

Play with Python

Introduction to programming with python

Vani

Peking University

December 11, 2012

quiz

猜一下下面程序的作用

quiz

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()))
```

quiz

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))
```

quiz

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))
```

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

quiz

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))
```

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

```
#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');
        else
            putchar('0');
    }
}
```

quiz

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))
```

```
main=interact$([x,y]->zipWith(\a b->if 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');
        else
            putchar('0');
    }
}
```

quiz

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))
```

► Haskell

```
main=interact$\[x,y]->zipWith(\a b->if 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');
        else
            putchar('0');
    }
}
```


quiz

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()))
```

► 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

```
#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');
        else
            putchar('0');
    }
}
```

quiz

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

```
#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');
        else
            putchar('0');
    }
}
```

Features

- ▶ 优势:

Features

- ▶ 优势:
 - ▶ 解释性

Features

- ▶ 优势:
 - ▶ 解释性
 - ▶ 可移植性

Features

- ▶ 优势：
 - ▶ 解释性
 - ▶ 可移植性
 - ▶ 丰富的库

Features

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

Features

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

Features

- ▶ 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

Features

▶ 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

Features

▶ 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

数

Code

数

► 整数

Code

数

- ▶ 整数
- ▶ 长整数

Code

数

- ▶ 整数
- ▶ 长整数

Code

```
>>> print 2**200
```

```
1606938044258990275541962092341162602  
522202993782792835301376
```

数

- ▶ 整数
- ▶ 长整数
- ▶ 浮点数

Code

```
>>> print 2**200
```

```
1606938044258990275541962092341162602  
522202993782792835301376
```


数

- ▶ 整数
- ▶ 长整数
- ▶ 浮点数
- ▶ 复数

Code

```
>>> print 2**200
```

```
1606938044258990275541962092341162602  
522202993782792835301376
```

数

- ▶ 整数
- ▶ 长整数
- ▶ 浮点数
- ▶ 复数

Code

```
>>> print 2**200
```

```
1606938044258990275541962092341162602  
522202993782792835301376
```

```
>>> print (-3+4j)*(1-2j)
```

```
5+10j
```

字符串

Code

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

Code

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

Code

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

Code

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

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

Code

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

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

Code

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


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
- ▶ 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
```

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
- ▶ 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
```

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
- ▶ 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。

Code

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。

- ▶ 乘方运算符 **

Code

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。

- ▶ 乘方运算符 **

Code

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

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.

- ▶ 乘方运算符 **
- ▶ 整除运算符 //

Code

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

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.

- ▶ 乘方运算符 **
- ▶ 整除运算符 //

Code

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

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


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。

- ▶ 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
```

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。

- ▶ 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
```

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。

- ▶ 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
```

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.

- ▶ 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
```

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。

- ▶ 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
2
```

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。

- ▶ 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！

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

>>> a = -10
>>> print a < 0 and -a or a
10
```

缩进

Code

缩进

- ▶ Python 里使用缩进来决定语句的分组, 就像 C++ 里的大括号

Code

缩进

- ▶ 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

Code

缩进

- ▶ 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

Code

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

缩进

- ▶ 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

Code

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

缩进

- ▶ 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

Code

```
>>> 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
```

缩进

- ▶ 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

Code

```
>>> 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
```

缩进

- ▶ 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

Code

```
>>> 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
```

判断语句

Code

判断语句

- ▶ if 的条件判断可以不加括号

Code

判断语句

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif

Code

判断语句

- ▶ if 的条件判断可以不加括号
- ▶ if 后面可以接 elif

Code

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

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

Code

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

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

Code

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

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

Code

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

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

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
```

循环语句

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。

Code

循环语句

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

循环语句

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
```

循环语句

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

循环语句

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

循环语句

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

循环语句

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
```

循环语句

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:

```
1
2
3
```

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

Code

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

Code

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

Code

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

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

Code

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

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
- ▶ 调用时必须加双括号

Code

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

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
- ▶ 调用时必须加双括号
- ▶ 函数内变量均为局部变量，可使用 `global` 关键字声明全局变量

Code

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

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
- ▶ 调用时必须加双括号
- ▶ 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！

Code

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

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
- ▶ 调用时必须加双括号
- ▶ 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

Code

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

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
- ▶ 调用时必须加双括号
- ▶ 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

Code

```
>>> def Double(a):  
...     return a * 2  
>>> print Double(3)  
>>> print Double("kill")  
6  
killkill  
  
>>> def f(a, b, c):  
...     return a + b + c  
>>> print f(*[1, 2, 3])  
6
```

lambda 函数

可通过 lambda 关键字声明匿名函数, 格式为:

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

Code

lambda 函数

可通过 lambda 关键字声明匿名函数, 格式为:

$$\text{lambda } [arg1[, arg2, \dots]] : \text{expression}$$

- ▶ lambda 表达式的返回值即为一个函数

Code

lambda 函数

可通过 lambda 关键字声明匿名函数, 格式为:

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

- ▶ lambda 表达式的返回值即为一个函数

Code

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

Output:
42

lambda 函数

可通过 lambda 关键字声明匿名函数, 格式为:

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

- ▶ 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

lambda 函数

lambda 函数

- ▶ 一个可行的实现:

lambda 函数

- ▶ 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
```

lambda 函数

- ▶ 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 combinator...)

列表

Code

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

Code

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- 列表中各个元素的类型可以不同

Code

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- 列表中各个元素的类型可以不同

Code

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

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- ▶ 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表

Code

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

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- ▶ 列表中各个元素的类型可以不同
- ▶ 可以创建多维列表

Code

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

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

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- ▶ 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

Code

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

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

列表

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- ▶ 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

Code

```
>>> 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

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- ▶ 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

Code

```
>>> 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

列表用于储存一个序列的项目，可用一对方括号来声明，利用下标索引来访问元素

- ▶ 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

Code

```
>>> 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]
```

列表相关函数

Code

列表相关函数

[l:r] 用于获得一个列表的
子序列

Code

列表相关函数

[l:r] 用于获得一个列表的
子序列

Code

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

列表相关函数

[l:r] 用于获得一个列表的
子序列

Code

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

列表相关函数

`[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

Code

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

列表相关函数

[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

Code

```
>>> 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]
```

列表相关函数

`[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

Code

```
>>> 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]
```


列表相关函数

`[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

Code

```
>>> 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
```

列表相关函数

[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

Code

```
>>> 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
```

列表相关函数

`[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

Code

```
>>> 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

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

元组

Code

元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

Code

元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

- ▶ 元组的定义中的圆括号是可以省略的

Code

元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

- ▶ 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

元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

- ▶ 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
```


元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

- ▶ 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

Code

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

元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

- ▶ 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
```

元组

元组与列表十分类似，不过元组是不可添加元素的。元组通过圆括号中用逗号分割的项目定义

- ▶ 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
```

字典

Code

字典

字典类似于 C++ STL 里的 `map`，用于建立关键字到值的映射，声明格式如下：

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

Code

字典

字典类似于 C++ STL 里的 `map`，用于建立关键字到值的映射，声明格式如下：

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

- 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

Code

字典

字典类似于 C++ STL 里的 `map`，用于建立关键字到值的映射，声明格式如下：

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

- ▶ 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：

Code

字典

字典类似于 C++ STL 里的 `map`，用于建立关键字到值的映射，声明格式如下：

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

- ▶ 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：

Code

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


字符串相关函数

Code

字符串相关函数

`s.join(list)` 用于连接字符串

Code

字符串相关函数

`s.join(list)` 用于连接字符串

Code

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

字符串相关函数

`s.join(list)` 用于连接字符串

`s.split(se)` 用于分割字符串

Code

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

字符串相关函数

`s.join(list)` 用于连接字符串

`s.split(se)` 用于分割字符串

Code

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

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

字符串相关函数

`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

Code

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

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

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

序列相关函数

Code

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

Code

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

Code

```
>>> 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

Code

```
>>> 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

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]
```

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

`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]
```

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

`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
```

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

`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

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
```

输入和输出

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

Code

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

Code

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

Code

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

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
- ▶ `print` 默认在输出后面添加换行

Code

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

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
- ▶ `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
```

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
- ▶ `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
```

Trick

- ▶ 如何方便地读入一行内空格分割的整数

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
```

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

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
```


Trick

- ▶ 转置一个矩阵

Trick

► 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)]
```

Trick

- ▶ 快速排序

Trick

► 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]
```

修饰器

- ▶ 先来围观一个函数：

Code

修饰器

- ▶ 先来围观一个函数：

Code

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

修饰器

- ▶ 先来围观一个函数：
- ▶ 怎样得到它的运行时间

Code

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

修饰器

- ▶ 先来围观一个函数：
- ▶ 怎样得到它的运行时间

Code

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

foo()
```


修饰器

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

Code

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

foo()
```

修饰器

- ▶ 如果可以有一个获取其他函数运行时间的函数...

Code

修饰器

- ▶ 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)
```

修饰器

- ▶ 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

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)
```

修饰器

- ▶ 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！

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)
```

修饰器

- ▶ 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！

Code

```
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()
```

修饰器

- ▶ 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

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

foo()
```

修饰器

- ▶ 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()
```


修饰器

- ▶ 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

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()

@getTime
def foo2():
    print "Enjoy Python!"

foo2()
```

修饰器

- ▶ 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

Code

Output:

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

带参数的修饰器

Code

带参数的修饰器

- ▶ 修饰器可不可以传递参数？

Code

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

Code

```
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

```
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
```

常用的库

常用的库

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

常用的库

`numpy` 科学计算包，包括一个强大的数组对象 `Array` 和线性代数，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

常用的库

- `numpy` 科学计算包，包括一个强大的数组对象 `Array` 和线性代数，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

常用的库

`numpy` 科学计算包，包括一个强大的数组对象 `Array` 和线性代数，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

我的作业

Code

```
# -*- 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

Code

```
while (True):
    s = raw_input("please input the query word(\"q\" for quit):\n")
    if (s == "q"): break
    tmp = []
    if (s in word_list):
        pos = word_list.index(s)
        for i in range(0, len(vector)):
            if i != pos:
                tmp.append((get(vector[i], vector[pos]), i))
        tmp.sort()
        k = 5
        for i in range(1, k+1):
            print word_list[tmp[len(tmp) - i][1]]
    else:
        print "this word doesn't exist"encode
```

更高端的 Python

更高端的 Python

- ▶ 面向对象的编程，封装继承相关

更高端的 Python

- ▶ 面向对象的编程，封装继承相关
- ▶ 闭包

更高端的 Python

- ▶ 面向对象的编程，封装继承相关
- ▶ 闭包
- ▶ 魔术方法

更高端的 Python

- ▶ 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！