chentong_biology@163.com

Python_training 教程

陈同 (Chen Tong)(chentong_biology@163.com)

2014 年 8 月 21 日

Contents

chentong_biology@163.com

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。

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic License.

1 背景介绍

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：……

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”。在我教授过的朋友中，大家在写作 Python 的过程中都觉得很不可思议，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。

chentong_biology@163.com

- 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.

1.3 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。这儿推荐的两个集成包有完全免费的Anaconda和对学术用户、教育用户免费的Canopy. 这两个分发包集成了常用的数值计算、图形处理、多维可视化和其它有用的工具包如 `IPython`，可以节省大量的安装时间。

1.4 如何运行 Python 命令和脚本

- 对于初学者，本手册推荐直接在 `IPython Notebook` 下学习 `Python` 命令和脚本。我们这套教程也是用 `IPython 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 `Ipython notebook`[回车] 即可启动 `Ipython notebook`。
  - Windows 用户可以新建一个 `Ipython_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
    ipython notebook
    pause
    ```

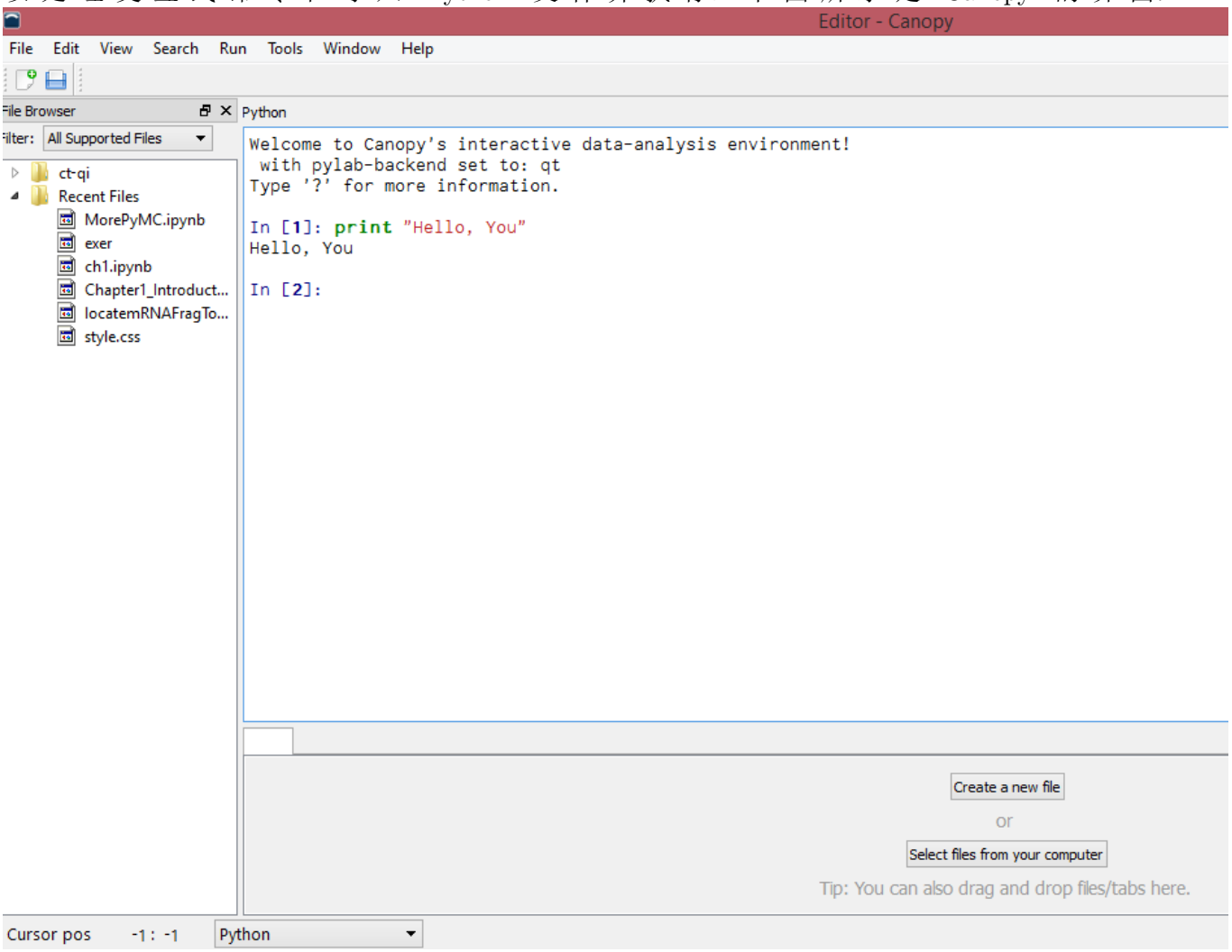
  - `Ipython notebook` 启动后会打开默认的浏览器（需要在图形用户界面下工作），这时可以新建或打开相应路径下的 ipynb 文件。

- 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。

chentong_biology@163.com

```
        @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。下图所示是 Canopy 的界面：



- 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。

chentong_biology@163.com

1.5 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 Python 程序事例

```
In [6]: # 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,
```

```
>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
```

chentong_biology@163.com

```
TGTCCACCATGGCACCTGGCATGCTCCGTGCCCCAAGGAGGCTCGTGGTCTTCCGGAATGGTGACCCGAA

In [10]: 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 aDict.items():
             print key
             print ''.join(valueL)

>NM\_011283 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCACACAAACTATTTACTCTTTTCTTCGTAAGGAAAGGTT

>NM\_0112835 gene=Rp1 CDS=128-6412
AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGTTTATACAGACCACACAAACTATTTACTCTTTTCTTCGTAAGGAAAGGTT

>NM\_001011874 gene=Xkr4 CDS=151-2091
gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcgcggccggggaaggaagccagggcgaggcgaggaggtggcgggaggaggagacagcaggga

>NM\_001195662 gene=Rp1 CDS=55-909
AAGCTCAGCCTTTGCTCAGATTCTCCTCTTGATGAAACAAAGGGATTTCTGCACATGCTTGAGAAATTGCAGGTCTCACCCAAAATGAGTGACACACCTTCTACTA

   %\end{Verbatim}
```

3 Python 语法

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 `Ipython 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**。

```
In [123]: print "不合适的缩进会引发错误，不该有的缩进"
          a = 'No indent'
           b = '我前面有个空格……'
```

chentong_biology@163.com

```
  File "<ipython-input-123-085115ffae95>", line 3
    b = ' 我前面有个空格……'
    ^
IndentationError: unexpected indent
```

```
In [124]: print "不合适的缩进，应该有，却溜掉的缩进"
          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-124-bd7d3136801b>", line 5
    print " 我应该被缩进，我从属于 for 循环!!!\n"
        ^
IndentationError: expected an indented block
```

```
In [203]: 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 "<ipython-input-203-1af46ff5a29f>", line 5
    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？"

    ^
IndentationError: unindent does not match any outer indentation level
```

3.2 变量、数据结构、流程控制

- 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。

chentong_biology@163.com

  - 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`。
  - 元组 (set，集合)：元组和列表十分类似，但元组中不允许重复值出现。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 `set`，可以使用 `aSet`。
  - 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。

chentong_biology@163.com

- 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。

3.2.1 数值变量操作

```
In [204]: print "数值变量"
          a = 5   # 注意等号两边的空格，为了易于辨识，操作符两侧最后有空格，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)
```

```
数值变量
5

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

```
In [205]: # 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，操作符两侧最后有空格，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.
```

chentong_biology@163.com

```
In [206]: 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:
              c = a
              a = b
              b = c
              #print " 保留节目，Python 特有，不通过中间变量直接做交换，神奇吧!! "
              #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
```

```
In [3]: 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 "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
2 * (a+b) = 16
取余数:  a % b = 2
取余数是很好的判断循环的地方，因为每个固定的数余数就会循环一次
```

3.2.2 字符串变量操作

```
In [208]: print "字符串变量"

          a = "Hello, welcome to Python"

          #a = 123
```

chentong_biology@163.com

```
#a = str(a)

print "The string a is:", a
print

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

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

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 <type 'str'>
```

```
In [209]: 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 character of a is %s\n" % a[len(a)-1]
print "\n这部分很重要啊，字符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 character of a is n
```

这部分很重要啊，字符的索引和切片操作是及其常用的。

```
In [4]: a = "oaoaoaoa"

print "遍历字符串"
for i in a:
```

chentong_biology@163.com

```
            print i

        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'''
```

```
遍历字符串
o
a
o
a
o
a
o
a
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
```

```
In [211]: 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 pos != -1:
              print pos + 1
              new = a[pos+1:].find('o')
```

chentong_biology@163.com

```
            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
```

```
In [212]: 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我了。

```
In [1]: 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

```
HelloPython
```

chentong_biology@163.com

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。

```
In [2]: 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'''

        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"
```

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!!
```

```
In [5]: 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
```

chentong_biology@163.com

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

        print "如果我只第一个 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！

如果我只第一个 o 怎么办呢？

HellO, Python
```

```
In []: print "字符串帮助"
       help(str)
```

```
In [4]: 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>
```

```
In [218]: 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
```

chentong_biology@163.com

```
        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

```
The memory index of b is 37117024
The memory index of b is 56557120
The memory index of b is 57454016
The memory index of b is 57334640
The memory index of b is 57334640
The memory index of b is 57350952
The memory index of b is 57201408
The memory index of b is 57142240
```

```
In [219]: print "字符串转数组"
          str1 = "array"
          print list(str1)
          a = list(str1)
          a.reverse()
          print ''.join(a)
```

字符串转数组

```
['a', 'r', 'r', 'a', 'y']
yarra
```

```
In [8]: print "数字字符串转数值"
        a = '123'
        print a+'1', int(a)+1
        a = '123.5'
        print a + 1
        print float(a)+1
```

数字字符串转数值

```
1231 124
```

```
        ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-8-e09ea72bfa6e> in <module>()
```

chentong_biology@163.com

```
      3 print a+'1', int(a)+1
      4 a = '123.5'
----> 5 print a + 1
      6 print float(a)+1


TypeError: cannot concatenate 'str' and 'int' objects
```

3.2.3 列表操作

```
In [220]: print "# 构建一个数组"
          aList = [1,2,3,4,5]
          print aList
          print
          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。"
```

```
# 构建一个数组
[1, 2, 3, 4, 5]

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。
```

```
In [221]: aList = []
          print "# 向数组中增加元素"
          aList.append(6)
          print aList

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

chentong_biology@163.com

```
# 向数组中增加元素
[6]

# 向数组中增加一个数组

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

```
In [5]: aList = [1,2,3,4,3,5]
        print "在数组中删除元素"
        aList.remove(3)
        print
        print aList

        aList.pop(3)
        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
```

```
In [223]: aList = [1,2,3,4,5]

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

          print "# 输出数组中大于 3 的元素"
          print

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

```
# 遍历数组的每个元素
```

chentong_biology@163.com

```
1
2
3
4
5
# 输出数组中大于 3 的元素

4
5
```

```
In [224]: 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
```

```
In [225]: 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 > 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 3, 且小于 10 的元素

0
1
2
3
4
26
27
```

chentong_biology@163.com

```
28
29
```

```
In [6]: 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:
            # 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
```

```
In [7]: 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] #Wrong
        #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
```

chentong_biology@163.com

```
In [228]: 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]
```

3.2.4 元组操作

```
In [229]: print "构建一个元组"
          aSet = set([1,2,3])
          print aSet

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

```
构建一个元组
set([1, 2, 3])
增加一个元素
set([1, 2, 3, 4])
set([1, 2, 3, 4])
```

```
In [230]: print "采用转换为元组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))
```

```
采用转换为元组去除列表中的重复元素
[1, 2, 1, 3, 1, 5, 2, 4, 3, 3, 6]
```

chentong_biology@163.com

```
set([1, 2, 3, 4, 5, 6])
[1, 2, 3, 4, 5, 6]
```

3.2.5 Range 使用

```
In [231]: print "使用 range, 产生一系列的字符串"
          for i in range(16):
              if i % 4 == 0:
                  print i
          print "通过指定步长产生 4 的倍数的数"
          for i in range(0,16,4):
              print i
```

```
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
```

3.2.6 字典操作

```
In [232]: print "# 构建一个字典"
          aDict = {1:2,3:4,'a':'b','d':'c'}

          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
```

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

```
In [233]: print
          print "输出字典的键值对 (key-value)"
```

chentong_biology@163.com

```
        for key, value in aDict.items():
            print key,value
```

输出字典的键值对 (key-value)

```
a b
1 2
3 4
e f
d c
5 6
```

```
In [234]: print "有序输出字典的键值对 (key-value)"
          keyL = aDict.keys()
          print 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)

```
['a', 1, 3, 'e', 'd', 5]
[1, 3, 5, 'a', 'd', 'e']
1 2
3 4
5 6
a b
d c
e f
```

```
In [235]: 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 aDict.items():
              print key
```

chentong_biology@163.com

```
            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]}

key2
        3
        4
        5
key
        1
        2
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。
```

```
In [236]: 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 aDict.items():
              print key
              for subKey, subV in 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
{}
```

chentong_biology@163.com

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

key2
        1       2
        3       4
key
        subkey          subvalue
```

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。

4 输入输出

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。

```
In []: a = raw_input("Please input a string\n> ")

       print "The string you typed in is: ", a

Please input a string
> a
The string you typed in is:  a

In []: print "这是一个保留例子，仅供玩耍\n"
       lucky_num = 5
       c = 0

       while True:
           b = int(raw_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 = raw_input("Do you want to try again? [Y/N] \n")
                   if c == 'Y':
                       try_again = 1
                       break
```

chentong_biology@163.com

```
            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

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。

```
In [9]: 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 >>fh, context
        #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)
```

```
In [2]: print "以读写模式 (r) 读入一个名为 (Test_file.txt) 的文件"

        print

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

chentong_biology@163.com

以读写模式 (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.
```

```
In [10]: 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 "在 print 语句后加上逗号 (,) 可以阻止 Python 对每次输出自动添加的换行符"
         print

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

         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。
在 print 语句后加上逗号 (,) 可以阻止 Python 对每次输出自动添加的换行符

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

2. Practise with real demands.



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.
```

chentong_biology@163.com

2. Practise with real demands.

5 实战练习（一）

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

@HWI-ST1223:80:D1FMTACXX:2:1101:1243:2213 1:N:0:AGTCAA TCTGTGTAGC-CNTGGCTGTCCTGGAACTCACTTTGTAGACCAGGCTGGCATGCA + BC-CFFFFFFHH#4AFHIJJJJJJJJJJJJJJJJJIJIJJJJJGHIJJJJJJJJJ

```
@EAS100R:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCAG
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACA
+
!*((((***+))%%%++)(%%%%).1***-+*))**55CCF>>>>>>CCCCCCC65
```

These four lines represent a single read.

The first line (always starting with '@') specifies the name of the read.

The second line is the actual sequence of the read. A,C,G,T and N characters are allowed in this line. N is used if the real base couldn't be determined.

The third line contains either only the '+' sign or it may contain the read name starting after the '@' sign.

Finally the last line is the ASCII encoding of the quality of the base exactly two lines above. Please check http://en.wikipedia.org/wiki/FASTQ_format for detailed description.

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)

chentong_biology@163.com

  - 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

chentong_biology@163.com

  - 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
```

chentong_biology@163.com

```
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。

chentong_biology@163.com

```
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。
- 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**。

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` 和 `range`。

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。

```
In [29]: def print_hello():
             print "Hello, you!"

         print_hello()
```

```
Hello, you!
```

```
In [30]: def hello(who):
             print "Hello, %s!" % who

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

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

chentong_biology@163.com

```
In [31]: 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]
```

```
In [32]: def read(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 aDict.items():
                 aDict[name] = ''.join(lineL)
             return aDict

         print read("data/test1.fa")
         read("data/test2.fa")
```

```
{'>NM_0112835 gene=Rp15 CDS=128-6412': 'AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCAGTGACGT
```

```
Out[32]: {'>NM_001011874 gene=Xkr4 CDS=151-2091': 'gcggcggcgggcgagcgggcgctggagtaggagctggggagcggcg
          '>NM_001195662 gene=Rp1 CDS=55-909': 'AAGCTCAGCCTTTGCTCAGATTCTCCTCTTGATGAAACAAAGGGATTTC
          '>NM_011283 gene=Rp1 CDS=128-6412': 'AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCCA
          '>NM_0112835 gene=Rp1 CDS=128-6412': 'AATAAATCCAAAGACATTTGTTTACGTGAAACAAGCAGGTTGCATATCC
```

chentong_biology@163.com

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。

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

```
In [57]: import os
         os.getcwd()
         #help(os.getcwd)
         #os.remove(r'D:\project\github\PBR_training\script\splitName.py')
         #os.system('rm file')

Out[57]: 'D:\\project\\github\\PBR_training'

In [58]: from os import getcwd
         getcwd()

Out[58]: 'D:\\project\\github\\PBR_training'
```

chentong_biology@163.com

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。

```
In [59]: %%writefile testSys.py
         import sys
         print sys.argv

Overwriting testSys.py

In [60]: %run testSys 'abc' cde 1234

['testSys.py', "'abc'", 'cde', '1234']

In [61]: %%writefile cat.py
         import sys

         def read_print_file(filename):
             for line in open(filename):
                 print line,
         #------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
                 print >>sys.stderr, "Usage: python %s filename" % sys.argv[0] # 一般提示信息输出
                 sys.exit(0)
             file = sys.argv[1]
             read_print_file(file)
         #--------END main------------------
         if __name__ == '__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
             main()

Overwriting cat.py

In [62]: %run cat

Usage: python cat.py filename

In [63]: %run cat data/test1.fq

@HWI-ST1223:80:D1FMTACXX:2:1101:1243:2213 1:N:0:AGTCAA
TCTGTGTAGCCNTGGCTGTCCTGGAACTCACTTTGTAGACCAGGCTGGCATGCACCACCACNNNCGGCTCATTTGTCTTTNNTTTTTGTTTTGTTCT
+
BCCFFFFFFHH#4AFHIJJJJJJJJJJJJJJJJIJIJJJJJGHIJJJJJJJJJJJJJJIIJ###--5ABECFFDDEEEEE##,5=@B8?CDD<AD>C
@HWI-ST1223:80:D1FMTACXX:2:1101:1375:2060 1:N:0:AGTCAA
```

chentong_biology@163.com

```
NTGCTGAGCCACGACAAGGATCCCAGAGGGCCNAGCCCTGCATCTTGTATGGACCAGTTACNCATCAAAAGAGACTACTGTAGGCACCATCAATCAC
+
#1:DDDD;?CFFHDFEEIGIIIIIIG;DHFGG#)0?BFBDHBFF<FCFEFD;@DD@A=7?E#,,,;=(>3;=;;C>ACCC@CCCCCBBBCCAACCCC
@HWI-ST1223:80:D1FMTACXX:2:1101:1383:2091 1:N:0:AGTCAA
NGTTCGTGTGGAACCTGGCGCTAAACCATTCGTAGACGACCTGCTTCTGGGTCGGGGTTTCGTACGTAGCAGAGCAGCTCCCTCGCTGCGATCTATT
+
#1=DDFDFHHHHHJGJJJJJJJJJJJJJJIJIGDHIHIGIJJJJJJJIIIGHHFDD3>BDDBDDDDDDDDDDBDCCBDDDDDDDDDDDBBDDDDEE
@HWI-ST1223:80:D1FMTACXX:2:1101:1452:2138 1:N:0:AGTCAA
NTCTAGGAGGTCTAGAAAGCCCAGGCCACCGGTACAAACATCAAGGGTGTTACGGATGTGCCGCTCTGAACCTCCAGGACGACTTTGATTTCAACTA
+
#4=DFFEFHHHHHJJJJJIJJJJHIIJGJJJJ@GIIJJJJJJIJJJJFGHIIIJJHHHDFFFFDDDDDDDDDDDDCDDDDDDDDDDCCCEDEDDDI
```

使用 optparse，功能更强大 (保留内容)

In [64]:
```
%%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",
```

chentong_biology@163.com

```
                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 >>sys.stderr,\
                "--Successful %s" % strftime(timeformat, localtime())
    if __name__ == '__main__':
        startTime = strftime(timeformat, localtime())
        main()
        endTime = strftime(timeformat, localtime())
        fh = open('python.log', 'a')
        print >>fh, "%s\n\tRun time : %s - %s " % \
            (' '.join(sys.argv), startTime, endTime)
        fh.close()
```

```
Overwriting skeleton.py
```

```
In [65]: %run skeleton -h
```

```
Usage: skeleton.py -i file
```

chentong_biology@163.com

```
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
```

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。

9 更多 Python 内容

9.1 单语句块

```
In [90]: if True:
             print 'yes'

         if True: print 'yes'
```

chentong_biology@163.com

```
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
```

9.2 列表综合，生成新列表的简化的 for 循环

```
In [91]: 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]
```

```
In [92]: print "列表综合可以做判断的"
         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]
```

```
In [93]: print "列表综合也可以嵌套的"
         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:
```

chentong_biology@163.com

```
        #     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]
```

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。

```
In [94]: 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-94-1cdfb2fe6c3a> 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
```

9.3 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，

chentong_biology@163.com

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。

```
In [95]: print "求和函数"
         f = lambda x,y: 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
```

```
In [96]: print "单个参数的 map, lambda 调用"
         aList = [1,2,3,4,5]
         print map(lambda x: x**2, aList)

         print "多个参数的 map, lambda 调用"
         f = lambda x,y: x + y
         print map(f,[1,2,3],[4,5,6])

         print "参数为字符串"
         print map(lambda x: x.upper(), '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']
```

```
In [97]: print "输出所有的奇数"
         aList = [1,2,3,4,5]
         print filter(lambda x: x%2, aList)
```

```
输出所有的奇数
[1, 3, 5]
```

```
In [98]: print "列表求和"
         aList = [1,2,3,4,5]
         print reduce(lambda a,b: a+b, aList)
```

```
列表求和
15
```

chentong_biology@163.com

```
In [99]: 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
```

exec, eval (执行字符串 python 语句, 保留节目)

```
In [100]: a = 'print "Executing a string as a command"'
          exec(a)
```

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

```
In [101]: a = '(2 + 3) * 5'
          eval(a)
```

```
Out[101]: 25
```

10 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-python-january-iap-2008/

http://my.oschina.net/taogang/blog/286955