# Python 骑士

## 第一部：基础语法

By——一个路过的假面骑士

# 第一章. 绪论

## 为何使用python?

我们为什么学习Python？python 相对于其他的语言有哪些优势?Python 语言是一门高级语言，诞生于1990年。从1972年诞生了C语言之后的40年间，先后有600多种语言横空出世，以各自独特的特色演绎于历史舞台子上，但时间是检验真理的唯一标准，大浪淘沙，有些编程语言因其狭窄的领域应用范围而退出了历史舞台，埋没于时代之中，而有些编程语言至今为众人所熟知，为众人所使用！但不可否认的是，这些编程语言为编程界都作出了巨大的贡献！

发明python的仁兄可能到现在都想不到哇，他的伟大发明，成为了一个全栈开发工程师的炒鸡利器语言。python是1980s（20世纪80年代）开始构想的，1989年被Guido van Rossum开发出来，作为ABC编程语言的继承者。Van Rossum是python的主要作者。到现在为止，全球的python开发人员已经创造了新高，python已经俨然跻身成为世界级主流开发语言之一，而且有大热的趋势。

无论你是想玩游戏开发，还是小型应用；无论你是想玩普通软件，还是人工智能，python都会给你提供一个倍而棒的开发平台和各种各样的模块给你随便用。避免了重复造轮子，能够直接套用方法，支持面向对象，能够把java的代码缩减一大半，这样的语言他不香嘛?

## python的优缺点

大凡程序设计语言，必有优缺点之分。咱们可以康康python的优缺点到底都有些啥：

优点：

**1. Python程序简单易懂，初学者入门容易。**

**2. 开发效率高，有强大的第三方库，可以在基础库的基础上再开发，直接用，爽。**

**3. 使用高级语言，编写程序无需考虑底层细节。**

**4. 可移植性。**

**5. 可联动性，部分程序可与C或C++联动（Cython优化）。**

**6. 语言更加灵活多变**

缺点：

**1. 安全性不咋地**

**2. 运行大程序比较慢，不怎么支持多线程**

**3. 对多核CPU的空间资源利用性8太行**

##对比几种语言，体会python的代码风格

咱们要是用不同的语言编写hello world这个差距就可以看出来了吧：
C语言版：

```
#include<stdio.h>
int main()
{
        printf("hello,world.\n");
        return 0;
}
```

c++版

```
#include<iostream>
using namespace std;
int main()
{
        cout<<"hello world"<<endl;
        return 0;
}
```

java版

```
pubilc class Hello{
        public static void main(String[] args){
                System.Out.println("hello,world.");
        }
}
```

而python只要一句：

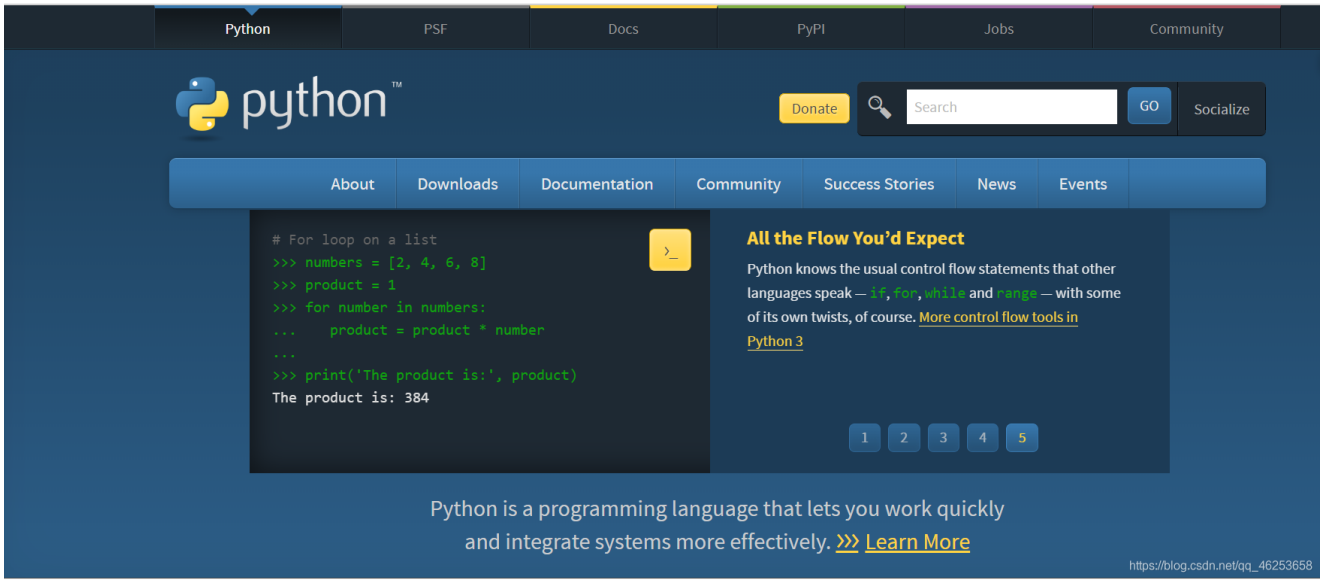
```
print("hello,world")
```

看看人家python，给咱家省了多大的代码量。不仅语句精简，使用灵活，连分号都给省掉了。真是没有对比就没有伤害哇。

# python的安装方法

## 普通IDLE

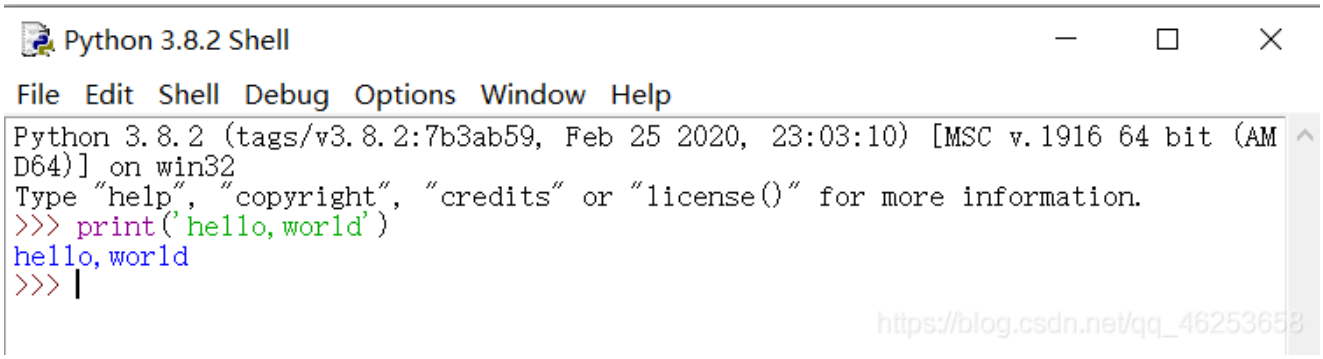
python语言最简单的编译器就是小IDLE。这里我给一个下载的网址链接:www.python.org
进去之后可以看到这样一个页面：




进入download界面，找适合于windows系统的python版本，注意：**python和python3的语法那可是有很大区别的哟，而且现在python2已经没人维护了，大家都在用python3。**

建议3.7.x会比较好一点呢，亲[doge]

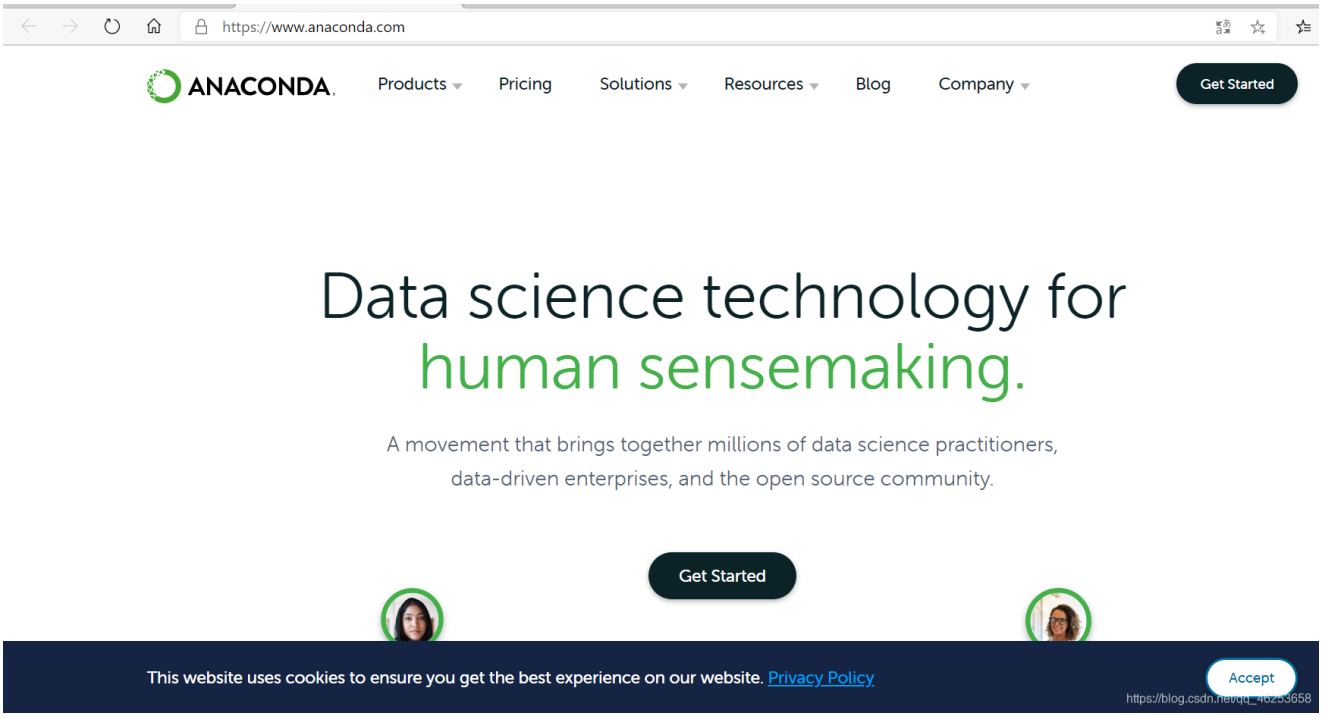
打开以后界面是这个样子的：




可以看到，这是即时编写即时编译运行的，尽管你可能察觉不到，但实际上，它的速率会比较慢。当然，如果我想像C一样编写完整的程序去编译运行可不可以呢？当然是可以的。点击左上角的file,新建一个python-file文件就可以在里面码很多行代码辣。

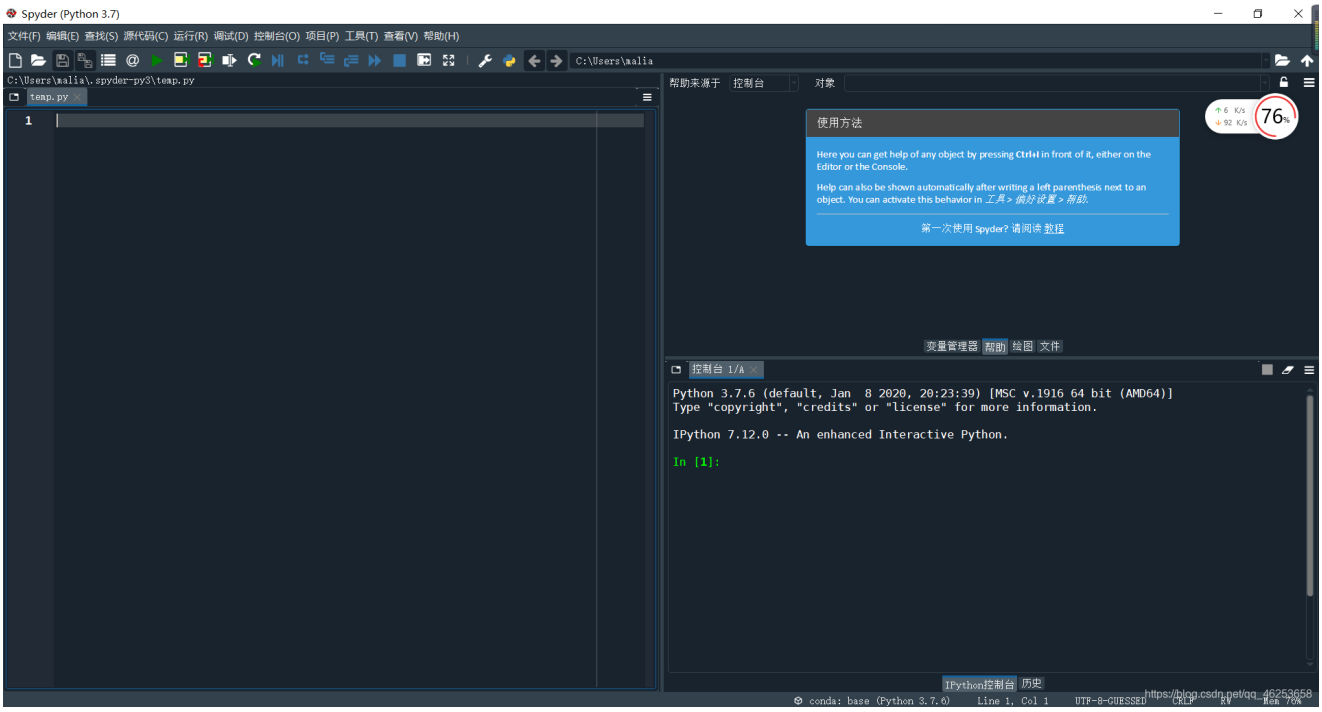
## anaconda版

相比于开发库比较少的IDLE，这里更推荐anaconda，它拥有的python库更多，然后编辑工具也多。有jupyter notebook, spyder, shell等。这里也附上一个链接anaconda




找到适合自己操作系统的版本下过来，然后按照提示一步步安装就可以了。这个没啥子技巧。就是尽量装anaconda就不要再装python3的IDLE了，可能引起冲突。

下好以后可以发现在文件夹下面有很多东西。可以点到spyder里面，那是个不错的编辑器。主要是文件式编程，不同于IDLE的即时交互式编程。而且spyder的界面也比IDLE帅很多//属于男人的浪漫




## 补充几点：

1. 要是IDLE缺什么库，就把命令行窗口cmd打开，输入pip install ***，如果是在anaconda里面，也可以使用conda install **。

2. 如果在安装过程中把python加进了环境变量，也可以在vscode当中编写程序。效果无需多说，自动代码补齐真他娘香。
3. 我准备说啥来着？...忘了，后面想到啥说啥吧哈哈

# 说说我吧

为啥我想写这个玩意呢？一方面是对自己的复习，另一方面也希望能够帮到他人。由于python我纯粹自学，所以可能也有些不够精进的地方。也希望能够在这部《python骑士》当中有新的收获和体会。

那么基础的部分我们将以这样的一个顺序展开：

1. 数据类型，运算符与表达式
2. 输入输出语句，if系列语句
3. 循环语句的使用
4. 列表，元组，字典与集合
5. 异常捕捉与报错
6. 函数式编程
7. 面向对象初步
8. 文件读取与写入
9. 小程序设计的思想

**撒，新的传说开始了**

# 第二章.数据类型，运算符和表达式

## 2.1 数据类型

与其他程序设计语言的数据类型很类似，也有不同。python的基本数据类型大致上可以分成如下几种：

**str, int, float, complex，Bool**

即：字符串，整数，浮点，复数，布尔。

**字符（串）**类编码ASCII码，遵循unicode规则。通常是用utf-8。

如果不太清楚这几个概念的话，推荐看看这篇文章ASCII、Unicode、GBK、UTF-8之间的关系

**整数**类：不同于C/C++对int 32bit/64bit机的限制，在Python3当中，int数据是没有固定大小的。这也就是说，int可以存储超大的数进行运算。

**浮点数**类: 取消了double，只保留float够用了

**复数**类：分实部和虚部，虚数单位用j表示。一般用得不多

**布尔**类：属于逻辑类数据。只有两种取值，True或False。注意在赋值的时候要首字母大写。

type()可以用来查看变量的数据类型。那么，我们在IDLE中测试一下如下的代码：

```
File  Edit  Shell  Debug  Options  Window  Help
>>> a
1
>>> type(a)
<class 'int'>
>>> a*=5
>>> a
5
>>> a/6
0.8333333333333334
>>> #不同于C，它可以自动转换为合适的类型
>>> b=0.5
>>> type(b)
<class 'float'>
>>> b/5
0.1
>>> 3.1415/2.78
1.1300359712230217
>>> #python程序对浮点数的计算可能不会太精确。明明知道答案是4，它可能给一个4.0000
000001。后面可能会出现，见怪不怪了
>>> #类型转换
>>> a=3.1415
>>> int(a)
3
>>> b=5
>>> float(b)
5.0
>>> c=1+2j
>>> type(c)
<class 'complex'>
>>> d=2+5j
>>> c*d
(-8+9j)
>>> complex(a)
(3.1415+0j)
>>> int(c)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    int(c)
TypeError: can't convert complex to int
>>> #说明复数类不能转换为整数。浮点也是不行的
                                                                    Ln: 43  Col: 25
```

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x=True
>>> type(x)
<class 'bool'>
>>> x & False
False
>>> x|True
True
>>> #在python中，&是逻辑与，|是逻辑或
>>> y='c'
>>> type(y)
<class 'str'>
>>> z='har'
>>> type(z)
<class 'str'>
>>> y+z
'char'
```



## 2.2 运算符和表达式

与C/C++类似的，python也会有一些数学运算与逻辑运算，但有些位置与C/C++不同。我这里呢，把python常见的运算符给罗列一下

- ^, |, &,~,>>,<< 按位异或，或，与，非，右移，左移
- and, or, not 逻辑与或非
- is 判断是否为同一对象
- in 判断元素是否在集体数据中
- == 判断是否相等
- !=,>,<,>=,<= 不等，大于小于
- +,-,*,/,%,= 加减乘除，余数，赋值
- +=,-=,*=,/= 与C语言中类似的用法，注意python当中是没有i++或++i的
- //,** 求整除商，求幂

优先级如图

以下表格列出了从最高到最低优先级的所有运算符：

| 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 |
| not or and | 逻辑运算符 |

这里很多同学可能对我们的逻辑运算不太理解，想要理解逻辑运算，除了要有一定的逻辑论基础以外，还要了解我们的二进制。
这里呢我也放个博客上来二进制
当然慕课上面也有很多资源大家自己取就可以了

## 思考题：

计算下列各式的值，然后用python编一下，看看对不对

```
x=10
1. x^5
```

```
2. x|12
3. x&2
4. ~x
5. x<<2
6. x>>1
7. x==10
8. x>=6
9. x==10 and x<=6
10. y=7
11. x==y
12. x is y
13. not x%2==0
14. x*=4
15. x//7
16. x**2
17. x in [7,8,9,10]
```

# 第三章. 输入输出与if系列语句

## 3.1 输入输出以及其他内置函数

python的变量使用之前大家也看到了，不像C/C++要带一个int 啥的来声明。使用起来更加方便。然后输出语句我们也知道是print，可是你真的知道输出语句的语法吗?
说到输出我们就不得不提及一种数据结构就是字符串。字符串是多个字符的集合，在C++/JAVA中我们一般都不把它当作基本数据结构的。Python的print函数输出的可以是字符串，可以是单个数据，也可以是集体数据。不过，当print内部涉及多种类型时就要注意了。

```
>>> x=10
>>> print('jjieguo:'+x)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print('jjieguo:'+x)
TypeError: can only concatenate str (not "int") to str
>>> print('jieguo:',x)
jieguo: 10
>>> print(x)
10
>>> a=[1,2,3]
>>> print(a)
[1, 2, 3]
>>>
```



python的print不像Java里一样可以+，这里直接用逗号隔开就行了
那如果我想以另一种方式，像C语言一样在字符串里面嵌入整数怎么办呢？我们有这样的语句：
C语言中是这样的：

```
printf("hello,it is the %dth soldier",x);
```

python里面变成了这样的：

```
print("hello,it is the %dth soldier"%x)
```

哦对了，字符串用双引号还是单引号都是无所谓的，这个不同于其他语言。不过注意不能用中文引号就可以了。

说完输出我们再谈输入。python的输入不像C里面scanf那么单纯，但也没有java里面new Scanner那么复杂。python中的输入函数input只能输入字符（串）。是个什么情况我们可以去康康：

```
>>> x=input('input the number')
input the number10
>>> x
'10'
>>> type(x)
<class 'str'>
>>>
```

*input 的括号里面可以输入提示信息*

看到了吧，input的这个输入函数，返回的是个字符串。那既然是字符串，就不能与数字随意运算了。那如果我们要输入一个数字怎么办呢？不要慌，我们可以使用强制类型转换（就是“目标类型（待转换数据）”）

```
x=int(input('input the number'))
```

### 其他内置函数举例

内置函数bin(). oct(). hex()可以将整数转为二进制，八进制，十六进制形式
ord()用来返回单个字符的Unicode码，chr()返回Unicode码对应的值
list(). tuple(). dict(). set(). frozenset()可以将其他类型数据转为列表，元组，字典，可变集合，和不可变集合
max(), min(), sum()这三个函数用来计算包含列表，元组等包含有限个元素可迭代对象中所有元素最大值，最小值以及所有元素之和。
sorted()对可迭代对象进行排序
enumerate()函数从来枚举可迭代对象中的元素，返回可迭代的enumerate对象，其中每个元素都是包含索引和值得元组
iter()方法用来返回指定对象的迭代器,next()方法用来返回可迭代对象中的下一个元素，等价于zip等对象的__next__()方法
map(),reduce(),filter()
这几个是常用的函数，也是Python函数式编程的重要体现，Python3版本reduce()并非内置函数，而是放到了标准库functools中。reduce()函数可以将一个接受两个参数的函数以迭代累计方式从左至右依次作用到一个序列或迭代器对象的所有元素上，并且允许指定一个初始值
**具体的使用就以后再说吧**

## 3.2 if语句

接下来我们要讲解的是python当中的流程控制。这是python语言的一个重头戏。结构分顺序结构，选择结构和循环结构。顺序结构就是一条线地把语句串行下来，这个都没什么讲头。我们今天重点学习分支。

首先是最基本的if和if-else。if语句就是说：在满足条件时执行某操作；没有else就不管，有一个else就执行另一个操作。

```
if 条件:
    执行语句1
(else:
    执行语句2)
```

比如说，我们编写一个语句，判断这个学生的成绩算不算高。我们以85分为标准线来衡量：

```
if score>=85:
    print('great!')
else:
    print('ganbade')
```

然后是if-elif-else。在python当中，没有switch语句，所以只能用连续if-else代替。但是注意，在连续if-else的过程当中，else if被缩写成elif。
看一个例子：我们编写一个程序，将$sigmoid(x)=\frac{1}{1+e^{-x}}$进行二值化操作：即

$$mysigmoid(x)=\begin{cases}1, x>0\\0.5, x=0\\0, x<0\end{cases},$$

就可以编写下面的语句：

```
if x>0:
    y=1
elif x=0:
    y=0.5
else:
    y=0
```

## 例题

1. 三个杯子换饮料——数值交换
   这是顺序结构里面相当经典的一个问题了。x=1,y=2，现在要交换他俩的值，怎么做？我们这样考虑啊，红杯子里面装满了可乐，蓝杯子里面装满了橙汁，怎么把红蓝的饮料互换过来？聪明如你可能已经想到了：要拿一个空杯子，把可乐倒进空杯子，橙汁倒进红杯子，可乐再倒进蓝杯子，空杯子仍然还是那个空杯子。有借有还，懂?那么这里我们也引入一个空杯子t，写如下语句就可以实现

```
x=1,y=2
t=x,x=y,y=t
```

那么你们可以思考，现在有三个数x=1,y=8,z=3
变成x=3,y=1,z=8,应该怎么弄呢

2. 分段函数：写出函数$relu(x)=\begin{cases}x, x>=0\\0, x<0\end{cases}$,

```
if x>=0:
    y=x
else:
    y=0
```

## 习题

1. 编写判断平年闰年的程序
2. 猜拳程序：1.从控制台输入你要出的拳 —石头(1)/剪刀(2)/布(3)2.电脑随即出拳 3.比较胜负（提示：要用random模块，自己可以在CSDN上面查资料，很简单的）
3. 收税问题：假设，我是说假设啊：1000元以内税率5%，超过一千不到3000的部分收10%。超过三千不到一万的部分收15%，超过一万部分的收20%。编写程序，对于给定的输入x能够计算要收多少税
   （要求编写完整的python文件，不是交互式界面编程）

# 第四章. 循环语句

## while语句

与其它类型的语言类似，python的while语句规则仍然是

```
while 条件:
    语句
```

可以看一个例子哈：

```
while i<100:
     i+=1
```

编写语句，最后结果等于多少呢？聪明如你自己试一试吧

## for语句

与c语言的for语句不同，python的for语句是这样的：

```
for 元素 in 集合:
    操作
```

例如，我们如果要输出1，2，3，4，5，我们可以这样子：

```
for i in range(5):
    print(i+1)
```

这里的range(5）是创建一个从0到5的列表，最后一个元素是4（range的数是列表元素的个数）那么range(5)=【0，1，2，3，4】

## 习题

到现在为止，我们已经学习了python的基本流程控制，已经可以解一些简单的问题了。

1. 求1到100的所有数求和

```
sum=0
for i in range(100)
    a=i+1
    sum+=a
```

2. 打印九九乘法表

这道题要进行循环加流程控制。一方面要控制乘号两边的数大小，另一方面也要控制行列。

```
row=1
while row<=9:
    col=1
    while col <=row:
        print('%d*%d=%d\t' %(row,col,row*col) ,end='')
        col+=1
    print()
    row+=1
```

```
>>> while row<=9:
      col=1
      while col <=row:
        print('%d*%d=%d\t' %(row,col,row*col) ,end='')
        col+=1
      print()
      row+=1


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

3. 求1000以内的所有质数

这里需要知道一个定理，就是我们在判断x是否是质数的时候只需要遍历到根号x。而一般情况下，我们也会用x/2来代替。

range(2,100)是从2开始一直到99，这样的一个顺序列表。

```
#求质数
i=2
for i in range (2,1000):
```

```
    j=2
    for j in range(2,int(i/2)):
        if(i%j==0):
            break
    else:
        print(i)
```

```
>>> i=2
>>> for i in range (2,1000):
    j=2
    for j in range(2,int(i/2)):
        if(i%j==0):
            break
    else:
        print(i)


2
3
4
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
```



4. 模拟登录

用户登陆程序需求:

1.输入用户名和密码

2.判断用户名和密码是否正确(name = ‘root’,passwd=‘westos’)

3.为了防止暴力破解，登陆仅有三次机会，如果超过三次，程序就报错

```
for i  in range(3):
    yonghu=input('用户名：')
    mima=input('密码:')
    if ((yonghu=='root')and(mima=='westos')):
        print('登陆成功')
        break
    else:
        print('用户名或密码错误')
        print('您还剩余%d次机会'%(2-i))
else:
```

```
print('您的机会已用尽，请稍后再试')
```

# 思考题

1. 求30的阶乘
2. 打印20以内的斐波那契数
3. 打印杨辉三角
4. 判断一个整数是否是回文数。回文数是指正序（从左向右）和倒序（从右向左）读都是一样
   的整数。
   示例:
   示例 1:
   输入: 121
   输出: true
   #示例 2:
   输入: -121
   输出: false
   解释: 从左向右读, 为 -121 。 从右向左读, 为 121- 。因此它不是一个回文数。
   #示例 3:
   输入: 10
   输出: false
   解释: 从右向左读, 为 01 。因此它不是一个回文数（leetcode经典题）

# 第五章.异常捕获与处理

## 5.1 异常种类：

Python的异常处理能力是很强大的，它有很多内置异常，可向用户准确反馈出错信息。在Python中，异常也是对象，可对它进行操作。BaseException是所有内置异常的基类，但用户定义的类并不直接继承BaseException，所有的异常类都是从Exception继承，且都在exceptions模块中定义。Python自动将所有异常名称放在内建命名空间中，所以程序不必导入exceptions模块即可使用异常。一旦引发而且没有捕捉SystemExit异常，程序执行就会终止。如果交互式会话遇到一个未被捕捉的SystemExit异常，会话就会终止。

内置异常类的层次结构如下：

```
BaseException  # 所有异常的基类
 +-- SystemExit  # 解释器请求退出
 +-- KeyboardInterrupt  # 用户中断执行(通常是输入^C)
 +-- GeneratorExit  # 生成器(generator)发生异常来通知退出
 +-- Exception  # 常规异常的基类
      +-- StopIteration  # 迭代器没有更多的值
      +-- StopAsyncIteration  # 必须通过异步迭代器对象的__anext__()方法引发以停止迭代
      +-- ArithmeticError  # 各种算术错误引发的内置异常的基类
      |    +-- FloatingPointError  # 浮点计算错误
      |    +-- OverflowError  # 数值运算结果太大无法表示
      |    +-- ZeroDivisionError  # 除(或取模)零 (所有数据类型)
      +-- AssertionError  # 当assert语句失败时引发
      +-- AttributeError  # 属性引用或赋值失败
      +-- BufferError  # 无法执行与缓冲区相关的操作时引发
      +-- EOFError  # 当input()函数在没有读取任何数据的情况下达到文件结束条件(EOF)时引发
      +-- ImportError  # 导入模块/对象失败
      |    +-- ModuleNotFoundError  # 无法找到模块或在在sys.modules中找到None
      +-- LookupError  # 映射或序列上使用的键或索引无效时引发的异常的基类
      |    +-- IndexError  # 序列中没有此索引(index)
      |    +-- KeyError  # 映射中没有这个键
      +-- MemoryError  # 内存溢出错误(对于Python 解释器不是致命的)
      +-- NameError  # 未声明/初始化对象 (没有属性)
      |    +-- UnboundLocalError  # 访问未初始化的本地变量
      +-- OSError  # 操作系统错误，EnvironmentError，IOError，WindowsError，socket.
      |    +-- BlockingIOError  # 操作将阻塞对象(e.g. socket)设置为非阻塞操作
      |    +-- ChildProcessError  # 在子进程上的操作失败
```

```
|    +-- ConnectionError  # 与连接相关的异常的基类
|    |    +-- BrokenPipeError  # 另一端关闭时尝试写入管道或试图在已关闭写入的套接
|    |    +-- ConnectionAbortedError  # 连接尝试被对等方中止
|    |    +-- ConnectionRefusedError  # 连接尝试被对等方拒绝
|    |    +-- ConnectionResetError    # 连接由对等方重置
|    +-- FileExistsError  # 创建已存在的文件或目录
|    +-- FileNotFoundError  # 请求不存在的文件或目录
|    +-- InterruptedError  # 系统调用被输入信号中断
|    +-- IsADirectoryError  # 在目录上请求文件操作(例如 os.remove())
|    +-- NotADirectoryError  # 在不是目录的事物上请求目录操作(例如 os.listdir()
|    +-- PermissionError  # 尝试在没有足够访问权限的情况下运行操作
|    +-- ProcessLookupError  # 给定进程不存在
|    +-- TimeoutError  # 系统函数在系统级别超时
+-- ReferenceError  # weakref.proxy()函数创建的弱引用试图访问已经垃圾回收了的对象
+-- RuntimeError  # 在检测到不属于任何其他类别的错误时触发
|    +-- NotImplementedError  # 在用户定义的基类中，抽象方法要求派生类重写该方法或
|    +-- RecursionError  # 解释器检测到超出最大递归深度
+-- SyntaxError  # Python 语法错误
|    +-- IndentationError  # 缩进错误
|         +-- TabError  # Tab和空格混用
+-- SystemError  # 解释器发现内部错误
+-- TypeError  # 操作或函数应用于不适当类型的对象
+-- ValueError  # 操作或函数接收到具有正确类型但值不合适的参数
|    +-- UnicodeError  # 发生与Unicode相关的编码或解码错误
|         +-- UnicodeDecodeError  # Unicode解码错误
|         +-- UnicodeEncodeError  # Unicode编码错误
|         +-- UnicodeTranslateError  # Unicode转码错误
+-- Warning  # 警告的基类
     +-- DeprecationWarning  # 有关已弃用功能的警告的基类
     +-- PendingDeprecationWarning  # 有关不推荐使用功能的警告的基类
     +-- RuntimeWarning  # 有关可疑的运行时行为的警告的基类
     +-- SyntaxWarning  # 关于可疑语法警告的基类
     +-- UserWarning  # 用户代码生成警告的基类
     +-- FutureWarning  # 有关已弃用功能的警告的基类
     +-- ImportWarning  # 关于模块导入时可能出错的警告的基类
     +-- UnicodeWarning  # 与Unicode相关的警告的基类
     +-- BytesWarning  # 与bytes和bytearray相关的警告的基类
     +-- ResourceWarning  # 与资源使用相关的警告的基类。被默认警告过滤器忽略。
```

## 5.2 异常捕获方法

当发生异常时，我们就需要对异常进行捕获，然后进行相应的处理。python的异常捕获常用try...except...结构，把可能发生错误的语句放在try模块里，用except来处理异常，每一个try，都必须至少对应一个except。此外，与python异常相关的关键字主要有：

| 关键字 | 关键字说明 |
| --- | --- |
| try/except | 捕获异常并处理 |
| pass | 忽略异常 |
| as | 定义异常实例（except MyError as e） |
| else | 如果try中的语句没有引发异常，则执行else中的语句 |
| finally | 无论是否出现异常，都执行的代码 |
| raise | 抛出/引发异常 |



1. 所有异常

```
try:
    <语句>
except:
    print('异常说明')
```

2. 指定异常

```
try:
    <语句>
except <异常名>:
    print('异常说明')
```

3. 多异常

```
try:
    <语句>
except (<异常名1>, <异常名2>, ...):
    print('异常说明')
```

或者

```
try:
     <语句>
except <异常名1>:
      print('异常说明1')
except <异常名2>:
      print('异常说明2')
except <异常名3>:
      print('异常说明3')
```

4. else

```
try:
     <语句>
except <异常名1>:
      print('异常说明1')
except <异常名2>:
      print('异常说明2')
else:
      <语句>  # try语句中没有异常则执行此段代码
```

5. finally

```
str1 = 'hello world'
try:
    int(str1)
except IndexError as e:
    print(e)
except KeyError as e:
    print(e)
except ValueError as e:
    print(e)
```

```
else:
    print('try内没有异常')
finally:
    print('无论异常与否,都会执行我')
```

6. raise 触发

```
raise [Exception [, args [, traceback]]]
```

语句中Exception是异常的类型（例如ValueError），参数是一个异常参数值。该参数是可选的，如果不提供，异常的参数是"None"。最后一个参数是跟踪异常对象，也是可选的（在实践中很少使用）。

7. 采用traceback模块查看异常
   发生异常时，Python能“记住”引发的异常以及程序的当前状态。Python还维护着traceback（跟踪）对象，其中含有异常发生时与函数调用堆栈有关的信息。记住，异常可能在一系列嵌套较深的函数调用中引发。程序调用每个函数时，Python会在“函数调用堆栈”的起始处插入函数名。一旦异常被引发，Python会搜索一个相应的异常处理程序。如果当前函数中没有异常处理程序，当前函数会终止执行，Python会搜索当前函数的调用函数，并以此类推，直到发现匹配的异常处理程序，或者Python抵达主程序为止。这一查找合适的异常处理程序的过程就称为“堆栈辗转开解”（StackUnwinding）。解释器一方面维护着与放置堆栈中的函数有关的信息，另一方面也维护着与已从堆栈中“辗转开解”的函数有关的信息。

```
try:
    block

except:

    traceback.print_exc()
```

比如

```
try:
    1/0
except Exception as e:
    print(e)
```

这样它可以帮我们追溯到出错点，有兴趣的可以试试
这部分主要是转载别人的，用的不多但有用。就不布置题目了

# 第六章. 列表，元组，字典与集合

## 6.1. 列表

与C语言数组类似，python也有自己的集合数据类型。我们先从最简单的列表说起。列表不同于C数组，它是一个整体，非常灵活。每一个元素的类型可以不同，无需像C一样指定其类型，这是个大大的优点。在C当中，你们有谁见过a[2]=3,a[3]='string'这种写法的？没有吧。但是在python里面，这样的写法竟然是合法的！

### 6.1.1 创建列表的方法：

```
1. a=[]#创建一个空列表
2. a=[1,2,3,4,5]#指定列表
3. a=range()#利用range函数
4. a=list("abcdef")#将其他类型转换为列表，但是注意：单一元素是不能转换的
```

这里补充讲两点：
一. range函数：语法是range(start, end, step)。其中start为起始元素大小；end为最后的元素加步长；step为步长。step为正数则增长，为负数就递减。start不指定就是0，step不指定就是1。
二. 列表的元素本身也可以是列表，也可以是其他的集体数据类型比如字典

### 6.1.2 列表操作

**列表操作包含以下函数:**
1、cmp(list1, list2)：比较两个列表的元素
2、len(list)：列表元素个数
3、max(list)：返回列表元素最大值
4、min(list)：返回列表元素最小值
5、list(seq)：将元组转换为列表

**列表操作包含以下方法:**

1、list.append(obj)：在列表末尾添加新的对象

2、list.count(obj)：统计某个元素在列表中出现的次数

3、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）

4、list.index(obj)：从列表中找出某个值第一个匹配项的索引位置

5、list.insert(index, obj)：将对象插入列表

6、list.pop(obj=list[-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

7、list.remove(obj)：移除列表中某个值的第一个匹配项

8、list.reverse()：反向列表中元素

9、list.sort([func])：对原列表进行排序

另外，字符串里面还有一招复制粘贴，叫做'abc'*3='abcabcabc'

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3,4,5,6]
>>> a.append(7)
>>> a
[1, 2, 3, 4, 5, 6, 7]
>>> a.count(1)
1
>>> len(a)
7
>>> sum(a)
28
>>> a.extend([8,9])
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a.index(4)
3
>>> a.pop()
9
>>> a
[1, 2, 3, 4, 5, 6, 7, 8]
>>> a.remove(4)
>>> a
[1, 2, 3, 5, 6, 7, 8]
>>> a.reverse()
>>> a
[8, 7, 6, 5, 3, 2, 1]
>>> a.sort()
>>> a
[1, 2, 3, 5, 6, 7, 8]
>>> a*2
[1, 2, 3, 5, 6, 7, 8, 1, 2, 3, 5, 6, 7, 8]
>>> min(a),max(a)
(1, 8)
>>>
```



列表删除还可以用del，判断元素是否在列表当中可以in。比如 i in a返回一个布尔值例如True。

关于append和extend这里一篇博客大家可以康康区别

## 6.1.3 习题

1. =通讯录管理系统=
   1.增加姓名和手机
   2.删除姓名
   3.修改手机
   4.查询所有用户
   5.根据姓名查找手机号
   6.退出
   ==========

```
name=[]
number=[]
a=('''
====通讯录管理系统====
1.增加姓名和手机
2.删除姓名
3.修改手机
4.查询所有用户
5.根据姓名查找手机号
6.退出
======================
请选择:
''')
while True:
    b=input(a)
    if b not in ("1","2","3","4","5","6"):
        input("输入有误请重新输入")
    else:
        if b==("1"):

            name1=str(input("请输入姓名"))
            if name1 in name:
                print("已有此联系人请重新输入")
                continue
            else:
                name.append(name1)
                number1=str(input("请输入手机号"))
                number.append(number1)
                print("输入完成")
        elif b==("2"):
            name1=str(input("请输入要删除的联系人"))
            c=name.index(name1)
            name.remove(name1)
            del number[c]
            print("")
        elif b==("3"):
```

```
            name1=str(input("请输入要修改的联系人"))
            c=name.index(name1)
            d=str(input("要修改的手机号"))
            number[c]=d
            print("修改完成")
        elif b==("4"):
            for i in name:
                print("所有用户有",i)
        elif b==("5"):
            name1=str(input("请输入您要查找的联系人"))
            c=name.index(name1)
            print("您要查找的手机号是",number[c])
        elif b==("6"):
            print("感谢使用")
            break
        else:
            print("输入有误请重新输入")
```

2.

```
1.系统里面有多个用户，用户的信息目前保存在列表里面
    users = ['root','westos']
    passwd = ['123','456']
2.用户登陆(判断用户登陆是否成功
    1).判断用户是否存在
    2).如果存在
        1).判断用户密码是否正确
        如果正确，登陆成功，推出循环
        如果密码不正确，重新登陆，总共有三次机会登陆
    3).如果用户不存在
    重新登陆，总共有三次机会
```



```
users = ['root','westos']
passwords = ['123','456']

#尝试登录次数
trycount = 0

while trycount < 3:
        inuser = input('用户名: ')
        inpassword = input('密码: ')

        trycount += 1

        if inuser in users:
                index = users.index(inuser)
                password = passwords[index]

                if inpassword == password:
                        print('%s登录成功' %(inuser))
                break
                else:
```

```
                print('%s登录失败 : 密码错误' %inuser)
        else:
                print('用户%s不存在' %inuser)
else:
        print('尝试超过三次，请稍后再试')
```

3. 冒泡排序法：大家还记得那个换杯子的原理吗？现在要利用换杯子原理，不断交换列表中两个相邻元素的值使其排序。大家动手做一下。这个列表为【2，8，1，6，5，4，7，3】

# 6.2 元组

元组与列表类似。不过元组是小括号。然后相对于列表的灵活性，元组是不可变的，一旦创建就没办法修改元素。然后如果元组只有一个元素，后面要加一个逗号。这是单元素元组的特性，所以创建元组的时候也可以像“a=3,”这样来创建。

现在我们来比较一下元组和列表：

1. 元组中的数据一旦定义就不允许更改。
2. 元组没有append()、extend()和insert()等方法，无法向元组中添加元素。
3. 元组没有remove()或pop()方法，也无法对元组元素进行del操作，不能从元组中删除元素。
4. 从效果上看，tuple()冻结列表，而list()融化元组。
5. 元组的速度比列表更快。如果定义了一系列常量值，而所需做的仅是对它进行遍历，那么一般使用元组而不用列表。
6. 元组对不需要改变的数据进行“写保护”将使得代码更加安全。
7. 元组可用作字典键（特别是包含字符串、数值和其它元组这样的不可变数据的元组）。列表永远不能当做字典键使用，因为列表不是不可变的。

# 6.3 字典

字典这种数据比较奇葩，它是python中内置的映射数据。字典的每个元素都由键-值对构造而成。

# 6.3.1 特性

字典的基本行为在很多方面与序列类似：

（1）len(d) 返回 d 中项（键-值对）的数量

（2）d[k] 返回关联到键 k 上的值

（3）d[k] = v 将值 v 关联到键 k 上

（4）del d[k] 删除键为 k 的项

（5）k in d 检查 d 中是否有含有键为 k 的项

尽管字典和列表有很多特性相同，但也有下面一些重要的区别。

（1）键类型：字典的键不一定为整型数据（但也可能是），也可能是其他不可变类型，比如浮点型

（2）自动添加：即使那个键起初在字典中并不存在，也可以为它分配一个值，这样字典就会建立新的

（3）成员资格：表达式 k in d (d 为字典) 查找的是键，而不是值。

在每个转换（conversion specifier）中的 % 字符后面，可以加上用圆括号括起来的键，后面

# 6.3.2 方法

**dict()** -> 生成一个空字典。
dict(mapping) -> 从映射对象（键，值）对新字典进行初始化。
dict(iterable) -> 通过iterable(可迭代对象)对新字典进行初始化。
dict(**kwargs) ->在关键字参数列表中使用name = value对新字典进行初始化。
**clear()函数**
描述：删除字典d中所有的键值对。
语法： d.clear() -> None 返回一个空字典
**copy()函数**
描述：生成一个新字典，浅复制字典d中的所有键值对。即父对象不变，但子对象改变。
语法： d.copy() ->返回字典d的浅拷贝
**fromkeys()函数**
描述：创建一个新字典，以可迭代对象iterable中元素做字典的键，value 为字典所有键对应的

初始值，默认值为None。

语法：fromkeys(iterable, value=None, /)

iterable —— 可迭代的对象。如 列表，元组，集合，字典，字符串等

value —— 可迭代的对象或数字，字母。如 列表，元组，集合，字典，字符串等可迭代对象

**get()函数**

描述：返回d[k],若字典d中无k键，则返回d

语法： d.get(k,d) -> D[k] if k in D, else d. d 默认值为 None.

k —— 键

d —— 若k不在字典d中，则返回自定义的d,d默认值为None.

**items()函数**

描述：返回字典d中所有键值对

语法： d.items()

**keys()函数**

描述：返回字典d中所有键的信息。

语法：d.keys()

**values()函数**

描述：返回字典d中所有值的信息。

语法：d.values()

**pop()函数**

描述：键存在则返回相应的值，同时删除该键值对，否者返回默认值d。

语法： d.pop(key,d) -> v 删除指定的键并返回相应的值。如果未找到key，则返回d（如果给定），否则引发KeyError错误

key —— 要删除的键

d —— 如果没有指定要删除的key,则返回d值。

**popitem()函数**

描述：随机从字典d中取出一个键值对，并以元组(key,value)的形式返回,同时将该键值对从字典d中删除。若字典d为空，则引发KeyError。

语法：d.popitem() -> (k, v) 返回键值对

**setdefault()函数**

描述：如果字典d存在键k，则返回d[k],否者返回d,并将d[k]的值设为d,且将该键值对(k,d)保存在字典d中。

语法：d.setdefault(k,d) -> d.get(k,d) , also set d[k]=d if k not in d

k —— 要查找或想要添加的键

d —— 如果字典d不存在k键，则返回d

**update()函数**

描述：将a中的所有对象添加到字典d中

语法：d.update(a) -> None 返回值为空

a —— 可以是：

①一个字典

②关键字 如 six=6

③一个包含一个及一个以上元组的列表
④一个包含一个及一个以上列表的元组
⑤一个zip()函数 如 zip(["a","b"],[1,2]) 等
弄一个大程序来康康这些操作吧

```
d = dict() #创建空字典
dt = {} #创建的是空字典，不是空集合
print(d,dt,type(d),type(dt))

d1 = dict(zip([1,2,3],["a","b","c"]))  #具有明显的映射关系
d2 = dict(zip(("a","b","c"),(1,2,3,5))) #第二个元组多个5，不影响映射。
d3 = dict({1:"a",2:"b",5:"c"})
print(d1,d2,d3)

d1 = dict([("a",66),(2,5)]) #可迭代的对象的元素只能是值，不能是未定义的变量。
print(d1)

#d1 等效于：
"""d = {}
iterable = [("a",66),(2,5)]
for k, v in iterable:
    d[k] = v"""

d2 = dict([(1,c),(2,"a")]) #程序报错，c可做变量，但未定义。
print(d2)

d1 = dict(one=1,two=2,three=3)
print(d1)

d = {"one":1,"two":2,"three":3}
print("清空前:",d)
d.clear() #清空字典d所有键值对
print("清空后:",d)

import copy

d1 = {"小明":98,"小红":99,"小东":[33,56,65]}
print("原始数据: ",d1)

d2 = d1 # 引用拷贝 d1和d2指向同一个内存地址，d2只是d1的一个别名。
d3 = d1.copy() #浅拷贝，d3和d2的父对象相互独立，但子对象[33,56,65]指向同一个内存地址。
d4 = copy.copy(d1) #深拷贝，父对象，子对象都相互对立。

d1["小明"] = 86 #将字典d1中父对象"小明"的分数修改为 86
d2["小东"].append(95)  #向字典d1中子对象列表[33,56,65]添加一个元素 95
```

```
print("修改后的数据:",d1)
print("引用拷贝:",d2) #引用拷贝，父对象，子对象都改变
print("浅拷贝:",d3) #浅拷贝，父对象不改变，但子对象改变
print("深拷贝:",d4) #深拷贝，父对象，子对象都不改变。

d1 = {}
d2 = {}
d3 = {}
d4 = {}

print(d1.fromkeys("123")) # iterable为字符串，value为默认值None
print(d2.fromkeys([1,2,"a"],["a",1])) #iterable为列表，value为列表
print(d3.fromkeys({1,"b",2},(1,2))) #iterable为集合，value为元组
print(d4.fromkeys({"a":1,"b":2},1)) #iterable为字典，value为数字 1

d = {"one":1,"two":2,"three":3}
print(d.get("one",666)) #键"one"存在，返回d["one"]，不返回d值 666
print(d.get("five",666)) #键"five"不存在，返回d值 666
print(d.get("five")) #键"five"不存在，返回d的默认值 None

d = {"one":1,"two":2,"three":3}
print(d.items(),type(d.items()))
d = {"one":1,"two":2,"three":3}
print(d.keys(),type(d.keys()))
d = {"one":1,"two":2,"three":3}
print(d.values(),type(d.values()))
d = {"one":1,"two":2,"three":3,"four":4,"five":5}
print(d.pop("one")) #返回 键"one"的值
print(d.pop("six",6)) #字典d中无 "six"键，返回d值给定
print(d)
print(d.pop("seven")) #字典d中无 "seven"键，且没有给定d值，程序报错

d = {"one":1,"two":2,"three":3,"four":4,"five":5}
d1 = dict() #创建空字典
print(d.popitem(),type(d.popitem))
print(d1.popitem()) #字典d1为空，程序会报错

d = {"one":1,"two":2,"three":3,"four":4,"five":5}
print(d.setdefault("one",36)) #字典d中存在键"one",返回键"one",对应的值
print(d.setdefault("six",6)) # 字典d中不存在键 "six",返回6,并将("six",6)键值对保存在
print(d) #字典d增加了一对新的键值对。

d = {"one":1,"two":2}
d1 = {"three":3,"five":5} #字典
ls = [("six",6),("seven",7)] #包含元组的列表
t = (["eight",8],["ten",10]) #包含列表的元组
d.update(d1)
```

```
d.update(ls)
d.update(t)
d.update(eleven=11,twelve=12) #关键字
d.update(zip(["thirteen","fourteen"],[13,14])) #zip
print(d)
print(d.update(d1)) #返回值为空
```

## 6.3.3 深拷贝与浅拷贝

深拷贝与浅拷贝的区别
深拷贝与浅拷贝的区别

## 6.3.4 习题

数字重复统计

随机生成1000个整数
数字范围[20,100]
升序输出所有不同的数字及其每个数字的重复次数

```
import random   ##生成随机数
all_nums = []  ##定义空列表存取随即数
for item in range(1000):   ##定义随即数的个数
     all_nums.append(random.randint(20,100))  ##在列表中添加随即数并指定随机数的范围
     sorted_nums = sorted(all_nums)  ##排序
     num_dict = {}  ##定义空字典
for num in sorted_nums:  ##对排序的数进行编历
    if num in num_dict:   ##如果这个数在字典中
        num_dict[num] += 1   ##字典中这个数对应的value数值加1
    else:
        num_dict[num] = 1   ##如果不在则定义value数值为1
print(num_dict)
```

# 6.4 集合

## 6.4.1 特性

集合是一个无序的，不重复的数据集合。也无法通过数字进行索引（字典也是无序的）其基本功能包括下面两种：

1. 去重：把一个还有重复元素的列表或元组等数据类型转变成集合，其中的重复元素只出现一次。使用set（）方法。
2. 进行关系测试：测试两组数据之间的交集，差集，并集等数据关系。
   大括号或 set() 函数可以用来创建集合。
   集合表示：{元素，元素 ， 元素 ， }
   注意:想要创建空集合,你必须使用set() 而不是 {} 。{}用于创建空字典
   集合的添加删除：
   .add( ) 添加一项
   .update() 添加多项
   .remove()可以删除一项
   循环遍历:和列表用法相同

集合对象还支持 并 、交、 差、 对称差集、 等数学运算：
union( 联合)并集 把两个集合合并在一起如果有重复留一个 表示符：&
intersection(交)交集 输出相同的元素 表示符： |
difference(差)差集 减掉另一个 剩下自己的一部分 表示符： -
sysmmetric difference(对称差集) 减掉相同的输出剩余的 表示符：^

## 6.4.2 用法

| | |
|---|---|
| add() | 为集合添加元素 |
| clear() | 移除集合中的所有元素 |
| copy() | 拷贝一个集合 |
| difference() | 返回多个集合的差集 |
| difference_update() | 移除集合中的元素，该元素在指定的集合也存在。 |
| discard() | 删除集合中指定的元素 |
| intersection() | 返回集合的交集 |
| intersection_update() | 返回集合的交集。 |
| isdisjoint() | 判断两个集合是否包含相同的元素，如果没有返回 True，否则返回 False。 |
| issubset() | 判断指定集合是否为该方法参数集合的子集。 |
| issuperset() | 判断该方法的参数集合是否为指定集合的子集 |
| pop() | 随机移除元素 |
| remove() | 移除指定元素 |
| symmetric_difference() | 返回两个集合中不重复的元素集合。 |
| symmetric_difference_update() | 移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。 |
| union() | 返回两个集合的并集 |
| update() | 给集合添加元素 |



## 6.4.3 习题

小明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性他先用计算机生成了N个1～1000之间的随机整数(N<=1000),
N是用户输入的，对于其中重复的数字，只保留一个，把其余相同的数字去掉，不同的数对应着不同的学生的学号，
然后再把这些数从小到大排序，按照排好的顺序去找同学做调查，请你协助明明完成“去重”与排序工作

分析：集合可以用来去重，即每生成一个随机数便将其加入到定义的空集合中集合即可
sorted函数可以对集合进行排序

```
import random
# 接收用户输入
N = int(input('N:'))
# 定义空集合;用集合便可以实现自动去重(集合里面的元素是不可重复的)
gather = set([])
# 生成N个1～1000之间的随机整数
for i in range(N):
    num = random.randint(1,1000)
```

```
    # add:添加元素
    gather.add(num)

print(gather)
# sorted: 集合的排序
print(sorted(gather))
```

# 6.5 区别与练习

这里我就放三篇博客
第一篇
第二篇
第三篇

| | 列表 | 元组 | 集合 | 字典 |
|---|---|---|---|---|
| 英文 | list | tuple | set | dict |
| 可否读写 | 读写 | 只读 | 读写 | 读写 |
| 可否重复 | 是 | 是 | 否 | 是 |
| 存储方式 | 值 | 值 | 键(不能重复) | 键值对(键不能重复) |
| 是否有序 | 有序 | 有序 | 无序 | 无序 |
| 初始化 | [1,'a'] | ('a',1) | set([1,'a']) 或{1,2} | {'a':1,'b':2} |
| 添加 | append | 只读 | add | d['key'] = 'value' |
| 读元素 | res[2:] | t[0] | 无 | d['a'] |



**练习题**

1. 根据用于指定月份，打印该月份所属的季节。
   提示: 3,4,5 春季 6,7,8 夏季 9,10,11 秋季 12, 1, 2 冬季
2. 栈的工作原理：先进后出
   入栈
   出栈
   栈顶元素
   栈的长度
   栈是否为空

```
s=[]
info="""
     #1.入栈
     #2.出栈
```

```
        #3.栈顶元素
        #4.栈的长度
        #5.栈是否为空
    """
    print(info)
    while 1:
        choice=input('请输入选择：')
        if choice == '1':
            item=input('请输入入栈元素：')
            s.append(item)
            print('%s入栈成功' %(item))
        elif choice == '2':
            if len(s)==0:
                print('栈为空，不能出栈')
            else:
                item=s.pop()
                print('%s出栈成功' %(item))
        elif choice == '3':
            if len(s)==0:
                print('栈为空，无栈顶元素')
            else:
                print('栈顶元素为：%s' %(s[-1]))
        elif choice == '4':
            print(len(s))
        elif choice == '5':
            if len(s) == 0:
                print('栈为空')
            else:
                print('栈不为空')
        elif choice == 'q':
            print('欢迎下次使用')
            break
        else:
            print('请输入正确的选择')
```

3. 明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性
   他先用计算机生成了N个1～1000之间的随机整数(N<=1000),N是用户输入的，对于其中重复的数字，只保留一个，把其余相同的数字去掉，不同的数对应着不同的学生的学号，然后再把这些数从小到大排序，按照排好的顺序去找同学做调查，请你协助明明完成“去重”与排序工作

```
import random
# 先生成n个随机数
# 有先选择集和
s = set([])
```

```
for i in range(int(input('N:'))):
    s.add(random.randint(1,1000))
print(sorted(s))
```

4. 随机生成100个卡号;
   卡号以6102009开头， 后面3位依次是 （001， 002， 003， 100），
   生成关于银行卡号的字典， 默认每个卡号的初始密码为"redhat";
   输出卡号和密码信息， 格式如下:
   卡号 密码
   6102009001 000000

```
card=[]
for i in range(100):
    card.append('6102009%.3d' %(i))
card_id={}.fromkeys(card,'redhat')
print('卡号\t\t\t\t\t密码')
for k,v in card_id.items():
    print('%s\t\t\t%s'%(k,v))
```

5. 后台管理员只有一个用户: admin, 密码: admin
   当管理员登陆成功后， 可以管理前台会员信息.
   会员信息管理包含:
   1.添加会员信息
   2.删除会员信息
   3.查看会员信息
   4.退出

```
user=['admin']
passwd=['admin']
huiyuan=[]
huiyuan1=[]
inuser=input('用户名: ')
inpasswd=input('密码: ')
if user[0]==inuser and passwd[0]==inpasswd:
    print('登陆成功')
    info="""
             #1.添加会员信息
             #2.删除会员信息
             #3.查看会员信息
         """
    print(info)
    while 1:
```

```
            choice=input('请输入选择：')
            if choice == '1':
                str=input('请输入增加的会员名称:')
                if str in huiyuan:
                    print('%s用户存在'%(str))
                else:
                    str1=input('请输入增加的%s密码:'%(str))
                    huiyuan.append(str)
                    huiyuan1.append(str1)
                    print('%s会员增加成功' %(str))
            elif choice == '2':
                if len(huiyuan) == 0:
                    print('当前无会员信息')
                else:
                    str=input('请输入要删除的会员信息：')
                    if str not in huiyuan:
                        print('无%s会员' %(str))
                    else:
                        num=huiyuan.index(str)
                        huiyuan.pop(num)
                        huiyuan1.pop(num)
                        print('%s会员删除成功' %(str))
            elif choice == '3':
                if len(huiyuan) == 0:
                    print('当前无会员信息')
                else:
                    print('当前会员用户为：'+' '.join(huiyuan))
                    print('当前会员密码为：'+' '.join(huiyuan1))
            elif choice == 'q':
                break
            else:
                print('请输入正确的指令')
    else:
        print('登陆失败')
```

# 第七章 函数编程

## 7.1 编写函数

与C语言类似，我们也可以编写函数将代码模块化。而且没有C语言对返回值类型的限制，我们的编写显得更加灵活。
函数编写的语法规则是这个样子的：

```
def func(**args):
    do ******
    return xxxx
```

func叫做函数名称，**args是参数列表，do跟执行语句，return是返回值，可以不写。然后你比如说我们编写一个写阶乘的函数

```
def jiecheng(n):
    val=1
    for i in range(1,n+1):
        val*=i
    return val
```

```
>>> def jiecheng(n):
    val=1
    for i in range(1,n+1):
        val*=i
    return val

>>> jiecheng(10)
3628800
>>> 10*9*8*7*6*5*4*3*2*1
3628800
>>>
```

## 7.2 递归

递归函数本质就是自己的函数里面调用自己。
**优点：**

递归使代码看起来更加整洁、优雅
可以用递归将复杂任务分解成更简单的子问题
使用递归比使用一些嵌套迭代更容易
**缺点:**

递归的逻辑很难调试、跟进
递归调用的代价高昂（效率低），因为占用了大量的内存和时间。

使用递归函数需要注意防止栈溢出。在计算机中函数调用是通过栈这种数据结构实现的。每当进入一个函数调用，栈就会加一层栈帧；每当函数返回就会减一层栈帧。由于栈的大小不是无限的，因此递归调用次数过多会导致栈溢出。

解决递归调用栈溢出的方法是通过尾递归优化。

尾递归是指函数返回时调用函数本身，并且 return 语句不能包含表达式。这样，编译器和解释器就可以对尾递归进行优化。使递归本身无论调用多少次都只占用一个栈帧，从而避免栈溢出的情况。

两个经典的问题就是斐波那契数列与汉诺塔问题
斐波那契的python代码（不进行任何空时优化）

```
def fabonacci(n):
     if n==1 or n==2:
        return 1
     else:
        return fabonacci(n-1)+fabonacci(n-2)
```

汉诺塔：
汉诺塔问题是递归函数的经典应用，它来自一个古老传说：在世界刚被创建的时候有一座钻石宝塔A，其上有64个金蝶。所有碟子按从大到小的次序从塔底堆放至塔顶。紧挨着这座塔有另外两个钻石宝塔B和C。从世界创始之日起，波罗门的牧师就一直在试图把塔A上的碟子移动到C上去，其间借助于塔B的帮助。每次只能移动一个碟子，任何时候都不能把一个碟子放在比它小的碟子上面。当牧师们完成这个任务时，世界末日也就到了。
下面我们对汉诺塔问题进行求解：
（1）将塔A上的n -1个碟子借助C塔先移动到B塔上；
（2）把塔A上剩下的一个碟子移动到塔C上；
（3）将n - 1个碟子从B塔借助塔A移动到塔C上。
很显然，这是一个递归求解的过程，假设碟子数n=3时，汉诺塔问题的求解过程如下图所示：

```
def Hanoi(n, A, B, C) :
  if (n == 1) :
    move(A, c)    #表示只有一个碟子时，直接从A塔移动到C塔
```

```
    else :
        Hanoi(n - 1, A, C, B)  #将剩下的A塔上的n-1借助C塔移动到B塔
        move(A, C)              #将A上最后一个直接移动到C塔上
        Hanoi(n - 1, B, A, C)  #将B塔上的n-1个碟子借助A塔移动到C塔
```

但是重复递归会有一个问题，就是会重复计算很多东西从而耗费大量时间空间；另外python对递归深度有自己的限制，超出限制会他娘的直接炸掉

## 7.3 习题

案例1：
编写一个函数cacluate, 可以接收任意多个参数,返回的是一个元组.
元组的第一个值为所有参数的平均值, 第二个值是大于平均值的所有数.
案例2：

编写一个函数, 接收字符串参数, 返回一个元组,‘ehllo WROLD’
元组的第一个值为大写字母的个数, 第二个值为小写字母个数.
案例3：

编写函数, 接收一个列表(包含30个1~100之间的随机整形数)和一>个整形数k, 返回一个新列表.
函数需求:

- 将列表下标k之前对应(不包含k)的元素逆序;
- 将下标k及之后的元素逆序;
  [1,2,3,4,5] 2 [2,1,5,4,3]
  [1,2,3,4,5,6,7,8,9] 4 [4,3,2,1,9,8,7,6,5]
  案例四：
  题目要求：
  对于一个十进制的正整数，定义f(n)为其各位数字的平方和，如：
  f(13)=12 + 32 =10
  下面给出三个正整数k，a，b，你需要计算有所烧个正整数n满足a<=n<=b,且k*f(n)=n

输入：
第一行包括3个正整数k，a，b，k>=1,a,b<=10**18,a<=b;
输出：
输出对应的答案

范例：
输入：51 5000 10000
输出：3

思路

定义函数

接收变量

存储整型k，a，b

判断是否满足条件

答案地址

第八章 面向对象简介

面向对象（Object Oriented Programming）编程主要有三大特性：继承、封装、多态。那就从这三个方面来学习一下，也会具体说说Python类的一些特性，如新式类和旧式类，**slots**，super等等。

1、封装
封装是最好理解的，它是将内容都封装到类中，隐藏了具体的细节，然后通过属性方式来访问。

如：

```
class A:

      name = 's'
      def __init__(self,a,b):
          self.a = a
          self.b = b
```

上面一个简单的例子，将a,b两个值封装到A的属性a,b中。我们访问则是通过A().a,A().b访问。

属性也分类属性和实例属性，name是类属性，a,b是实例属性。当我们定义类属性后，就不要在实例属性中定义了，否则会覆盖掉类属性。

现在具体说一下属性。

类属性一般都保存在类的__dict__中，而实力属性一般都保存在实例的__dict__中。我们可以把相应的值打印出来：

```
class A(object):
   age = 43
   def __init__(self):
       print 'A'
       self.name = 'a'
       self.age = 23

   def test(self):
        print 'test'

a = A()
print a.__dict__
print a.age
print a.__class__.__dict__
```

```
输出：
A
{'age': 23, 'name': 'a'}
23
{'__module__': '__main__', 'age': 43, '__dict__': <attribute '__dict__' of 'A' c
```

具体分析一下上面的例子：

age分别有一个类属性和实例属性，分别保存在实例和类的__dict__中。
当我们print age的时候，是打印实例属性，而不是类属性，这和点号运算搜索过程有关。
实例本身的dict，a.**dict**;
类的dict,a.**class**.**dict**;
调用定义的__getattr__或者__getattribute__方法（新式类，具体在我的Python内置特性一文中有详细介绍）
子类;
如果经过上述几个步骤依然找不到属性，就会抛出ArrtibuteError。

> 那现在问题来了，实例的方法到底是什么访问过程呢，即a.test()？其实a.test()其实就相
>
> 当一个函数(function)定义在了class语句的块中（或者由 type 来创建的)，它会转成一个
>
> 所以， bound method 就是绑定了一个实例的方法， 否则叫做 unbound method .它们都

3.类的__dict__并不是真正意义的dict，而是一个dictproxy。它是只读的，比如，你想做：

```
a.__class__.__dict__['s'] = 1


TypeError: 'dictproxy' object does not support item assignment
```

说到属性访问，要提一下Python的描述符：property，经过property装饰的方法就可以通过属性的方式访问。比如我定义了一个

```
class A(object):

    @property
    def test（self,x）:
      pass
```

就可以通过A().test访问，此外，我们也可以通过property,.setter以及property.getter等方法来实现一些定制的读写属性。比如：

```
class A(object):

   @property
   def score(self):
      return self._score

   @score.setter
   def score(self,value):
     if value>100:
        raise ValueError('int must be lower than 100')
     self._score = value


a = A()
a.score = 101
```

系统会抛出ValueError的异常。

学过Java的都知道Java规定了public,private等属性，但在Python中是没有的，Python也可以分出不同类别的属性，比如可以用双下划线来表示，但这只是规定了一种规则，却不是强制限制，一切凭自觉。嗯，Python开发者都是素质高的一类人，哈哈。

不过，你非要写成这种形式，可以写一些代码实现：

```
class PrivateExc(Exception):
        pass


class A(object):

        def __setattr__(self, key, value):
                if key not in self.privates:
                        raise PrivateExc('{0} not in {1} '.format(key,self))
                else:
                        self.__dict__[key] = value

class B(A):
        privates = ['age']

        def __init__(self,age):
                self.age = age

        def __getattr__(self, item):
                return item

b = B(12)
```

```
b.age = 30
print b.age
b.name = 'h'

30
Traceback (most recent call last):
  File "D:/Study/Programming/Python/myworks/weiborobot/__init__.py", line 32, ir
    b.name = 'h'
  File "D:/Study/Programming/Python/myworks/weiborobot/__init__.py", line 15, ir
    raise PrivateExc('{0} not in {1} '.format(key,self))
__main__.PrivateExc: name not in <__main__.B object at 0x00000000025A7780>
```

2、继承

在Python中，继承分为单继承和多重继承（Java只有单继承），单继承比较好理解，单纯地继承父类的属性和方法。 这里只说一下多重继承。有的时候我们定义一个子类，并不是希望定义非常复杂的父类进行单继承来实现多种功能，Python支持 多继承，用的最多的就是Minxin，这在Django中也经常用到。即我希望可以实现多种功能。比如一个视图类CreateView，我既希望继承父类View的方法，又想继承表单的一些功能，那:

```
class BaseCreateView(ModelFormMixin, ProcessFormView):
    pass
```

多重继承主要考虑的问题是假如一个子类的父类有两个，如果两个父类都有相同的方法，那到底该继承哪一个父类的方法呢？这就涉及到了方法的查找顺序。

对于你定义的每一个类，Python会计算出一个所谓的方法解析顺序(MRO)列表。 当你使用super() 函数时，Python会在MRO列表上继续搜索下一个类。 只要每个重定义的方法统一使用super() 并只调用它一次， 那么控制流最终会遍历完整个MRO列表，每个方法也只会被调用一次 。 这个MRO列表就是一个简单的所有基类的线性顺序表。为了实现继承，Python会在MRO列表上从左到右开始查找基类，直到找到第一个匹配这个属性的类为止。

不同形式的类的多重继承方法也不同，类主要分为经典类和新式类。新式类指的是继承自object的类，其他的就叫做经典类。 经典类使用的继承查找顺序是深度优先，新式类使用的继承方法是C3广度优先。 深度优先和广度优先是算法中两个比较重要的搜索算法。

现在看经典类的例子:

```
class A:
    def output(self):
        print 'A'


class B(A):
```

```
    pass

class C(A):
    def output(self):
        print 'C'


class D(B,C):
    pass


d = D()
d.output()
```

输出是A

对于子类D，我希望它继承C的output，但是它最后是继承了A的output方法。对于这种问题，就产生了新式类。

新式类的写法主要是要继承object.

```
class A(object):
    def output(self):
        print 'A'


class B(A):
    pass

class C(A):
    def output(self):
        print 'C'


class D(B,C):
    pass


d = D()
d.output()
```

输出C

既然说到了新式类，那新式类除了继承带来的优势，还有什么新的特性呢?

MRO 该表
__slots__等新属性;
super;
descriptor;
（1）MRO表

上面已经说过了，即关于多重继承时候，搜索过程。

（2）__slots__等新属性;

在新式类中加入了很多的新特性。你定义完一个新式类后，打印属性，会有如下这些：

```
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribut
```

__getattribute__就是新式类的一种用法，要比__getattr__更灵活。

当我们在新式类中定义了__slots__后， Python就会为实例使用一种更加紧凑的内部表示。 实例

目前存在的最大误区是__slots__可作为封装工具防止添加新属性，虽然它的确可实现这个目的，但这不是__slots的初衷，它就是用来优化数据结构的，不过我们还是拿个例子来显示说一下用__slots__后会产生哪些效果：
class Haibo(object):
**slots** = (‘name’,‘height’)

d = D()
d.age = 23
print d.age
我们对Haibo这个类，只限定它拥有name和height两个属性，当我们试着要给实例添加age属性时，就会抛出ArributeError异常.这隐士地表明了，定义__slots__之后，就不具有__dict__特性了。

__slots__很有用，它能够为程序节省很多的内存消耗，尤其是对于那些只需要简单数据结构的类。

（3）super

super在我们编写代码时，做类继承时用得非常多，可以说遍地都是，这也证明了它给带来的优势。

一般当写子类时，我们要重构父类的方法，并也希望继承父类的方法，就要用super，用了super

此外，super最大最大的优势是在多重方面的应用。它为实例提供了不同的mro策略。

简单例子：

```
class A(object):

   def __init__(self):
       print 'A'

class B(A):

    def __init__(self):
        print 'B'
        super(B,self).__init__()

b = B()


def super(cls, inst):
    mro = inst.__class__.mro()
    return mro[mro.index(cls) + 1]
```

两个参数 cls 和 inst 分别做了两件事：

1. inst 负责生成 MRO 的 list
2. 通过 cls 定位当前 MRO 中的 index, 并返回 mro[index + 1]

当我们使用了super类的时候，就会向mro表中写入。还是拿个例子：

```
class A(object):

   def __init__(self):
       print 'A'


class B(A):

    def __init__(self):
        print 'B'
        super(B,self).__init__()


class C(A):

    def __init__(self):
        print 'C'
```

```
        super(C,self).__init__()

class D(B,C):
    pass

d = D()
print d.__class__.__mro__
```

输出：

B
C
A
(<class ‘**main**.D’>, <class ‘**main**.B’>, <class ‘**main**.C’>, <class ‘**main**.A’>, <type ‘object’>)

搜索原则：

子类会先于父类被检查
多个父类会根据它们在列表中的顺序被检查
如果对下一个类存在两个合法的选择，选择第一个父类
（4）描述符
3、多态
多态即多种形态，在编译阶段无法确定类型，在运行时确定状态。Python不像Java那样需要声明变量类型，需要在运行时确定变量的类型，其实这就是多态的体现。

多态本质上是对不同的对象可以实现相同的操作，通常是为了接口重用，即不同的类实现相同的方法。（运算符重载其实也是一种多态。）

上面的例子中其实就是一种多态，子类进行了运算符重载，不同的子类实现了相同的方法output，但输出的结果是不一样的，这就是多态。 Python中引用了“鸭子类型”，它就是一种多态，什么是鸭子类型？ “当看到一只鸟走起来像鸭子、游泳起来像鸭子、叫起来也像鸭子，那么这只鸟就可以被称为鸭子。” 我们并不关心对象是什么类型，到底是不是鸭子，只关心行为。 比如一些像文件的对象，如StringIO，可以像文件一样操作，使用文件的方法。

例子：

```
class Duck:
    def quack(self):
        print 'gua gua'

class Bird:
    def quack(self):
        print 'bird is like bird'
```

```
def speak(obj):
    obj.quack()


obj1 = Duck()
obj2 = Bird()


speak(obj1)
speak(obj2)
```

speak函数输入了两个不同的参数，他们输出的结果是不同的，这就是多态。

类的方法：

1、普通方法
普通方法很好理解，大多数都是普通方法，即第一个参数是类本身，即self.

2、类方法
classmethod装饰的方法，传入的第一个参数是cls

3、静态方法
staticmethod装饰的方法，不需要传入参数。

```
class Animal():

    def mth1(self):
        print 'normal method'

    @classmethod
    def mth2(cls):
        print 'class method'

    @staticmethod
    def mth3():
        print 'static method'


a = Animal()
a.mth1()
Animal.mth2()
Animal.mth3()
a.mth2()
a.mth3()
```

*这篇文章我是转载的，具体的面向对象在下一部提的更详细。这篇可能看的有点懵，不过问题不大，主要就是要把握好面向对象的三大特性：封装继承与多态，知道一个类有哪几个部分，怎么写，然后像__init__,self啥的搞清楚就行了。[原文链接]*

# 第九章 文件

## 9.1文件的创建与写入

如何创建文本文件

使用Python，您可以通过使用代码创建一个.文本文件(古鲁99.txt)，我们在这里演示了如何做到这一点

第1步)

```
f= open("guru99.txt","w+")
```

我们声明变量f来打开一个名为textfile.txt的文件。OPEN采用两个参数，我们要打开的文件和表示我们想对文件执行的权限或操作类型的字符串。
在这里，我们在参数中使用了“w”字母，它指示写和加号，这意味着如果库中不存在文件，它将创建一个文件。
“w”旁边的可用选项是“r”表示读，“a”表示附加和加号，意思是如果没有，则创建它。
第二步)

for i in range(10):

```
  f.write("This is line %d\r\n" % (i+1))
```

我们有一个for循环，它运行在10个数字的范围内。
使用写函数将数据输入到文件中。
我们想在文件中迭代的输出是“这是行号”，我们用写函数声明它，然后用百分比d(显示整数)声明它。
因此，我们基本上是在输入我们正在写的行号，然后将它放入一个回车和一个新的行字符中。
第3步)

```
f.close()
```

如何将数据附加到文件中

还可以将新文本附加到已经存在的文件或新文件中。

第1步)

f=open(“guru99.txt”, “a+”)

再次，如果您可以在代码中看到加号，它表示如果它不存在，它将创建一个新文件。但是在我们的例子中，我们已经有了这个文件，所以我们不需要创建一个新的文件。

第二步)

for i in range(2):

```
  f.write("Appended line %d\r\n" % (i+1))
```

这将以附加模式将数据写入文件。

如何读取文件

不仅可以从Python创建.txt文件，还可以“读取模式”®调用.txt文件。

第1步)以读取模式打开文件

```
f=open("guru99.txt", "r")
```

第二步)我们使用代码中的模式函数来检查文件是否处于打开模式。如果是，我们继续前进

```
if f.mode == 'r':
```

第3步)使用f.read读取文件数据并将其存储在可变内容中

```
contents =f.read()
```

第4步)印刷内容

## 9.2 文件的操作

文件打开方法：open（name[,mode[buf]]）
name:文件的路径
mode：打开方式
buf：缓冲buffering大小

mode 说明 注意
r 只读方式打开 文件必须存在
w 只写方式打开 文件不存在创建文件， 文件存在则清空文件内容
a 追加方式打开 文件不存在则创建文件
r+/w+ 读写方式打开
a+ 追加和读写方式打开
rb、wb、ab、rb+、wb+、ab+:二进制方式打开

```
# 只读方式
# f = open("hello.py")
# c = f.read()
# print c



# 读写方式
# f = open('1.txt','w')
# f.write("test write")
# f.close()

#追加方式打开
# f = open("hello.py",'a')
# f.write("print 'write test'")
# f.close()
```

文件的读取方式
read([size]):读取文件（读取size个字节，默认读取全部）
readline([size]):读取一行
readlines([size]):读取完文件(实际最大为缓冲区的大小)，返回每一行所组成的列表（不推荐）
iter:使用迭代器进行文件读取

```
# 文件读取方式
# f = open("1.txt")
# c = f.read()
# c = f.readline()
# c = f.readline(2)
# c = f.readlines()
# print c
# 迭代器进行文件读取
# iter_f = iter(f)
# lines = 0
# for line in iter_f:
#     lines += 1
#
# print lines
```

文件的写入方式
write(str):将字符串写入文件
writelines(sequence_of_strings):写多行到文件

```
# 文件的写入 与 写缓存
# f = open("test.txt",'w')
# f.writelines("123123123")
# f.writelines(('223','xczcxc','gfweqq'))
# f.flush()        #把缓冲区的内容写入硬盘
# f.close()
```

Python文件的指针操作
seek(offset[,whence]):移动文件指针
offset:偏移量 可以为负数
whence:偏移相对位置;

Python文件指针定位方式
os.SEEK_SET:相对文件起始位置
os.SEEK_CUR:相对文件当前位置
os.SEEK_END:相对文件结尾位置

```
f = open("1.txt",'r+')
# f.tell() //当前文件的偏移
# f.read(3)
f.seek(-5,os.SEEK_END)
# f.read()
```

python 文件的属性
file.fileno() //文件描述符
file.mode:文件打开权限
file.encoding:文件编码格式
file.closed:文件是否关闭
使用os模块操作文件
常用操作方法:
os.open(filename,flag[,mode]):打开文件
flag:打开文件的方式
os.O_CREAT:创建文件
os.O_RDONLY:只读方式打开
os.O_WRONLY:只写方式打开
os.O_RDWR:读写方式打开

os.read(fd,buffersize):读取文件
os.write(fd,str):写入文件
os.lseek(fd,pos,how):文件指针操作
os.close(fd):关闭文件
其他的操作方法:
access(path,mode) 判断该文件权限、F_OK存在,权限： R_OK(读) 、W_OK(写) 、X_OK(执行)

listdir(path) 返回当前目录下所有文件组成列表

remove(path) 删除文件

rename(old,new) 修改文件名或者目录名

mkdir(path,[,mode]) 创建目录

makedirs(path,[,mode]) 创建多级目录

removedirs(path) 删除多级目录

rmdir(path) 删除目录（必须为空目录）

示例:

```
fd = os.open("hello.txt",os.O_CREAT|os.O_RDWR)
n = os.write(fd,"test write")
l = os.lseek(fd,0,os.SEEK_SET)
str = os.read(fd,5)
os.close(fd)
```

## 9.3 任务

这其实也是转载的博客
第一篇
第二篇
任务只有一个:
创建一个文本文件hello.txt，利用python向里面写入任意一本小说当中的一段话。然后读取它并打印出来。