目录

本书收集历时一年多时间，此电子书可以免费使用，如果您觉得此电子书对您有帮助，可以小额赞助，以便我后期更好地完善此书。

也可以加入到 QQ 群（118140582），有问题可以互相交流，备注请写：Python 基础教程。

说明: 本书基于《byte of python》 电子书及网络材料,结合自身测试,旨在抛砖引玉

python 与脚本语言

首先我们知道，脚本与脚本解释器这两个概念是统一的、密不可分的。
脚本本质上就是文本文件（如用 ASCII 码所写的），这种文本文件用文本编辑器打开与普通的文本文件没有区别，只不过扩展名不是".txt"，python 脚本的扩展名通常是".py"。计算机仅有脚本还是不能工作（它只是文本文件），我们需要一种工具软件，用来把脚本中的代码解释成计算机指令，同时让计算机立即执行这些指令，这种工具就是**脚本解释器**。很显然，使用不同语法规则编写的脚本应使用不同的解释器来解释执行，因为每种脚本实际上是专为其解释器而写的，是按照解释器能识别的语法规则而写的。python 是众多"脚本解释器+语法规则"的一种，类似还有 bash、perl、javascript 等等。

Python 是高层次的面向对象的编程语言。优点是代码简洁、简单易学，缺点是运行速度较慢。我们可以在 python 代码中调用 C 编译生成的动态链接库来提升程序运行速度。很多发布的 linux 系统都自带了 python 解释器，因此 python 在 linux 平台上应用广泛，很适合 GUI（图形界面）、测试程序等各类软件的开发。建议学习时把书上提供的示例代码上机操作一遍，动手写 python 代码可以迅速掌握 python。

学习摘要

1. 两种 python 执行方式（Linux, Mac OS 下）

a. 进入 python 的 shell 下逐行执行：
在终端（Terminal）下键入"python"按回车（解释器被启动），进入 python 的 shell，此时可以输入 python 语句，按回车执行。按"左侧 ctrl + D"退出 python 的 shell。

b. 在终端下直接调用 python 脚本执行：
在终端命令行进入相应目录，运行"python hello.py"；或者可以在脚本文件首行指定解释器，终端命令行只需输入"./hello.py"调用，此方式见下文"创建并执行脚本文件"。

【超级终端实际就是一种 shell，linux 默认指定为 bash。】

2. 创建并执行脚本文件（windows 下学习 看本节最后的备注即可）

a. 创建一个名为"hello.py"的脚本文件(黄色背景表示脚本的内容):

```
[root@Linux ~]    #vi hello.py
#! /usr/bin/python
print "hello, world"
```

b. 赋予脚本文件可执行权限:

```
[root@Linux ~] #chmod a+x hello.py
```

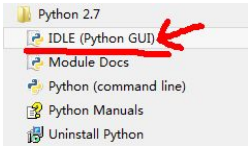
c. 执行脚本文件:

```
[root@Linux ~] #./hello.py
```

【超级终端下会显示："hello, world"】

d. 步骤说明
在文件开头写" #!/usr/bin/python"，这是指定该脚本使用的解释器所在的目录。因为 python 的默认安装目录是"/usr/bin/python"，所以这里如此指定，真实情况要按照 python 的实际安装目录来指定。也可以不在文件开始处指出解释器，这种情况下应在终端命令行中指定解释器，如下：

[root@Linux ~] #python ./hello.py 或 python hello.py



备注：在 winxp,win7,win8 下也是一样的，
先安装 python，在 [开始] -> [所有程序] 下找 python2.7, IDLE
点击 File, new window 可以像文本一样编写程序，按 F5 运行

3. python 中的缩进

缩进是**语法规则相关的**，它决定语句的层次（类似 C, Java 等语言中用"{}"来表示层次一样）。同一层次语句必须有相同缩进，否则解释器会报错。Python 函数没有明确的开始（begin）或者结束（end），也没有用大括号来标记函数从哪里开始从哪里停止。唯一的定界符就是一个冒号(:)和代码自身缩进。例如：

```
def function_name():
    I = 3
    print "number is",   i
```

4. 字符串的使用

1. 单引号与双引号在 python 中等效使用

a. 三引号（'''或"""）可用来写跨行的字符串，在其中，单、双引号均可使用。

b. Python 中字符串"what's this"表示方式

方式 1：'what \'s this'. 这里用到了转义符'\'

方式 2："what's this"

c. 行末的单独一个反斜杠表示字符串在下一行继续，而不是开始一个新的行：

```
"This is the first sentence.\
This is the second sentence."
```

等价于 "This is the first sentence. This is the second sentence."

d. 用 r 或 R 指定自然字符串

r"Happy\n", 一般'\n'代表回车，因为用'r'指定为自然字符串，因此'\n'不会被解释器理解为回车换行符，而是'\'和'n'这两个字符，因此打印时会打印出"Happy\n"这 7 个字符。在正则表达式中常用到。

e. 用 u 或 U 指定使用 unicode 编码. 如：u"Happy"

f. 连写的两个字符串会被解释器理解为一个连续的字符串

"what's"" your name"会被理解为"what's your name"

g. 字符串 strip(), find(), replace(), join(),split() 方法的掌握，后面有讲到。

5. 说明几个运算符：

"not、and、or"：（布尔）非、（布尔）与、（布尔）或

"**"表示"幂"。如：3**4 = 81 (3 的 4 次幂) 也可以用 pow(3,4)

"//"表示"取整除"。如：4//3.0 = 1.0

/ 取整除或取商（浮点数），7/3.0=2.333... 4/3=1 8/3=2 9/float(4)=2.25

备注：在 Python3.x 中 / 是真正除，Python2.x 可以在代码开头加一句 from __future__ import division 这样去改变除法的行为，使得其和 Python3.x 一致。

% 取余数 8%2=0 4%3=1 5%2=1 7%4=3

思考：如何判断整除？即一个整数能不能被另一个整数整除，如何判断？(余数是 0 就是整除)

答: if x%m == 0:print "yes" 或 if not a%b:print "yes" 或者 if a==(a/b *b):print "yes"

6. 数据结构--简介

python 中每种数据类型都被当作对象。虽然 Python 中的每个值都有一种数据类型，但我们不需要声明变量的数据类型，因为 python 会根据每个变量的初始赋值情况分析其类型，并在内部对其进行跟踪，当然这种"i = long(12)"带类型来定义变量的方式也是支持的。Python 有多种内置数据类型，各类型关键字可用于类型强制转化、定义变量等, 如用 a=list()定义一个数组，用 help(set)学习集合。以下是比较重要的一些：

a. **布尔型**: 关键字 bool，值为 True 或 False，逻辑上的真、假。用于循环或判断
b. **整数**: 关键字 int，如 3, -1，也叫整型。
c. **浮点数**: 关键字 float，如 3.2，7.92, 也叫浮点型。
d. **字符串型**: 关键字 str，如"hello python"；关键字 unicode, 如 u"hello"。
e. **列表**: 关键字 list，是值的有序序列，如[3, 2, 5]，值可以是任何类型。
f. **元组**: 关键字 tuple，类似于列表，但值不可变的序列，如 (1,3,5)，(1,)。
g. **集合**，关键字 set，是装满无序值的包。**三大特性：**无序性，确定性，互异性。
h. **字典**，关键字 dict，是键值对的无序包。

(了解)【要使用 C 语言类型的数据，需要导入 ctypes 模块，以便使用 c_int、c_long 等类型。可以在 python 的 shell 先"import ctypes"然后"help(ctypes)"查看具体规定】

用关键字可以创建一个列表,元组,集合和字典,如 my_dict = dict() 这样就创建了一个名称为 my_dict 的字典。
list1 = list() 或 list2 = [] 创建一个空的列表
tuple1= tuple() 或 tuple2 = () 创建一个空的元组
a = (1) 或 a = ('a') # a 是 tuple 类型吗？不是，后文有解释
set_a = set() 创建一个集合，新版也可以用 set_a = {1,2,3} 来创建一个集合

字符串为什么有两种？
由于历史原因，Python 开始是不支持 unicode 的(Python 诞生时还没有 unicode)，后来加入了 unicode 的支持。也就是说如果用到中文我们就会用到 unicode

数据结构--"列表（list）"

list 是一组有顺序的数据，它的元素（对象）个数是变的（后面要说的元组则是固定的）。

a. 各种内建方法

```
>>>list_a = [1, 2]              #  定义 list_a 含有 1,2 两个对象
>>>list_a.append(2)            #  添加一个对象到末尾
>>>list_a                      #  查看 list_a 中内容
[1, 2, 2]                      #  shell 中显示
>>>list_a.count(2)             #  返回 list_a 中对象是 2 的个数
2                              #  shell 中显示有 2 个值为 2 的对象
>>>list_b = [3, 4]             #  定义 list_b 含有 3，4 两个对象
>>>list_a.extend(list_b)       #  在 list_a 末尾添加添加 list_b
>>>list_a                      #  查看 list_a 中内容
[1, 2, 2, 3, 4]                #  shell 中显示
>>>list_a.index(3)             #  返回第 1 个匹配的指定值在 list 中的位置，list 中第 1 个
                                 对象的位置是 0，第 2 个对象位置是 1，以此类推
3                              #  shell 中显示
>>>list_a.insert(2，'ok')      #  在 list_2 位置是 2 的对象前添加对象'ok',返回 None
>>>list_a                     #  查看 list_a 中内容
[1, 2, 'ok', 2, 3, 4]
>>>list_a.pop()               #  删除并返回 list 中最后一个对象
>>>list_a.remove(2)           #  删除第 1 次出现的匹配指定值的对象,没有指定值的话 ValueError
                                 如何删除所有的 2 呢？while 2 in list_a: list_a.remove(2)
>>>list_a                     #  查看 list_a 中内容   可以想想如何删除所有的 2
[1, 'ok', 2, 3, 4]
>>>list_a.reverse()           #  将 list_a 中的所有对象的位置反转
>>>list_a                     #  查看 list_a 中内容
[4, 3, 2, 'ok', 1]
>>> List=[1,5,7,1111,2,1.5]
>>> sorted(List)              # 有返回值，List 本身不变
[1, 1.5, 2, 5, 7, 1111]
>>> List
[1, 5, 7, 1111, 2, 1.5]
>>> List.sort()               # 就地操作，无返回值，结果保存到 List 本身
>>> List
[1, 1.5, 2, 5, 7, 1111]
```

list_a.sort(cmp=None, key=None, reverse=False)：对 list_a 排序

key：指定"取值函数"。取值函数带 1 个参数。sort 方法从 list_a 中取出元素，作为实参传递给取值函数，取值函数做些处理，再将处理后的返回值传递给比较函数。

cmp：指定"比较函数"。比较函数带 2 个参数。比较函数接收取值函数传送过来的 2 个参数，经过某种比较，返回比较结果（-1, 0, 1）。在写具体比较函数时，往往会用到 python 内建函数"cmp()"

reverse: 指定为 True 时，将排序后序列顺序转置，否则无动作。

【list_a.sort()不指定任何参数时，对 list_a 按小从到大排序,返回值为 None, sorted(list_a)排序时 list_a 不变，而是返回一个 list】

b. **列表使用：**下标方式，结合切片操作符 ':'

```
>>> list_a[2] = 'ok'          # 给列表中某一项赋值
>>> list_b = list_a[2:]       # 将 list_a[2]起到最后一个元素，把值赋给 list_b
>>> list_b = list_a[:3]       # 相当于 list_a[0:3]赋给 list_b，不包括 list_a[3]
>>> list_b = list_a[1:4]      # 将 list_a[1]至 list_a[3]赋给 list_b，不包括 list_a[4]
>>> list_b = list_a[1:-1]     # 将 list_a[1]到 list_a 中最后一项(不包括最后一项)赋给
                              list_b。-1 表示最后一项，-2 表示倒数第 2 项，以此类推
```

c. **遍历 list 中数据**

```
for value in list_a:
        print value    # 在循环中对 value 进行操作
```

【临时变量 value 遍历 list_a，循环取值，直到遍历完整个列表】

d. **用 列表解析(list comprehension) 定义列表**

```
>>>my_list = [2*i for i in range(0, 5) if i>2]     # 形式为 [表达式，变量范围，条件]
```

(1) 表达式：2*i，i 是变量

(2) 变量范围：for i in range(0,5)

(3) 条件：if i>2

再举一例：

```
>>>list_a = [2, 3, 4, 5, 6]
>>>list_b = [3*i for i in list_a if i%2==0]
>>>print list_b     # 终端显示[6, 12, 18]
```

e. **列表相关其它函数**（初学者了解，但都非常有用！）

(1) any(List) List 中任何一个元素都有逻辑真，则返回 True，全为假返回 False

all(List) List 中所有元素都为真时返回 True,否则返回 False

(2) max(List) min(List) 返回列表中的最大值 或 最小值

(3) enumerate() 在列表中**同时循环 索引和元素**，如：

```
List=["MaoZedong", "DengXiaoping", "JiangZemin", "HuJintao", "XiJinping"]
for index, entry in enumerate(List):
    print index, entry
```

(4) 求和函数：sum，如 sum([1,2,3,4]) 可求 1+2+3+4

sum(range(101)) 可求 1+2+3+4+...+100 的和

(5) 过滤函数：filter(func,list) 对 list 进行过滤，保留满足条件的，返回一个新的 list, function 的返回值只能是 True 或 False

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0    #函数中只有一句可写成一行，但不建议，读起来费劲
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
>>> def f(x): return x != 'a'
>>> filter(f, "abcdef")
'bcdef'
```

例子：找出 1 到 10 之间的奇数

```
filter(lambda x:x%2!=0, range(1,11))
```

(6) map(func,list) 把 list 中每一个元素操作，返回一个新的 list

```
>>> def cube(x): return x* *3   # x**3   equals to   x*x*x
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
>>> def cube(x) : return x + x
...
>>> map(cube , "abcde")
['aa', 'bb', 'cc', 'dd', 'ee']
```

另外 map 也支持多个 sequence，这就要求 function 也支持相应数量的参数输入：

```
>>> def add(x, y): return x+y
>>> map(add, range(8), range(8))
[0, 2, 4, 6, 8, 10, 12, 14]
```

(7) reduce(func,list [,init]) 如果给 init 的话用 init 和 list[0]用 func 函数处理，得到的结果再与 list[1]经过 func 处理，依此类推，直到结束，如果不给 init，第一次直接从 list 中取出两个(即 list[0]和 list[1])

```
>>> def add(x,y): return x + y
>>> reduce(add, range(1, 11))
55 （注：1+2+3+4+5+6+7+8+9+10）
>>> reduce(add, range(1, 11), 20)
75 （注：20+1+2+3+4+5+6+7+8+9+10）
```

利用 reduce 求阶乘方法

```
def f(n): return reduce(lambda x,y:x*y , xrange(1,n+1))
```

7. 数据结构--"元组（tuple）"

tuple 是一组有序**且不可改变的数据（数组是可以改变的）**，它的元素（对象）个数是固定的。我们有时候会强制转换 my_tuple = tuple(list_a)，由 list_a 得到一个元组 my_tuple；或者 list_a = list(my_tuple) 由一个元组得到一个列表。关于使用方法，也是下标方式，与列表一样。记住，它的值不能改变，因此不能赋值。

特殊地：（元组也不是那么不可变）

1. 元组中的列表可以被改变，如：t=([2],3,7); t[0][0]=100 运行后看看 t 变成什么了
2. t=("first","second"); t=t+("third","fourth") 看看 t 现在变成了什么

思考: 1 中 t[0]能被改变吗?如果想改变的话应该怎么做?

注意：一个元素的元组一定要小心，要在后面加上逗号

```
Tuple_a = ("Weizhong Tu",)  # 注意后面的逗号，少了就成了 string 类型了
Tuple_b = (999,) # 999 后面的逗号少了的话就成了整型数字
```

8. 数据结构--"字符串（str 和 unicode）"

字符串同列表，元组一样，都是**序列**。序列的特点是可能使用"**索引**"和"**切片**"，即我们可以取得序列中的某一项，或者某一部分。

1. 索引: a="student"　　a[2]是 a 中第三个字母'u' (从 0 算起，2 就是第三个)

2. 切片: "string"**[start:end**:[step=1]**]**中 start,end 必须有一个,和列表切片相同,step 不给时默认为 1

"**stude**nts"[1:5]是取第二个字母到第五个，不包括第六个(口诀：**要前不要后**)

常见操作：

```
>>> my_str ='Jason'     # 定义字符串 my_str
>>> my_str.startswith('Jas')   # 若 my_str 以'Jas'开头，返回 True，否则返回 False
>>> my_str.endswith('son')   # 若 my_str 以'son'结尾，返回 True，否则返回 False
```

备注：.startswith(), .endswith() 均可以 tuple 为参数，满足其中之一即返回 True

String.startswith(**('a','b')**) # 当 String 以 a 或 b 开头均返回 True

```
>>> my_str.count('a')          # my_str 中有几个 'a'
>>> if 'a' in my_str:          # 判断 my_str 中是否含有字符'a'，有返回 True，否则返回 False
>>> my_str.find('a')   # 返回'a'在 my_str 中的位置，找不到则返回-1 (不是 0,因为 0 是第一个，是找到了的意思)  -1 在逻辑上是真，0 在逻辑上是假，所以查找 字符串是否包含 更推荐用 in 方法。
```

>>> my_str.**index**('a') 与 my_str.**find**('a') 功能相同，但找不到时发生 ValueError

```
>>> ss = '***'     # 定义字符串 ss
>>> my_list = ['a', 'b', 'c']   # 定义列表 my_list
>>> mylist_to_string = ss.join(my_list)   # 使用 ss 来连接 my_list 中各元素，返回组成的字符串给 list_a。有返回值，ss 与 my_list 均未改变。
    mylist_to_string 值为 'a***b***c'
>>> list_a = my_str.replace('a', 'b')   # my_str 中的'a'替换成'b'形成的字符串赋给 list_a。my_str 实际未变
>>> a = u'中国' # 定义 unicode 字符串，如果在文件中在文件开头加上 #coding=utf-8
```

总结：unicode('Learn python') 转化为 unicode 字符串 str(u'learn django') 转化为 str 字符串。一般不这样转化，在处理中文地用到 unicode，建议用 utf-8 编码的 unicode

在 Python2.7 中，处理含有中文等 unicode 的字符时，读入时要 data.decode('utf-8') 变成 unicode 字符，输出 f.write(data.encode('utf-8'))，即处理过程 **多用 decode，使得字符串尽量都是 unicode 这样可以简化操作。**也可以在文档开头引入 from __future__ import unicode_literals 使得不带 u'' 的字符串也是 unicode 类型。

python3.x 中统一为 unicode 了，不再区分，所有问题变得更简单，但是由于一些包并不兼容 Python 3.x ，所以 Python 2.x 也还被广泛使用。

关于字符串的处理还有一个强大的工具就是**正则表达式**，见后文。

9. 数据结构--"字典（dict）"

字典就是"键名/数值"对（简称'键值对'）无序的合集。键名是唯一的，不可变的，键值是可以改变的。{key1:value1, key2:value2}

```
>>>my_dict = {'Jason':2011, 'Anson':2012}    # 定义字典，包含 2 个键值对
>>>my_dict["Wuhan"]=120   # 增加一个新的,返回{'Jason':2011, 'Anson':2012,"Wuhan":120}
>>>my_dict["Wuhan"]=2013 # 改变值,返回{'Jason':2011, 'Anson':2012,"Wuhan":2013}
>>>my_dict["Wuhan"]   #得到 2013，也就是取值，找不到时 KeyError,除此和 del 之外相当于 my_dict.get('Wuhan')
>>>my_dict.items()                    # 返回一个列表:列表元素是元组（每个键值对组成一个二维元组）
```

【for item in my_dict.items() 可用来在 my_dict 中遍历，类似列表中介绍的遍历方法】

```
>>>'Jason' in my_dict          # 判断 my_dict 中是否有键名'Jason'，有返回 True，否则返回 False,相当于 my_dict.has_key('Jason')
>>>del my_dict['Jason']        # 删除键名为'Jason'的键值对
>>>dict_a = my_dict.copy()     # 将 my_dict 复制给新建字典 dict_a
>>>data = my_dict.get('Jason')  # 从键名为'Jason'的对中取出值，赋给 data，dict.get(key,default=None), default 默认为 None 所以，找不到时返回 None
2011                            # 屏幕显示内容
>>>my_dict.setdefault('k','d') # 如果'k'存在，返回对应的值，如果不存在，赋值为'd'
>>>my_dict.keys()   # 返回所有键名组成的列表
>>>my_dict.values()   # 返回所有键值组成的列表
>>>my_dict.pop('Jason')   # 删除键名为'Jason'的键值对，返回该对的值，key 不存在时报错，用 pop(key,None)
>>>my_dict.popitem()    # 随时删除一个键值对，返回二维元组(key, value)
>>>my_dict.clear()   # 清除所键值对
>>>my_dict.update(dict_a, **dict_b)   # 在 my_dict 中添加 dict_a, dict_b 中各键值。可以只用第一个参数，即一次只添加一个字典
>>>for key in my_dict:                # 直接在 my_dict.keys()遍历
        print key,my_dict[key]
>>>my_dict.setdefault(key,value)  当存在 key 时获取它，没有时赋值为 value，并返回值
```

字典排序：sorted(my_dict) 相当于 sorted(mydict.keys()), 没有 my_dict.sort()

【更快的迭代方法：(python 3.x 中取消了这些方法)

.iteritems() **.iter**keys() **.iter**values() 和 items() .keys() .values()功能相当，但只用于迭代】

有顺序的字典：

```
from collections import OrderDict
Dict1 = OrderDict() # Dict1 就是一个有顺序的字典
```

字典默认值： 一般字典中 key 不存在时使用 DICT[key] 时会引发 KeyError, 如果希望 key 不存在时返回一个默认值，除了了 DICT.get(key,default_value) 之外，还可以用

```
from collections import defaultdict
Dict1 = defaultdict(lambda: 'No') # 这样，当访问一个不存在的值时就返回 'No'
```

数据结构--"集合（set）"

我所理解的，**集合**的概念就是数学上所称的集合的概念。

集合具有**互异性，确定性，无序性**。

```
>>> set_a = {1, 2}              # 定义集合，或用 set_a=set()定义空集合
>>>set_a.add(4)                 # 向 set_a 中添加元素 4
>>>set_a.remove(2)      # 从集合 set_a 中删除 2，没有的话 KeyError
>>>set_a.discard(2)             # 删除集合 set_a 中的元素 2,remove 的友好版本
>>> set_a.pop()       # 从集合 set_a 中随机删除一个值，并返回该值。从空集合中 pop
                      会引发 KeyError，.pop() 无参数，不同于字典。
>>> set_a.clear()       # 清空集合
>>> set_b = set_a.copy()       # 将 set_a 内容复制给 set_b（浅拷贝）。
```

【"set_b = set_a"表示"引用"关系，并非是复制,也可用 set_b=set_a[:] 来复制】

```
>>> set_a.update([5,9,2])      # 将 5，9，2（可以是任意多个值）加入集合，根据数字中
                               集合的概念，相同的值不会重复加入。
>>> set_c = set_a | set_b      # 并集 set_c 为 set_a 和 set_b 的并集
>>> set_c = set_a & set_b      # 交集 set_c 为 set_a 和 set_b 的交集
>>> set_c = set_a - set_b      # 差集 set_c 为 set_a 去掉其与 set_b 公共的部分
>>> set_c = set_b ^ set_a      # 异或 set_c 为 set_b 并 set_a，再去掉二者公共部分
```

s |= t 并，s 与 t 并集，存到 s 中,相当于 s=s.union(t) 或 s.update(t)

s &= t 交，s 与 t 交集，存到 s 中,相当于 s=s.intersection(t) 或 s.intersection_update(t)

s -= t 差，s 与 t 差集，存到 s 中,相当于 s=s.difference(t) 或 s.difference_update(t)

s ^= t 异或，s 与 t 不同的部分并到一起 (s 并 t，再去掉 s 交 t) ，存到 s 中,相当于 s=s.symmetric_difference(t) 或 s.symmetric_difference_update(t)

```
>>> if 3 in set_a:       # 判断 set_a 中是否有元素 3
>>> set_a <= set_b    # 判断 set_a 是否为 set_b 的子集，返回 True 或 False,相当于 set_a.issubset(set_b)
>>> set_a >= set_b    # 判断 set_a 是否为 set_b 的超集，返回 True 或 False,相当于 set_a.issupset(set_b)
```

【if 语句中，空集合为 False，任何非空集合为真值】

思考：字典和集合都是用{},那么如何定义一个空的集合？

```
>>> a={}              #这样是定义一个空字典，也可用 a=dict()
>>> type(a)              #<type 'dict'>
>>> a=set()          #定义一个空集合
>>> type(a)              #<type 'set'>
```

10. 数据类型之间的转换

利用关键字可以很容易地实现：比如 int, str, list, tuple, set 等

1.利用集合的互异性对列表去重：list_a = list(set(list_a)) 把 list_a 变成集合，再变成列表

2.字典-->列表 dict_a.items() dict_a.keys() dict_a.values()，用 list(dict_a)相当于.keys()

3.字符串与数字转换： 变成数字 int("2013") 变成字符串 str(120)

4.数字类型之间转化：整形变浮点型 float(2) 得到 5 除以 2 的准确值可以用 5/float(2)

5.元组，列表，集合变字符串，用 “”.join(sequence) 字符串变 list 用 string.split()

备注：print 函数会对输出对象调用 str()，所以 print "I am", 23, "years old."这样写也不报错，但是用"I am" + 23 + "years old"就会报错，这时就得转换一下 str(23)

11. 选择语句、循环语句

选择语句(if-elif-else)：

# if-else 	
x=input("Please input a integer:")
if x>0:
　　print "x>=0"
else:
　　print "x<0"
	# if-elif-...-else
	score =raw_input("score: ")
	score = int(score)
	if (score>=90) and (score<=100):
	　　print "A"
	elif (score>=80 and score<90):
	　　print "B"
	elif (score>=60 and score<80):
	　　print "C"
	else: print "D"
		# if
		x=100
		if x==100:
		　　print "x=100"
		
		条件等于要用两个等号，因为一个等号是赋值！

循环语句(for, while)：

for i **in range**(101)：
　　print i
for 可以在一个一个可迭代的对象里面循环，比如 string, list, tuple, dict, set 等，最常用 的是 list 和 dict
	x=0
	s=0
	while x <= 100: #直到 x>100 时停止
	　　s = s + x　# 也可写成 s += x
	print s

# continue **for** i **in range**(11): 　　**if** i ==7: 　　　　**continue** 　　**print** i 执行后会发现没有 7，因为 i=7 时执行了 continue 语句，直接到下一个循环，后面的不执行	**# break** x=0;s=0 # 分号可以让两个语句写在一行 **while True**: 　　**if** x>100: 　　　　**break** 　　s = s + x 　　x = x + 1 **print** s 当 x=101 时会执行 break 语句，跳出循环，如果没有 break，将是一个死循环，一直不断执行，永不停止，直到死机！

【备注：另外还有 for-else 和 while-else 语句，可以自学】

12. 关于模块

从功能上讲，每个用 python 编码的合理的脚本应是一个功能模块，用".py"作扩展名。Python 也能够调用由 c/c++编译生成的动态链接库，这种能力大大提升了整个程序的运行速度【如内建模块，系统中并没有它们的源代码】。

a. import 语句

import 语句用来导入其它模块，主要有两种用法：

用法 1："import A"表示导入整个模块 A

用法 2："from A import B"表示导入模块 A 下的对象 B（B 可以是任何对象，python 中一切皆为对象，如类，变量等）。

【假设模块 A 中有子模块 B，而 B 中有个函数 C，我们要调用这个函数 C。如果"import A"，就要"A.B.C()"，用"from A import B"，只要"B.C()"，就是说可以从导入的具体对象开始引用，用"from A.B import C"，只要"C()"】

默认导入路径在 sys.path 变量中定义。可以使用 sys.path.append("目录 ")来添加导入路径。例如：

```
import sys
sys.path.append("/usr/test")     #   将/usr/test 目录添加到 sys.path 变量中
import MyModule                  #   导入在/usr/test 中的 MyModule 模块
```

b. 直接使用 python 编码的模块的情况。例如我用们 os 模块创建文件夹：

```
import os                        # 导入 os 模块
os.mkdir("/tmp/folder")          # 调用 os 模块的 mkdir()函数
```

【使用 import 导入模块时，模块的主代码块（即函数与类之外的部分）会被立即执行，可以使用下述方法来控制是否在被导入时执行(独立运行时模块的__name__为"__main__"，当被导入时不是"__main__",是文件名)：

```
if __name__ == "__main__":
        #   主代码块放在这里
```

】

c. (了解)通过内建的 ctypes 模块，加载由 c/c++编译得到的动态链接库的情况。具体请看下文中，"常用模块功能"中的"ctypes 模块"。

d. (了解)使用 cython 工具，将类似 python 风格的代码编译成 c 代码，然后再用上例中介绍的方法编译成动态链接库，以供加载调用。例如：

```
cython some.pyx              # 调用 cython  命令将 cython 文件 some.pyx 编译成 some.c
gcc  some.c  –fPIC  –shared  –I  /usr/include  –o  some.so
```

【cython 算是另一种类 python 语言，混合了 python 与 c 的编程风格，有自己独立的"编译器+语法规则"。用途是使用较简洁的类 python 代码，编译生成 c 语言源码，最终目的是生成动态链接库供 python 代码加载调用（当然由 c 源码编译为动态链接库的过程不是 cython 编译器的工作），以提高 python 程序的运行速度。使用 cython 还可以快速应用 python2 代码到 python3 中。cython 官网下载 http://www.cython.org/】

13. 定义一个函数

a. def my_func(arg1, arg2):

```
    # 函数执行代码块
```

【使用"def"关键字表达开始定义一个函数，函数名后接参数（不需指定类型），参数个数不限。然后要加一个冒号。函数执行代码块是在同一长度缩进之下。】

b. def add(A=1,B=2): # 缺省变量（默认参数）

```
    print "A:",A,"B:",B
```

调用时执行 add()时 A=1,B=2，执行 add(A=3) B 默认为 2，执行 add(2,3)时 A=2,B=3，执行 add(B=5) A 为默认值 1,用起来比较灵活

【**NOTE**：默认参数放最后较好！】def add(x=3,y):print x+y 这样 y 不指定会出错！改为 add(y,x=3)

c. def my_func(arg, *args):

```
    print arg, args
```

【*前缀表示从第 2 个参数起所有参数都会作为一个元组存在于 args 中，调用上面这个函数就可以看到实现情况了。】例子见 d

【*args 后还可以有参数吗？ 用 fun(arg, *args, other_arg=default)】

d. def my_func(args):**

【**前缀表示从第 1 个参数起所有参数都会作为字典键值对存在于 args 中】

例如：myfunc(a=1,b=2) args 就是一个字典被传进去，即{"a":1,"b":2}

```
# Example for *args
def a(*s):
    print sum(s) # s 是一个 tuple
a(1,2,3,4)
#调用就会得到 1+2+3+4 的和，参数个数不受限制
```

```
# Example for **args
def where(**s):
    for i in s: # s is a dictionary, you can print s
        print i,"comes from",s[i]
where(tuweizhong="Xinyang",John="America") # 调用
```

e. **lambda 语句：**用于创建新的函数对象，并在运行中返回它们，例如：

```
>>>twice = lambda s:s*2【参数：s，表达式：s*2，函数对象：twice】
>>>print twice(7)
14
```

可用 lambda 语句来定义函数，使它能够创建*符合某种规则*的函数，例如：

```
>>>def create_multiple_func(n):              # 定义一个用来创建函数对象的函数
...     return lambda s:s*n
>>>multiple_2 = create_multiple_func(2)    # 创建"乘 2"函数
>>>multiple_3 = create_multiple_func(3)    # 创建"乘 3"函数
>>>print multiple_2(7)                      # 调用"乘 2"函数
14
>>>print multiple_3(7)                      # 调用"乘 3"函数
21
```

【lambda 语句只能用单个表达式来创建函数对象】

f. **递归函数：**(函数可以调用自己本身)

```
def factorial(n): # 一个求阶乘的函数
    if (n==0 or n==1): return   1
    else: return n*factorial(n-1)
print factorial(5) # 得到 5! =5*4*3*2*1 的值
```

g. 函数返回值，关于 return

备注：当函数中没有 return 语句时就是 return None 的意思

def func(c): return c 调用时用 A=func(c) 返回值就会被保存在 A 中，可以返回任何值，如 string,list，甚至返回一个函数	def func(a,b,c): return a,b,c 函数返回的是一个 tuple，即等价于 return (a,b,c)	def func(a,b,c): return [a,b,c] 返回的是一个 list 【NOTE：有多个 return 时，运行一个其中一个函数就停止！后面所有语句不会再执行】

def twice(s):
return s*2
 	
A = twice(4)
 	
#返回值保存在 A 中，定义函数的好处是可以在另一个地方，比如另一个函数中调用，把代码分成一个个功能块
	# find all odd
	a=[1,2,3,4,5,6]
	def findOdd(List):
	result=[]
	for x in List:
	if x%2:
	result.append(x)
	return result
	r = findOdd(a) # 结果存到 r 中，r 是一个列表
		Python 中的关键字
		None True False
		
		0 和 None 在逻辑上为 False
		-1 在逻辑上为 True
		一个有趣的例子：
		a=5; b=10
		c=[b,a][a>b]
		a>b 是 True, [b,a][True] 即[b,a][1]
		返回 a,b 中较大的一个

h. 局部变量和全局变量

a=3
def plus():
b=4
print a+b
 	
a 在函数外，是全局变量，b 在函数内，是局部变量，也就是说 b 只能在函数在使用，在函数外 print b 试试看
	a=3
	b=4
	def plus():
	a=6
	b=7
	print "a:",a,"b:",b
	
	print "a: ",a,"b:",b
	为什么函数中 a,b 赋值不会把原来的覆盖掉呢？
		def plus()
		global a # a 为全局变量
		a=5
		plus() # run this function
		print a
		# global 声明一个变量为全局变量，这样在函数外就可以使用这个变量，不用 global 声明的话会报错，因为找不到 a

函数运行时，先找局部变量，再找全局变量，都找不到就报错！函数运行后，函数内的非全局变量变会被释放！如果函数内变量要用在函数外，用 global 声明！

14. 关于 python 的类

a. 定义类用关键字"class"，域和方法列在一个缩进块中。类中的方法都要有一个表示自己的参数，通常定义为"self",如：

```
class Jason(继承的父类):
    def __init__(self, name, age=29):     # age=29 是指定默认参数值
        # 函数代码。实例化一个对象时，自动调用该函数，相当于构造函数
    【在不使用任何域的时候，可以不定义__init__，但通常都会定义】
    def show(self, some): #参数中必须有 self，代表类或实例本身
        # 函数代码
obj = Jason('chiweixi', 28);     # 参数传递给 Jason 类中__init__函数中除 self 之外的 2 个参数，它们是用于实例化时接收的
```

b. 类与对象的域

```
class People:
    number = 0     # 定义类的域（类的属性），所有类的实例会共用！
    def __init__(self, name):
        self.name = name    # 定义实例属性，可以在类中的任何函数中定义
Jason = People('chiweixi') # 实例化 （number 和 name 都是实例属性，属性也叫成员变量）
```

类的域（类属性）可以这样引用：People.number，即通过类名就可以使用，即使没有**实例化**任何对象；实例属性必须在将类实例化为对象之后，通过实例来引用，如 Jason.name。

c. 类以"__"为前缀的，python 认为它是私有的，外部调用会引发 AttributeError，其它情况均为公共。有个惯例，"_"为前缀标志只希望在类或对象中使用，但语法上仍是公共。

d. 关于继承（自己学）

e. 模仿一个文件读写的例子（by tuweizhong）

一般我们用 open(path).read()来读取一个文件，模拟一下

```
class open: #  也可以用 class open(object):
    def __init__(self,filePath):
        self.filePath=filePath
        print "I am working"
    def read(self):#  读取文本
        #  读内容的代码写这里
        print "Finished!"
    def write(self): #写文本
        print "write something!"
```

在调用的时候就是 open(path).read() # 发现了没，open 的括号里面的参数写在__init__里面了，当调用函数时，__init__会先执行，初始化一些操作。更多知识请搜索

15. 命令行参数

```
import sys
print sys.argv       # a list
```

首先进入脚本所在目录，然后运行 python 脚本名称 参数

例如把上面存为 try.py 进入 try.py 脚本所在目录,运行 python try.py 123 会得到

["try.py","123"] 这样我们就可以捕捉命令行传入的参数，进行处理
运行 python try.py have a try 会得到
["try.py","have","a","try"]

16. 引用和复制一个对象

python 中，"a = b"表示的是对象 a 引用对象 b，对象 a 本身没有单独分配内存空间(**重要**：不是复制！)，它指向计算机中存储对象 b 的内存。因此，要想将一个对象复制为另一个对象，不能简单地用等号操作，要使用其它的方法。如序列类的对象是（列表、元组）要使用切片操作符（即':'）来做复制："a = b[:]"。[建议学习相关模块: **copy**]
问题：字符串不是引用: a="tuweizhong";b=a 试试改变 b 的值看看 a 变不变，数组呢？(字符串不是可变对象在改变时会重新申请内存, id(b)会发生变化)

```
>>> a = [1,2,3]
>>> b = a
>>> b[0] = 5
>>> b
[5, 2, 3]
>>> a
[5, 2, 3]
```

```
>>> c = a[:]
>>> c[0] = 999
>>> c
[999, 2, 3]
>>> a
[5, 2, 3]
```

```
>>> d = [a,c]
>>> d
[[5, 2, 3], [999, 2, 3]]
>>> e = d[:]
>>> e[0][0] = 444
>>> e
[[444, 2, 3], [999, 2, 3]]
>>> a
[444, 2, 3]
```

b = a 其实 b 和 a 是指向同一个内存地址，可以用 id(a)和 id(b)来看是否相同。
c = a[:]是对 a 进行了浅拷贝(与 c = copy.copy(a) 相当)，所以改变 c 时发现 a 没有受到影响，我们用同样的方式对 d 进行了浅拷贝，发现 e 改变的时候影响到了 a，也就是说浅拷贝不会拷贝引用中的引用，如果想完全拷贝一份，应该用深拷贝：

```
import copy
e = copy.deepcopy(d) # 再尝试去改变 e 看看 a 和 c 会不会受到影响（答案是不会）
```

17. 常用内建函数

print 函数：

```
print 'name: {0}, age: {1}' .format (name, age) # 用 format 进行格式化
print('name:%s, age:%s' %(name,age)) # 用 %s 进行格式化
```

raw_input()：# 还有 input()，它们有什么区别呢?试试看

```
A=raw_input("Please input string A")，获得的输入的字符串，并存在 A 中
```

help()：

用于获取帮助或者自学！在 python 代码中，或者在 python 的 shell 中，如果已经定义了某个函数"some_func()",那么只要输入"help(some_func)"就会进入 help 界面下显示 some_func 的文档字符串。此时，按"q"键退出 help 界面。

range(a, b): # range([start=0,]**end**[,step=1]) # 中括号的意思是这个参数可以省,当有两个参数时意思是 range(start,end)，step 默认为 1，自己指定 step 必须是三个参数！

```
>>>list_a = range(2, 5)      # 初始化 list_a, 相当于 list_a=[2, 3, 4]
>>>list_a                   #  查看 list_a 内容
[2, 3, 4]
```

range(5)和 range(0,5)和 range(0,5,1)一样得到[0,1,2,3,4]没有 5！(要前要不后，和切片操作一样，5 是取不到的), range(1,15,4)就是[1,5,9,13], range(2,4)就是[2,3]，记住没有 4

dir(模块名)：

列出模块定义的标识符，不指定模块名时列出当前模块的标识符。

len()：返回对象中元素的个数。可被用于字符串、列表、元组，字典、集合等。

cmp (a, b)：比较 a, b 的值

当 a>b，返回 1；a=b，返回 0，a<b，返回-1。

exec():

执行字符串行式的 python 命令，单词：execute:执行 (**一段代码**)

```
>>>command = "print 'ok'"
>>>exec command
ok
```

eval():

提取字符串中内容，单词：evaluate:求值 例如：

```
>>>print eval("3*2")  # 试试 eval("range(11)"),看看能得到什么
6
```

repr():

和 str()类似，将对象转换为字符串形式返回，例如：

```
>>>v = ['a', 'b']      # 变量 v 初始化为列表['a', 'b']
>>>s = repr(v)        # 将 v 转化为字符串"['a', 'b']"
>>>print s            # 打印字符串
['a', 'b']
```

str 对人友好，repr 对 python 友好，也就是说，str 是给人看的，repr 是给编译器看的

体会一下： eval(repr(object)) == object

zip(list_a, list_b):

用两个 list 生成一个对象是元组的 list.举例说明，list_a=[1, 3, 5, 7]， list_b=[2, 4]，list_c=zip(a, b)。List_c 实际为[(1, 2), (3, 4)]。zip()返回的 list 中对象的个数是两个参数 list 中对象个数少的那个，本例中 2 只有两个元素。

pow():

pow(a,b):返回 a 的 b 次幂，同 a**b

pow(a,b,c)：返回(a 的 b 次幂)除以 c 的余数，即 (a**b) % c

```
>>> pow(2,3)
8
>>> pow(2,3,3)
2
>>> pow(2,3,7)
1
```

file 操作：

```
my_file = open('file.txt', 'w')   # 在当前目录中创建文件名为"file.txt"的文件
my_file.write('something')      # 向文件中写入数据
my_file.close()                 # 关闭文件
my_file = open('filename', 'r')    # 只读方式打开
my_file.readline()              # 从文件中读出一行
my_file.close()                 # 关闭文件
```

一行一行地操作一个文件

```
f=open("D:/test.txt")   # 默认是"r",即读
for line in f:   # 这里的 f 不用写成 f.readlines()  f 本身就可以迭代，更省内存
    print line   # 一行一行地操作
f.close()   # 处理完后关闭文件
```

18. 常用模块功能：

【首先 import 导入所需模块，比如 import os】

a. os 模块

os.system("终端命令 1")：在 python 中执行"终端命令 1"，如同在终端中执行
os.mkdir("/mnt/share"):创建目录"/mnt/share"
os.makedirs(r"C:/a/b/c") 是 super-mkdir,创建所有子目录和可选权限, makedirs(name, mode=511)
os.sep :目录分隔符，使用它会提高代码可移植性。例："scripts"+os.sep+"hello.py"
os.name:正在使用的 os 平台字符串。window 平台显示'nt'，linux/unix 平台显示'posix'
os.getcwd():返回当前工作目录
os.getenv('PATH')：读取环境变量'PATH'的值
os.putenv(键，值)：设置环境变量。
os.listdir('/root')：列出指定目录下所有文件和目录名(不会递归显示子目录)
os.walk('/root') 一种遍历目录的方法,例如:(不建议用 os.path.walk, python 3.x 中删掉了)

```
import os
def VisitDir(path):
  # root 为当前遍历目录,dirs 为当前目录下的目录列表,files 为当前目录下的文件列表
  for root,dirs,files in os.walk(path):
      for filename in files:
          print os.path.join(root,filename)
```

os.remove('/root/hello.py')：删除指定文件
os.rename("a.txt","b.txt") 重命名
os.linesep：当前平台使用的行终止符。windows 平台下'\r\n'，linux 下'\n'，Mac 下'\r'
os.path.split('/root/test/hello.py'):
将完整路径分开为目录名和文件名，返回一个 2 维元组，(目录字符串，文件名字符串)，本例中返回('/root/test', 'hello.py')
os.path.isfile(路径)：判断所给路径是否为文件，返回 True or False
os.path.isdir(路径)：判断所给路径是否为目录，返回 True or False
os.path.exists("/mnt/share"):判断目录是否存在，返回 True 或 False
os.path.getctime(path) 获取创建时间 os.path.getsize(path) 获取文件大小,单位 b
os.path.abspath(__file__): 在文件中运行，获取当前文件的绝对路径
os.path.dirname(path): 取得文件或目录路径的上一级

b. sys 模块

sys.exit(status):
相当于 raise SystemExit(status)，用于退出正在运行的程序，即退出 interpreter
【可用 help(sys.exit)、help(SystemExit)查看详情】
sys.version:python 版本信息，如"2.7.6……"
sys.stdin: 标准输入流

```
>>>my_str = sys.stdin.readline()
>>>print my_str
```

sys.stdout: 标准输出流

```
>>>sys.stdout.write('ok')    # 终端会显示"ok"
```

sys.stderr: 标准错误流

>>>sys.stderr.write('ok') # 因为默认标准错误输出是终端，因此效果同上

sys.path.append('/usr/test'):

代码执行时动态添加 import 搜索目录。仅当程序执行至该句时，目录'/usr/test'才会被加入 sys.path 变量中，程序退出后 sys.path 中不保存'/usr/test'目录。

sys.path.insert(index, '/usr/test'):

向 sys.path 变量中插入目录。将'/usr/test'目录插入到第 index 个目录之前，比如 index 为 0，sys.path.insert(0, '/usr/test')是将'/usr/test'插入到 sys.path 变量中作为第一个目录，这样就可替换某个软件的系统自带版本。如将 python3.3 替换 python2.7.6 等。

c. time 模块

time.sleep(2):系统延迟 2 秒，即暂停 2 秒

time.time(): 当前时间的秒数

time.ctime(): 当前时间字符串

time.strftime('%Y-%m-%d %H:%M:%S') 获取当前年月日 时分秒

d. cPickle 模块

用于存储对象至文件，及从中恢复。

```
>>>import cPickle
>>>cPickle.dump(object, file)          # 将对象 object 存储到文件 file 中
>>>my_obj = cPicke.load(file)          # 从文件 file 中取出对象给 my_obj
```

【可以"import cPickle as p"，这样代码中用 p 来代替 cPickle，好处是只要将 cPicke 改成 pickle，就切换了两个模块的使用】

【pickle 是 python 脚本，cPickle 是 c 代码生成的动态链接库，速度是 pickle 模块的 1000 倍】

e. ctypes 模块

ctypes 模块用于在 python 中创建和操作 c 语言数据类型。我们通常会使用该模块加载动态链接库的功能，使用范例如下：

步骤 1：编译动态链接库(some.c-->some.so)

```
gcc  some.c  –fPIC  –shared  –I  /usr/include  –o  some.so
```

【参数说明】

-fPIC: 编译为位置独立的代码，不用此项则编译的代码是位置相关的

-shared: 指定生成动态链接库

-I：some.c 中" #include"要链接非默认目录的话，指定该目录。本例为：/usr/include

步骤 2：使用编译好的动态链接库 some.so

```
from ctypes import cdll          # linux 系统下,导入 cdll 子模块，window 下用导入 windll
my_libc = cdll.LoadLibrary("./some.so")  # 加载当前文件夹下的"some.so"动态链接库
my_libc.hello()                  # 调用 some.so 的 hello()函数
```

【在 windows 系统中，将"cdll"替换成"windll"就可以了。当然加载的文件扩展名应是".dll"】

f. 复制和移动文件

```
import shutil
shutil.copyfile("a.txt","b.txt") # 将 a.txt 复制一份，名称为 b.txt
shutil.move(path,newpath) # 移动文件夹 或 文件
```

g. **其它重要模块**：datetime, json, sqlite3,等，自学

19. 正则表达式(初学者了解)

	记号	说明	示例	匹配哪些?
	literal	匹配该字符	this	this
	re1|re2	|为管道符号,多选一	Wuhan|Jilin	Wuhan 或 Jilin
	.	除换行符外任何字符	a.b	aab, a9b, a b, a?b …
边界	^	匹配字符串的开始	^Dear	Dear 开头的字符串
	$	匹配字符串的结束	txt$	txt 结尾的字符串
次数	*	前面的字符出现 0 次或多次	5*	0 个或多个 5
	+	前面的字符串出现 1 次或多次	a+	1 个或多个 a
	?	前面的字符出现 0 或 1 次	N?	0 个或 1 个 N
	{N}	前面的字符出现 N 次	X{5}	5 个 X
	{M,N}	前面的字符出现 M 到 N 次	W{2,5}	2 个或 3 个或 4 个或 5 个 W
	[…]	括号中任何一个字符	[aeiou]	a, e, i, o, u 任何一个
	[x-y]	匹配 x 到 y 之间的	[0-9]匹配数字	[A-Za-z]匹配一个字母
	[^…]	不匹配出现的任一个	[^aeiou]	不是 a, e, i, o, u 任何一个
(次数)?	(*|+|?|{})?	非贪婪，即找出长度最小且符合要求的(自己的理解)	[0-9]([a-z]{5}?)	1 个数字和 5 个字母或者只有 1 个数字
	(…)	匹配括号中的正则表达式，并保存为**子组**	([0-9]{3})	三个数字
	\d	任意数字，同[0-9]，（和\D 是反义，即任何非数字）	\d{11}	11 位数字，如手机号
	\w	匹配任何数字字母和[A-Za-z0-9]同义，和\W 反义	\w+	数字和字母构成的字符串均可匹配
	\s	匹配空白符,和[\n\t\r\v\f]相同，和\S 是反义		
	\b	匹配单词边界	\bThe\b	匹配 The，不匹配 There,或 together
	\	取消通配符特殊含义	\.	匹配字符.(点)本身
	\nn	匹配已保存的子组，参考上面的(…)	price: \16	
	\A(\Z)	匹配字符串的起始(结束)	\ADear	Dear 开头的

re 模块核心函数：

函数/方法	描述
模块的函数	
compile(*pattern*)	返回一个 regex 对象
re 模块的函数和 regex 对象的方法 （若用 regex 对象，以下 pattern 参数不需要,如右）	**re.compile(pattern)**.match(string) **re.compile(pattern)**.findall(string)
re.match(*pattern*,string)	匹配(从字符串开头开始)
re.search(*pattern*,string)	搜索(在字符串中找有没有符合的)
re.findall(*pattern*,string)	找出所有，返回 list
re.split(*pattern*,string,max=0)	分割所有符合 pattern 的地方，返回 list
re.sub(*pattern*,repl,string,max=0)	替换，把 string 中符合 pattern 部分替换成 repl 字符串(另有 subn()会多返回一个替换次数信息)
匹配函数对象的方法	
group(num=0)	返回全部匹配对象或指定编号是 num 的**子组，经过测试**,没有子组时 m.group()若匹配成功返回第一个,注意 group()等价于 group(0),看第一个子组用 m.group(1)
groups()	返回一个包含全部匹配的子组的元组，若匹配不成功返回空元组

实战:

1. 提取<string>和</string>之间的部分

```
import re
s = "<string>Weizhong Tu</string><string>Python is interesting</string>"
pattern = "<string>(.+)</string>" # 这里为什么要加括号，不加会怎样，试试看，理解一下子组
reg = re.compile(pattern) # reg 为 regex 对象
result_list = reg.findall(s) #
print result_list
```

结果我们却得到了 result_list = ['Weizhong Tu</string><string>Python is interesting']
这并不是我们想要的结果,re 好像默认是从最大范围内搜索我们要匹配的字符串,这里就要用到非贪婪匹配,把 pattern 修改为 pattern = "<string>(.+**?**)</string>"再试试看,想想 pattern = "<string>(.+)**?**</string>" 行不行呢,为什么?

2. 提取字符串中手机号

```
import re
s="Weizhong Tu: tel 18207149053 qq 291583814 China Mobile: tel 13800138000 qq 123456789"
pattern = "\d{11}" # pattern = "[0-9]{11}", pattern = "1[358][0-9]{9}"  may be better
result_list = re.findall(pattern,s) # re.compile(pattern).findall(s) may be better, why? learn it yourself !
print result_list
```

3. 判断字符串中是否有小写字母

```
import re
s="SL4a"
if re.search("[a-z]",s): # match 和 search 匹配成功返回_sre.SRE_Match object，不成功时返回 None
    print "lowercase letter found!"
```

4 查看匹配的子组

```
import re
s = re.search('([a-z])([0-9])', 'a1b2c3ddddd4e5') # 正则表达式中含有两个子组
if s:
    print s.groups() # 用 group(1) 查看第一个，group(2)查看第二个，注意 group(0)等价于 group()
```

你会发现这里匹配到一个就停止了，并不是把所有的都找出来，想找出所有的用 findall

问题：re.search('a*' , 'caaaaat').group() 你觉得结果是？为什么？（结果是空字符串）

正则表达式进阶：

正则中用()括号括起来的部分组成一个子组，(?:正则)可以不保存此子组

\数字 模式可以匹配前面的子组，在替换和重复时非常有用，如：

```
re.sub(r'(\b[a-z]+) \1', r'\1', 'cat cat in the the hat hat') # 得到 'cat in the hat'
```

匹配 abcdef 或 xyzdef 的方法：r'(abc|xyz)def' 也可以 r'(?:abc|xyz)def'不保存子组

使用 named group 来使用子组

```
m = re.match(r'(?P<first_name>\w+) (?P<last_name>\w+)', 'Weizhong Tu')
print m.group('first_name')
print m.groupdict()
```

后向肯定：r'正则 2(?=正则 1)' 功能：找出符合正则 2，且右侧满足正则 1 的内容

后向否定：r'正则 2(?!正则 1)' 功能：找出符合正则 2，且右侧不满足正则 1 的内容

前向肯定：r'(?<=正则 1)正则 2' 功能：找出符合正则 2，且左侧满足正则 1 的内容

前向否定：r'(?<!正则 1)正则 2' 功能：找出符合正则 2，且左侧不满足正则 1 的内容

例如从字符串 s 中找出"{ label }"这样，在花括号{}中间，左右各一个空格 label：

```
re.findall(r"(?<=\{ )(.*?)(?= \})", s)
```

即左边是花括号和一个空格，右边是一个空格和花括号

20. Python 异常处理

python 默认的异常处理是终止程序运行，并在屏幕上显示一个消息，python 解释器会响应"ctrl+c"组合键，抛出 KeyboardInterrupt 的异常。我们可以用下面形式替代 python 默认的异常处理：

```
# 我暂且将这个形式叫异常分支
try:
        # 主代码块
except EOFError:    #    EOFError 异常处理分支
        #    EOFError 异常处理代码块
except XXX:         #    XXX 异常处理分支,比如 ValueError
        #    XXX 异常处理代码块
else:     # 主代码块执行无异常后，进入这里
        # else 代码块
finally:    # 程序一定要执行的代码，不管是否发生了异常
        # 一定执行的代码块
```

我们可以自己发起一个异常：

代码块中： raise Jason('chiweixi', 28) # 发起一个带参数的 Jason 类的异常

异常分支中： except Jason, obj: # 接收 Jason 类的异常，实例化 obj（obj 接收了参数

```
print "{0}, {1}".format (obj.name, obj.age)
```

使用 assert 语句：

判断条件是否为真，若为真，引发异常 AssertionError。例如：

```
>>>i = 100
>>>assert i<50
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

21. with 语句

在 try 块中获取资源和随后的在 finally 块释放释放资源是一种常见的模式。因此，这还有一个 with 语句，它使这一切以一个干净的方式运行:

```
with open("poem.txt") as f:
    data = f.read()
# handle data here, here f is closed file, you needn't close file yourself
```

不同的是，我们使用的是带有 with 语句的 open 函数 -- 把关闭文件的工作留给 with open 自动完成。在幕后所发生的是，使用 with 语句有一个协议。它通过 open 语句获取返回的对象(这里称之为"thefile")，在开始代码块之前执行 thefile.__enter__，代码块执行完毕执行 thefile.__exit__。这一主题的更多讨论已超本电子书范围，更多的解释请参考 PEP 343：http://www.python.org/dev/peps/pep-0343/

22. 文档字符串

了解即可，用于提供一些帮助或说明，函数（模块或类）的第一个逻辑行是其文档字符串，惯例是一个多行的字符串。首行以大写字母开始，句号结尾；次行是空行；第三行开始是详细描述。举例：

```
"'Prints the maximum of two numbers.

The two values must be integers.……"'
```

使用"函数名.__doc__"来得到某函数的文档字符串，例如这样用：

```
print some_func.__doc__
```

23. 绝对路径 与 相对路径

说明：绝对路径和相当路径的概念相当重要，不仅仅在 python 中，任何一种编程语言，还有 Linux 终端都很常用，非常重要。

概括：以 '/' 开头的都是绝对路径，开头没有'/'的，都是相对路径，'./'(一个点)是当前目录，'../'(两个点)上一级目录，例如:

1. f = open('twz.txt') 是打开**脚本同一个目录中** twz.txt 文件；如果是在终端，就是当前的工作目录中的 twz.txt，工作目录可用 os 模块中 os.getcwd()获取，用 os.chdir(path)对其进行更改。
2. f = open('/home/tu/test.txt') 绝对路径
3. f = open('tu/test.txt') 相对路径，打开当前目录中的 tu 这个目录中的 test.txt 文件。
4. f = open('../test.txt') 相对路径，当前目录上一级目录中的 test.txt 文件

绝对路径和相对路径的转化：

```
>>> import os
>>> os.path.abspath('./')      # 显示当前目录的绝对路径
'D:\\Python27' (windows)     /home/tu (Linux)
```

文件所在路径的获取（不能在终端测试，写在文件中测试）

```
import os
os.path.abspath(__file__)      # 获取该文件的绝对路径
os.path.dirname(__file__)      #获取该文件所在目录的绝对路径
```

24. 其它应了解的

a. python 中区分大小写

b. Python 中任何数据结构都不需要使用定义语句（也没有定义语句）在使用前预先定义，也不需要声明数据类型，在使用变量时只需要给它们赋值即可
思考：列表没有定义可以直接用 list.append()向里面加值吗？ (答案：不可以！)

c. print 语句结尾可以使用"，"来消除每个 print 语句自动打印的换行符。常用于循环语句中。如 print i,

d. sys 模块提供的功能与 python 解释器及其环境有关

e. 没有返回值的 return 语句等价于"return None". "None"是 python 中的关键字，意思是"空的"，注意首字母大写。

f. ";"符号表示一个语句（或逻辑行）的结束
i=5;print i # 两个语句写在一行的方法,但不推荐这么做,影响阅读

g. 函数外变量如果会用于某函数内，在这个函数内要用"global"声明要使用的函数外变量,先声明，再赋值使用！不赋值只用可不声明

h. 空的 string,list,tuple,dict,set 在逻辑上均为 False
a="" #or a=() or a=[] or a={} or a=set()
if a: print "True"
else: print "False"
但是注意右图所示 a is False 返回的是 False！

```
>>> a=set()
>>> a is True
False
>>> a is False
False
```

```
>>> if not a:print "yes"
yes
>>>
```

i. 当元组中元素只有一个时
一个元组中有一个元素时要在后面加一个逗号，如 a = (1,) 或 a = ('a',)
为什么呢？想一想，（1+2）+3 是什么，是数字，同样，(1)也是数字 1，这样 a = (1) 就相当于 a = 1，你可以 type(a)看一下，字符串也是如此，('a'+'b') 返回的是字符串，当 a = ('string') 时，相当于 a = 'string'，要想是元组类型，得在后面加一个逗号。
提示：列表和元组每个元素后面都加一个逗号是一个好的习惯！
如：a = ['I', 'love', 'Python',] 或 b = ('Python', 'is', 'easy',) # 这是合法的
甚至 dict1 = {'name':'WeizhongTu', 'website':'www.ziqiangxuetang.com', } 也是合法的

j. python 中 的三元操作符
smaller = a if a<b else b

k. 第一次导入指定模块时，解释器执行了 3 个步骤：
(1) 找到模块文件
(2) 编译成位码（需要时），为了提高执行速度
(3) 执行模块代码来创建其所定义的对象

25. 继续学习什么

本教程将 python 入门的基本知识作了介绍。个人建议可以动手写一个 python 程序，用它来处理一些简单的任务，比如替换一个目录中文件的名称为 1,2,3…，在每个文件开头加上你的姓名（或者要求使用都输入一个值）等等，以下是推荐书目：

1.《byte of python》电子书 http://linux.chinaitlab.com/manual/Python_chinese/

2.《python 核心编程》[美] Wesley J. Chun

3. Python 100 道练手题： http://www.ziqiangxuetang.com/python/python-tutorial.html

其它知识点：1.python 装饰器，2.类的多重继承，3.类中的类方法和静态方法，4.数据库读写, 5. assert, raise, yield 6. unpack tuple, unpack dict，生成器，迭代器 7, 单元测试 unittest 8, 测试脚本速度模块:timeit

优秀 Python 工具或库推荐：bpython, pip, fabric, virtualenv

Python 优秀资源汇总：http://dwz.cn/learn_python

26. 如何自学

1. 利用 python 解释器，它是一个好东西，推荐用用 bpython 或 ipython
2. 引入想要学习的模块，然后用 help(模块名)进行学习
3. 利用 google 或百度搜索学习
4. 关键地方 print 一下变量，看看是否和预期相同，不对的话接着调试修改

关于编程的一些建议(要养成的好习惯)

1. **代码书写要便于阅读。例如：每行仅书写一条语句（尤其注意 if/for/while 语句）**
2. **编码一致性**
   a. 在同一个项目中的代码要保持一致
   b. 在同一个模块中的代码要保持一致
3. **关于缩进：**
   建议为 4 个空格。若使用 Tab 键，建议将其定义为 4 个空格。
   【缩进在 python 中是语法相关的，python 解释器通过缩进来判断代码块的从属关系，请谨慎使用 Tab, 特别是 Tab 和空格混用的时候千万要小心！】
4. **建议代码行最大长度限定为 80 个字符。**
   当前行未输入完毕，如果继续输入将超过限定长度，此时在当前行尾输入'\'后再回车换行（这样 python 解释器认为另起的新行与当前行是同一逻辑行）
5. **编码尽量用 ASCII 码，有特殊需要时可用 utf-8 码,中文可用 gbk**
   文件头加 #-*- coding:utf-8 -*- 或 #coding=utf-8 或 #coding:utf-8
6. **"import"语句使用规范：**
   a. 多条"import "语句要分行书写，不建议使用"import A, B"形式。
   b. 特殊地，可以使用"from X import A, B" 形式
   c. 导入各模块的书写顺序(自上而下):
      "import 标准内建模块"
      "import 第三方模块"
      "import 自建模块"
7. **工程文件目录建在 python 安装目录下，使用"import"语句导入模块时建议使用绝对路径**
8. **空行的使用：**
   a. 类外函数之间： 空 2 行
   b. 类内 **methods** 之间： 空 **1** 行
   c. 各类之间： 空 **2** 行
9. **空格的使用：**
   a. 避免在**[], (), {}**内与括号相接触的位置使用空格，类似**['a'], ('b')**等形式都要避免（在语法上合法，看个人习惯，有时候**{ 'c' }**更清晰）
   b. 双目操作符前后各加一个空格（建议有且仅有一个空格，不建议使用多个或无空格）
   c. 特殊地，**function** 或 **method** 参数初始化时，**"="** 前后都不加空格
10. **关于注释：**
    a. 注释语言统一为英语
    b. 注释用' # '开头。（提醒：' #' 后要跟一个空格）
    c. 同一注释行中，句子间用两个空格
    d. 各注释行间若有空白行，同样以' #'作行首
    e. 若代码本身简单易懂，或命名本身实现了自注释，勿加多余注释
    f. 代码修改同时要修改对应注释，否则危害很大，同代码不一致的注释比没有更糟
    g. 注释是完整句子时，首单词的首字母大写；注释较短时可省去末尾的句号

11. 公共 module, function, class, method 中都要有 docstring（用'''....'''形式）；私有 method 要在"def" 语句后加注释说明

12. 命名规则

a. 保持一致性，至少保持自建代码的一致性
b. 避免使用单个字母做变量，尤其避免字母 l，o 用作变量,易和数字混淆!
c. 命名字符串不要太长。注:本次会议中，该指标没有量化
d. 类名采用首字母大写的各单词直接相连方式命名。例： StudentBoy
e. 函数名第二个单词起大写 getStudentName() 必要是可以用下划线 get_name()
f. 非类名采用字母均小写的各单词以下划线相连的方式命名。例：student_boy
g. 属内部的 function, class, method 命名要以下划线开头。_local_time,_LocalTime
h. 模块名称用小写 module.py, 注意下会发现导入的模块的名称一般都是小写的
i. 异常相关的命名用"Error"或"error"开头
j. 全局变量与 function 命名规则一样
k. 类内默认参数定义为'self'
l. 当对参数命名可能发生重名时，在原名后加下划线作新名。例：print_
m. 常量命名采用大写的各单词以下划线相连方式命名。例：MAX_NAME_LENGTH
n. 缩写，单词较长时可以用缩写，常用去辅音，如 function -> fn

最后总结

Python 中有数据类型的分类：

1. 序列和非序列(序列可以用于 for in 循环，进行迭代)

序列：字符串，列表，元组，集合，字典 **非序列**：数字（整型，浮点型等），布尔值

2. 按可变和不可变分：（不可变是指变量重新赋值后 id 发生了变化）

可变：列表，集合，字典 **不可变**：字符串 **不能修改**：元组

参考文献

1. Python 优秀资源汇总：http://dwz.cn/learn_python (在此网址获取以下内容)
2. 《byte of python》
3. 《Dive into Pyhon3》
4. http://docs.python.org/library/ctypes.html
5. http://developer.51cto.com/art/201006/204838.htm
6. Thread: http://blog.csdn.net/jgood/archive/2009/06/26/4299476.aspx
7. Cython: http://gashero.javaeye.com/blog/649516
8. python 一些常用的东西：http://www.besttome.com/html/useful_in_python.html
9. http://www.besttome.com/html/python_bif_filter_map_reduce_lambda.html
10. 文件目录操作：http://l90z11.blog.163.com/blog/static/18738904220131215331838 9/
11. http://www.liaoxuefeng.com/ 廖雪峰的 python 教程（非常好，推荐）
12. 自强学堂 Python 教程：http://www.ziqiangxuetang.com/python/