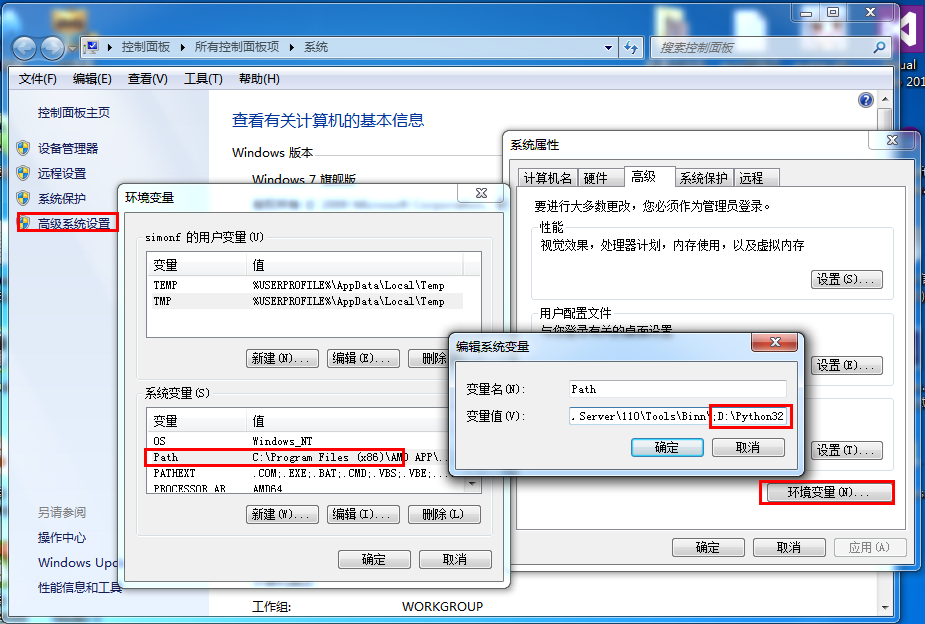
# 第一章 python基础

## 1、python初识

python官网下载地址：https://www.python.org/downloads/

在Windows上安装需要设置环境变量，设置方法如下：

右键点击"计算机"，然后点击"属性"，然后点击"高级系统设置"，选择"系统变量"窗口下面的"Path",双击即可！，然后在"Path"行，添加python安装路径即可(我的D:\Python32)，所以在后面，添加该路径即可。 ps：记住，路径直接用分号"；"隔开！



最后设置成功以后，在cmd命令行，输入命令"python"，就可以有相关显示。

**运行python的工具**

python工具有很多这里推荐2个（PyCharm和visual studio code）都可以百度下载，

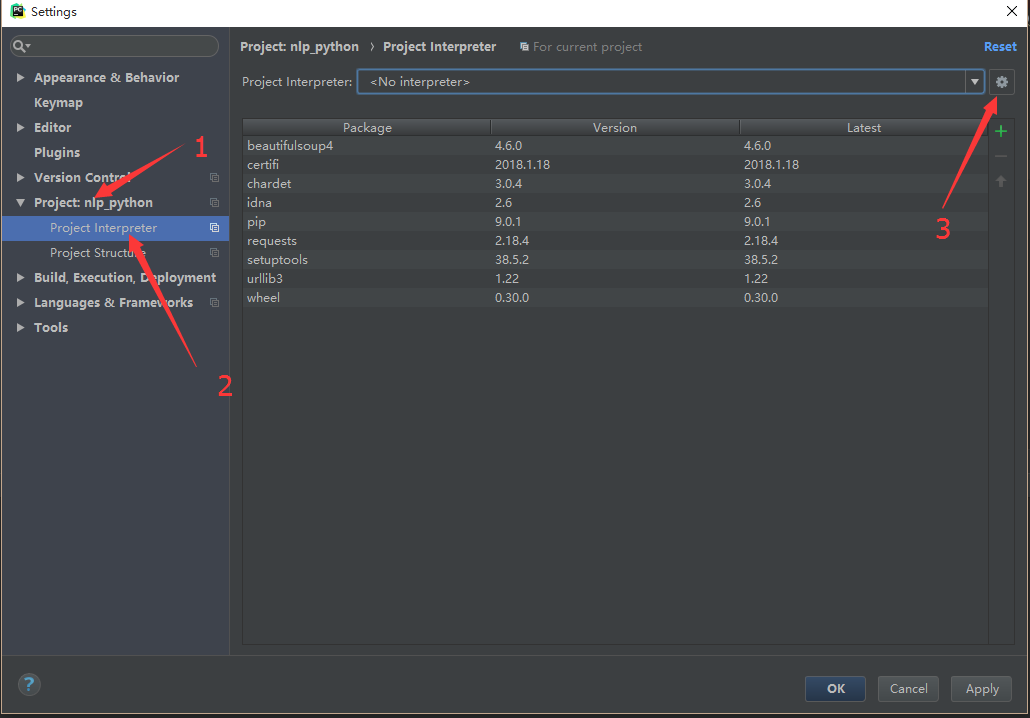
PyCharm：是需要收费的：下载地址：

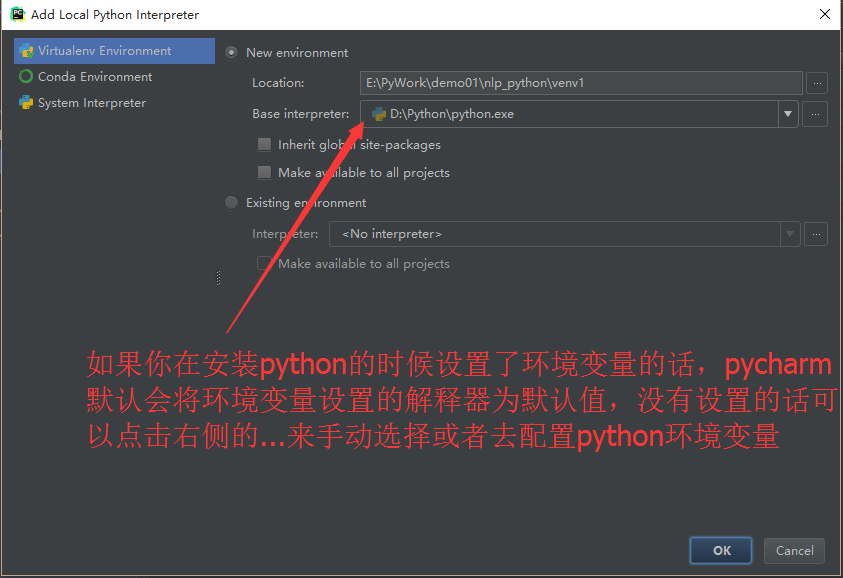
visual studio code：免费的工作，下载地址：https://code.visualstudio.com/

我们这里使用Pycharm工具，破解方法参考：https://blog.csdn.net/u014044812/article/details/78727496

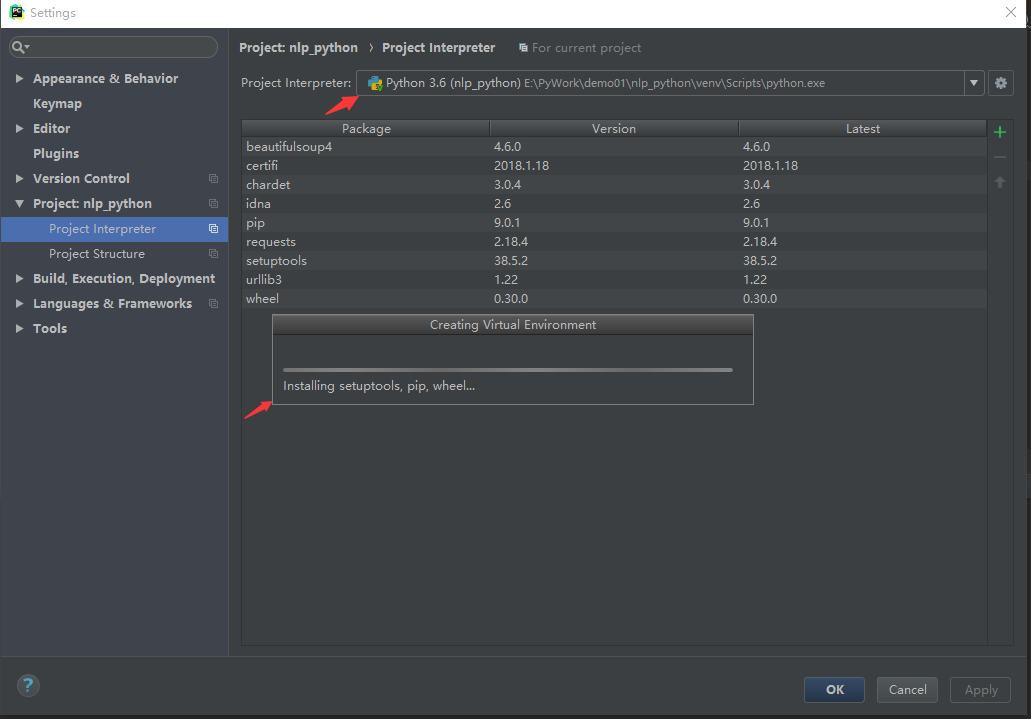
刚开始需要配置pycharm的python环境

点击File->Setting->Project:your project name->Project Interpreter->右侧小圆圈->Add Local





也可以选择系统解释器，一样的。点击OK之后，如下，过一会就好啦



关于python的学习都可以参考Alex的博客

博客地址：http://www.cnblogs.com/alex3714/articles/5885096.html/

## 2、基本语法/流程控制

参考：http://www.cnblogs.com/alex3714/articles/5465198.html

### 2.1、变量

a = 1+3\*4  
print (a)

输出结果是：13

--------使用end关键字参数，就会打印在一起----------

print('hello ',end='')  
print('world') #输出结果hello world

----------使用sep关键字参数，替换默认的分隔字符串-------------------

print('can','dogs','mice',sep='\_\_') #输出结果can\_\_dogs\_\_mice

上面可以看出变量名为 a ，变量值为 1+3+4 ，print是打印

**漂亮的打印**

如果程序中导入 pprint 模块， 就可以使用 pprint()和 pformat()函数，将会漂亮打印出一个字典了。

*import* pprint  
message = 'It was a bright cold day in April, and the clocks were striking thirteen.'  
count = {}  
*for* character *in* message:  
 count.setdefault(character, 0)  
 count[character] = count[character] + 1

pprint.pprint(count) #

print(pprint.pformat(count)) #

输出如下样的形式

'r': 5,

's': 3,

't': 6,

'w': 2,

'y': 1}

**变量定义规则：**

1.变量名只能是 字母 数字 或者下划线的任意组合

2.变量名的第一个字符不能是数字

3.以下关键字不能设置为变量名： 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield'

**python里的内容注释：**

当前行注释：# 被注释内容

多行注释：""" 被注释内容 """ （前面三个单引号或者双引号，后面三个单引号或双引号）

### 2.2、用户输入

name = input("你的名字：")  
print("hello " + name)

输出的结果如下：

你的名字：fan

hello fan

可以看出input是提示用户输入内容的

|  |
| --- |
| 在行末使用续行字符\，将一条指令写成多行。可以把\看成是“这条指令在下一行继续”。\续行字符之后的一行中，缩进并不重要 |

### 2.3、数据类型简介

python中6个标准的数据类型：

不可变数据：数字、字符串、元组

可变数据：列表、字典、集合

#### 2.3.1、数字（number）：

int（整型） 如 1 2 3 就表示整数类型

long（长整型）python3 没有长整型了

float（浮点型） 如 1.2 2.3 带小数点的都是浮点型

complex（复数）如1 + 2j 1.1+2.2j

#### 2.3.2、布尔值（bool）

真或假

0或者1

bool(......) 可以转成布尔值

None : 列表，元组 字典 0 返回空的话 都是为假 False

#### 2.3.3、字符串（String）

python中单引号和双引号使用完全相同

字符串可以用 + 运算符连接在一起，用 \* 运算符重复。

Python 中的字符串有两种索引方式，从左往右以 0 开始，从右往左以 -1 开始。

Python中的字符串不能改变。

Python 没有单独的字符类型，一个字符就是长度为 1 的字符串。

字符串的截取的语法格式如下：变量[头下标:尾下标]

word = '字符串'

sentence = "这是一个句子。"

paragraph = """这是一个段落，

可以由多行组成"""

**字符串格式化输出（占位符）**

name = "alex"

print "i am %s " % name #格式化输出字符串是 %s 整数 %d 浮点数 %f

#输出: i am alex

**字符串常用功能**：移除空白，分割，长度，索引，切片

#### 2.3.4、列表（list）

创建列表的方式：

name\_list = ['fangx','hmm','fzy']

或者

name\_list = list(['fangx','hmm','fzy'])

**基本操作：**索引，切片，追加，删除，长度，切片，循环，包含

#### 2.3.5、元组（tuple）

元组是不可变的列表，创建元组方式：

ages = ('fangx','hmm','fzy')

或者

ages = tuple(('fangx','hmm','fzy'))

#### 2.3.6、字典（dictionary）

字典是无序的，创建字典方式：

person = {"name": "mr.wu", 'age': 18}

或者

person = dict({"name": "mr.wu", 'age': 18})

**基本操作：**索引,新增,删除,键、值、键值对,循环,长度

#### 2.3.7、集合（set）

集合是无序不重复的，创建方式：

student = {'Tom', 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}

print(student) # 输出集合，重复的元素被自动去掉

#### 2.3.8、数据类型转换

有时候，我们需要对数据内置的类型进行转换，数据类型的转换，你只需要将数据类型作为函数名即可。

以下几个内置的函数可以执行数据类型之间的转换。这些函数返回一个新的对象，表示转换的值。

|  |  |
| --- | --- |
| **函数** | **描述** |
| int(x [,base]) | 将x转换为一个整数 |
| float(x) | 将x转换到一个浮点数 |
| complex(real [,imag]) | 创建一个复数 |
| str(x) | 将对象 x 转换为字符串 |
| repr(x) | 将对象 x 转换为表达式字符串 |
| eval(str) | 用来计算在字符串中的有效Python表达式,并返回一个对象 |
| tuple(s) | 将序列 s 转换为一个元组 |
| list(s) | 将序列 s 转换为一个列表 |
| set(s) | 转换为可变集合 |
| dict(d) | 创建一个字典。d 必须是一个序列 (key,value)元组。 |
| frozenset(s) | 转换为不可变集合 |
| chr(x) | 将一个整数转换为一个字符 |
| ord(x) | 将一个字符转换为它的整数值 |
| hex(x) | 将一个整数转换为一个十六进制字符串 |
| oct(x) | 将一个整数转换为一个八进制字符串 |

### 2.4、数据运算符

以下运算符都是a=10,b=21

#### 2.4.1、算术运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| + | 加 - 两个对象相加 | a + b 输出结果 31 |
| - | 减 - 得到负数或是一个数减去另一个数 | a - b 输出结果 -11 |
| \* | 乘 - 两个数相乘或是返回一个被重复若干次的字符串 | a \* b 输出结果 210 |
| / | 除 - x 除以 y | b / a 输出结果 2.1 |
| % | 取模 - 返回除法的余数 | b % a 输出结果 1 |
| \*\* | 幂 - 返回x的y次幂 | a\*\*b 为10的21次方 |
| // | 取整除 - 返回商的整数部分 | 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0 |

#### 2.4.2、比较运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| == | 等于 - 比较对象是否相等 | (a == b) 返回 False。 |
| != | 不等于 - 比较两个对象是否不相等 | (a != b) 返回 True。 |
| > | 大于 - 返回x是否大于y | (a > b) 返回 False。 |
| < | 小于 - 返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意，这些变量名的大写。 | (a < b) 返回 True。 |
| >= | 大于等于 - 返回x是否大于等于y。 | (a >= b) 返回 False。 |
| <= | 小于等于 - 返回x是否小于等于y。 | (a <= b) 返回 True。 |

#### 2.4.3、赋值运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| = | 简单的赋值运算符 | c = a + b 将 a + b 的运算结果赋值为 c |
| += | 加法赋值运算符 | c += a 等效于 c = c + a |
| -= | 减法赋值运算符 | c -= a 等效于 c = c - a |
| \*= | 乘法赋值运算符 | c \*= a 等效于 c = c \* a |
| /= | 除法赋值运算符 | c /= a 等效于 c = c / a |
| %= | 取模赋值运算符 | c %= a 等效于 c = c % a |
| \*\*= | 幂赋值运算符 | c \*\*= a 等效于 c = c \*\* a |
| //= | 取整除赋值运算符 | c //= a 等效于 c = c // a |

#### 2.4.4、位运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| & | 按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0 | (a & b) 输出结果 12 ，二进制解释： 0000 1100 |
| | | 按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。 | (a | b) 输出结果 61 ，二进制解释： 0011 1101 |
| ^ | 按位异或运算符：当两对应的二进位相异时，结果为1 | (a ^ b) 输出结果 49 ，二进制解释： 0011 0001 |
| ~ | 按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1。**~x**类似于 **-x-1** | (~a ) 输出结果 -61 ，二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。 |
| << | 左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。 | a << 2 输出结果 240 ，二进制解释： 1111 0000 |
| >> | 右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数 | a >> 2 输出结果 |

#### 2.4.5、逻辑运算符

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **逻辑表达式** | **描述** | **实例** |
| and | x and y | 布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。 | (a and b) 返回 20。 |
| or | x or y | 布尔"或" - 如果 x 是 True，它返回 x 的值，否则它返回 y 的计算值。 | (a or b) 返回 10。 |
| not | not x | 布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。 | not(a and b) 返回 False |

#### 2.4.6、成员运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| in | 如果在指定的序列中找到值返回 True，否则返回 False。 | x 在 y 序列中 , 如果 x 在 y 序列中返回 True。 |
| not in | 如果在指定的序列中没有找到值返回 True，否则返回 False。 | x 不在 y 序列中 , 如果 x 不在 y 序列中返回 True |

#### 2.4.7、身份运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| is | is 是判断两个标识符是不是引用自一个对象 | **x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False |
| is not | is not 是判断两个标识符是不是引用自不同对象 | **x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。 |

#### 2.4.8、优先级运算符

|  |  |
| --- | --- |
| **运算符** | **描述** |
| \*\* | 指数 (最高优先级) |
| ~ + - | 按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@) |
| \* / % // | 乘，除，取模和取整除 |
| + - | 加法减法 |
| >> << | 右移，左移运算符 |
| & | 位 'AND' |
| ^ | | 位运算符 |
| <= < > >= | 比较运算符 |
| <> == != | 等于运算符 |
| = %= /= //= -= += \*= \*\*= | 赋值运算符 |
| is is not | 身份运算符 |
| in not in | 成员运算符 |
| and or not | 逻辑运算符 |

### 2.5、表达式

#### 2.5.1、if条件语句

语法形式：

if condition\_1:

statement\_block\_1

elif condition\_2:

statement\_block\_2

else:

statement\_block\_3

**注意：**

1、每个条件后面要使用冒号 :，表示接下来是满足条件后要执行的语句块。

2、使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。

**示例参考：**

age = int(input("请输入你家狗狗的年龄: ")) #必须是数字(int)，输入字符串会报错  
*if* age < 0:  
 print("你是在逗我吧!")  
*elif* age == 1:  
 print("相当于 14 岁的人。")  
*elif* age == 2:  
 print("相当于 22 岁的人。")  
*elif* age > 2:  
 human = 22 + (age - 2) \* 5  
 print("对应人类年龄: ", human)  
# 退出提示  
input("点击 enter 键退出")

**下列输入字符串会提示你输入数字，需要if嵌套**

age = input("请输入你家狗狗的年龄: ")  
*if* age.isdigit(): #添加了isdigit 内置函数，排断是不是数字  
 age = int(age)  
 *if* age < 0:  
 print("你是在逗我吧!")  
 *elif* age == 1:  
 print("相当于 14 岁的人。")  
 *elif* age == 2:  
 print("相当于 22 岁的人。")  
 *elif* age > 2:  
 human = 22 + (age - 2) \* 5  
 print("对应人类年龄: ", human)  
*else* :  
 print("请输入数字")  
# 退出提示  
input("点击 enter 键退出")

**if嵌套的示例**

num=int(input("输入一个数字："))  
*if* num%2==0:  
 *if* num%3==0:  
 print ("你输入的数字可以整除 2 和 3")  
 *else*:  
 print ("你输入的数字可以整除 2，但不能整除 3")  
*else*:  
 *if* num%3==0:  
 print ("你输入的数字可以整除 3，但不能整除 2")  
 *else*:  
 print ("你输入的数字不能整除 2 和 3")

#### 2.5.2、while循环

while循环叫做死循环，一旦触发就会一直运行

**写法：**

count = 0 #计数器初始值，一般设为0，  
*while True*: #条件为真一直运行，如果不想一直运行 就设置条件如运行10次 count<10   
 print("hello python",count)  
 count +=1 #运行一次加1个

**参考：**

nian = -1  
ling = 20  
count = 0  
*while* nian != ling *and* count < 5: #运行5次，不加 count < 5就会一直运行  
 nian = int(input("猜你数字："))  
 *if* nian < ling:  
 print("猜小了")  
 #break  
 *elif* nian > ling:  
 print("猜大了")  
 *else*:  
 print("猜对了")  
 count += 1  
*else*:  
 print("游戏结束")

#### 2.5.3、for循环

**简单示例：**

*for* i *in* range(10): #运行10次输出0 到9的数字  
 print(i)

----示例2--------------------------------------------------------------------

*for* i *in* range(10):  
 *if* i < 5:  
 *continue*  #结束本次循环直接进入下次循环

print(i) #小于*5*的都不输出

----示例3-------------------------------------------------

*for* i *in* range(10):  
 *if* i > 5:  
 *break* #结束循环print(i) #输出1 到 5

**猜数字示例：**

nian = -1  
ling = 20  
*for* i *in* range(3):  
 nian = int(input("猜你数字："))  
 *if* nian < ling:  
 print("猜小了")  
 *elif* nian > ling:  
 print("猜大了")  
 *else*:  
 print("猜对了")  
 *break  
else*:  
 print("游戏结束")

**多级break：**

# fiag = *False  
for* i *in* range(10):  
 *if* i <5:  
 *continue* print(i)  
 *for* j *in* range(10):  
 print("2j",j)  
 *if* j ==6:  
 fiag = *True   
 break #跳出  
 if* fiag:  
 *break #发现上面跳出也跟着跳出*

**使用正则表达式的search()函数实现指定位置搜索**

参考：https://blog.csdn.net/caimouse/article/details/78262846

**查找电话号码（参考114）**

*import* re  
phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')  
mo = phoneNumRegex.search('My number is 415-555-4242.')  
print( 'Phone number found: '+mo.group())

## 3、列表、字典、集合

参考：http://www.cnblogs.com/alex3714/articles/5717620.html

### 3.1、字符串操作

字符串是最常用的数据类型，可以用单引号‘’或者双引号“”创建。特性：不可修改

**创建方法**

a = "fangx"  
b = 'huangmm'

字符串也可以用中括号截取字符串

print(a[1:]) #从下标1开始到最后  
print(b[:5]) #从第一个开始到5

**转义字符：**在字符中使用特殊字符时，用反斜杠 \ 转义字符

|  |  |
| --- | --- |
| **转义字符** | **描述** |
| \(在行尾时) | 续行符 |
| \\ | 反斜杠符号 |
| \' | 单引号 |
| \" | 双引号 |
| \a | 响铃 |
| \b | 退格(Backspace) |
| \e | 转义 |
| \000 | 空 |
| \n | 换行 |
| \v | 纵向制表符 |
| \t | 横向制表符（空格） |
| \r | 回车 |
| \f | 换页 |
| \oyy | 八进制数，yy代表的字符，例如：\o12代表换行 |
| \xyy | 十六进制数，yy代表的字符，例如：\x0a代表换行 |
| \other | 其它的字符以普通格式输出 |

**字符串运算符:** a=Hello b=Python

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **实例** |
| + | 字符串连接 | a + b 输出结果： HelloPython |
| \* | 重复输出字符串 | a\*2 输出结果：HelloHello |
| [] | 通过索引获取字符串中字符 | a[1] 输出结果 **e** |
| [ : ] | 截取字符串中的一部分 | a[1:4] 输出结果 **ell** |
| in | 成员运算符 - 如果字符串中包含给定的字符返回 True | **'H' in a** 输出结果 1 |
| not in | 成员运算符 - 如果字符串中不包含给定的字符返回 True | **'M' not in a** 输出结果 1 |
| r/R | 原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母 **r**（可以大小写）以外，与普通字符串有着几乎完全相同的语法。 | print( r'\n' ) print( R'\n' ) |
| % | 格式字符串 |  |

**字符串格式化**

print("我叫%s ,今年%d岁。" % ("小明",10))

**格式化符号**

|  |  |
| --- | --- |
| **符   号** | **描述** |
| %c | 格式化字符及其ASCII码 |
| %s | 格式化字符串 |
| %d | 格式化整数 |
| %u | 格式化无符号整型 |
| %o | 格式化无符号八进制数 |
| %x | 格式化无符号十六进制数 |
| %X | 格式化无符号十六进制数（大写） |
| %f | 格式化浮点数字，可指定小数点后的精度 |
| %e | 用科学计数法格式化浮点数 |
| %E | 作用同%e，用科学计数法格式化浮点数 |
| %g | %f和%e的简写 |
| %G | %f 和 %E 的简写 |
| %p | 用十六进制数格式化变量的地址 |

**字符串的内置函数**

a = "hello python"  
  
print(a.capitalize()) #首字母大写  
print(a.casefold()) #大写变小写  
print(a.center(50,"-")) #输出 -----------hello python---------  
print(a.count("h")) #统计h出现了几次  
print(a.endswith("on")) #排断是否on结尾，是就返回真（true）  
print(a.find("p")) #查找第一个h所在的位置  
print(a.swapcase()) #大小写互换

**重点记忆：**replace/find/join/strip/startswith/split/upper/format/

|  |  |
| --- | --- |
| **方法** | **描述** |
| capitalize() | 将字符串的第一个字符转换为大写 |
| center(width, fillchar) | 返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。 |
| count(str, beg= 0,end=len(string)) | 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str  出现的次数 |
| bytes.decode(encoding="utf-8", errors="strict") | Python3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。 |
| encode(encoding='UTF-8',errors='strict') | 以 encoding 指定的编码格式编码字符串，如果出错默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace' |
| endswith(suffix, beg=0, end=len(string)) | 检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False. |
| expandtabs(tabsize=8) | 把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8 。 |
| find(str, beg=0 end=len(string)) | 检测 str 是否包含在字符串中，如果指定范围 beg 和 end ，则检查是否包含在指定范围内，如果包含返回开始的索引值，否则返回-1 |
| index(str, beg=0, end=len(string)) | 跟find()方法一样，只不过如果str不在字符串中会报一个异常. |
| isalnum() | 如果字符串至少有一个字符并且所有字符都是字母或数字则返 回 True,否则返回 False |
| isalpha() | 如果字符串至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False |
| isdigit() | 如果字符串只包含数字则返回 True 否则返回 False.. |
| islower() | 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False |
| isnumeric() | 如果字符串中只包含数字字符，则返回 True，否则返回 False |
| isspace() | 如果字符串中只包含空白，则返回 True，否则返回 False. |
| istitle() | 如果字符串是标题化的(见 title())则返回 True，否则返回 False |
| isupper() | 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False |
| join(seq) | 以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串 |
| len(string) | 返回字符串长度 |
| ljust(width[, fillchar]) | 返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格。 |
| lower() | 转换字符串中所有大写字符为小写. |
| lstrip() | 截掉字符串左边的空格或指定字符。 |
| maketrans() | 创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。 |
| max(str) | 返回字符串 str 中最大的字母。 |
| min(str) | 返回字符串 str 中最小的字母。 |
| replace(old, new [, max]) | 把 将字符串中的 str1 替换成 str2,如果 max 指定，则替换不超过 max 次。 |
| rfind(str, beg=0,end=len(string)) | 类似于 find()函数，不过是从右边开始查找. |
| rindex( str, beg=0, end=len(string)) | 类似于 index()，不过是从右边开始. |
| rjust(width,[, fillchar]) | 返回一个原字符串右对齐,并使用fillchar(默认空格）填充至长度 width 的新字符串 |
| rstrip() | 删除字符串字符串末尾的空格. |
| split(str="", num=string.count(str)) | num=string.count(str)) 以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num 个子字符串 |
| splitlines([keepends]) | 按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。 |
| startswith(str, beg=0,end=len(string)) | 检查字符串是否是以 obj 开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查。 |
| strip([chars]) | 在字符串上执行 lstrip()和 rstrip() |
| swapcase() | 将字符串中大写转换为小写，小写转换为大写 |
| title() | 返回"标题化"的字符串,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle()) |
| translate(table, deletechars="") | 根据 str 给出的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 deletechars 参数中 |
| upper() | 转换字符串中的小写字母为大写 |
| zfill (width) | 返回长度为 width 的字符串，原字符串右对齐，前面填充0 |
| isdecimal() | 检查字符串是否只包含十进制字符，如果是返回 true，否则返回 false。 |

### 3.2、列表的操作

列表是我们最常用的数据类型之一，通过列表可以对数据类型实现最方便的存储、修改等操作，列表的索引号从0开始。

**列表函数和方法如下表**

|  |
| --- |
| **函数** |
| len(list) 列表元素个数 |
| max(list) 返回列表元素最大值 |
| min(list) 返回列表元素最小值 |
| list(seq) 将元组转换为列表 |
| **方法** | |
| list.append(obj) 在列表末尾添加新的对象 | |
| list.count(obj) 统计某个元素在列表中出现的次数 | |
| list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表） | |
| list.index(obj) 从列表中找出某个值第一个匹配项的索引位置 | |
| list.insert(index, obj) 将对象插入列表 | |
| list.pop([index=-1]]) 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值 | |
| list.remove(obj) 移除列表中某个值的第一个匹配项 | |
| list.reverse() 反向列表中元素 | |
| list.sort(cmp=None, key=None, reverse=False) 对原列表进行排序 | |
| list.clear() 清空列表 | |
| list.copy() 复制列表 | |

**定义列表：**

names = ['fangx','hmm','fanzy','huangmm','xiaom','htian']

print(names[0]) #从左到右 数字从0开始查找，显示fangx  
print(names[-1]) #从右到左，从-1开始查找，显示huangmm

**切片操作**

print(names[1:4]) #取1到4之前的值，包括1，但不包括4  
print(names[1:-1]) #取1到-1之前的值，不包括-1  
print(names[1:]) #取1到最后的值

print(names[:3]) #取0到3之间的值，不包括3  
print(names[1::2]) #取从1到最后，隔一个取一个，

**追加**

列表名.append(“新加的内容”)，只能追加一个内容到最后

列表名.extend(),可以加多个，需要提取定义好列表

names.append("新加的")

b=[1,"nih",3,4,5] #新加一个列表  
names.extend(b) #把b列表的所有值，加到names里

names += [1,2,3,4] #这样操作也可以把值加到names列表里

**插入**

可以在任意位置插入新的内容

names.insert(2,"fanxh") #从位置2的值，前面插入

**修改**

names[2] = "xiugai" #修改位置2的值  
print(names)

**删除**

*del* names[3] #删除位置3的值  
names.remove("xiaom") #删除指定的值  
names.pop() #删除最后一个值，填写下标，可以删除指定的位置值

**深浅拷贝**

参考：https://jingyan.baidu.com/article/b0b63dbf3ea8d64a483070d6.html

*import* copy   
spam = ['A','B','C','D']  
cheese =copy.copy(spam) #也可以这样写spam.copy(),表示创建了第二个列表，能独立于第一个列表修改  
cheese[1] ='2' #如果要复制的列表中包含了列表， 那就使用 copy.deepcopy()函数来代替  
print(spam) #输出结果：['A', 'B', 'C', 'D']  
print(cheese) #输出结果：['A', '2', 'C', 'D']

**统计**

print(names.count("hmm")) #统计names列表的值 hmm 出现了几次

**排序**

names.sort() #排序，需要注意不同的数据类型不能放在一起排序  
names.reverse() #翻转，倒着排列

**获取下标**

print(names.index("hmm")) #查找hmm在什么位置

**神奇的8球**

*import* random  
messages = ['It is certain',  
'It is decidedly so',  
'Yes definitely',  
'Reply hazy try again',  
'Ask again later',  
'Concentrate and ask again',  
'My reply is no',  
'Outlook not so good',  
'Very doubtful']  
print(messages[random.randint(0, len(messages))])

### 3.3、元组的操作

元组和列表类似，不同之处就是不能修改，所以元组也成为只读的列表，元组用小括号创建，列表是使用中括号，字典和集合都是大括号。它只有2个方法：count和index

**创建元组**

names = ('fangx','hmm','fanzy','huangmm','xiaom','htian')

**元组包含的内置函数**

|  |  |
| --- | --- |
| **方法及描述** | **实例** |
| len(tuple) | >>> tuple1 = ('Google', 'Runoob', 'Taobao') |
| 计算元组元素个数。 | >>> len(tuple1) |
|  | 3 |
|  | >>> |
| max(tuple) | >>> tuple2 = ('5', '4', '8') |
| 返回元组中元素最大值。 | >>> max(tuple2) |
|  | '8' |
|  | >>> |
| min(tuple) | >>> tuple2 = ('5', '4', '8') |
| 返回元组中元素最小值。 | >>> min(tuple2) |
|  | '4' |
|  | >>> |
| tuple(seq) | >>> list1= ['Google', 'Taobao', 'Runoob', 'Baidu'] |
| 将列表转换为元组。 | >>> tuple1=tuple(list1) |
|  | >>> tuple1 |
|  | ('Google', 'Taobao', 'Runoob', 'Baidu') |

### 3.4、集合的操作

集合是一个无序不重复元素的序列，使用大括号创建，创建空集合必须是set(),因为{ }是用来创建空字典的。主要作用：去重，和关系测试。**创建集合方式如下**：

a = {1,2,3,4,5,}  
b = set({"a","b"})

**集合的操作**

a.add("a") #添加一个元素  
a.update("b","c","d") #添加多个元素，这个参数适用 列表 元组 字典等

a.remove(1) #移出值为 1 的元素

a.discard(2) # 这个方式如果元素不存在 也不会报错

a.pop() #随机删除

print(len(a)) #计算有多少个元素

a.clear() #清空集合

**关系测试**

print(a-b) # 集合a中包含的元素 减去集合b中的元素后集合a中的元素  
print(a|b) #集合a和集合b所有的元素  
print(a&b) #集合a和集合b都包含的元素  
print(a^b) #集合a和集合b都不包含的元素

### 3.5、字典的操作

字典是另一种可变容器模型，且可存储任意类型对象，字典的每个键值（key-value）对用冒号（：）分隔，键必须是唯一且必须是不可变的（如字符串，数字或者元组），但值则不必，可以是任意数据类型。特点就是无序，和key是唯一，天生去重。**创建方式如下：**

a = {'a':1,'b':2,'c':3,'d':4} #创建字典  
print(a) #打印所有字典的内容  
print(a['a']) #打印键a 的值 返回的结果就是 1

**增加**

a['e']=5 #在打印字典a的时候 就会多出 e:5的内容

**修改**

a['a']=0 #这样就把key a的值修改为了0

**删除**

a.pop("a") #删除字典键a

print(a) #打印出来后键a和值都删除了  
*del* a["b"] #另一种删除方式

a.clear() #清空字典

**查找**

print("b" *in* a) #返回真（True）或者假（Frea）  
print(a.get("a"))  
print(a["a"])

**方法**

**字典内置函数和方法**

|  |  |
| --- | --- |
| **函数及描述** | **实例** |
| len(dict) | >>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'} |
| 计算字典元素个数，即键的总数。 | >>> len(dict) |
| 3 |
| str(dict) | >>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'} |
| 输出字典，以可打印的字符串表示。 | >>> str(dict) |
| "{'Name': 'Runoob', 'Class': 'First', 'Age': 7}" |
| type(variable) | >>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'} |
| 返回输入的变量类型，如果变量是字典就返回字典类型。 | >>> type(dict) |
| <class 'dict'> |

**方法**

|  |  |
| --- | --- |
| **函数** | **描述** |
| radiansdict.clear() | 删除字典内所有元素 |
| radiansdict.copy() | 返回一个字典的浅复制 |
| radiansdict.fromkeys() | 创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值 |
| radiansdict.get(key, default=None) | 返回指定键的值，如果值不在字典中返回default值 |
| key in dict | 如果键在字典dict里返回true，否则返回false |
| radiansdict.items() | 以列表返回可遍历的(键, 值) 元组数组 |
| radiansdict.keys() | 返回一个迭代器，可以使用 list() 来转换为列表 |
| radiansdict.setdefault(key,值) | 和get()类似, 但如果键不存在于字典中，不写值，将会添加键并将值设为default（none） |
| radiansdict.update(dict2) | 把字典dict2的键/值对更新到dict里 |
| radiansdict.values() | 返回一个迭代器，可以使用 list() 来转换为列表 |
| pop(key[,default]) | 删除字典给定键 key 所对应的值，返回值为被删除的值。key值必须给出。 否则，返回default值。 |
| popitem() | 随机返回并删除字典中的一对键和值(一般删除末尾对)。 |

### 3.6、文件的操作

打开文件的模式有：r：只读模式（默认）；w：只写模式。【不可读；不存在则创建；存在则删除内容；】 a：追加模式。【可读；   不存在则创建；存在则只追加内容；】

"+" 表示可以同时读写某个文件；r+，可读写文件。【可读；可写；可追加】；w+，写读；a+，同a

"U"表示在读取时，可以将 \r \n \r\n自动转换成 \n （与 r 或 r+ 模式同使用）rU和r+U

"b"表示处理二进制文件（如：FTP发送上传ISO镜像文件，linux可忽略，windows处理二进制文件时需标注）rb、wb、ab

**基本操作**

f = open('fangx.txt',encoding="UTF-8") #打开fangx.txt的文件，需要指定字符编码  
f\_line=f.readline() #读取一行  
print(f\_line) #打印出来  
a = f.read() #读取所有内容

f.close() #关闭文件，操作文件最后一定要加上

只取某一行

*import* linecache  
cou = linecache.getline('fangx.txt',2)  
print(cou)

---排断文件是否存在用exists()

*from* os.path *import* exists  
b,c = "fangx.txt","ss"  
print("%r" % exists(c))

**常用的函数和方法**

|  |  |
| --- | --- |
| **方法** | **描述** |
| file.close() | 关闭文件。关闭后文件不能再进行读写操作。 |
| file.flush() | 刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件, 而不是被动的等待输出缓冲区写入。 |
| file.fileno() | 返回一个整型的文件描述符(file descriptor FD 整型), 可以用在如os模块的read方法等一些底层操作上。 |
| file.isatty() | 如果文件连接到一个终端设备返回 True，否则返回 False。 |
| file.next() | 返回文件下一行。 |
| file.read([size]) | 从文件读取指定的字节数，如果未给定或为负则读取所有。 |
| file.readline([size]) | 读取整行，包括 "\n" 字符。 |
| file.readlines([sizeint]) | 读取所有行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比 sizeint 较大, 因为需要填充缓冲区。 |
| file.seek(offset[, whence]) | 设置文件当前位置 |
| file.tell() | 返回文件当前位置。 |
| file.truncate([size]) | 从文件的首行首字符开始截断，截断文件为 size 个字符，无 size 表示从当前位置截断；截断之后后面的所有字符被删除，其中 Widnows 系统下的换行代表2个字符大小。 |
| file.write(str) | 将字符串写入文件，返回的是写入的字符长度。 |
| file.writelines(sequence) | 向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。 |

## 4、函数

参考：http://www.cnblogs.com/alex3714/articles/5740985.html

### 4.1、函数介绍

函数是指将一组语句的集合通过一个名字（函数名）封装起来，要执行这个函数，只需要调用函数名即可。

**特性：**减少重复的代码，使程序变得可扩展，使程序变得易维护。

**定义函数语法：**

def 函数名（参数列表）：

函数体

1、函数代码块以 def 关键词开头，后接函数标识符名称和小括号 ()。

2、任何传入参数和自变量必须放在小括号中间，小括号之间可以用于定义参数。

3、函数的第一行语句可以选择性地使用文档字符串，用于存放函数说明。

4、函数内容以冒号起始，并且缩进。

5、 return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

**简单示例**

*def* fan(): #函数名  
 print("hello python")  
fan() #调用函数

**带参数的**

*def* fan(*x*,*y*): #x,y表示形参   
 res = *x* \* *y  
 return* res #返回函数执行结果  
c = fan(4,6) #4,6表示实参  
print(c)

**形参**变量只是才被调用的时候才分配内存，在调用结束时，即刻释放所分配的内存单元。因此形参只在函数内部有效，函数调用结束返回主调用函数后则不能在使用该形参变量。

**实参**可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，他们都必须有确定的值，以便把这些值传给形参。因此应预先用赋值，输入等办法是参数获得确定值。

**返回值**

要想获取函数的执行结果，就可以用return语句把结果返回。

需要注意：

1、函数在执行过程中只要遇到return语句，就会停止执行并返回结果。（可以理解为return语句表示函数的结束）

2、如果未在函数中指定return，那么这个函数的返回值为None(没有)

**异常处理**

发生 ZeroDivisionError。根据错误信息中给出的行号，我们知道 spam()中的 return 语句导致了一个错误。错误可以由 try 和 except 语句来处理。那些可能出错的语句被放在 try 子句中。如果错误发生，程序执行就转到接下来的 except 子句开始处

*def* spam(*d*):  
 *try*:  
 *return* 42 / *d  
 except* ZeroDivisionError:  
 print("error,不能除0")  
print(spam(0))

---输出结果如下----------------

error,不能除0

None

当这个函数除0时就会报错，如果在try子句中的代码导致一个错误，程序执行就立即转到except子句的代码。

**return语句用法：**

*def* sum(*arg1*, *arg2*):  
 # 返回2个参数的和."  
 total = *arg1* + *arg2* print("函数内 : ", total)  
 *return* total  
  
total = sum(10,20)  
print ("函数外 : ", total)

### 4.2、默认参数

*def* stu(*name*,*age*,*course*,*country*="CN"): #默认参数，如果参数都一样，就可以在这里面指定，但必须在最后，调用的时候可以不  
 print("---注册学生信息---") #用写这个参数，如果写的话就覆盖这个默认参数  
 print("姓名：",*name*)  
 print("年龄：",*age*)  
 print("国籍：",*country*)  
 print("课程：",*course*)  
stu("王尼玛",18,"python") #位置传参，指定了默认参数，位置必须一一对应, 不能由缺少或者多余

stu("张三丰",17,"linux","CN") #位置传参，  
stu(course="linux",age=18,name="王善刚") #正常情况下，给需要按照顺序，不想按顺序就用关键字参数，

stu("王大村",22,"mysql","US") #US 就覆盖了 默认参数（CN）

### 4.3、非固定参数

当函数在定义的时候不确定用户传入什么参数，就可以使用非固定参数，**\*表示元组，\*\*表示字典**

*def* stu\_register(*name*, *age*, *\**args): # \*args 会把多传入的参数变成一个元组形式  
 print(*name*, *age*, *args*)  
stu\_register("Alex", 22)  
stu\_register("Jack", 32, "CN", "Python")

---------输出结果如下------------

Alex 22 () #没有传值，所以为空

Jack 32 ('CN', 'Python') #传了值变成了一个元组

字典的方式

*def* stu\_register(*name*, *age*, *\**args, *\*\**kwargs): # \*kwargs 会把多传入的参数变成一个dict形式  
 print(*name*, *age*, *args*, *kwargs*)  
stu\_register("Alex", 22)  
stu\_register("Jack", 32, "CN", "Python", sex="Male", province="ShanDong")  
stu\_register("Fan", 18, "CN", "Mysql", "Male","shanghai")

--------------输出结果如下-------------

Alex 22 () {} #没有传值

Jack 32 ('CN', 'Python') {'sex': 'Male', 'province': 'ShanDong'} #传了对应的值

Fan 18 ('CN', 'Mysql', 'Male', 'shanghai') {} #没有指定字典参数，都变成了元组

### 4.4、变量作用域

变量的作用域决定了在哪一部分程序可以访问哪个特定的变量名称。**Python的作用域一共有4种**，分别是：

1、L （Local） 局部作用域

2、E （Enclosing） 闭包函数外的函数中

3、G （Global） 全局作用域

4、B （Built-in） 内建作用域

以 L –> E –> G –>B 的规则查找，即：在局部找不到，便会去局部外的局部找（例如闭包），再找不到就会去全局找，再者去内建中找。

**全局与局部变量**

在子程序中定义的变量称为**局部变量**，在程序的一开始定义的变量（没有缩进的）称为**全局变量**。全局变量作用域是整个程序，局部变量作用域是定义该变量的子程序。当全局变量与局部变量同名时：在定义局部变量的子程序内，局部变量起作用，在其他地方全局变量起作用。

当内部作用域想修改外部作用域的变量时，就要用到global和nonlocal关键字了。

global是修改全局的变量，修改嵌套作用域（外层非全局作用域）中的变量则需要 nonlocal 关键字。

name = "fangx" #全局变量  
print("在吗",name) #打印全局变量  
*def* change\_name(*name*):  
 print("你好", *name*) #这里打印的还是全局变量  
 name = "huangmm" #函数内的局部变量  
 print("after change", *name*) #这里打印了局部变量  
change\_name(name)  
print("在外面看看name改了么?", name) #打印的还是全局变量

------输出结果----------

在吗 fangx

你好 fangx

after change huangmm

在外面看看name改了么? fangx

----------------------------------------------------

num = 1 #全局变量

def fun1():

global num # 需要使用 global 关键字声明,要写在顶头

print(num) #打印的还是全局

num = 123

print(num) #打印了局部变量

fun1()

print(num) #最后打印被修改成了局部变量

### 4.5、嵌套函数

name = "Alex"  
#---------------------------  
*def* change\_name():  
 name = "fangx"  
 #-----------------------  
 *def* change\_name2():  
 name = "hmm"  
 print("第3层打印", name)  
 change\_name2() # 调用内层函数  
 #-------------------------  
 print("第2层打印", name)  
change\_name()  
#----------------------------  
print("最外层打印", name)

### 4.6、递归函数

在函数内部，可以调用其他函数，如果一个函数在内部调用自身本事，这个函数就是递归函数。

**递归特性：**

1.必须有一个明确的结束条件

2.每次进入更深一层递归时，问题规模相比上次上次递归都应有所减少

3.递归效率不高，递归层次过多会导致栈溢出。（在计算机中，函数调用时通过栈（stack）这种数据结构实现的，每当进入一函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧，由于栈的大小不是无限的，所以，递归调用的次数过多，到导致栈溢出）

需要注意：递归的效率在某些场景会很低，但凡是递归可以实现的循环都可以实现。

可参考：http://www.cnblogs.com/lln7777/archive/2012/03/14/2396164.html

*def* calc(*n*):  
 print(*n*)  
 *if* int(*n* / 2) == 0:  
 *return n  
 return* calc(int(*n* / 2))  
calc(10)

-----------输出结果结果------------------

10

5

2

1

---------裴波那契数递归写法-----

*def* fibo(*n*):  
 *if n* <= 1:  
 *return n  
 return* fibo(*n*-1) + fibo(*n*-2)  
print(fibo(10))

**递归函数应用案例**，二分查找

data = [1, 3, 6, 7, 9, 12, 14, 16, 17, 18, 20, 21, 22, 23, 30, 32, 33, 35]  
  
*def* binary\_search(*dataset*, *find\_num*):  
 print(*dataset*)  
  
 *if* len(*dataset*) > 1:  
 mid = int(len(*dataset*) / 2)  
 *if dataset*[mid] == *find\_num*: # find it  
 print("找到数字", *dataset*[mid])  
 *elif dataset*[mid] > *find\_num*: # 找的数在mid左面  
 print("\033[31;1m找的数在mid[%s]左面\033[0m" % *dataset*[mid])  
 *return* binary\_search(*dataset*[0:mid], *find\_num*)  
 *else*: # 找的数在mid右面  
 print("\033[32;1m找的数在mid[%s]右面\033[0m" % *dataset*[mid])  
 *return* binary\_search(*dataset*[mid + 1:], *find\_num*)  
 *else*:  
 *if dataset*[0] == *find\_num*: # find it  
 print("找到数字啦", *dataset*[0])  
 *else*:  
 print("没的分了,要找的数字[%s]不在列表里" % *find\_num*)  
  
binary\_search(data, 66)

### 4.7、匿名函数

匿名函数就是不在使用def语句这样的标准形式来定义一个函数,使用lambda来创建匿名函数

1、ambda 只是一个表达式，函数体比 def 简单很多。

2、lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。

3、lambda 函数拥有自己的命名空间，且不能访问自己参数列表之外或全局命名空间里的参数。

4、虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

# 可写函数说明

sum = lambda arg1, arg2: arg1 + arg2

# 调用sum函数

print ("相加后的值为 : ", sum( 10, 20 ))

print ("相加后的值为 : ", sum( 20, 20 ))

*---示例------------------------*

*def* calc(*n*):  
 *return n* \* *n*print(calc(10))  
# 上面的函数换成匿名函数的写法  
calc = *lambda* n : n \* n  
print(calc(10))  
#匿名函数只要和其他函数搭配使用的，如下  
res = map(*lambda* x:x\*2,[1,5,7,4,8])  
*for* i *in* res:  
 print(i)

-----------输出结果--------------

2

10

14

8

16

### 4.8、函数式编程

函数是python内建支持的一种封装，我们通过把大段代码拆成函数，通过一层一层的函数调用，就可以把复杂任务分解成简单的任务，这种分解可以称之为**面向过程的程序设计**。函数就是面向过程的程序设计的基本单元。

函数式编程中的函数这个术语不是指计算机中的函数（实际上是Subroutine），而是指数学中的函数，即自变量的映射。也就是说一个函数的值仅决定于函数参数的值，不依赖其他状态。比如sqrt(x)函数计算x的平方根，只要x不变，不论什么时候调用，调用几次，值都是不变的。

通俗来讲，函数式编程就是用编程语言去实现数学函数，这种函数内对象是永恒不变的，要么参数是函数，要么返回值是函数，没有for和while循环，所有的循环都是由递归去实现，无变量的赋值（即不用变了去保存状态），无赋值即不改变

Python对函数式编程提供部分支持。由于Python允许使用变量，因此，Python不是纯函数式编程语言。

**定义：**函数式编程 是一种["编程范式"](http://en.wikipedia.org/wiki/Programming_paradigm)（programming paradigm），也就是如何编写程序的方法论。

主要思想是把运算过程尽量写成一系列嵌套的函数调用。

*def* cal(*x*):  
 *return* 2\**x*+1  
print(cal(4))

### 4.9、高阶函数

变量可以指向函数，函数的参数能接收变量，那么一个函数就可以接收另一个函数作为参数，这种函数我们成为高阶函数。如下示例

*def* add(*x*, *y*, *f*):  
 *return f*(*x*) + *f*(*y*)  
res = add(3,6,abs)  
print(res)

-------------

*def* f(*n*):  
 *return n*\**n  
def* foo(*a*,*b*,*func*):  
 aet =*func*(*a*)+func(*b*)  
 *return* aet  
print(foo(3,2,f))

### 4.10、内置函数

参考：https://docs.python.org/3/library/functions.html?highlight=built#ascii

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Built-in Functions** | | | | |
| [abs()](https://docs.python.org/3/library/functions.html?highlight=built#abs) | [delattr()](https://docs.python.org/3/library/functions.html?highlight=built#delattr) | [hash()](https://docs.python.org/3/library/functions.html?highlight=built#hash) | [memoryview()](https://docs.python.org/3/library/functions.html?highlight=built#func-memoryview) | [set()](https://docs.python.org/3/library/functions.html?highlight=built#func-set) |
| [all()](https://docs.python.org/3/library/functions.html?highlight=built#all) | [dict()](https://docs.python.org/3/library/functions.html?highlight=built#func-dict) | [help()](https://docs.python.org/3/library/functions.html?highlight=built#help) | [min()](https://docs.python.org/3/library/functions.html?highlight=built#min) | [setattr()](https://docs.python.org/3/library/functions.html?highlight=built#setattr) |
| [any()](https://docs.python.org/3/library/functions.html?highlight=built#any) | [dir()](https://docs.python.org/3/library/functions.html?highlight=built#dir) | [hex()](https://docs.python.org/3/library/functions.html?highlight=built#hex) | [next()](https://docs.python.org/3/library/functions.html?highlight=built#next) | [slice()](https://docs.python.org/3/library/functions.html?highlight=built#slice) |
| [ascii()](https://docs.python.org/3/library/functions.html?highlight=built#ascii) | [divmod()](https://docs.python.org/3/library/functions.html?highlight=built#divmod) | [id()](https://docs.python.org/3/library/functions.html?highlight=built#id) | [object()](https://docs.python.org/3/library/functions.html?highlight=built#object) | [sorted()](https://docs.python.org/3/library/functions.html?highlight=built#sorted) |
| [bin()](https://docs.python.org/3/library/functions.html?highlight=built#bin) | [enumerate()](https://docs.python.org/3/library/functions.html?highlight=built#enumerate) | [input()](https://docs.python.org/3/library/functions.html?highlight=built#input) | [oct()](https://docs.python.org/3/library/functions.html?highlight=built#oct) | [staticmethod()](https://docs.python.org/3/library/functions.html?highlight=built#staticmethod) |
| [bool()](https://docs.python.org/3/library/functions.html?highlight=built#bool) | [eval()](https://docs.python.org/3/library/functions.html?highlight=built#eval) | [int()](https://docs.python.org/3/library/functions.html?highlight=built#int) | [open()](https://docs.python.org/3/library/functions.html?highlight=built#open) | [str()](https://docs.python.org/3/library/functions.html?highlight=built#func-str) |
| [breakpoint()](https://docs.python.org/3/library/functions.html?highlight=built#breakpoint) | [exec()](https://docs.python.org/3/library/functions.html?highlight=built#exec) | [isinstance()](https://docs.python.org/3/library/functions.html?highlight=built#isinstance) | [ord()](https://docs.python.org/3/library/functions.html?highlight=built#ord) | [sum()](https://docs.python.org/3/library/functions.html?highlight=built#sum) |
| [bytearray()](https://docs.python.org/3/library/functions.html?highlight=built#func-bytearray) | [filter()](https://docs.python.org/3/library/functions.html?highlight=built#filter) | [issubclass()](https://docs.python.org/3/library/functions.html?highlight=built#issubclass) | [pow()](https://docs.python.org/3/library/functions.html?highlight=built#pow) | [super()](https://docs.python.org/3/library/functions.html?highlight=built#super) |
| [bytes()](https://docs.python.org/3/library/functions.html?highlight=built#func-bytes) | [float()](https://docs.python.org/3/library/functions.html?highlight=built#float) | [iter()](https://docs.python.org/3/library/functions.html?highlight=built#iter) | [print()](https://docs.python.org/3/library/functions.html?highlight=built#print) | [tuple()](https://docs.python.org/3/library/functions.html?highlight=built#func-tuple) |
| [callable()](https://docs.python.org/3/library/functions.html?highlight=built#callable) | [format()](https://docs.python.org/3/library/functions.html?highlight=built#format) | [len()](https://docs.python.org/3/library/functions.html?highlight=built#len) | [property()](https://docs.python.org/3/library/functions.html?highlight=built#property) | [type()](https://docs.python.org/3/library/functions.html?highlight=built#type) |
| [chr()](https://docs.python.org/3/library/functions.html?highlight=built#chr) | [frozenset()](https://docs.python.org/3/library/functions.html?highlight=built#func-frozenset) | [list()](https://docs.python.org/3/library/functions.html?highlight=built#func-list) | [range()](https://docs.python.org/3/library/functions.html?highlight=built#func-range) | [vars()](https://docs.python.org/3/library/functions.html?highlight=built#vars) |
| [classmethod()](https://docs.python.org/3/library/functions.html?highlight=built#classmethod) | [getattr()](https://docs.python.org/3/library/functions.html?highlight=built#getattr) | [locals()](https://docs.python.org/3/library/functions.html?highlight=built#locals) | [repr()](https://docs.python.org/3/library/functions.html?highlight=built#repr) | [zip()](https://docs.python.org/3/library/functions.html?highlight=built#zip) |
| [compile()](https://docs.python.org/3/library/functions.html?highlight=built#compile) | [globals()](https://docs.python.org/3/library/functions.html?highlight=built#globals) | [map()](https://docs.python.org/3/library/functions.html?highlight=built#map) | [reversed()](https://docs.python.org/3/library/functions.html?highlight=built#reversed) | [\_\_import\_\_()](https://docs.python.org/3/library/functions.html?highlight=built#__import__) |
| [complex()](https://docs.python.org/3/library/functions.html?highlight=built#complex) | [hasattr()](https://docs.python.org/3/library/functions.html?highlight=built#hasattr) | [max()](https://docs.python.org/3/library/functions.html?highlight=built#max) | [round()](https://docs.python.org/3/library/functions.html?highlight=built#round) |  |

## 5、迭代器与装饰器

参考：http://www.cnblogs.com/alex3714/articles/5765046.html

http://www.cnblogs.com/alex3714/articles/5169958.html

### 5.1、列表生成式

通过列表生成式，我们可以直接创建一个列表，但是受到内存限制，列表容量肯定有限，而且创建比较大元素的列表，就会占用很大的存储空间，如果我们仅仅需要访问前面几个元素，后面大多数元素占用的空间就浪费掉了。所以列表的元素按照某种方法推算出来，我们就可以在循环的过程中推算出后续的元素了，从而节省大量空间。这种方式在python中称为 生成器（generator）

创建生成器，第一种方法是把列表生成式的[]改成()

c = [i+1 *for* i *in* range(10)] #创建的是列表  
print(c)

#打印结果 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

c = (i+1 *for* i *in* range(10)) #创建的是generator  
print(next(c)) #通过next()函数获得返回值  
print(next(c))

#打印结果

1

2

generator保存的是算法，每次调用用next()，就可以计算下个元素的值，直到计算最后一个元素，就会报错误，所以正确的使用方法是for循环，我们创建一个generator后，基本上不会使用next()，而是用for循环来迭代它。

c = (i+1 *for* i *in* range(10))

*for* i *in* c:  
 print(i)

generator非常强大，如果算法比较复杂，用类似列表的for循环无法实现的时候，还可以调用函数来实现。著名的斐波拉契数列（Fibonacci），除第一个和第二个数外，任意一个数都可由前两个数相加得到，所以用列表写不出来，需要用到函数。

*def* fib(*max*):  
 n = 0 #多个变量赋值可以写在一起 n,a,b = 0,0,1  
 a = 0  
 b = 1  
 *while* n <*max*:  
 t = (b,a+b)   
 a=t[0]  
 b=t[1] #上面这个赋值可以简写为 a，b = b，a+b

print(b)  
 n += 1  
fib(10)

输入结果

1

2

3

5

8

13

21

34

55

89

如果把上面的函数变成generator，还需要把print(b) 变成yield b

*def* fib(*max*):  
 n,a,b = 0,0,1  
 *while* n <*max*:  
 a,b = (b,a+b)  
 # print(b)  
 *yield* b  
 n += 1

return ‘done’  
f = fib(10)  
print(next(f)) #建议使用 for循环打印  
*while True*: # while循环的方法  
 *try*:  
 x = next(f)  
 print("f",x)  
 *except* StopIteration *as* e:  
 print(e.value)  
 *break*

还可通过yield实现在单线程的情况下实现并发运算的效果

*import* time  
*def* consumer(*name*):  
 print("%s 准备吃包子啦!" %*name*)  
 *while True*:  
 baozi = *yield* print("包子[%s]来了,被[%s]吃了!" %(baozi,*name*))  
*def* producer(name):  
 c = consumer('A')  
 c.\_\_next\_\_()  
 print("老子开始准备做包子啦!")  
 *for* i *in* range(10):  
 time.sleep(1)  
 print("做了2个包子!")  
 c.send(i)  
producer("alex")

### 5.2、迭代器

可以直接用于for循环的一种是集合数据类型，如列表，元组，字典，集合，字符串等，一种是生成器和包含带yield的函数。这些可以直接用于for循环的对象统称为可迭代对象（Iterable）。可以使用isinstance()判断一个对象是否是Iterable对象。

可以被next()函数调用并不断返回下一个值的对象称为迭代器（iterator）

生成器都是迭代器(Iterator对象，但列表/字典/字符串等虽然是可迭代对象(Iterable却不是迭代器(Iterator)，把列表，字典，字符串等可迭代对象（Iterable）变成迭代器可以使用iter()函数。

it = iter([1, 2, 3, 4, 5])  
# 循环:  
*while True*:  
 *try*:  
 # 获得下一个值:  
 x = next(it)  
 print(x)  
 *except* StopIteration:  
 # 遇到StopIteration就退出循环  
 *break*

### 5.3、装饰器

在不修改原来的代码情况下，添加新的功能，保证和原来的调用方式一样，就是装饰器

比如有个需求，给dd()和foo()函数计算运行的时间，在不修改原来的情况下，添加个tim()函数。

装饰器函数：

*import* time  
*def* tim(*ss*):  
 *def* inn():  
 start = time.time()  
 ss()  
 end = time.time()  
 print("spend %s" %(end-start))  
 *return* inn  
@tim #相当于 foo = tim(foo) 这个就是装饰器  
*def* foo():  
 print("foo")  
 time.sleep(2)  
@tim  
*def* dd():  
 print("dd")  
 time.sleep(2)  
dd()  
foo()

带参数的装饰器

user\_status = *False* #用户登录了就把这个改成True  
*def* login(*auth\_type*): #把要执行的模块从这里传进来  
 *def* auth(*func*):  
 *def* inner(*\**args,*\*\**kwargs):#再定义一层函数  
 *if* auth\_type == "qq":  
 \_username = "alex" #假装这是DB里存的用户信息  
 \_password = "abc!23" #假装这是DB里存的用户信息  
 *global* user\_status  
 *if* user\_status == *False*:  
 username = input("user:")  
 password = input("pasword:")  
 *if* username == \_username *and* password == \_password:  
 print("welcome login....")  
 user\_status = *True  
 else*:  
 print("wrong username or password!")  
 *if* user\_status == *True*:  
 *return* func(\**args*,\*\**kwargs*) # 看这里看这里，只要验证通过了，就调用相应功能  
 *else*:  
 print("only support qq ")  
 *return* inner #用户调用login时，只会返回inner的内存地址，下次再调用时加上()才会执行inner函数  
 *return* auth  
#-----------------------------------------------  
*def* home():  
 print("---首页----")  
@login('qq')  
*def* america():  
 #login() #执行前加上验证  
 print("----欧美专区----")  
*def* japan():  
 print("----日韩专区----")  
@login('weibo')  
*def* henan(style):  
 print("----河南专区----")  
#----------------------------------  
home()  
# america = login(america) #你在这里相当于把america这个函数替换了  
#henan = login(henan)  
# #那用户调用时依然写  
america()  
# henan("3p")

### 5.4、数据序列化

## 6、常用模块学习

参考：http://www.cnblogs.com/alex3714/articles/5161349.html

模块就是用一砣代码实现某个功能的代码集合。类似于函数式编程和面向过程编程，函数式编程则完成一个功能，其他代码用来调用即可，提供了代码的重要性和代码间的耦合。对应一个复杂的功能，可能需要多个函数才能完成（函数又可以在不同的.py文件中）n个文件组成的代码集合就称为模块。

模块分为三种：自定义模块，内置标准模块，开源模块

**自定义模块和开源模块的使用参考：http://www.cnblogs.com/wupeiqi/articles/4963027.html**

### 6.1、import语句

想使用python源文件，只需要在另一个源文件里执行import语句。当解释器遇到import语句，如果模块在,当前的搜索路径就会被导入。

搜索路径是一个解释器会先进行搜索的所有目录列表，如果想导入模块support，需要把吧命令放在脚本的顶端。

#### 6.1.1、time

时间模块可以通过help(time) 查看帮助

time.sleep(3) #停止多长时间

# print(time.gmtime()) #结构化时间 是按照英国时间

print(time.localtime()) #北京的时间

print(time.strftime("%Y-%m-%d %H:%M:%S")) #格式化时间，显示当前的时间 \*常用

print(time.strptime("2018-08-27 00:00:09","%Y-%m-%d %H:%M:%S")) #把格式化时间转换成结构化时间

print(time.ctime()) #打印当前时间，括号加上秒数从1970年开始计算时间

print(time.mktime(time.localtime())) 把结构化时间计算时间戳

**datetime**

print(datetime.datetime.now()) #显示当前格式化时间

#### 6.1.2、random

随机数模块

print(random.randint(1,8)) #生成1到8的 随机数，包含8

print(random.randrange(1,8)) #随机生成1到8的数，不包含8

print(random.choice("hello")) #随机生成填写的数，可以填写列表

print(random.sample([1,2,3,4,5],3)) #从一个列表随机选择3个数

**随机生成5位数字的验证码**

*def* v\_code():  
 code = ""  
 *for* i *in* range(5):  
 add\_num = random.randrange(10)  
 code+=str(add\_num)  
 print(code)  
v\_code()

-------字母和数字混合的随机验证码---------

*def* v\_code():  
 code = ""  
 *for* i *in* range(5):  
 add=random.choice([random.randrange(10),chr(random.randrange(65,91))])  
 code+=str(add)  
 print(code)

#### 6.1.3、os

os.path.join()函数 传递文件路径，更改 os.path介绍参考：https://docs.python.org/3/library/os.path.html

os.getcwd()函数 获取当前工作路径  
os.chdir()函数 改变当前工作路径

os.makedirs()函数 创建新文件夹（目录）

调用 os.path.abspath(path)将返回参数的绝对路径的字符串。这是将相对路径转换为绝对路径的简便方法

调用 os.path.isabs(path)，如果参数是一个绝对路径，就返回 True，如果参数是一个相对路径，就返回 False。

调用 os.path.relpath(path, start)将返回从 start 路径到 path 的相对路径的字符串。如果没有提供 start，就使用当前工作目录作为开始路径。

调用 os.path.getsize(path)将返回 path 参数中文件的字节数。

调用 os.listdir(path)将返回文件名字符串的列表，包含 path 参数中的每个文件（请注意，这个函数在 os 模块中，而不是 os.path）

如果 path 参数所指的文件或文件夹存在， 调用 os.path.exists(path)将返回 True，否则返回 False。

如果 path 参数存在，并且是一个文件， 调用 os.path.isfile(path)将返回 True， 否则返回 False。

如果 path 参数存在， 并且是一个文件夹， 调用 os.path.isdir(path)将返回 True，否则返回 False。

利用 os.path.exists()函数，检测盘符是否存在，返回 True，否则返回 False

#### 6.1.4、pyperclip

模块拷贝粘贴字符串pyperclip 模块有 copy()和 paste()函数， 可以向计算机的剪贴板发送文本， 或从它接收文本。将程序的输出发送到剪贴板， 使它很容易粘贴到邮件、文字处理程序或其他软件中。 pyperclip 模块不是 Python 自带的。要安装它， 请遵从附录 A 中安装第三方模块的指南

#### 6.1.5、re(正则表达式)

正则表达式本身是一种小型的、高度专业化的编程语言，而在python中，通过内嵌集成re模块，程序媛们可以直接调用来实现正则匹配。正则表达式模式被编译成一系列的字节码，然后由用C编写的匹配引擎执行。

参考：http://www.cnblogs.com/tina-python/p/5508402.html

向 re.compile()传入一个字符串值，表示正则表达式，它将返回一个 Regex 模式  
对象（或者就简称为 Regex 对象）。

6.1.6、shelve模块

利用 shelve 模块， 你可以将 Python 程序中的变量保存到二进制的 shelf 文件中。这样， 程序就可以从硬盘中恢复变量的数据。 shelve 模块让你在程序中添加“保存”和“打开” 功能。例如， 如果运行一个程序，并输入了一些配置设置，就可以将这些设置保存到一个 shelf 文件，然后让程序下一次运行时加载它们。

#### 6.1.6、shutil模块

shutil（或称为 shell 工具）模块中包含一些函数，让你在 Python 程序中复制、移动、改名和删除文件。要使用 shutil 的函数，首先需要 import shutil

shutil.copy(source, destination)，将路径 source 处的文件复制到路径 destination处的文件夹（source 和 destination 都是字符串）。

shutil.copy()将复制一个文件， shutil.copytree()将复制整个文件夹，以及它包含的文件夹和文件

### 6.2、from ... import语句

python的from语句让你从模块中导入一个指定的部分到当前命名空间中

import 语句的另一种形式包括 from 关键字，之后是模块名称， import 关键字和  
一个星号， 例如 from random import \*。  
使用这种形式的 import 语句，调用 random模块中的函数时不需要 random.前缀。  
但是， 使用完整的名称会让代码更可读， 所以最好是使用普通形式的 import 语句

6.3、包

## 7、面向对象编程

参考：http://www.cnblogs.com/alex3714/articles/5188179.html

http://www.cnblogs.com/alex3714/articles/5213184.html

## 8、Socket编程

参考：http://www.cnblogs.com/alex3714/articles/5227251.html

## 9、多线程与进程

参考：http://www.cnblogs.com/alex3714/articles/5230609.html

## 10、异步IO、缓存、队列

参考：http://www.cnblogs.com/alex3714/articles/5248247.html

## 11、python操作数据库

参考：http://www.cnblogs.com/alex3714/articles/5950372.html

http://www.cnblogs.com/alex3714/articles/5978329.html

## 12、Django

参考：http://www.cnblogs.com/alex3714/articles/5457672.html

http://www.cnblogs.com/alex3714/articles/7131523.html

参考： http://www.cnblogs.com/alex3714/articles/5512568.html

http://www.cnblogs.com/alex3714/articles/5535652.html

## 13、Flask

## 14、实现登陆随机验证码

http://www.cnblogs.com/alex3714/articles/6662365.html

## 15、常规设计模式

http://www.cnblogs.com/alex3714/articles/5760582.html

http://www.cnblogs.com/alex3714/articles/6661911.html

## 16、常用算法

http://www.cnblogs.com/alex3714/articles/5474411.html

# 第二章 强化练习

## 1、实验练习题



### 1、编写登陆接口

要求：输入用户名和密码，认证成功后显示欢迎信息，输错三次后就退出



### 2、循环输入数字（1 2 3 4 5 6 8 9 10）

for i in range(1,10): #第一种for循环方式  
 if i == 7:  
 continue #表示跳过本次循环，继续下个循环  
 else :  
 print(i)  
--------------------------------------------------------------------------------------  
count = 1 #第二种while循环  
*w*hile count <10:  
 if count == 7:  
 pass #占位指令，写上他不会报错  
 else:  
 print(count)  
 count +=1

### 3、求1至100的所有数的和

print(sum(range(1,101))) #方法1

-------------------------------------------------------------------------

c = 1  
s = 0  
*while* c < 101: #方法2  
 s = s +c  
 c = c + 1  
 # s = c + s  
 print(s)  
-----------------------------------------------------------------------  
a = 0  
*for* i *in* range(1,101): #方法3  
 a = a +i  
 print(a)

### 4、输出1至100内的所有奇数或者偶数

*for* i *in* range(1,101):   
 a = i % 2  
 *if* a == 1: #条件等于1的都是奇数，等于0的都是偶数  
 print(i)

### 5、求1-2+3-4+5 ... 99的所有数的和

s = 0

n = 0  
*for* i *in* range(1,101):  
 a = i % 2  
 *if* a == 1:  
 s = s + i  
 *elif* a == 0:  
 n = n + i  
b = s - n  
print(b)

-------------------------------------------------------------------------

n = 1  
s = 0  
*while* n < 101:  
 te= n % 2  
 *if* te == 0:  
 s = s - n  
 *else*:  
 s = s + n  
 n = n + 1  
print(s)

### 6、比较3个数字的大小

num1 = input("数字1：")

num2 = input("数字2：")  
num3 = input("数字3：")  
  
*if* num1 > num2:  
 num4 = num1  
 *if* num4 > num3:  
 print(num4)  
 *else*:  
 print(num3)  
*else*:  
 num4 = num2  
 *if* num4 > num3:  
 print(num4)  
 *else*:  
 print(num3)

### 6、程序购物车

要求：启动程序，提示存入金额，然后打印列表，允许用户根据商品编号购买商品，选择商品后，检测余额是否够，够就直接扣款，不够就提醒，可以随时退出，退出时，打印已购买的商品和余额。

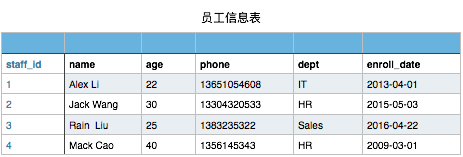


### 7、三级菜单

要求：打印省 市 县三级菜单，可返回上一级，随时可退出程序。



### 8、函数练习实现增删改查操作



当然此表你在文件存储时可以这样表示

1,Alex Li,22,13651054608,IT,2013-04-01

现需要对这个员工信息文件，实现增删改查操作

1、可进行模糊查询，语法至少支持下面3种:

　　select name,age from staff\_table where age > 22

　　select  \* from staff\_table where dept = "IT"

     select  \* from staff\_table where enroll\_date like "2013"

查到的信息，打印后，最后面还要显示查到的条数

2、可创建新员工纪录，以phone做唯一键，staff\_id需自增

3、可删除指定员工信息纪录，输入员工id，即可删除

4、可修改员工信息，语法如下:

　　UPDATE staff\_table SET dept="Market" WHERE where dept = "IT"

**注意：以上需求，要充分使用函数，请尽你的最大限度来减少重复代码！**

****

### 9、把列表种的每个值加1

列表: a=[0,1,2,3,4,5,6,7,8,9]

方法1：创建一个空列表b，循环把a列表的值加1，然后打印出来就行了

b = []   
*for* i *in* a:  
 b.append(i+1)  
print(b)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] #打印结果

方法2：列表生成器方式创建

c = [i+1 *for* i *in* range(10)]  
print(c)

### 10、模拟实现一个ATM + 购物商城程序

1、额度 15000或自定义

2、实现购物商城，买东西加入 购物车，调用信用卡接口结账

3、可以提现，手续费5%

4、每月22号出账单，每月10号为还款日，过期未还，按欠款总额 万分之5 每日计息

5、支持多账户登录

6、支持账户间转账

7、记录每月日常消费流水

8、提供还款接口

9、ATM记录操作日志

10、提供管理接口，包括添加账户、用户额度，冻结账户等。。。

11、用户认证用装饰器

参考：https://github.com/triaquae/py3\_training/tree/master/atm

https://www.processon.com/view/link/589eb841e4b0999184934329

### 11、开发一个简单的python计算器

1、实现加减乘除及拓号优先级解析

2、用户输入 1 - 2 \* ( (60-30 +(-40/5) \* (9-2\*5/3 + 7 /3\*99/4\*2998 +10 \* 568/14 )) - (-4\*3)/ (16-3\*2) )等类似公式后，必须自己解析里面的(),+,-,\*,/符号和公式(不能调用eval等类似功能偷懒实现)，运算后得出结果，结果必须与真实的计算器所得出的结果一致

### 12、Collatz序列

编写一个名为 collatz()的函数，它有一个名为 number 的参数。如果参数是偶数，那么 collatz()就打印出 number // 2， 并返回该值。如果 number 是奇数， collatz()就打印并返回 3 \* number + 1。  
 然后编写一个程序， 让用户输入一个整数， 并不断对这个数调用 collatz()， 直到函数返回值１（令人惊奇的是， 这个序列对于任何整数都有效， 利用这个序列，你迟早会得到 1！ 既使数学家也不能确定为什么。 你的程序在研究所谓的“Collatz序列”，它有时候被称为“最简单的、 不可能的数学问题”）。

*import* sys

*def* collatz(*number*):  
 print(*number*)  
 *if number* == 1:  
 sys.exit()  
 *elif number* % 2 == 1:  
 t = *number* \* 3 + 1  
 collatz(t)  
 *else*:  
 t = *number* // 2  
 collatz(t)  
# if \_\_name\_\_ == '\_\_main\_\_':  
n = input('Enter number: ')  
*try*:  
 n = int(n)  
 collatz(n)  
*except* ValueError *as* verror:  
 print('ValueError: You need input digital.')

### 13、检测输入的是不是整数

*---------方法1---------------*

*try*:  
 num = int(input(">"))  
 print(num)  
*except* ValueError:  
 print("请输入数字")  
--------方法2-----------------------  
fan = input(">")  
*if* fan.isdigit():  
 fan=int(fan)  
 print(fan)  
*else*:  
 print("请输入数字")

### 14、逗号代码

假定有下面这样的列表：spam = ['apples', 'bananas', 'tofu', 'cats'] 编写一个函数，它以一个列表值作为参数，返回结果是一个字符串。该字符串包含列表的所有值，值之间以逗号和空格分隔，并在最后一个表项之前插入 and。例如，将前面的 spam 列表传递给函数，将返回'apples, bananas, tofu, and cats'。

利用.join()方法操作

spa = ['app','ban','tofu','cats']  
*def* fan(*g*):  
 *return* ','.join(*g*[:-1] + ['and ' + *g*[-1]])  
print(fan(spa))

### 15、字符图网格

假定有一个列表的列表， 内层列表的每个值都是包含一个字符的字符串， 像这样：  
grid = [['.', '.', '.', '.', '.', '.'],

['.', 'O', 'O', '.', '.', '.'],

['O', 'O', 'O', 'O', '.', '.'],

['O', 'O', 'O', 'O', 'O', '.'],

['.', 'O', 'O', 'O', 'O', 'O'],

['O', 'O', 'O', 'O', 'O', '.'],  
['O', 'O', 'O', 'O', '.', '.'],  
['.', 'O', 'O', '.', '.', '.'],  
['.', '.', '.', '.', '.', '.']]  
你可以认为 grid[x][y]是一幅“图” 在 x、 y 坐标处的字符， 该图由文本字符组成。 原点(0, 0)在左上角， 向右 x 坐标增加， 向下 y 坐标增加。复制前面的网格值， 编写代码用它打印出图像。  
..OO.OO..  
.OOOOOOO.  
.OOOOOOO.  
..OOOOO..  
...OOO...  
....O....

grid = [['.', '.', '.', '.', '.', '.'],  
 ['.', 'O', 'O', '.', '.', '.'],  
 ['O', 'O', 'O', 'O', '.', '.'],  
 ['O', 'O', 'O', 'O', 'O', '.'],  
 ['.', 'O', 'O', 'O', 'O', 'O'],  
 ['O', 'O', 'O', 'O', 'O', '.'],  
 ['O', 'O', 'O', 'O', '.', '.'],  
 ['.', 'O', 'O', '.', '.', '.'],  
 ['.', '.', '.', '.', '.', '.']]  
*for* i *in* range(len(grid[0])):  
 *for* j *in* range(len(grid)):  
 print(grid[j][i],end='')  
 print()

### 16、好玩游戏的物品清单

你在创建一个好玩的视频游戏。用于对玩家物品清单建模的数据结构是一个字典。其中键是字符串，描述清单中的物品，值是一个整型值，说明玩家有多少该物品。例如，字典值{'rope':1,'torch':6,'gold coin':42, 'dagger':1,'arrow':12}意味着玩家有一条绳索、6个火把、42枚金币等。写一个名为displayInventory()的函数，它接受任何可能的物品清单，并显示如下：

Inventory:

12 arrow

42 gold coin

1 rope

6 torch

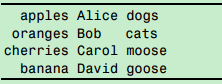
1 dagger

Total number of items: 62

sap = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}  
  
*def* diplay():  
 print("Inventory")  
 num = 0  
 *for* i, y *in* sap.items():  
 print(y, i)  
 num +=y  
 print('Total number of items: %s' % num)  
diplay()

### 17、表格打印

接受字符串的列表的列表，将它显示在组织良好的表格中， 每列右对齐。假定所有内层列表都包含同样数目的字符串。例如，该值可能看起来像这样：你的 printTable()函数将打印出：





## 2、理论练习题

### 2.1、关于函数的习题

1、当函数调用返回时，局部作用域中的变量发生了什么

  答：局部变量的作用域只在函数内部，在函数返回后，局部变量的内存已经释放了。因此，如果函数返回的是局部变量的值，不涉及地址，程序不会出错。但是如果返回的是局部变量的地址(指针)的话，程序运行后会出错。

2、什么是返回值？返回值可以作为表达式的一部分吗？

答：返回值时函数调用求值的结果。像所有值一样，返回值可以作为表达式的一部分。

3、如果函数没有返回语句，对它调用的返回值是什么？

答：如果函数没有return语句，它的返回值就是None。

4、如何强制函数中的一个变量指的是全局变量？

答：global语句强制函数中的一个变量引用该全局变量。

5、None的数据类型是什么？

答: None的数据类型是None Type。

6、import areallyourpetsnamederic语句做了什么？

答：import语句导入了areallyourpetsnamederic模块(顺便说一句，这不是一个真正的Python模块)。

7、如果在名为spam的模块中，有一个名为bacon()的函数，在引入span后，如何调用它？

答：该函数可以通过spam.bacon()调用。

8、如何防止程序在遇到错误时崩溃？

答：将可能导致错误的代码行放在一个try子句中。

9、try子句中发生了什么？except子句中发生了什么？

答：可能导致错误的代码放在try子句中。发生错误时，要执行的代码放在except子句中

### 2.2、关于列表的习题

1、如何将'hello'赋给列表的第三个值，而列表保存在名为 spam 的变量中？（假定变量包含[2, 4, 6, 8, 10]）

答：spam[2] = ‘hello’

对接下来的 3 个问题，假定 spam 包含列表['a', 'b', 'c', 'd']。  
2、spam[int('3' \* 2) / 11]求值为多少？

答：'d' (注意'3'\*2是字符串'33'，它被传入int(),就变成了33，然后再除以11。这最终求值为3.在使用值的地方，都可以使用表达式。)

3、spam[-1]求值为多少？

答：d

4、spam[:2]求值为多少？

答：a和b （表示 0：2，2位置之前的数）

对接下来的 3 个问题。假定 bacon 包含列表[3.14, 'cat', 11, 'cat', True]。

5、bacon.index('cat')求值为多少？

答：1 （index是查找下标）

6、bacon.append(99)让 bacon 中的列表值变成什么样？

答：多了一个99的数字

7、bacon.remove('cat')让 bacon 中的列表时变成什么样？

答：下标1的cat 被删除了

8、列表连接和复制的操作符是什么？

答：连接的操作符 是 + ，复制是\* (乘号)

9、append()和 insert()列表方法之间的区别是什么？

答：append是在列表的最后面加参数，insert可以插入任何位置

10、从列表中删除值有哪两种方法？

答：知道下标可以用del，知道具体值可以用remove()

11、请说出列表值和字符串的几点相似之处。

答：列表和字符串都可以传入len()，都有下标和切片，用于for循环，连接或复制，并与in和not in操作符一起使用

12、列表和元组之间的区别是什么？

答：列表可以修改，元组不可以修改，列表是用[]创建，元组是用（）创建

13、如果元祖中只有一个整数值 42，如何输入该元组？

答：spam=(42,)  末尾的逗号是必须的，如果没有逗号，用type()查看类型是int

14、列表值得到元组形式？如何从元组值得到列表形式？

答：分别使用tuple()和list()函数。

15、包含列表的变量，实际上并未真正接包含列表。它们包含的是什么？

答：包含了对列表值得引用

16、copy.copy()和 copy.deepcopy()之间的区别是什么？

答：copy.copy()函数将浅拷贝列表，而copy.deepcopy()函数将深拷贝列表。也就是说，只有copy.deepcopy()会复制列表内的所有列表。

### 2.3、关于字典的习题

1、空字典的代码是怎样的？

答：{}

2、一个字典包含键'fow'和值 42， 看起来是怎样的？

答：dict = [“fow”:42]

3、字典和列表的主要区别是什么？

答：字典是{}来创建，列表是[]来创建，字典是无序,通过key访问，列表是有序的

4、如果 spam 是{'bar': 100}， 你试图访问 spam['foo']， 会发生什么？

答：报错

5、如果一个字典保存在 spam 中， 表达式'cat' in spam 和'cat' in spam.keys()之间的区别是什么？

答：是一样的，没有区别，都是检查是否有cat这个key

6、如果一个字典保存在变量中， 表达式'cat' in spam 和'cat' in spam.values()之间的区别是什么？

答：values()是检查是否有一个值

7、下面代码的简洁写法是什么？

if 'color' not in spam:

spam['color'] = 'black'

答：spam.setdefault(‘color’,’black’)

8、什么模块和函数可以用于“漂亮打印”字典值？

答：pprint.pprint()

### 2.4、关于字符串习题

1、什么是转义字符？

答：字符表示字符串中的一些字符，这些字符用别的方式很难在代码中打印出来。

2、转义字符\n 和\t 代表什么？

答：换行和横向制表符

3、如何在字符串中放入一个倒斜杠字符\？

答：\\表示反斜杠

4、字符串"Howl's Moving Castle"是有效字符串。为什么单词中的单引号没有转义， 却没有问题？

答：因为使用双引号引住的字符串

5、如果你不希望在字符串中加入\n， 怎样写一个带有换行的字符串？

答：就直接换行，让它变成多行的字符串

6、下面的表达式求值为什么？

• 'Hello world!'[1] 查询下标1的 = e

• 'Hello world!'[0:5] 查询0到5的 = hello

• 'Hello world!'[:5] 查询0到6的 =hello

• 'Hello world!'[3:] 查询3之后的 = lo world!

7、下面的表达式求值为什么？  
• 'Hello'.upper() 变成大写

• 'Hello'.upper().isupper() 返回True

• 'Hello'.upper().lower() 变成小写

8、下面的表达式求值为什么？  
• 'Remember, remember, the fifth of November.'.split() 以空格分隔成列表

• '-'.join('There can be only one.'.split()) 把列表已 - 拼接成字符串

9、什么字符串方法能用于字符串右对齐、 左对齐和居中？

答：rjust() 左对齐 ljust() 右对齐 center() 居中

10、如何去掉字符串开始或末尾的空白字符？

答 strip() 删除两边空白，rstrip()删除左边 lstrip() 删除右边

### 2.5、正则表达式

1. 创建Regex对象的函数是什么？

答：re.compile()函数返回Regex对象。

2. 在创建Regex对象时，为什么常用原始字符串？

答：使用原始字符串是为了让反斜杠不必转义。

3. search()方法返回什么？

答：search()方法返回Match对象。

4. 通过Match对象，如何得到匹配该模式的实际字符串？

答：group()方法返回匹配文本的字符串。

5. 用 r'(\d\d\d)-(\d\d\d - \d\d\d\d)' 创建的正则表达式中，分组0表示什么？分组1呢？分组2呢？

答：分组0表示整个匹配，分组1包含第一组括号，分组2包含第二组括号。

6. 括号和句点在正则表达式语法中有特殊的含义。如何制定正则表达式匹配真正的括号和句点字符？

答：句号和括号可以用反斜杠转义：\.、\（和\）。

7. findall()方法返回一个字符串的列表，或字符串元组的列表。是什么决定它提供哪种返回？

答：如果正则表达式没有分组，就返回字符串的列表。如果正则表达式有分组，就返回字符串元组的列表。

8. 在正则表达式中，|字符表示什么意思？

答：|表示匹配2组中的"任何一个"。

9. 在正则表达式中，？字符表示什么意思？

答：？字符可以表示"匹配前面分组0次或1次"，或用于表示非贪婪匹配。

10. 在正则表达式中，+和\*字符之间的区别是什么？

答：+匹配1次或多次。\*匹配0次或多次。

11. 在正则表达式中，{3}和{3,5}之间的区别是什么？

答：{3}精确匹配前面分组的3次实例。{3,5}匹配3至5次实例。

12. 在正则表达式中，\d、\w和\s缩写字符类是什么意思？

答：数字、单词和空白字符。

13. 在正则表达式中，\D、\W和\S缩写字符类是什么意思？

答：分别匹配一个字符，它不是数字、单词或空白字符。

14. 如何让正则表达式不区别大小写？

答：将re.I 或者 re.IGNORECASE作为第二个参数传入re.compile(),让匹配不区分大小写。

15. 字符.通常匹配什么？如果 re.DOTALL作为第二个参数传递给re.compile(),它会匹配什么？

答：字符.通常匹配任何字符，换行符除外。

16. .\*和.\*?之间的区别是什么？

答：.\*表示贪婪匹配，.\*？表示非贪婪匹配。

17. 匹配所有数字和小写字母的字符分类语法是什么？

答：[0-9a-z]或[a-z0-9]

18. 如果 numRegex = re.compile(r'\d+')，那么numRegex.sub('X','12 drummers,11pipers,five rings,3hens')返回什么？

答：'X drummers,X pipers,five rings,X hens'

19. 将re.VERBOSE 作为第二个参数传递给re.compile(),让你能做什么？

答：添加空格和注释

20. 如何写一个正则表达式，匹配每3位就有一个逗号的数字？它必须匹配以下数字：

  '42'  '1,234' '6,368,745' 但不会匹配：  '12,34,567'(逗号之间只能有两位数字)  '1234'(缺少逗号)

答：re.compile(r'^\d{1,3}(,{3})\*$')

21. 如何写一个正则表达式，匹配姓名Makamoto的完整姓名？你可以假定名字总是出现在姓前面，是一个大写字母开头的单词。

该正则表达式必须匹配： 'Satoshi Nakamoto' 'Alice Nakamoto' 'RoboCop Nakamoto'

但不匹配：'satoshi Nakamoto' 'Mr.Nakamoto' 'Nakamoto' 'Satoshi nakamoto'

答：re.compile(r'[A-Z][a-z]\*\sNakamoto')

22. 如何编写一个正则表达式匹配一个句子，他的第一个词是Alice、Bob或Carol,第二个词是eats、pets或throws,第三词是apples、cats或baseablls.该句子以句点结束。这个正则表达式不区分大小写。它必须匹配：

'Alice eats apples' 'Bob pets cats' 'Carol throws baseballs' 'Alice throws Apples' 'BOB EATS CATS'

但不匹配： 'RoboCop eats apples' 'ALICE THROWS FOOTABLLS' 'carol eats 7 cats'

答：re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseablls)\.,re.IGNOREACSE'）

### 2.6、文件操作

1．相对路径是相对于什么？

答：从一个目录为起点到你所在的目录

2．绝对路径从什么开始？

答：从根目录为起点到你所在的目录

3． os.getcwd()和 os.chdir()函数做什么事？

答：os.getcwd() 获取当前目录，os.chdir改变当前目录

4． .和..文件夹是什么？

答：点是当前目录 点点是上级目录

5． 在 C:\bacon\eggs\spam.txt 中， 哪一部分是目录名称， 哪一部分是基本名称？

答：C:\bacon\eggs\目录 spam.txt 文件名称

6．可以传递给 open()函数的 3 种“模式”参数是什么？

答： r 只读模式 w 只写模式 a 追加模式

7．如果已有的文件以写模式打开，会发生什么？

答：删除原来的内容 重新写新的内容

8． read()和 readlines()方法之间的区别是什么？

答：read() 读取文件内容，readlines()把每行内容变成列表

9． shelf 值与什么数据结构相似？

答：helf虽然同字典很类似，但略有差别，key只能是字符串类型，不能是数值。

## 3、自动化脚本汇总

### 3.1、密码保管箱

编写一个脚本，保存所有账号和密码用到的时候只需要执行这个脚本，就可以把密码复制出来，方便保存账号密码。

参考：Python编程快速上手.PDF（103）

把账号和密码写到字典里，

### 3.2、操作剪切板的内容

在编辑一个非常大的文本时，希望在每行前面添加星号。用python脚本将这个任务自动化。（从剪贴板中取得文本，在每一行开始处加上星号，然后将这段新的文本粘贴出来）参考（105）

分3个步棸：1、取到复制的文本内容；2、分离文本中的行，并添加星号；3、将处理好的文本粘贴出来。

利用pyperclip模块，这个模块需要安装。方法如下：

*import* pyperclip  
  
text = pyperclip.paste() #1.获取剪切板的内容  
lines = text.split('\n') #利用换行符把它变成一个列表  
*for* i *in* range(len(lines)): #计算这个列表个数，然后循环这个列表  
 lines[i] = "\* "+lines[i] #在每行列表前面加个星号  
text='\n'.join(lines) #将每个列表值以换行符分隔，变成一行  
pyperclip.copy(text) #3.把内容粘贴出来

### 3.3、提取文本种的电话号码或者E-mail地址

假设你有一个无聊的任务， 要在一篇长的网页或文章中， 找出所有电话号码和邮件地址。如果手动翻页， 可能需要查找很长时间。如果有一个程序， 可以在剪贴板的文本中查找电话号码和 E-mail 地址， 那你就只要按一下 Ctrl-A 选择所有文本，按下 Ctrl-C 将它复制到剪贴板，然后运行你的程序。它会用找到的电话号码和 E-mail地址， 替换掉剪贴板中的文本。参考（126）

### 3.4、强口令检测

写一个函数，它使用正则表达式， 确保传入的口令字符串是强口令。 强口令的定义是： 长度不少于 8 个字符， 同时包含大写和小写字符， 至少有一位数字



### 3.5、strip()的正则表达式版本

写一个函数，它接受一个字符串， 做的事情和 strip()字符串方法一样。如果只传入了要去除的字符串， 没有其他参数， 那么就从该字符串首尾去除空白字符。否则， 函数第二个参数指定的字符将从该字符串中去除。

参考：https://www.cnblogs.com/wudongwei/p/8992963.html

### 3.6、随机生成试题

参考（144）



### 3.7、多重剪贴板

参考（148）

然后在编写一个扩展多重剪贴板增加一个 delete <keyword>命令行参数，它将从 shelf 中删除一个关键字。然后添加一个 delete 命令行参数，它将删除所有关键字。



### 3.8、疯狂填词

创建一个疯狂填词（Mad Libs）程序，它将读入文本文件， 并让用户在该文本文件中出现 ADJECTIVE、 NOUN、 ADVERB 或 VERB 等单词的地方， 加上他们自己的文本。例如，一个文本文件可能看起来像这样：

The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN wasunaffected by these events.

程序将找到这些出现的单词， 并提示用户取代它们。

Enter an adjective:

silly

Enter a noun:

chandelier

Enter a verb:

screamed

Enter a noun:

pickup truck

以下的文本文件将被创建：结果应该打印到屏幕上， 并保存为一个新的文本文件

The silly panda walked to the chandelier and then screamed. A nearby pickup truck was unaffected by these events.

### 3.9、正则表达式查找

编写一个程序， 打开文件夹中所有的.txt 文件，查找匹配用户提供的正则表达式的所有行。结果应该打印到屏幕上

# 第三章 自动化运维

# 第四章 爬虫

# 第五章 web 前端

# 第七章 项目实战

## 1、项目实战

http://www.cnblogs.com/alex3714/articles/6567636.html

## 2、网站用户访问质量分析监测

http://www.cnblogs.com/alex3714/articles/5714238.html

## 3、分布式监控系统开发

http://www.cnblogs.com/alex3714/articles/5450798.html

## 4、堡垒机开发

参考：http://www.cnblogs.com/alex3714/articles/5286889.html

http://www.cnblogs.com/alex3714/articles/6100840.html

## 5、开发web聊天

http://www.cnblogs.com/alex3714/articles/5337630.html

## 6、开发BBS论坛

http://www.cnblogs.com/alex3714/articles/5311625.html

## 7、构建微电影视频网站

# 第八章 脚本汇总

# 第九章 问题总结