGA', 'TAA', 'UAA', 'TAG', 'UGA']
```

上面的模式中使用了正则表达式的 2 个特殊符号，`|` 和 `[]`。

`A|B`: 表示 A 或 B 有一个匹配就可以，如上面的 `TAA|TAG`；如果想限定 `|` 两端的字符的范围，需要使用括号界定，如 `T(AA|T)AG` 则表示可以匹配 `TAAAG` 或 `TTAG`。

`[TU]`: 中括号的任意字符都可以匹配，既该位置可以是 `T`，也可以是 `U`。此外还可以使用 `[A-Z]` 表示所有大写字母，`[A-Za-z]` 表示所有英文字母，`[0-9]` 表示所有阿拉伯数字，也可以写 `[W-Z5-9_]` 表示一部分字母、数字和下划线。

匹配某一个 motif

```
## 匹配某一个 motif，
## 要求 motif
##    第 1 位为 A，
##    第 2 位任意字符，
##    第 3 位为 T，
##    中间间隔 3-5 个任意碱基，
##    随后一位是 G

seqL = ["ACGTACGT", "ACTCCCG","ACTCCGGG","AGTTTTTG"]

## . 表示任意字符 (不包括换行)
## {3,5} 表示前面的字符出现次数为 3-5 次
```

```
pattern = re.compile("A.T.{3,5}G")

print("Matched", "\t","Matched part")
for seq in seqL:
    match = pattern.search(seq)
    if match:
        print(seq, "\t",match.group())
```

```
Matched      Matched part
ACTCCCG      ACTCCCG
ACTCCGGG     ACTCCGGG
AGTTTTTG     AGTTTTTG
```

根据空格切割字符串

假如有这个一个字符串，"A B C D , E, F"，是由逗号，和数目不等的空格、TAB 键或其组合连接起来的，想拆分开获取单独的部分。

```
"A B  C D , E, F".split(' ')
```

```
['A', 'B', '', 'C', 'D', ',', 'E,', 'F']
```

```
## [] 都熟悉了，
## \s: 表示所有的空白，包括空格，TAB 键，换行等
## +：表示前面的字符出现 1 次或多次
import re
pattern = re.compile("[,\s]+")

seq = "A B  C D , E, F"

pattern.split(seq)
```

```
['A', 'B', 'C', 'D', 'E', 'F']
```

记忆匹配

假如我们有一些 fq.gz 文件，想获取其文件名，并输出。

```
## root 和 leaf 是样品名字
## 第一个下划线后面的是生物重复 rep1, rep2
```

```
## 第二个下划线后的 1 和 2 分别代表双端测序的左端和右端。
fqL = ["root_rep1_1.fq.gz", "root_rep1_2.fq.gz",
       "root_rep2_1.fq.gz","root_rep2_2.fq.gz",
       "leaf_rep1_1.fq.gz", "leaf_rep1_2.fq.gz",
       "leaf_rep2_1.fq.gz","leaf_rep2_2.fq.gz"]

## * 表示前面字符出现任意多次
## () 在这表示记忆匹配，可以使用下标获取
## \ 是转义字符， \. 把. 转化为一个正常字符，即这儿匹配的是一个真正的.，
## 而不是任意字符
pattern = re.compile("([^_]*)_([^_]*)_[12]\.fq\.gz")

for fq in fqL:
    match = pattern.search(fq)
    sample = match.group(1)
    rep    = match.group(2)
    print(sample,rep)
```

```
root rep1
root rep1
root rep2
root rep2
leaf rep1
leaf rep1
leaf rep2
leaf rep2
```

匹配替换

国人的名字一般姓在前，名在后，老外写法是名在前，姓在后，现在需要做一个转换操作。

```
## 把下面的名字转为姓在后，名在前
nameL = ["Chen Tong", "Liu Yongxin", "Wang Ying"]

## \w: 表示单词字符，等同于 [A-Za-z0-9_]
pattern = re.compile("(\w+) (\w+)")

## \2, \1 表示记忆匹配的第一个和第二个，具体的计数方式已最左侧括号为准，
## 最左第一个括号是\1, 第二个是\2.
for name in nameL:
    print(pattern.sub(r"\2 \1", name))
```

```
Tong Chen
```

Yongxin Liu
Ying Wang

更多正则表达式的规则见下图，剩下的就是勤学多练了。

语法	说明	表达式实例	完整匹配的字符串
	字符		
一般字符	匹配自身	abc	abc
.	匹配任意除换行符"\n"外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。 如果字符串中有字符*需要匹配，可以使用*或者字符集[*]。	a\.c a\\c	a.c a\c
[...]	字符集（字符类）。对应的位置可以是字符集中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用]、-或^，可以在前面加上反斜杠，或把]、-放在第一个字符，把^放在非第一个字符。	a[bcd]e	abe ace ade
	预定义字符集（可以写在字符集[...]中）		
\d	数字：[0-9]	a\dc	a1c
\D	非数字：[^\d]	a\Dc	abc
\s	空白字符：[<空格>\t\r\n\f\v]	a\sc	a c
\S	非空白字符：[^\s]	a\Sc	abc
\w	单词字符：[A-Za-z0-9_]	a\wc	abc
\W	非单词字符：[^\w]	a\Wc	a c
	数量词（用在字符或(...)之后）		
*	匹配前一个字符0或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
*? +? ?? {m,n}?	使 * + ? {m,n}变成非贪婪模式。	示例将在下文中介绍。	
	边界匹配（不消耗待匹配字符串中的字符）		
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
$	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	abc$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配\w和\W之间。	a\b!bc	a!bc
\B	[^\b]	a\Bbc	abc
	逻辑、分组		
|	|代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配右边的表达式。 如果|没有被包括在()中，则它的范围是整个正则表达式。	abc|def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号'('，编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式中的|仅在该组中有效。	(abc){2} a(123|456)c	abcabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
	特殊构造（不作为分组）		
(?:...)	(...)的不分组版本，用于使用'|'或后接数量词。	(?:abc){2}	abcabc
(?iLmsux)	iLmsux的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介绍。	(?i)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern |no-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配yes-pattern，否则需要匹配no-pattern。 |no-patern可以省略。	(\d)abc(?(1)\d|abc)	1abc2 abcabc

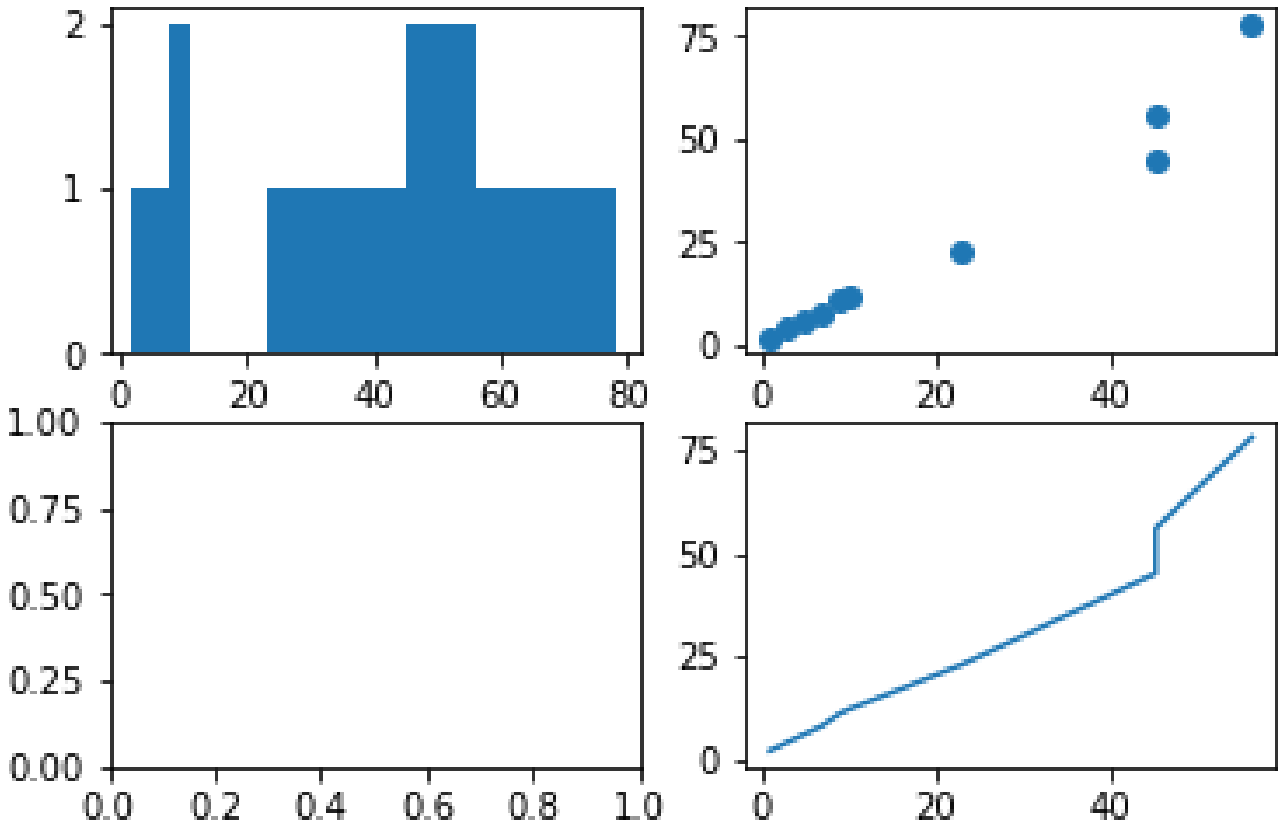
图片来源于 https://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

2.10 Python 画图

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from numpy.random import randn
```

```
x = [1, 3, 5, 7, 9, 10, 23, 45, 45, 56]
y = [2, 4, 6, 8, 11, 12, 23, 45, 56, 78]


fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)  # 创建 4 个图的 Figure 对象，最后的 1 为选中第一个
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

ax1.hist(x, y)
ax2.scatter(x, y)
ax4.plot(x, y)

plt.show()
```

2.10.0.1 Figure 和 Subplot




```
fig, axes = plt.subplots(2, 3)
axes
```

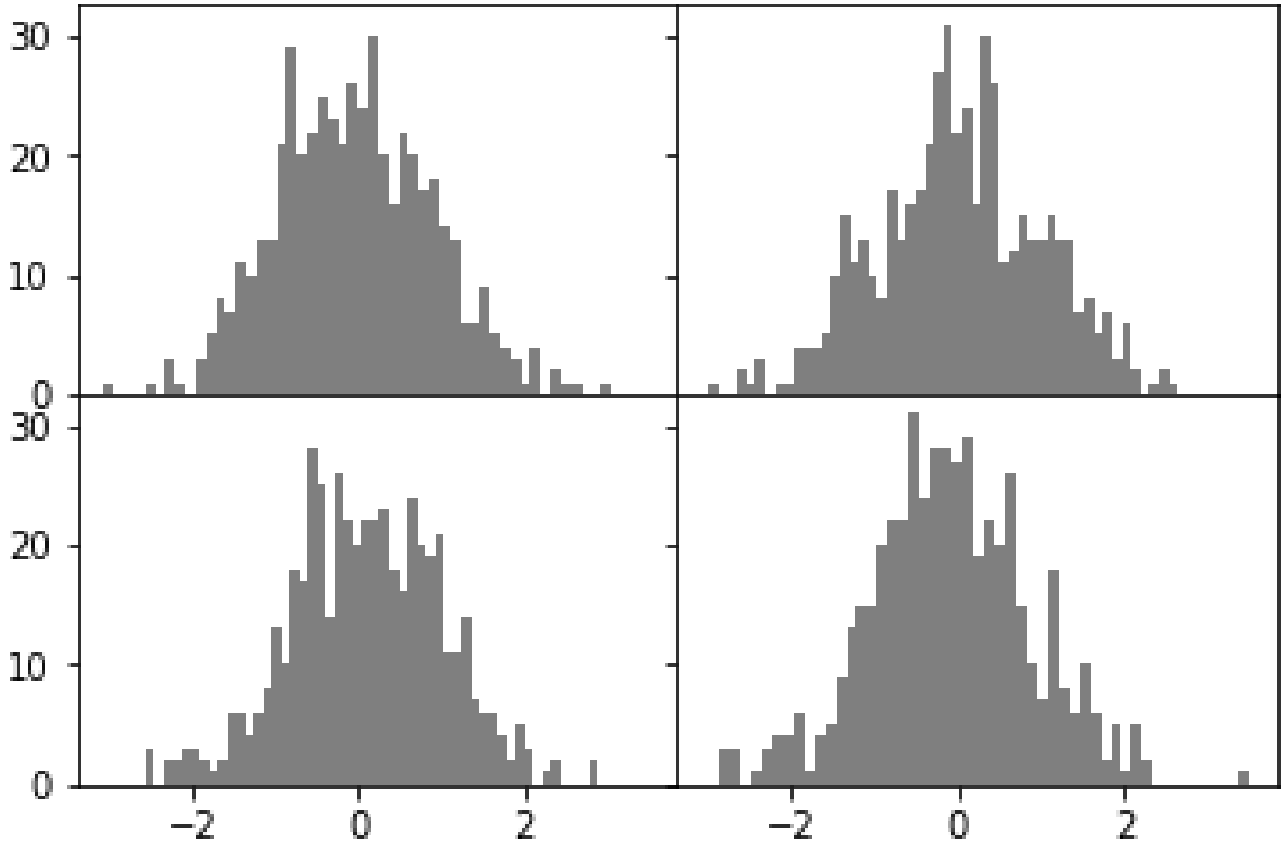
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B5D88D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000009A87FD0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B3467B8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000000009CF0390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B6CC080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B5D9978>]], dtype=object)
```

创建 2 X 3 图像，可以相当于对二维数组进行索引 参数 说明 nrows subplot 行数 ncols subplot 列数 sharex 所有图使用相同的 x 轴 sharey 所有图使用相同的 y 轴 subplot_kw 用于创建各 subplot 的关键字字典 |

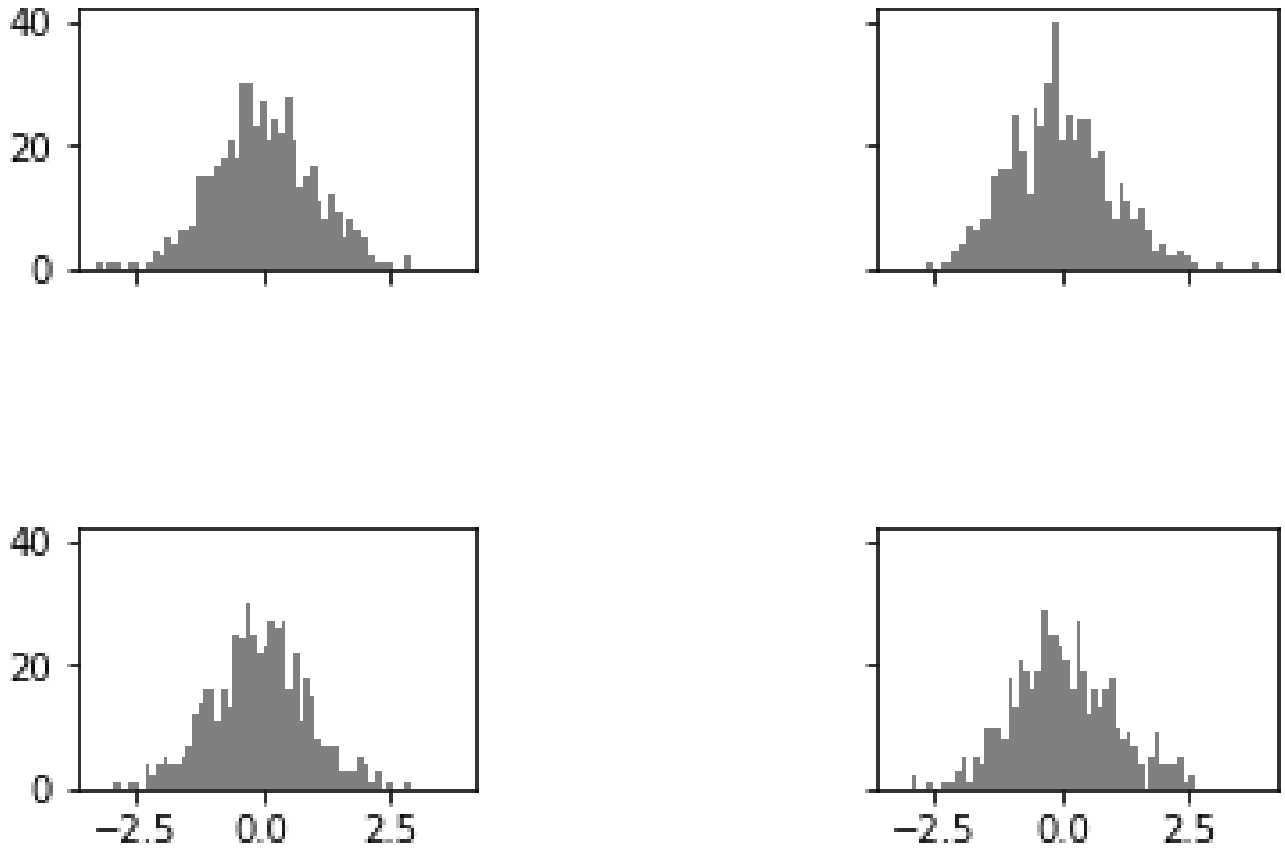
2.10.0.2 调整 subplot 周围间距

```
subplots_ajust(left=None,bottom=None,right=None,top=None,wspace=None,hspace=None)
wspace和hspace控制宽度和高度的百分比
```

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```

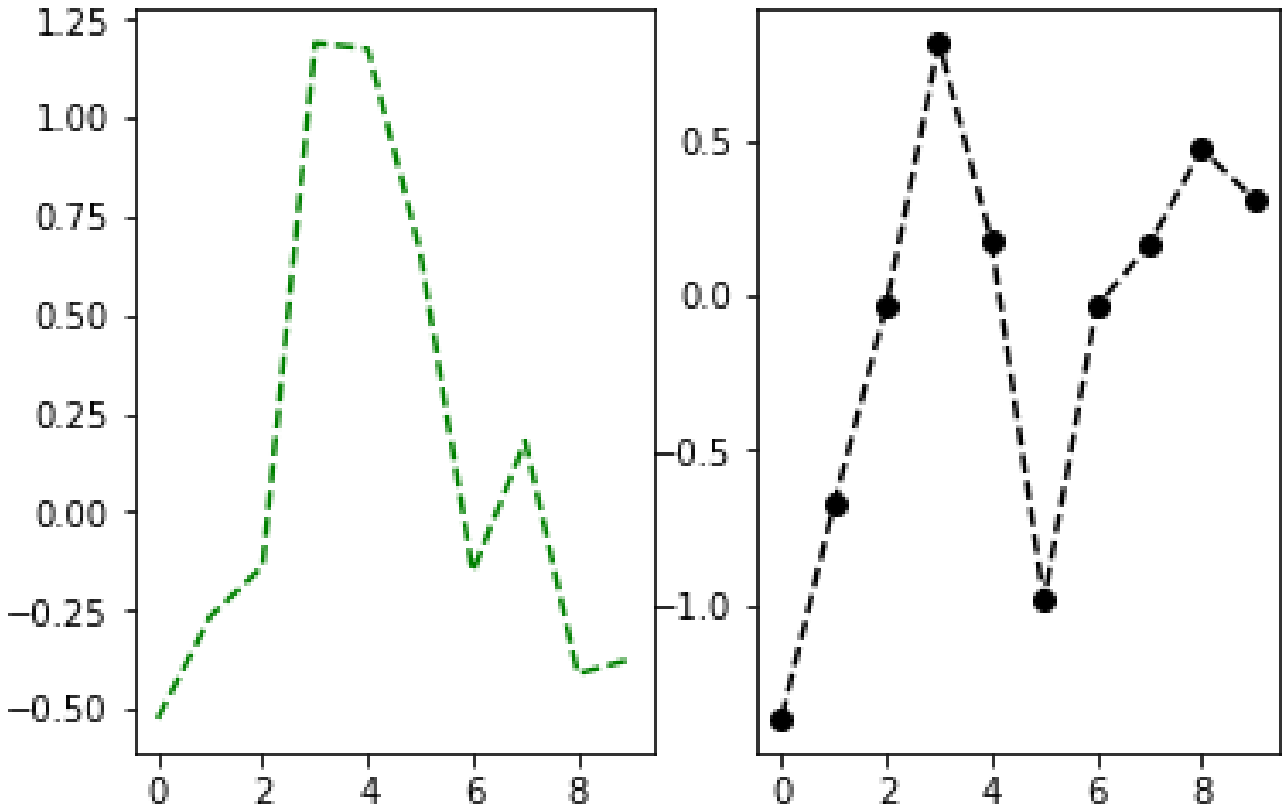



```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=1, hspace=1)
plt.show()
```



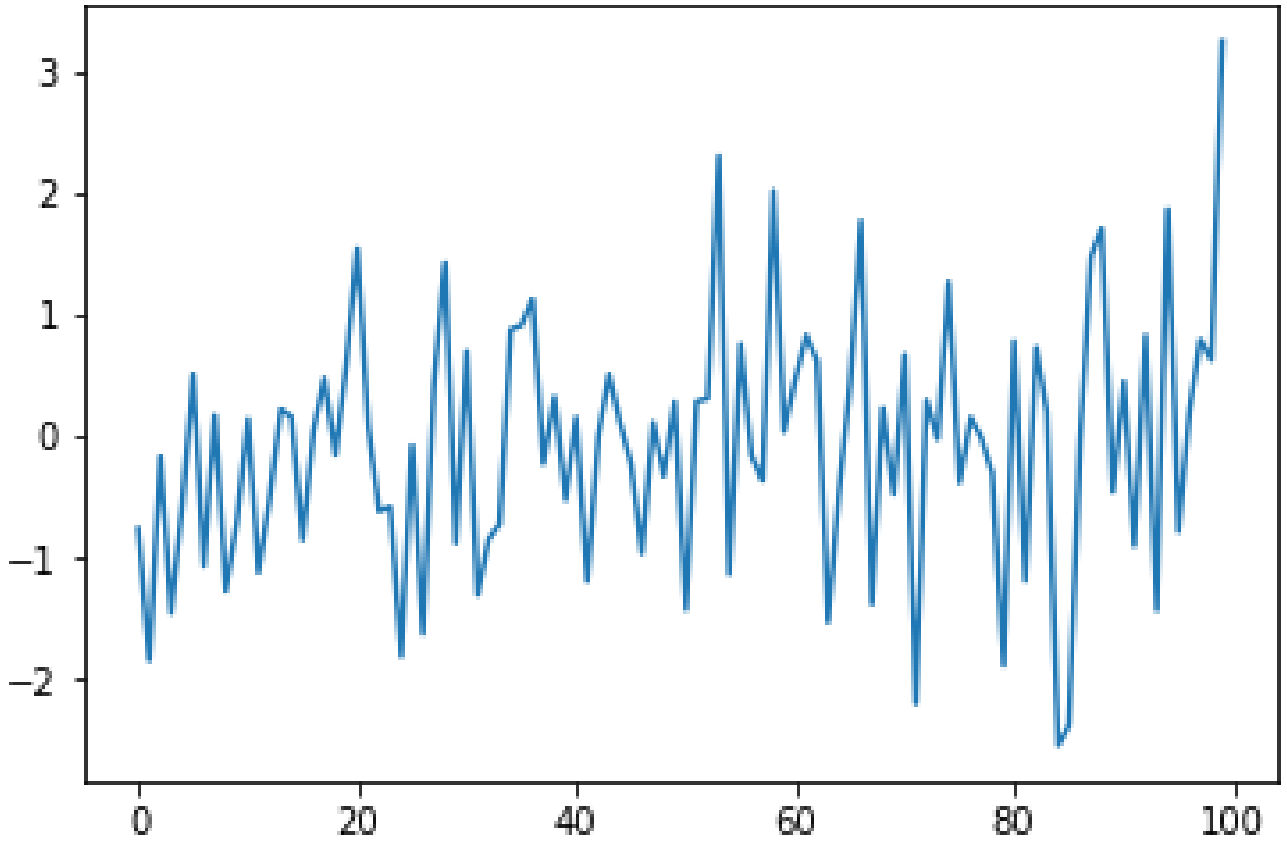

```
"""
绘制绿色虚线
ax.plot(x,y,'g--')
另一种方式
ax.plot(x,y,linestyle='--',color='g')
标记点（maker）
"""
fig, axes = plt.subplots(1, 2)

axes[0].plot(randn(10), 'g--')   # green ---
axes[1].plot(randn(10), 'ko--')   # k：black o：圆点

plt.show()
```

2.10.0.3 颜色标记和线型

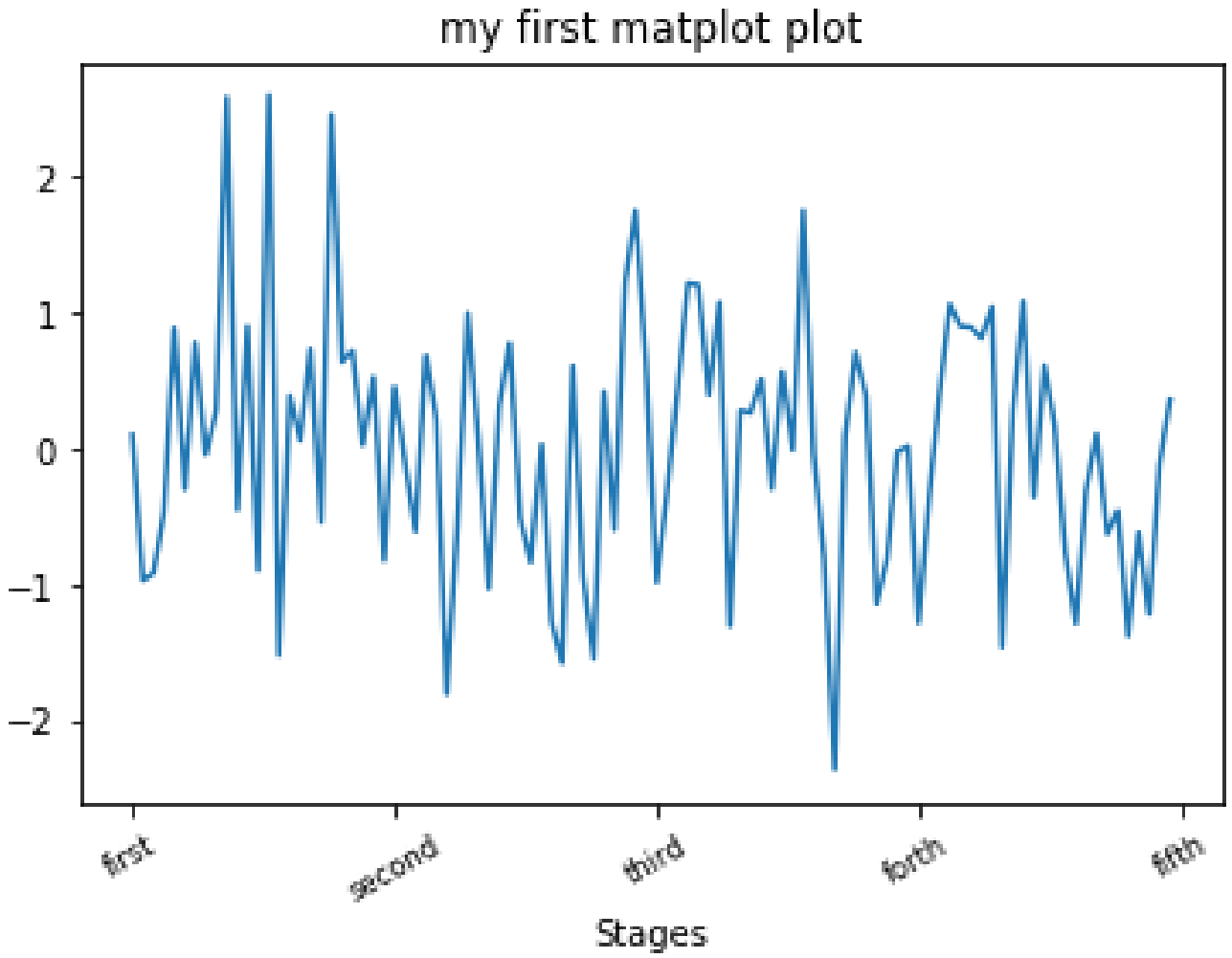



```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(100))
plt.show()
```

2.10.0.4 刻度、标签和图例




```
""" 修改上图的轴"""

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(100))

ticks = ax.set_xticks([0, 25, 50, 75, 100])  # 设置刻度
labels = ax.set_xticklabels(
    ['first', 'second', 'third', 'forth', 'fifth'], rotation=30, fontsize='small')  # 设置 x 轴标

ax.set_title('my first matplot plot')  # 设置图片标题

ax.set_xlabel('Stages')  # 设置 x 轴名称

plt.show()
```

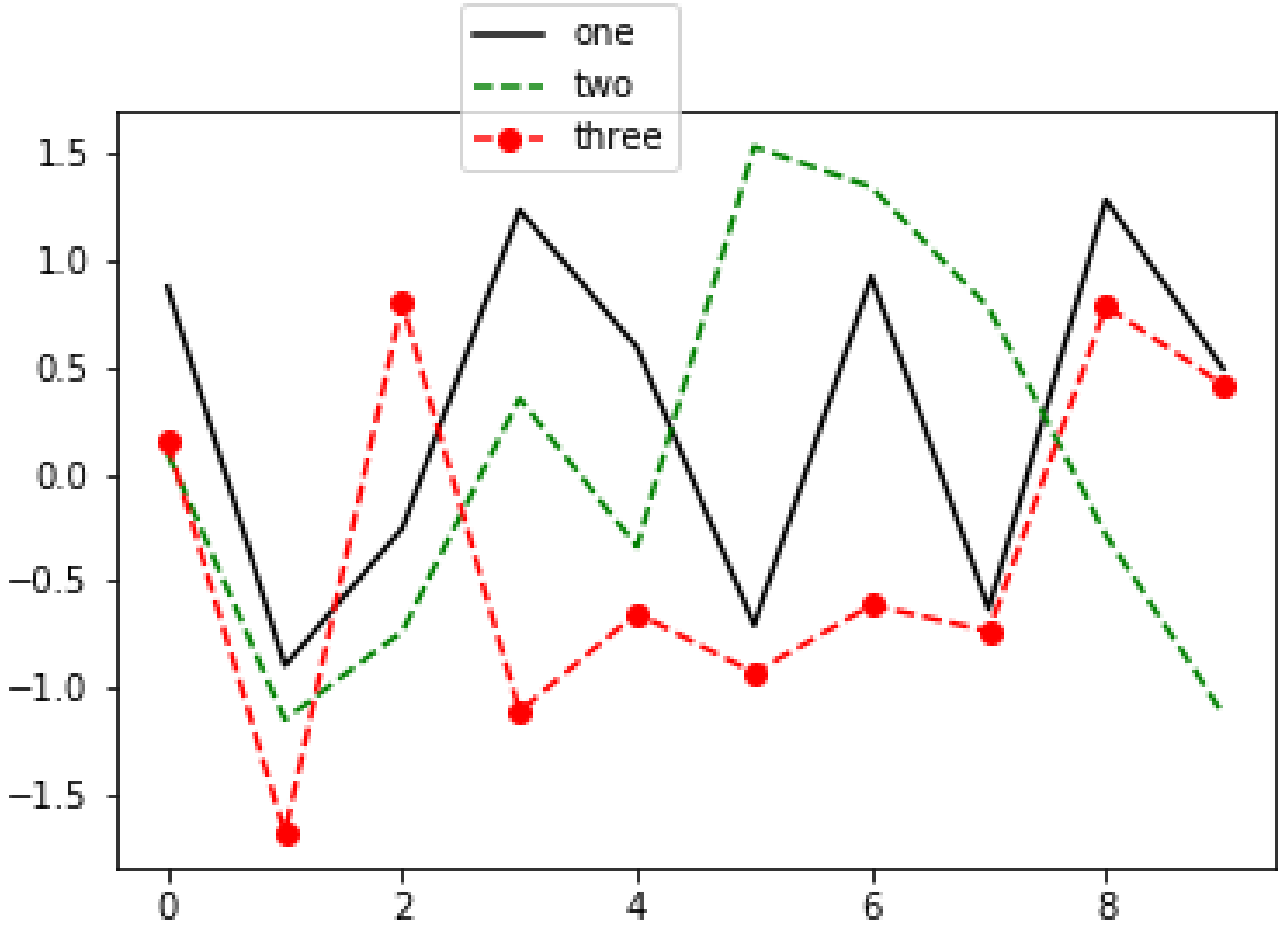


2.10.0.5 添加图例 legend

```
https://matplotlib.org/api/legend_api.html?highlight=legend#module-matplotlib.legend
‘best’          0
‘upper right’   1
‘upper left’    2
‘lower left’    3
‘lower right’   4
‘right’         5
‘center left’   6
‘center right’  7
‘lower center’  8
‘upper center’  9
‘center’        10
bbox_to_anchor= (0.5,0.8)
```

bbox_to_anchor被赋予的二元组中，第一个数值用于控制legend的左右移动，值越大越向右边移动，第二个数值用于控制legend的上下移动，值越大，越向上移动

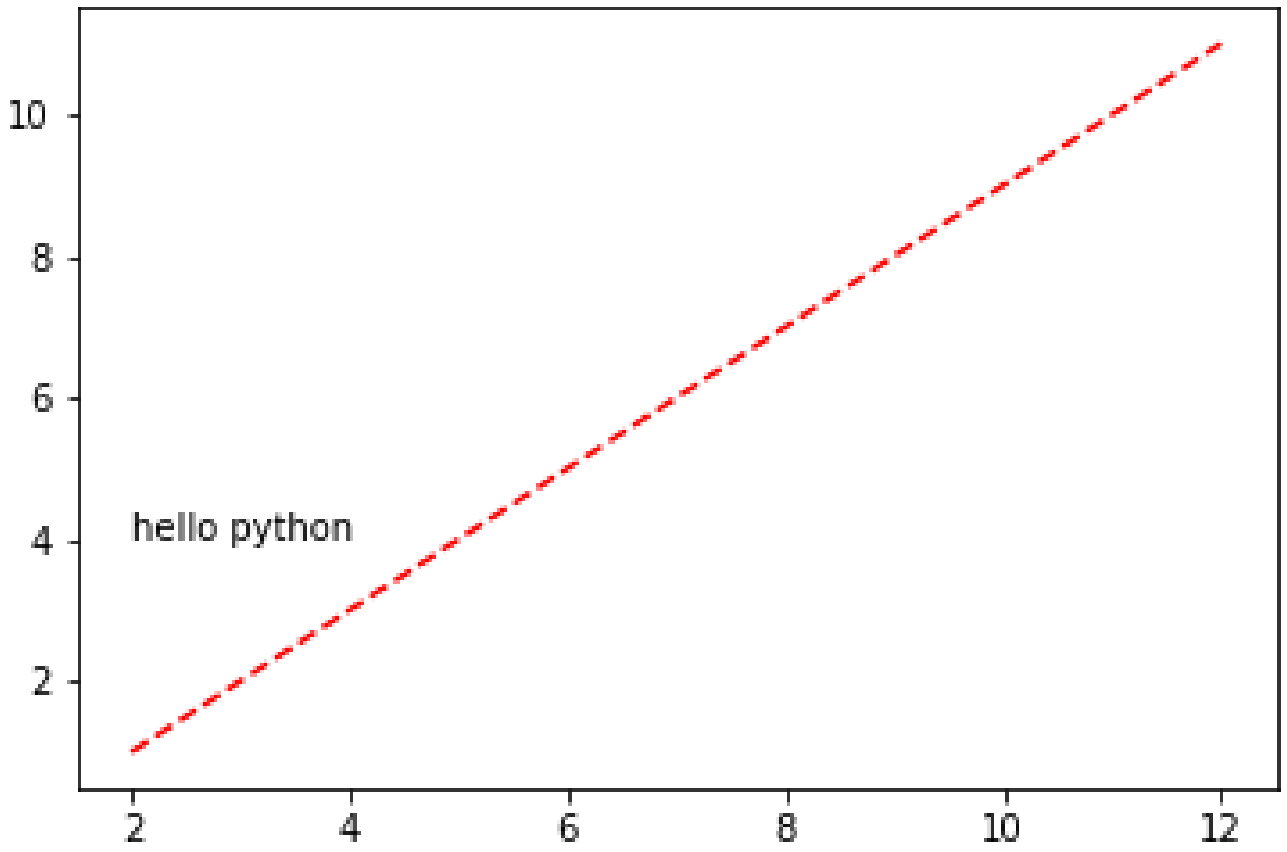
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(randn(10), 'k', label='one')  # 画一条线，k 黑色
ax.plot(randn(10), 'g--', label='two')  # 画第二条线，g 绿色 - -类型
ax.plot(randn(10), 'ro--', label='three')  # 画第三条线红色 ，类型 ...

ax.legend(loc=0, bbox_to_anchor=(0.5, 0.9))


plt.show()
```




```
x = [2, 4, 6, 8, 10, 12]
y = [1, 3, 5, 7, 9, 11]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
```

```
##ax = fig
ax.plot(x, y, 'r--')

ax.text(2, 4, 'hello python')
plt.show()
```

2.10.0.6 注解




```
x = [2, 4, 6, 8, 10, 12]
y = [1, 3, 5, 7, 9, 11]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y, 'r--')

ax.text(2, 4, 'hello python')
```

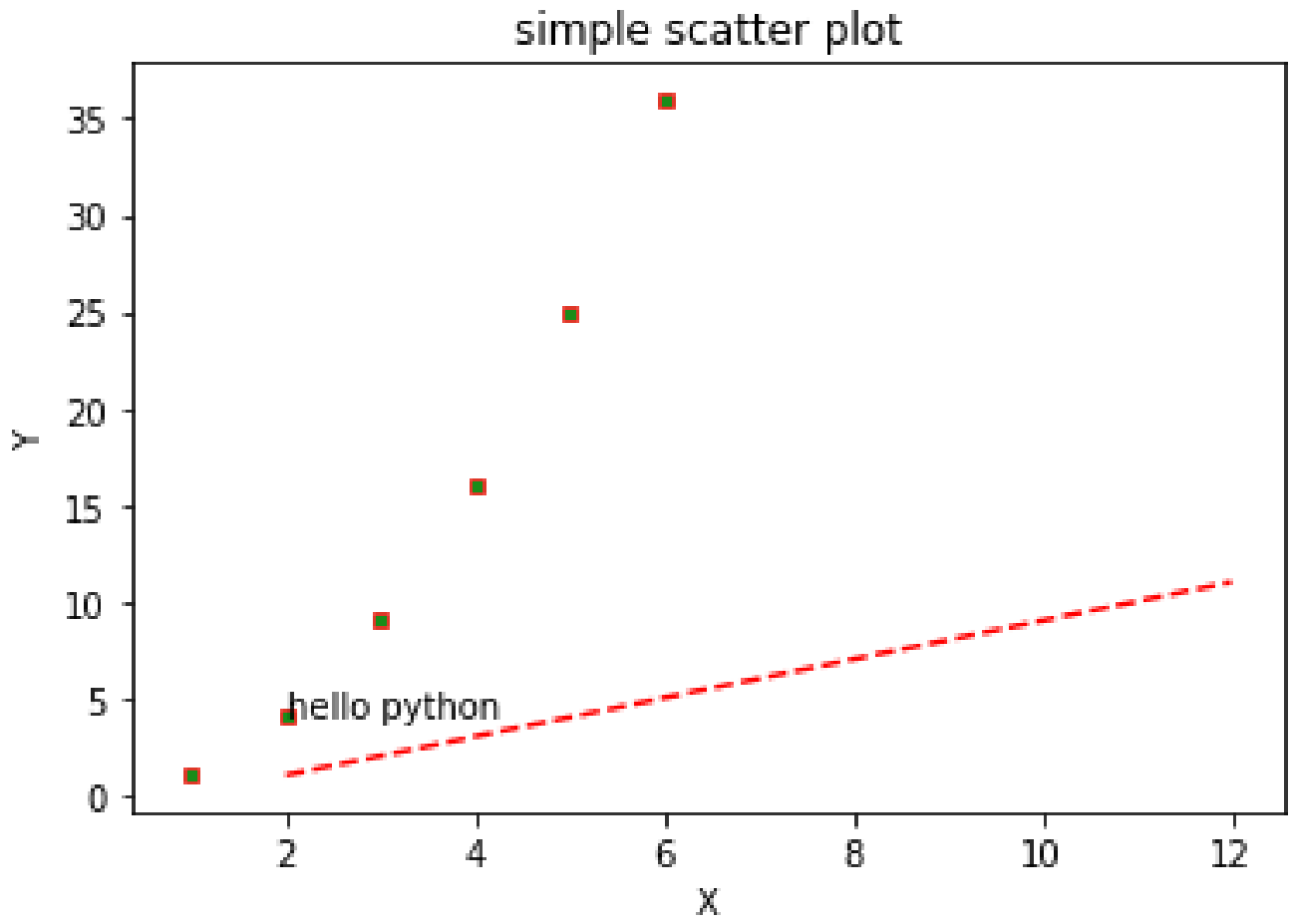
```
## bbox_inches 减除当前图片周围空白部分
plt.savefig('figpath.jpg', dpi=300, bbox_inches='tight')
```

2.10.0.7 图片保存

2.10.0.8 matplotlib 配置绘图之前可以调整各种参数字体，全局所有图片的大小等。 实例

```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]
plt.scatter(x,  # x 轴数据为汽车速度
            y,  # y 轴数据为汽车的刹车距离
            s=20,  # 设置点的大小
            c='green',  # 设置点的颜色
            marker='s',  # 设置点的形状
            alpha=0.9,  # 设置点的透明度
            linewidths=0.8,  # 设置散点边界的粗细
            edgecolors='red'  # 设置散点边界的颜色
            )

plt.title('simple scatter plot')
plt.xlabel('X')  # x 轴名称
plt.ylabel('Y')

plt.show()  # 展示绘图
```

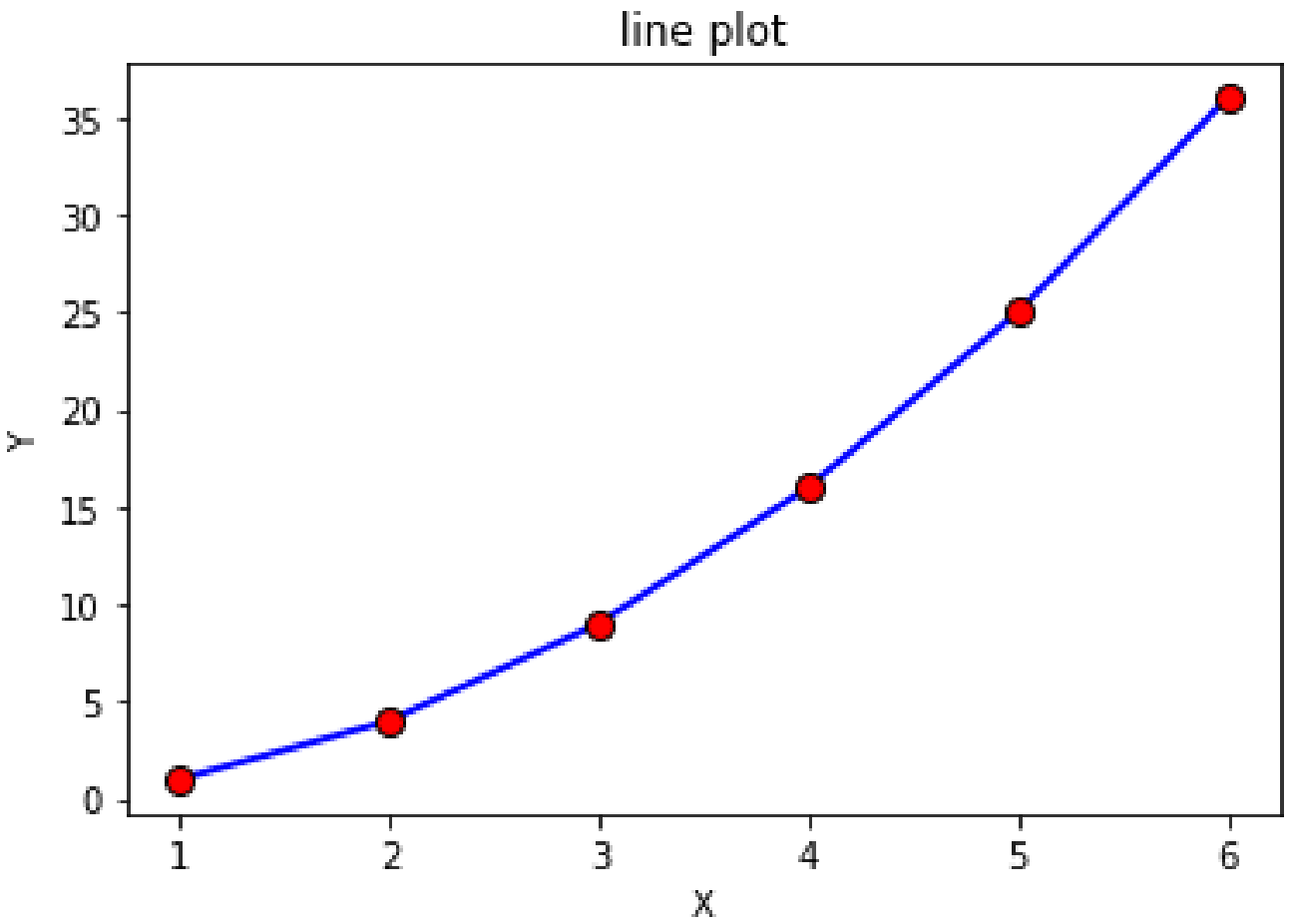
2.10.0.8.1 绘制散点图




```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]

plt.plot(x,  # x 轴数据
         y,  # y 轴数据
         linestyle='-',  # 折线类型
         linewidth=2,  # 折线宽度
         color='blue',  # 折线颜色
         marker='o',  # 点的形状
         markersize=8,  # 点的大小
         markeredgecolor='black',  # 点的边框色
         markerfacecolor='red')  # 点的填充色

## 添加标题和坐标轴标签
plt.title('line plot')
plt.xlabel('X')
plt.ylabel('Y')
```

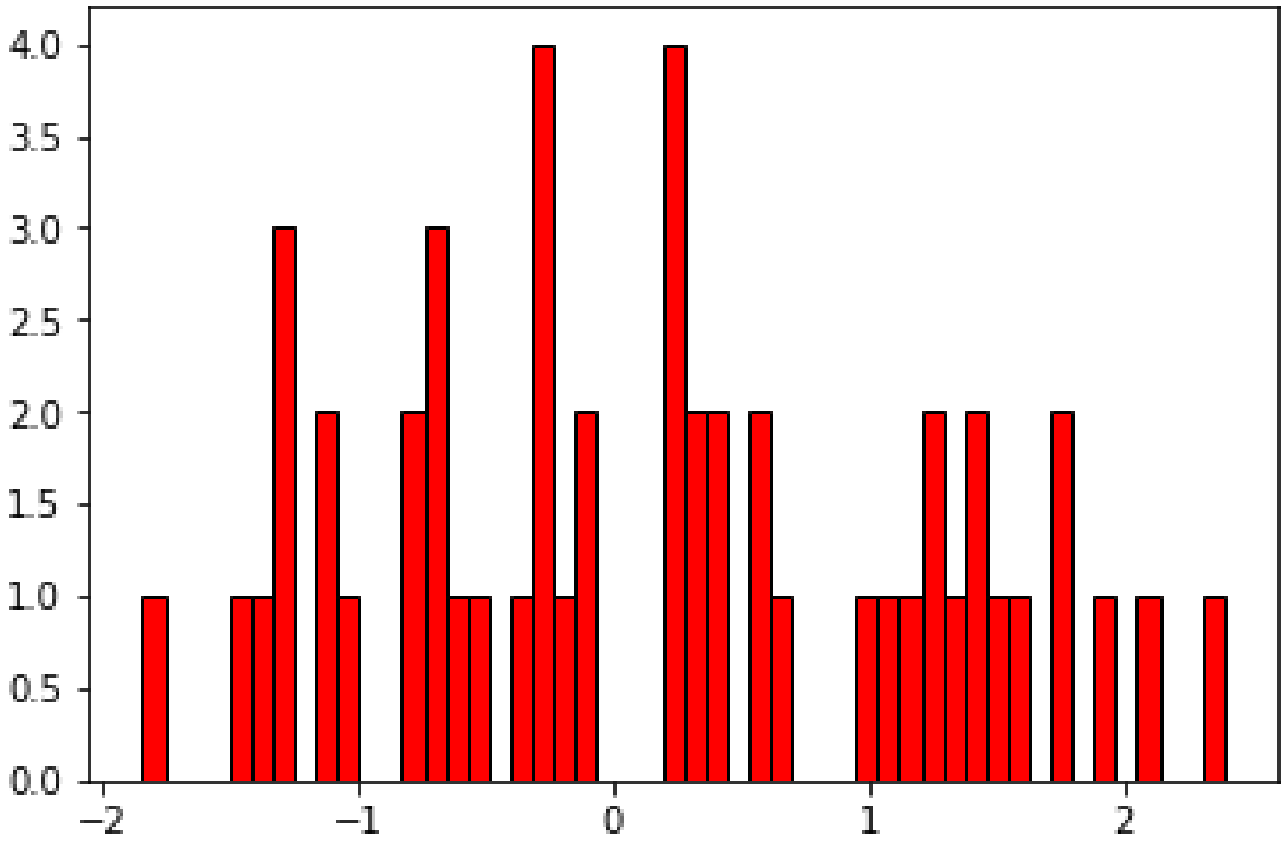
```
plt.show()
```

2.10.0.8.2 折线图



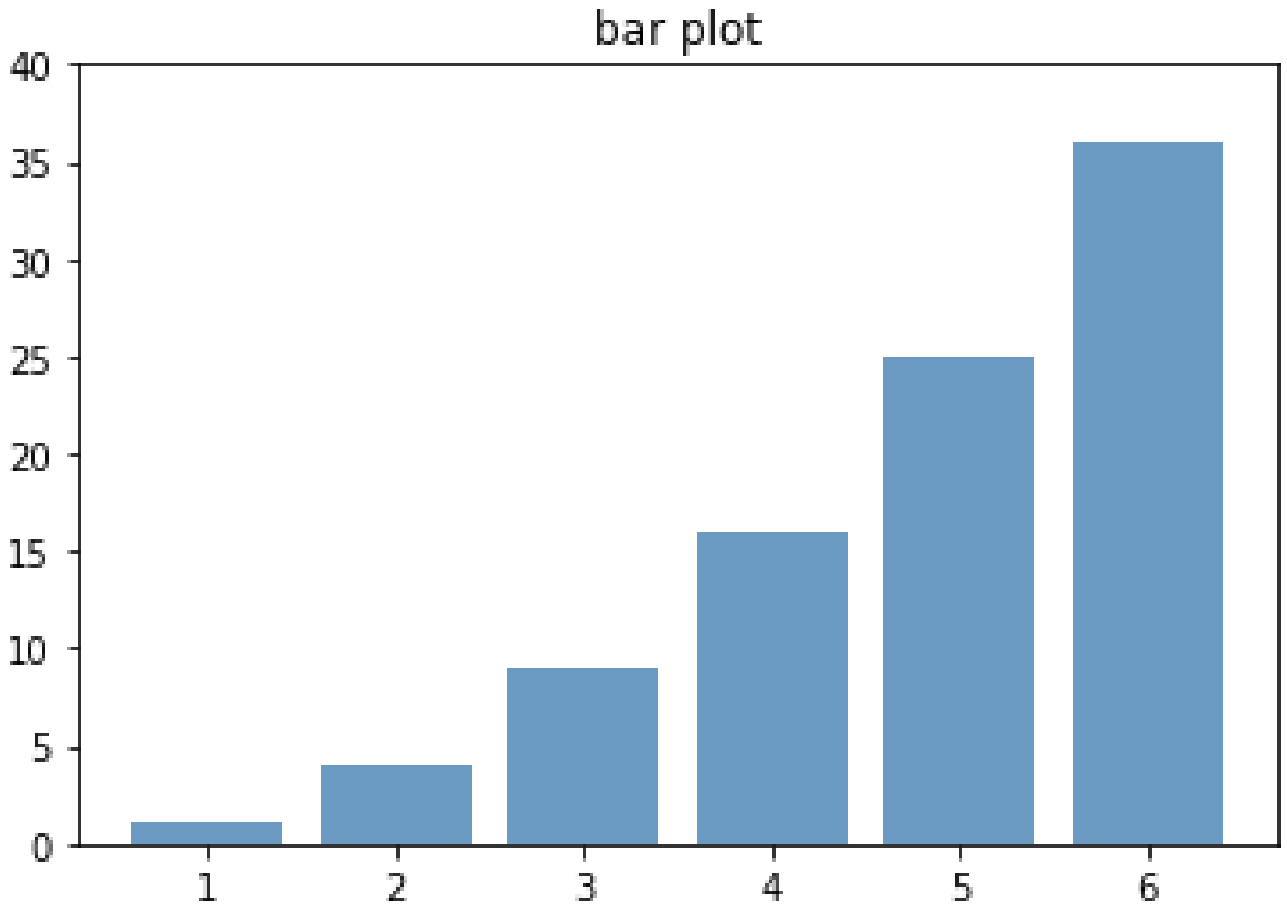

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.hist(np.random.randn(50),  # 绘图数据
         bins=50,  # 指定直方图的条形数为 20 个
         color='red',  # 指定填充色
         edgecolor='k',  # 指定直方图的边界色
         label='histogram')  # 为直方图呈现标签
plt.show()
```

2.10.0.8.3 直方图

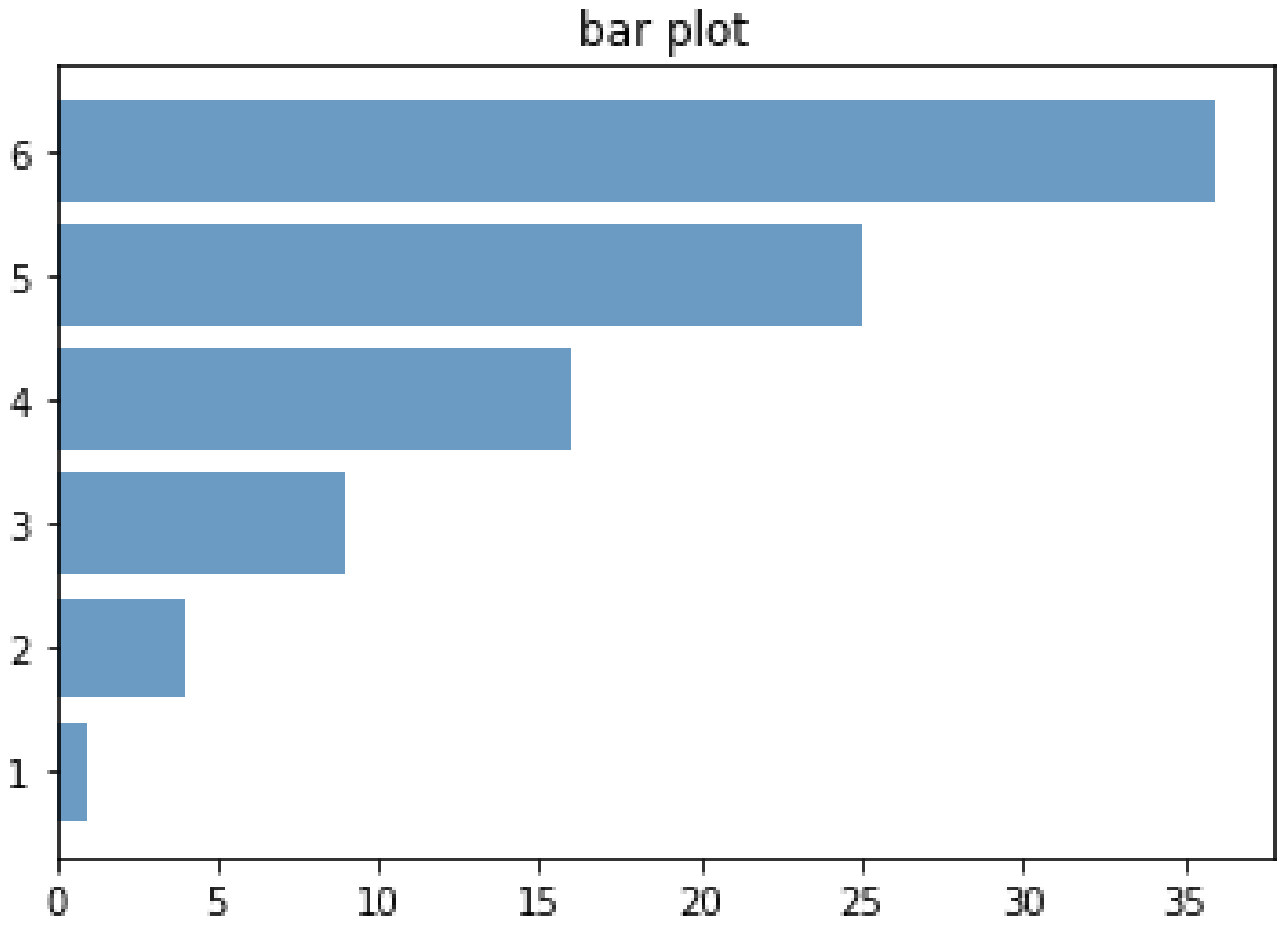



```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]

plt.bar(x, y,
        color='steelblue',
        alpha=0.8)

plt.title('bar plot')

plt.ylim([0, 40])
plt.show()
```

2.10.0.9 直条图

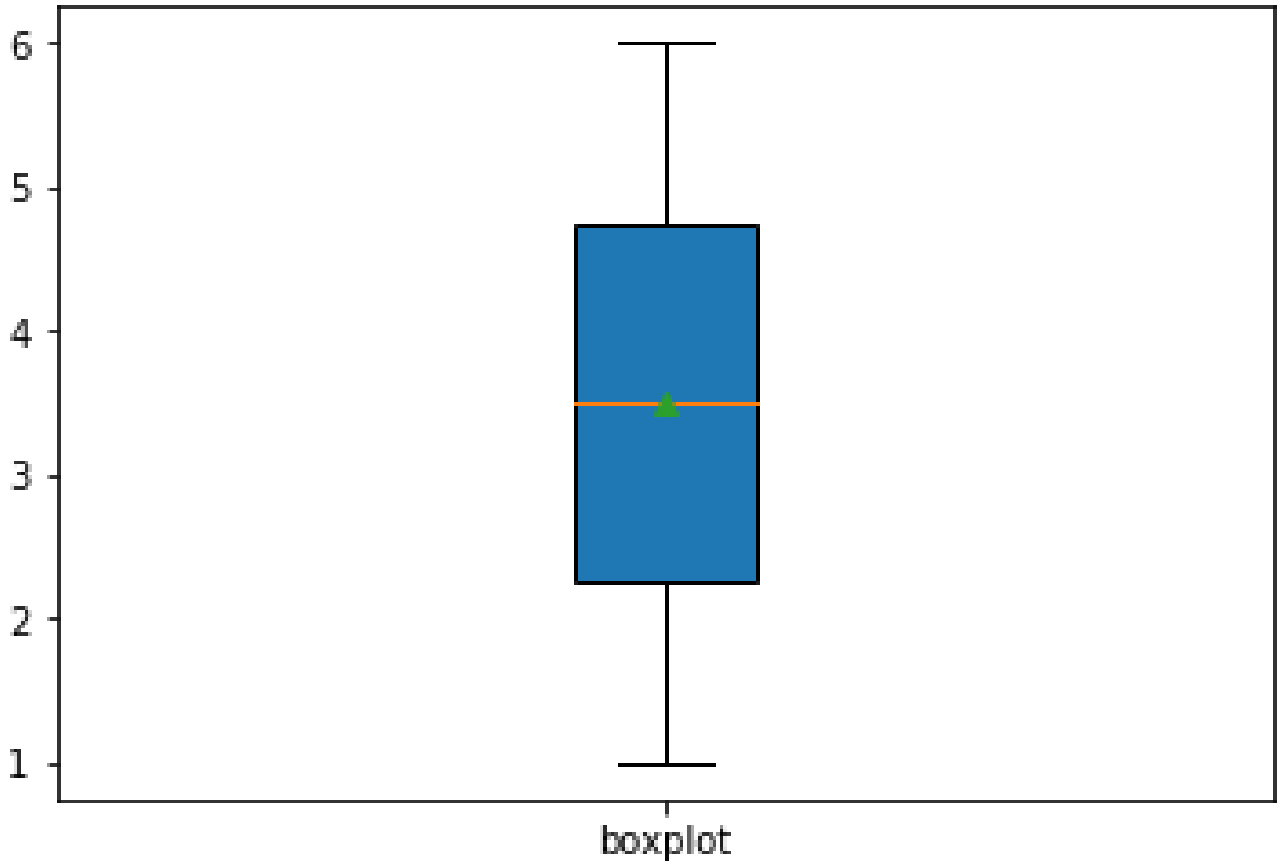



```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]

plt.barh(x, y,
         color='steelblue',
         alpha=0.8)

plt.title('bar plot')

plt.show()
```




```
x = [1, 2, 3, 4, 5, 6]
plt.boxplot(x,
            patch_artist=True,   # 箱体添加颜色
            labels=['boxplot'],   # 添加具体的标签名称
            showmeans=True, )

## 显示图形
plt.show()
```

2.10.0.10 箱线图




```
np.random.seed(2)  # 设置随机种子
df = pd.DataFrame(np.random.rand(5, 4),
                  columns=(['A', 'B', 'C', 'D']))
df
```

```
<tr style="text-align: right;">
  <th></th>
  <th>A</th>
  <th>B</th>
  <th>C</th>
  <th>D</th>
</tr>
```

```
<tr>
  <th>0</th>
  <td>0.435995</td>
  <td>0.025926</td>
  <td>0.549662</td>
  <td>0.435322</td>
</tr>
```

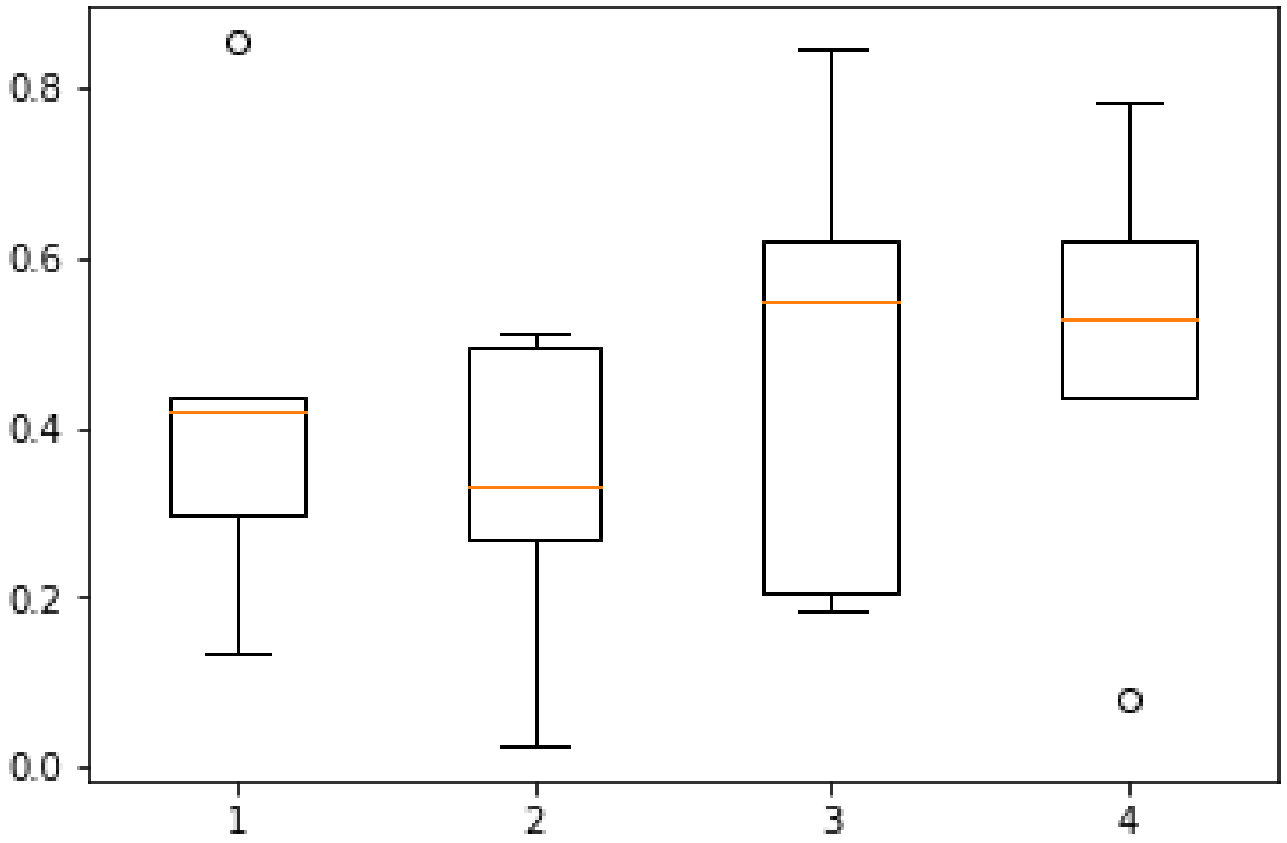
```
<tr>
  <th>1</th>
  <td>0.420368</td>
  <td>0.330335</td>
  <td>0.204649</td>
  <td>0.619271</td>
</tr>
<tr>
  <th>2</th>
  <td>0.299655</td>
  <td>0.266827</td>
  <td>0.621134</td>
  <td>0.529142</td>
</tr>
<tr>
  <th>3</th>
  <td>0.134580</td>
  <td>0.513578</td>
  <td>0.184440</td>
  <td>0.785335</td>
</tr>
<tr>
  <th>4</th>
  <td>0.853975</td>
  <td>0.494237</td>
  <td>0.846561</td>
  <td>0.079645</td>
</tr>
```

```
data = []
for i in range(4):
    data.append(df.iloc[:, i])
data
```

```
[0    0.435995
 1    0.420368
 2    0.299655
 3    0.134580
 4    0.853975
 Name: A, dtype: float64, 0    0.025926
 1    0.330335
 2    0.266827
 3    0.513578
 4    0.494237
 Name: B, dtype: float64, 0    0.549662
 1    0.204649
```

```
2    0.621134
3    0.184440
4    0.846561
Name: C, dtype: float64, 0    0.435322
1    0.619271
2    0.529142
3    0.785335
4    0.079645
Name: D, dtype: float64]
```

```
plt.boxplot(data)
plt.show()
```




```
plt.boxplot(x, notch=None, sym=None, vert=None,
            whis=None, positions=None, widths=None,
            patch_artist=None, meanline=None, showmeans=None,
            showcaps=None, showbox=None, showfliers=None,
            boxprops=None, labels=None, flierprops=None,
            medianprops=None, meanprops=None,
            capprops=None, whiskerprops=None)
```

x：指定要绘制箱线图的数据；
notch：是否是凹口的形式展现箱线图，默认非凹口；
sym：指定异常点的形状，默认为+号显示；
vert：是否需要将箱线图垂直摆放，默认垂直摆放；
whis：指定上下须与上下四分位的距离，默认为1.5倍的四分位差；
positions：指定箱线图的位置，默认为[0,1,2…]；
widths：指定箱线图的宽度，默认为0.5；
patch_artist：是否填充箱体的颜色；
meanline：是否用线的形式表示均值，默认用点来表示；
showmeans：是否显示均值，默认不显示；
showcaps：是否显示箱线图顶端和末端的两条线，默认显示；
showbox：是否显示箱线图的箱体，默认显示；
showfliers：是否显示异常值，默认显示；
boxprops：设置箱体的属性，如边框色，填充色等；
labels：为箱线图添加标签，类似于图例的作用；
filerprops：设置异常值的属性，如异常点的形状、大小、填充色等；
medianprops：设置中位数的属性，如线的类型、粗细等；
meanprops：设置均值的属性，如点的大小、颜色等；
capprops：设置箱线图顶端和末端线条的属性，如颜色、粗细等；
whiskerprops：设置须的属性，如颜色、粗细、线的类型等；

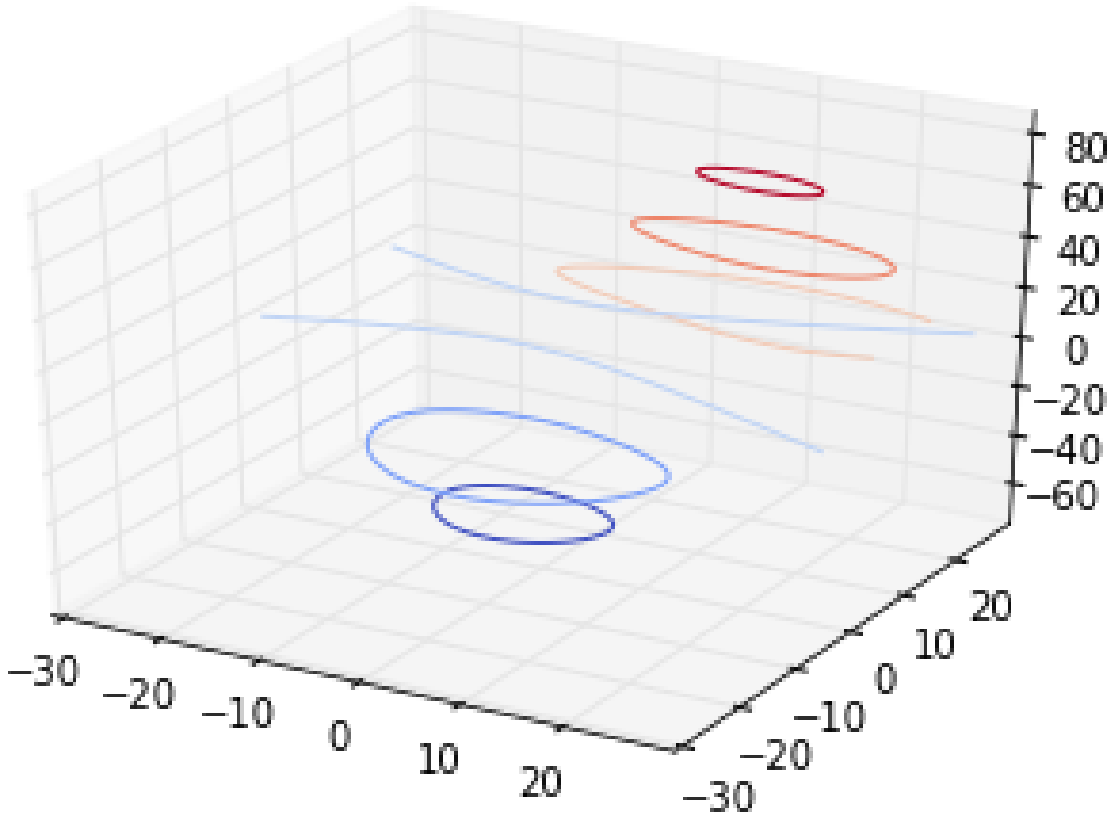
```
data = [0.2, 0.3, 0.4, 0.1]
plt.pie(data)
plt.show()
```

2.10.0.10.1 饼图

```
plt.pie(x, explode=None, labels=None, colors=None,
        autopct=None, pctdistance=0.6, shadow=False,
        labeldistance=1.1, startangle=None,
        radius=None, counterclock=True, wedgeprops=None,
        textprops=None, center=(0, 0), frame=False)
```

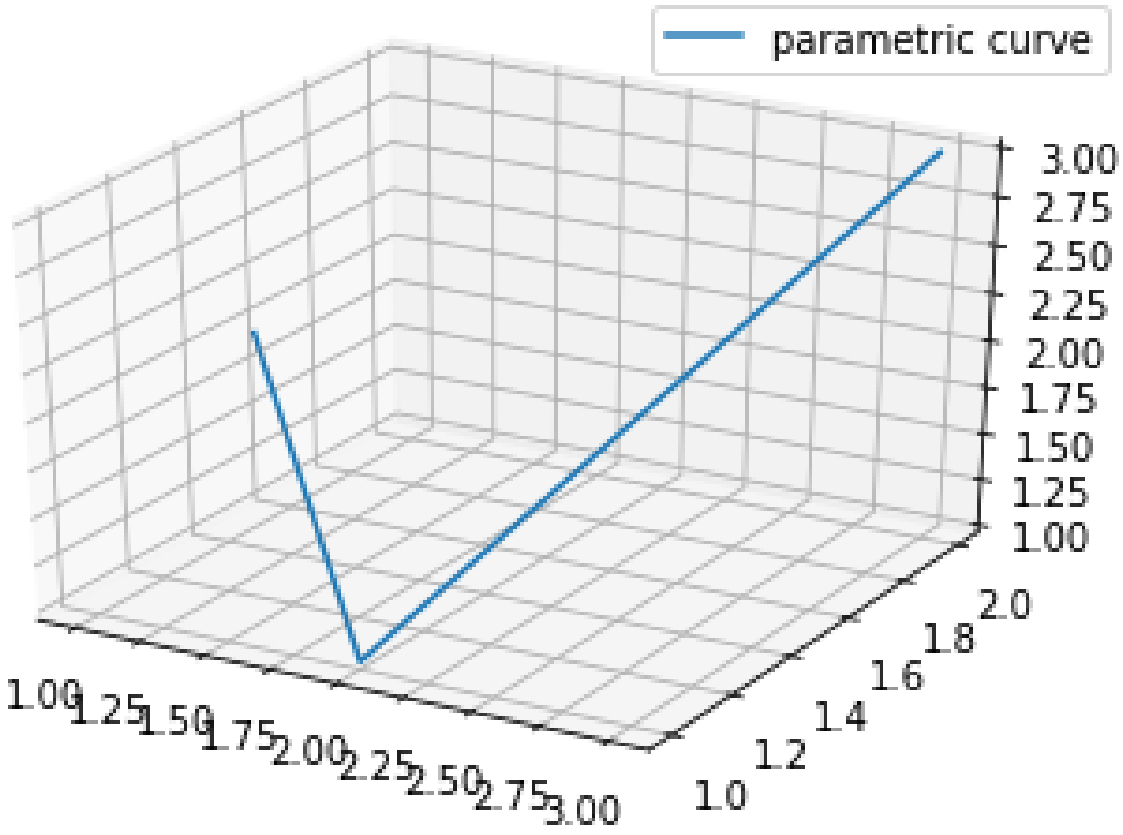
x：指定绘图的数据；
explode：指定饼图某些部分的突出显示，即呈现爆炸式；
labels：为饼图添加标签说明，类似于图例说明；
colors：指定饼图的填充色；
autopct：自动添加百分比显示，可以采用格式化的方法显示；
pctdistance：设置百分比标签与圆心的距离；
shadow：是否添加饼图的阴影效果；
labeldistance：设置各扇形标签（图例）与圆心的距离；
startangle：设置饼图的初始摆放角度；
radius：设置饼图的半径大小；
counterclock：是否让饼图按逆时针顺序呈现；
wedgeprops：设置饼图内外边界的属性，如边界线的粗细、颜色等；
textprops：设置饼图中文本的属性，如字体大小、颜色等；
center：指定饼图的中心点位置，默认为原点
frame：是否要显示饼图背后的图框，如果设置为True的话，需要同时控制图框x轴、y轴的范围和饼图的中心

```
%matplotlib inline
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
cset = ax.contour(X, Y, Z, cmap=cm.coolwarm)
ax.clabel(cset, fontsize=9, inline=1)

plt.show()
```



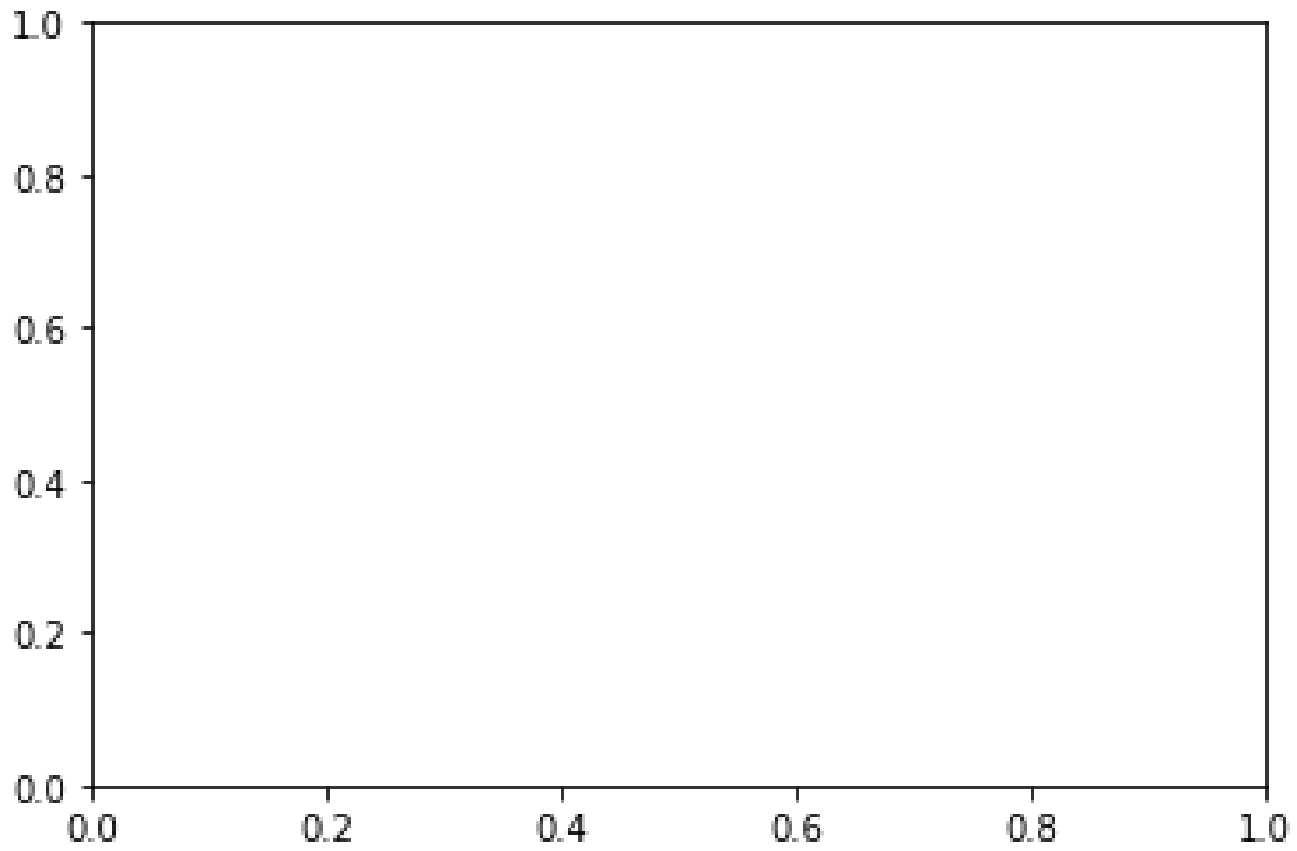

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10
```

```
fig = plt.figure()
ax = fig.gca(projection='3d')
##theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
##z = np.linspace(-2, 2, 100)
##r = z**2 + 1
##x = r * np.sin(theta)
##y = r * np.cos(theta)
x = [1, 2, 3]
y = [1.5, 1, 2]
z = [2, 1, 3]
ax.plot(x, y, z, label='parametric curve')
ax.legend()

plt.show()
```

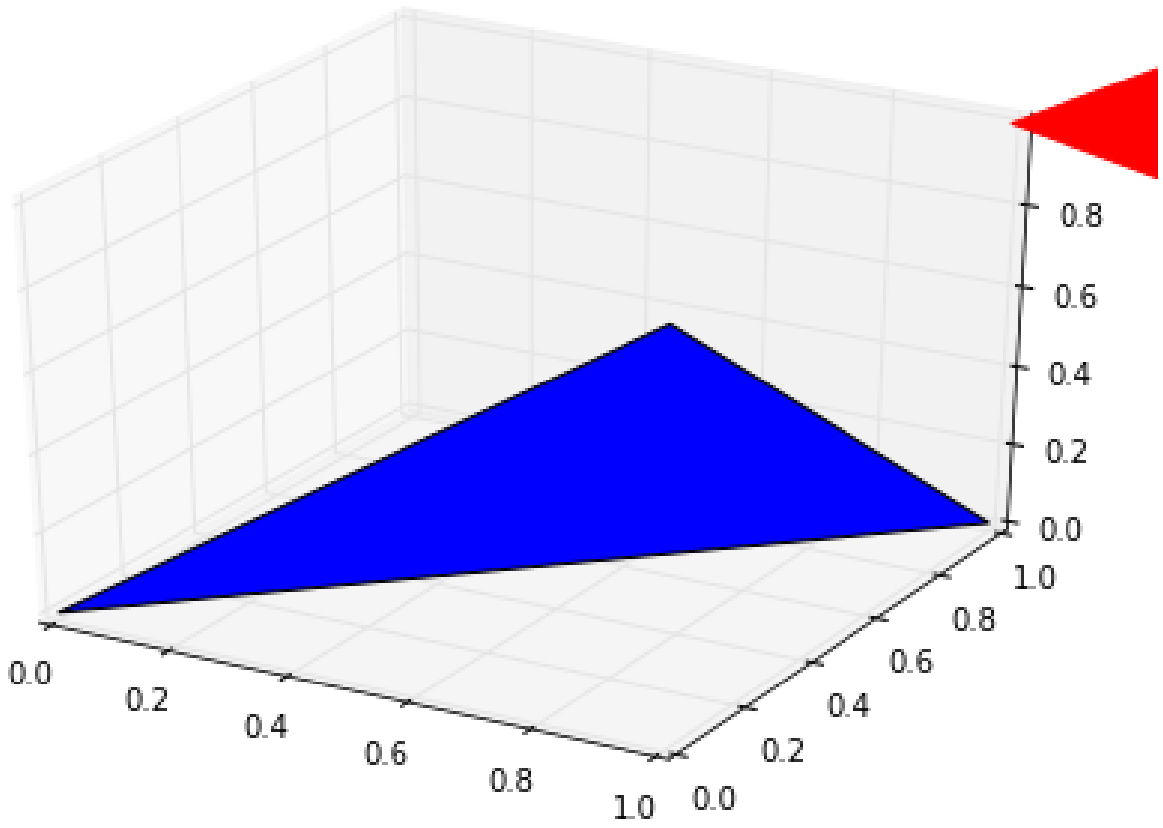



```
from matplotlib import pyplot as plt
from matplotlib.patches import Rectangle
someX, someY = 0.5, 0.5
fig, ax = plt.subplots()
currentAxis = plt.gca()
currentAxis.add_patch(Rectangle((someX - 0.1, someY - 0.1), 0.2, 0.2,
```

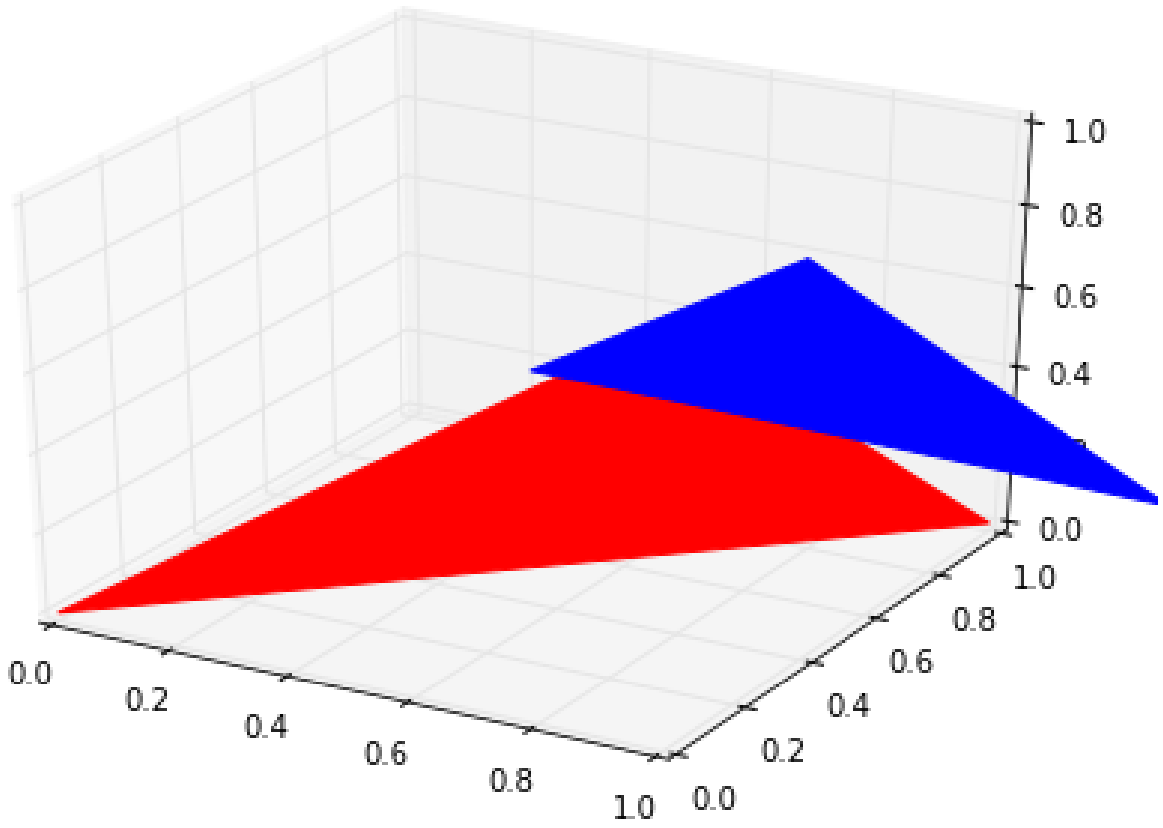
```
                                alpha=1, facecolor='none'))
plt.show()
```



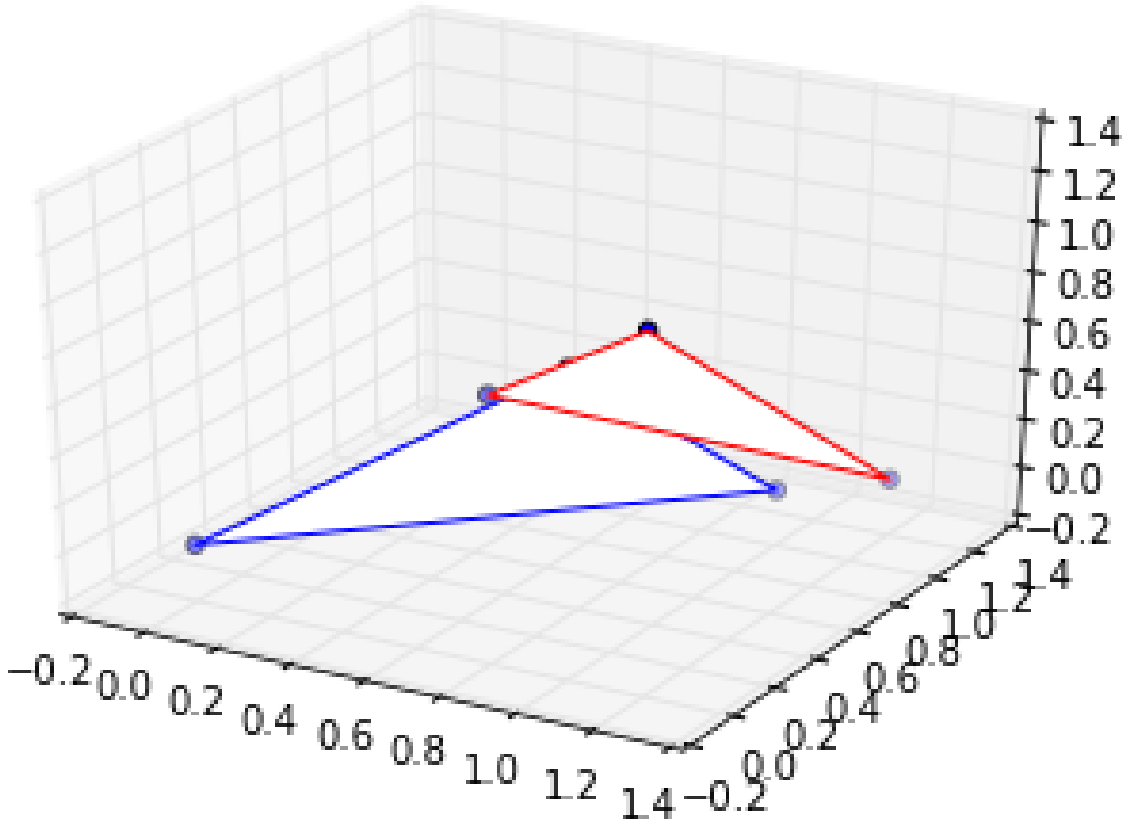

```
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import matplotlib.pyplot as plt
fig = plt.figure()
ax = Axes3D(fig)

x = [1,2,2]
y = [1,0,2]
z = [1,2,0]
verts = [zip(x, y,z)]
ax.add_collection3d(Poly3DCollection(verts,edgecolors='red', facecolors='red'))
x = [0,1,1]
y = [0,0,1]
z = [0,1,0]
verts = [zip(x, y,z)]
verts = [[(1,1,1), (2,0,2),(2,2,0)],[(0,0,0),(1,0,1),(1,1,0)]]
ax.add_collection3d(Poly3DCollection(verts))
plt.show()
```




```
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import matplotlib.pyplot as plt
fig = plt.figure()
ax = Axes3D(fig)

verts = [[(0.5,0.5,0.5), (1.2,0,1.2),(1.2,1.2,0)],[(0,0,0),(1,0,1),(1,1,0)]]
ax.add_collection3d(Poly3DCollection(verts, edgecolors=['blue','red'], facecolors=['blue','red'
plt.show()
```




```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = [0.5, 1.2, 1.2, 0, 1, 1]
y = [0.5, 0,   1.2, 0, 0, 1]
z = [0.5, 1.2, 0,   0, 1, 0]

poly3d = [[(0.5,0.5,0.5), (1.2,0,1.2),(1.2,1.2,0)],[(0,0,0),(1,0,1),(1,1,0)]]
ax.scatter(x,y,z)
ax.add_collection3d(Poly3DCollection(poly3d, edgecolors=['red','blue'], facecolors='w', linewid

plt.show()
```



2.11 Reference

- http://www.byteofpython.info/
- http://woodpecker.org.cn/abyteofpython_cn/chinese/index.html
- http://www.python-course.eu/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-189-a-gentle-introduction-to-programming-using-pyt
- http://my.oschina.net/taogang/blog/286954

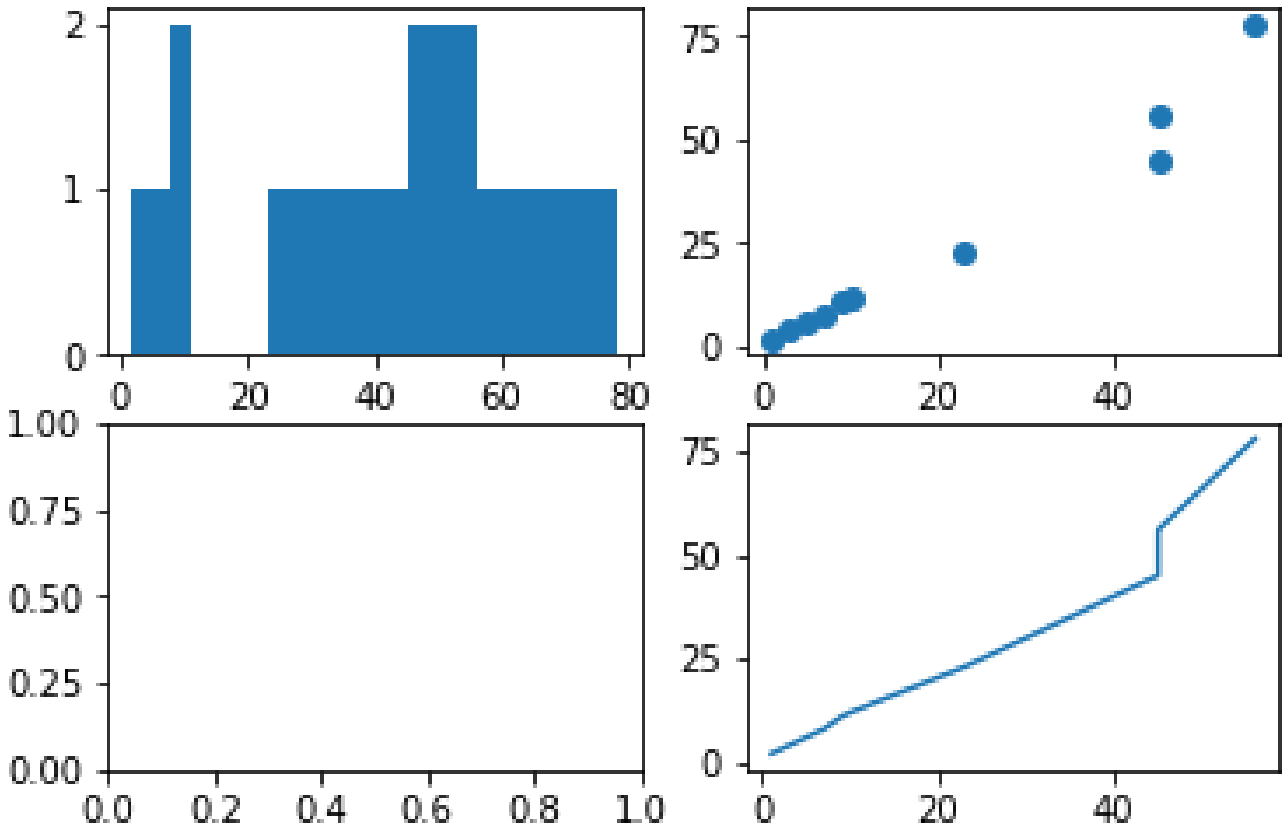
3 Python 作图

3.1 绘图基础

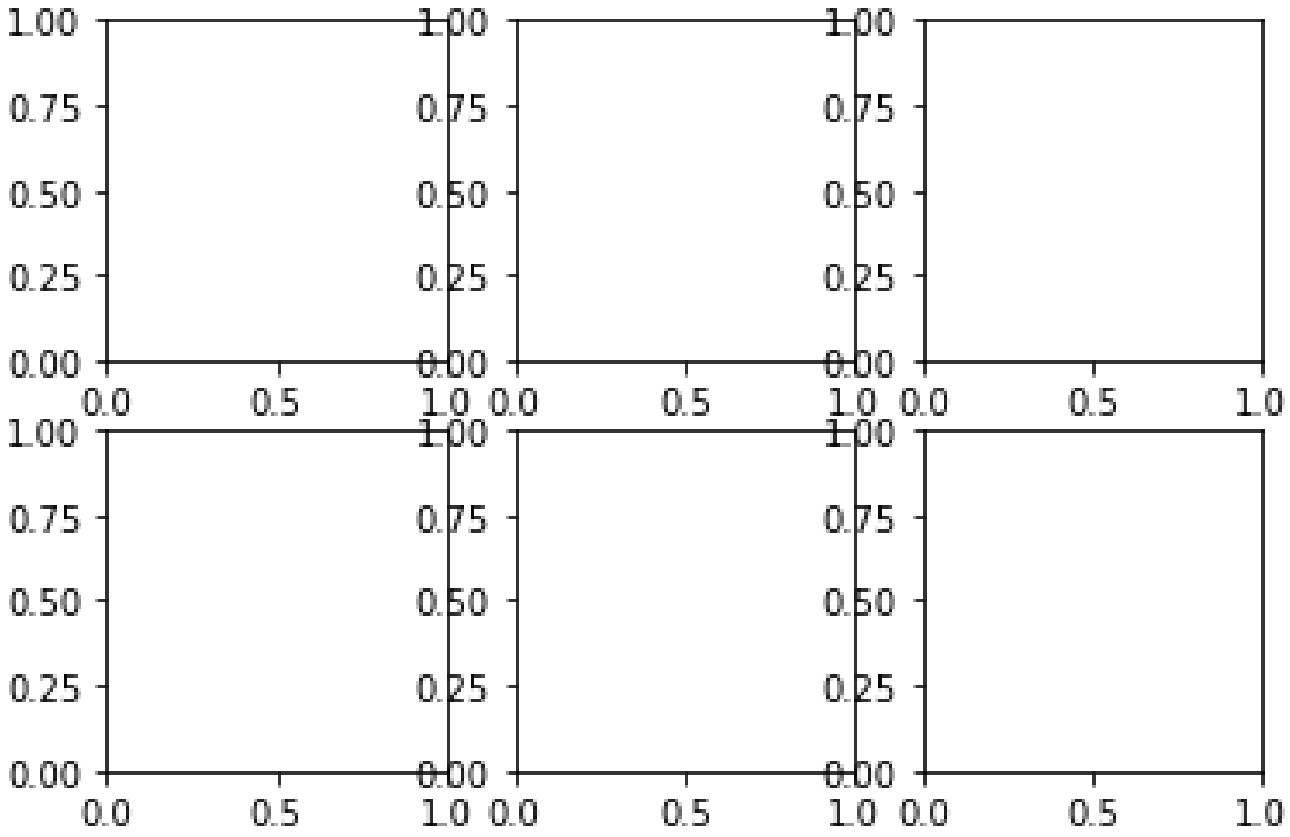
3.1.1 Figure 和 Subplot

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from numpy.random import randn
```

```
x = [1, 3, 5, 7, 9, 10, 23, 45, 45, 56]
y = [2, 4, 6, 8, 11, 12, 23, 45, 56, 78]


fig = plt.figure()  # 创建一个 Figure
ax1 = fig.add_subplot(2, 2, 1)  # 创建 4 个图的 Figure 对象，最后的 1 为选中第一个
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

ax1.hist(x, y)  # 在第一图中绘制直方图
ax2.scatter(x, y)  # 散点图
ax4.plot(x, y)  # 线图

plt.show()
```




```
fig, axes = plt.subplots(2, 3)  # 创建一个 Figure 绘制 2 X3 图
fig
```




```
axes
```

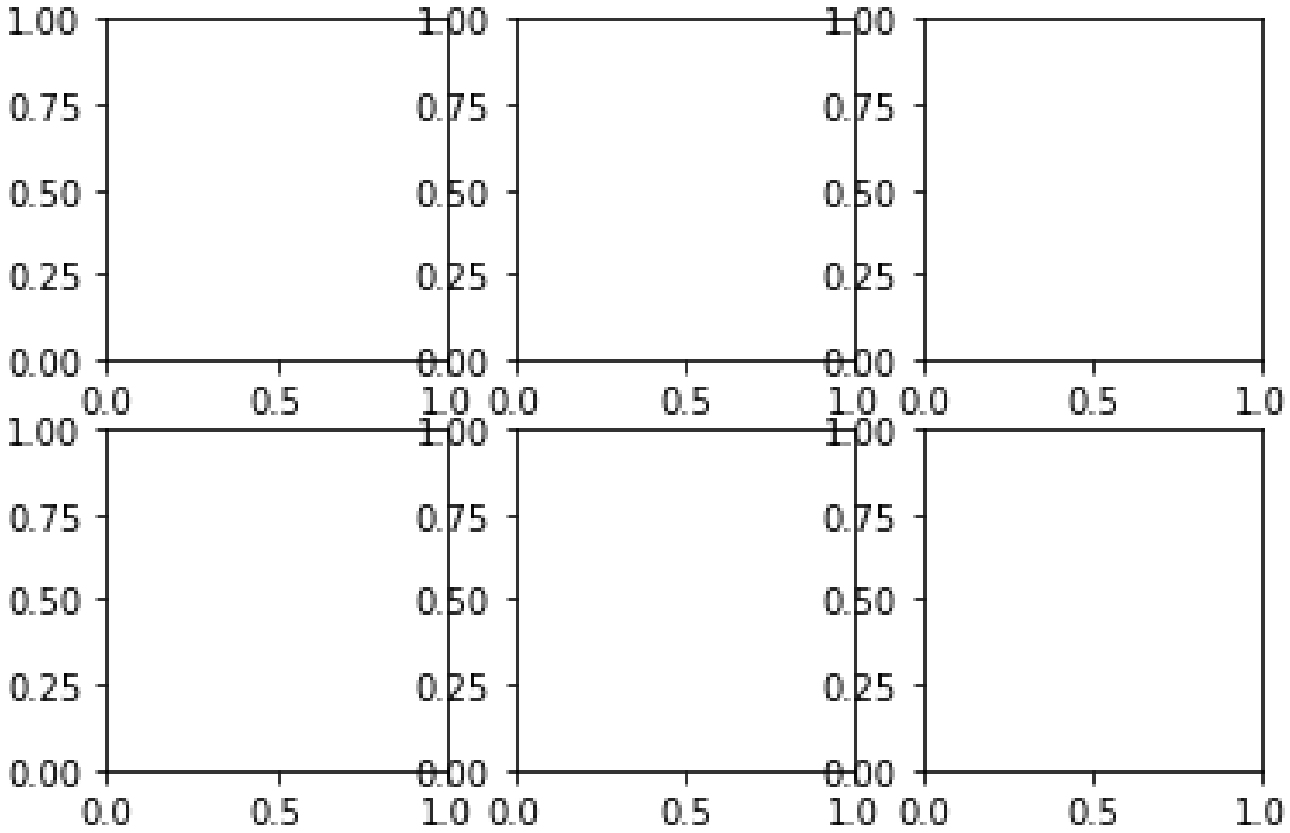
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8436a71780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f842f2438d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f842f27d9e8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f842f236978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f842f1f19b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f842f199710>]], dtype=object)
```

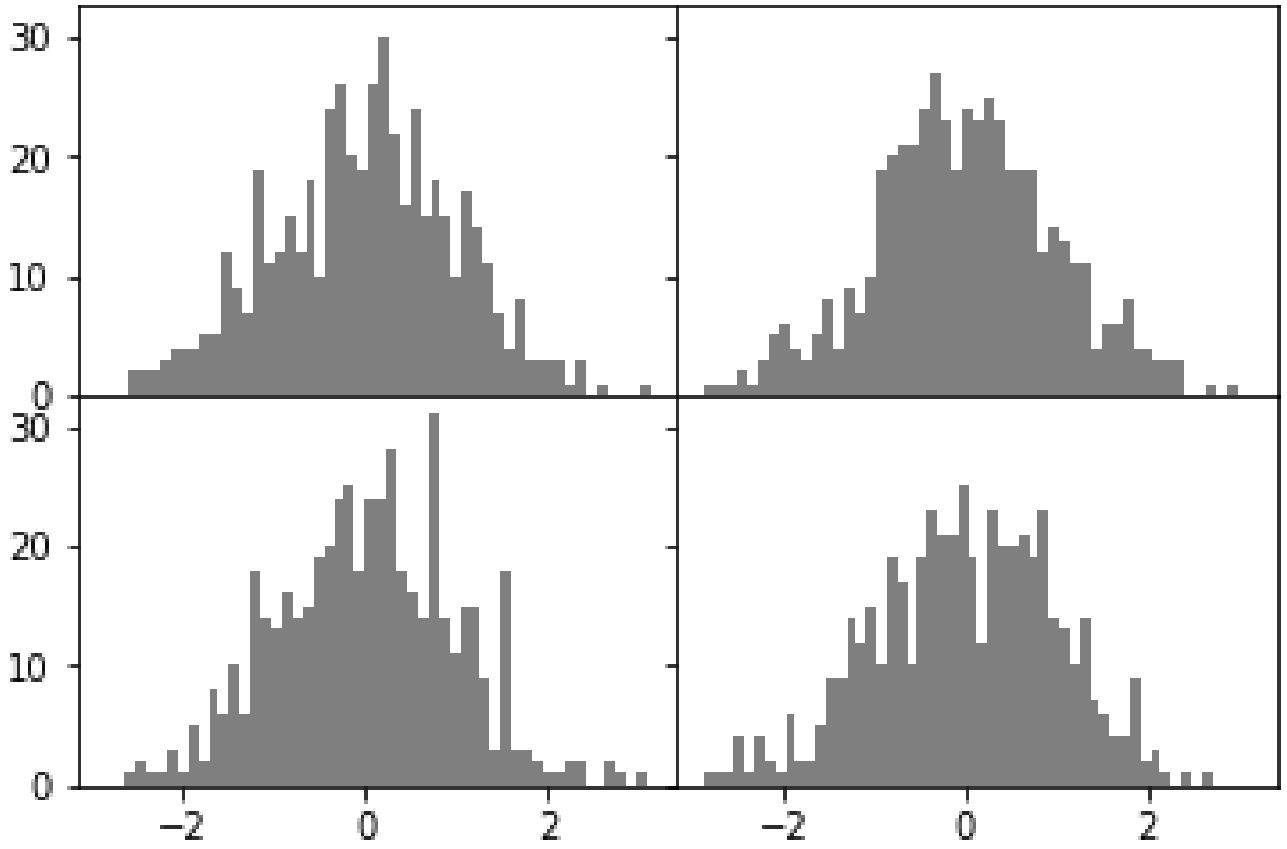
创建 2 X 3 图像，可以相当于对二维数组进行索引 参数 说明 nrows subplot 行数 ncols subplot 列数 sharex 所有图使用相同的 x 轴 sharey 所有图使用相同的 y 轴 subplot_kw 用于创建各 subplot 的关键字字典 |

3.1.2 调整 subplot 周围间距

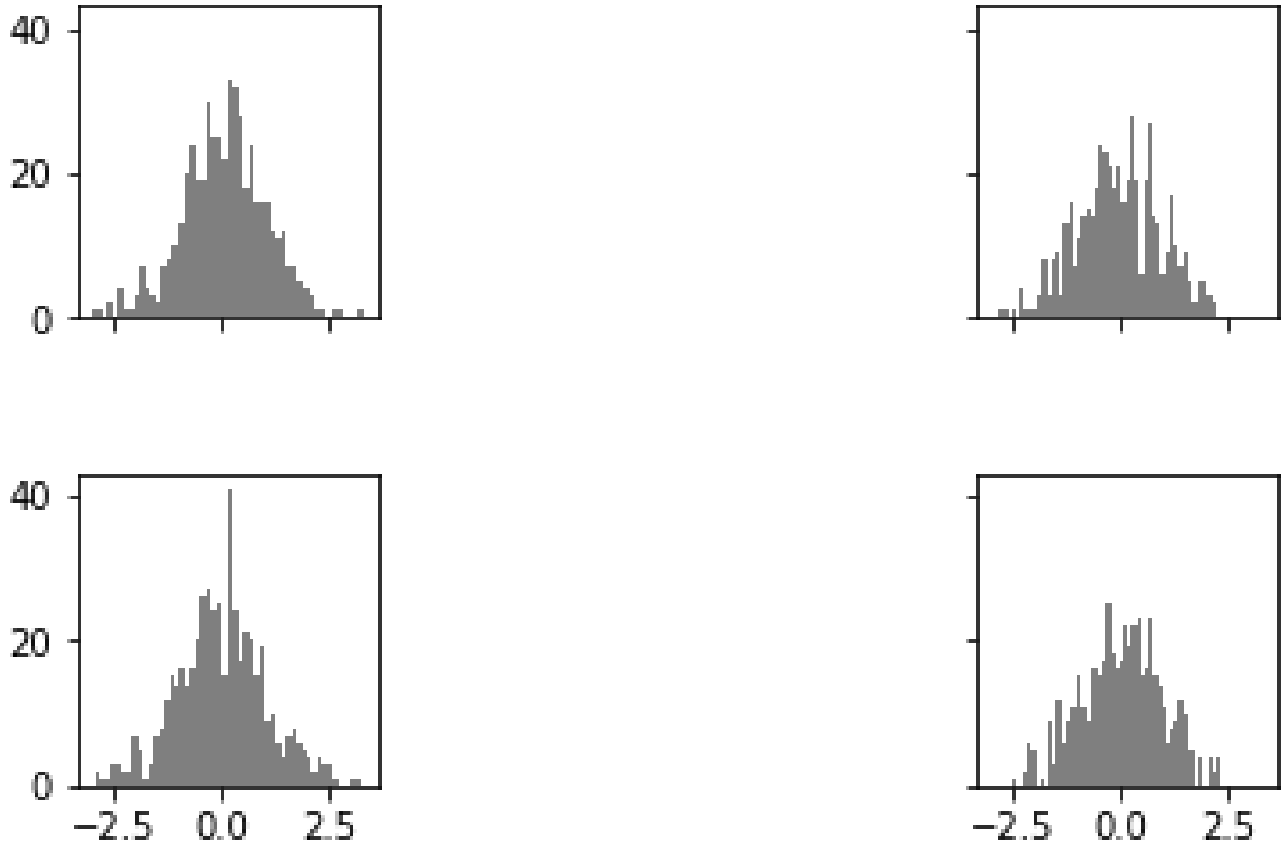
```
subplots_ajust(left=None,bottom=None,right=None,top=None,wspace=None,hspace=None)
wspace和hspace控制宽度和高度
```

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```

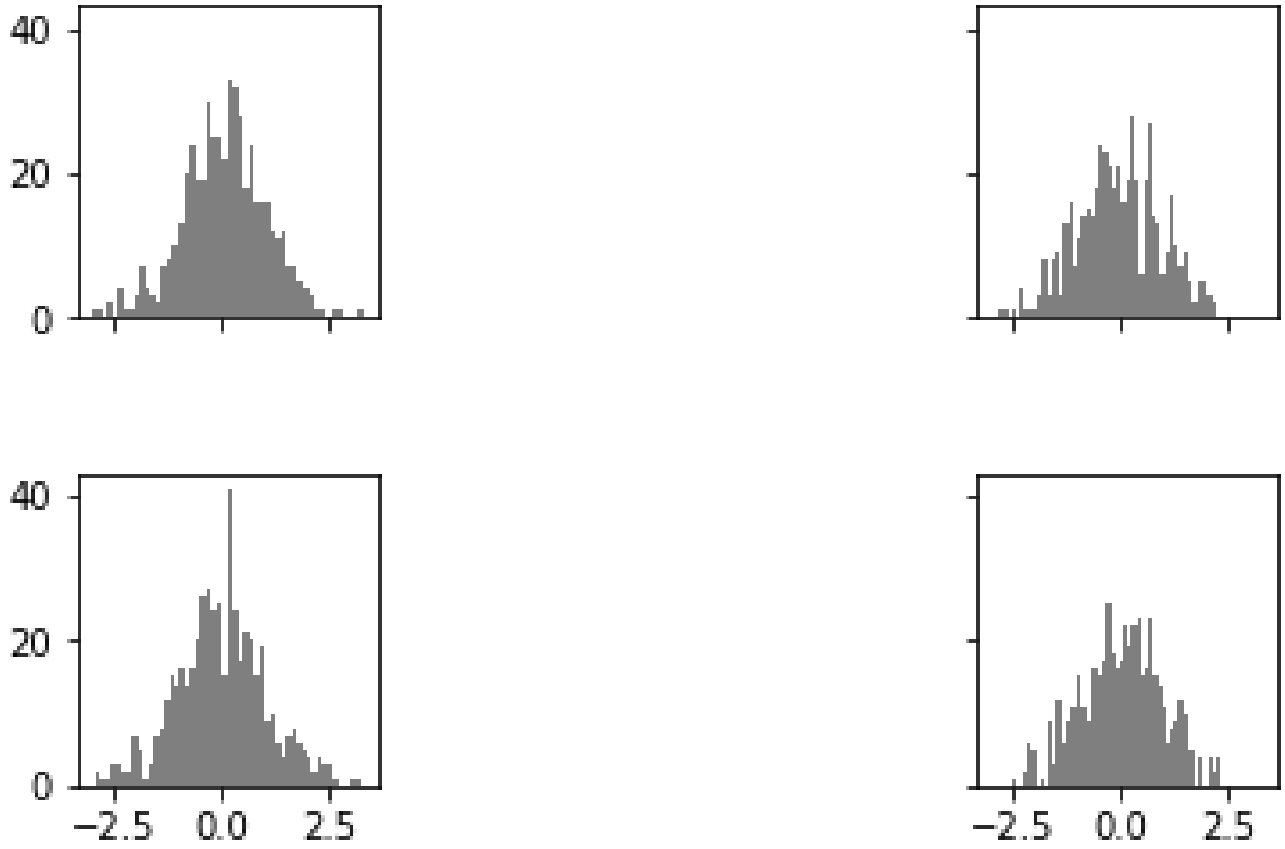





```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=2, hspace=0.5)
plt.show()
```

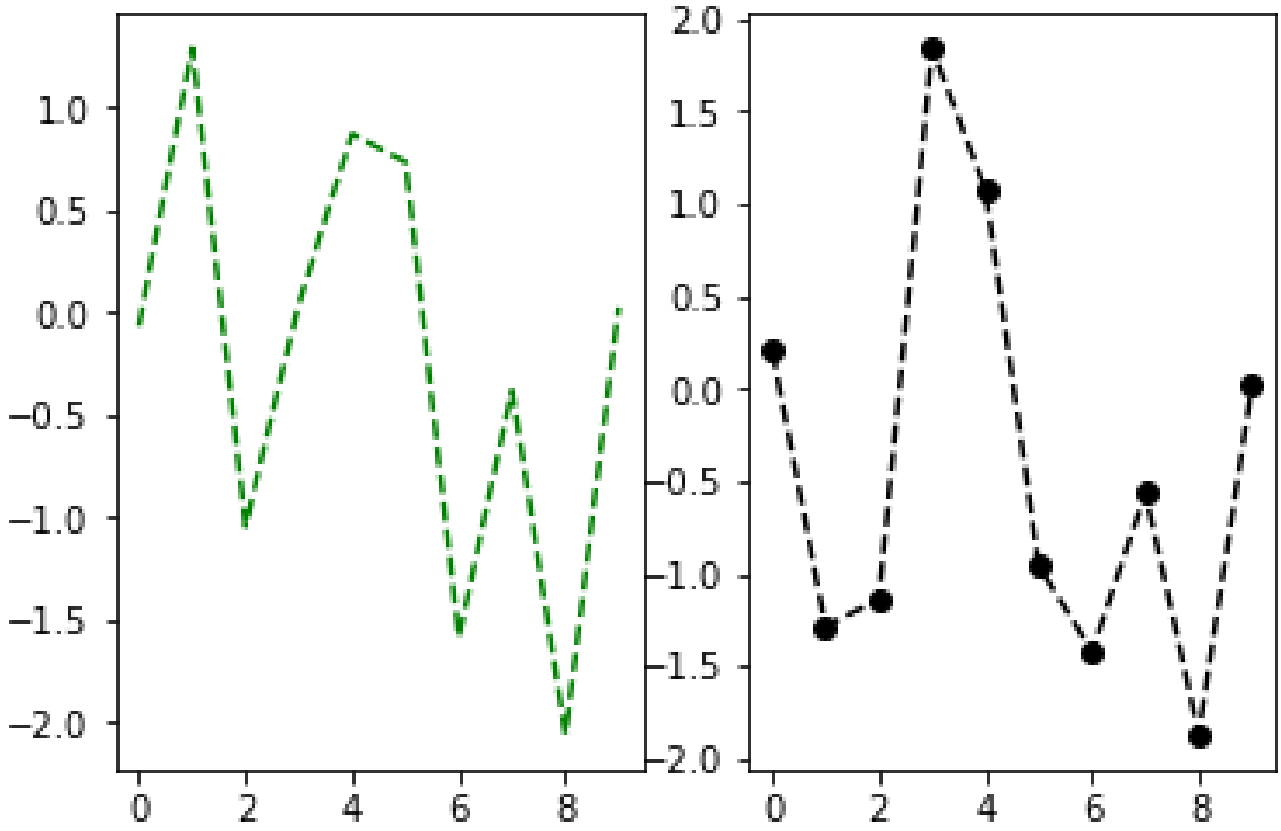



```
fig
```



3.1.3 颜色标记和线型

```
"""
绘制绿色虚线
ax.plot(x,y,'g--')
另一种方式
ax.plot(x,y,linestyle='--',color='g')
标记点（maker）
"""
fig, axes = plt.subplots(1, 2)

axes[0].plot(randn(10), 'g--')   # green ---
axes[1].plot(randn(10), 'ko--')   # k：black o：圆点

plt.show()
```

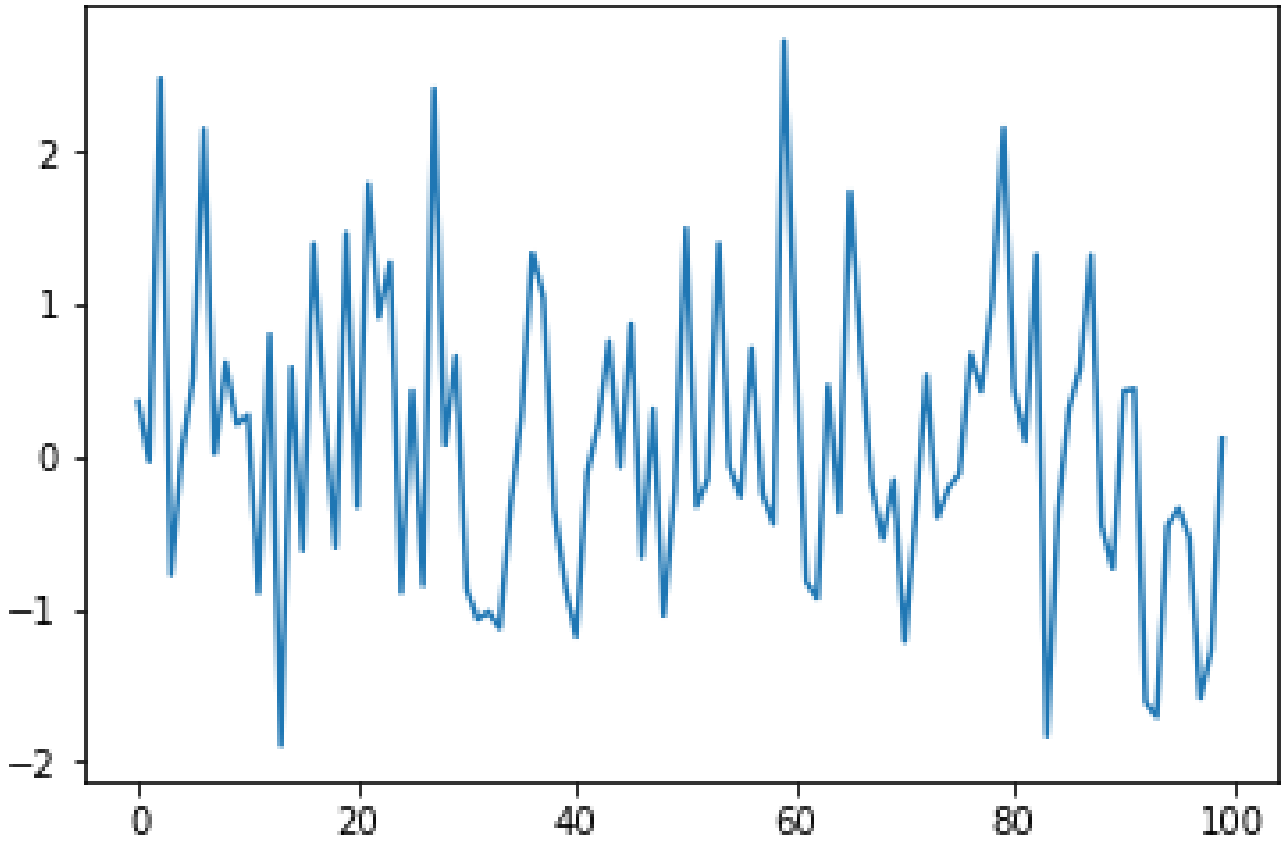



```
character   description
'-' solid line style
'--'    dashed line style
'-.'    dash-dot line style
':' dotted line style
'.' point marker
',' pixel marker
'o' circle marker
'v' triangle_down marker
'^' triangle_up marker
'<' triangle_left marker
'>' triangle_right marker
'1' tri_down marker
'2' tri_up marker
'3' tri_left marker
'4' tri_right marker
's' square marker
'p' pentagon marker
'*' star marker
'h' hexagon1 marker
'H' hexagon2 marker
'+' plus marker
```

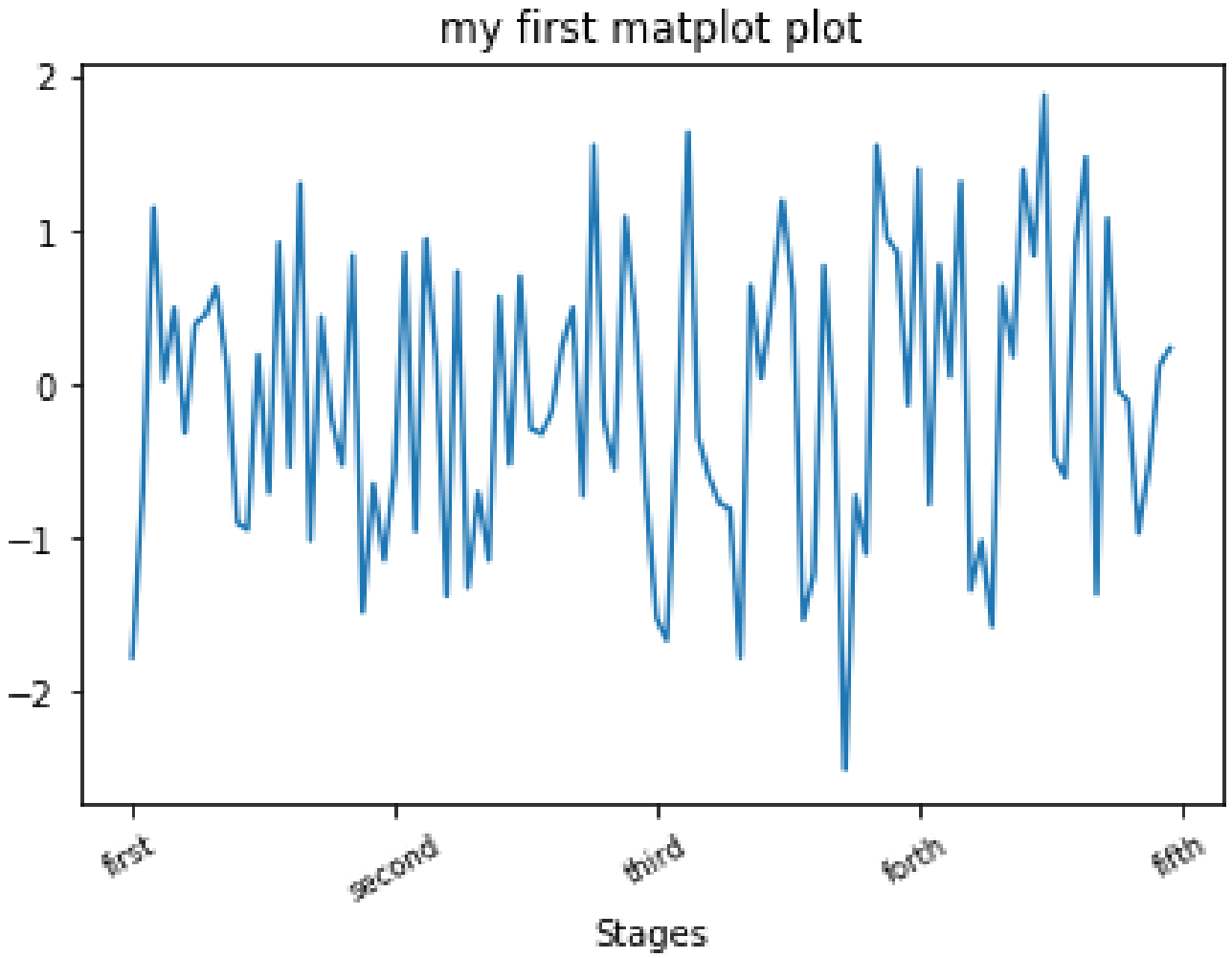
```
'x' x marker
'D' diamond marker
'd' thin_diamond marker
'|' vline marker
'_' hline marker

The following color abbreviations are supported:

character   color
‘b’ blue
‘g’ green
‘r’ red
‘c’ cyan(青色)
‘m’ magenta（紫红色）
‘y’ yellow
‘k’ black
‘w’ white

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot
```

3.1.4 刻度、标签和图例

```
fig = plt.figure()  # 创建一个 Figure
ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(100))
plt.show()
```




```
""" 修改上图的轴"""

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(100))

ticks = ax.set_xticks([0, 25, 50, 75, 100])  # 设置刻度
labels = ax.set_xticklabels(
    ['first', 'second', 'third', 'forth', 'fifth'], rotation=30, fontsize='small')  # 设置 x 轴标

ax.set_title('my first matplot plot')  # 设置图片标题

ax.set_xlabel('Stages')  # 设置 x 轴名称

plt.show()
```



3.1.5 添加图例 **legend**

https://matplotlib.org/api/legend_api.html?highlight=legend#module-matplotlib.legend

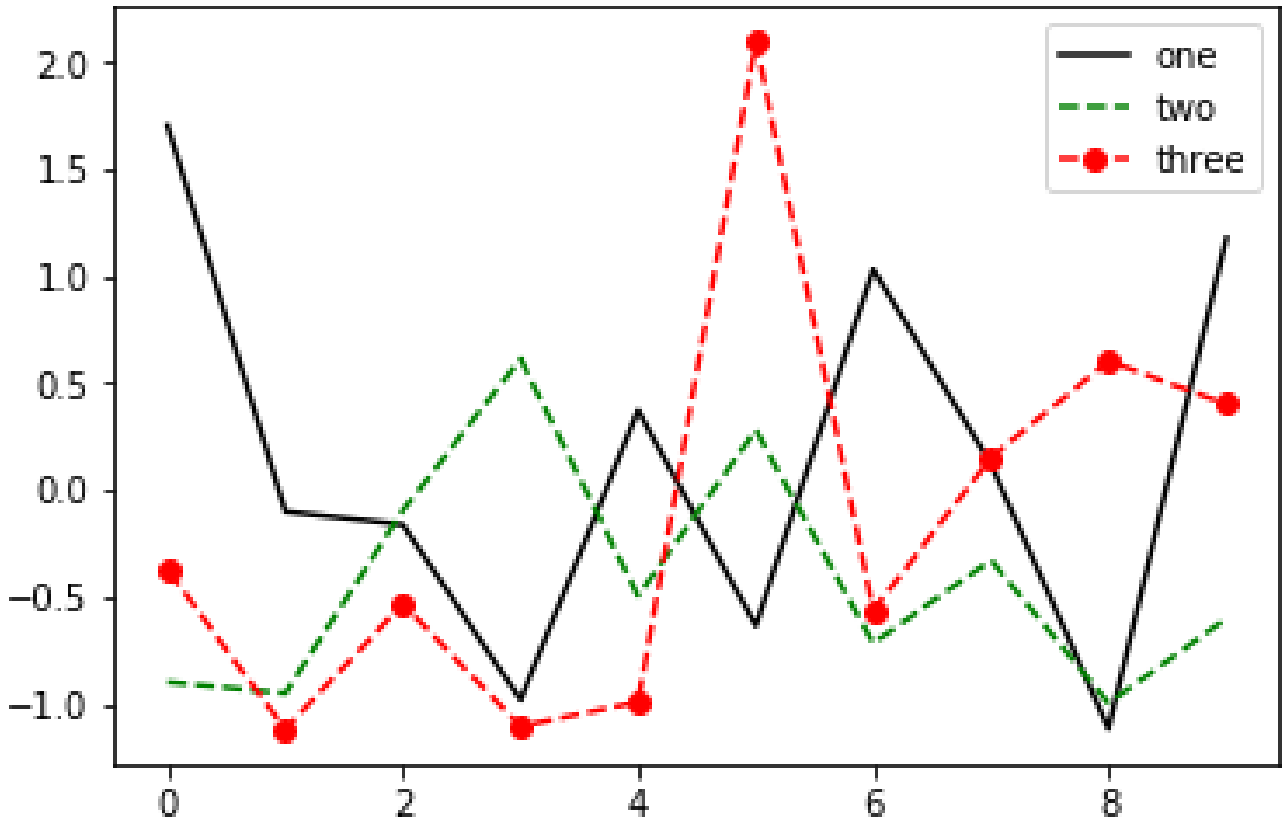
‘best’	0
‘upper right’	1
‘upper left’	2
‘lower left’	3
‘lower right’	4
‘right’	5
‘center left’	6
‘center right’	7
‘lower center’	8
‘upper center’	9
‘center’	10

bbox_to_anchor= (0.5,0.8)

bbox_to_anchor被赋予的元组中，第一个数值用于控制legend的左右移动，值越大越向右边移动，

第二个数值用于控制legend的上下移动，值越大，越向上移动

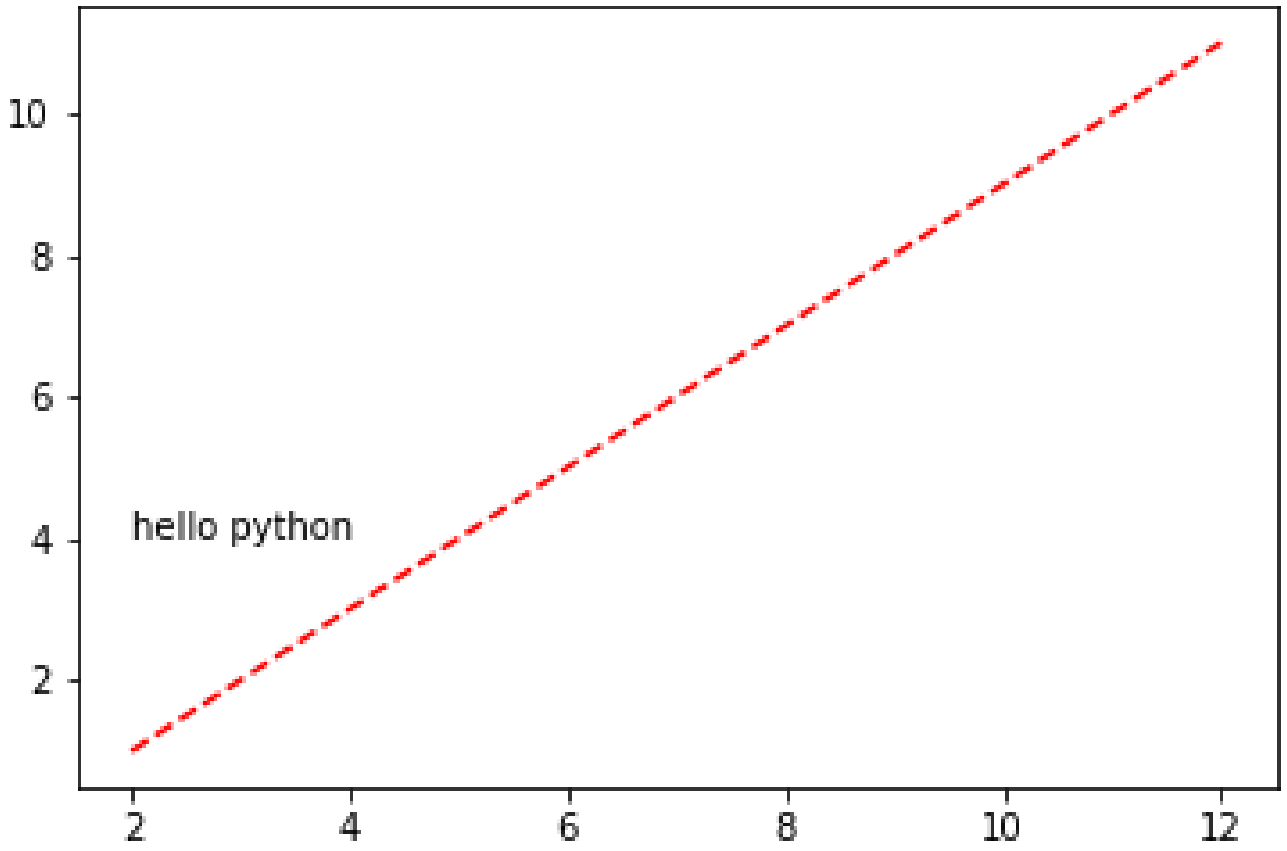
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(randn(10), 'k', label='one')  # 画一条线，k 黑色
ax.plot(randn(10), 'g--', label='two')  # 画第二条线，g 绿色 - -类型
ax.plot(randn(10), 'ro--', label='three')  # 画第三条线红色 ，类型 ...

ax.legend(loc=0, bbox_to_anchor=(1, 1))


plt.show()
```



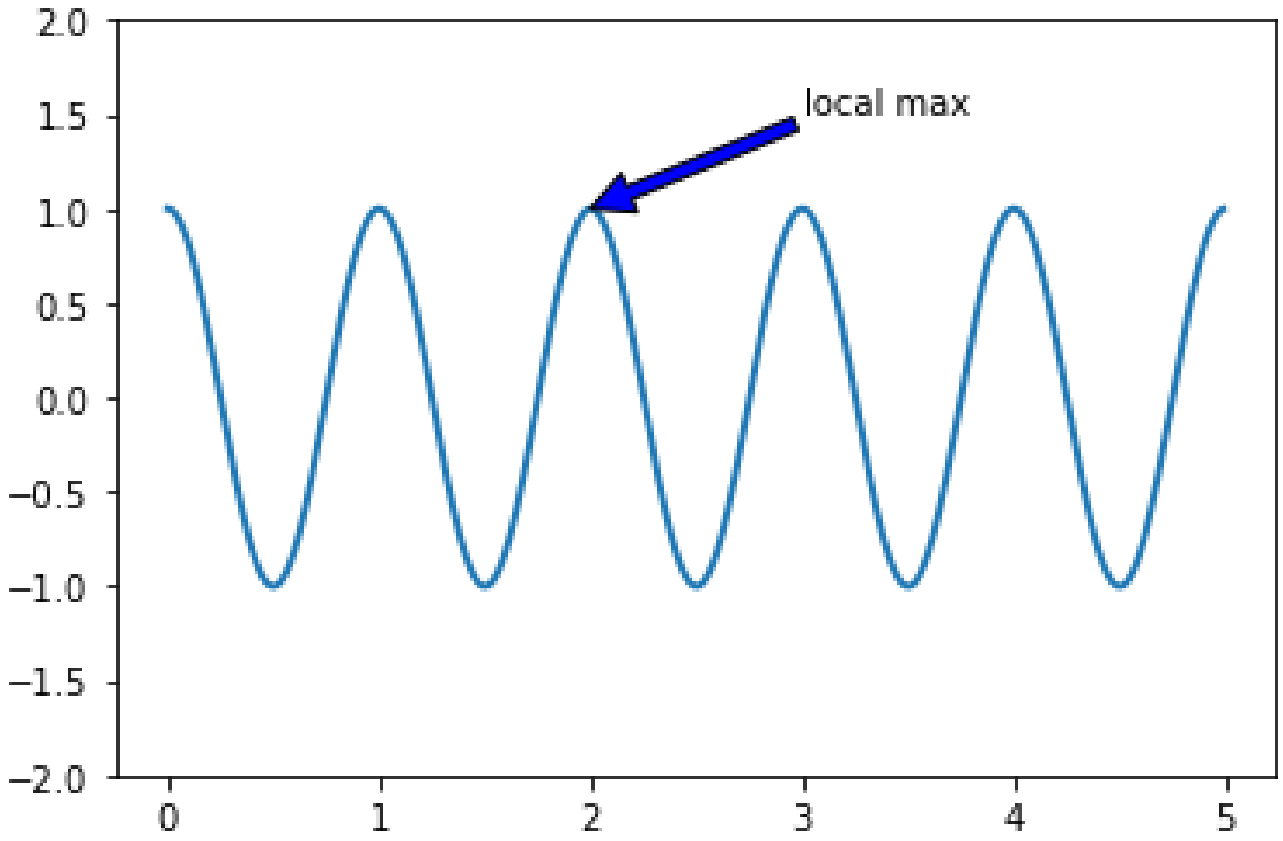
3.1.6 注解

```
x = [2, 4, 6, 8, 10, 12]
y = [1, 3, 5, 7, 9, 11]
fig = plt.figure()
```

```
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y, 'r--')

ax.text(2, 4, 'hello python')
plt.show()
```




```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='blue')   # xy=（要注释的点），xytext=(注释在图中的位置）
             )

plt.ylim(-2, 2)
plt.show()
```



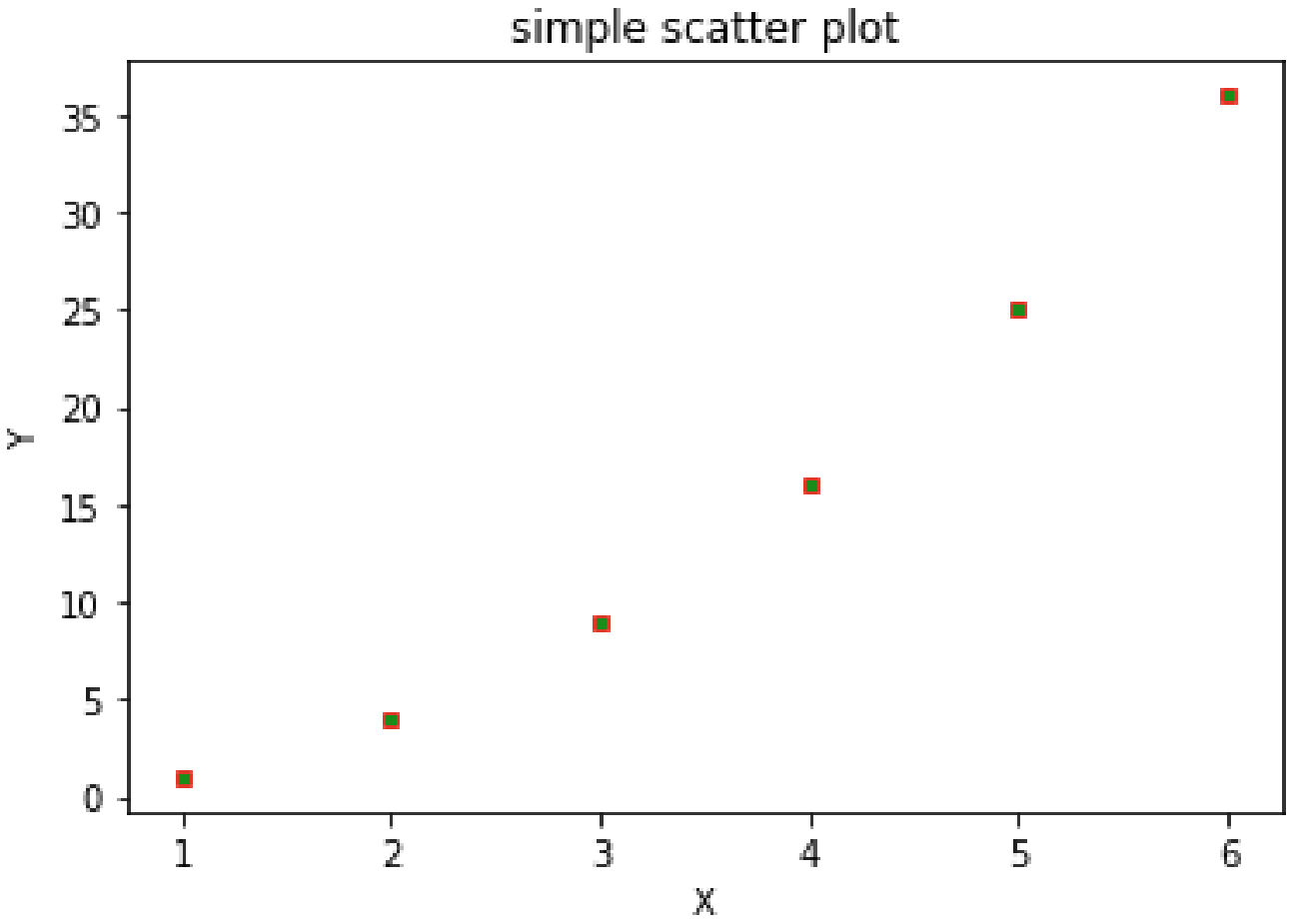
3.1.7 图片保存

```
x = [2, 4, 6, 8, 10, 12]
y = [1, 3, 5, 7, 9, 11]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y, 'r--')

ax.text(2, 4, 'hello python')

# bbox_inches 减除当前图片周围空白部分
plt.savefig('figpath.jpg', dpi=300, bbox_inches='tight')
```

3.2 绘图实例

3.2.1 绘制散点图

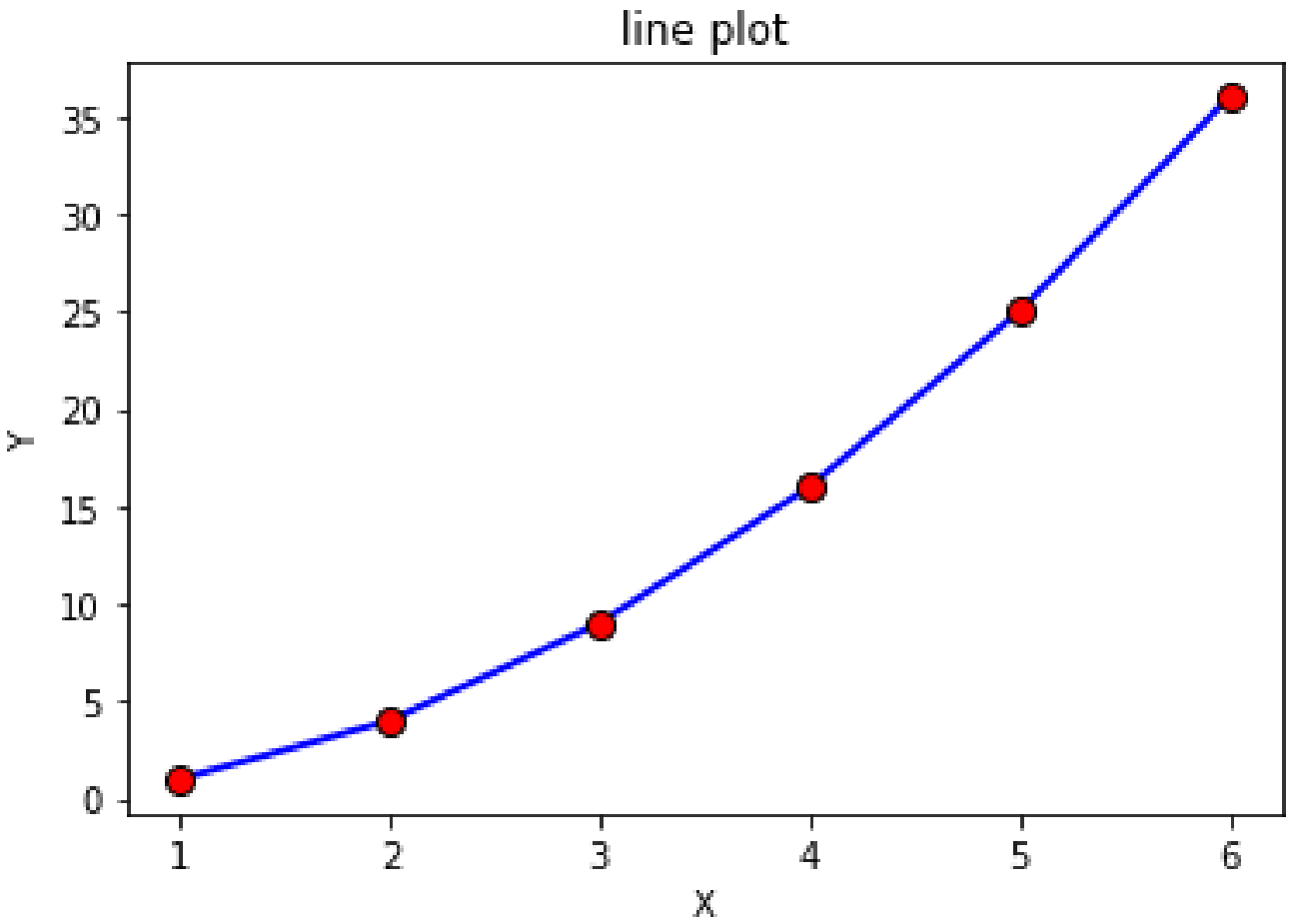
```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]
plt.scatter(x,  # x 轴数据
            y,  # y 轴数据
            s=20,  # 设置点的大小
            c='green',  # 设置点的颜色
            marker='s',  # 设置点的形状
            alpha=0.9,  # 设置点的透明度
            linewidths=0.8,  # 设置散点边界的粗细
            edgecolors='red'  # 设置散点边界的颜色
            )

plt.title('simple scatter plot')
plt.xlabel('X')  # x 轴名称
plt.ylabel('Y')

plt.show()  # 展示绘图
```



3.2.2 折线图

```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]

plt.plot(x,  # x 轴数据
         y,  # y 轴数据
         linestyle='-',  # 折线类型
         linewidth=2,  # 折线宽度
         color='blue',  # 折线颜色
         marker='o',  # 点的形状
         markersize=8,  # 点的大小
         markeredgecolor='black',  # 点的边框色
         markerfacecolor='red')  # 点的填充色


# 添加标题和坐标轴标签
plt.title('line plot')
```

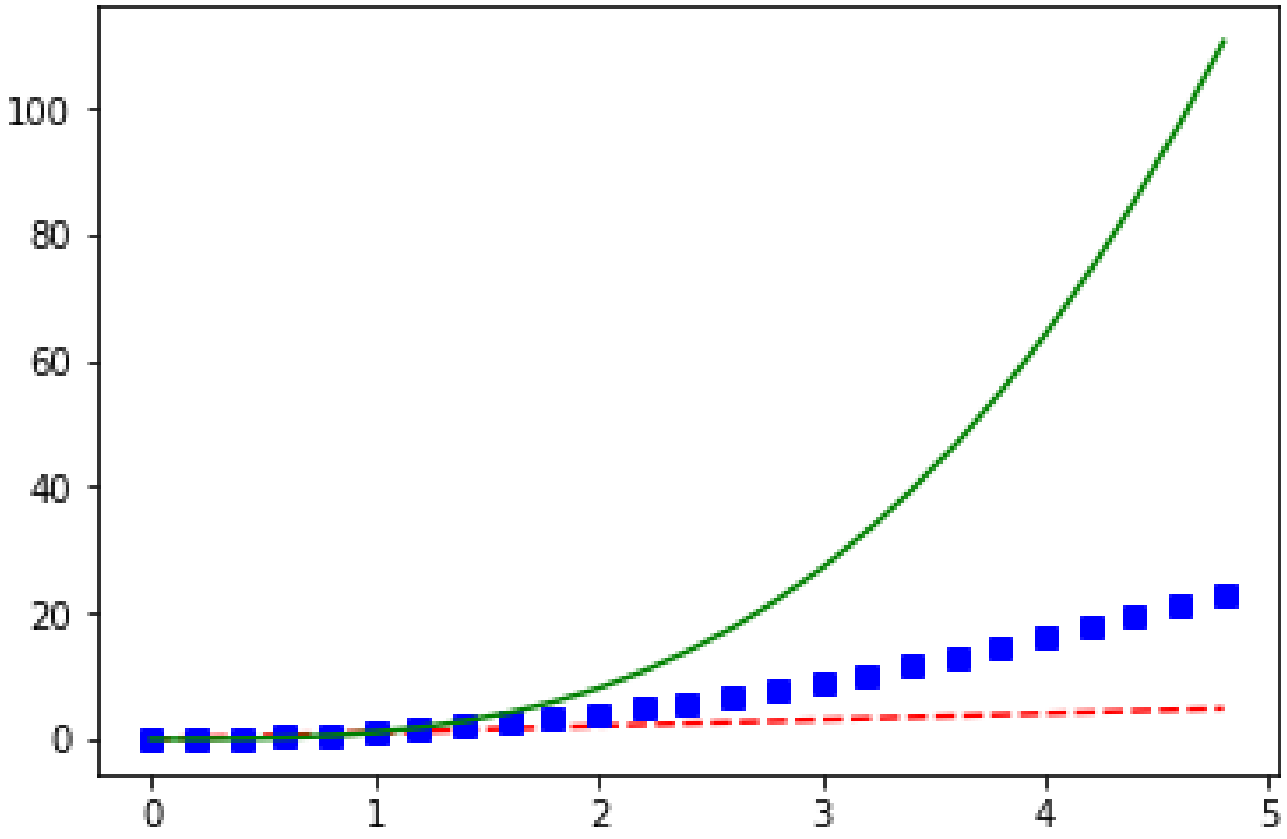
```
plt.xlabel('X')
plt.ylabel('Y')

plt.show()
```



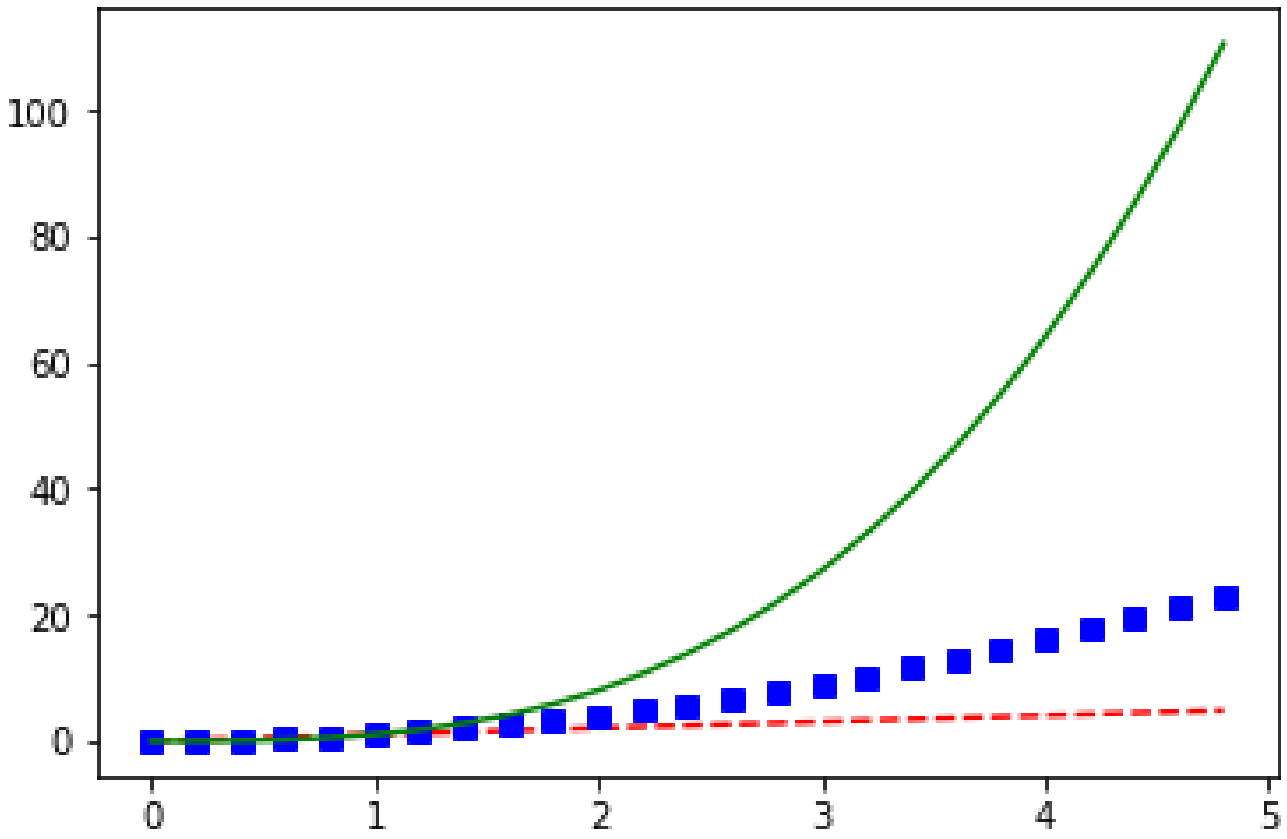

```
import numpy as np
import matplotlib.pyplot as plt


t = np.arange(0., 5., 0.2)

t
```

```
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ,  1.2,  1.4,  1.6,  1.8,  2. ,
        2.2,  2.4,  2.6,  2.8,  3. ,  3.2,  3.4,  3.6,  3.8,  4. ,  4.2,
        4.4,  4.6,  4.8])
```

```
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g')
plt.show()
```

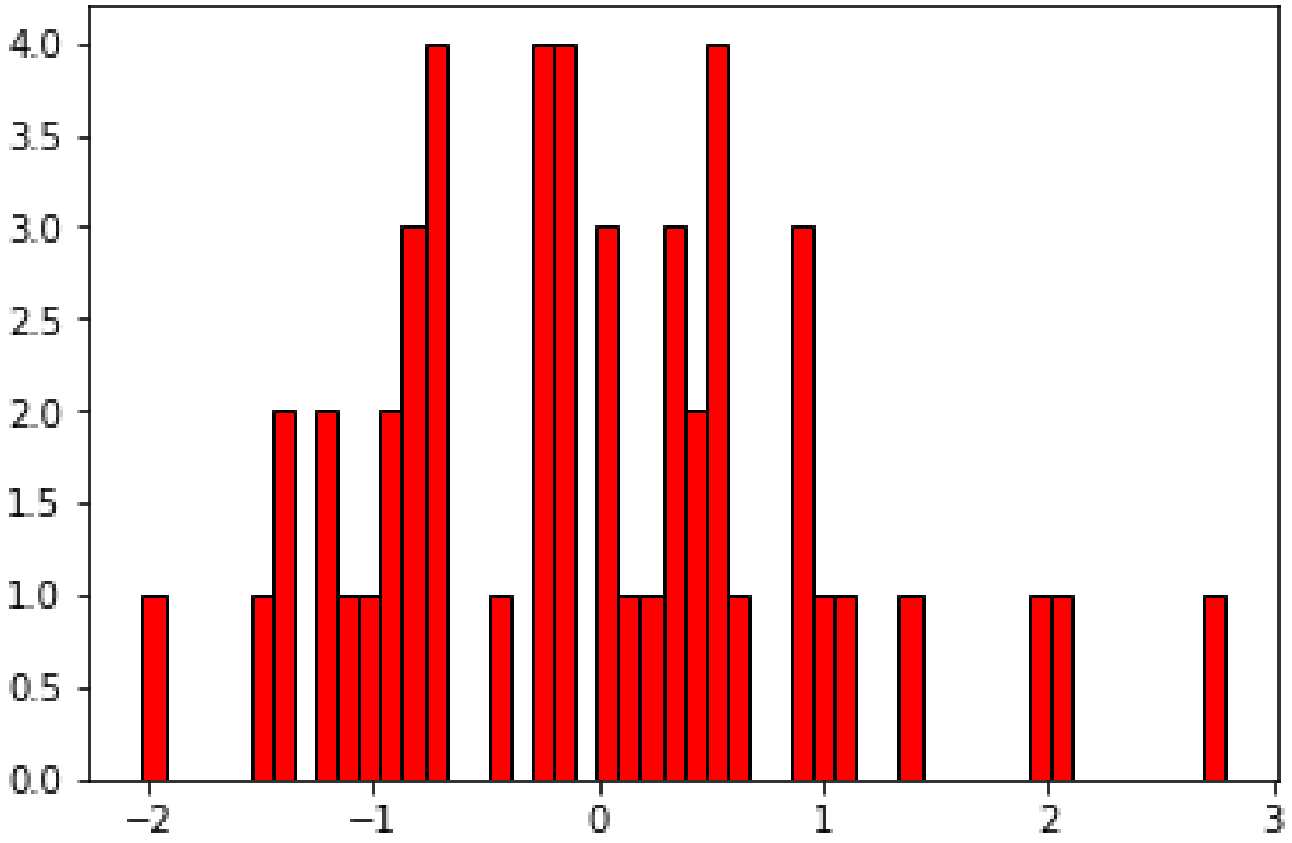



```
# 同一效果
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(t, t, 'r--')
ax.plot(t, t**2, 'bs')  # 'bs' 表示 blue square marker
ax.plot(t, t**3, 'g')
fig
```



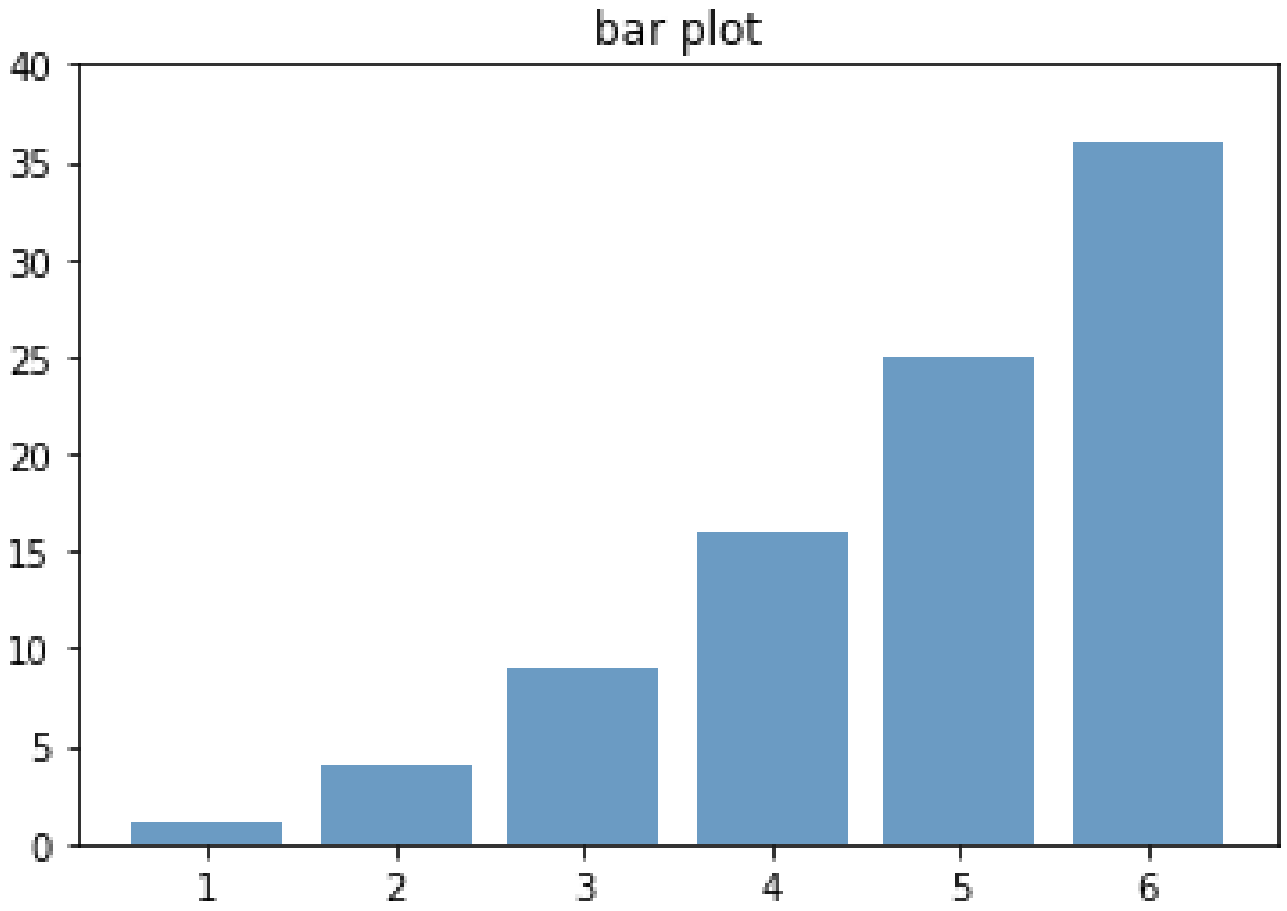
3.2.3 直方图

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

plt.hist(np.random.randn(50),  # 绘图数据
         bins=50,  # 指定直方图的条形数
         color='red',  # 指定填充色
         edgecolor='k')  # 指定直方图的边界色 k :black
plt.show()
```

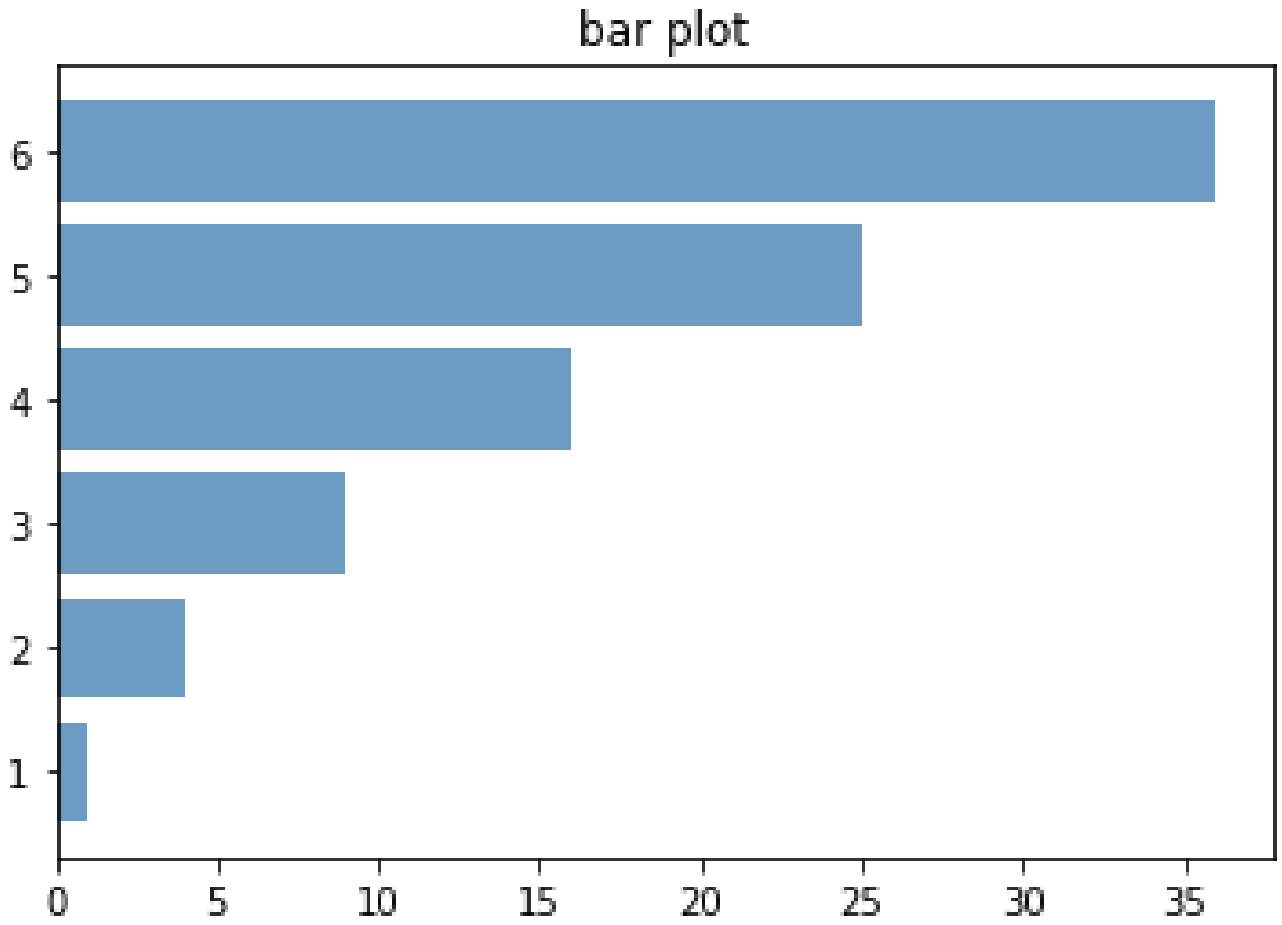


3.2.4 直条图

```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]

plt.bar(x, y,
        color='steelblue',
        alpha=0.8)

plt.title('bar plot')

plt.ylim([0, 40])
plt.show()
```

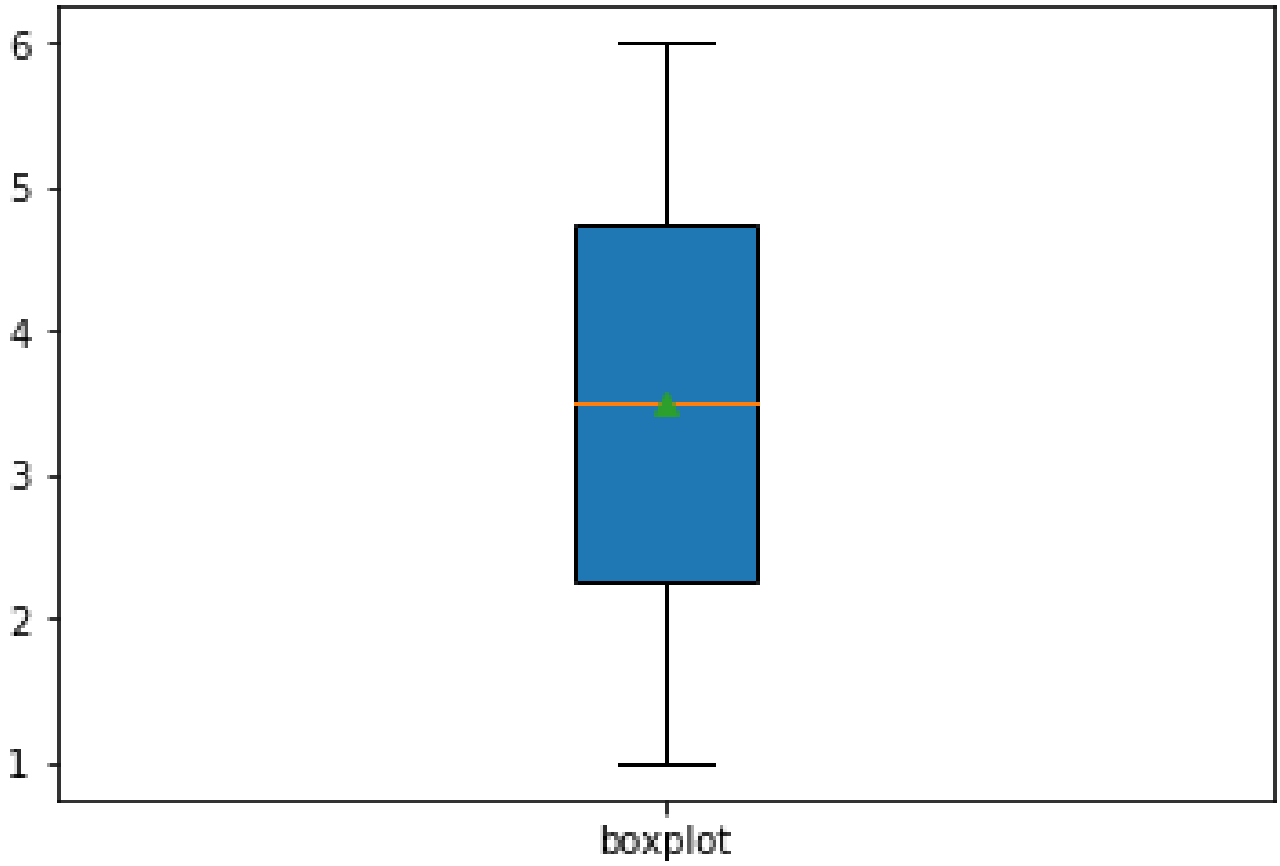



```
x = [1, 2, 3, 4, 5, 6]
y = [1, 4, 9, 16, 25, 36]

plt.barh(x, y,
         color='steelblue',
         alpha=0.8)

plt.title('bar plot')

plt.show()
```



3.2.5 箱线图

```
x = [1, 2, 3, 4, 5, 6]
plt.boxplot(x,
            patch_artist=True,   # 箱体添加颜色
            labels=['boxplot'],   # 添加具体的标签名称
            showmeans=True)

# 显示图形
plt.show()
```




```
np.random.seed(2)   # 设置随机种子
df = pd.DataFrame(np.random.rand(5, 4),
                  columns=(['A', 'B', 'C', 'D']))
df
```

```
<tr style="text-align: right;">
  <th></th>
  <th>A</th>
  <th>B</th>
  <th>C</th>
  <th>D</th>
</tr>
```

```
<tr>
  <th>0</th>
  <td>0.435995</td>
  <td>0.025926</td>
  <td>0.549662</td>
  <td>0.435322</td>
</tr>
```

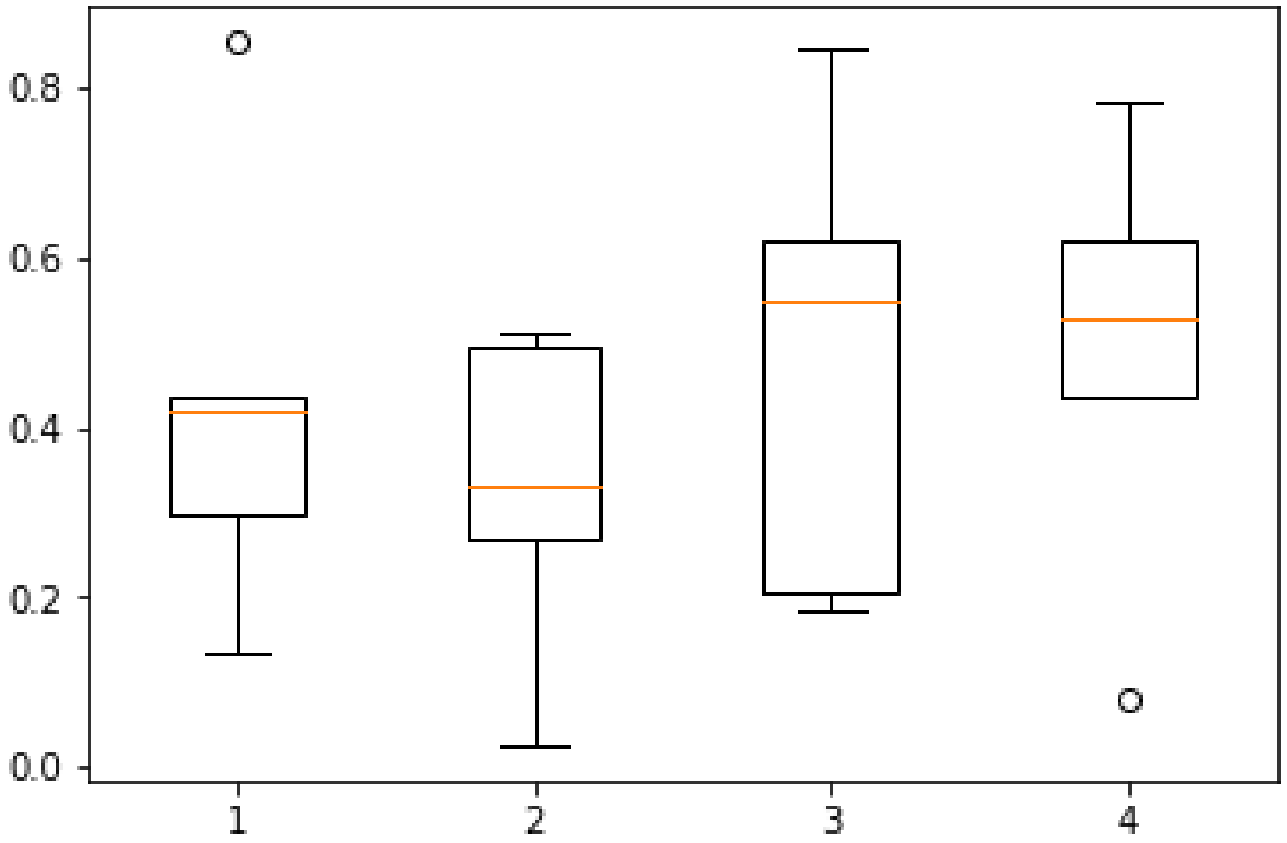
```
<tr>
  <th>1</th>
  <td>0.420368</td>
  <td>0.330335</td>
  <td>0.204649</td>
  <td>0.619271</td>
</tr>
<tr>
  <th>2</th>
  <td>0.299655</td>
  <td>0.266827</td>
  <td>0.621134</td>
  <td>0.529142</td>
</tr>
<tr>
  <th>3</th>
  <td>0.134580</td>
  <td>0.513578</td>
  <td>0.184440</td>
  <td>0.785335</td>
</tr>
<tr>
  <th>4</th>
  <td>0.853975</td>
  <td>0.494237</td>
  <td>0.846561</td>
  <td>0.079645</td>
</tr>
```

```
data = []
for i in range(4):
    data.append(df.iloc[:, i])
data
```

```
[0    0.435995
 1    0.420368
 2    0.299655
 3    0.134580
 4    0.853975
 Name: A, dtype: float64, 0    0.025926
 1    0.330335
 2    0.266827
 3    0.513578
 4    0.494237
 Name: B, dtype: float64, 0    0.549662
 1    0.204649
```

```
 2    0.621134
 3    0.184440
 4    0.846561
 Name: C, dtype: float64, 0    0.435322
 1    0.619271
 2    0.529142
 3    0.785335
 4    0.079645
 Name: D, dtype: float64]
```

```
plt.boxplot(data)
plt.show()
```

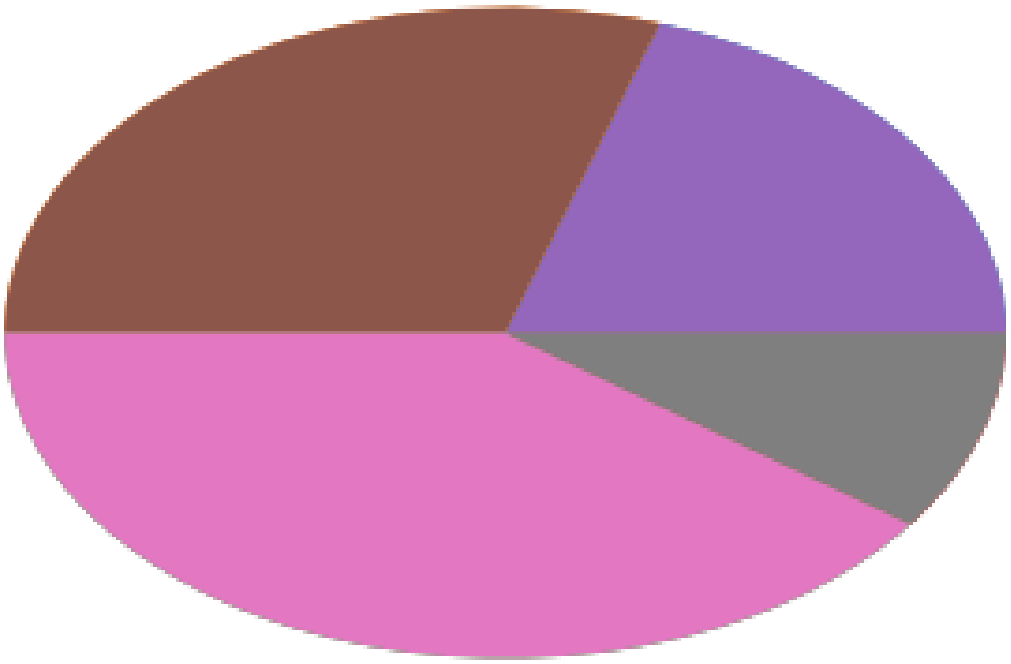



```
plt.boxplot(x, notch=None, sym=None, vert=None,
            whis=None, positions=None, widths=None,
            patch_artist=None, meanline=None, showmeans=None,
            showcaps=None, showbox=None, showfliers=None,
            boxprops=None, labels=None, flierprops=None,
            medianprops=None, meanprops=None,
            capprops=None, whiskerprops=None)
```

x：指定要绘制箱线图的数据；
notch：是否凹口的形式展现箱线图，默认非凹口；
sym：指定异常点的形状，默认为+号显示；
vert：是否需要将箱线图垂直摆放，默认垂直摆放；
whis：指定上下须与上下四分位的距离，默认为1.5倍的四分位差；
positions：指定箱线图的位置，默认为[0,1,2…]；
widths：指定箱线图的宽度，默认为0.5；
patch_artist：是否填充箱体的颜色；
meanline：是否用线的形式表示均值，默认用点来表示；
showmeans：是否显示均值，默认不显示；
showcaps：是否显示箱线图顶端和末端的两条线，默认显示；
showbox：是否显示箱线图的箱体，默认显示；
showfliers：是否显示异常值，默认显示；
boxprops：设置箱体的属性，如边框色，填充色等；
labels：为箱线图添加标签，类似于图例的作用；
filerprops：设置异常值的属性，如异常点的形状、大小、填充色等；
medianprops：设置中位数的属性，如线的类型、粗细等；
meanprops：设置均值的属性，如点的大小、颜色等；
capprops：设置箱线图顶端和末端线条的属性，如颜色、粗细等；
whiskerprops：设置须的属性，如颜色、粗细、线的类型等；

3.2.6 饼图

```
data = [0.2, 0.3, 0.4, 0.1]
plt.pie(data)
plt.show()
```

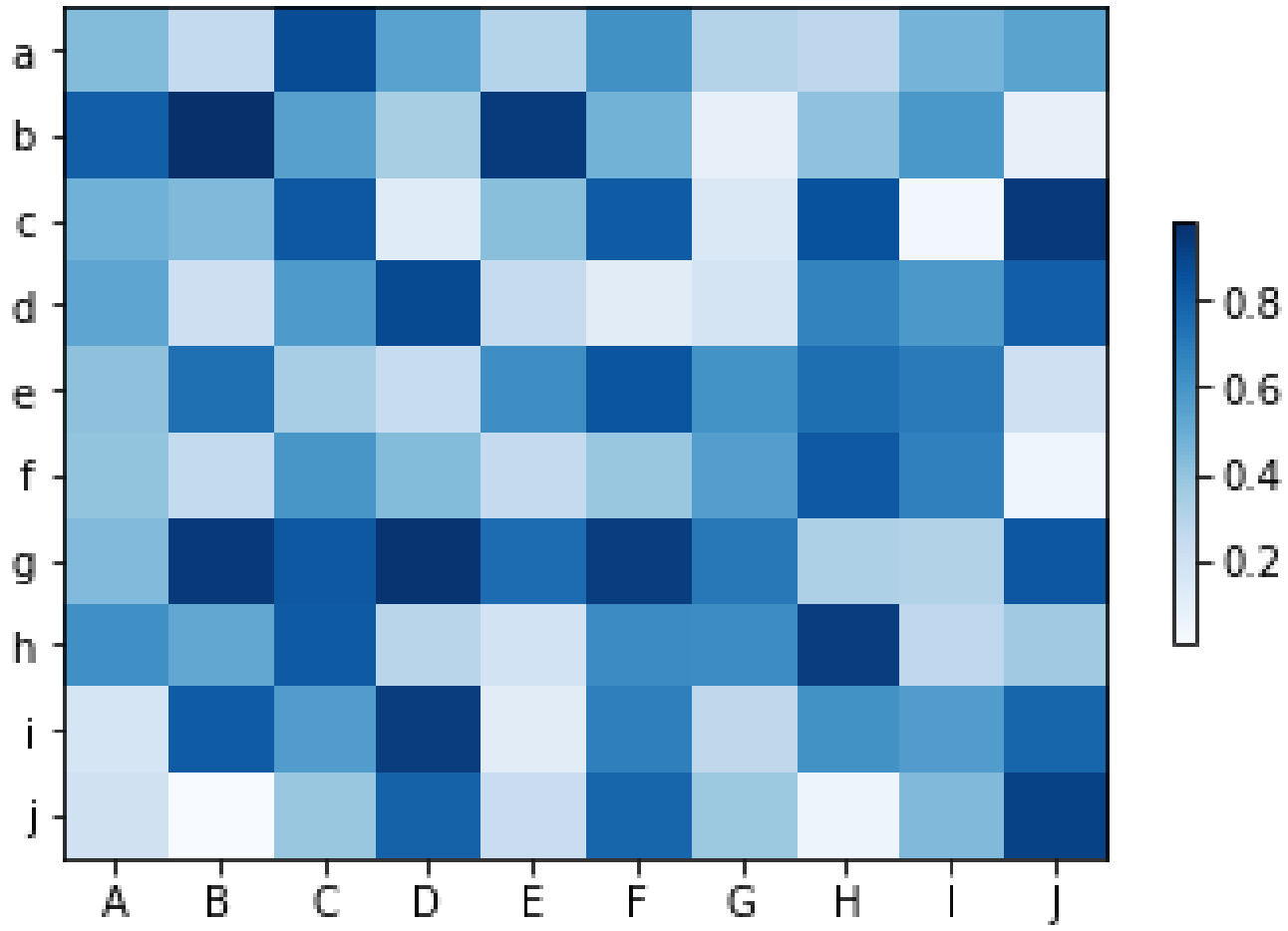


```
plt.pie(x, explode=None, labels=None, colors=None,
        autopct=None, pctdistance=0.6, shadow=False,
        labeldistance=1.1, startangle=None,
        radius=None, counterclock=True, wedgeprops=None,
        textprops=None, center=(0, 0), frame=False)
```

x：指定绘图的数据；
explode：指定饼图某些部分的突出显示，即呈现爆炸式；
labels：为饼图添加标签说明，类似于图例说明；
colors：指定饼图的填充色；
autopct：自动添加百分比显示，可以采用格式化的方法显示；
pctdistance：设置百分比标签与圆心的距离；
shadow：是否添加饼图的阴影效果；
labeldistance：设置各扇形标签（图例）与圆心的距离；
startangle：设置饼图的初始摆放角度；
radius：设置饼图的半径大小；
counterclock：是否让饼图按逆时针顺序呈现；
wedgeprops：设置饼图内外边界的属性，如边界线的粗细、颜色等；
textprops：设置饼图中文本的属性，如字体大小、颜色等；
center：指定饼图的中心点位置，默认为原点
frame：是否要显示饼图背后的图框，如果设置为True的话，需要同时控制图框x轴、y轴的范围和饼图的中心

3.2.7 绘制基因矩阵的热图

```
# 来源于百度经验
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
from matplotlib import axes


def draw_heatmap(data, xlabels, ylabels):  # 自定义函数，3 个参数
    cmap = cm.Blues
    figure = plt.figure(facecolor='w')
    ax = figure.add_subplot(1, 1, 1, position=[0.1, 0.15, 0.8, 0.8])
    ax.set_yticks(range(len(ylabels)))
    ax.set_yticklabels(ylabels)
    ax.set_xticks(range(len(xlabels)))
    ax.set_xticklabels(xlabels)
    vmax = data[0][0]
    vmin = data[0][0]
    for i in data:
        for j in i:
            if j > vmax:
                vmax = j
            if j < vmin:
                vmin = j
    map = ax.imshow(data, interpolation='nearest', cmap=cmap, aspect='auto',
                    vmin=vmin, vmax=vmax)  # interpolation 插值，cmap:colormap
    cb = plt.colorbar(mappable=map, cax=None, ax=None, shrink=0.5)  # 绘制颜色条
    plt.show()


a = np.random.rand(10, 10)
xlabels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
ylabels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
draw_heatmap(a, xlabels, ylabels)
```



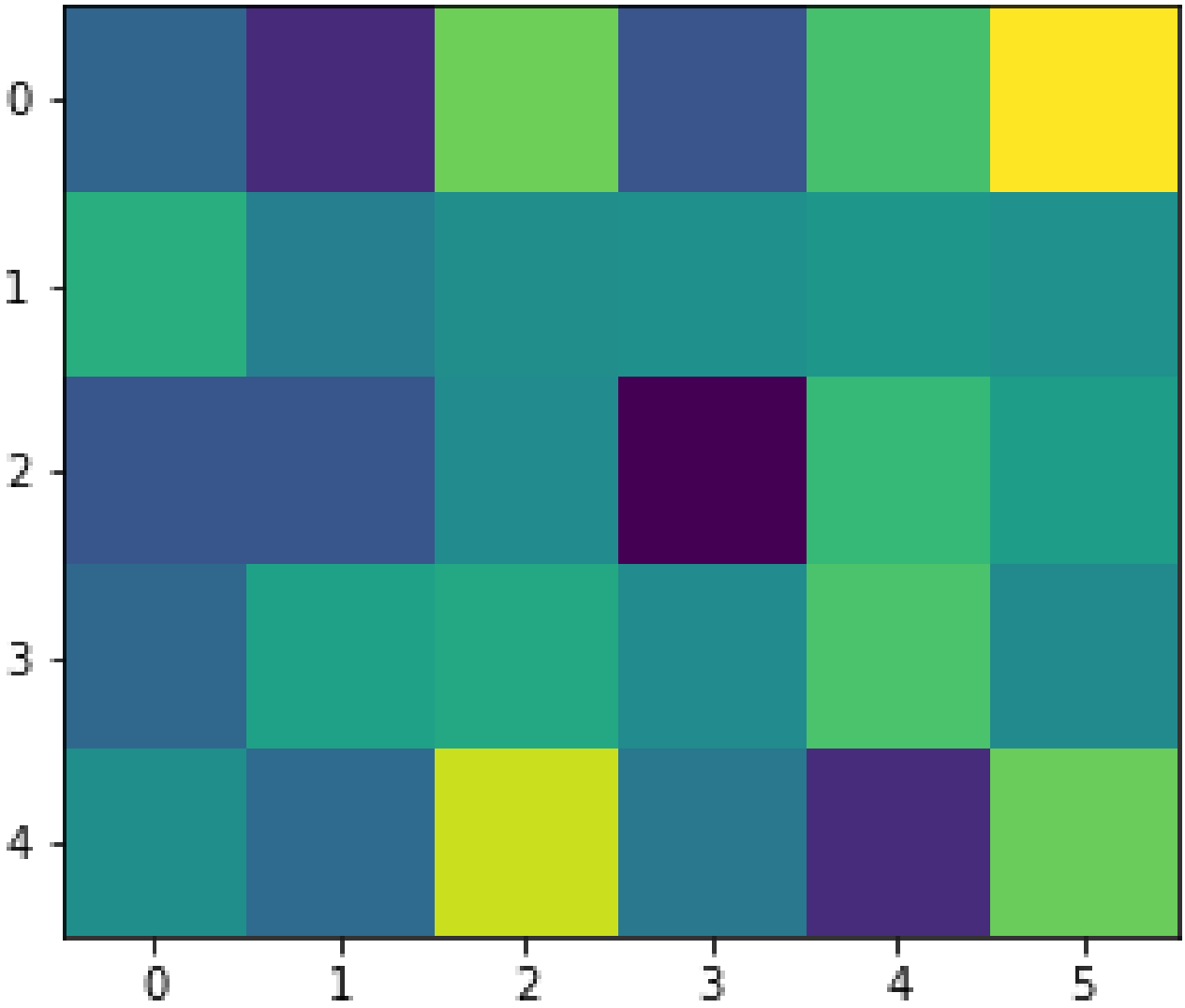

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5, 6))
df
```

```
<tr style="text-align: right;">
  <th></th>
  <th>0</th>
  <th>1</th>
  <th>2</th>
  <th>3</th>
  <th>4</th>
  <th>5</th>
</tr>
```

```
<tr>
  <th>0</th>
  <td>-0.596967</td>
```

```
  <td>0.080979</td>
  <td>-1.890122</td>
  <td>-0.629949</td>
  <td>-1.187113</td>
  <td>-0.002018</td>
</tr>
<tr>
  <th>1</th>
  <td>1.264042</td>
  <td>0.044216</td>
  <td>-1.669342</td>
  <td>0.805476</td>
  <td>-0.173634</td>
  <td>-0.051918</td>
</tr>
<tr>
  <th>2</th>
  <td>0.793684</td>
  <td>1.746997</td>
  <td>0.785712</td>
  <td>-0.157126</td>
  <td>0.072688</td>
  <td>0.310271</td>
</tr>
<tr>
  <th>3</th>
  <td>0.988716</td>
  <td>0.404472</td>
  <td>0.967111</td>
  <td>-0.612160</td>
  <td>-0.259917</td>
  <td>-0.067299</td>
</tr>
<tr>
  <th>4</th>
  <td>0.033911</td>
  <td>1.177149</td>
  <td>-0.914153</td>
  <td>-0.579912</td>
  <td>-0.100110</td>
  <td>0.747296</td>
</tr>
```

```
plt.imshow(df)

plt.show()
```



读入基因表达数据，绘制某基因的箱线图

```
%cd F: \mega\ehbio\python\python 课程包\merge
import os
os.getcwd()
```

```
F:\mega\ehbio\python\python课程包\merge
```

```
'F:\\mega\\ehbio\\python\\python课程包\\merge'
```

```
import pandas as pd
df = pd.read_table("ehbio.xls", sep="\t", index_col=0, header=0)
df.head(5)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>MCF10F</th>
  <th>MCF12A</th>
  <th>T47DKBluc</th>
  <th>UACC812</th>
  <th>UACC893</th>
  <th>ZR751</th>
  <th>ZR7530</th>
  <th>ZR75B</th>
</tr>
<tr>
  <th>EnsEMBL_Gene_ID</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>ENSG00000000003</th>
  <td>91.574061</td>
  <td>13.040870</td>
  <td>73.387310</td>
  <td>26.579280</td>
  <td>50.676579</td>
  <td>31.812999</td>
  <td>121.459918</td>
  <td>2.293898</td>
</tr>
<tr>
  <th>ENSG00000001167</th>
  <td>164.221408</td>
  <td>170.101403</td>
  <td>106.114416</td>
  <td>138.140311</td>
  <td>106.342032</td>
```

```
  <td>68.458022</td>
  <td>73.910964</td>
  <td>61.625519</td>
</tr>
<tr>
  <th>ENSG00000005471</th>
  <td>0.000000</td>
  <td>0.073680</td>
  <td>0.536226</td>
  <td>0.440649</td>
  <td>0.066213</td>
  <td>0.548865</td>
  <td>0.611036</td>
  <td>0.556531</td>
</tr>
<tr>
  <th>ENSG00000066629</th>
  <td>24.630773</td>
  <td>18.627459</td>
  <td>34.619613</td>
  <td>79.187879</td>
  <td>0.058835</td>
  <td>25.493800</td>
  <td>25.917323</td>
  <td>32.528112</td>
</tr>
<tr>
  <th>ENSG00000154258</th>
  <td>0.896612</td>
  <td>0.214261</td>
  <td>0.000000</td>
  <td>0.040907</td>
  <td>0.021120</td>
  <td>0.000000</td>
  <td>0.033571</td>
  <td>0.177808</td>
</tr>
```

```
df.shape
```

```
(36953, 8)
```

```
df.size
df.ndim
df.values
```

```
array([[  9.15740610e+01,   1.30408701e+01,   7.33873101e+01, ...,
          3.18129988e+01,   1.21459918e+02,   2.29389818e+00],
       [  1.64221408e+02,   1.70101403e+02,   1.06114416e+02, ...,
          6.84580224e+01,   7.39109636e+01,   6.16255191e+01],
       [  0.00000000e+00,   7.36800986e-02,   5.36226145e-01, ...,
          5.48864965e-01,   6.11036013e-01,   5.56531115e-01],
       ...,
       [  2.71701826e+00,   5.55925621e+00,   2.93123839e+00, ...,
          3.71765369e+00,   2.42540639e+00,   4.56656068e+00],
       [  1.74266818e+01,   3.72964028e+00,   3.49013629e+01, ...,
          5.04982802e+01,   3.75784692e+01,   4.79996494e+01],
       [  0.00000000e+00,   0.00000000e+00,   3.52988338e+00, ...,
          8.15259649e+00,   6.08178979e-01,   3.25252710e+00]])
```

```
control = []
for i in range(3):
    control.append(df.iloc[0, i])
control
```

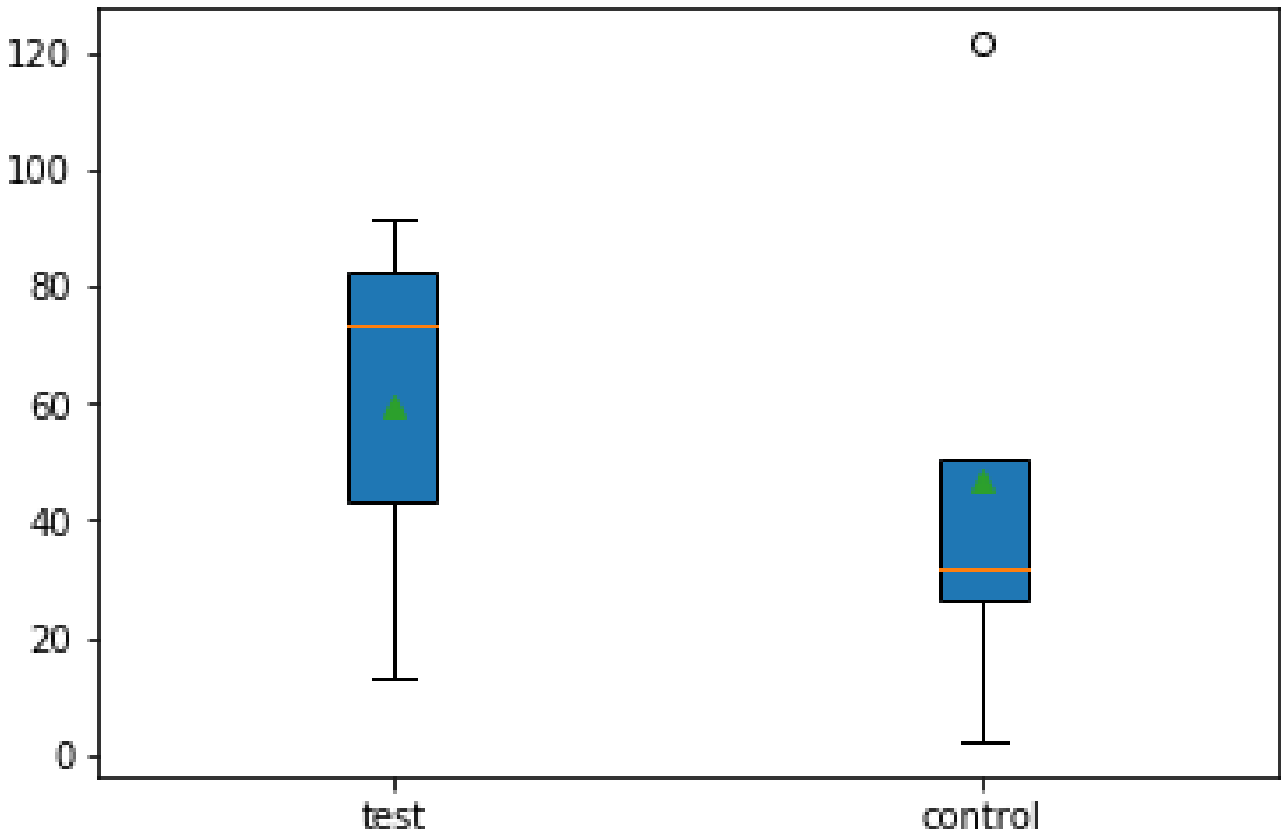
```
[91.574060981100004, 13.0408701074, 73.387310079800002]
```

```
test = []
for i in range(3, 8):
    test.append(df.iloc[0, i])
test
```

```
[26.579280359299997,
 50.676578905299998,
 31.812998762600003,
 121.4599177489,
 2.2938981847999997]
```

```
gene = [control, test]
```

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.boxplot(gene, patch_artist=True,   # 箱体添加颜色
           labels=['test', 'control'],   # 添加具体的标签名称
           showmeans=True, )
plt.show()
```



热图

```
df2 = df.head(10)   # 取部分数据
```

```
df2
```

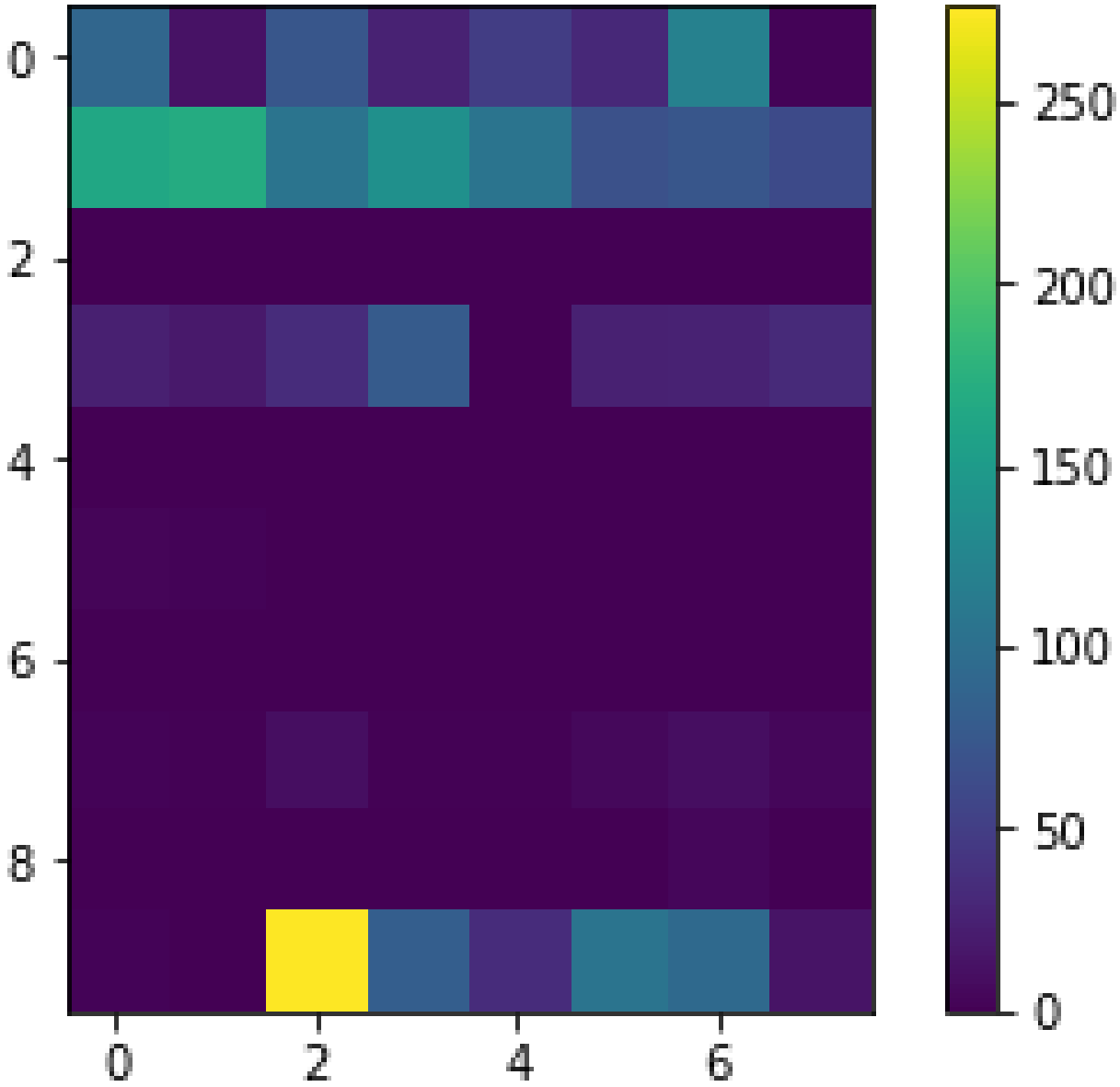
```
<tr style="text-align: right;">
  <th></th>
  <th>MCF10F</th>
  <th>MCF12A</th>
  <th>T47DKBluc</th>
  <th>UACC812</th>
  <th>UACC893</th>
  <th>ZR751</th>
  <th>ZR7530</th>
  <th>ZR75B</th>
</tr>
<tr>
  <th>EnsEMBL_Gene_ID</th>
  <th></th>
  <th></th>
```

```
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003</th>
  <td>91.574061</td>
  <td>13.040870</td>
  <td>73.387310</td>
  <td>26.579280</td>
  <td>50.676579</td>
  <td>31.812999</td>
  <td>121.459918</td>
  <td>2.293898</td>
</tr>
<tr>
  <th>ENSG00000001167</th>
  <td>164.221408</td>
  <td>170.101403</td>
  <td>106.114416</td>
  <td>138.140311</td>
  <td>106.342032</td>
  <td>68.458022</td>
  <td>73.910964</td>
  <td>61.625519</td>
</tr>
<tr>
  <th>ENSG00000005471</th>
  <td>0.000000</td>
  <td>0.073680</td>
  <td>0.536226</td>
  <td>0.440649</td>
  <td>0.066213</td>
  <td>0.548865</td>
  <td>0.611036</td>
  <td>0.556531</td>
</tr>
<tr>
  <th>ENSG00000066629</th>
  <td>24.630773</td>
  <td>18.627459</td>
  <td>34.619613</td>
```

```
  <td>79.187879</td>
  <td>0.058835</td>
  <td>25.493800</td>
  <td>25.917323</td>
  <td>32.528112</td>
</tr>
<tr>
  <th>ENSG00000154258</th>
  <td>0.896612</td>
  <td>0.214261</td>
  <td>0.000000</td>
  <td>0.040907</td>
  <td>0.021120</td>
  <td>0.000000</td>
  <td>0.033571</td>
  <td>0.177808</td>
</tr>
<tr>
  <th>ENSG00000154262</th>
  <td>3.431313</td>
  <td>2.481636</td>
  <td>0.000000</td>
  <td>0.383060</td>
  <td>0.000000</td>
  <td>0.000000</td>
  <td>0.000000</td>
  <td>0.105758</td>
</tr>
<tr>
  <th>ENSG00000154263</th>
  <td>0.859071</td>
  <td>0.936069</td>
  <td>0.589547</td>
  <td>0.259898</td>
  <td>0.083258</td>
  <td>0.476794</td>
  <td>0.526162</td>
  <td>0.221172</td>
</tr>
<tr>
  <th>ENSG00000154265</th>
  <td>2.453971</td>
  <td>1.480553</td>
  <td>9.884425</td>
  <td>2.076723</td>
  <td>1.118819</td>
  <td>5.864268</td>
```

```
  <td>10.180796</td>
  <td>4.383174</td>
</tr>
<tr>
  <th>ENSG00000154269</th>
  <td>0.216739</td>
  <td>0.000000</td>
  <td>0.854111</td>
  <td>0.970053</td>
  <td>0.738803</td>
  <td>0.521094</td>
  <td>4.648410</td>
  <td>0.000000</td>
</tr>
<tr>
  <th>ENSG00000154274</th>
  <td>2.980792</td>
  <td>0.217999</td>
  <td>276.123679</td>
  <td>82.820443</td>
  <td>34.664053</td>
  <td>106.350607</td>
  <td>94.854069</td>
  <td>14.976808</td>
</tr>
```

```
i = plt.imshow(df2)
plt.colorbar(i)
plt.show()
```



https://matplotlib.org/api/pyplot_api.html#□□□□□□□□□。

4 Python 实战

欢迎访问我们的视频课程 https://bioinfo.ke.qq.com。

4.1 Python 实战

4.1.1 ID 转换

不同数据库的名字互相转换，比如 NCBI 的 ENTEZ ID 转 Gene symbol，ENSEMBL 编号转 ENTREZ ID。

ID 转换一般需要 2 个文件，一个文件是要转换的 ID，另外一个文件是 ID 的对应关系。

以我们的测试文件 (**GRCh38.idmap**) 为例，第一列为 `ENSEMBLE ID`，第二列为 `Gene symbol`，第三列为 `Entrez ID`。

```
Gene ID Associated Gene Name    EntrezGene ID
ENSG00000252303 RNU6-280P
ENSG00000281771 Y_RNA
ENSG00000281876 RP11-399E6.4    101929901
ENSG00000281766 RYBP    23429
ENSG00000281518 FOXO6   100132074
ENSG00000281614 INPP5D  3635
ENSG00000280584 OBP2B   29989
ENSG00000281230 SERTAD4 56256
ENSG00000281917 SLC16A1 6566
```

待转换的文件 (**ensm.id**) 内容为，期望转换为 `Gene symbol`。

```
ENSG00000252303
ENSG00000281771
ENSG00000281256
ENSG00000283272
ENSG00000280864
ENSG00000280792
ENSG00000282878
ENSG00000283276
ENSG00000281822
ENSG00000281384
ENSG00000280505
```

```
ENSG00000281764
ENSG00000281316
ENSG00000280963
ENSG00000280775
ENSG00000281876
ENSG00000281766
ENSG00000281518
ENSG00000281614
ENSG00000280584
ENSG00000281230
ENSG00000281917
```

我们怎么做呢？直观看起来也比较简单，一个个的去比较、匹配、提取就可以了。

```
idD = {}
for line in open("data/GRCh38.idmap"):
    lineL = line.strip().split("\t")
    ensm_id = lineL[0]
    symbol  = lineL[1]
    idD[ensm_id] = symbol
#--------------------------------
for line in open("data/ensm.id"):
    ensm = line.strip()
    print(ensm,idD[ensm],sep=": ")
```

```
ENSG00000252303: RNU6-280P
ENSG00000281771: Y_RNA
ENSG00000281256: RP11-222G7.2
ENSG00000283272: Clostridiales-1
ENSG00000280864: RP11-654C22.2
ENSG00000280792: RP11-315F22.1
ENSG00000282878: RP11-399E6.1
ENSG00000283276: ABBA01000934.1
ENSG00000281822: RNU1-62P
ENSG00000281384: AC093802.1
ENSG00000280505: RP11-654C22.1
ENSG00000281764: RP11-399E6.2
ENSG00000281316: DPPA2P2
ENSG00000280963: SERTAD4-AS1
ENSG00000280775: RNA5SP136
ENSG00000281876: RP11-399E6.4
ENSG00000281766: RYBP
ENSG00000281518: FOXO6
ENSG00000281614: INPP5D
ENSG00000280584: OBP2B
```

```
ENSG00000281230: SERTAD4
ENSG00000281917: SLC16A1
```

```
# 首先读入 GRCh38.idmap 文件

# 定义 3 个变量，文件名，ensembl_id 所在列 (0-start, source_col), symbol 所在列 (1,target_col)
# 提前定义的好处是，修改起来会比较方便。
map_file = "data/GRCh38.idmap"
source_col = 0
target_col = 1
aDict = {}
for line in open(map_file):
    lineL = line.strip().split("\t")
    source = lineL[source_col]
    target = lineL[target_col]
    aDict[source] = target

# 读入 ensm.id 文件
# 边读边处理

id_file = "data/ensm.id"
for line in open(id_file):
    source_id = line.strip()
    map_id = aDict.get(source_id, source_id)
    print(map_id)

# 输出到文件

with open("data/symbol.id","w") as fh:
    id_file = "data/ensm.id"
    for line in open(id_file):
        source_id = line.strip()
        map_id = aDict.get(source_id, source_id)
        print(map_id, file=fh)
```

```
RNU6-280P
Y_RNA
RP11-222G7.2
Clostridiales-1
RP11-654C22.2
RP11-315F22.1
RP11-399E6.1
ABBA01000934.1
RNU1-62P
AC093802.1
RP11-654C22.1
```

```
RP11-399E6.2
DPPA2P2
SERTAD4-AS1
RNA5SP136
RP11-399E6.4
RYBP
FOXO6
INPP5D
OBP2B
SERTAD4
SLC16A1
```

4.1.2 每条染色体基因数目统计

GTF 文件存储基因的注释信息。

1. seqname - name of the chromosome or scaffold; chromosome names can be given with or without the 'chr' prefix. Important note: the seqname must be one used within Ensembl, i.e. a standard chromosome name or an Ensembl identifier such as a scaffold ID, without any additional content such as species or assembly. See the example GFF output below.
2. source - name of the program that generated this feature, or the data source (database or project name)
3. feature - feature type name, e.g. Gene, Variation, Similarity
4. start - Start position of the feature, with sequence numbering starting at 1.
5. end - End position of the feature, with sequence numbering starting at 1.
6. score - A floating point value.
7. strand - defined as + (forward) or - (reverse).
8. frame - One of '0', '1' or '2'. '0' indicates that the first base of the feature is the first base of a codon, '1' that the second base is the first base of a codon, and so on..
9. attribute - A semicolon-separated list of tag-value pairs, providing additional information about each feature.

```
chr1    HAVANA  gene    11869   14409   .   +   .   gene_id "ENSG00000223972.5"; gene_type "tra
chr1    HAVANA  gene    14404   29570   .   -   .   gene_id "ENSG00000227232.5"; gene_type "unp
chr1    ENSEMBL gene    17369   17436   .   -   .   gene_id "ENSG00000278267.1"; gene_type "miR
```

如果统计每条染色体基因数目，只需要读第一列和第三列就可以了。

```
chr_gene_cntD = {}
for line in open("data/gencode.gene.gtf"):
    lineL = line.strip().split("\t")
    chrName = lineL[0]
    regionType = lineL[2]
    if (regionType == "gene"):
```

```
        if chrName not in chr_gene_cntD:
            chr_gene_cntD[chrName] = 1
        else:
            chr_gene_cntD[chrName] += 1
#---------------------------------
for chrName, cnt in list(chr_gene_cntD.items()):
    print(chrName,"has",cnt,'genes.')
```

```
chr1 has 5397 genes.
chr2 has 4150 genes.
chr3 has 3163 genes.
chr4 has 2633 genes.
chr5 has 2993 genes.
chr6 has 3001 genes.
chr7 has 2980 genes.
chr8 has 2444 genes.
chr9 has 2350 genes.
chr10 has 2306 genes.
chr11 has 3381 genes.
chr12 has 3047 genes.
chr13 has 1383 genes.
chr14 has 2289 genes.
chr15 has 2247 genes.
chr16 has 2597 genes.
chr17 has 3111 genes.
chr18 has 1206 genes.
chr19 has 2997 genes.
chr20 has 1436 genes.
chr21 has 880 genes.
chr22 has 1385 genes.
chrX has 2476 genes.
chrY has 594 genes.
chrM has 37 genes.
```

```
gtf_file = "data/gencode.gene.gtf"

# 一般字典命名，会在行尾加一个大写的 D，作为类型代表
chr_gene_cntD = {}

for line in open(gtf_file):
    # 每行按 TAB 键分成 4 份
    lineL = line.split('\t',3)
    chr_name = lineL[0]
    feature  = lineL[2]
    if feature == "gene":
        # 如果 chr_name 已经出现过，则获取其值，然后加 1
```

```
        # 若没出现过，获取 0，加 1，表示第一次出现
        chr_gene_cntD[chr_name] = chr_gene_cntD.get(chr_name,0) + 1
#-------------------------------------
for chr_name, count in chr_gene_cntD.items():
    print(chr_name, count)
```

```
chr1 5397
chr2 4150
chr3 3163
chr4 2633
chr5 2993
chr6 3001
chr7 2980
chr8 2444
chr9 2350
chr10 2306
chr11 3381
chr12 3047
chr13 1383
chr14 2289
chr15 2247
chr16 2597
chr17 3111
chr18 1206
chr19 2997
chr20 1436
chr21 880
chr22 1385
chrX 2476
chrY 594
chrM 37
```

4.1.3 所有外显子总长度统计

bed12 格式 (前 3 列必须，其它可选)

1. chrom - The name of the chromosome (e.g. chr3, chrY, chr2_random) or scaffold (e.g. scaffold10671).
2. chromStart - The starting position of the feature in the chromosome or scaffold. The first base in a chromosome is numbered 0.
3. chromEnd - The ending position of the feature in the chromosome or scaffold. The chromEnd base is not included in the display of the feature.

4. name - Defines the name of the BED line. This label is displayed to the left of the BED line in the Genome Browser window when the track is open to full display mode or directly to the left of the item in pack mode.
5. score - A score between 0 and 1000. If the track line useScore attribute is set to 1 for this annotation data set, the score value will determine the level of gray in which this feature is displayed (higher numbers = darker gray). This table shows the Genome Browser's translation of BED score values into shades of gray:
6. thickStart - The starting position at which the feature is drawn thickly (for example, the start codon in gene displays). When there is no thick part, thickStart and thickEnd are usually set to the chromStart position.
7. thickEnd - The ending position at which the feature is drawn thickly (for example the stop codon in gene displays).
8. itemRgb - An RGB value of the form R,G,B (e.g. 255,0,0). If the track line itemRgb attribute is set to "On", this RBG value will determine the display color of the data contained in this BED line. NOTE: It is recommended that a simple color scheme (eight colors or less) be used with this attribute to avoid overwhelming the color resources of the Genome Browser and your Internet browser.
9. blockCount - The number of blocks (exons) in the BED line.
10. blockSizes - A comma-separated list of the block sizes. The number of items in this list should correspond to blockCount.
11. blockStarts - A comma-separated list of block starts. All of the blockStart positions should be calculated relative to chromStart. The number of items in this list should correspond to blockCount.

基因注释 bed12 格式

```
chr1    11868   14409   ENST00000456328.2   0   +   14409   14409   0   3   359,    109,
1189,   0,744,1352,
chr1    12009   13670   ENST00000450305.2   0   +   13670   13670   0   6   48,49,85,78,
154,218,    0,169,603,965,1211,1443,
chr1    17368   17436   ENST00000619216.1   0   -   17436   17436   0   1   68, 0,
chr1    29553   31097   ENST00000473358.1   0   +   31097   31097   0   3   486,    104,
122,    0,1010,1422,
chr1    30266   31109   ENST00000469289.1   0   +   31109   31109   0   2   401,134,    0,
709,
chr1    30365   30503   ENST00000607096.1   0   +   30503   30503   0   1   138,    0,
```

第十列是外显子的大小，所有我们只需要把他们加在一起就好了。

Note：实际计算时需要考虑不同转录本之间存在重叠，需要对 bed 文件预处理，只保留唯一的外显子位置，然后再加和。

```
exonSizeSum = 0
for line in open("data/gencode.gene.bed12"):
    lineL = line.split()
    exonSize = lineL[10]
    exonSizeL = exonSize.strip(',').split(',')
```

```
    for i in exonSizeL:
        exonSizeSum += int(i)
print(exonSizeSum)
```

```
291762517
```

```
bed12 = "data/gencode.gene.bed12"

total_exon_sum = 0

for line in open(bed12):
    # 从右侧分割为 3 个元素的列表
    # 关注的元素在第二位，索引为 1
    # 去掉末尾的逗号
    lineL = line.rsplit("\t",2)
    exonSize = lineL[1].strip(',')
    exonSizeL = [int(i) for i in exonSize.split(',')]
    exonSum = sum(exonSizeL)

    total_exon_sum += exonSum
print("Total exon size is", total_exon_sum, "nt.")

# 更清晰展示数字大小
print("Total exon size is", "{:,}".format(total_exon_sum), "nt.")
```

```
Total exon size is 291762517 nt.
Total exon size is 291,762,517 nt.
```

```
print("{a[a]}".format(a={'a':"ehbio"}))
```

```
ehbio
```

4.2 Python 小技巧

链似比较

```
x = 5
print("1 < x < 10 is", 1 < x < 10)
print("10 > x <= 9", 10 > x <= 9)
```

```
1 < x < 10 is True
```

```
10 > x <= 9 True
```

解释正则表达式

```
import re
re.compile("^[a-z]*$", re.DEBUG)
```

```
AT AT_BEGINNING
MAX_REPEAT 0 MAXREPEAT
  IN
    RANGE (97, 122)
AT AT_END
```

```
re.compile(r'^[a-z]*$', re.UNICODE|re.DEBUG)
```

```
re.compile("^[a-z][0-9]+$", re.DEBUG)
```

```
AT AT_BEGINNING
IN
  RANGE (97, 122)
MAX_REPEAT 1 MAXREPEAT
  IN
    RANGE (48, 57)
AT AT_END
```

```
re.compile(r'^[a-z][0-9]+$', re.UNICODE|re.DEBUG)
```

```
re.compile("^[a-z]([0-9]+)$", re.DEBUG)
```

```
AT AT_BEGINNING
IN
  RANGE (97, 122)
SUBPATTERN 1 0 0
  MAX_REPEAT 1 MAXREPEAT
    IN
```

```
      RANGE (48, 57)
AT AT_END
```

```
re.compile(r'^[a-z]([0-9]+)$', re.UNICODE|re.DEBUG)
```

enumerate (不再使用 len)

```
a = ['s','x','b','d']

# Preferred way
for index, item in enumerate(a):
    print(index,item)

print("\n")

# Old way
for i in range(len(a)):
    print(i,a[i])
```

```
0 s
1 x
2 b
3 d
```

```
0 s
1 x
2 b
3 d
```

列表解析、字典解析、元组解析

```
# 获得系列坐标点
a = ((i,j) for i in range(3) for j in range(2))
a
```

```
<generator object <genexpr> at 0x7fa8d04d7fc0>
```

```
for i in a:
    print(i)
```

```
(0, 0)
(0, 1)
(1, 0)
(1, 1)
(2, 0)
(2, 1)
```

```
str1 = "I love sheng xin bao dian"
print([i for i in str1.split() if i.endswith('n')])
```

```
['xin', 'dian']
```

```
a = {i:i*2 for i in range(5)}
a
```

```
{0: 0, 1: 2, 2: 4, 3: 6, 4: 8}
```

```
[(x, y) for x in range(4) if x % 2 == 1 for y in range(4)]
```

```
[(1, 0), (1, 1), (1, 2), (1, 3), (3, 0), (3, 1), (3, 2), (3, 3)]
```

列表索引，反序

```
a = [1,2,3,4,5]
a[::2]
```

```
[1, 3, 5]
```

```
a[::-1]
```

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

for..else; 若 `for` 循环中未执行 `break`, 则 `else` 会被执行

```
found = False
for i in range(2,5):
    if i == 1:
        found = True
        break
if not found:
    print("i was never 1")
```

```
i was never 1
```

```
for i in range(2,5):
    if i == 1:
        break
else:
    print("i was never 1")
```

```
i was never 1
```

原位替换

```
a = 5
b = 6
c = 7
a, b = b,a
print("a is",a)
print("b is",b)
```

```
a is 6
b is 5
```

```
a, (b,c) = c, (a,b)
print("a is",a)
print("b is",b)
print("c is",c)
```

```
a is 7
b is 6
c is 5
```

```
first, *middle_all, last = (1,2,3,4,5,6)
middle_all
```

```
[2, 3, 4, 5]
```

集合操作

```
a = set(["sheng", "xin", "bao","dian","best","tutotials"])
b = set(["hong", "ji", "yin","zu","best","tutotials"])
a | b  # union
```

```
{'bao', 'best', 'dian', 'hong', 'ji', 'sheng', 'tutotials', 'xin', 'yin', 'zu'}
```

```
a & b # intersection
```

```
{'best', 'tutotials'}
```

```
a ^ b # Symmetric Difference
```

```
{'bao', 'dian', 'hong', 'ji', 'sheng', 'xin', 'yin', 'zu'}
```

Negative round

```
print("round 整数:",str(round(1234.5678, -2)))

print("round 小数:",str(round(1234.5678, 2)))
```

```
round整数: 1200.0
round小数: 1234.57
```

多行字符串的嵌套

```
# \可以，但是第二行需要起头
system_command = "s-plot pheatmap -f matrix \
-t heatmap -a TRUE"
print(system_command)
```

```
s-plot pheatmap -f matrix -t heatmap -a TRUE
```

```
# 字符串中包含换行符
# 切第二行要起头，不然会有较多空格
system_command = """s-plot pheatmap -f matrix
-t heatmap -a TRUE"""
print(system_command)
print(system_command.replace('\n', ' '))
```

```
s-plot pheatmap -f matrix
-t heatmap -a TRUE
s-plot pheatmap -f matrix  -t heatmap -a TRUE
```

```
# 类元组的写法，既可以跨行，又可以自由格式
# 需要注意 2 点
   # 类元组，无逗号
   # 字符串连接时不会自动加空格，空格需要保存在字符串里面
system_command = ("s-plot pheatmap -f matrix "
                  "-t heatmap -a TRUE")
print(system_command)
```

```
s-plot pheatmap -f matrix -t heatmap -a TRUE
```

```
# 多一步 join；
system_command = ["s-plot pheatmap -f matrix",
                  "-t heatmap -a TRUE"]
print(' '.join(system_command))
```

```
s-plot pheatmap -f matrix -t heatmap -a TRUE
```

矩阵转置

```
a = [(1,2), (3,4), (5,6)]
b = zip(*a)
for i in b:
    print(i)
```

```
(1, 3, 5)
(2, 4, 6)
```

zip 转换两个列表为字典

```
keyL = [1,2,3]
valueL = ['a','b','v']
for i ,j in zip(keyL, valueL):
    print(i,j)
```

```
1 a
2 b
3 v
```

```
import pprint
pprint.pprint(dict(zip(keyL, valueL)))
```

```
{1: 'a', 2: 'b', 3: 'v'}
```

```
dict([(i,j) for i ,j in zip(keyL, valueL)])
```

```
{1: 'a', 2: 'b', 3: 'v'}
```

重复

```
'xyz' * 3
```

```
'xyzxyzxyz'
```

```
3 * 'xyz'
```

```
'xyzxyzxyz'
```

```
[1,2] * 3
```

```
[1, 2, 1, 2, 1, 2]
```

启动网络服务器，用于文件预览或传输

```
# run in commang line
# python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

sum 的另一用法，二维数组秒变 1 维

```
aList = [[1, 2, 3], [4, 5], [6], [7, 8, 9]]
sum(aList, [])
```

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

打开一个关于 Python 漫画的网站

```
#import antigravity
```

```
print('\n'.join([''.join([('Love'[(x-y) % len('Love')] if ((x*0.05)**2+(y*0.1)**2-1)**3-(x*0.05
```

```
            veLoveLov            veLoveLov
          eLoveLoveLoveLove    eLoveLoveLoveLove
        veLoveLoveLoveLoveLoveLoveLoveLoveLoveLov
       veLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveL
      veLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveLov
      eLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveLove
      LoveLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveL
      oveLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveLo
      veLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveLov
      eLoveLoveLoveLoveLoveLoveLoveLoveLoveLoveLove
       oveLoveLoveLoveLoveLoveLoveLoveLoveLoveLove
        eLoveLoveLoveLoveLoveLoveLoveLoveLoveLove
        LoveLoveLoveLoveLoveLoveLoveLoveLoveLoveL
          eLoveLoveLoveLoveLoveLoveLoveLoveLove
           oveLoveLoveLoveLoveLoveLoveLoveLove
            eLoveLoveLoveLoveLoveLoveLoveLove
              veLoveLoveLoveLoveLoveLoveLov
                oveLoveLoveLoveLoveLoveLo
                  LoveLoveLoveLoveLoveL
                     LoveLoveLoveLov
                        LoveLoveL
                           Lov
                            v
```

曼德勃罗集合

```
print('\n'.join([''.join(['*'if abs((lambda a:lambda z,c,n:a(a,z,c,n))(lambda s,z,c,n:z if n==0
```

```
*** ****************************************
  ****************************************
 ************************************* ***
 ****** * ************************** *
        *    * ************  * *
             *********
           ************
            ***********
                **
```

python 打印九九乘法表

```
print('\n'.join([' '.join(['%s*%s=%-2s' % (y,x,x*y) for y in range(1,x+1)]) for x in range(1,10
```

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12 4*4=16
1*5=5  2*5=10 3*5=15 4*5=20 5*5=25
1*6=6  2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7  2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8  2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9  2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

map, filter

```
# 过滤大于 4 的元素
a = [3,4,5]
[i for i in a if i<4]
```

```
[3]
```

```
list(filter(lambda x: x<4, a))
```

```
[5]
```

```
# 每个元素加 2
[i+2 for i in a]
```

```
[5, 6, 7]
```

```
map(lambda x: x+2, a)
```

```
<map at 0x7fa8d0502358>
```

```
list(map(lambda x: x+2, a))
```

```
[5, 6, 7]
```

算 2 的 1000 次方的各位数之和

```
sum(map(int, str(2**1000)))
```

```
1366
```

5 Pandas 学习教程

陈同 chentong_biology@163.com

欢迎访问我们的视频课程 https://bioinfo.ke.qq.com。

5.1 What is pandas

Pandas 是 python 中用于处理矩阵样数据的功能强大的包，提供了 R 中的 `dataframe` 和 `vector` 的操作，使得我们在使用 python 时，也可以方便、简单、快捷、高效地进行矩阵数据处理。

具体介绍详见http://pandas.pydata.org/。

- A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast **HDF5** format;
- Intelligent **data alignment** and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate **messy data into an orderly form**;
- Flexible **reshaping** and **pivoting** of data sets;
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets;
- Columns can be inserted and deleted from data structures for size mutability;
- Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets;
- High performance **merging** and **joining** of data sets;
- Hierarchical axis indexing provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
- Time series-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly optimized for performance, with critical code paths written in Cython or C.
- Python with pandas is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

```
%matplotlib inline

#import plotly
#plotly.offline.init_notebook_mode()

import matplotlib
matplotlib.style.use('ggplot')
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
import os
from glob import glob
```

5.2 Pandas 读取文件

5.2.1 获取目标文件

```
dir_1 = "py_data/"
glob(dir_1+'*')
```

```
['py_data/ENCFF060LPA.tsv',
 'py_data/ENCFF262OBL.tsv',
 'py_data/ENCFF289HGQ.tsv',
 'py_data/ENCFF673KYR.tsv',
 'py_data/gencode.v24.ENS2SYN',
 'py_data/meta.tsv',
 'py_data/gencode.gene.gtf',
 'py_data/gencode.gene.bed12',
 'py_data/ensm.id',
 'py_data/GRCh38.idmap']
```

5.2.2 查看目标文件内容和格式

Ipython 中可以通过在 Linux 命令前加！调用系统命令，更多使用见 http://ipython.org/ipython-doc/3/interactive/reference.html#system-shell-access.

```
!head -n 4 py_data/gencode.v24.ENS2SYN
```

```
gene_id gene_symbol
ENSG00000001460.17  STPG1
ENSG00000001461.16  NIPAL3
ENSG00000000938.12  FGR
```

```
!head -n 4 py_data/ENCFF060LPA.tsv
```

```
gene_id transcript_id(s)    length  effective_length    expected_count  TPM FPKM
```

```
ENSG00000000003.14   ENST00000373020.8,ENST00000494424.1,ENST00000496771.5,ENST00000612152.4,ENS
ENSG00000000005.5    ENST00000373031.4,ENST00000485971.1 940.50   720.47   0.00     0.00     0.00
ENSG00000000419.12   ENST00000371582.8,ENST00000371584.8,ENST00000371588.9,ENST00000413082.1,ENS
```

5.2.3 读取两列文件

```
ens2syn_file = dir_1+"/gencode.v24.ENS2SYN"
```

```
# pandas 中的计数都是从 0 开始的
# header=0: 指定第一行包含列的名字
# index_col=0: 指定第一列为行的名字
ens2syn = pd.read_table(ens2syn_file, header=0, index_col=0)
```

```
ens2syn.head()
```

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
<tr>
  <th>ENSG00000001461.16</th>
  <td>NIPAL3</td>
</tr>
<tr>
  <th>ENSG00000000938.12</th>
  <td>FGR</td>
</tr>
<tr>
  <th>ENSG00000004455.16</th>
  <td>AK2</td>
</tr>
<tr>
  <th>ENSG00000000460.16</th>
```

```
  <td>C1orf112</td>
</tr>
```

5.2.4 数据表的索引

* 数值索引和布尔值索引是按行选取
* 字符串索引是按列选取
* 行和列是等效的，应用于行的选取函数也可应用于列，反之亦然

```
ens2syn[:3]
```

5.2.4.1 按行选取数据

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
<tr>
  <th>ENSG00000001461.16</th>
  <td>NIPAL3</td>
</tr>
<tr>
  <th>ENSG00000000938.12</th>
  <td>FGR</td>
</tr>
```

```
ens2syn[ens2syn.index=="ENSG00000001460.17"]
```

5.2.4.2 取出索引中包含特定值的行

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
```

```
ens2syn[ens2syn['gene_symbol'].isin(['STPG1','FGR'])]
```

5.2.4.3 取出某列包含特定值列表的行

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
<tr>
  <th>ENSG00000000938.12</th>
  <td>FGR</td>
</tr>
```

```
# head: 只展示部分数据
ens2syn[ens2syn.index.str.contains(r'ENSG000000014')].head()
```

5.2.4.4 使用正则表达式选取符合要求的行

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
<tr>
  <th>ENSG00000001461.16</th>
  <td>NIPAL3</td>
</tr>
<tr>
  <th>ENSG00000001497.16</th>
  <td>LAS1L</td>
</tr>
```

5.2.5 读取多列文件

gzip, bzip 压缩的文件也可以直接读取，但是需要保证文件后缀的正确。read_table 默认参数可以自动检测文件的格式，根据文件的后缀 '.gz', '.bz2', '.zip', or 'xz' 分别使用 gzip, bz2, zip or xz 读取。

```
tsvL = glob(dir_1+'ENC*.tsv')
tsvL
```

```
['py_data/ENCFF060LPA.tsv',
 'py_data/ENCFF262OBL.tsv',
 'py_data/ENCFF289HGQ.tsv',
 'py_data/ENCFF673KYR.tsv']
```

```
index = 0
tsvFile = tsvL[index]
expr = pd.read_table(tsvFile, header=0, index_col=0)
expr.head(3)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>transcript_id(s)</th>
  <th>length</th>
  <th>effective_length</th>
  <th>expected_count</th>
  <th>TPM</th>
  <th>FPKM</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>ENSG00000000003.14</th>
  <td>ENST00000373020.8,ENST00000494424.1,ENST000004...</td>
  <td>2240.53</td>
  <td>2020.49</td>
  <td>5126.0</td>
  <td>6.64</td>
  <td>18.24</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>ENST00000373031.4,ENST00000485971.1</td>
  <td>940.50</td>
  <td>720.47</td>
  <td>0.0</td>
  <td>0.00</td>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>ENST00000371582.8,ENST00000371584.8,ENST000003...</td>
```

```
  <td>1072.03</td>
  <td>851.99</td>
  <td>3222.0</td>
  <td>9.91</td>
  <td>27.19</td>
</tr>
```

5.2.6 选取多列数据

列的输出顺序与给定的列名字的顺序一致

```
expr[['FPKM','TPM']].head(3)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>FPKM</th>
  <th>TPM</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000000003.14</th>
  <td>18.24</td>
  <td>6.64</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>27.19</td>
  <td>9.91</td>
</tr>
```

5.2.7 重命名列名字

从 Dataframe 中只选取一列时，数据框会被转换成 **Series**，因此需要使用 `pd.loc[:,[column_name]]`(虽然内部的方括号内只有一个值，但写法是必须的) 索引。

```
tsvFile
os.path.split(tsvFile)[-1][:-4]
```

```
'ENCFF060LPA'
```

```
# 因为要把多个文件的同一类型表达值合并到一个文件，我们使用文件名作为列的名字
name = os.path.split(tsvFile)[-1][:-4]
print(name)
expr_tpm = expr.loc[:,['TPM']] # 取出所有的行和名字为 TPM 的列
#expr_tpm.head()
# 给列重命名
expr_tpm.columns=[name]
expr_tpm[:3]
```

```
ENCFF060LPA
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
</tr>
```

5.2.8 合并矩阵

```
# 为了读取多个文件，定义一个函数简化操作
def readExpr_1(tsvFileL, typeL=['TPM','FPKM']):
    '''
    tsvFileL: lists of files waiting for reading
    resultD: a dictionary to save data matrix
            {'TPM':[mat1, mat2,...]
             'FPKM':[mat1, mat2, ...]}
    typeL; list of names for columns to be extracted
    '''
    resultD = {}
    for _type in typeL: resultD[_type] = []

    for tsvFile in tsvFileL:
        expr = pd.read_table(tsvFile, header=0, index_col=0)
        name = os.path.split(tsvFile)[-1][:-4]  #this option is very arbitary
        for _type in typeL:
            # add _ to type to avoid override Python inner function `type`
            expr_type = expr.loc[:,[_type]]
            expr_type.columns = [name]
            resultD[_type].append(expr_type)
    return resultD
#-----------------------------------------------------
```

```
exprD = readExpr_1(tsvL)
TPM_mat = exprD['TPM']
FPKM_mat = exprD['FPKM']
```

5.2.8.1 定义函数简化文件读取

5.2.8.2 使用 **pd.merge** 合并矩阵示例　先从刚才读取的矩阵中选出 2 个测试下 pandas 中的矩阵合并方法和效果

```
# 选取第一个矩阵
_idL = ['ENSG00000000003.14', 'ENSG00000000005.5','ENSG00000000419.12',
        'ENSG00000000457.13']
mat1 = TPM_mat[0]
mat1 = mat1[mat1.index.isin(_idL)]
mat1
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
</tr>
```

```
# 选取第二个矩阵
_idL = ['ENSG00000001561.6','ENSG00000000003.14', 'ENSG00000000419.12','ENSG00000001036.13']
mat2 = TPM_mat[1]
mat2 = mat2[mat2.index.isin(_idL)]
mat2
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
  <td>17.13</td>
```

```
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>18.86</td>
</tr>
<tr>
  <th>ENSG00000001036.13</th>
  <td>10.34</td>
</tr>
<tr>
  <th>ENSG00000001561.6</th>
  <td>2.47</td>
</tr>
```

基于索引 (index) 的合并 * outer: 合并所有的索引，缺失值填充 NA * inner：保留共有的索引 * left：使用第一个矩阵的索引 * right：使用第二个矩阵的索引

```
pd.merge(mat1, mat2, left_index=True, right_index=True, how="outer")
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
```

```
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000001036.13</th>
  <td>NaN</td>
  <td>10.34</td>
</tr>
<tr>
  <th>ENSG00000001561.6</th>
  <td>NaN</td>
  <td>2.47</td>
</tr>
```

```
pd.merge(mat1, mat2, left_index=True, right_index=True, how="inner")
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
</tr>
```

```
pd.merge(mat1, mat2, left_index=True, right_index=True, how="left")
```

```
<tr style="text-align: right;">
```

```
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>NaN</td>
</tr>
```

5.2.8.3 使用 pd.concat 合并矩阵示例

对于较多的数据表合并操作时，concat 比 merge 要简单快速很多。

```
pd.concat([mat1, mat2], axis=1)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
```

```
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000001036.13</th>
  <td>NaN</td>
  <td>10.34</td>
</tr>
<tr>
  <th>ENSG00000001561.6</th>
  <td>NaN</td>
  <td>2.47</td>
</tr>
```

```
pd.concat([mat1, mat2], axis=1, join="inner")
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
```

```
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
</tr>
```

```
mat3 = mat1.join(mat2, how="outer")
mat3
```

5.2.8.4 使用 pd.join 合并矩阵示例

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
```

```
  <td>0.86</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENSG00000001036.13</th>
  <td>NaN</td>
  <td>10.34</td>
</tr>
<tr>
  <th>ENSG00000001561.6</th>
  <td>NaN</td>
  <td>2.47</td>
</tr>
```

替换 NA 值为 0

```
mat3 = mat3.fillna(0)
mat3
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000005.5</th>
  <td>0.00</td>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
```

```
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000001036.13</th>
  <td>0.00</td>
  <td>10.34</td>
</tr>
<tr>
  <th>ENSG00000001561.6</th>
  <td>0.00</td>
  <td>2.47</td>
</tr>
```

去除所有值都为 0 的行

```
#Both works well here
#mat3[(mat3>0).any(axis=1)]
mat3.loc[(mat3>0).any(axis=1)]
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
</tr>
```

```
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>0.00</td>
</tr>
<tr>
  <th>ENSG00000001036.13</th>
  <td>0.00</td>
  <td>10.34</td>
</tr>
<tr>
  <th>ENSG00000001561.6</th>
  <td>0.00</td>
  <td>2.47</td>
</tr>
```

5.2.8.5 测试三种方法使用的内存和速度比较 速度：concat>join>>merge

内存：相当

```
# 不错的 reduce 教程
from functools import reduce
%timeit test_merge = reduce(lambda left,right: pd.merge(left,right,left_index=True,right_index=
```

```
3.04 ms ± 52.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%timeit test_merge = pd.concat(TPM_mat, axis=1)
```

```
1.29 ms ± 30.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%timeit TPM_mat[0].join(TPM_mat[1:], how="outer")
```

```
1.31 ms ± 11.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
# 首先安装 memory_profiler, 加载插件
%load_ext memory_profiler
```

```
%memit test_merge = reduce(lambda left,right: pd.merge(left,right,left_index=True,right_index=T
```

```
peak memory: 101.27 MiB, increment: 0.02 MiB
```

```
%memit test_merge = pd.concat(TPM_mat, axis=1)
```

```
peak memory: 101.28 MiB, increment: 0.00 MiB
```

```
%memit TPM_mat[0].join(TPM_mat[1:], how="outer")
```

```
peak memory: 101.28 MiB, increment: 0.00 MiB
```

```
# 读取多个文件，并且合并矩阵，定义一个函数简化操作
def concatExpr(tsvFileL, typeL=['TPM','FPKM']):
    '''
    tsvFileL: lists of files waiting for reading
    resultD: a dictionary to save data matrix
            {'TPM':[mat1, mat2,...]
             'FPKM':[mat1, mat2, ...]}
    typeL; list of names for columns to be extracted
    '''
    resultD = {}
    for _type in typeL: resultD[_type] = []

    for tsvFile in tsvFileL:
        expr = pd.read_table(tsvFile, header=0, index_col=0)
        name = os.path.split(tsvFile)[-1][:-4]  #this options is very arbitary
        for _type in typeL: # add _ to type to avoid override Python inner function `type`
            expr_type = expr.loc[:,[_type]]
            expr_type.columns = [name]
            resultD[_type].append(expr_type)
    #-----------------------------------------
    mergeD = {}
    for _type in typeL:
        mergeM = pd.concat(resultD[_type], axis=1)
        mergeM = mergeM.fillna(0) # Substitute all NA with 0
        mergeM = mergeM.loc[(mergeM>0).any(axis=1)] # Delete aoo zero rows.
        mergeD[_type] = mergeM
    return mergeD
#-----------------------------------------------------
```

```
typeL = ['TPM','FPKM']
exprD = concatExpr(tsvL, typeL)
```

```
TPM_mat = exprD['TPM']
FPKM_mat = exprD['FPKM']
```

```
TPM_mat.head()
```

5.2.8.6 重写函数完成文件的读写和矩阵的合并

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
  <th>ENCFF289HGQ</th>
  <th>ENCFF673KYR</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
  <td>1.03</td>
  <td>2.42</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
  <td>1.45</td>
  <td>1.80</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>2.48</td>
  <td>0.24</td>
```

```
  <td>0.38</td>
</tr>
<tr>
  <th>ENSG00000000460.16</th>
  <td>1.51</td>
  <td>5.36</td>
  <td>0.26</td>
  <td>0.16</td>
</tr>
<tr>
  <th>ENSG00000000938.12</th>
  <td>0.01</td>
  <td>0.05</td>
  <td>0.00</td>
  <td>0.00</td>
</tr>
```

5.2.9 矩阵数据提取

只保留表达矩阵中存储的基因的 `ID` 和 `Symbol` 对照表

```
# 回顾下数据格式
ens2syn.head(3)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
<tr>
  <th>ENSG00000001461.16</th>
  <td>NIPAL3</td>
</tr>
<tr>
  <th>ENSG00000000938.12</th>
```

```
  <td>FGR</td>
</tr>
```

```
ens2syn.shape
```

```
(48, 1)
```

```
ens2syn = ens2syn[ens2syn.index.isin(TPM_mat.index)]
```

```
ens2syn.shape
```

```
(48, 1)
```

```
ens2syn.head(3)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>

<tr>
  <th>ENSG00000001460.17</th>
  <td>STPG1</td>
</tr>
<tr>
  <th>ENSG00000001461.16</th>
  <td>NIPAL3</td>
</tr>
<tr>
  <th>ENSG00000000938.12</th>
  <td>FGR</td>
</tr>
```

5.2.10 读取 META data 文件

```
meta = "data/meta.tsv"
metaM = pd.read_table(meta, header=0, index_col=0)
# 重名了列的名字
oriColnames = metaM.columns.values
nameD = dict([(i,i.replace(' ','_')) for i in oriColnames])
metaM.rename(columns=nameD, inplace=True)
metaM.head(3)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>Biosample_term_id</th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
  <th>Biosample_organism</th>
  <th>Biosample_treatments</th>
  <th>Biosample_subcellular_fraction_term_name</th>
  <th>Biosample_phase</th>
  <th>Biosample_synchronization_stage</th>
  <th>Biosample_Age</th>
</tr>
<tr>
  <th>File accession</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>ENCFF673KYR</th>
  <td>CL:0000650</td>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
```

```
  <td>unknown, female</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENCFF262OBL</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>52 year</td>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>23 year</td>
</tr>
```

```
sampleL = TPM_mat.columns.values
metaM = metaM[metaM.index.isin(sampleL)]
# 同时索引行和列
metaM.iloc[:4,:5]
```

5.2.10.1 只保留前面提到的 4 个样品的数据

```
<tr style="text-align: right;">
  <th></th>
  <th>Biosample_term_id</th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
</tr>
<tr>
  <th>File accession</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENCFF673KYR</th>
  <td>CL:0000650</td>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
</tr>
<tr>
  <th>ENCFF262OBL</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
  <td>CL:0002558</td>
  <td>fibroblast of villous mesenchyme</td>
  <td>primary cell</td>
```

```
  <td>newborn</td>
  <td>male, female</td>
</tr>
```

```
# 假如只提取 `Biosample`开头的列
#meta_colL = ['Biosample term id', 'Biosample term name']

# Extract columns matching specific patterns
# Both works well, filter is more simple
#metaM.loc[:,metaM.columns.str.contains(r'^Biosample')]
metaM = metaM.filter(regex=("^Biosample"))
metaM
```

5.2.10.2 提取目标列信息

```
<tr style="text-align: right;">
  <th></th>
  <th>Biosample_term_id</th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
  <th>Biosample_organism</th>
  <th>Biosample_treatments</th>
  <th>Biosample_subcellular_fraction_term_name</th>
  <th>Biosample_phase</th>
  <th>Biosample_synchronization_stage</th>
  <th>Biosample_Age</th>
</tr>
<tr>
  <th>File accession</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
```

```
</tr>


<tr>
  <th>ENCFF673KYR</th>
  <td>CL:0000650</td>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENCFF262OBL</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>52 year</td>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>23 year</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
```

```
  <td>CL:0002558</td>
  <td>fibroblast of villous mesenchyme</td>
  <td>primary cell</td>
  <td>newborn</td>
  <td>male, female</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
</tr>
```

```
metaM.fillna('')
```

```
<tr style="text-align: right;">
  <th></th>
  <th>Biosample_term_id</th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
  <th>Biosample_organism</th>
  <th>Biosample_treatments</th>
  <th>Biosample_subcellular_fraction_term_name</th>
  <th>Biosample_phase</th>
  <th>Biosample_synchronization_stage</th>
  <th>Biosample_Age</th>
</tr>
<tr>
  <th>File accession</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>
```

```
<tr>
```

```
  <th>ENCFF673KYR</th>
  <td>CL:0000650</td>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
  <td>Homo sapiens</td>
  <td></td>
  <td></td>
  <td></td>
  <td></td>
  <td></td>
</tr>
<tr>
  <th>ENCFF262OBL</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td></td>
  <td></td>
  <td></td>
  <td></td>
  <td>52 year</td>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td></td>
  <td></td>
  <td></td>
  <td></td>
  <td>23 year</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
  <td>CL:0002558</td>
  <td>fibroblast of villous mesenchyme</td>
  <td>primary cell</td>
  <td>newborn</td>
```

```
  <td>male, female</td>
  <td>Homo sapiens</td>
  <td></td>
  <td></td>
  <td></td>
  <td></td>
  <td></td>
</tr>
```

5.3 Pandas 写入文件

5.3.1 写入文本文件

```
metaM.to_csv("pandas_data/meta2.tsv", sep="\t")
```

```
ens2syn.to_csv("pandas_data/gencode.v24.ENS2SYN", sep="\t")
```

```
TPM_mat.to_csv("pandas_data/TPM", sep='\t', float_format="%.2f")
```

5.4 PANDAS 矩阵的小应用

利用上面的矩阵操作，选取这两个基因相关的信息并绘制表达谱

```
targetL = ['KRIT1','AK2']
```

Gene_symbol 转换为 Gene_id

```
ensID = ens2syn[ens2syn["gene_symbol"].isin(targetL)]
ensID
```

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_symbol</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000004455.16</th>
  <td>AK2</td>
</tr>
<tr>
  <th>ENSG00000001631.14</th>
  <td>KRIT1</td>
</tr>
```

提取目标基因的表达

```
targetExpr = TPM_mat[TPM_mat.index.isin(ensID.index)]
targetExpr
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
  <th>ENCFF289HGQ</th>
  <th>ENCFF673KYR</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>
```

```
<tr>
  <th>ENSG00000001631.14</th>
  <td>6.21</td>
  <td>13.36</td>
  <td>1.15</td>
  <td>1.37</td>
</tr>
<tr>
  <th>ENSG00000004455.16</th>
  <td>15.57</td>
  <td>37.62</td>
  <td>2.31</td>
  <td>8.95</td>
</tr>
```

重命名矩阵的索引

```
ensID_dict = ensID.to_dict()
ensID_dict
```

```
{'gene_symbol': {'ENSG00000001631.14': 'KRIT1', 'ENSG00000004455.16': 'AK2'}}
```

```
targetExpr = targetExpr.rename(index=ensID_dict['gene_symbol'])
targetExpr
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
  <th>ENCFF289HGQ</th>
  <th>ENCFF673KYR</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>

<tr>
  <th>KRIT1</th>
  <td>6.21</td>
  <td>13.36</td>
  <td>1.15</td>
  <td>1.37</td>
</tr>
<tr>
  <th>AK2</th>
  <td>15.57</td>
  <td>37.62</td>
  <td>2.31</td>
  <td>8.95</td>
</tr>
```

转置矩阵以增加 META 信息

```
targetExpr_t = targetExpr.T
targetExpr_t
```

<table>
<tr style="text-align: right;">
  <th>gene_id</th>
  <th>KRIT1</th>
  <th>AK2</th>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>6.21</td>
  <td>15.57</td>
</tr>
<tr>
  <th>ENCFF262OBL</th>
  <td>13.36</td>
  <td>37.62</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
  <td>1.15</td>
  <td>2.31</td>
</tr>
<tr>
  <th>ENCFF673KYR</th>
  <td>1.37</td>
  <td>8.95</td>
</tr>
</table>

从 meta 矩阵中提取 4 列信息

```
metaM.head(3)
```

<table>
<tr style="text-align: right;">
  <th></th>
  <th>Biosample_term_id</th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
  <th>Biosample_organism</th>
  <th>Biosample_treatments</th>
</tr>

```
  <th>Biosample_subcellular_fraction_term_name</th>
  <th>Biosample_phase</th>
  <th>Biosample_synchronization_stage</th>
  <th>Biosample_Age</th>
</tr>
<tr>
  <th>File accession</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENCFF673KYR</th>
  <td>CL:0000650</td>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
</tr>
<tr>
  <th>ENCFF262OBL</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
```

```
  <td>52 year</td>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>CL:1001568</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
  <td>Homo sapiens</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>NaN</td>
  <td>23 year</td>
</tr>
```

```
meta_type = ["Biosample_term_name","Biosample_type", "Biosample_life_stage",
             "Biosample_sex"]
```

```
meta = metaM[meta_type]
meta
```

```
<tr style="text-align: right;">
  <th></th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
</tr>
<tr>
  <th>File accession</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENCFF673KYR</th>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
</tr>
```

```
<tr>
  <th>ENCFF262OBL</th>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
  <th>ENCFF060LPA</th>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
  <td>fibroblast of villous mesenchyme</td>
  <td>primary cell</td>
  <td>newborn</td>
  <td>male, female</td>
</tr>
```

```
target_expr_meta = targetExpr_t.join(meta, how="left")
target_expr_meta
```

```
<tr style="text-align: right;">
  <th></th>
  <th>KRIT1</th>
  <th>AK2</th>
  <th>Biosample_term_name</th>
  <th>Biosample_type</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
</tr>
```

```
<tr>
  <th>ENCFF060LPA</th>
  <td>6.21</td>
  <td>15.57</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
```

```
  <th>ENCFF262OBL</th>
  <td>13.36</td>
  <td>37.62</td>
  <td>pulmonary artery endothelial cell</td>
  <td>primary cell</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
  <td>1.15</td>
  <td>2.31</td>
  <td>fibroblast of villous mesenchyme</td>
  <td>primary cell</td>
  <td>newborn</td>
  <td>male, female</td>
</tr>
<tr>
  <th>ENCFF673KYR</th>
  <td>1.37</td>
  <td>8.95</td>
  <td>mesangial cell</td>
  <td>primary cell</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
</tr>
```

```
target_expr_meta.drop(["Biosample_term_name", "Biosample_type"], axis=1)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>KRIT1</th>
  <th>AK2</th>
  <th>Biosample_life_stage</th>
  <th>Biosample_sex</th>
</tr>
```

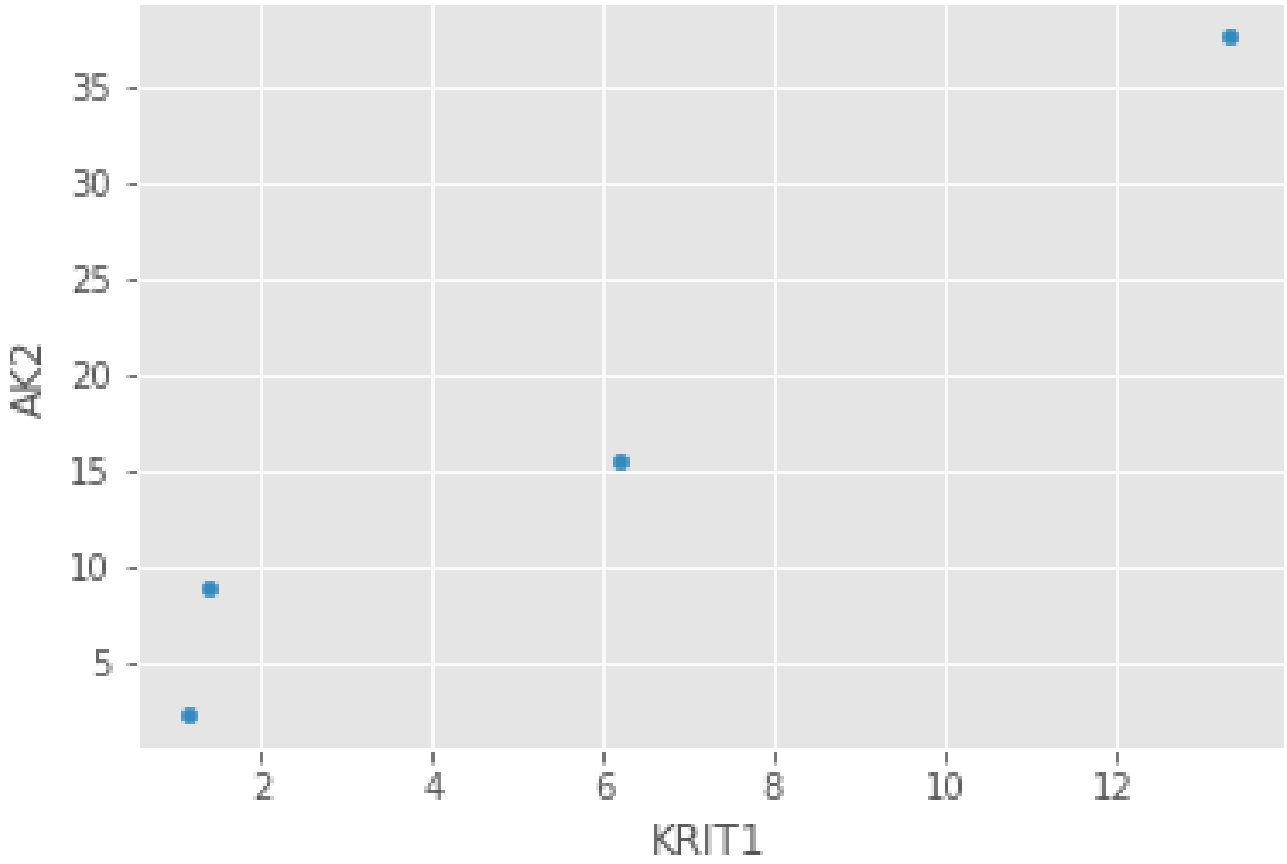
```
<tr>
  <th>ENCFF060LPA</th>
  <td>6.21</td>
  <td>15.57</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
```

```
  <th>ENCFF262OBL</th>
  <td>13.36</td>
  <td>37.62</td>
  <td>adult</td>
  <td>male</td>
</tr>
<tr>
  <th>ENCFF289HGQ</th>
  <td>1.15</td>
  <td>2.31</td>
  <td>newborn</td>
  <td>male, female</td>
</tr>
<tr>
  <th>ENCFF673KYR</th>
  <td>1.37</td>
  <td>8.95</td>
  <td>unknown, fetal</td>
  <td>unknown, female</td>
</tr>
```

绘制散点图

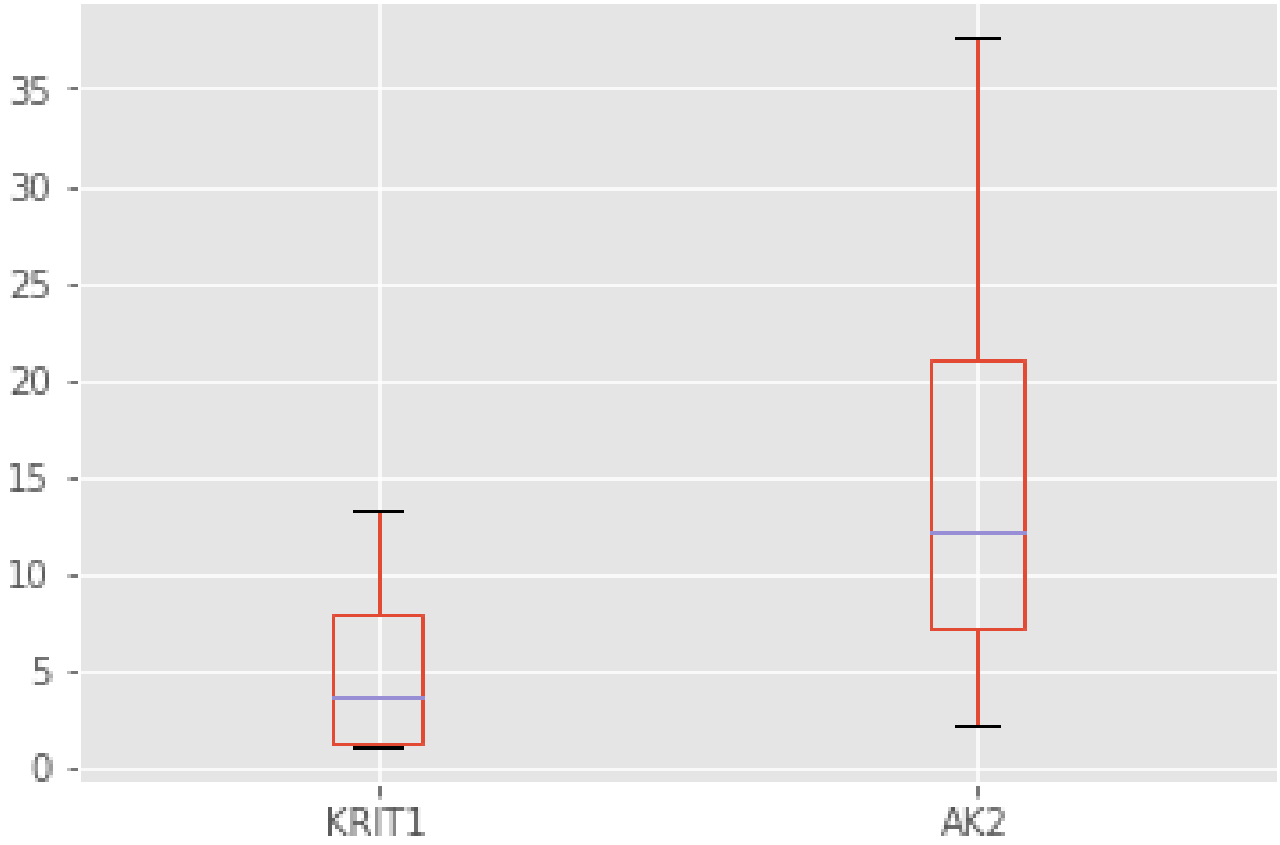
```
target_expr_meta.plot.scatter(x='KRIT1', y='AK2')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf95322390>
```



绘制箱线图

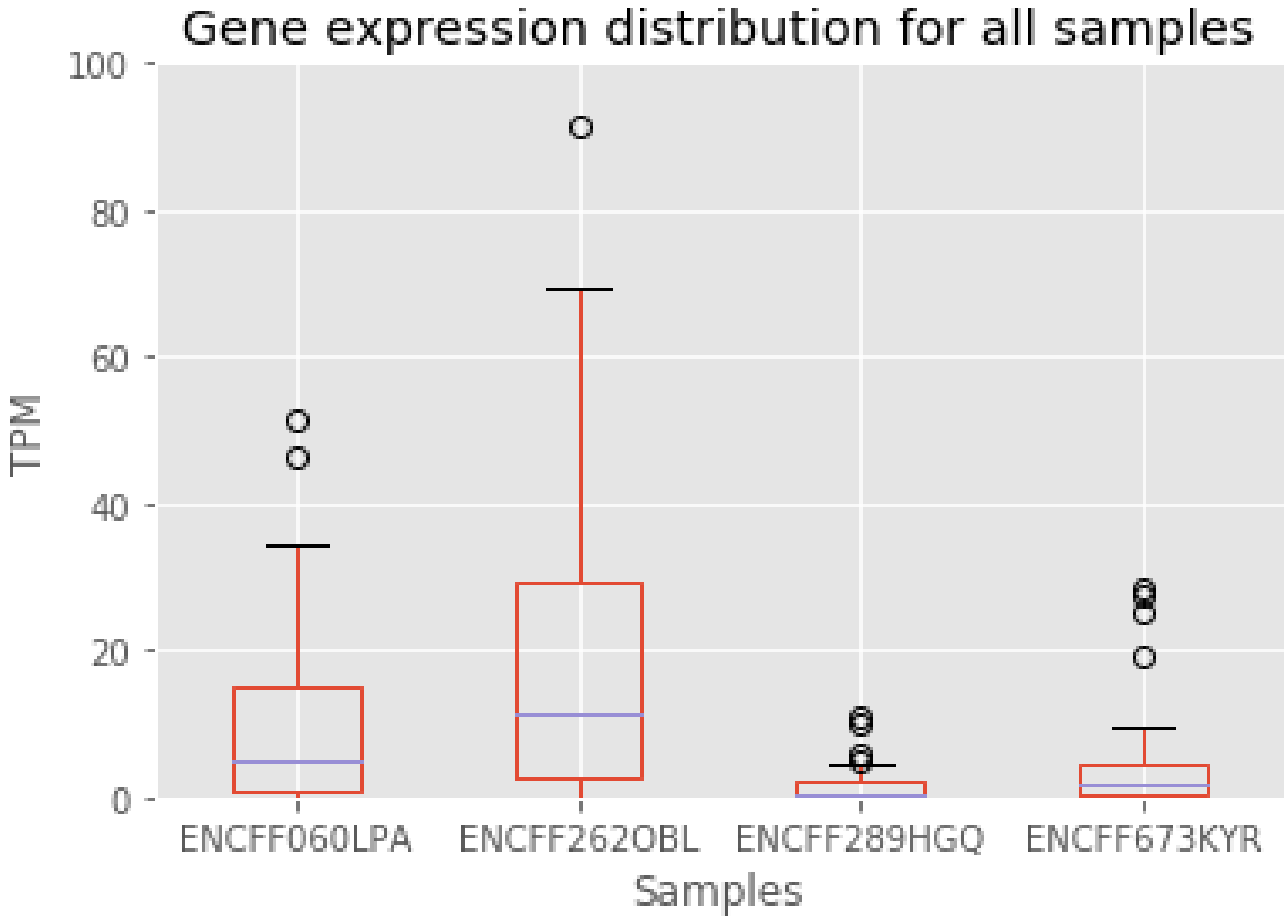
```
a = target_expr_meta.boxplot(["KRIT1", "AK2"])
```



绘制每个样品的基因表达分布

```
ax = TPM_mat.boxplot(list(TPM_mat.columns))
ax.set_ylim(0,100)
ax.set_ylabel("TPM")
ax.set_xlabel("Samples")
ax.set_title("Gene expression distribution for all samples")
```

```
Text(0.5,1,'Gene expression distribution for all samples')
```



更多坐标轴调整见 https://matplotlib.org/api/axes_api.html#axis-labels-title-and-legend

5.5 Seaborn 绘图

Seaborn 是基于 matplotlib 的 python 可视化库，提供更高级的接口和更好的定制性，支持 numpy 和 pandas 数据结构，和 scipy 和 statsmodels 的统计计算。

```
# 导入 seaborn 库，并给予一个更简短的名字，方便后续引用
import seaborn as sns
```

```
TPM_mat.head(3)
```

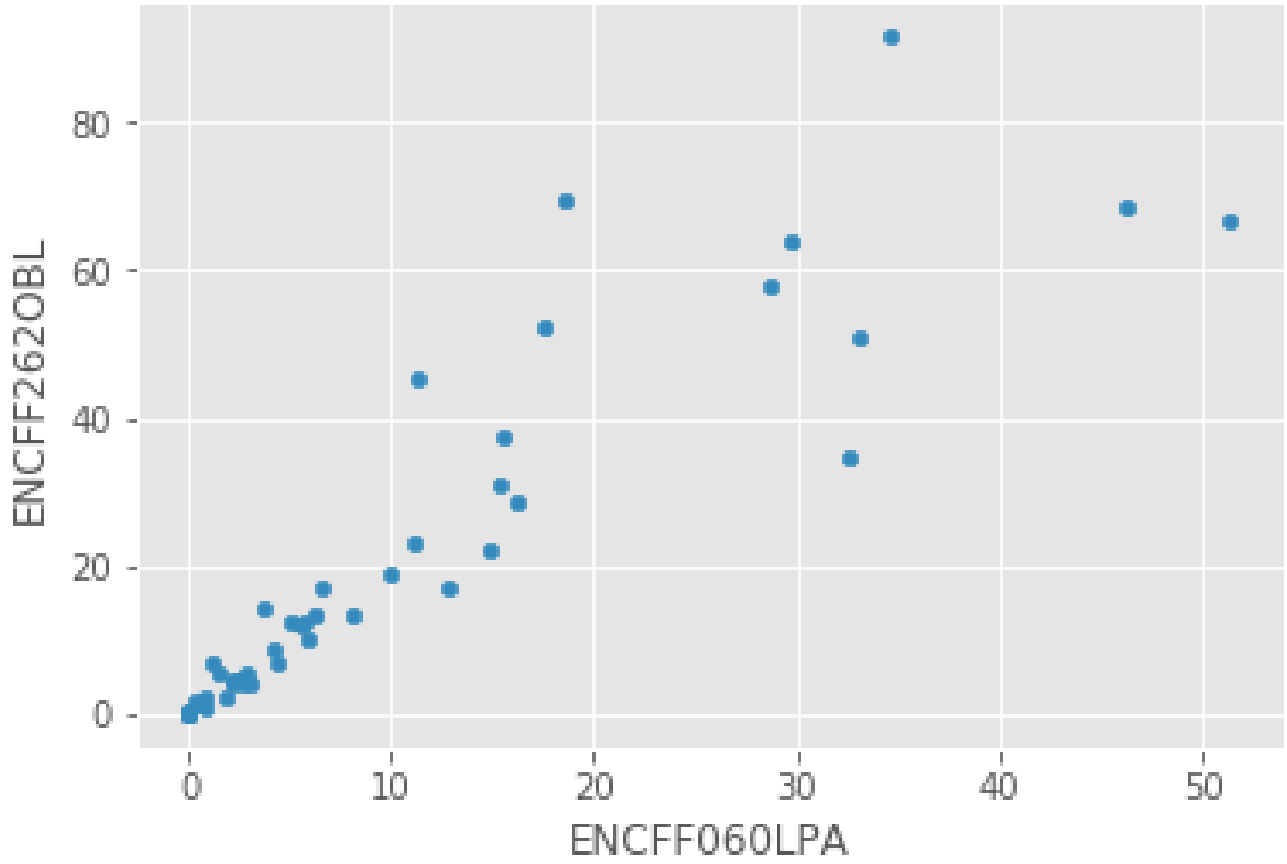
```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
```

```
  <th>ENCFF289HGQ</th>
  <th>ENCFF673KYR</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>


<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
  <td>1.03</td>
  <td>2.42</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
  <td>1.45</td>
  <td>1.80</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>2.48</td>
  <td>0.24</td>
  <td>0.38</td>
</tr>
```

Pandas 绘图

```
TPM_mat.plot(kind="scatter", x="ENCFF060LPA", y="ENCFF262OBL")
```

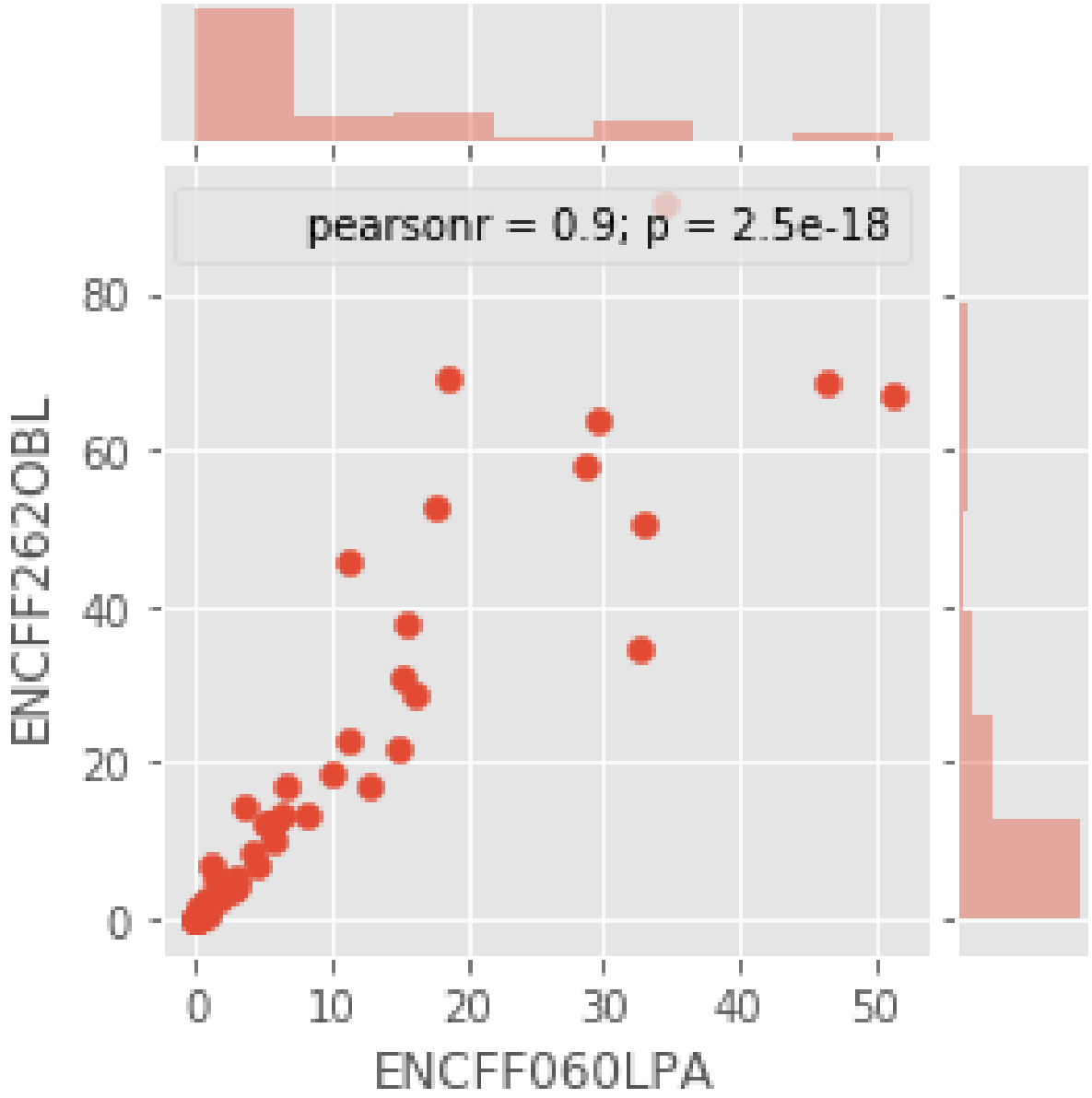
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf84af0a20>
```



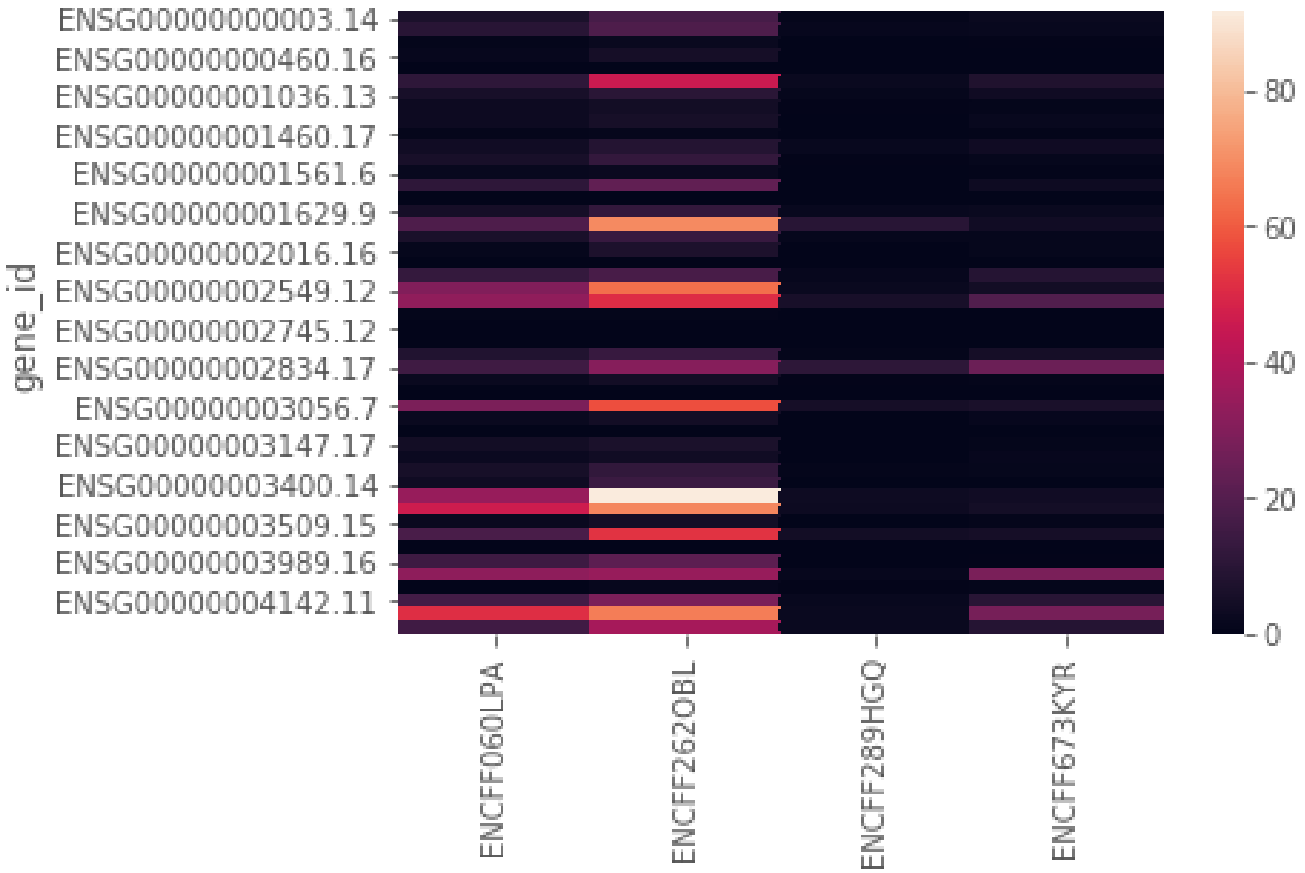
Seaborn 绘图 (还可以直接显示直方图，计算相关性)

```
sns.jointplot(x="ENCFF060LPA", y="ENCFF262OBL", data=TPM_mat, size=4)
```

```
<seaborn.axisgrid.JointGrid at 0x7fbf9531a1d0>
```

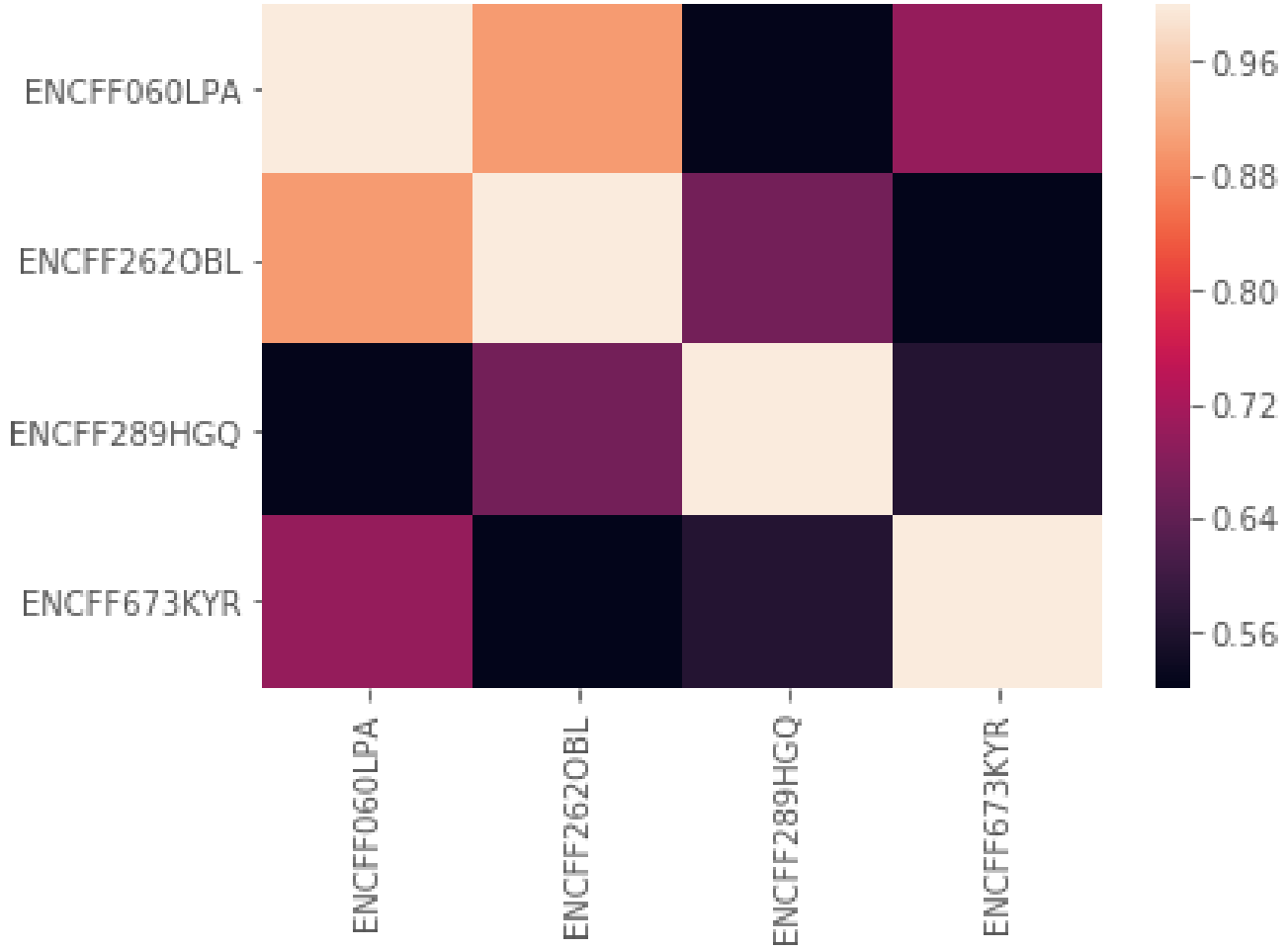



```
sp = sns.heatmap(TPM_mat)
```



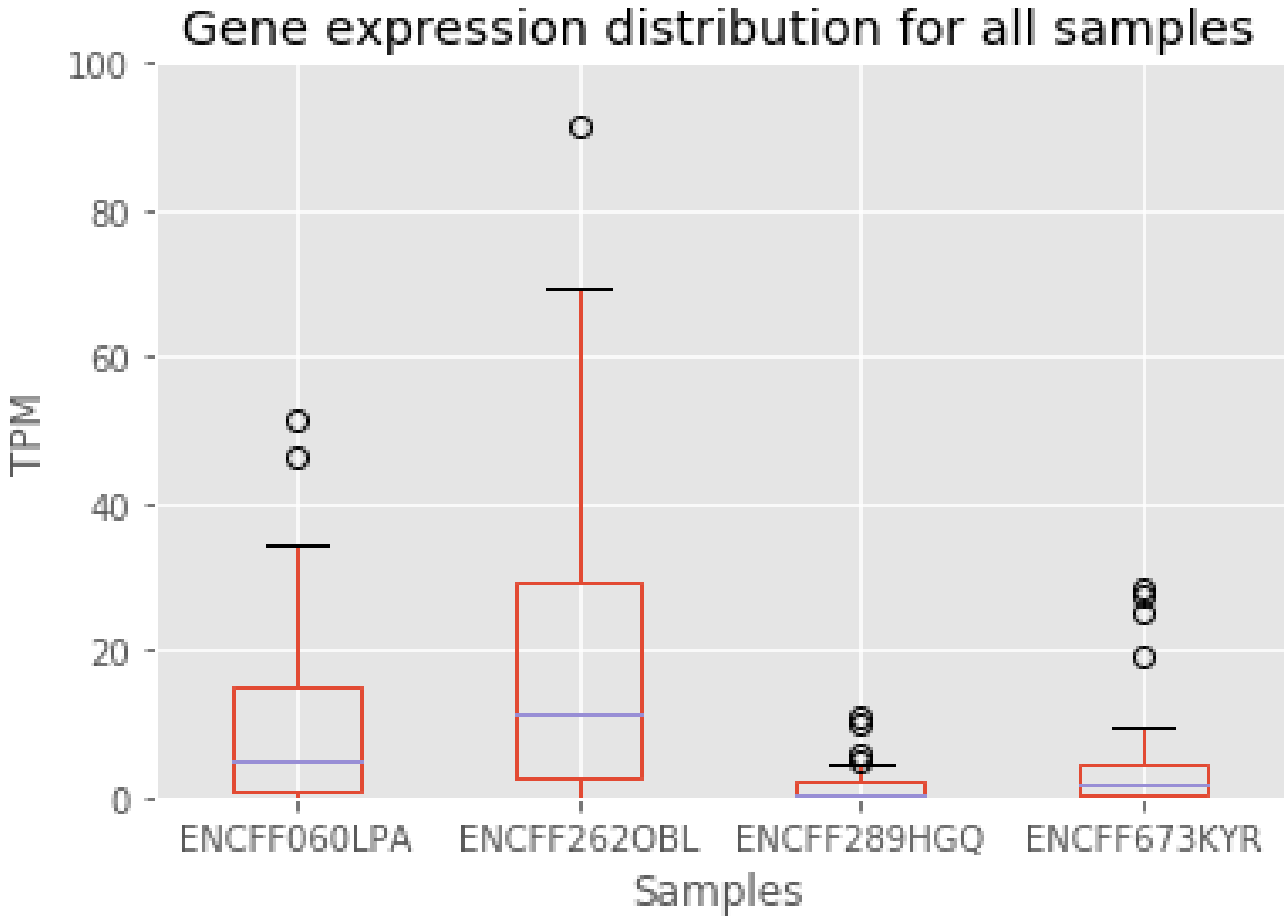

```
TPM_mat_cor = TPM_mat.corr()
sns.heatmap(TPM_mat_cor)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf8494a2b0>
```




```
ax = TPM_mat.boxplot(list(TPM_mat.columns))
ax.set_ylim(0,100)
ax.set_ylabel("TPM")
ax.set_xlabel("Samples")
ax.set_title("Gene expression distribution for all samples")
```

```
Text(0.5,1,'Gene expression distribution for all samples')
```




```
TPM_mat['gene_id'] = TPM_mat.index
TPM_mat.head(4)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>ENCFF060LPA</th>
  <th>ENCFF262OBL</th>
  <th>ENCFF289HGQ</th>
  <th>ENCFF673KYR</th>
  <th>gene_id</th>
</tr>
<tr>
  <th>gene_id</th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
  <th></th>
</tr>
```

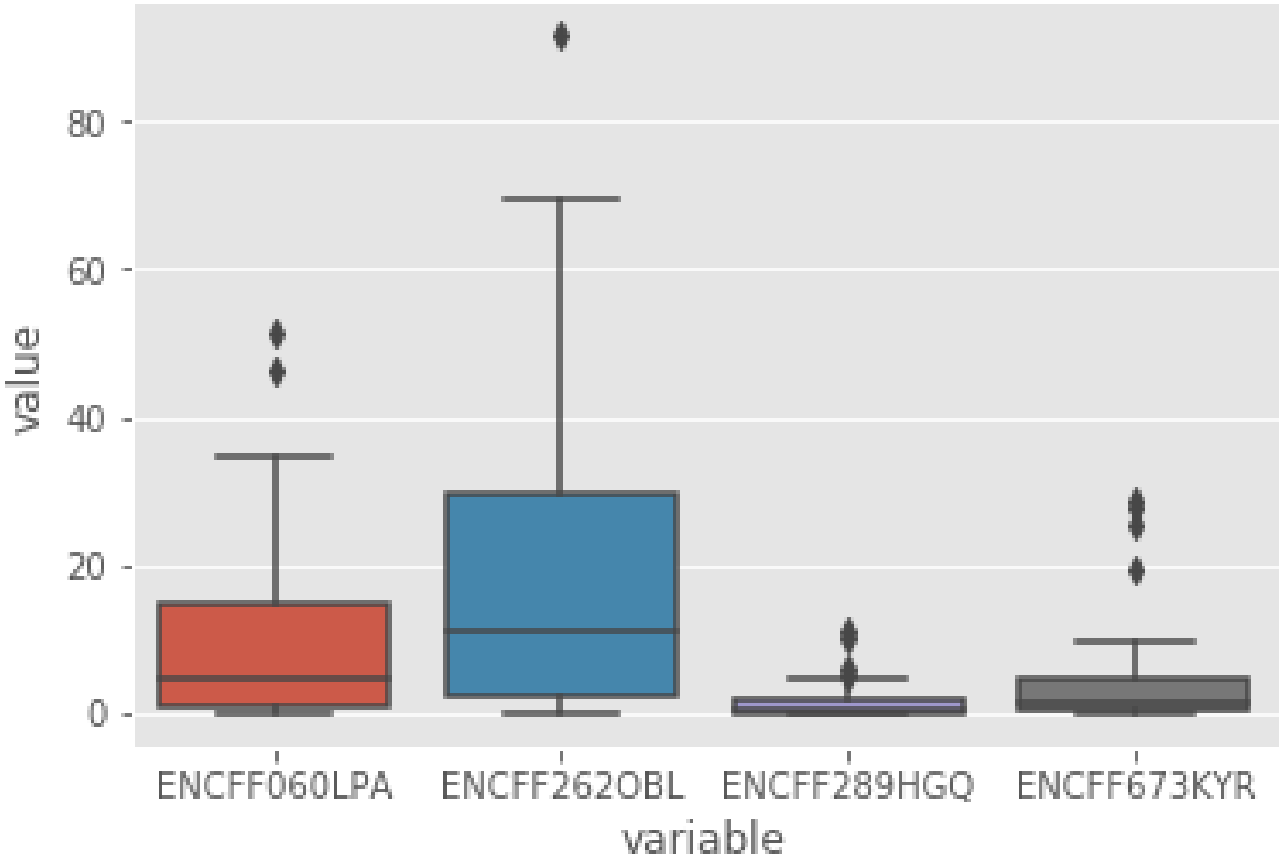
```
<tr>
  <th>ENSG00000000003.14</th>
  <td>6.64</td>
  <td>17.13</td>
  <td>1.03</td>
  <td>2.42</td>
  <td>ENSG00000000003.14</td>
</tr>
<tr>
  <th>ENSG00000000419.12</th>
  <td>9.91</td>
  <td>18.86</td>
  <td>1.45</td>
  <td>1.80</td>
  <td>ENSG00000000419.12</td>
</tr>
<tr>
  <th>ENSG00000000457.13</th>
  <td>0.86</td>
  <td>2.48</td>
  <td>0.24</td>
  <td>0.38</td>
  <td>ENSG00000000457.13</td>
</tr>
<tr>
  <th>ENSG00000000460.16</th>
  <td>1.51</td>
  <td>5.36</td>
  <td>0.26</td>
  <td>0.16</td>
  <td>ENSG00000000460.16</td>
</tr>
```

```
#http://pandas.pydata.org/pandas-docs/stable/generated/pandas.wide_to_long.html
TPM_melt = pd.melt(TPM_mat, id_vars=['gene_id'])

TPM_melt.head(3)
```

```
<tr style="text-align: right;">
  <th></th>
  <th>gene_id</th>
  <th>variable</th>
  <th>value</th>
</tr>


<tr>
```

```
  <th>0</th>
  <td>ENSG00000000003.14</td>
  <td>ENCFF060LPA</td>
  <td>6.64</td>
</tr>
<tr>
  <th>1</th>
  <td>ENSG00000000419.12</td>
  <td>ENCFF060LPA</td>
  <td>9.91</td>
</tr>
<tr>
  <th>2</th>
  <td>ENSG00000000457.13</td>
  <td>ENCFF060LPA</td>
  <td>0.86</td>
</tr>
```

```
ax = sns.boxplot(x="variable", y="value", data=TPM_melt)
```



6 Python 科学计算

6.1 NumPy

6.1.1 NumPy 数组

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

```
## Numpy 克服了 Python 中 list 速度慢的缺点，创建了新数据类型 ndarray
## ndarray 的每列元素一般是相同类型的，是浮点数、整型或字符串，这点和 list 不同
## 下面的例子测试对比了 ndarray 和 list 速度上的区别
## 首先建立一个 0...10^7-1 的 10^7 个元素的数组
arr=np.arange(1e7)
## 将 ndarray 转换为 list
larr=arr.tolist()
## 工具函数，模拟 ndarray 把 list 的每个元素乘以一个标量的运算
def list_times(alist, scalar):
    for i, val in enumerate(alist):
        alist[i]=val*scalar
    return alist
## 比较 ndarray 和 list 每个元素乘以一个标量的运行时间
## 在我的电脑上，ndarray 乘以一个标量的运行时间要比 list 快约 33 倍
%timeit arr*1.1
%timeit list_times(larr,1.1)
```

```
37 ms ± 1.49 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
1.22 s ± 41.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
## 两个 2 维 ndarray 相乘是对应元素相乘，而两个 matrix 相乘是进行矩阵乘法
## matrix 只有 2 维，以下代码运行会出错：shape too large to be a matrix
arr=np.zeros((3,3,3))
mat=np.matrix(arr)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-2-667d2452a8e0> in <module>()
```

```
      2 # matrix只有2维，以下代码运行会出错
      3 arr=np.zeros((3,3,3))
----> 4 mat=np.matrix(arr)


~/anaconda3/lib/python3.6/site-packages/numpy/matrixlib/defmatrix.py in __new__(subtype, data, d
    224             else:
    225                 intype = N.dtype(dtype)
--> 226             new = data.view(subtype)
    227             if intype != data.dtype:
    228                 return new.astype(intype)


~/anaconda3/lib/python3.6/site-packages/numpy/matrixlib/defmatrix.py in __array_finalize__(self,
    269                 return
    270             elif (ndim > 2):
--> 271                 raise ValueError("shape too large to be a matrix.")
    272         else:
    273             newshape = self.shape


ValueError: shape too large to be a matrix.
```

```
## 在 NumPy 中，创建数组有多种方法
## 首先创建一个 list，然后用 np.array() 方法把它包裹起来
alist=[1,2,3]
arr=np.array(alist)
arr
```

6.1.1.1 创建数组和定义数据类型

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

```
## 创建一个 5 个元素的全零数组
arr=np.zeros(5)
arr
```

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

```
## 创建一个从 0 到 99 的数组
arr=np.arange(100)
arr
```

```
array([ 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, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
## 10 到 99 的数组？
arr=np.arange(10,100)
arr
```

```
array([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, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
       61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
       78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
       95, 96, 97, 98, 99])
```

```
## 0 到 1，中间有 100 步，linear space
arr=np.linspace(0,1,100)
arr
```

```
array([ 0.        ,  0.01010101,  0.02020202,  0.03030303,  0.04040404,
        0.05050505,  0.06060606,  0.07070707,  0.08080808,  0.09090909,
        0.1010101 ,  0.11111111,  0.12121212,  0.13131313,  0.14141414,
        0.15151515,  0.16161616,  0.17171717,  0.18181818,  0.19191919,
        0.2020202 ,  0.21212121,  0.22222222,  0.23232323,  0.24242424,
        0.25252525,  0.26262626,  0.27272727,  0.28282828,  0.29292929,
        0.3030303 ,  0.31313131,  0.32323232,  0.33333333,  0.34343434,
        0.35353535,  0.36363636,  0.37373737,  0.38383838,  0.39393939,
        0.4040404 ,  0.41414141,  0.42424242,  0.43434343,  0.44444444,
        0.45454545,  0.46464646,  0.47474747,  0.48484848,  0.49494949,
        0.50505051,  0.51515152,  0.52525253,  0.53535354,  0.54545455,
        0.55555556,  0.56565657,  0.57575758,  0.58585859,  0.5959596 ,
        0.60606061,  0.61616162,  0.62626263,  0.63636364,  0.64646465,
        0.65656566,  0.66666667,  0.67676768,  0.68686869,  0.6969697 ,
        0.70707071,  0.71717172,  0.72727273,  0.73737374,  0.74747475,
        0.75757576,  0.76767677,  0.77777778,  0.78787879,  0.7979798 ,
        0.80808081,  0.81818182,  0.82828283,  0.83838384,  0.84848485,
```

```
        0.85858586,  0.86868687,  0.87878788,  0.88888889,  0.8989899 ,
        0.90909091,  0.91919192,  0.92929293,  0.93939394,  0.94949495,
        0.95959596,  0.96969697,  0.97979798,  0.98989899,  1.        ])
```

```
## log10 空间里 1 到 10 的数组，中间有 100 步
## numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
## base**start 是数组的第一个元素，base**stop 是数组的最后一个元素
arr=np.logspace(0,1,100,base=10.0)
arr
```

```
array([  1.        ,   1.02353102,   1.04761575,   1.07226722,
         1.09749877,   1.12332403,   1.149757  ,   1.17681195,
         1.20450354,   1.23284674,   1.26185688,   1.29154967,
         1.32194115,   1.35304777,   1.38488637,   1.41747416,
         1.45082878,   1.48496826,   1.51991108,   1.55567614,
         1.59228279,   1.62975083,   1.66810054,   1.70735265,
         1.7475284 ,   1.78864953,   1.83073828,   1.87381742,
         1.91791026,   1.96304065,   2.009233  ,   2.05651231,
         2.10490414,   2.15443469,   2.20513074,   2.25701972,
         2.3101297 ,   2.36448941,   2.42012826,   2.47707636,
         2.53536449,   2.59502421,   2.65608778,   2.71858824,
         2.7825594 ,   2.84803587,   2.91505306,   2.98364724,
         3.05385551,   3.12571585,   3.19926714,   3.27454916,
         3.35160265,   3.43046929,   3.51119173,   3.59381366,
         3.67837977,   3.76493581,   3.85352859,   3.94420606,
         4.03701726,   4.1320124 ,   4.22924287,   4.32876128,
         4.43062146,   4.53487851,   4.64158883,   4.75081016,
         4.86260158,   4.97702356,   5.09413801,   5.21400829,
         5.33669923,   5.46227722,   5.59081018,   5.72236766,
         5.85702082,   5.9948425 ,   6.13590727,   6.28029144,
         6.42807312,   6.57933225,   6.73415066,   6.8926121 ,
         7.05480231,   7.22080902,   7.39072203,   7.56463328,
         7.74263683,   7.92482898,   8.11130831,   8.30217568,
         8.49753436,   8.69749003,   8.90215085,   9.11162756,
         9.32603347,   9.54548457,   9.77009957,  10.        ])
```

```
## 创建一个 5*5 的全零数组
image=np.zeros((5,5))
image
```

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

```
## 创建一个 5*5*5 的全 1 的 cube
## astype() 方法将数组的元素全部设为整型
cube=np.zeros((5,5,5)).astype(int)+1
cube
```

```
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, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]]])
```

```
## 或者用更简单的方法创建全 1 数组，元素是 16 位浮点精度
cube=np.ones((5,5,5)).astype(np.float16)
cube
```

```
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.,  1.]],
```

```
       [[ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.]],

       [[ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.]],

       [[ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.],
        [ 1.,   1.,   1.,   1.,   1.]],

       [[ 1.,   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=float16)
```

```
## NumPy 在生成数组时，默认使用系统的字长来创建数组元素
## 在 64 位的 Python 环境中，数组元素默认为 64 位精度的浮点数
## 这种设定消耗大量内存，很多时候并非必要
## 在创建数组时，用户可以自己设定元素的精度，即把 dtype 参数设为 int, numpy.float16,
## numpy.float32, numpy.float64

## 下面定义了一个全零的整型数组
arr1=np.zeros(2,dtype=int)
## 下面定义了一个全零的浮点型数组
arr2=np.zeros(2,dtype=np.float32)
print(arr1)
print(arr2)
```

```
[0 0]
[ 0.  0.]
```

```
## 数组变形
## 创建一个 125 个元素的数组
arr1d=np.arange(125)
## 把数组变形为 5*5*5 的三维数组
arr3d=arr1d.reshape((5,5,5))
## 另一种效果相同的变形方法
```

```
arr3d=np.reshape(arr1d,(5,5,5))
print(arr3d)
## 把高维数组变形为一维数组
arr4d=np.zeros((10,10,10,10))
arr1d=arr4d.ravel()
print(arr1d.shape)
## 值得注意的是，数组的变形只是改变观察数组的角度，
## 并没有新创建数组，变形后的数组和变形前的数组使用的是相同的内存空间
## 因此改动其中一个数组的元素，另一个数组的元素也会跟着改变
## 要创建内存中完全不同的数组，需要使用 numpy.copy 函数
```

```
[[[  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]]

 [[ 50  51  52  53  54]
  [ 55  56  57  58  59]
  [ 60  61  62  63  64]
  [ 65  66  67  68  69]
  [ 70  71  72  73  74]]

 [[ 75  76  77  78  79]
  [ 80  81  82  83  84]
  [ 85  86  87  88  89]
  [ 90  91  92  93  94]
  [ 95  96  97  98  99]]

 [[100 101 102 103 104]
  [105 106 107 108 109]
  [110 111 112 113 114]
  [115 116 117 118 119]
  [120 121 122 123 124]]]
(10000,)
```

```
## 数组一般来说是只包含一种数据类型，不过有些时候数组可以用来存储更复杂的数据结构，
```

```
## 每列由不同的数据类型组成，叫做记录数组
## 创建一个全零数组，定义列的类型（i4：32 位整数，f4：32 位浮点数，a10：长度为 10 的字符串）
recarr=np.zeros((2,),dtype=('i4,f4,a10'))
## 创建我们想放进 recarr 数组的列
col1=np.arange(2)+1 # array([1,2])
col2=np.arange(2,dtype=np.float32) # array([0.,1.], dtype=float32)
col3=["Hello","World"]
## 创建一个列表，整合上面 3 列
toadd=list(zip(col1,col2,col3))
## 给 recarr 赋值
recarr[:]=toadd
recarr
## 结果中字符串前的"b"：python3.x 里默认的 str 是 (py2.x 里的)unicode,
## bytes 是 (py2.x) 的 str, b"前缀代表的就是 bytes
```

6.1.1.2 记录数组（Record Arrays）

```
array([(1,  0., b'Hello'), (2,  1., b'World')],
      dtype=[('f0', '<i4'), ('f1', '<f4'), ('f2', 'S10')])
```

```
## 给每一列赋一个名字，默认的名字是 f0, f1, f2
recarr.dtype.names=("Integers","Floats","Strings")
## 用列的名字访问一列
recarr["Integers"]
```

```
array([1, 2], dtype=int32)
```

```
## 要返回任意列，在 Python 的列表里不太容易，但在 NumPy 数组里很方便
## 定义一个列表
alist=[[1,2],[3,4]]
## 把列表转换为一个 NumPy 数组
arr=np.array(alist)
## 打印 (0,1) 元素
print(arr[0,1])
## 打印第二列
print(arr[:,1])
## 打印第二行
print(arr[1,:])
```

6.1.1.3 索引和切割

```
2
[2 4]
[3 4]
```

```
## 条件索引，常用 numpy.where()，可以返回数组中需要的索引，基于条件，不考虑维度
## 创建一个数组
arr=np.arange(5)
## 创建索引数组
index=np.where(arr>2)
print(index)
## 根据索引创建需要的数组
new_arr1=arr[index]
print(new_arr)
## 使用 np.delete() 删除特定的索引
## 删除 index 中包含的元素
new_arr2=np.delete(arr,index)
print(new_arr2)
## 用简单的布尔列表作为下标返回需要的数组
## 使用布尔索引获得需要的元素比 np.where() 要迅速，并且可以通过 ~index 轻易地反转布尔数组
index=arr>2
print(index)
new_arr=arr[index]
print(new_arr)
```

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

6.1.2 布尔语句和 NumPy 数组

```
## 创建一个图片
img1=np.zeros((20,20))+3
img1[4:-4,4:-4]=6
img1[7:-7,7:-7]=9
## 见 Plot A

## compound_index 变量存储所有大于 2 或小于 6 的下标
index1=img1>2
index2=img1<6
compound_index=index1 & index2
```

```
## 复合索引的语句也可以写成这样：
compound_index=(img1>3) & (img1<7)
img2=np.copy(img1)
img2[compound_index]=0
## 见 Plot B

## 使用更复杂的布尔数组
index3=img1==9
index4=(index1 & index2) | index3
img3=np.copy(img1)
img3[index4]=0
## 见 Plot C
```

```
## 仅变更数组中选中变量的值
## 依据标准正态分布（均值为 0，方差为 1）创建一个 100 个随机元素的数组
a=rand.randn(100)
print(a)
## 去掉不需要的元素
index=a>0.2
b=a[index]
## 在选出的元素上应用某些运算
b=b**2-2
## 把修改过的元素仍然放回原来的数组，这样就完成了对数组中某些值得变更
a[index]=b
print(a)
```

```
[-1.80712517 -0.13170284  1.93343213  0.80721035  2.52144275  0.16195953
  0.11878839  1.88961184 -0.75881407 -1.22866865 -0.73857745  0.64046896
  0.73563193  0.71783485 -0.37645069  0.90120663 -0.59161068 -1.16366655
 -0.50663906 -0.36447979 -1.8654699   0.92406343  0.8004173  -1.41016169
  0.32592465  0.725013    0.29738016 -2.21113871 -0.68122701  0.66455187
 -0.32413105  1.13627295  0.13185364  1.38536725  0.19462378 -1.80775106
 -0.06199759 -1.30422952 -0.30685345  0.08940313 -0.67023186 -0.15051653
 -0.15879759  1.35984567 -0.2328225  -1.7265208   0.67232358 -0.59098342
 -1.20227104 -1.08216219 -0.53158487  1.33859499  0.66763318 -0.56431746
  0.75334062  1.44683156  1.9512399   0.42510796  0.34020597  0.72372802
 -0.10372294  0.184639    0.01761624 -0.69082466  0.36432908 -0.56651026
  0.21865567 -0.42096267 -0.73512854 -0.05493638 -1.94869892  0.76399321
  1.41936848  0.05750032 -0.12937963 -0.02329108 -1.47156894 -0.84254776
  0.75245785  0.07682322  0.96333808  0.92090036  0.76489123  0.63520238
 -0.08737849 -1.2083732   0.21328152  0.92891946  0.26792553  0.0672728
 -1.74821241  0.69335543 -0.60076278 -0.11200103 -1.4392716  -0.35400569
 -0.05871362  0.8178716   1.08160928  0.77033013]
[-1.80712517 -0.13170284  1.73815979 -1.34841145  4.35767354  0.16195953
  0.11878839  1.57063289 -0.75881407 -1.22866865 -0.73857745 -1.58979952
 -1.45884566 -1.48471313 -0.37645069 -1.1878266  -0.59161068 -1.16366655
```

```
 -0.50663906 -0.36447979 -1.8654699  -1.14610677 -1.35933214 -1.41016169
 -1.89377312 -1.47435616 -1.91156504 -2.21113871 -0.68122701 -1.55837081
 -0.32413105 -0.70888379  0.13185364 -0.08075759  0.19462378 -1.80775106
 -0.06199759 -1.30422952 -0.30685345  0.08940313 -0.67023186 -0.15051653
 -0.15879759 -0.15081976 -0.2328225  -1.7265208  -1.547981   -0.59098342
 -1.20227104 -1.08216219 -0.53158487 -0.20816344 -1.55426593 -0.56431746
 -1.43247792  0.09332155  1.80733714 -1.81928322 -1.8842599  -1.47621775
 -0.10372294  0.184639    0.01761624 -0.69082466 -1.86726432 -0.56651026
 -1.9521897  -0.42096267 -0.73512854 -0.05493638 -1.94869892 -1.41631438
  0.01460689  0.05750032 -0.12937963 -0.02329108 -1.47156894 -0.84254776
 -1.43380718  0.07682322 -1.07197974 -1.15194253 -1.4149414  -1.59651793
 -0.08737849 -1.2083732  -1.95451099 -1.13710863 -1.92821591  0.0672728
 -1.74821241 -1.51925824 -0.60076278 -0.11200103 -1.4392716  -0.35400569
 -0.05871362 -1.33108605 -0.83012136 -1.40659148]
```

6.1.3 NumPy 读写文件

```
## NumPy 读文本文件中的矩阵
arr=np.loadtxt("data5/somefile.txt")
arr
```

```
array([[  2.,   3.,   5.],
       [  7.,  11.,  13.],
       [ 17.,  19.,  23.]])
```

```
## NumPy 写矩阵到文本文件中
## numpy.savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n',
##  header='', footer='', comments='# ', encoding=None)[source]
np.savetxt("data5/somenewfile.txt",arr,"%d","\t")
```

```
## loadtxt() 读取文件中复杂的数据结构
recarr=np.loadtxt("data5/example.txt", dtype={
        "names":("Gene_ID","Sample1","Sample2","Sample3"),
        "formats":("S6","f4","f4","f4")})
recarr
```

```
array([(b'Gene1',   2.29999995,   5.69999981,  11.13000011),
       (b'Gene2',  17.19000053,  23.29000092,  31.37000084)],
      dtype=[('Gene_ID', 'S6'), ('Sample1', '<f4'),
      ('Sample2', '<f4'), ('Sample3', '<f4')])
```

6.1.4 NumPy 的 Math 模块

```
## 解矩阵方程 AX=B
## 定义矩阵 A, B
A=np.matrix([[3,6,-5],[1,-3,2],[5,-1,4]])
B=np.matrix([[12],[-2],[10]])
## 解矩阵方程，等号两边同时左乘 A^(-1)
X=A**-1*B
print(X)
```

6.1.4.1 线性代数

```
[[ 1.75]
 [ 1.75]
 [ 0.75]]
```

```
## 使用 np.array，更快
a=np.array([[3,6,-5],[1,-3,2],[5,-1,4]])
b=np.array([[12],[-2],[10]])
x=np.linalg.inv(a).dot(b)
print(x)
```

```
[[ 1.75]
 [ 1.75]
 [ 0.75]]
```

6.2 SciPy

6.2.1 最优化和最小化

```
from scipy.optimize import curve_fit
```

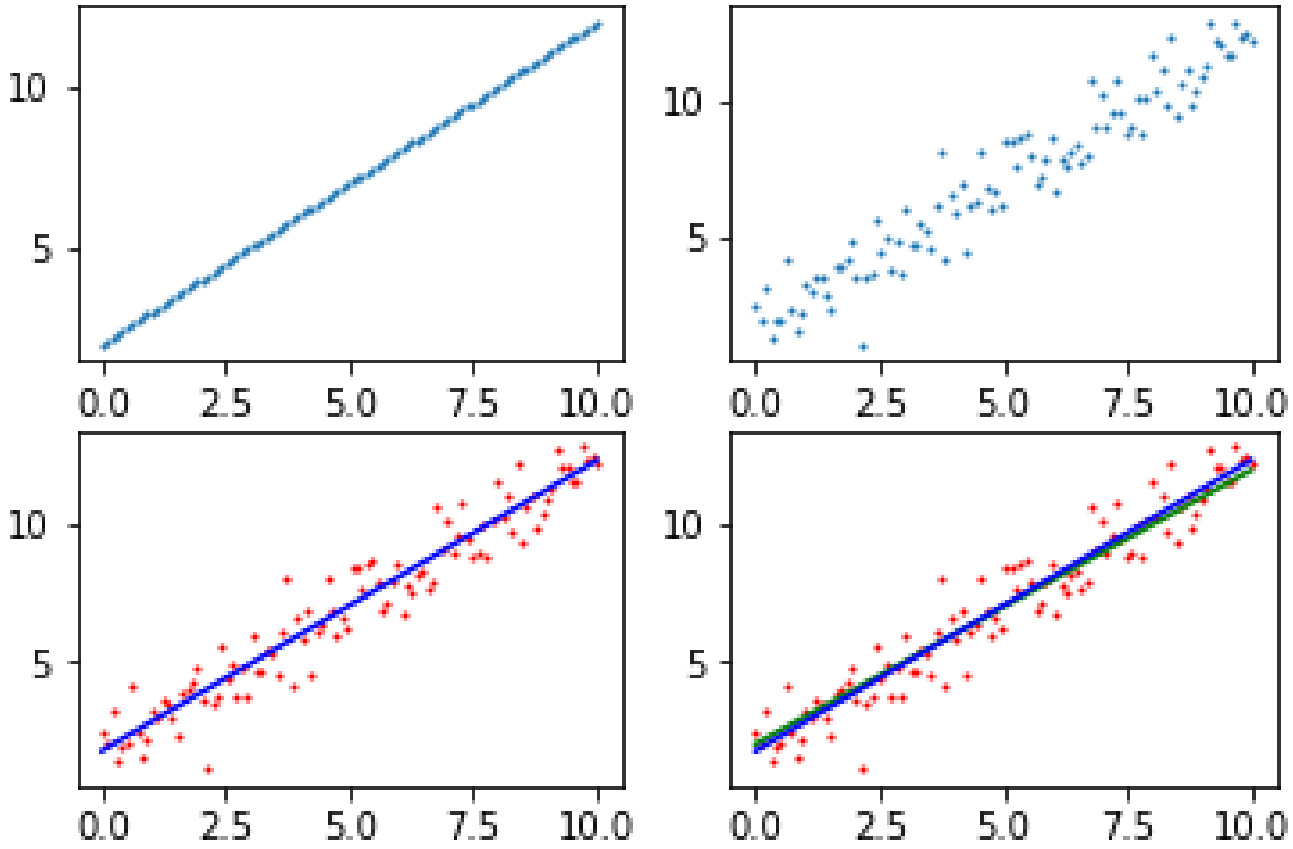
```
## 拟合线性分布
## 创建一个函数，用来建模和创建数据
```

```
def func(x,a,b):
    return a*x+b

## 生成干净的数据
x=np.linspace(0,10,100)
y=func(x,1,2)

## 加入噪声
yn=y+0.9*np.random.normal(size=len(x))

## 在有噪声的数据上应用 curve_fit
## popt 返回给定模型（func）下的参数的最佳拟合值
## pcov 返回一个矩阵表示拟合的质量，矩阵对角线上的值是各参数的方差
popt,pcov=curve_fit(func,x,yn)

print(popt)
```

6.2.1.1 数据建模和拟合

```
[ 1.06107439  1.7517619 ]
```

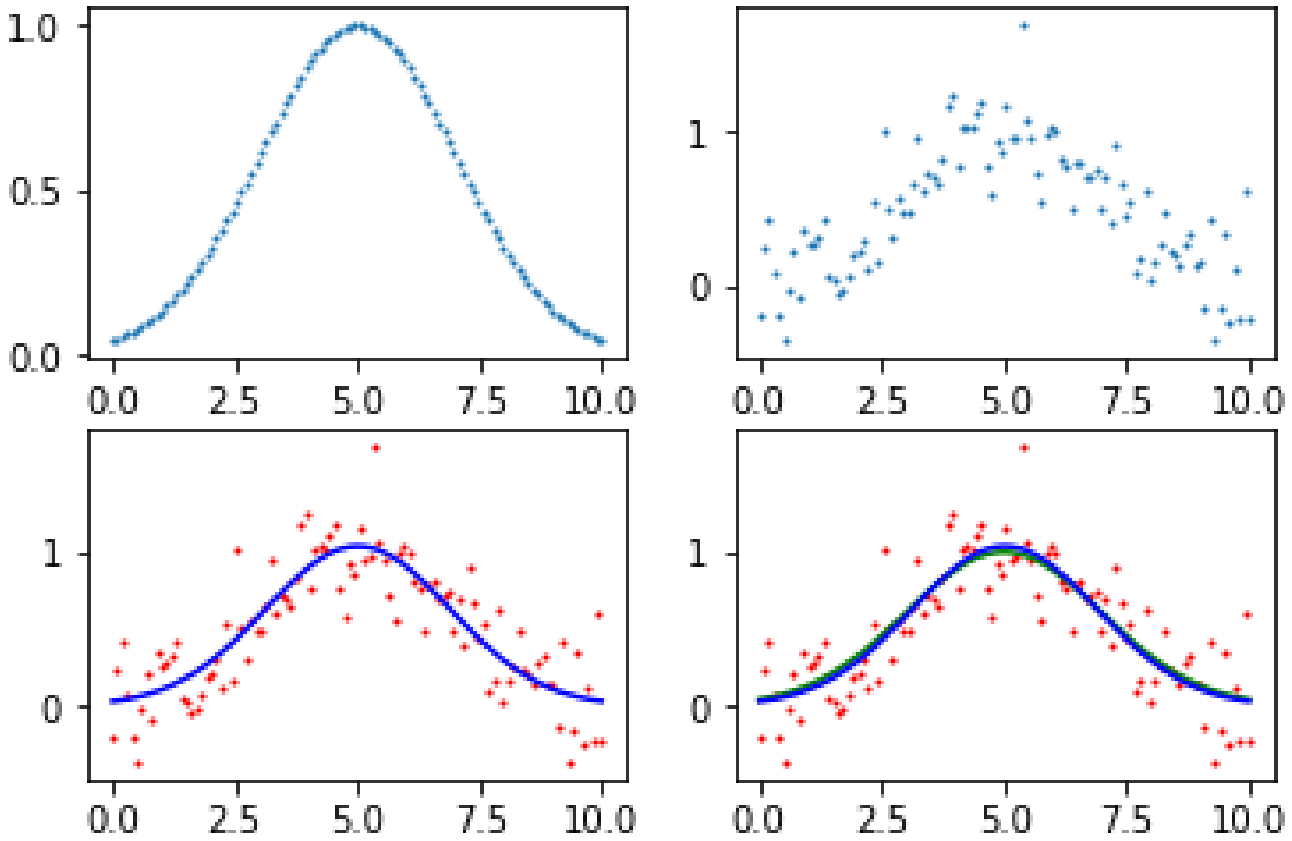
```
fig=plt.figure()

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

ax1.scatter(x,y,1)

ax2.scatter(x,yn,1)

ax3.scatter(x,yn,1,"red")
ax3.plot(x,popt[0]*x+popt[1],"blue")

ax4.scatter(x,yn,1,"red")
ax4.plot(x,y,"green")
ax4.plot(x,popt[0]*x+popt[1],"blue")

plt.show()
```




```
## 拟合高斯分布
## 创建一个函数，用来建模和创建数据
def func(x,a,b,c):
    return a*np.exp(-(x-b)**2/(2*c**2))

## 生成干净的数据
x=np.linspace(0,10,100)
y=func(x,1,5,2)

## 加入噪声
yn=y+0.2*np.random.normal(size=len(x))

## 在有噪声的数据上应用 curve_fit
## popt 返回给定模型（func）下的参数的最佳拟合值
## pcov 返回一个矩阵表示拟合的质量，矩阵对角线上的值是各参数的方差
popt, pcov = curve_fit(func, x, yn)

print(popt)
```

```
[ 1.03822218  5.01637962 -1.88413558]
```

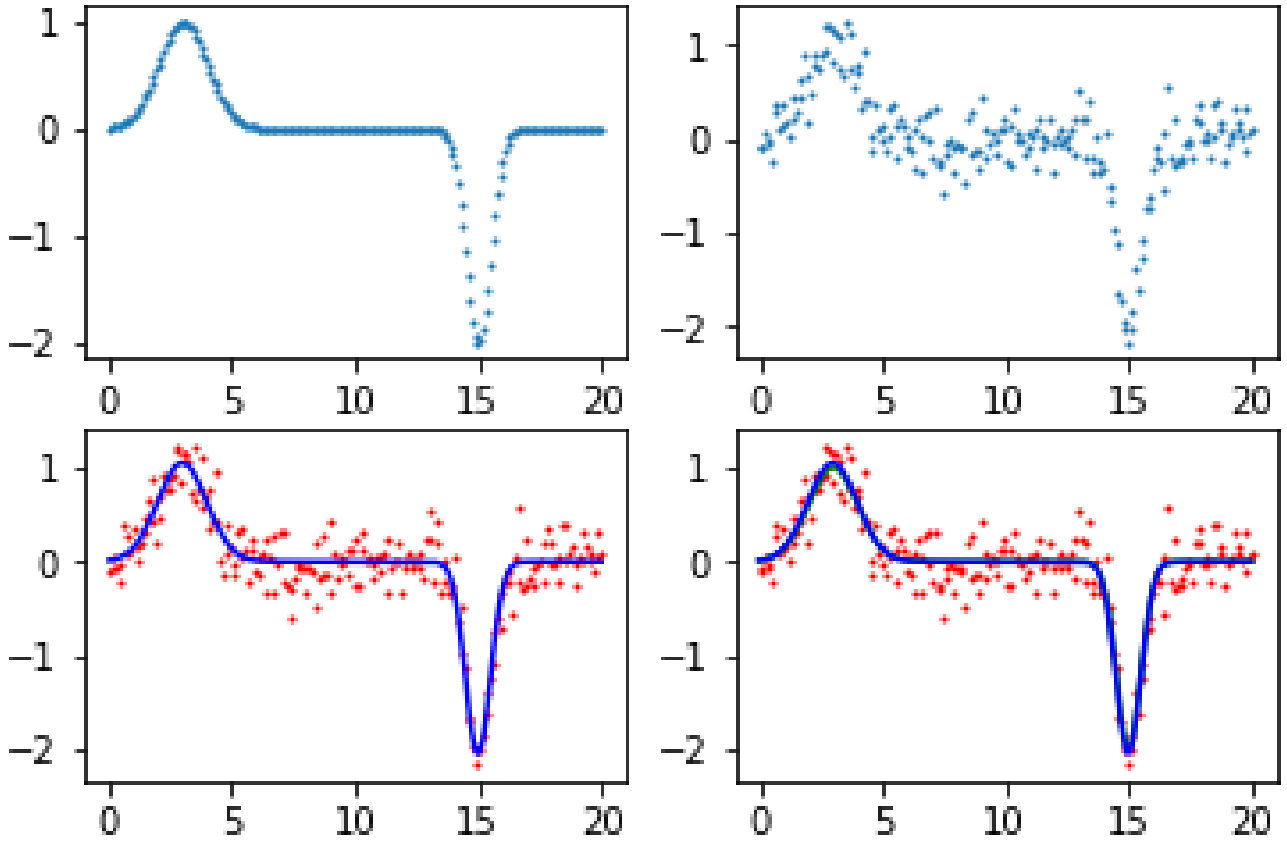
```
fig=plt.figure()

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

ax1.scatter(x,y,1)

ax2.scatter(x,yn,1)

ax3.scatter(x,yn,1,"red")
ax3.plot(x,popt[0]*np.exp(-(x-popt[1])**2/(2*popt[2]**2)),"blue")

ax4.scatter(x,yn,1,"red")
ax4.plot(x,y,"green")
ax4.plot(x,popt[0]*np.exp(-(x-popt[1])**2/(2*popt[2]**2)),"blue")

plt.show()
```



```
## 拟合两个高斯分布的线性组合
## 创建一个函数，用来建模和创建数据
def func(x,a0,b0,c0,a1,b1,c1):
    return a0*np.exp(-(x-b0)**2/(2*c0**2))+a1*np.exp(-(x-b1)**2/(2*c1**2))

## 生成干净的数据
x=np.linspace(0,20,200)
y=func(x,1,3,1,-2,15,0.5)

## 加入噪声
yn=y+0.2*np.random.normal(size=len(x))

## 在有噪声的数据上应用 curve_fit
## popt 返回给定模型（func）下的参数的最佳拟合值
## pcov 返回一个矩阵表示拟合的质量，矩阵对角线上的值是各参数的方差
popt, pcov = curve_fit(func, x, yn)

print(popt)
```

```
[ -2.04644312  14.99180963   0.50668406   1.05196114   2.97782597
   0.99894884]
```

```
fig=plt.figure()

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

ax1.scatter(x,y,1)

ax2.scatter(x,yn,1)

ax3.scatter(x,yn,1,"red")
ax3.plot(x,popt[0]*np.exp(-(x-popt[1])**2/(2*popt[2]**2))+
        popt[3]*np.exp(-(x-popt[4])**2/(2*popt[5]**2)),"blue")

ax4.scatter(x,yn,1,"red")
ax4.plot(x,y,"green")
ax4.plot(x,popt[0]*np.exp(-(x-popt[1])**2/(2*popt[2]**2))+
        popt[3]*np.exp(-(x-popt[4])**2/(2*popt[5]**2)),"blue")
ax4.legend(loc=0, bbox_to_anchor=(1,1))

plt.show()
```



```
from scipy.optimize import fsolve
```

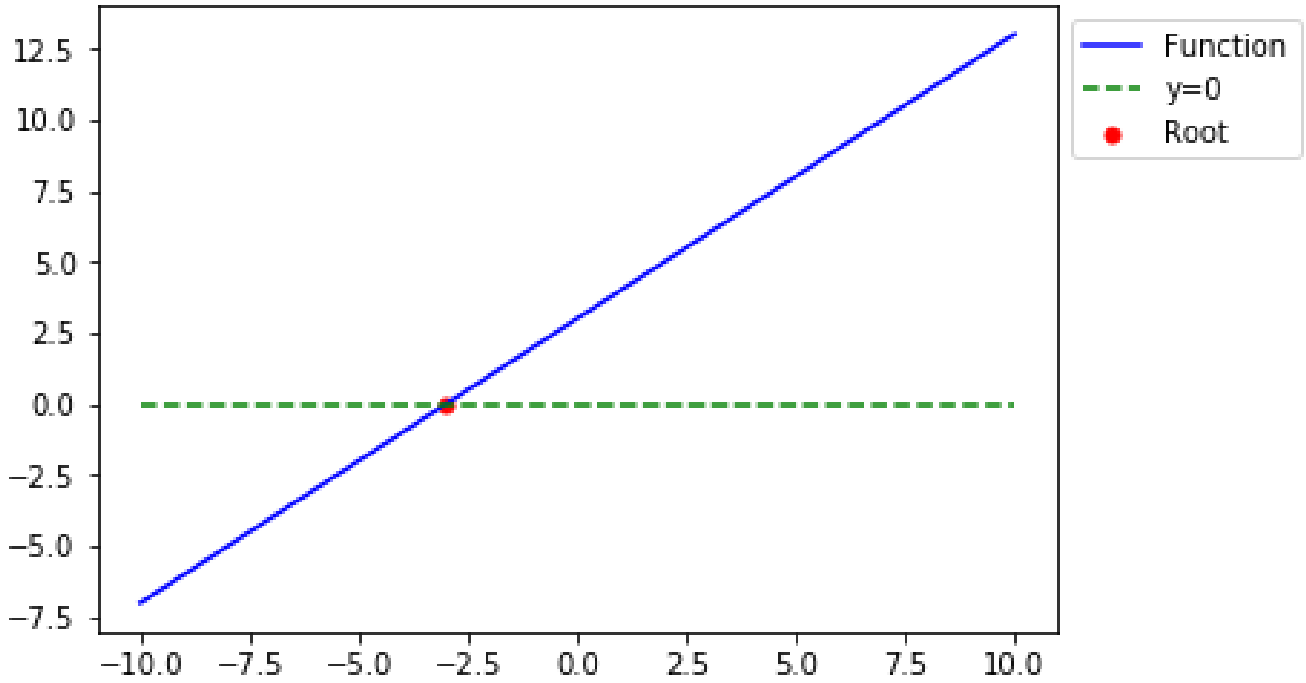
```
line=lambda x:x+3
solution=fsolve(line,-2)
print(solution)
```

6.2.1.2 函数的解

```
[-3.]
```

```
fig = plt.figure()
ax=fig.add_subplot(1,1,1)
x=np.linspace(-10,10,100)
zeros=np.zeros(100)
ax.plot(x,line(x),"blue",label="Function")
```

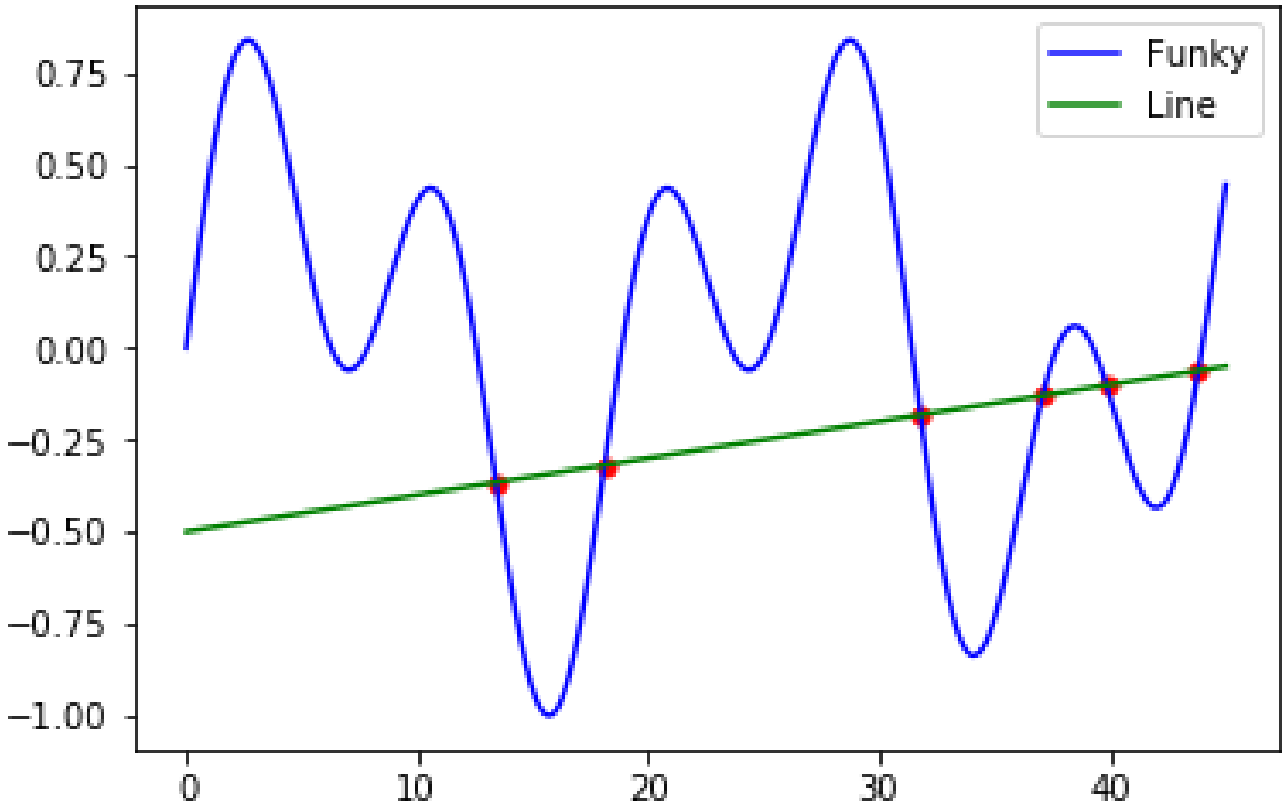
```
ax.plot(x,zeros,"g--",label="y=0")
ax.scatter(solution,line(solution),25,"red",label="Root")
ax.legend(loc="best", bbox_to_anchor=(1,1))
plt.show()
```




```
## 找到两个函数的交点
## 定义一个函数来简化求交点的过程
def findIntersection(func1,func2,x0):
    return fsolve(lambda x:func1(x)-func2(x),x0)

## 定义两个函数，准备求它们的交点
funky=lambda x:np.cos(x/5)*np.sin(x/2)
line=lambda x:0.01*x-0.5

## 设置函数的定义域，求两个函数的交点
x=np.linspace(0,45,10000)
result=findIntersection(funky,line,[15,20,30,35,40,45])

## 打印交点的坐标
print(result,line(result))
```

```
[ 13.40773078  18.11366128  31.78330863  37.0799992   39.84837786
  43.8258775 ] [-0.36592269 -0.31886339 -0.18216691 -0.12920001 -0.10151622 -0.06174122]
```

```
fig = plt.figure()
ax=fig.add_subplot(1,1,1)
x=np.linspace(0,45,10000)
ax.plot(x,funky(x),"blue",label="Funky")
ax.plot(x,line(x),"green",label="Line")
ax.scatter(result,line(result),25,"red")
ax.legend(loc="best", bbox_to_anchor=(1,1))
plt.show()
```



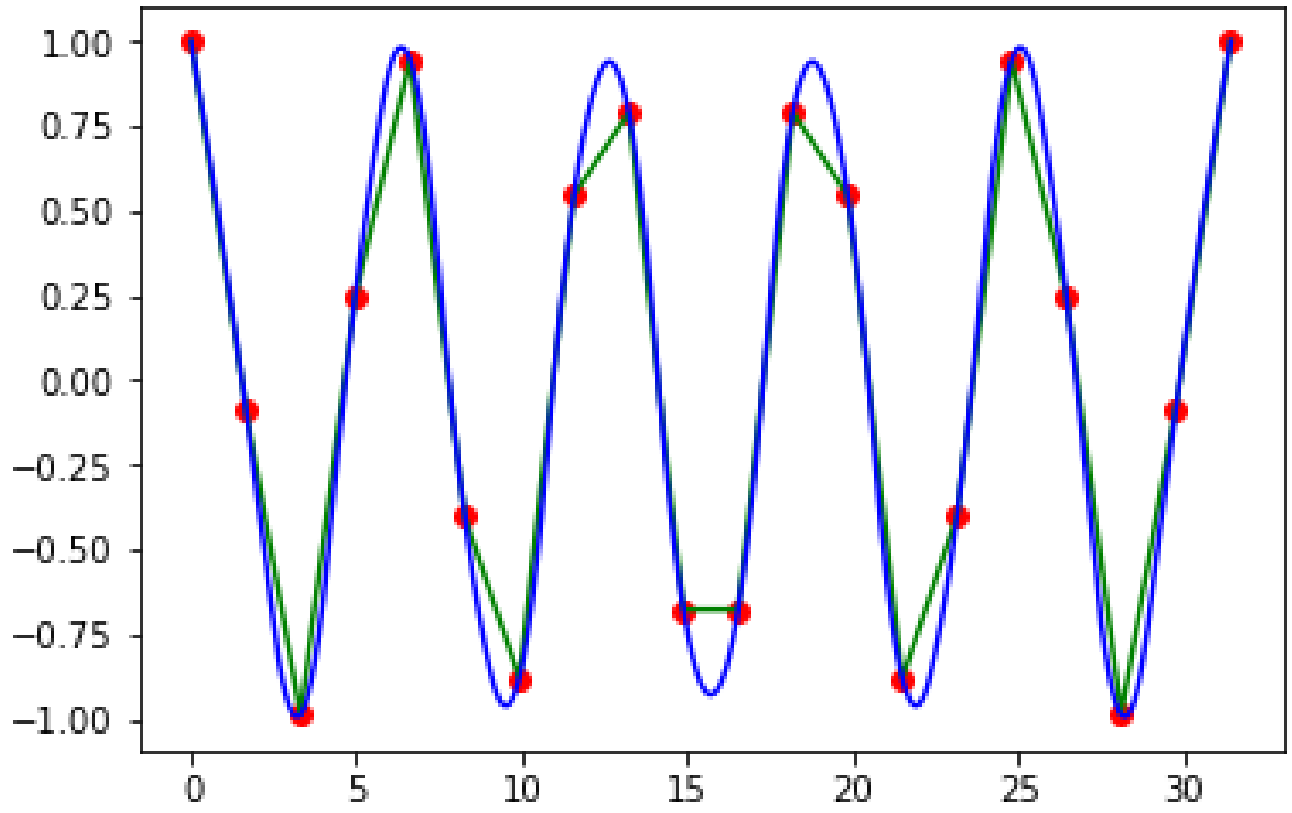
6.2.2 插值

- 插值法在离散数据的基础上补插连续函数，使得这条连续曲线通过全部给定的离散数据点。
- 插值是离散函数逼近的重要方法，利用它可通过函数在有限个点处的取值状况，估算出函数在其他点处的近似值。
- 常用来填充图像变换时像素之间的空隙。
- 计算插值有两种基本方法：
  - 用一个函数对整个数据集进行拟合
  - 对数据集不同的部分用几个不同函数拟合，各函数的连接点都是光滑的
- 第二种方法称为样条插值，当数据的函数形式复杂的时候，样条插值是非常强大的工具

```
from scipy.interpolate import interp1d
```

```
## 创建离散数据集
x=np.linspace(0,10*np.pi,20)
y=np.cos(x)

## 分别用线性函数和二次函数，对数据进行插值
fl=interp1d(x,y,kind="linear")
fq=interp1d(x,y,kind="quadratic")

## 得到两个连续函数上的点
xint=np.linspace(x.min(),x.max(),1000)
yintl=fl(xint)
yintq=fq(xint)
```

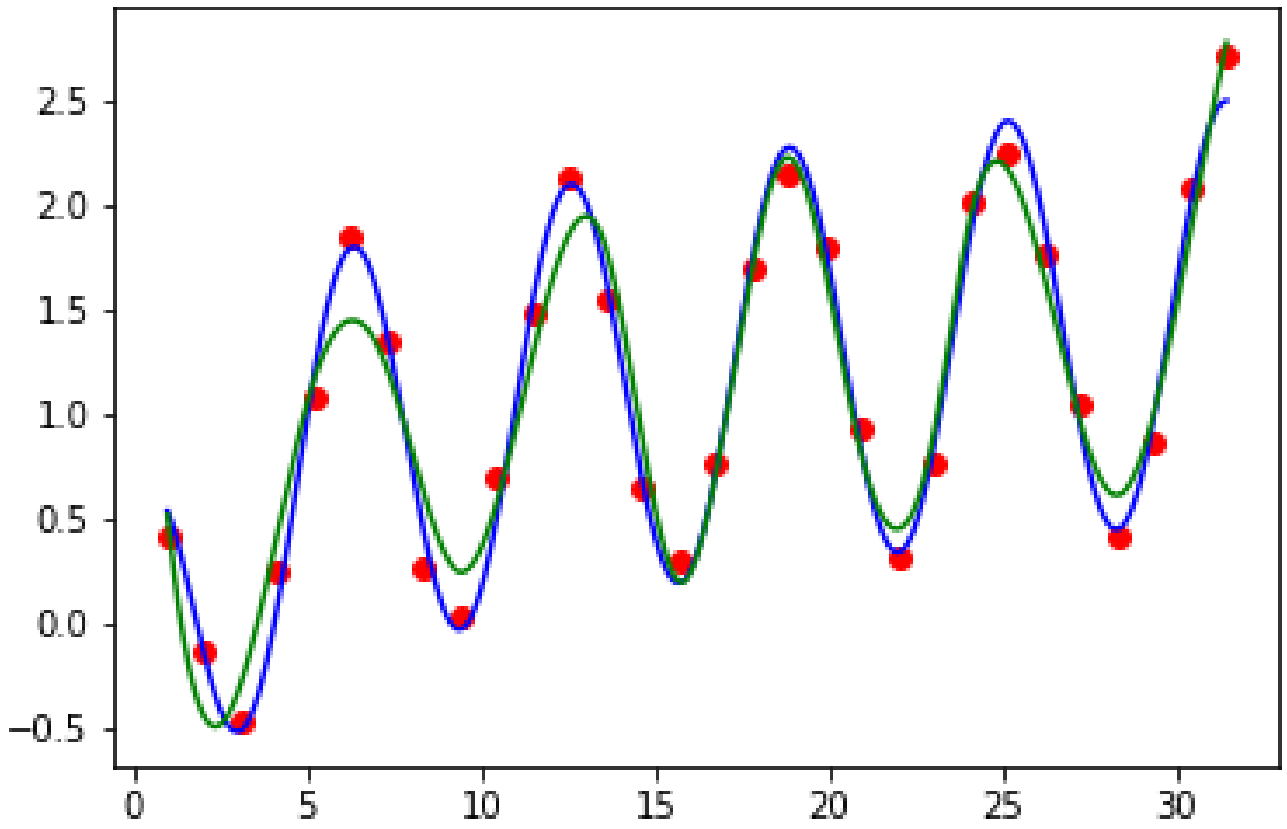
```
fig = plt.figure()
ax=fig.add_subplot(1,1,1)
ax.plot(x,y,"ro",xint,yintl,"g-",xint,yintq,"b-")
plt.show()
```



```
from scipy.interpolate import UnivariateSpline
```

```
## 对有噪声的函数进行插值
## 创建数据集，加入人工噪声
sample=30
x=np.linspace(1,10*np.pi,sample)
y=np.cos(x)+np.log10(x)+np.random.randn(sample)/10

## 对数据进行插值，参数 s 是光滑因子，若 s=0，插值会经过所有点不考虑噪声，若 s=1，则考虑噪声
f=UnivariateSpline(x,y,s=1)

## 得到插值函数上的点
xint=np.linspace(x.min(),x.max(),1000)
yint=f(xint)
```

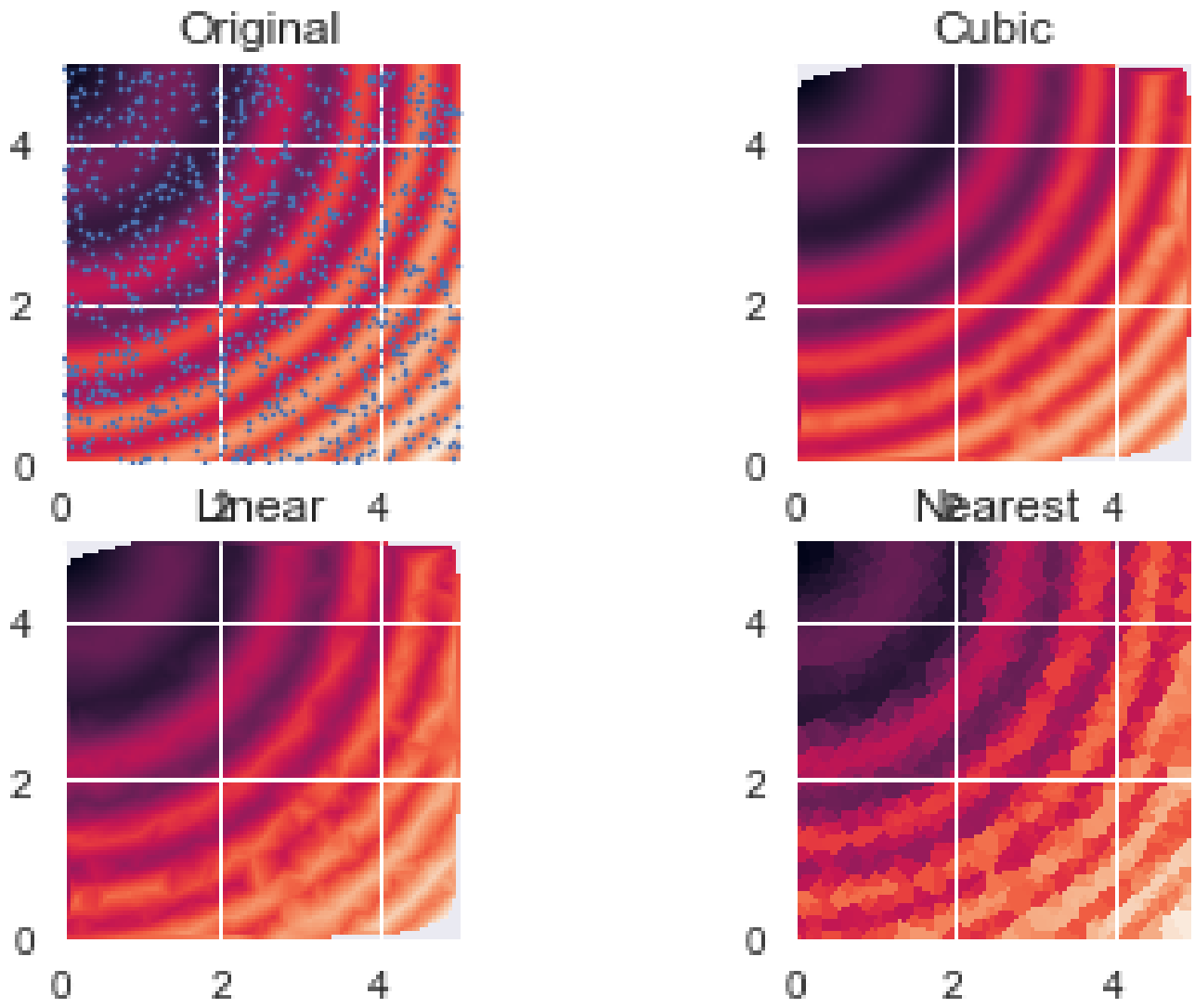
```
fig = plt.figure()
ax=fig.add_subplot(1,1,1)
x1000=np.linspace(1,10*np.pi,1000)
y1000=np.cos(x1000)+np.log10(x1000)
ax.plot(x,y,"ro",x1000,y1000,"b-",xint,yint,"g-")
plt.show()
```



```
from scipy.interpolate import griddata
```

```
# 创建一个 1000*1000 像素的图片，在图片上随机选 1000 个点，看插值函数如何能根据这 1000 个点重构图片
## 创建一个函数
ripple=lambda x,y:np.sqrt(x**2+y**2)+np.sin(x**2+y**2)
## 生成网格数据，复数 1000j 表示在 0 到 5 之间生成 1000 个点，包含 0 和 5
grid_x, grid_y = np.mgrid[0:5:1000j, 0:5:1000j]
## 随机生成插值函数可见的 1000 个点的抽样
xy = np.random.rand(1000, 2)
sample = ripple(xy[:,0] * 5 , xy[:,1] * 5)
## 用三次函数方法生成抽样数据的插值
grid_z0 = griddata(xy * 5, sample, (grid_x, grid_y), method='cubic')
## 用线性方法生成抽样数据的插值
grid_z1 = griddata(xy * 5, sample, (grid_x, grid_y), method='linear')
## 用 Nearest 方法生成抽样数据的插值
grid_z2 = griddata(xy * 5, sample, (grid_x, grid_y), method='nearest')
```

```
plt.subplot(221)
plt.title('Original')
plt.imshow(ripple(grid_x, grid_y).T,extent=(0,5,0,5))
plt.plot(xy[:,0]*5, xy[:,1]*5, 'b.',ms=3)
plt.subplot(222)
plt.title('Cubic')
plt.imshow(grid_z0.T,extent=(0,5,0,5))
plt.subplot(223)
plt.title('Linear')
plt.imshow(grid_z1.T,extent=(0,5,0,5))
plt.subplot(224)
plt.title('Nearest')
plt.imshow(grid_z2.T,extent=(0,5,0,5))
plt.show()
```


Original
Cubic
Linear
Nearest
0
2
4

7 易生信 Python 培训练习和考核题目

欢迎访问我们的视频课程 https://bioinfo.ke.qq.com。

1. `3*2**2` 的输出是多少？(1 分)
2. `8 % 4` 的输出是多少？(1 分)
3. `32 + '32'` 的输出是什么？(1 分)
4. `32 > '32'` 的输出是什么？(1 分)
5. `'Sheng Xin Bao Dian'.find('x')` 和`'Sheng Xin Bao Dian'.find('X')` 的输出分别是？(2 分)
6. 一句话计算`'Sheng Xin Bao Dian'` 字符串中 `n` 的数目？(1 分)
7. 写出下面 10 段程序的输出？(1 分/段)

```
aList = [1, 2, 3]    bList = aList    bList.append(4)    aList
```

```
aList = [1, 2, 3]    cList = aList[:]    cList.append(4)    aList
```

```
aList = [1,1,2,2,3,5,4,3]    aSet = set(aList)    aSet
```

```
[1, 2, 3] * 2
```

```
[i**2 for i in [1,2,3]]
```

```
dict([(i, i**2) for i in range(5)])
```

```
import re    re.findall("[Ii]mageGP", "www.ehbio.com/ImageGP")
```

```
' '.join(["Sheng", "Xin", "Bao", "Dian"])
```

```
def sumNumber(a, b):        return a + b    sumNumber(2,3)
```

```
def sumNumber(a, b):        return a + b    print(sumNumber(2,3))
```

10. 写程序以下面列表中每个元素为 `key`，元素出现的次数为 `value`，构建一个字典，并遍历字典按元素的 ASCII 码顺序输出？(5 分)

    ```
    aList = ['a', 'b', 'c', 'a', 'd','e', 'A']
    ```

11. 对教案中脑筋急转弯问题的解法进行优化; 问题是：现有 `100` 元钱，需要买 `100` 个物品，其中铅笔盒单价 `5` 元，笔单价 `3` 元，橡皮单价 `0.5` 元，怎么组合可以把 `100` 元花完，同时三种物品的个数和为 `100`，请用编程解决。(3 分)

12. 写程序用高斯的计算方式计算 `1+2+3+...+100` 的加和。(3 分)
13. 指出下面每个程序运行时可能会出现的错误。(1 分/段)

```
aList = [1,2,3]
aDict = {}
aDict[aList] = 1
b = aDict['a']

if 1:
print("Sheng xin bao dian great!")

32 + '32'

aList = [1, 2, 3]
aList.add(4)

aList = [1, 2, 3]
''.join(aList)

int('a')

3 / 0

for i in range(10)
print(L)

'Sheng Xin * 3

type = 1
```

14. Python 文件读写函数 `open` 的 `mode` 参数中 `r`, `w`, `a`, `t`, `b`, `x` 分别是什么意思？(3 分)
15. Python 中如何获取当前所在的工作目录? 如何修改工作目录？(3 分)
16. Python 中连接多个字符串的方法有哪些？优缺点是什么？(3 分)
17. `print("%.2f%%" % (1/3))` 的输出是什么？(2 分)
18. 描述下语句 `import pandas as pd` 做了什么操作？(2 分)
19. 教案中 IDmap 程序用 pandas 实现 (`GRCh38.idmap`,`ensm.id`)。(5 分)
20. Jupyter 中`%%writefile`, `%%run` 宏命令的用途是什么？(2 分)
21. 找出教案中 TP53 mRNA 序列中的 ORF (`human_TP53_mRNA.fa`)。(5 分)
22. 列出教案中大肠杆菌基因组中限制性内切酶 SecI 的切割位置 (`Ecoli.fa`)。(5 分)
23. 计算 `data/test1.fa` 中每条序列的 GC 含量。(5 分)
24. 不使用 pandas，写 Python 脚本处理 Pandas 教案中的 TPM 表达矩阵的提取和合并？(ENCFF060LPA.tsv, ENCFF262OBL.tsv, ENCFF289HGQ.tsv, ENCFF673KYR.tsv) (8 分)

25. 给定 FASTA 格式的文件 (test1.fa 和 test2.fa)，写一个程序 `cat.py` 读入文件，并输出到屏幕 (2 分)

    - 用到的知识点
      - open(file)
      - for .. in loop
      - print()
      - strip() function

26. 给定 FASTQ 格式的文件 (test1.fq), 写一个程序 `cat.py` 读入文件，并输出到屏幕 (2 分)

    - 用到的知识点
      - 同上

27. 写程序 `splitName.py`, 读入 test2.fa, 并取原始序列名字第一个空格前的名字为处理后的序列名字，输出到屏幕 (2 分)

    - 用到的知识点
      - split
      - 字符串的索引
    - 输出格式为：

```
>NM_001011874
gcggcggcgggcgagcgggcgctggagtaggagctg.......
```

28. 写程序 `formatFasta.py`, 读入 test2.fa，把每条 FASTA 序列连成一行然后输出 (2 分)

    - 用到的知识点
      - join
      - strip
    - 输出格式为:

```
>NM_001011874
gcggcggcgggc......TCCGCTG......GCGTTCACC......CGGGGTCCGGAG
```

29. 写程序 `formatFasta-2.py`, 读入 test2.fa，把每条 FASTA 序列分割成 80 个字母一行的序列 (2 分)

    - 用到的知识点
      - 字符串切片操作
      - range
    - 输出格式为

```
>NM_001011874
gcggcggcgc.(60个字母).TCCGCTGACG #(每行80个字母)
acgtgctacg.(60个字母).GCGTTCACCC
ACGTACGATG(最后一行可不足80个字母)
```

30. 写程序 `sortFasta.py`, 读入 test2.fa, 并取原始序列名字第一个空格前的名字为处理后的序列名字，排序后输出 (2 分)

- 用到的知识点
  - sort
  - dict
  - aDict[key] = []
  - aDict[key].append(value)

31. 提取给定名字的序列 (2 分)

- 写程序 `grepFasta.py`, 提取 fasta.name 中名字对应的 test2.fa 的序列，并输出到屏幕。
- 写程序 `grepFastq.py`, 提取 fastq.name 中名字对应的 test1.fq 的序列，并输出到文件。
  - 用到的知识点
    * print >>fh, or fh.write()
    * 取模运算，4 % 2 == 0

32. 写程序 `screenResult.py`, 筛选 test.expr 中 foldChange 大于 2 的基因并且 padj 小于 0.05 的基，可以输出整行或只输出基因名字。(4 分)

- 用到的知识点
  - 逻辑与操作符 and
  - 文件中读取的内容都为字符串，需要用 int 转换为整数，float 转换为浮点数

33. 写程序 `transferMultipleColumToMatrix.py` 将文件 (multipleColExpr.txt) 中基因在多个组织中的表达数据转换为矩阵形式，并绘制热图。(6 分)

- 用到的知识点
  - aDict[‘key’] = {}
  - aDict[‘key’][‘key2’] = value
  - if key not in aDict
  - aDict = {‘ENSG00000000003’: {“A-431”: 21.3, “A-549”, 32.5,…},“ENSG00000000003”:{},}
- 输入格式 (只需要前 3 列就可以)

```
Gene    Sample  Value   Unit    Abundance
ENSG00000000003 A-431   21.3    FPKM    Medium
ENSG00000000003 A-549   32.5    FPKM    Medium
ENSG00000000003 AN3-CA  38.2    FPKM    Medium
ENSG00000000003 BEWO    31.4    FPKM    Medium
ENSG00000000003 CACO-2  63.9    FPKM    High
ENSG00000000005 A-431   0.0     FPKM    Not detected
ENSG00000000005 A-549   0.0     FPKM    Not detected
ENSG00000000005 AN3-CA  0.0     FPKM    Not detected
ENSG00000000005 BEWO    0.0     FPKM    Not detected
ENSG00000000005 CACO-2  0.0     FPKM    Not detected
```

- 输出格式

```
Name    A-431   A-549   AN3-CA  BEWO    CACO-2
ENSG00000000460 25.2    14.2    10.6    24.4    14.2
ENSG00000000938 0.0 0.0 0.0 0.0 0.0
ENSG00000001084 19.1    155.1   24.4    12.6    23.5
ENSG00000000457 2.8 3.4 3.8 5.8 2.9
```

34. 写程序 `reverseComplementary.py` 计算序列 `ACGTACGTACGTCACGTCAGCTAGAC` 的反向互补序列。(2 分)

    - 用到的知识点
      - reverse
      - list(seq)

35. 写程序 `collapsemiRNAreads.py` 转换 smRNA-Seq 的测序数据。(5 分)

    - 输入文件格式 (mir.collapse, tab-分割的两列文件，第一列为序列，第二列为序列被测到的次数)

```
ID_REF        VALUE
ACTGCCCTAAGTGCTCCTTCTGGC        2
ATAAGGTGCATCTAGTGCAGATA        25
TGAGGTAGTAGTTTGTGCTGTTT        100
TCCTACGAGTTGCATGGATTC        4
```

    - 输出文件格式 (mir.collapse.fa, 名字的前 3 个字母为样品的特异标示，中间的数字表示第几条序列，是序列名字的唯一标示，第三部分是 x 加每个 reads 被测到的次数。三部分用下划线连起来作为 fasta 序列的名字。)

```
>ESB_1_x2
ACTGCCCTAAGTGCTCCTTCTGGC
>ESB_2_x25
ATAAGGTGCATCTAGTGCAGATA
>ESB_3_x100
TGAGGTAGTAGTTTGTGCTGTTT
>ESB_4_x4
TCCTACGAGTTGCATGGATTC
```

36. 简化的短序列匹配程序 (map.py) 把 short.fa 中的序列比对到 ref.fa, 输出短序列匹配到 ref.fa 文件中哪些序列的哪些位置。(10 分)

    - 用到的知识点
      - find
    - 输出格式 (输出格式为 bed 格式，第一列为匹配到的染色体，第二列和第三列为匹配到染色体序列的起始终止位置 (位置标记以 0 为起始，代表第一个位置；终止位置不包含在内，第一个例子中所示序列的位置是 (199,208](前闭后开，实际是 chr1 染色体第 199-206 的序列，0 起始). 第 4 列为短序列自身的序列.)。
    - 附加要求：可以只匹配到给定的模板链，也可以考虑匹配到模板链的互补链。这时第 5 列可以为短序列的名字，第六列为链的信息，匹配到模板链为'+'，匹配到互补链为'-'。注意匹配到互补链时起始位置也是从模板链的 5' 端算起的。

```
chr1    199 208 TGGCGTTCA
chr1    207 216 ACCCCGCTG
chr2    63  70  AAATTGC
chr3    0   7   AATAAAT
```

37. 备注：

    - 每个提到提到的 "用到的知识点" 为相对于前面的题目新增的知识点，请综合考虑。此外，对于不同的思路并不是所有提到的知识点都会用着，而且也可能会用到未提到的知识点。但是所有知识点都在前面的讲义部分有介绍。
    - 每个程序对于你身边会写的人来说都很简单，因此你一定要克制住，独立去把答案做出，多看错误提示，多比对程序输出结果和预期结果的差异。

- 学习锻炼“读程序”，即对着文件模拟整个的读入、处理过程来发现可能的逻辑问题。
- 程序运行没有错误不代表你写的程序完成了你的需求，你要去查验输出结果是不是你想要的。

38. 关于程序调试

- 在初写程序时，可能会出现各种各样的错误，常见的有缩进不一致，变量名字拼写错误，丢失冒号，文件名未加引号等，这时要根据错误提示查看错误类型是什么，出错的是哪一行来定位错误。当然，有的时候报错的行自身不一定有错，可能是其前面或后面的行出现了错误。
- 用脑袋运行程序：当程序写作完成后，自己尝试对着数据文件，一行一行的执行程序，来看程序的运行是否与自己想干的活一致，有没有纰漏。
- 当结果不符合预期时，要学会使用 **print** 来查看每步的操作是否正确，比如我读入了字典，我就打印下字典，看看读入的是不是我想要的，是否含有不该存在的字符；或者在每个判断句、函数调入的情况下打印个字符，来跟踪程序的运行轨迹

39. 如果视频或文档中没有答案的或者有错误的，请提交题目和自己的答案到 http://www.ehbio.com/Esx 提问寻求解答。

```
import re
re.findall("[Ii]mageGP", "www.ehbio.com/ImageGP")
```

```
['ImageGP']
```

```
' '.join(["Sheng", "Xin", "Bao", "Dian"])
```

```
'Sheng Xin Bao Dian'
```

```
# 写程序以下面列表中每个元素为 key，元素出现的次数为 value，
# 构建一个字典，并遍历字典按元素的 ASCII 码顺序输出？
aList = ['a', 'b', 'c', 'a', 'd','e', 'A']
aDict = {}
uniqList = list(set(aList))
aDict = dict([(i,aList.count(i)) for i in uniqList])
uniqList.sort()

for key in uniqList:
    print(key, aDict[key], sep="-->")
#for item in uniqList:
#    aDict[item] = aDict.get(item,0) + 1
```

```
A-->1
a-->2
b-->1
c-->1
d-->1
e-->1
```

```
# 写程序以下面列表中每个元素为 key，元素出现的次数为 value，
# 构建一个字典，并遍历字典按元素的 ASCII 码顺序输出？
aList = ['a', 'b', 'c', 'a', 'd','e', 'A']
aDict = {}
for item in aList:
    aDict[item] = aDict.get(item,0) + 1

keyL = list(aDict.keys())
keyL.sort()

for key in keyL:
    print(key, aDict[key], sep="\t")
```

```
A   1
a   2
b   1
c   1
d   1
e   1
```

```
aList = ['a', 'b', 'c', 'a', 'd', 'e', 'A']
aDict = {}
for i in aList:
    aDict.setdefault(i, 0)
    aDict[i] += 1
print(aDict)
```

```
{'a': 2, 'b': 1, 'c': 1, 'd': 1, 'e': 1, 'A': 1}
```

```
aDict = {}
aList = ['a', 'b', 'c', 'a', 'd','e', 'A']
#uniL = list(set(aList))
for i in aList:
    aDict[i] = aDict.get(i,0)+1
    #aDict += 1
    #break
print(aDict)
```

```
{'a': 2, 'b': 1, 'c': 1, 'd': 1, 'e': 1, 'A': 1}
```

```
aList = ['a', 'b', 'c', 'a', 'd','e', 'A']
aSet = set(aList)
aSet
```

```
for i in aSet:
    print (i,aList.count(i))
```

```
a 2
d 1
A 1
b 1
e 1
c 1
```

```
list = ['a', 'b', 'c', 'a', 'd','e', 'A']
dic = {}
for i in list:
    dic[i] = list.count(i)
print(dic)
```

```
# 写程序用高斯的计算方式计算 1+2+3+...+100 的加和
n = 100
(1+n) * (n/2)
```

```
5050.0
```

```
n = 99
(1+n) * (n/2)
```

```
4950.0
```

```
import pandas as pd
idmap = pd.read_table("data/GRCh38.idmap",
                      header=0, index_col=0)
ensmL = [line.strip() for line in open("data/ensm.id")]
idmap = idmap[idmap.index.isin(ensmL)]
idmap["Associated Gene Name"]
```

```
Gene ID
ENSG00000252303          RNU6-280P
ENSG00000281771              Y_RNA
ENSG00000281256       RP11-222G7.2
ENSG00000283272    Clostridiales-1
ENSG00000280864      RP11-654C22.2
ENSG00000280792      RP11-315F22.1
ENSG00000282878       RP11-399E6.1
```

```
ENSG00000283276     ABBA01000934.1
ENSG00000281822           RNU1-62P
ENSG00000281384         AC093802.1
ENSG00000280505       RP11-654C22.1
ENSG00000281764        RP11-399E6.2
ENSG00000281316             DPPA2P2
ENSG00000280963         SERTAD4-AS1
ENSG00000280775           RNA5SP136
ENSG00000281876        RP11-399E6.4
ENSG00000281766                RYBP
ENSG00000281518              FOXO6
ENSG00000281614             INPP5D
ENSG00000280584              OBP2B
ENSG00000281230            SERTAD4
ENSG00000281917            SLC16A1
Name: Associated Gene Name, dtype: object
```

```
mRNA = []
for line in open("data/human_TP53_mRNA.fa"):
    if line[0] == ">":
        continue
    else:
        mRNA.append(line.strip())
#--------------------------------
mRNA = ''.join(mRNA)
orf = re.compile("(A[TU]G(...){99,}?([TU]AA|[TU]AG|[TU]GA))")
for i in orf.findall(mRNA):
    print(i[0])
```

```
ATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTCTAGAGCCACCGTCCAGGGAGCAGGTA
ATGGGGAGTAGGACATACCAGCTTAGATTTTAAGGTTTTTACTGTGAGGGATGTTTGGGAGATGTAAGAAATGTTCTTGCAGTTAAGGGTTAGTT
ATGATGATCTGGATCCACCAAGACTTGTTTTATGCTCAGGGTCAATTTCTTTTTTCTTTTTTTTTTTTTTTTCTTTTTCTTTGAGACTGGGTC
```

```
#SecI 识别 CCNNGG
ecoli = []
for line in open("data/Ecoli.fa"):
    if line[0] != '>':
        ecoli.append(line.strip())
    #print(line.strip())
#---------------
ecoli = ''.join(ecoli)

secI = re.compile("CC..GG")
for i in secI.finditer(ecoli):
    print(i.start())
```

```
174
299
556
```

```
lineL = []
for line in open("data/Ecoli.fa"):
    line = line.strip()
    if line[0] == ">":
        continue
    else:
        lineL.append(line)
#---------------------------
seq = ''.join(lineL)

seci = re.compile("CC[ATGC]GG")
print(seci)
```

```
re.compile('CC[ATGC]GG')
```

```
help(re.finditer)
```

```
Help on function finditer in module re:

finditer(pattern, string, flags=0)
    Return an iterator over all non-overlapping matches in the
    string.  For each match, the iterator returns a match object.

    Empty matches are included in the result.
```

```
gc_cnt = 0
seqLen = 0
for line in open("data/test1.fa"):
    if line[0] == '>':
        if seqLen:
            gc_percent = gc_cnt / seqLen * 100
            print(key, round(gc_percent,2),"%")
            gc_cnt = seqLen = 0
        key = line.split()[0][1:]
    else:
        line = line.strip().upper()
        gc_cnt += line.count('C') + line.count('G')
        seqLen += len(line)
#-------------------------------
if seqLen:
```

```
    gc_percent = gc_cnt / seqLen * 100
    print(key, round(gc_percent,2),"%")
```

```
NM_001011874 81.43 %
NM_001195662 42.86 %
NM_0112835 37.14 %
NM_011283 37.14 %
```

```
for line in open("data/test1.fa"):
    if line[0] == ">":
        continue
    else:
        line = line.strip().upper()
        gc_cnt = line.count("G") + line.count("C")
        total_len = len(line)
        gc_per = gc_cnt / total_len * 100
        print(round(gc_per,2))
```

```
81.43
42.86
37.14
37.14
```

```
count = 5
for line in open("data/ENCFF060LPA.tsv"):
    if count == 0:
        break
    print(line)
    count -= 1
```

```
gene_id transcript_id(s)    length  effective_length    expected_count  TPM FPKM

ENSG00000000003.14  ENST00000373020.8,ENST00000494424.1,ENST00000496771.5,ENST00000612152.4,ENS

ENSG00000000005.5   ENST00000373031.4,ENST00000485971.1 940.50  720.47  0.00    0.00    0.00

ENSG00000000419.12  ENST00000371582.8,ENST00000371584.8,ENST00000371588.9,ENST00000413082.1,ENS

ENSG00000000457.13  ENST00000367770.5,ENST00000367771.10,ENST00000367772.8,ENST00000423670.1,EN
```

```
fileL = ["data/ENCFF060LPA.tsv", "data/ENCFF262OBL.tsv",
         "data/ENCFF289HGQ.tsv", "data/ENCFF673KYR.tsv"]
'''
```

```
aDict = {
    'ENCFF060LPA': {'a': 5, 'b': 3},
    'ENCFF289HGQ': {'a': 3, 'c': 5},
}

aDict = {
    'a': {'ENCFF060LPA': 5, 'ENCFF289HGQ':3},
    'b': {'ENCFF060LPA': 3},
    'c': {'ENCFF289HGQ': 5}
}
'''
aDict = {}
labelL = []

for file_name in fileL:
    label = file_name.replace('data/','')
    label = label.replace('.tsv','')
    labelL.append(label)
    #print(label)
    header = 1
    count = 3
    #print(label)
    for line in open(file_name):
        if header == 1:
            header -= 1
            continue
        #--------------------
        lineL = line.strip().split('\t')
        gene  = lineL[0]
        FPKM  = lineL[-1]
        aDict.setdefault(gene, {})
        aDict[gene][label] = FPKM
        #if count:
        #    print("\t",aDict)
        #    count -= 1
#-------------------------------------
print("GeneName", '\t'.join(labelL), sep="\t")

count = 5
for gene,exprD in list(aDict.items()):
    exprL = [exprD.get(label, '0') for label in labelL]
    exprL = [gene] + exprL
    #if count:
    print('\t'.join(exprL))
    #    count -= 1
```

```
GeneName    ENCFF060LPA ENCFF262OBL ENCFF289HGQ ENCFF673KYR
ENSG00000000003.14  18.24   26.74   10.84   15.07
ENSG00000000005.5   0.00    0.00    0.00    0.00
ENSG00000000419.12  27.19   29.44   15.23   11.17
ENSG00000000457.13  2.37    3.87    2.53    2.37
ENSG00000000460.16  4.14    8.37    2.70    0.98
```

```
for line in open("data/test2.fa"):
    if line[0] == '>':
```

```
>NM_001011874 gene=Xkr4 CDS=151-2091
gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccggggaaggaagccagggcg
aggcgaggaggtggcgggaggaggagacagcagggacaggTGTCAGATAAAGGAGTGCTCTCCTCCGCTG
CCGAGGCATCATGGCCGCTAAGTCAGACGGGAGGCTGAAGATGAAGAAGAGCAGCGACGTGGCGTTCACC
CCGCTGCAGAACTCGGACAATTCGGGCTCTGTGCAAGGACTGGCTCCAGGCTTGCCGTCGGGGTCCGGAG
>NM_001195662 gene=Rp1 CDS=55-909
AAGCTCAGCCTTTGCTCAGATTCTCCTCTTGATGAAACAAAGGGATTTCTGCACATGCTTGAGAAATTGC
AGGTCTCACCCAAAATGAGTGACACACCTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCA
AGTTCCTTCCCCTCGCCATTCAAATATCACTCATCCTGTAGTGGCTAAACGCATCAGTTTCTATAAGAGT
GGAGACCCACAGTTTGGCGGCGTTCGGGTGGTGGTCAACCCTCGTTCCTTTAAGACTTTTGACGCTCTGC
TGGACAGTTTATCCAGGAAGGTACCCCTGCCCTTTGGGGTAAGGAACATCAGCACGCCCCGTGGACGACA
CAGCATCACCAGGCTGGAGGAGCTAGAGGACGGCAAGTCTTATGTGTGCTCCCACAATAAGAAGGTGCTG
>NM_011283 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCAC
ACAAACTATTTACTCTTTTCTTCGTAAGGAAAGGTTCAACTTCTGGTCTCACCCAAAATGAGTGACACAC
CTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCAAGTTCCTTCCCCTCGCCATTCAAATAT
CACTCATCCTGTAGTGGCTAAACGCATCAGTTTCTATAAGAGTGGAGACCCACAGTTTGGCGGCGTTCGG
GTGGTGGTCAACCCTCGTTCCTTTAAGACTTTTGACGCTCTGCTGGACAGTTTATCCAGGAAGGTACCCC
TGCCCTTTGGGGTAAGGAACATCAGCACGCCCCGTGGACGACACAGCATCACCAGGCTGGAGGAGCTAGA
GGACGGCAAGTCTTATGTGTGCTCCCACAATAAGAAGGTGCTGCCAGTTGACCTGGACAAGGCCCGCAGG
CGCCCTCGGCCCTGGCTGAGTAGTCGCTCCATAAGCACGCATGTGCAGCTCTGTCCTGCAACTGCCAATA
TGTCCACCATGGCACCTGGCATGCTCCGTGCCCCAAGGAGGCTCGTGGTCTTCCGGAATGGTGACCCGAA
>NM_0112835 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCAC
ACAAACTATTTACTCTTTTCTTCGTAAGGAAAGGTTCAACTTCTGGTCTCACCCAAAATGAGTGACACAC
CTTCTACTAGTTTCTCCATGATTCATCTGACTTCTGAAGGTCAAGTTCCTTCCCCTCGCCATTCAAATAT
CACTCATCCTGTAGTGGCTAAACGCATCAGTTTCTATAAGAGTGGAGACCCACAGTTTGGCGGCGTTCGG
GTGGTGGTCAACCCTCGTTCCTTTAAGACTTTTGACGCTCTGCTGGACAGTTTATCCAGGAAGGTACCCC
TGCCCTTTGGGGTAAGGAACATCAGCACGCCCCGTGGACGACACAGCATCACCAGGCTGGAGGAGCTAGA
GGACGGCAAGTCTTATGTGTGCTCCCACAATAAGAAGGTGCTGCCAGTTGACCTGGACAAGGCCCGCAGG
CGCCCTCGGCCCTGGCTGAGTAGTCGCTCCATAAGCACGCATGTGCAGCTCTGTCCTGCAACTGCCAATA
TGTCCACCATGGCACCTGGCATGCTCCGTGCCCCAAGGAGGCTCGTGGTCTTCCGGAATGGTGACCCGAA
```

8 生信教程文章集锦

8.1 生信宝典

生信的作用越来越大，想学的人越来越多，不管是为了以后发展，还是为了解决眼下的问题。但生信学习不是一朝一夕就可以完成的事情，也许你可以很短时间学会一个交互式软件的操作，却不能看完程序教学视频后就直接写程序。也许你可以跟着一个测序分析流程完成操作，但不懂得背后的原理，不知道什么参数需要修改，结果可以出来，却把我不住对还是错。

学习生信从来就不是一个简单的事，需要做好持久战的心理准备。

在学习时，我们都希望由浅入深的逐步深入，不断地练习和实践，这就是为什么我们需要一本书，因为书很系统。但生信发展的历史短于计算机编程的历史，如果想要一门程序设计的入门数据，每种语言都可以找到几本。但想要一个囊括生信的书，就有些难了。本身生信跨领域，需要多学科的知识，而其内部又有不少分子，都囊括了太大，包括的少又有些隔靴搔痒的感觉。

我们当时都是零基础下自学 Linux, 自学 Python，自学 R，自学高通量测序；这些学习经历，之前都零星地记录在博客里。现在回头去看几年前自己记录的东西，觉得好简单，而当时却费了很大的力气。这些零星的随手记，当时也只是为了自己看，到现在确实只有自己能看得懂，不便惠及更多的人。

因此我们创建了生信宝典，希望从不同的角度传播知识。这个不同有三点含义，一是形式上的不同，摒弃之前主编们单人作战想写啥就写啥，而是有组织有计划的内容聚合，提供一系列的教程，由入门到提高。二是内容的不同，不去用网上现有教程的通用数据做例子，而是拿实际生物数据，讲述如何解释生信中普遍碰到的问题，讲述如何处理自己的数据。三是立足点不同。在写作时，我们回到了当年，在回忆中用整个阶段的学习去指导当初的那个小白，从那些会了的人觉得微不足道而不会的人又迈不过的坎入手，直击痛点。知识点的收录依据不是是否炫酷，是否难，而是是否必要。如果必要，再简单，也要提及；如果不必要，再炫酷，也暂不纳入。

通过大量的生信例子、关键的注释和浓缩的语句形成下面的一系列学习教程。每一篇内容都不多，可以当做小说阅读，也可以跟着去练，反复几遍，每读一次都会有不同的收获和体会。

8.1.1 系列教程

- 生物信息之程序
- 如何优雅的提问
- 生信宝典视频教程
- 好色之旅-画图三字经
- 转录组分析的正确姿势
- 生信的系列教程
- 生信的系列书籍
- 文章用图的修改和排版 (1)

- 文章用图的修改和排版 (2)
- 简单强大的在线绘图
- 简单强大的在线绘图-升级版
- 论文图表基本规范
- 学术图表的基本配色方法
- 英语写作常见错误总结和学习视频
- 教育部推出首批 490 门 “国家精品在线开放课程

8.1.2 NGS 分析工具评估

- 39 个转录组分析工具，120 种组合评估 (转录组分析工具哪家强-导读版)
- 39 个转录组分析工具，120 种组合评估 (转录组分析工具大比拼（完整翻译版）)
- 无参转录组分析工具评估和流程展示

8.1.3 宏基因组教程

- 微生物组入门必读 + 宏基因组实操课程
- 扩增子图表解读-理解文章思路
- 扩增子分析流程-把握分析细节
- 扩增子统计绘图-冲击高分文章
- 宏基因组分析教程
- 4500 元的微生物组培训资料

8.1.4 系列宣传

- 转录组分析的正确姿势
- 120 分的转录组考题，你能得多少
- 生物信息作图系列 R、Cytoscape 及图形排版和 Python 编程培训研讨班开课了
- 维密摔倒不可怕，关键时有人搀一把，坚持走下去
- 生物信息作图系列 - R、网络图及文章图形排版
- 易生信转录组培训总结和优惠分享
- 生物信息 9 天速成班 — 你也可以成为团队不可或缺的人
- Python 没有捷径，但可以加速，零基础九天你也可以会编程
- 小学生都学 Python 了，你还不知道怎么开始-资源帖
- 一个月学会 Python 的 Quora 指南和资料放送
- 扩增子分析基本流程和结果解读
- 微生物组——扩增子分析专题实战开课啦
- 如何入门生信 Linux
- 3 分和 30 分文章差距在哪里？

8.1.5 生信生物知识

- 生物研究中不可缺少的数字概念，多少，多大，多快

8.1.6 文献精读

- CRISPR-CAS9 发展历程小记
- 一场大病引起的诺贝尔 2017 年生理学奖角逐
- Science 搞反狗脑 - 人脑和狗脑一样？
- 一篇压根不存在的文献被引用 400 次 ?! 揭开” 幽灵文献” 的真面目
- 基于人工智能的文献检索，导师查找，更聪明
- GeenMedical：文献查询、筛选、引用排序、相似文献、全文下载、杂志分区、影响因子、结果导出、杂志评述、直接投稿，一站服务
- YANDEX 搜索，不翻墙稳定使用近谷歌搜索
- Nature 我的研究对后人毫无用途：21% 的学术论文自发布后从未被引用
- SCI-HUB 镜像, SSH 隧道访问学校内网
- 为了速成生物学，一位程序员探索了 “爆款” 基因背后的秘密

8.1.7 Linux

- Linux-总目录
- Linux-文件和目录
- Linux-文件操作
- Linux 文件内容操作
- Linux-环境变量和可执行属性
- Linux - 管道、标准输入输出
- Linux - 命令运行监测和软件安装
- Linux-常见错误和快捷操作
- Linux-文件列太多，很难识别想要的信息在哪列；别焦急，看这里。
- Linux-文件排序和 FASTA 文件操作
- Linux-应用 Docker 安装软件
- Linux 服务器数据定期同步和备份方式
- VIM 的强大文本处理方法
- Linux - Conda 软件安装方法
- 查看服务器配置信息
- Linux - SED 操作，awk 的姊妹篇
- Linux - 常用和不太常用的实用 awk 命令
- Bash 概论 - Linux 系列教程补充篇

8.1.8 CIRCOS 系列

- CIRCOS 圈图绘制 - circos 安装
- CIRCOS 圈图绘制 - 最简单绘图和解释
- CIRCOS 圈图绘制 - 染色体信息展示和调整
- CIRCOS 增加热图、点图、线图和区块属性

8.1.9 R 统计和作图

- 在 R 中赞扬下努力工作的你，奖励一份 CheatShet
- 别人的电子书，你的电子书，都在 bookdown
- R 语言 - 入门环境 Rstudio
- R 语言 - 热图绘制 (heatmap)
- R 语言 - 基础概念和矩阵操作
- R 语言 - 热图简化
- R 语言 - 热图美化
- R 语言 - 线图绘制
- R 语言 - 线图一步法
- R 语言 - 箱线图（小提琴图、抖动图、区域散点图）
- R 语言 - 箱线图一步法
- R 语言 - 火山图
- R 语言 - 富集分析泡泡图（文末有彩蛋）
- R 语言 - 散点图绘制
- 一文看懂 PCA 主成分分析
- 富集分析 DotPlot，可以服
- R 语言 - 韦恩图
- R 语言 - 柱状图
- R 语言 - 图形设置中英字体
- R 语言 - 非参数法生存分析
- 基因共表达聚类分析和可视化
- R 中 1010 个热图绘制方法
- 还在用 PCA 降维？快学学大牛最爱的 t-SNE 算法吧, 附 Python/R 代码
- 一个函数抓取代谢组学权威数据库 HMDB 的所有表格数据
- 文章用图的修改和排版
- network3D: 交互式桑基图
- network3D 交互式网络生成

8.1.10 扩增子三步曲

- 1 图表解读-理解文章思路

- 2 分析流程-把握分析细节

- 扩展 1：视频教程-夯实分析思路
- 扩展 2：QIIME2 教程-了解分析趋势
- 3 统计绘图-冲击高分文章

8.1.11 宏基因组分析专题

- 1 背景知识-Shell 入门与本地 blast 实战
- 2 数据质控 fastqc, Trimmomatic, MultiQC, khmer
- 3 组装拼接 MEGAHIT 和评估 quast
- 4 基因注释 Prokka
- 5 基于 Kmer 比较数据集 sourmash
- 6 不比对快速估计基因丰度 Salmon
- 7bwa 序列比对, samtools 查看, bedtools 丰度统计
- 8 分箱宏基因组 binning, MaxBin, MetaBin, VizBin
- 9 组装 assembly 和分箱 bin 结果可视化—Anvio
- 10 绘制圈图-Circos 安装与使用
- MetaPhlAn2 分析有参宏基因组

8.1.12 NGS 基础

- NGS 基础 - FASTQ 格式解释和质量评估
- NGS 基础 - 高通量测序原理
- NGS 基础 - 参考基因组和基因注释文件
- NGS 基础 - GTF/GFF 文件格式解读和转换
- 本地安装 UCSC 基因组浏览器
- 测序数据可视化 (一)
- 测序文章数据上传找哪里
- GO、GSEA 富集分析一网打进
- GSEA 富集分析 - 界面操作
- 去东方，最好用的在线 GO 富集分析工具
- 生信软件系列 - NCBI 使用

8.1.13 癌症数据库

- UCSC XENA - 集大成者 (TCGA, ICGC)
- ICGC 数据库使用
- TCGA 数据库在线使用

8.1.14 Python

- Python 学习 - 可视化变量赋值、循环、程序运行过程
- Python 极简教程（一）
- Python 教程（二）
- Python 教程（三）
- Python 教程（四）
- Python 教程（五）
- Python 教程（六）
- Pandas，让 Python 像 R 一样处理数据，但快
- Python 解析 psiBlast 输出的 JSON 文件结果
- 为啥我的 Python 这么慢 - 项查找 (二)
- 为啥我的 Python 这么慢 (一)
- Python 资源
- 关于 Python 中的 __main__ 和编程模板

8.1.15 NGS 软件

- Rfam 12.0+ 本地使用（最新版教程）
- 轻松绘制各种 Venn 图
- ETE 构建、绘制进化树
- psRobot：植物小 RNA 分析系统
- 生信软件系列 - NCBI 使用
- 去东方，最好用的在线 GO 富集分析工具

8.1.16 Cytoscape 网络图

- Cytoscape 教程 1
- Cytoscape 之操作界面介绍
- 新出炉的 Cytoscape 视频教程

8.1.17 分子对接

- 来一场蛋白和小分子的风花雪月
- 不是原配也可以-对接非原生配体
- 简单可视化-送你一双发现美的眼睛
- 你需要知道的那些前奏

8.1.18 生信宝典之傻瓜式

- 生信宝典之傻瓜式 (一) 如何提取指定位置的基因组序列
- 生信宝典之傻瓜式 (二) 如何快速查找指定基因的调控网络
- 生信宝典之傻瓜式 (三) 我的基因在哪里发光 - 如何查找基因在发表研究中的表达
- 生信宝典之傻瓜式 (四) 蛋白蛋白互作网络在线搜索
- 生信宝典之傻瓜式 (五) 文献挖掘查找指定基因调控网络

8.1.19 生信人写程序

- 生信人写程序 1. Perl 语言模板及配置
- 生信人写程序 2. Editplus 添加 Perl, Shell, R, markdown 模板和语法高亮

8.1.20 小技巧系列

- 参考文献中杂志名字格式混乱问题一次解决

8.1.21 招聘

- 易汉博欢迎您加入

8.2 宏基因组

http://mp.weixin.qq.com/s/5jQspEvH5_4Xmart22gjMA

宏基因组/微生物组是当今世界科研最热门的研究领域之一，为加强本领域的技术交流与传播，推动中国微生物组计划发展，中科院青年科研人员创立 “宏基因组” 公众号，目标为打造本领域纯干货技术及思想交流平台。

本公众号每日推送，工作日分享宏基因组领域科研思路、实验和分析技术，理论过硬实战强；周末科普和生活专栏，轻松读文看片涨姿势。目前经过近半年发展，分享过百篇原创文章，已有 14000+ 小伙伴在这里一起交流学习，感兴趣的赶快关注吧。

8.2.1 精选文章推荐

5000+

- 微生物组入门必读 + 宏基因组实操课程
- 你想要的生信知识全在这—生信宝典
- 生物信息 9 天速成班—成为团队不可或缺的人
- 3 分和 30 分文章差距在哪里？
- 肠道菌群在人体中的作用
- 看完此片我想把身上的细菌寄生虫供起来
- 岛国科普—生命大跃进
- 我们的未来在哪里？
- 论文图表基本规范
- DNA 提取发 Nature
- 实验 vs 数据分析，谁对结果影响大?

3000+

- 扩增子图表解读-理解文章思路
- 扩增子分析流程-把握分析细节
- 扩增子统计绘图-冲击高分文章
- 宏基因组分析教程
- 4500 元的微生物组培训资料
- Co-occurrence 网络图在 R 中的实现
- 学术图表的基本配色方法
- 一文读懂进化树
- 自学生信-biostar handbook
- 漱口水增加糖尿病，高血压风险

1000+

- 微生物组——扩增子分析专题培训开课啦！！！
- 最简单漂亮的免费在线生信绘图工具
- 小学生都学 Python 了，你还不知道怎么开始
- 五彩进化树与热图更配-ggtree 美颜进化树
- 扩增子分析还聚 OTU 就真 OUT 了
- 主流非聚类方法 dada2,deblur 和 unoise3 介绍与比较
- 16S 预测微生物群落功能 0 概述 1KO 通路 PICRUSt 2 元素循环 FAPROTAX 3 表型 bugbase 4KO 通路 Tax4Fun

- 一文读懂微生物组
- 2017 年发展简史和十大热文盘点

8.2.2 培训、会议、征稿、招聘

- 3 月 10-19 日，北京，微生物组——扩增子分析专题培训
- 5 月 11-13 日，北京，中国肠道大会

8.2.3 科研经验

- 如何优雅的提问
- 公众号搜索方法大全
- 科研团队成长三部曲：1 云笔记 2 云协作 3 公众号
- 文献阅读 1 热心肠 2SemanticScholar 3geenmedical
- 生信编程模板 Perl Shell R
- 生物信息之程序学习

- Endnote X8 云同步：有网随时读文献
- 论文 Figures，你不能不知道的秘密
- 转录组分析的正确姿势
- 整个世界都是你的已知条件

8.2.4 软件和数据库使用

- SILVAngs:16S/18S 在线分析1 2
- METAGENassist 帮你搞定宏基因分析的所有图形需求
- Windows 不用虚拟机或双系统，轻松实现 linux shell 环境：gitforwindows
- 一条命令轻松绘制 CNS 顶级配图-ggpubr
- ggbiplot-最好看的 PCA 图
- LDA 分析、作图及添加置信-ggord
- ggrepel-解决散点图样品标签重叠，方便筛选样品
- Alpha 多样性稀释曲线 rarefraction curve
- 微生物组间差异分析神器-STAMP

- 扩增子分析神器 USEARCH
- 微生物扩增子数据库大全

- antiSMASH：微生物次生代谢物基因簇预测

- 微生物网络构建：MENA, LSA, SparCC 和 CoNet

- Cytoscape: MCODE 增强包的网络模块化分析
- FUNGuild：真菌功能注释
- 在线 RaxML 构建系统发育树
- Genevestigator: 查找基因在发表研究中的表达

- psRobot：植物小 RNA 分析系统
- RepeatMasker：基因组重复序列注释

8.2.5 扩增子学习三步曲

8.2.5.1 1 图表解读-理解文章思路

8.2.5.2 2 分析流程-把握分析细节

- 扩展 1：视频教程-夯实分析思路
- 扩展 2：QIIME2 教程-了解分析趋势

8.2.5.3 3 统计绘图-冲击高分文章

8.2.6 宏基因组分析专题

- 1 背景知识-Shell 入门与本地 blast 实战
- 2 数据质控 fastqc, Trimmomatic, MultiQC, khmer
- 3 组装拼接 MEGAHIT 和评估 quast
- 4 基因注释 Prokka
- 5 基于 Kmer 比较数据集 sourmash
- 6 不比对快速估计基因丰度 Salmon
- 7bwa 序列比对, samtools 查看, bedtools 丰度统计
- 8 分箱宏基因组 binning, MaxBin, MetaBin, VizBin
- 9 组装 assembly 和分箱 bin 结果可视化—Anvi'o
- 10 绘制圈图-Circos 安装与使用
- MetaPhlAn2 分析有参宏基因组

8.2.7 R 统计绘图

- 视频教程：R 语言 recharts 包绘制交互式图形
- R 语言聚类分析–cluster, factoextra
- 堆叠柱状图各成分连线画法：突出展示组间物种丰度变化
- R 相关矩阵可视化包 ggcorrplot

8.2.8 实验设计与技术

- 微生物样本取样及微生物基因组 DNA 提取建议
- 样品生物学重复数据选择 1 必要性 2 需要多少重复？
- 样品命名 注意事项
- 扩增子引物选择 16S 结构 16S 单 V4 区是最佳选择?
- 海洋可培养微生物的鉴定与分类
- 怎么取粪便样品
- Rhizosphere、Rhizoplane 根际土如何取
- Nat. Biotechnol. 扩增子测序革命—用 16S 及 18S rRNA 全长进行微生物多样性研究

8.2.9 基础知识

- Microbiota, metagenome, microbiome 区别
- 16S 测序，不知道 OTU 你就 out 了！
- 计量宏基因组学数据分析的方法及进展
- 排序方法比较大全 PCA、PCoA、NMDS、CCA
- LEfSe 分析，你真的懂了么
- 宏基因组基础知识梳理
- 扩增子 SCI 套路1 微群落结构差异 2 组间差异分析 3 系统总结
- 环境因子关联分析——我应该选择 CCA 还是 RDA 分析？
- “P 值” 背后那些不可不知的事儿
- Adonis 和 ANOSIM 方法组间整体差异评估原理
- 轻松看懂机器学习十大常用算法
- 一文读懂 “随机森林” 在微生态中的应用
- 你想知道的 “ROC 曲线”
- 人体对微生物的管控
- 简单读懂微生物基因组的泛基因组学

8.2.10 必读综述

- Nature：宏基因组关联分析
- Nature：肠道菌群如何划分肠型
- Nature: 来自细菌的通告——群感效应简介
- Nature：呼吸道菌群—呼吸道健康的守门人
- Nature: 拥抱未知-解析土壤微生物组的复杂性
- Cell：代谢控制中的脑肠轴
- 研究微生物，只靠多组学根本不够

- 中国微生物组计划—农作物微生物组

- Annu Rev：植物微生物组—系统见解与展望

- Annual Reviews| 微生物组与人
- 微生物组学与植物病害微生物防治

- 组学重建真菌现有分类系统

- 微生物应用 | 农业废弃物资源化利用

- 宏基因组学入门1 初识 2 进一步 3 拼接

- 肠道微生物与人类密切相关的方方面面
- Nature: 测序技术的前世今生
- Nature Reviews：全新的益生元定义和范围
- 原核转录组非编码 RNA 研究

8.2.11 高分文章套路解读

- Nature: 培养组学—高通量细菌分离培养鉴定
- SR: 真菌培养组学同揭示人类肠道真菌群落结构
- Nature: 地球微生物组计划首发成果—揭示地球多尺度微生物多样性
- Nature：如何做一篇肠道菌群免疫的顶级文章
- Nat Biotech: 宏表观组—DNA 甲基化辅助宏基因组 binning
- Nature: 甘露糖苷选择性抑制致病性大肠杆菌

- Nature: 拟南芥根微生物组的结构和组成

- Nature: 地球上最古老的热液喷口发现早期生命迹象
- Nature Method: 宏基因组软件评估—人工重组宏基因组基准数据集

- Nature Genetics：微生物基因组如何适应植物？(news & views)
- NC：降低微生物群落复杂度突破组装难题

- NC：自体免疫水泡皮肤病中鉴定基因与微生物组互作

- GigaScience: 植物 MWAS 研究—谷子产量与微生物组关联分析

- Microbiome：微生物组研究中你必须注意的细节

- Microbiome：HiSeq 平台 16S 扩增子文库构建方法

- Microbiome: 简单套路发高分文章–杨树微生物组

- Microbiome：肠道菌群失衡促进高血压

- Microbiome：重新定义 “卫生” 概念

- ME：网络分析揭示微生物群落应对环境扰动的稳定性机制
- SR: 土壤细菌定量方法结合相对丰度分析揭示种群的真实变化

8.2.12 科普视频-寓教于乐

- BBC 人体奥秘之细胞的暗战
- BBC 人体奥秘 Inside.the.Human.Body
- NG 人体内旅行 Inside.the.Living.Body
- NG 子宫日记 Womb

- NHK: 再造人类生命的神奇细胞
- CCTV9 让尸体说话-法医密档
- 豆瓣 8.9，惹哭亿万中国人的纪录片-本草中华
- 2 分钟视频回顾植物学家钟扬的贡献
- 一顿“寄生虫大餐”，或能治好干净引来的免疫病
- 只要 11 天，浓度 1000 倍的抗生素也无效——超级细菌
- 致命病毒为何疯狂袭击人类？都怪我们那群会飞的远房亲戚
- 土豆上的小霉菌引发百万人死亡和逃难，却造就全球 7 千万后裔
- 看完这些能控制大脑的寄生虫，你会怀疑人类！
- 梅毒狂想曲

8.2.13 友军文章汇总推荐

- 学习生信的系列教程——纯生信一作发 IF>20 的大神

9 公司简介

易汉博基因科技(北京)有限公司

公司简介

易汉博基因科技（北京）有限公司由中国科学院及武汉大学等高校的优秀博士、硕士毕业生创建，致力于利用最前沿与最精确的生物信息分析技术，帮助解析基础研究和临床应用相关的高通量数据的专业数据服务公司。

易汉博技术团队在生物信息领域具有很高的造诣，团队成员以科学问题为导向，依托精湛的生物信息分析技术和丰富的实验生物学设计经验，在攻读学位或博士后进修期间获得了多个创新性研究成果，并发表于 *Cell*，*PNAS*，*JBC*，*NAR*，*New Phytologist*，*Bioinformatics* 等杂志，其中创始人陈博士发现并协调验证的 miRNA 调控 RNA 甲基化修饰 m^{6}A 形成的全新作用机制的工作更以封面文章的形式发表于干细胞领域顶尖杂志 *Cell Stem Cell* 中。这一工作在解析 m^{6}A 修饰形成的位点选择机制、拓展 miRNA 的新功能和发现新的细胞重编程调控因素方面均取得了开创性的成果。

易汉博为您提供最具性价比的标准生物信息服务和科研合作式的定制化生物信息分析和数据解读服务，帮助您充分地利用生物大数据来促进新的科学发现。

因为专注，所以专业。让易汉博同您一起，领略生命之美，护航健康生活。

业务范围

- 转录组数据分析
- lncRNA 鉴定分析
- 芯片数据分析
- smRNA 测序分析
- 基因组测序分析
- ChIP-seq 数据分析
- 单基因功能注释
- 生物网络分析
- 生物数据定制分析`

服务特色

超值的标准分析　专业的项目设计　深入的定制分析　精准的结果呈现

周期短	生物信息专业博士	科研问题为导向	信得过的分析
宽度广	国家重大项目经验	数据模式为依据	看得懂的结果
价格低	国际一流科研视角	合作交流为模式	用的上的图文

联系我们

邮箱：vip@ehbo.com
电话：010-51732515　15210822685
网址：www.ehbio.com
地址：北京市西城区德胜门外大街 11 号 1 幢 220