Play with Python Introduction to programming with python

Vani

Peking University

December 11, 2012

猜一下下面程序的作用

2 / 31

猜一下下面程序的作用

print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])

猜一下下面程序的作用

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

Vani (Peking University) Play with Python December 11, 2012 2

猜一下下面程序的作用

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

```
\label{lem:main} \verb|main=interact$(\[x,y]->zipWith(\a b->if a==b then '0' else '1')x y).lines \\
```

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

```
\label{lem:main} \verb|main=interact$(\[x,y]=\] -\[ x = b \] -\[ x = b \
```

```
#include <atdio.bb
#include <atdio.bc
#include <atdio.action() {
    char sirton() action() {
        int main(int argo, char *argv()) {
            char sirton() action();
        int i;
            cean("Wa\nWa", si, s2);
        for (i = 0; i < atrlen(si); i++) {
            if (sat(i) = s2(i))
            putchar('1');
            else
            putchar('0');
        }
}</pre>
```

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

C

```
#isclude cetdio.bc
finclude cetting.bc
int main(int arge, char *argv[]) {
    char a![100], s2[100];
    int i;
    iscnf("%ahu%a", s1, s2);
    for (i = 0; i < strlen(s1); i++) {
        if (s[i] = s2[i])
        putchar('1');
        putchar('0');
    }
}</pre>
```

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

Haskell

```
main=interact([x,y]-zipWith(a b-zif a==b then '0' else '1')x y).lines
```

► C

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
   char s1[100], s2[100];
   int i;
   scanf("%s\n%s", s1, s2);
   for (i = 0; i < strlen(s1); i++) {
       if (s1[i] == s2[i])
     putchar('1');
     putchar('0');
```

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

Ruby

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

Haskell

```
\label{lem:main} \verb|main=interact$(\[x,y]->zipWith(\a b->if a==b then '0' else '1')x y).lines \\
```

C

```
#include <artio.nb
#include <artinc.nb
int main(int argc, char *argv[]) {
    char *s[100], s2[100];
    int i;
    scan("%a\n%a", s1, s2);
    for (i = 0; i <artinc.sez[i])
        putchar('1');
        putchar('1');
        else
        putchar('0');
    }
}</pre>
```

猜一下下面程序的作用

Python

```
print ''.join([['0','1'][i!=j] for i,j in zip(raw_input(),raw_input())])
```

Ruby

```
$><<"%0#gets=~/$/b"%($_.to_i(2)^gets.to_i(2))
```

Haskell

```
main=interact([x,y]-zipWith(a b->if a==b then '0' else '1')x y).lines
```

C

▶ 优势:

- ▶ 优势:
 - ▶ 解释性

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库
 - ▶ 甜甜的语法糖

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库
 - ▶ 甜甜的语法糖
- ▶ 劣势

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库
 - ▶ 甜甜的语法糖
- ▶ 劣势
 - ▶ 执行速度慢

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库
 - ▶ 甜甜的语法糖
- ▶ 劣势
 - ▶ 执行速度慢
 - ▶ 奇怪的面向对象机制

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库
 - ▶ 甜甜的语法糖
- ▶ 劣势
 - ▶ 执行速度慢
 - ▶ 奇怪的面向对象机制
 - ▶ 命名混乱

3 / 31



4 / 31

▶ 整数



- ▶ 整数
- ▶ 长整数



- ▶ 整数
- ▶ 长整数

>>> print 2**200

1606938044258990275541962092341162602 522202993782792835301376

- ▶ 整数
- ▶ 长整数
- ▶ 浮点数

>>> print 2**200

1606938044258990275541962092341162602 522202993782792835301376

- ▶ 整数
- ▶ 长整数
- ▶ 浮点数
- ▶ 复数

>>> print 2**200

1606938044258990275541962092341162602 522202993782792835301376

- ▶ 整数
- ▶ 长整数
- ▶ 浮点数
- ▶ 复数

>>> print 2**200

1606938044258990275541962092341162602 522202993782792835301376

5+10j

字符串

字符串是一种特殊的列表。字符串可以由单引号,双引号以及三引号来声明, 其中单引号和双引号是完全相同的,三引号用来声明多行字符串



Vani (Peking University) Play with Python December 11, 2012 5 / 31

字符串

字符串是一种特殊的列表。字符串可以由单引号,双引号以及三引号来声明, 其中单引 号和双引号是完全相同的,三引号用来声明多行字符串

▶ 可以直接使用加号来连接字符串, 用乘号重复字符串



Vani (Peking University) Play with Python December 11, 2012 5 / 31

字符串

字符串是一种特殊的列表。字符串可以由单引号,双引号以及三引号来声明, 其中单引 号和双引号是完全相同的,三引号用来声明多行字符串

▶ 可以直接使用加号来连接字符串, 用乘号重复字符串



▶ 可以直接使用加号来连接字符串, 用乘号重复字符串

Code

- >>> a="a+"
- >>> b='b'
- >>> print a+b, b*3

▶ 可以直接使用加号来连接字符串, 用乘号重复字符串

```
Code
>>> a="a+"
>>> b='b'
>>> print a+b, b*3
a+b bbb
```

5 / 31

Code

- ▶ 可以直接使用加号来连接字符串, 用乘号重复字符串
- ▶ Python 支持格式化字符串,类似 C 里面 printf() 的字符串格式化

>>> a="a+" >>> b='b' >>> print a+b, b*3 a+b bbb

- ▶ 可以直接使用加号来连接字符串, 用乘号重复字符串
- ▶ Python 支持格式化字符串,类似 C 里面 printf() 的字符串格式化

Code

```
>>> a="a+"
>>> b='b'
```

>>> print a+b, b*3

a+b bbb

>>> c=74

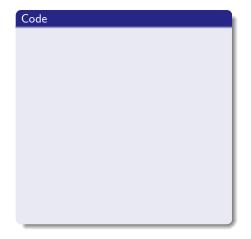
- ▶ 可以直接使用加号来连接字符串, 用乘号重复字符串
- ▶ Python 支持格式化字符串,类似 C 里面 printf() 的字符串格式化

Code

```
>>> a="a+"
>>> b='b'
>>> print a+b, b*3
a+b bbb
>>> c=74
>>> print "%d is a lucky number."%(c)
74 is a lucky number.
```

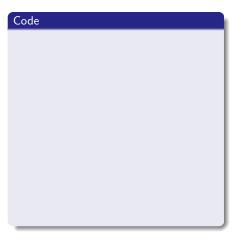
运算符

Python 的大部分运算符与 C 的功能一样,下面简单介绍一下一些与 C 不同的运算符.



Vani (Peking University) Play with Python December 11, 2012 6 / 31

▶ 乘方运算符 **



Vani (Peking University) Play with Python December 11, 2012 6 / 31

▶ 乘方运算符 **

Code

>>> print 2**10 1024

Vani (Peking University) Play with Python December 11, 2012 6 / 31

- ▶ 乘方运算符 **
- ▶ 整除运算符 //

Code

>>> print 2**10 1024

Vani (Peking University) Play with Python December 11, 2012 6 / 31

- ▶ 乘方运算符 **
- ▶ 整除运算符 //

Code

>>> print 2**10 1024

>>> print 1.5//0.2 7.0

6 / 31

- ▶ 乘方运算符 **
- ▶ 整除运算符 //
- ► 逻辑运算符 and, or, not. 短路计算 在 Python 里也适用

Code

>>> print 2**10 1024

>>> print 1.5//0.2 7.0

- ▶ 乘方运算符 **
- ▶ 整除运算符 //
- ▶ 逻辑运算符 and, or, not. 短路计算在 Python 里也适用

Code

>>> print 2**10 1024

>>> print 1.5//0.2 7.0

 $>\!>\!>$ print True and not False True

- ▶ 乘方运算符 **
- ▶ 整除运算符 //
- ► 逻辑运算符 and, or, not. 短路计算 在 Python 里也适用
- ▶ Python 没有自增自减运算符,但 有复合赋值运算符

Code

>>> print 2**10 1024

>>> print 1.5//0.2 7.0

>>> print True and not False True

- ▶ 乘方运算符 **
- ▶ 整除运算符 //
- ► 逻辑运算符 and, or, not. 短路计算 在 Python 里也适用
- ▶ Python 没有自增自减运算符,但有复合赋值运算符

Code

```
>>> print 2**10
1024
```

```
>>> print 1.5//0.2 7.0
```

>>> print True and not False True

```
>>> a=42
>>> a%=10
>>> print a
2
```

- ▶ 乘方运算符 **
- ▶ 整除运算符 //
- ► 逻辑运算符 and, or, not. 短路计算 在 Python 里也适用
- ▶ Python 没有自增自减运算符,但 有复合赋值运算符
- 利用短路计算模拟问号操作符.

Code

```
>>> print 2**10 1024
```

```
>>> print 1.5//0.2 7.0
```

>>> print True and not False True

```
>>> a=42
>>> a%=10
>>> print a
```

- ▶ 乘方运算符 **
- ▶ 整除运算符 //
- ► 逻辑运算符 and, or, not. 短路计算 在 Python 里也适用
- ▶ Python 没有自增自减运算符,但有复合赋值运算符
- ▶ 利用短路计算模拟问号操作符. 有 缺陷!

```
>>> print 2**10
1024
>>> print 1.5//0.2
7.0
>>> print True and not False
True
>>> a=42
>>> a%=10
>>> print a
2
>>> a = -10
>>> print a < 0 and -a or a
10
```



▶ Python 里使用缩进来决定语句的 分组, 就像 C++ 里的大括号



Vani (Peking University) Play with Python December 11, 2012 7 /

- ▶ Python 里使用缩进来决定语句的 分组, 就像 C++ 里的大括号
- ▶ 不要忘记语句尾有一个冒号



- ▶ Python 里使用缩进来决定语句的 分组, 就像 C++ 里的大括号
- ▶ 不要忘记语句尾有一个冒号

```
>>> if a < 10:
... print "a<10"
... else:
... if a >= 10 and a < 20:
... print "a >= 10 and a < 20"
... else:
... print "a >= 20"
```

- ▶ Python 里使用缩进来决定语句的 分组. 就像 C++ 里的大括号
- 不要忘记语句尾有一个冒号
- ▶ 错误的缩进会引发错误

```
>>> if a < 10:
... print "a<10"
... else:
... if a >= 10 and a < 20:
... print "a >= 10 and a < 20"
... else:
... print "a >= 20"
```

- ▶ Python 里使用缩进来决定语句的 分组. 就像 C++ 里的大括号
- 不要忘记语句尾有一个冒号
- ▶ 错误的缩进会引发错误

```
>>> if a < 10:
... print "a<10"
... else:
... if a >= 10 and a < 20:
... print "a >= 10 and a < 20"
... else:
... print "a >= 20"
>>> i=5
>>> print "Value is",i
```

- ▶ Python 里使用缩进来决定语句的 分组. 就像 C++ 里的大括号
- ▶ 不要忘记语句尾有一个冒号
- ▶ 错误的缩进会引发错误

```
>>> if a < 10:
...     print "a<10"
...     else:
...     if a >= 10 and a < 20:
...         print "a >= 10 and a < 20"
...     else:
...         print "a >= 20"
>>> i=5
>>>     print "Value is",i

Print 'value is'
IndentationError: unexpected indent
```

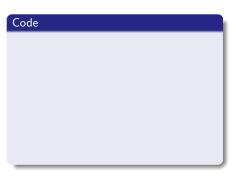
- ▶ Python 里使用缩进来决定语句的 分组. 就像 C++ 里的大括号
- ▶ 不要忘记语句尾有一个冒号
- ▶ 错误的缩进会引发错误
- ▶ 缩进可以使用一个制表或者四个空格,但请不要混合使用制表和空格缩进

```
>>> if a < 10:
... print "a<10"
... else:
... if a >= 10 and a < 20:
... print "a >= 10 and a < 20"
... else:
... print "a >= 20"
>>> i=5
>>> print "Value is",i

IndentationError: unexpected indent
```



▶ if 的条件判断可以不加括号



Vani (Peking University) Play with Python December 11, 2012 8 / 31

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif



Vani (Peking University) Play with Python December 11, 2012 8 / 31

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif

```
a=10
if a < 10:
    print "less than 10"
elif a == 10:
    print "equal to 10"
else:
    print "greater than 10"</pre>
```

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif
- ▶ Python 里没有 switch 语句,可以 使用 if...elif 来代替

Code a=10

```
a=10
if a < 10:
    print "less than 10"
elif a == 10:
    print "equal to 10"
else:
    print "greater than 10"</pre>
```

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif
- ▶ Python 里没有 switch 语句,可以 使用 if...elif 来代替

Code a=10

```
a=10
if a < 10:
    print "less than 10"
elif a == 10:
    print "equal to 10"
else:
    print "greater than 10"</pre>
```

else:

判断语句

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif
- ▶ Python 里没有 switch 语句,可以 使用 if…elif 来代替
- ▶ if 的三元表达式:

X if C else Y

a=10 if a < 10: print "less than 10" elif a == 10: print "equal to 10"</pre>

print "greater than 10"

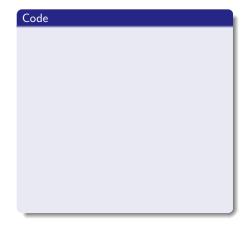
- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif
- ▶ Python 里没有 switch 语句,可以 使用 if...elif 来代替
- ▶ if 的三元表达式:

X if C else Y

```
Code
a=10
if a < 10:
    print "less than 10"
elif a == 10:
    print "equal to 10"
else:
    print "greater than 10"

>>> a = 10
>>> print "even" if a % 2 == 0 else "odd"
even
```

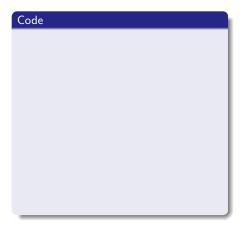
Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.



Vani (Peking University) Play with Python December 11, 2012

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

▶ while 语句后可接 else 从句, else 后的内容在循环结束后或者 break 后执行



Vani (Peking University) Play with Python December 11, 2012

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

▶ while 语句后可接 else 从句, else 后的内容在循环结束后或者 break 后执行

```
a=74
while a % 42 != 0:
    a+=1
else:
    print "a is", a
```

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

while 语句后可接 else 从句, else 后的内容在循环结束后或者 break 后执行

```
Code
a=74
while a % 42 != 0:
    a+=1
else:
    print "a is", a
Output:
a is 84
```

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

- ▶ while 语句后可接 else 从句,else 后的内容在循环结束后或者 break 后执行
- ▶ for 语句的格式为

for [item] in [volume]:

其中 volume 必须为可遍历的类型,比如列表,迭代器,而 item 必须与 volume 内元素类型相符

Code

```
a=74
while a % 42 != 0:
    a+=1
else:
    print "a is", a
```

Output: a is 84

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

- ▶ while 语句后可接 else 从句,else 后的内容在循环结束后或者 break 后执行
- ▶ for 语句的格式为

for [item] in [volume]:

其中 volume 必须为可遍历的类型,比如列表,迭代器,而 item 必须与 volume 内元素类型相符

▶ for 语句后也可接 else

```
a=74
while a % 42 != 0:
    a+=1
else:
    print "a is", a

Output:
a is 84
```

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

- ▶ while 语句后可接 else 从句,else 后的内容在循环结束后或者 break 后执行
- ▶ for 语句的格式为

```
for [item] in [volume]:
```

其中 volume 必须为可遍历的类型, 比如列表,迭代器,而 item 必须 与 volume 内元素类型相符

▶ for 语句后也可接 else

```
a=74
while a % 42 != 0:
    a+=1
else:
    print "a is", a

Output:
a is 84

for i in range(0, 11):
    print i
```

Python 里的有 while 循环语句和 for 循环语句 其中 while 循环和 C 用法基本相同,而 for 循环用于遍历列表或者迭代器.

- ▶ while 语句后可接 else 从句,else 后的内容在循环结束后或者 break 后执行
- ▶ for 语句的格式为

```
for [item] in [volume]:
```

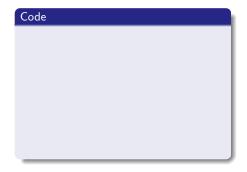
其中 volume 必须为可遍历的类型,比如列表,迭代器,而 item 必须与 volume 内元素类型相符

▶ for 语句后也可接 else

```
Code
a=74
while a % 42 != 0:
    a+=1
else:
    print "a is", a
Output:
a is 84
for i in range(0, 11):
    print i
Output:
2
```

定义函数

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

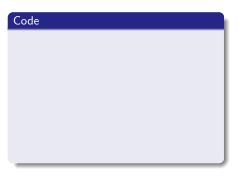


Vani (Peking University) Play with Python December 11, 2012 10 / 31

定义函数

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

▶ 函数的参数列表不用指定类型



Vani (Peking University) Play with Python December 11, 2012 10 / 31

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

Code

▶ 函数的参数列表不用指定类型

```
>>> def Double(a):
... return a * 2
>>> print Double(3)
>>> print Double("kill")
```

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

▶ 函数的参数列表不用指定类型

```
>>> def Double(a):
... return a * 2
>>> print Double(3)
>>> print Double("kill")
6
killkill
```

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

- 函数的参数列表不用指定类型
- ▶ 调用时必须要加双括号

```
>>> def Double(a):
... return a * 2
>>> print Double(3)
>>> print Double("kill")
6
killkill
```

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

- ▶ 函数的参数列表不用指定类型
- ▶ 调用时必须要加双括号
- ► 函数内变量均为局部变量,可使用 global 关键字声明全局变量

```
>>> def Double(a):
... return a * 2
>>> print Double(3)
>>> print Double("kill")
6
killkill
```

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

- ▶ 函数的参数列表不用指定类型
- ▶ 调用时必须要加双括号
- ► 函数内变量均为局部变量,可使用 global 关键字声明全局变量
- ▶ 不推荐!

```
>>> def Double(a):
... return a * 2
>>> print Double(3)
>>> print Double("kill")
6
killkill
```

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一写变量名

- ▶ 函数的参数列表不用指定类型
- ▶ 调用时必须要加双括号
- ► 函数内变量均为局部变量,可使用 global 关键字声明全局变量
- ▶ 不推荐!
- ▶ 当参数为一个序列时,如果在序列 前加一个*,则自动将序列分割成 参数表

```
>>> def Double(a):
... return a * 2
>>> print Double(3)
>>> print Double("kill")
6
killkill
```

函数通过关键字 def 定义,后接一个函数标识符名称,然后跟一对圆括号,其中包括一 写变量名

- 函数的参数列表不用指定类型
- 调用时必须要加双括号
- ▶ 函数内变量均为局部变量,可使用 global 关键字声明全局变量
- ▶ 不推荐!
- ▶ 当参数为一个序列时,如果在序列 前加一个*,则自动将序列分割成 参数表

```
>>> def Double(a):
      return a * 2
>>> print Double(3)
>>> print Double("kill")
killkill
>>> def f(a, b, c):
... return a + b + c
>>> print f(*[1, 2, 3])
```

可通过 lambda 关键字声明匿名函数, 格式为:

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



11 / 31

可通过 lambda 关键字声明匿名函数, 格式为:

▶ lambda 表达式的返回值即为一个 函数



可通过 lambda 关键字声明匿名函数, 格式为:

▶ lambda 表达式的返回值即为一个 函数

Code

```
print (lambda x: x + 1)(41)
```

Output: 42

可通过 lambda 关键字声明匿名函数, 格式为:

- ▶ lambda 表达式的返回值即为一个 函数
- ▶ 思考: 如何使用 lambda 函数来定 义阶乘函数 fac(x)

Code

```
print (lambda x: x + 1)(41)
```

Output: 42

Vani (Peking University)

▶ 一个可行的实现:

▶ 一个可行的实现:

```
Code
```

```
# Construct assitant function
YA = lambda x: lambda y: y(lambda : x(x)(y))
YB = YA(YA)

# Factor function
fact0 = lambda self: lambda n: (n==0) and 1 or n * self()(n-1)
fact = YB(fact0)

# >>> fact(6)
# 720
```

▶ 一个可行的实现:

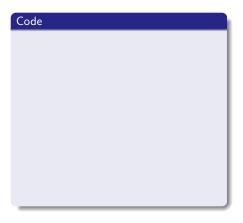
```
Code
```

```
# Construct assitant function
YA =lambda x: lambda y: y(lambda : x(x)(y))
YB = YA(YA)

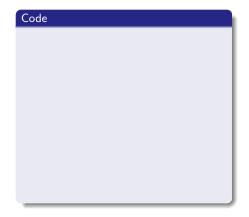
# Factor function
fact0 = lambda self: lambda n: (n==0) and 1 or n * self()(n-1)
fact = YB(fact0)

# >>> fact(6)
# 720
```

▶ 可自行学习函数式编程相关知识 (lambda 演算, Y combanitor...)



13 / 31



Code

▶ 列表中各个元素的类型可以不同

Code

>>> print [3, "123"]
[3, "123"]

▶ 列表中各个元素的类型可以不同

Code

>>> print [3, "123"]
[3, "123"]

- ▶ 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表

- ▶ 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表

```
>>> print [3, "123"]
[3, "123"]
```

```
>>> a = [[1,2,3],[4,5,6]]
print a[0][1]
```

- 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表
- ▶ 列表生成器

```
>>> print [3, "123"]
[3, "123"]
```

```
>>> a = [[1,2,3],[4,5,6]]
print a[0][1]
```

- 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表
- ▶ 列表生成器

```
>>> print [3, "123"]
[3, "123"]
>>> a = [[1,2,3],[4,5,6]]
print a[0][1]
5
>>> print [i for i in range(0, 3)]
[0, 1, 2]
```

- 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表
- ▶ 列表生成器

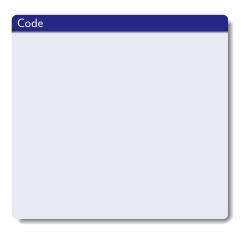
```
>>> print [3, "123"]
[3, "123"]
>>> a = [[1,2,3],[4,5,6]]
print a[0][1]
5
>>> print [i for i in range(0, 3)]
[0, 1, 2]
```

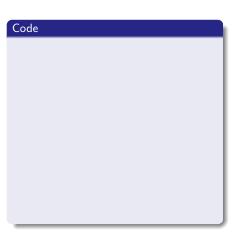
- ▶ 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表
- ▶ 列表生成器

```
>>> print [3, "123"]
[3, "123"]
>>> a = [[1,2,3],[4,5,6]]
print a[0][1]
5

>>> print [i for i in range(0, 3)]
[0, 1, 2]
>>> print [i for i in range(0, 11) \
...if i % 3 == 0]
[0, 3, 6, 9]
```

数据结构





Code

>>> a = [3,2,1,4,5,6,7]

```
>>> a = [3,2,1,4,5,6,7]
>>> print a[1:4]
[3,2,1]
```

list.append(obj) 用于向 list 添加一个对象 obj

Code

>>> a = [3,2,1,4,5,6,7] >>> print a[1:4] [3,2,1]

list.append(obj) 用于向 list 添加一个对象 obj

```
>>> a = [3,2,1,4,5,6,7]
>>> print a[1:4]
[3,2,1]
>>> a.append(8)
>>> print a
[3,2,1,4,5,6,7,8]
```

list.append(obj) 用于向 list 添加一个对象 obj

len(list) 获得列表长度

```
>>> a = [3,2,1,4,5,6,7]
>>> print a[1:4]
[3,2,1]
>>> a.append(8)
>>> print a
[3,2,1,4,5,6,7,8]
```

list.append(obj) 用于向 list 添加一个对象 obj

len(list) 获得列表长度

```
>>> a = [3,2,1,4,5,6,7]

>>> print a[1:4]

[3,2,1]

>>> a.append(8)

>>> print a

[3,2,1,4,5,6,7,8]

>>> print len(a)

8
```

list.append(obj) 用于向 list 添加一个对象 obj

len(list) 获得列表长度

list.sort 列表内元素排序

```
>>> a = [3,2,1,4,5,6,7]
>>> print a[1:4]
[3,2,1]
>>> a.append(8)
>>> print a
[3,2,1,4,5,6,7,8]
>>> print len(a)
8
```

Code

>>> print len(a)

>>> a.sort() >>> print a [1,2,3,4,5,6,7,8]

[l:r] 用于获得一个列表的 子序列

list.append(obj) 用于向 list 添加一个对象 obj

len(list) 获得列表长度

list.sort 列表内元素排序

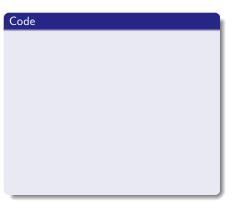
>>> a = [3,2,1,4,5,6,7] >>> print a[1:4] [3,2,1] >>> a.append(8) >>> print a [3,2,1,4,5,6,7,8]





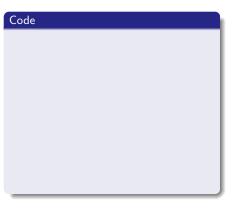
Vani (Peking University) Play with Python December 11, 2012 15 / 31

▶ 元组的定义中的圆括号是可以省略的



Vani (Peking University) Play with Python December 11, 2012 15 / 31

- ▶ 元组的定义中的圆括号是可以省略的
- ► 于是 Python 中变量赋值与交换两 个变量可写成如右所示:



Vani (Peking University) Play with Python December 11, 2012 15 / 31

- ▶ 元组的定义中的圆括号是可以省略的
- ► 于是 Python 中变量赋值与交换两 个变量可写成如右所示:

Code

```
>>> a,b = 1,2
>>> print a, b
1 2
>>> a,b = b,a
>>> print a, b
2 1
```

15 / 31

- ▶ 元组的定义中的圆括号是可以省略的
- ► 于是 Python 中变量赋值与交换两 个变量可写成如右所示:
- ▶ zip 函数接受 n 个等长序列作为参数,返回一个序列,其中第 i 项为由 n 个序列第 i 项构成的一个元组

```
>>> a,b = 1,2
>>> print a, b
1 2
>>> a,b = b,a
>>> print a, b
2 1
```

- ▶ 元组的定义中的圆括号是可以省略的
- ► 于是 Python 中变量赋值与交换两 个变量可写成如右所示:
- ▶ zip 函数接受 n 个等长序列作为参数,返回一个序列,其中第 i 项为由 n 个序列第 i 项构成的一个元组
- ▶ 可以同时遍历若干的序列

```
>>> a,b = 1,2
>>> print a, b
1 2
>>> a,b = b,a
>>> print a, b
2 1
```

- ▶ 元组的定义中的圆括号是可以省略的
- ► 于是 Python 中变量赋值与交换两个变量可写成如右所示:
- zip 函数接受 n 个等长序列作为参数,返回一个序列,其中第 i 项为由 n 个序列第 i 项构成的一个元组
- ▶ 可以同时遍历若干的序列

```
Code
>>> a,b = 1,2
>>> print a, b
1 2
>>> a,b = b,a
>>> print a, b
2 1
>>> a, b = [0,1,2],[3,4,5]
>>> for i, j in zip(a, b):
      print i. j
0.3
1 4
2 5
```

 $d = \{key1 : value1, key2 : value1\}$

$$d = \{key1 : value1, key2 : value1\}$$

▶ 字典的 items 返回一个元组的列表,其中每个元组都包含一对项目——关键字与对应的值



 $d = \{key1 : value1, key2 : value1\}$

- ▶ 字典的 items 返回一个元组的列表,其中每个元组都包含一对项目——关键字与对应的值
- ▶ 使用 items 来遍历字典:

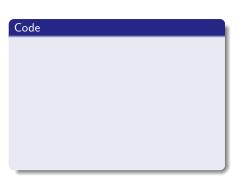
```
d = \{key1 : value1, key2 : value1\}
```

- ▶ 字典的 items 返回一个元组的列表,其中每个元组都包含一对项目——关键字与对应的值
- ▶ 使用 items 来遍历字典:

```
>>> a = "a": 1, "b": 2, "c": 3
>>> for i, j in a.items():
... print i, j
a 1
b 2
c 3
```



s.join(list) 用于连接字符串



s.join(list) 用于连接字符串

```
>>> a = ["Hello", "World", "!"]
>>> print "_".join(a)
Hello_World_!
```

s.join(list) 用于连接字符串 s.split(se) 用于分割字符串

```
>>> a = ["Hello", "World", "!"]
>>> print "_".join(a)
Hello_World_!
```

s.join(list) 用于连接字符串 s.split(se) 用于分割字符串

```
>>> a = ["Hello", "World", "!"]
>>> print "_".join(a)
Hello_World_!

>>> print "Hello_World_!".split("_")
["Hello", "World", "!"]
```

- s.join(list) 用于连接字符串
- s.split(se) 用于分割字符串
 - s.strip() 用于去除字符串开始和 结尾的空白字符

```
>>> a = ["Hello", "World", "!"]
>>> print "_".join(a)
Hello_World_!

>>> print "Hello_World_!".split("_")
["Hello", "World", "!"]

>> print " Hello ! ".strip()
Hello !
```



map(func, list) 将 func 作用 list 内每 个元素,返回结果组成 的序列



Vani (Peking University) Play with Python December 11, 2012 18 / 31

map(func, list) 将 func 作用 list 内每个元素,返回结果组成的序列

```
>>> a = [1, 2, 3]
>>> print map(lambda x: x * 2, a)
[2, 4, 5]
```

序列相关函数

map(func, list) 将 func 作用 list 内每 个元素,返回结果组成 的序列

filter(func, list) 将 func 作用于 list 内 每个元素,返回令 func 为真的元素组成的序列

```
>>> a = [1, 2, 3]
>>> print map(lambda x: x * 2, a)
[2, 4, 5]
```

```
map(func, list) 将 func 作用 list 内每
个元素,返回结果组成
的序列
```

filter(func, list) 将 func 作用于 list 内 每个元素,返回令 func 为真的元素组成的序列

Code

```
>>> a = [1, 2, 3]
>>> print map(lambda x: x * 2, a)
[2, 4, 5]
```

```
>>> a = [1, 2, 3, 4]
>>> print filter(lambda x: x %2 == 0, a)
[2, 4]
```

18 / 31

```
map(func, list) 将 func 作用 list 内每个元素,返回结果组成的序列
```

filter(func, list) 将 func 作用于 list 内 每个元素,返回令 func 为真的元素组成的序列

reduce(func, list) 通过二元函数 func 将 list 缩减为一个值

```
>>> a = [1, 2, 3]
>>> print map(lambda x: x * 2, a)
[2, 4, 5]
```

```
>>> a = [1, 2, 3, 4]
>>> print filter(lambda x: x %2 == 0, a)
[2, 4]
```

```
map(func, list) 将 func 作用 list 内每个元素,返回结果组成的序列
```

filter(func, list) 将 func 作用于 list 内 每个元素,返回令 func 为真的元素组成的序列

reduce(func, list) 通过二元函数 func 将 list 缩减为一个值

Code

>>> a = [1, 2, 3]

```
>>> print map(lambda x: x * 2, a)
[2, 4, 5]
>>> a = [1, 2, 3, 4]
>>> print filter(lambda x: x %2 == 0, a)
[2, 4]
>>> a = [1, 2, 3, 4]
```

>>> print reduce(lambda x, y: x * y, a)

24

```
map(func, list) 将 func 作用 list 内每
个元素,返回结果组成
的序列
```

filter(func, list) 将 func 作用于 list 内 每个元素,返回令 func 为直的元素组成的序列

reduce(func, list) 通过二元函数 func 将 list 缩减为一个值

sum(list), max(list), min(list) 字面意思

```
>>> a = [1, 2, 3]
>>> print map(lambda x: x * 2, a)
[2, 4, 5]
```

```
>>> a = [1, 2, 3, 4]
>>> print filter(lambda x: x %2 == 0, a)
[2, 4]
```

```
>>> a = [1, 2, 3, 4]
>>> print reduce(lambda x, y: x * y, a)
24
```

Python 一般使用 raw_input 和 print 来标准输入和输出



Vani (Peking University) Play with Python December 11, 2012 19 / 31

输入和输出

Python 一般使用 raw_input 和 print 来标准输入和输出

▶ raw_input() 默认读入一行字符串, 需要使用 int() 函数来强制转化为 整数



Vani (Peking University) Play with Python December 11, 2012 19 / 31

Python 一般使用 raw input 和 print 来标准输入和输出

▶ raw_input() 默认读入一行字符串, 需要使用 int() 函数来强制转化为 整数

```
>>> a = int(raw_input())
>>> b = int(raw_input())
>>> print a + b
1
3
4
```

Python 一般使用 raw input 和 print 来标准输入和输出

- ▶ raw_input() 默认读入一行字符串, 需要使用 int() 函数来强制转化为 整数
- ▶ print 默认在输出后面添加换行

```
>>> a = int(raw_input())
>>> b = int(raw_input())
>>> print a + b
1
3
4
```

Python 一般使用 raw input 和 print 来标准输入和输出

- ▶ raw_input() 默认读入一行字符串, 需要使用 int() 函数来强制转化为 整数
- ▶ print 默认在输出后面添加换行
- ▶ 取消换行:

```
>>> a = int(raw_input())
>>> b = int(raw_input())
>>> print a + b
1
3
4
```

Python 一般使用 raw_input 和 print 来标准输入和输出

- ▶ raw_input() 默认读入一行字符串, 需要使用 int() 函数来强制转化为 整数
- ▶ print 默认在输出后面添加换行
- ▶ 取消换行:

Code >>> a = int(raw_input()) >>> b = int(raw_input()) >>> print a + b 1 3 4 >>> print 3, >>> print 4 3 4

▶ 如何方便地读入一行内空格分割的整数

20 / 31

Trick

▶ 如何方便地读入一行内空格分割的整数

Code

```
>>> a,b,c = map(int, raw_input().split())
1 2 3
>>> print a + b + c
6
```

Vani (Peking University) Play with Python December 11, 2012 20 / 31

▶ 读入两行长度相同的字符串,相同位输出 1,不同位输出 0

Trick

▶ 读入两行长度相同的字符串,相同位输出 1,不同位输出 0

Code

```
>>> print "".join(["0","1"][i == j] for i, j in zip(raw_input(), raw_input()))
abacb
bbabb
01101
```

Vani (Peking University) Play with Python December 11, 2012 21 / 31

▶ 转置一个矩阵

▶ 转置一个矩阵

Code

```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> print zip(*a)
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

Vani (Peking University) Play with Python December 11, 2012 22 / 31

▶ 快速排序

▶ 快速排序

Code

```
>>> def qsort(a):
... return [] if a == [] else qsort(filter(lambda x: x <= a[0], a[1:])) + \
[a[0]] + qsort(filter(lambda x: x > a[0], a[1:]))
>>> print qsort([3, 2, 4, 1, 5])
[1, 2, 3, 4, 5]
```

Vani (Peking University) Play with Python December 11, 2012 23 / 31

▶ 先来围观一个函数:



December 11, 2012

▶ 先来围观一个函数:

```
def foo():
    print "Hello World!"
foo()
```

- ▶ 先来围观一个函数:
- ▶ 怎样得到它的运行时间

foo()

```
def foo():
    print "Hello World!"
```

- ▶ 先来围观一个函数:
- ▶ 怎样得到它的运行时间

```
import time
def foo():
    start = time.clock()
    print "Hello World!"
    end = time.clock()
    print "Time used:", end - start
foo()
```

- ▶ 先来围观一个函数:
- ▶ 怎样得到它的运行时间
- ▶ 无法扩展:(

```
import time
def foo():
    start = time.clock()
    print "Hello World!"
    end = time.clock()
    print "Time used:", end - start

foo()
```

▶ 如果可以有一个获取其他函数运行时间的函数...

▶ 如果可以有一个获取其他函数运行时间的函数...

Code

```
import time
def foo():
    print "Hello World!"

def getTime(func):
    start = time.clock()
    func()
    end = time.clock()
    print "Time used:", end - start

getTime(foo)
```

December 11, 2012

- ▶ 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式

```
import time
def foo():
    print "Hello World!"

def getTime(func):
    start = time.clock()
    func()
    end = time.clock()
    print "Time used:", end - start

getTime(foo)
```

- ▶ 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式
- ▶ 直接获取一个新的函数!

```
import time
def foo():
    print "Hello World!"

def getTime(func):
    start = time.clock()
    func()
    end = time.clock()
    print "Time used:", end - start

getTime(foo)
```

- 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式
- ▶ 直接获取一个新的函数!

```
import time
def getTime(func):
    def wrapper():
        start = time.clock()
        func()
        end = time.clock()
        print "Time used:", end - start
    return wrapper

def foo():
    print "Hello World!"
foo = getTime(foo)

foo()
```

foo()

- 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式
- ▶ 直接获取一个新的函数!
- ▶ Python 的语法糖

import time def getTime(func): def wrapper(): start = time.clock() func() end = time.clock() print "Time used:", end - start return wrapper def foo(): print "Hello World!" foo = getTime(foo)

- 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式
- ▶ 直接获取一个新的函数!
- ▶ Python 的语法糖

Code import time def getTime(func): def wrapper(): start = time.clock() func() end = time.clock() print "Time used:", end - start return wrapper @getTime def foo(): print "Hello World!" foo()

- ▶ 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式
- ▶ 直接获取一个新的函数!
- ▶ Python 的语法糖
- ▶ 良好的扩展性

```
import time
def getTime(func):
    def wrapper():
        start = time.clock()
        func()
        end = time.clock()
        print "Time used:", end - start
    return wrapper
@getTime
def foo():
    print "Hello World!"
foo()
@getTime
def foo2():
    print "Enjoy Python!"
foo2()
```

- ▶ 如果可以有一个获取其他函数运行时间的函数...
- ▶ 要修改函数调用的形式
- ▶ 直接获取一个新的函数!
- ▶ Python 的语法糖
- ▶ 良好的扩展性

Output:

Hello World! Time used: 0.0 Enjoy Python! Time used: 0.0

▶ 修饰器可不可以传递参数?



- ▶ 修饰器可不可以传递参数?
- ▶ 多了一层处理嵌套而已

def deco(arg): def getTime(foo): def wrapper(): start = time.clock() foo() end = time.clock() print arg, "used time:", end - start return wrapper return getTime @deco("foo1") def foo1(): print "Hello World!" @deco("foo2") def foo2(): print "Enjoy Python!" foo1() foo2()

- ▶ 修饰器可不可以传递参数?
- ▶ 多了一层处理嵌套而已
- 实质是面向切面的编程

```
def deco(arg):
    def getTime(foo):
        def wrapper():
            start = time.clock()
            foo()
            end = time.clock()
        print arg, "used time:", end - start
        return wrapper
    return getTime
@deco("foo1")
def foo1():
    print "Hello World!"
@deco("foo2")
def foo2():
    print "Enjoy Python!"
foo1()
foo2()
```

带参数的修饰器

- ▶ 修饰器可不可以传递参数?
- ▶ 多了一层处理嵌套而已
- 实质是面向切面的编程

Code

Output:

Hello World!

foo1 used time: 0.0

Enjoy Python!

foo2 used time: 0.0

numpy 科学计算包,包括一个强大的数组对象 Array 和线性袋鼠,傅立叶变换等常用函数,一般与 scipy 库配套使用

- numpy 科学计算包,包括一个强大的数组对象 Array 和线性袋鼠,傅立叶变换等常用函数,一般与 scipy 库配套使用
 - scipy 科学计算包,包括最优化,线性代数,积分,插值,图像处理,常微分 方程等常用计算的模块

- numpy 科学计算包,包括一个强大的数组对象 Array 和线性袋鼠,傅立叶变换等常用函数,一般与 scipy 库配套使用
 - scipy 科学计算包,包括最优化,线性代数,积分,插值,图像处理,常微分 方程等常用计算的模块
 - re 正则表达式库,内置一个正则表达式引擎,可用于做一些复杂的字符串 分析

- numpy 科学计算包,包括一个强大的数组对象 Array 和线性袋鼠,傅立叶变换等常用函数,一般与 scipy 库配套使用
 - scipy 科学计算包,包括最优化,线性代数,积分,插值,图像处理,常微分 方程等常用计算的模块
 - re 正则表达式库,内置一个正则表达式引擎,可用于做一些复杂的字符串 分析
- urllib, urllib2: 网络编程库,可方便地抓取网页,避免烦琐的 socket 编程

```
# * coding: utf-8 *
from numpy import *
from scipy.linalg import *
from scipy.sparse import *
from scipy.sparse.linalg import svds
word list, row, col, data= [], [], [], []
f_id_word = open("id_word.txt", "r")
for i in f id word:
word_list.append(i.split(',')[0])
f id word.close()
f_data = open("data", "r")
for i in f data:
poem_id, word_id, v = i.split()
row.append(int(word id)-1)
col.append(int(poem id)-1)
data.append(float(v))
a = coo matrix((data, (row, col)))
ret = svds(a. 100)
```

我的作业

▶ 面向对象的编程, 封装继承相关

- ▶ 面向对象的编程, 封装继承相关
- ▶ 闭包

- ▶ 面向对象的编程, 封装继承相关
- ▶ 闭包
- ▶ 魔术方法

- ▶ 面向对象的编程, 封装继承相关
- 闭包
- ▶ 魔术方法

感兴趣的同学可自行查阅相关资料

End.

欢迎提问!