

# Hello world

2017年1月21日 19:12

## 1.命令行编辑方式

打开cmd

输入python

输入 print 'hello world'

回车

## 2.IDE编辑方式

打开Spyder

新建一个文件

#开头的为行注释

"""三双引号开头和结尾的为块注释

输入 print 'hello world'

保存

点击运行按钮，在控制台看见运行结果

# 对象与类型

2017年1月26日 14:58

## 1.字符串 str

用成对单引号或双引号表示

## 2.整数 int

十进制：21

八进制：025

十六进制：0X15

## 3.浮点数 float

1.48 21.0 21. 2.1E2=210

## 4.布尔型 bool

## 5.复数

1+1j

## 6.查看对象类型的函数 type ('小明' )

结果：<type'str'>

type ( 15 )

结果：<type'int'>

但是type(true)是错误的语法

7.1+1结果是2 , '1' + '1' 结果是 '11'

## 8.CPU中有专门的浮点数运算单元FPU

# 算术运算

2017年1月31日 14:13

## 1.算术运算

$\pi * 3^2$ 在Python中如下表示

$3.14 * 3 * 3$

| 算数运算符 | 含义                    | 举例              |
|-------|-----------------------|-----------------|
| +     | 加法 ( Addition )       | $10 + 20 = 30$  |
| -     | 减法 ( Subtraction )    | $10 - 20 = -10$ |
| *     | 乘法 ( Multiplication ) | $10 * 20 = 200$ |
| /     | 除法 ( Division )       | $10 / 2 = 5$    |
| %     | 求余 ( Modulus )        | $10 \% 3 = 1$   |
| **    | 指数 ( Exponent )       | $2 ** 3 = 8$    |

2.  $5/9=0$

$5.0/9=0.555555$

不同类型变量的算术运算涉及到类型转换

- $\text{bool} \rightarrow \text{int} \rightarrow \text{float} \rightarrow \text{complex}$

- 如：

- $1.0 + 3 = 4.0$

- $\text{True} + 3.0 = 4.0$

## 3.Python的数学函数模块math

Import math , 就可以使用其中相关的数学函数了

dir ( math ) 可以查看math中有哪些函数

```
>>> dir(math)
['_doc_', '_file_', '_name_', '_package_', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

调用方法

Math.sin(100)

使用`help(math.sin)`可以查看`math`中`sin`函数的用法

# 关系运算

2017年1月31日 14:32

1. 关系运算只有两种结果，true or false

2.

| 关系运算符   | 含义                        | 举例                |
|---------|---------------------------|-------------------|
| ==      | 等于 ( equal )              | 10 == 20 is false |
| != , <> | 不等于 ( not equal )         | 10 != 20 is true  |
| >       | 大于 ( greater )            | 10 > 20 is false  |
| <       | 小于 ( less )               | 10 < 20 is true   |
| >=      | 大于等于 ( greater or equal ) | 10 >= 20 is false |
| <=      | 小于等于 ( less or equal )    | 10 <= 20 is true  |

# 逻辑运算

2017年1月31日 14:37

## 1.逻辑运算符

And

Or

Not

A and B or C相当于

(A and B) or C

从左往右

# 增量赋值运算

2017年1月31日 15:14

| 增量赋值       | 等价表示         |
|------------|--------------|
| $x += 2$   | $x = x + 2$  |
| $x -= 2$   | $x = x - 2$  |
| $x *= 2$   | $x = x * 2$  |
| $x /= 2$   | $x = x / 2$  |
| $x \% = 2$ | $x = x \% 2$ |
| $x ** = 2$ | $x = x ** 2$ |

# 运算符优先级

2017年1月31日 14:58

## 1. 运算符优先级

- ❖ 括号：**()**
  - ❖ 一元运算：**+** , **-**
  - ❖ 幂次：**\*\***
  - ❖ 算术运算：**\*** , **/** , **%** , **//**
  - ❖ 算术运算：**+** , **-**
  - ❖ 比较运算：**==** , **!=** , **<>** , **<=** , **>=**
  - ❖ 逻辑非：**not**
  - ❖ 逻辑与：**and**
  - ❖ 逻辑或：**or**
  - ❖ 赋值运算：**=** , **\*=** , **/=** , **+=** , **-=** , **%=** , **//=**
- 规则1：  
自上而下  
括号最高  
逻辑最低
- 规则2：  
自左向右  
依次结合

算术运算>关系运算>逻辑运算



# 变量

2017年1月31日 15:07

## 1.查看某个变量的值

print 变量名

## 2.变量的定义或者命名

( 1 ) 字母数字下划线 , 不能数字开始

( 2 ) 不能是关键字

python关键字如下 :

|        |        |        |          |         |
|--------|--------|--------|----------|---------|
| and    | del    | from   | not      | while   |
| as     | elif   | global | or       | with    |
| assert | else   | if     | pass     | yield   |
| break  | except | import | print    | class   |
| exec   | in     | raise  | continue | finally |
| is     | return | def    | for      | lambda  |
| try    |        |        |          |         |

# 输入输出操作

2017年1月31日 15:21

## 1. raw\_input()

输入的东西都变成字符串

可以采用

```
x=int(raw_input("XXX:"))
```

的形式来得到

XXX :

等待用户输入一个值

## 2.print

将多个字符串组合输出到一行

```
print 'The area for the circle of radius', radius, 'is', area
```

用逗号隔开

## 3.转义符

| 常用转义符 | 含义                   |
|-------|----------------------|
| \n    | 回车 ( Newline )       |
| \t    | 制表符 ( Tab )          |
| \\    | 一个 \                 |
| \a    | 响铃 ( Bell )          |
| \'    | 单引号 ( Single quote ) |
| \"    | 双引号 ( Double quote ) |

# 选择结构

2017年2月1日 15:01

## 1.if语句

```
if score >=60 :  
    print 'Yes'
```

或者

```
if score >=60 :  
    print 'Yes'  
else:  
    print 'No'
```

## 2.嵌套的if语句

```
if score >=60 :  
    if gender == '女' :  
        print 'Yes'
```

相当于

```
if score >=60 and gender == '女' :  
    print 'Yes'
```

## 3.一类特殊的选择语句

|    |           |
|----|-----------|
| 1. | if x > 0: |
| 2. | y = 1     |
| 3. | else:     |
| 4. | y = -1    |

等价于:

|    |                        |
|----|------------------------|
| 1. | y = 1 if x > 0 else -1 |
|----|------------------------|




# 多分支结构

2017年2月1日 15:27

## 1. 多个if-else嵌套

```
7 score = 78
8
9 if score >= 90:
10     print 'A'
11 else:
12     if score >= 80:
13         print 'B'
14     else:
15         if score >= 70:
16             print 'C'
17         else:
18             if score >= 60:
19                 print 'D'
20             else:
21                 print 'E'
```

## 2. elif的使用，相当于else : if

|   |   |  |
|---|---|--|
| <pre>7 score = 78 8 9 if score &gt;= 90: 10     print 'A' 11 else: 12     if score &gt;= 80: 13         print 'B' 14     else: 15         if score &gt;= 70: 16             print 'C' 17         else: 18             if score &gt;= 60: 19                 print 'D' 20             else: 21                 print 'E'</pre> |  | <pre>23 if score &gt;= 90: 24     print 'A' 25 elif score &gt;= 80: 26     print 'B' 27 elif score &gt;= 70: 28     print 'C' 29 elif score &gt;= 60: 30     print 'D' 31 else: 32     print 'E'</pre> |
|---|---|--|

elif-else语句：

## 3. 求二次方程的根的代码示例

```
8 import math
9
10 a = float(raw_input('Input a: '))
11 b = float(raw_input('Input b: '))
12 c = float(raw_input('Input c: '))
13
14 if a == 0:
15     print 'The equation is linear, not quadratic'
16 else:
17     delta = b ** 2 - 4 * a * c
18     if delta < 0:
19         print 'No real roots!'
20     elif delta == 0:
21         print 'Only one root is ', -b / (2 * a)
22     else:
23         root = math.sqrt(delta)
24         s1 = (-b + root) / (2 * a)
25         s2 = (-b - root) / (2 * a)
26
27     print 'Two distinct solutions are: ', s1, s2
28
```



# 循环结构

2017年2月1日 15:54

## 1.while循环

- ❖ 循环体外设定循环可执行的初始条件
  - ❖ 书写需重复执行的代码（循环体）
  - ❖ 设定循环条件并在循环体内设定条件改变语句
- ❖ 打印字符串 5 次

```
count = 0
9
10 while count < 5:
11     print 'Programming is fun!'
12     count += 1
13
```

## 2.累加程序

```
8 s = 0
9 i = 1
10
11 while i < 10:
12     s += i
13     i += 1
14
15 print 'sum is ', s
16
```

## 3.死循环的终止方式：Ctrl+C

## 4.break和continue

```
8 count = 0
9
10 while count < 5:
11     if count > 2:
12         break
13     print 'Programming is fun!'
14     count += 1
15
```

**break** 结束当前循环体

```
8 count = 0
9
10 while count < 5:
11     count += 1
12     if count > 2:
13         continue
14     print 'Programming is fun!'
15
```

**continue** 结束当次循环

## 5.for循环

- ❖ 计算 $1+2+3+...+10$ 的值

```
8 s = 0
9
10 for i in range(11):
11     s += i
12
```

range 函数生成 0, 1, ..., 10 序列

## ❖ 计算 $1+2+3+\dots+10$ 的值

```
8 s = 0
9
10 for i in range(11):
11     s += i
12
13 print 'sum is:', s
14
```

range 函数生成 0, 1, ..., 10 序列

这里的 i 每次接循环结尾默认加1

## 6.range()函数，生产一个等差序列

- ❖ `range(2, 10)` → [2, 3, 4, 5, 6, 7, 8, 9]
- ❖ `range(2, 10, 3)` → [2, 5, 8]
- ❖ `range(10, 2, -1)` → [10, 9, 8, 7, 6, 5, 4, 3]

序列包含初始值，但不包含结束值

也可以`range ( stop )` 只有一个参数，就是结束值

如`range(4)`则生成0,1,2,3

## 7.for循环例子

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots + \frac{1}{i!}$$

```
8 import math
9
10 e = 1
11
12 for i in range(1, 100):
13     e += 1.0 / math.factorial(i)
14
15 print 'e is', e
16
```

```
8 e = 1
9 factorial = 1
10
11 for i in range(1, 100):
12     factorial *= i
13     e += 1.0 / factorial
14
15 print 'e is', e
16
```

## 8.循环嵌套



## ❖ 打印乘法表

|   |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|
| 1 | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
| - | -  | -  | -  | -  | -  | -  | -  | -  |
| 1 | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
| 2 | 4  | 6  | 8  | 10 | 12 | 14 | 16 | 18 |
| 3 | 6  | 9  | 12 | 15 | 18 | 21 | 24 | 27 |
| 4 | 8  | 12 | 16 | 20 | 24 | 28 | 32 | 36 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 |

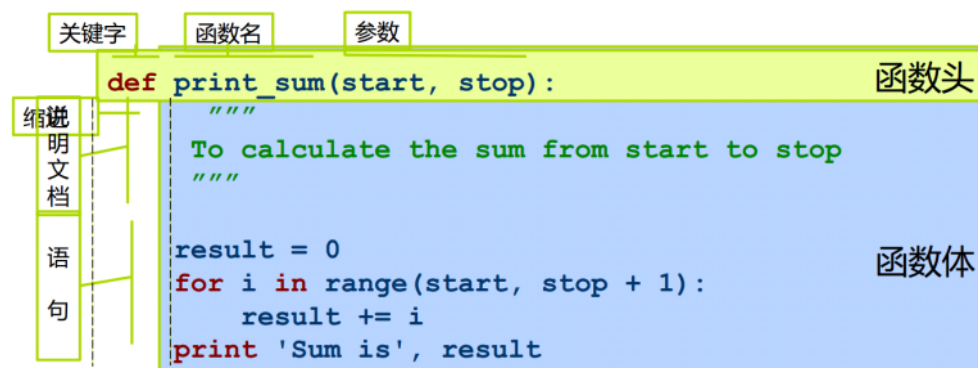
```
8 for i in range(1, 10):
9     for j in range(1, 10):
10         print format(i * j, '3'),
11     print
12
```

p-;

# 函数

2017年2月8日 13:56

## 1. 函数定义方式



例如：

### ❖ 定义函数

```
def print_sum(start, stop):  
    result = 0  
    for i in range(start, stop + 1):  
        result += i  
    print 'Sum is', result
```

**形式参数 (形参, parameter)** points to the parameters `start, stop` in the function definition.

### ❖ 调用函数

```
print_sum(1, 10)
```

**实际参数 (实参, argument)** points to the arguments `1, 10` in the function call.

## 2. 缺省值

```
def defaultParameters(arg1, arg2=2, arg3=3):  
    print 'arg1=', arg1  
    print 'arg2=', arg2  
    print 'arg3=', arg3
```

若调用为 `defaultParameters(10)`

则 `arg1=10, arg2=2, arg3=3`

若调用为 `defaultParameters(10,10)`

则 `arg1=10, arg2=10, arg3=3`

若调用为 `defaultParameters(10,10,10)`

则 `arg1=10, arg2=10, arg3=10`

### 3.有返回值的函数-用return

```
def sum(start, stop):  
    result = 0  
    for i in range(start, stop + 1):  
        result += i  
  
    return result
```

- 函数调用完成后，返回数据
- return语句终止当前函数的执行
- return后的语句将被忽略

一个函数只能用一次return,当执行了return就说明函数结束了，不管下面是否还有语句

### 4.作用域

函数内部定义的为局部变量，是不能再函数外调用的  
局部变量和全局变量相同的话，那么函数内部操作的是局部变量

要想在函数内部对外面的全局变量进行修改，那么在函数内部再次用

Global x对x再定义一次

```
x = 1  
  
def increase():  
    global x  
    x = x + 1  
    print x  
  
increase()  
print x
```

注意，这里的 increase()写成increase(x),那么程序出错，原因是x既是局部变量又是全局变量

# 递归函数

2017年2月9日 13:04

## 1.递归概念

函数内部调用了自身

## 2.例子——求N！

```
def p(n):  
    if n == 1 or n == 0:  
        return 1  
    else:  
        return n * p(n-1)  
  
n = int(raw_input("请输入一个整数:"))  
print n, " !的值为 :", p(n)
```

## 3.递归函数的常用形式

|   |      |
|---|------|
| def p(n):<br>if n == 1 or n == 0:<br>return 1 | 初始条件 |
| else:<br>return n * p(n-1)                    | 递归   |

掐头去尾留中间

## 4.斐波那契数列

斐波那契数列：1, 1, 2, 3, 5, 8, 13, 21.....

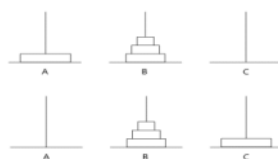
|   |   |      |
|---|---|------|
| def fib(n):<br>if n == 1 or n == 2:<br>return 1 | $f(n) = \begin{cases} 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ f(n-1) + f(n-2) & \text{if } n > 2 \end{cases}$ | 初始条件 |
| else:<br>return fib(n-1) + fib(n-2)             |   | 递归   |

## 5.汉诺塔

❖ 定义函数hanoi(n, A, B, C)表示把A上的n个盘子移动到C上，其中可以用到B

```
def hanoi(n, A, B, C):  
    if n == 1:  
        print "Move disk ", n, " from ", A, " to ", C  
    else:  
        hanoi(n-1, A, C, B)  
        print "Move disk ", n, " from ", A, " to ", C  
        hanoi(n-1, B, A, C)
```

```
n = int(raw_input("请输入一个整数: "))  
hanoi(n, '左', '中', '右')
```



## 6.随机函数的使用

```
import random
```

```
Random.uniform(a,b)
```

产生从a到b的一个随机数

# 字符串基础

2017年2月19日 11:06

## 1.字符串表示

用成对的单引号和双引号

也可以用成对三引号，这样的话，全部的格式信息都会保留

## 2.字符串基本运算

( 1 ) len()函数——获得字符串长度

( 2 ) 拼接+： 'A'+'B'='AB'

( 3 ) 重复\*： 'A'\*3='AAA'

( 4 ) in:判断一个字符串是否是另一个字符串的子串，大小写敏感

( 5 ) for语句：实现枚举

```
>>> my_str = 'hello world'
>>> for char in my_str:
...     print char
...
h
e
l
l
o

w
o
r
l
d
```

## 3.字符串支持in的操作

If c in 'aeiouAEIOU':

.....

# 字符串索引

2017年2月19日 11:26

## 1.前向索引和后向索引

|                |     |     |    |    |    |    |    |    |    |    |    |
|----------------|-----|-----|----|----|----|----|----|----|----|----|----|
| forward index  | 0   | 1   | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|                | h   | e   | l  | l  | o  |    | w  | o  | r  | l  | d  |
| backward index | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

调用索引的方式

字符串变量名[索引号]

## 2.切片--[start:finish]

对于s='hello world'

s[7:10]为：'orl'

s[:10]为：'hello worl'

s[0:10:2]为：'hlowr'

求元字符串的逆：s[::-1]

## 3.字符串不可变，不能通过

s[1]=a,这样对其中一个改变值的方法是不行的

可以对整个变量重新赋值来改变

## 4.replace(old,new)方法

s='hello world'

s=s.replace('e','a')

则s为'hallo world'

这是将所有的都会替换

## 5.find:返回字符串中某个字符第一次出现的索引值

Find('l'):得到2

6.split ( ) : 字符串切分

s.split,则将s以空格切分

7.index ( )

检查字符串中是否含有某个子串

```
str.index(str, beg=0, end=len(string))
```

str为子串内容，beg为开始位置索引，end为结束位置索引



# 人名游戏

2017年2月20日 9:48

## 1.文件操作

打开文件

```
f = open(filename, mode)
```

- mode 有 r ( 读 , 默认 ) , w ( 写 ) 等

按行读取文件内容

```
for line in f:
```

```
    pass
```

关闭文件

```
f.close()
```

写文件

```
f.write(str)
```

## 2.strip()方法

去掉一个字符串开头或者结尾的空格、回车等

```
s=s.strip()
```

## 3.title ( ) 方法

将一个字符串转换为首字母大写 , 其他字母小写的形式

# 字符串比较

2017年2月20日 10:07

## 1.比较方法

比较首字母，ASCII大的就大，首字母相同就往后比较，空格最小

## 2.字符串格式化--Format()

format 方法，如：

```
>>> print "Hello {} good {}".format(5, 'DAY')
Hello 5 good DAY.
```

括号的格式

- {field name:align width.precision type}

```
>>> print math.pi
3.14159265359
>>> print 'PI is {:.4f}'.format(math.pi)
PI is 3.1416
>>> print 'PI is {:.9.4f}'.format(math.pi)
PI is 3.1416
>>> print 'PI is {:e}'.format(math.pi)
PI is 3.141593e+00
```

## 3.正则表达式

.表示任意字符

\d+表示一系列数字

[a-z]表示任意一个小写字母

**判断一个人名是否含有 C.A 模式**

```
import re

f = open('names.txt')

pattern = '(C.A)'

for line in f:
    name = line.strip()
    result = re.search(pattern, name)
    if result:
        print 'Find in {}'.format(name)

f.close()
```

注意：要调用re这个头文件



# 字符串其他方法

2017年3月3日 16:09

## 1.eval ( ) 方法——字符串计算方法

eval('2\*3+4')返回10

# 列表

2017年2月25日 12:28

## 1.列表概念

各种类型的数据的组合

例如：`lst=[5.4,'hello',2]`

支持索引

如：`lst[1]='hello'`

## 2.列表与字符串相同功能

( 1 ) 索引

( 2 ) 拼接和重复

( 3 ) 成员 in操作

( 4 ) 求长度len ( )

( 5 ) 循环 for操作

但是内容可变

注意：

若`a=[1,2,3]`

`b=a`

则b和a都指向这个列表，通过b改变和通过a改变是一样的。

## 3.列表的方法

```
my_list[0] = 'a'  
my_list[0 : 2] = [1.2, 3, 5.6]  
my_list.append(), my_list.extend() #追加元素  
my_list.insert() #任意位置插入元素  
my_list.pop(), my_list.remove() #删除元素  
my_list.sort() #排序  
my_list.reverse() #逆序
```

例如

Lst.append('abc'):在原来lst的后面加上 ‘abc’

Lst.extend(['abc','def'])：在原来列表后面增加两个元素

Lst.insert(3,'abc')在原来列表索引为3的地方插入 ‘abc’

Lst.pop():删除最后一个元素，并将其返回

Lst.pop(3):删除索引值为3的元素，并返回

Lst.remove('abc')：删除lst中的元素'abc'

Lst.sort():先数字后字符串，从小到大的顺序排列

Lst.reverse():与sort相反，进行逆序排列

#### 4.列表内建函数

Sum(lst),max(lst),min(lst)

# 列表赋值与查找

2017年2月25日 13:24

## 1. 一个列表赋值给另外一个列表

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

```
( 1 ) b=a
```

则b并不是一个新的列表，b与a指向同一个列表

```
( 2 ) b=a[:]
```

将a中全部元素赋值给b，那么b就相当于一个跟a一样的新的列表

## 2. 列表作为函数参数

因为列表名相当于指针，所以当直接用列表名作为函数参数时，在函数内对列表的修改就会改变列表的实际内容  
但是通过以上方式，只能改变指针所指地址的内容，并不能改变指针所指的方向，也就是说直接在函数内部

a=[1,2,3]，对原来的列表是不起作用的

## 3. 列表查找方法：lst.index()

输入参数为要查找的元素

返回参数为所找元素第一次出现的索引值

## 4. 二分查找—前提是列表事先排序

```
def bi_search(lst, v):  
    low = 0  
    up = len(lst) - 1  
  
    while low <= up:  
        mid = (low + up) / 2  
        if lst[mid] < v:  
            low = mid + 1  
        elif lst[mid] == v:  
            return mid  
        else:  
            up = mid - 1  
  
    return -1
```

**时间复杂度：O(log<sub>2</sub> n)**



# 时间复杂度

2017年2月25日 13:47

## 1.量化一个算法运行的时间

用O表示，只保留高阶项

$$4n + 4 = O(n)$$

$$137n + 271 = O(n)$$

$$n^2 + 3n + 4 = O(n^2)$$

$$2^n + n^3 = O(2^n)$$

# 排序

2017年2月26日 10:21

## 1.选择排序

### (1) 方法1

每次将最小的元素找出来，放在第一个位置，再找后面最小的元素，放在第二个位置，以此类推

最小的元素找出来后删除并插到第一个位置的操作：

```
lst.insert(0,lst.pop(min))
```

### (2) 方法2

每次找出最小的元素与第一个进行交换，第二小的与第二个进行交换

时间复杂度

## 总运行时间

- $n + (n - 1) + \dots + 2 + 1$

## 时间复杂度

- $O(n^2)$

## 2.冒泡排序

前后两个比较，若前面比后面大，就交换，这样一轮走下来，就将最大的元素放在了最后。多次这样的循环就实现了排序

不过需要设置一个变量，来判断是否已经排好序

```
def bubble_sort(lst):
    exchanged = True
    top = len(lst) - 1
    while exchanged:
        exchanged = False
        for i in range(top):
            if lst[i] > lst[i+1]:
                swap(lst, i, i+1)
                exchanged = True
        top -= 1
```

### 3.用于排序的内建函数

#### ( 1 ) sorted()函数

Sorted(a)返回对a排序的结果，但是原来的a并没有改变

#### ( 2 ) a.sort()

对a进行排序，a被改变

Sort()函数的参数使用

```
def f(a):
    return a[1]

students.sort(key = f, reverse = True)

print students
```

这样就实现了对学生按照分数从高到低排序

# 嵌套列表

2017年2月26日 10:44

## 1.列表内包含列表

```
lst=[[1,2,3],[a,b,c],[4,5,6]]
```

## 2.列表的解析或推导

如生成值为  $\{x^2 : x \in \{1 \dots 9\}\}$  的列表

可以用如下方法

```
lst = [x**2 for x in range(1, 10)]
```

其他实例：

### 列表推导实现求平均分

- `sum([x[1] for x in students]) / len(students)`

### 使用列表解析对所输入数字 x 的因数求和

- 如：如果输入 6，应该显示12，即  $1 + 2 + 3 + 6 = 12$
- `sum([i for i in range(1, x + 1) if x % i == 0])`

## 3.匿名函数 lambda

定义方式：`g=lambda x :x**2`

即输入参数为x，输出参数为x的平方

调用方式: `g(8)`,返回64

运用于排序

```
students.sort(key = lambda x: x[1], reverse=True)
```

# 元组

2017年2月27日 8:29

## 1.什么是元组

元组即不可变列表，列表的除了改变内容的方法不能对元组使用外，其他方法均可用

## 2.元组创建方法

可以用括号，也可以不用

```
my_tuple = 1, 'a', 3.14, True
my_tuple = (1, 'a', 3.14, True)
```

## 3.元组赋值

**交换两个值**

```
temp = a
a = b
b = temp
```

**或者**

```
a, b = b, a
```

**如切分一个邮件地址**

```
name, domain = 'car@hit.edu.cn'.split('@')
```

将这个域名以@为界进行切片，然后赋值给元组

## 4.函数返还值也可以是一个元组

Return max,min

## 5.DSU模式：装饰、排序、反装饰

```
def sort_by_length(words):  
    # decorate  
    t = []  
    for word in words:  
        t.append((len(word), word))  
  
    # sort  
    t.sort(reverse = True)  
  
    # undecorate  
    res = []  
    for length, word in t:  
        res.append(word)  
  
    return res
```

先在列表t中存放单词和自身长度组成的元组

再根据单词自身长度对元组列表排序

最后在res列表中存放排序后的单词，除去自身长度

以上代码等同于

```
words.sort(key = lambda x: len(x), reverse = True)
```

# 字典 ( dict )

2017年3月3日 18:41

## 1.什么是字典

字典是一系列键值对

如

| 姓名 ( 键 ) | 电话号码 ( 值 ) |
|----------|------------|
| John     | 86411234   |
| Bob      | 86419453   |
| Mike     | 86412387   |
| .....    | .....      |

## 2.创建、访问字典方式

### 创建字典

- 使用 `{ }` 创建字典
- 使用 `:` 指明 键:值 对
  - `my_dict = {'John': 86411234, 'Bob': 86419453, 'Mike': 86412387}`
- 键必须是不可变的且不重复，值可以是任意类型

### 访问字典

- 使用 `[ ]` 运算符，键作为索引
  - `print my_dict['Bob']`
  - `print my_dict['Tom']` #WRONG!

注意，实际生成的字典内，键值对是无序的，并不像我们输入的顺序一样

键可以使用元组，但是不能使用列表

## 3.给字典添加键值对

增加一个新的对

- `my_dict['Tom'] = 86417639`

## 4.字典的常用方法

**len(my\_dict)**

- 字典中键-值对的数量

**key in my\_dict**

- 快速判断 key 是否为字典中的键：O(1)
- 等价于 my\_dict.has\_key(key)

**for key in my\_dict:**

- 枚举字典中的键，注：键是无序的

**更多的方法**

- my\_dict.items() – 全部的键-值对
- my\_dict.keys() – 全部的键
- my\_dict.values() – 全部的值
- my\_dict.clear() – 清空字典

## 5.判断一个字符串中每个字符的次数

```
count = {}
for i in 'abcdad':
    if i in count:
        count[i] += 1
    else:
        count[i] = 0
```



# 小说中最常见的10个单词

2017年3月4日 19:21

```
f = open('emma.txt')
word_freq = {}
for line in f:
    words = line.split() # split a line by blanks
    for word in words:
        if word in word_freq:
            word_freq[word] += 1
        else:
            word_freq[word] = 1

word_freq_lst = []
for word, freq in word_freq.items():
    word_freq_lst.append((freq, word))

word_freq_lst.sort(reverse = True)

for freq, word in word_freq_lst[:10]:
    print word, freq
```

# 翻转字典

2017年3月4日 19:24

```
def invert_dict(d):  
    inverse = {}  
    for key in d:  
        val = d[key]  
        if val in inverse:  
            inverse[val].append(key)  
        else:  
            inverse[val] = [key]  
    return inverse
```

# 集合 ( set )

2017年3月4日 19:25

## 1.基本概念和基本操作

### 集合

- 无序不重复元素 ( 键 ) 集
- 和字典类似，但是无 “值”

### 创建

- `x = set()`
- `x = {key1, key2, ...}`

### 添加和删除

- `x.add('body')`
- `x.remove('body')`

### 集合的运算符

| 运算符            | 含义  |
|----------------|-----|
| -              | 差集  |
| &              | 交集  |
|                | 并集  |
| !=             | 不等于 |
| ==             | 等于  |
| in             | 成员  |
| for key in set | 枚举  |

## 2.中文分词的例子

```
def load_dic(filename):
    f = open(filename)
    word_dic = set()
    max_length = 1
    for line in f:
        word = unicode(line.strip(), 'utf-8')
        word_dic.add(word)
        if len(word) > max_length:
            max_length = len(word)
    return max_length, word_dic

def fmm_word_seg(sentence, word_dic, max_length):
    begin = 0
    words = []
    sentence = unicode(sentence, 'utf-8')

    while begin < len(sentence):
        for end in range(min(begin + max_length, len(sentence)),
                           begin, -1):
            word = sentence[begin:end]
            if word in word_dic or end == begin + 1:
                words.append(word)
                break
        begin = end
    return words

max_len, word_dic = load_dic('lexicon.dic')
words = fmm_word_seg(raw_input(), word_dic, max_len)
for word in words:
    print word
```

# 对象 ( class )

2017年3月4日 19:50

## 1.对象-用户自定义类型

```
class Point(object):  
    """Represents a point in 2-D space."""
```

用class的方法就创建了一个用户自定义的类，即对象，就像定义函数一样去定义对象

## 2.对象实例化 ( instance )

```
blank=Point ( )
```

## 3.对象属性-对象定义时的内部变量

```
>>> blank.x = 3.0  
>>> blank.y = 4.0
```

也可以用相同的方法读出一个实例的属性值

```
>>> print blank.y  
4.0  
>>> x = blank.x  
>>> print x  
3.0
```

## 4.实例可以作为参数传递

```
>>> print_point(blank)  
(3.0, 4.0)
```

## 5.对象编程实例

```
import math
```

```
class Point(object):  
    x=0  
    y=0
```

```
def dis(Point1,Point2):  
    s=0  
    s=math.sqrt((Point1.x-Point2.x)**2+(Point1.y-Point2.y)**2)  
    return s  
  
P1=Point()  
P2=Point()  
P1.x=3  
P1.y=4  
P2.x=6  
P2.y=8  
  
print dis(P1,P2)
```

# 对象复制

2017年3月4日 20:29

## 1.copy()函数的使用

需要调用copy模块

```
>>> p1 = Point()
>>> p1.x = 3.0
>>> p1.y = 4.0

>>> import copy
>>> p2 = copy.copy(p1)
```

这时候，p1和p2具有相同的属性值，但并不是同一个对象，即彼此独立

```
>>> p1 is p2
False
>>> p1 == p2
False
```

## 2.copy并不复制对象内部嵌套的对象

即复制后，两个对象内部嵌套的对象仍旧指向同一个地方，没有隔离

```
>>> box2 = copy.copy(box)
>>> box2 is box
False
>>> box2.corner is box.corner
True
```

这叫做浅复制。

## 3.deepcopy方法，实现深度复制

```
>>> box3 = copy.deepcopy(box)
>>> box3 is box
False
>>> box3.corner is box.corner
False
```

# 调试

2017年3月4日 20:36

## 1.询问不确定对象的类型

`type(p)`

## 2.判断一个对象是否拥有某种属性

```
>>> hasattr(p, 'x')
```

```
True
```

```
>>> hasattr(p, 'z')
```

```
False
```

第一个参数可以是任何对象； 第二个参数是一个字符串，代表了某个属性的名字。

## 3.\_\_dict\_\_属性

这个属性是对象中其他属性的属性名和值形成的字典

```
>>> p = Point(3, 4)
```

```
>>> print p.__dict__
```

```
{'y': 4, 'x': 3}
```

```
def print_attributes(obj):
```

```
    for attr in obj.__dict__:
```

```
        print attr, getattr(obj, attr)
```

这个函数实现对字典\_\_dict\_\_的遍历打印

`getattr ( )` 返回的是字典中attr键对应的值



# 类和函数

2017年3月6日 9:56

## 1.对象作为函数的调用方式

```
class Time(object):
    """Represents the time of day.

    attributes: hour, minute, second
    """

    def print_time(t):
        print '%.2d:%.2d:%.2d' % (t.hour, t.minute, t.second)

time=Time()
time.hour = 11
time.minute = 59
time.second = 30

print_time(time)
```

## 2.纯函数

```
def add_time(t1, t2):
    sum = Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second
    return sum
```

这样只是单纯地调用输入对象的属性，对输入对象本身不具有任何影响的函数成为纯函数

## 3.过程

```
def increment(time, seconds):
    time.second += seconds

    if time.second >= 60:
        time.second -= 60
        time.minute += 1

    if time.minute >= 60:
        time.minute -= 60
        time.hour += 1
```

这个函数是直接对被调用的对象进行修改的，称为过程

#### 4.利用assert语句检验是否正确

```
def add_time(t1, t2):  
    assert valid_time(t1) and valid_time(t2)  
    seconds = time_to_int(t1) + time_to_int(t2)  
    return int_to_time(seconds)
```

只有当assert后面的条件是true时，才能往下执行，否则报错

# 类和方法

2017年3月7日 18:29

## 1.什么是方法

方法就是针对特定对象，具有具体含义的函数

方法是在类内部定义的函数

## 2.方法的定义方式及调用方式

定义如下：

```
class Time(object):  
    def print_time(time):  
        print '%.2d:%.2d:%.2d' % (time.hour, time.minute, time.second)
```

调用方法1：

```
>>> Time.print_time(start)  
09:45:00
```

调用方法2：

```
>>> start.print_time()  
09:45:00
```

在方法中，主语被赋值为第一个参数，所以在这里`start`被赋值到`time`上了。

## 3.约定方法的第一个参数写作self

所以上面的方法定义改写成如下形式

```
class Time(object):  
    def print_time(self):  
        print '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
```

4.注意，方法中实例本身始终为第一个参数，所以当参数不为实例本身时，相当于该方法调用了两个参数

```
end = start.increment(1337)  
end.print_time()
```

这里的`start.increment(1337)`输入参数为`start`和`1337`两个参

数

## 5.init方法

对象初始化的时候调用

全名为\_\_init\_\_

一个Time类的init方法应该这样定义

```
inside class Time:

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
```

这样的话，如果将对象实例化的时候，如果没有给实例的属性赋值，那么各个属性就是初始化的结果

可以在实例化的同时给属性赋值，这个值到底赋给哪个属性，就看init中的参数顺序

```
>>> time = Time (9)
>>> time.print_time()
09:00:00
```

上面的9自动赋值给了hour

## 6.\_\_str\_\_方法

返回对象的字符串表达

例：

```
# inside class Time:

    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
```

不需要使用Time.str()

当调用print时，即相当于调用了这个方法

```
>>> time = Time(9, 45)
>>> print time
09:45:00
```

一般写一个类，前两个方法就是\_\_int\_\_和\_\_str\_\_

# 运算符重载

2017年3月8日 15:31

## 1.加法重载

```
# inside class Time:

    def __add__(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)

>>> start = Time(9, 45)
>>> duration = Time(1, 35)
>>> print start + duration
11:20:00
```

注意这里之所以定义`__add__`这样的方法就可以实现+的重载，是因为Python中语法的特殊规定。

## 2.isinstance(实例名，对象名)

判断某个实例是否是该对象的实例

## 3.加法重载时无法实现交换律的问题

调用`__radd__`

```
def __radd__(self, other):
    return self.__add__(other)
```

这样无论`1337+time` 还是`time+1337`都能实现了

## 4.\_\_cmp\_\_可以比较两个对象大小

前者大于后者则返回正数

相等返回0

后者大于前者则返回负数



# 类型分发

2017年3月8日 15:43

## 1.类型分发

根据输入的参数类型来实现不同的运算

```
# inside class Time:

    def __add__(self, other):
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def add_time(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)

    def increment(self, seconds):
        seconds += self.time_to_int()
        return int_to_time(seconds)
```

根据输入的other的类型，决定调用哪种具体的方法去实现



# 多态

2017年3月8日 15:47

## 1.什么是多态

类型分发是A函数根据输入参数的类型，决定调用B函数还是C函数来解决问题

多态则是在一个函数里判断类型后直接进行不同的处理，而不是去调用其他方法

# 接口和实现

2017年3月8日 15:57

## 1.信息隐藏

对象的属性不能直接访问，应该通过方法才能实现对对象属性的读取和改变

# 继承

2017年3月8日 16:03

## 1.什么是继承

在已经存在了的类基础上定义一个新的类

## 2.定义子类方式

```
class Hand(Deck):  
    """Represents a hand of playing cards."""
```

在父类（ deck ）基础上定义子类（ hand ）

这样的话，hand里面就算不写什么方法，也可以直接调用deck的方法，但是我们不希望hand调用deck的初始化方法，所以需要重写hand自己的\_\_init\_\_方法

## 3.方法覆盖注意点

子类中需要重写父类的方法，那么尽量名字和参数保持一致

# 常用操作

2017年7月26日 13:00

## 1.查看变量类型

`print type ( 变量名 )` 即可显示变量类型

`type`这个结构体中存储着各种变量类型

# 各类英文名称

2017年7月26日 13:03

## 1.常用英文名称

tuple：元组

# 常见数据类型的使用

2017年7月26日 13:21

## 1.创建字典

( 1 )

```
avgs[digit] = darknesses[digit] / n
```

( 2 )

```
distances = {k: abs(v - darkness) for k, v in  
avgs.iteritems() }
```

2.