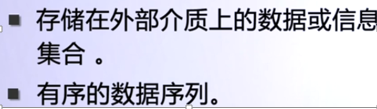
Python对文件的操作

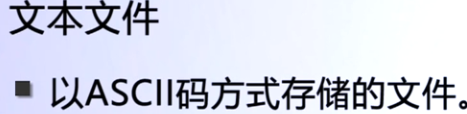
# 文件介绍

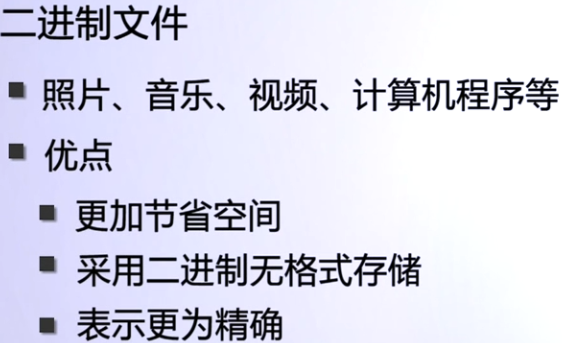
## 文件特点



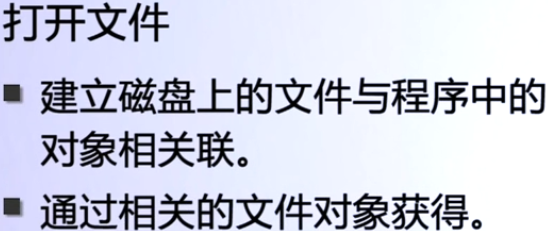
## 文本分类

根据文件数据分为**文本文件**和**二进制文件**。





# 文件的打开：



## 文件的打开模式：r、w、rb、wb、a等

### w模式与a模式的区别

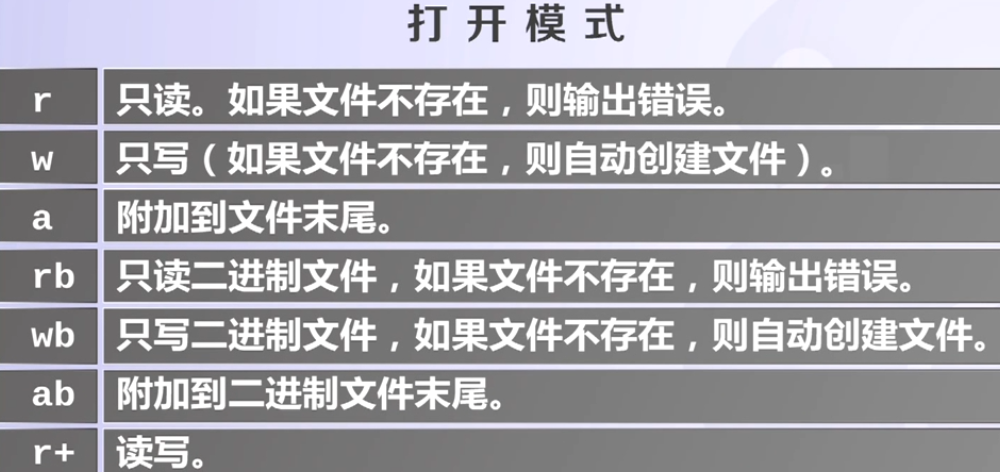
**w模式会把原来的内容清空，再写入新的数据；**

**a模式会在原来的文件后添加，即append**。

### rb模式与wb模式的使用

**r为读取文本，rb为读取二进制，同理，w为写入文本，wb为写入二进制。对于图片、视频等文件需要利用rb或wb模式。**





## 文件打开方法：利用open函数实现

### open函数的使用方法:



**需要传入文件的路径名，和打开模式，此方法返回一个文件对象**。

**在不指定文件模式的时候，默认为 ‘r’。**

**示例**：





### 编解码问题

在操作文本时，肯定离不开**编码解码**的问题，所以，提供了**decode方法**进行解码。

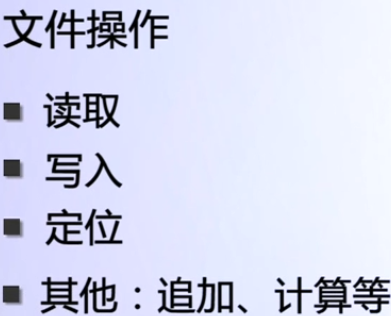
**Python提供了一个模块：codecs，这个模块能够人为设定解码方式。**

f = open('E://aaa/a.txt','r',encoding='utf-8')

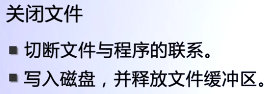
### 利用with...open as f：模板操作文件

**with...open as f：即使出现异常信息，也能够将文件对象f关闭。建议使用这种方式操作文件。**

## 文件操作



## 文件关闭



# 如何创建文件对象

## 利用open函数创建file对象

**文件对象**： 利用**open函数**可以返回一个文件对象。

**语法：file\_object = open(fiel\_name, access\_mode=’r’, buffering=-1)**

## 参数介绍：

### File\_name

是要打开的文件名字的字符串，可以使相对路径或者绝对路径，

### access\_mode可选参数

表示文件打开的模式，**’r’,’w’,’a’ 分别代表读取，写入和追加**。还有个’U’模式，代表通用换行符支持。

**‘r’或者’U’模式打开的文件必须是已经存在的，使用’w’模式打开的文件若存在则首先清空，然后重新创建**。**以’a’模式打开的文件是为追加数据准备的，所有写入数据都将追加到文件的末尾**。

### buffering参数

另一个可选参数buffering用于指示访问文件所采用的缓冲方式，0表示不缓冲，1表示缓冲一行数据，大于1代表用给定值作为缓冲区大小，不提供参数或者负值代表使用系统默认缓冲机制。

　　File（）和open（）的用法相同。

# 文件对象的方法：

## read方法介绍:

### read()没有参数就的，就是把所有的数据都出来。如果文件中的内容有10个G，内存一下子就会溢出，所以，一般指定size进行读取数据。

### read(size):指定读取size个字符。

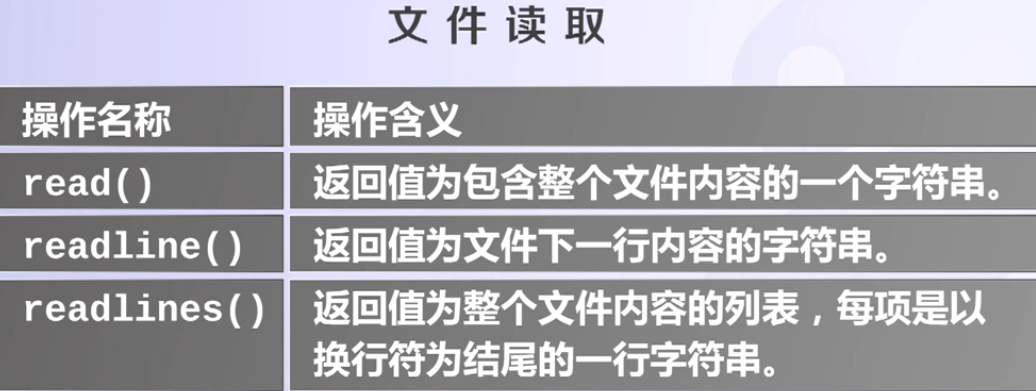
### readline方法：每次读取一行。

### readlines方法：读取所有行，返回一个list。



### read方法选择使用的原则

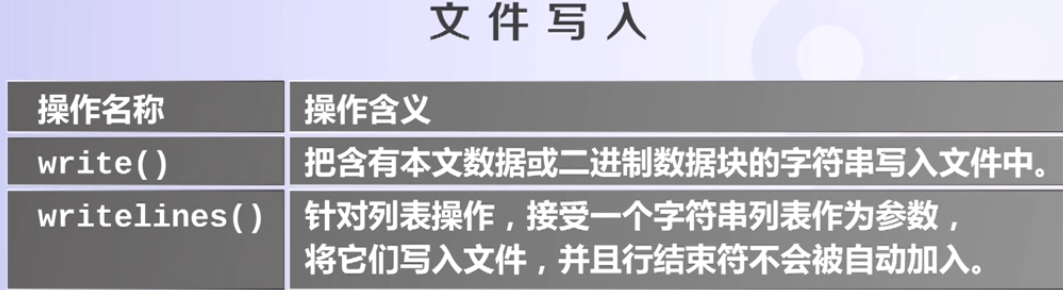
**总结：如果不知道文件的具体大小，用read(size)方法最好；如果文件较小，可以直接使用read()方法，一次读取所有数据；如果是配置文件，用readlines()方法最好，可以单独一行一行地处理。**

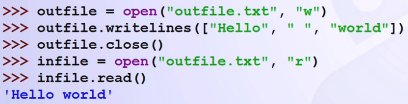


## write方法：

### write（str）：写入一个字符串。

### writeline（）：写入一行。





## close方法：

**文件操作后，一定要关闭**。

但是，一旦出现异常，文件的close方法就可能执行不到，所以需要利用try。。。。。finally。。。语句，但是引入了新的功能语句，with语句。所以必须会使用with语句。

## tell方法：获取当前文件的指针。

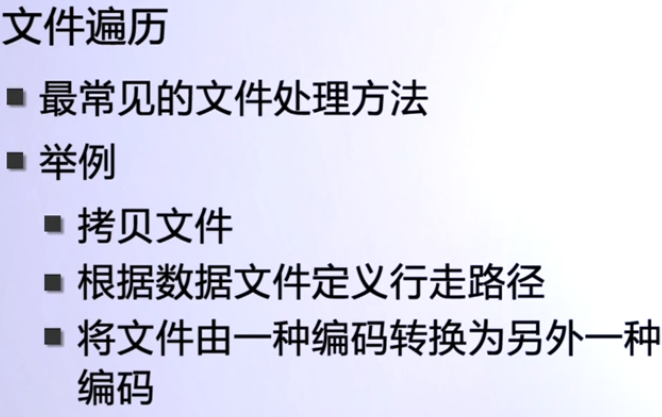
指针是以**字节数**计算的。

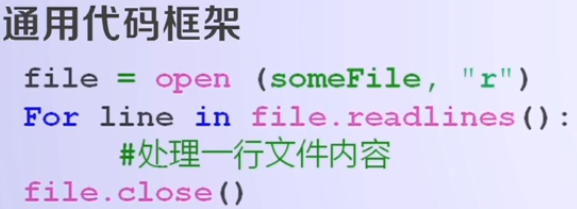
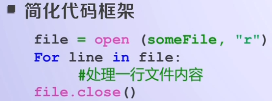
## seek方法：指定文件指针，文件内移动。

seek（）方法可以**在文件中移动文件指针到不同的位置**，offset字节代表对于某个位置偏移量，位置的默认值为0，代表从文件开头算起，1代表从当前位置算起，2代表从文件末尾算起。和C语言总的fseek（）类似。



# 文件遍历



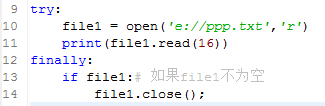


# try ...except...finally语句：和with语句。

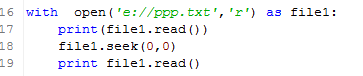
**善于使用with语句操作文件IO是一个很好的习惯。**

**说白了，with语句就是可以让系统自动调用close方法，不需要人为调用了。**

## try...finally语句



## with语句： 由于每次都要写close方法，比较繁琐，所以，引入了with语句，Python自动调用close方法。



## 注意：Python中的with语句与Java中try-with-resource语句差不多，只不过是Java中没有出现with关键词，是try(){}。而Python中with语句使用的是with-as。

# Python中对目录和文件的操作

**对于目录及文件的操作方法在os库中**

**需要导入 os库**，即 **import os**

## 判断目录或文件是否存在：os.path.exists(path)

**os.path.exists(path)**

**path既可以为目录路径也可以为文件路径**。



## 删除文件或空目录

删除文件：os.remove(filePath)

删除空目录： os.rmdir(dirPath)

递归删除空目录：os.removedirs(dirsPath)

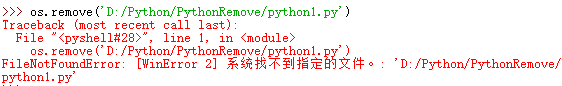
### 删除文件os.remove(filePath)

删除文件成功后，不返回任何值：



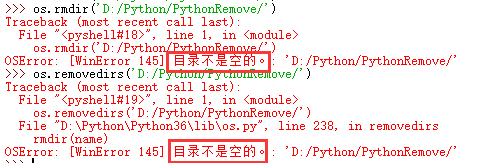
如果文件找不到，删除失败，则报出错误：

**所以，在删除文件前，首先判断是否存在，存在再删除。以防程序报出错误。**



### 删除空目录os.rmdir()与os.removedirs()

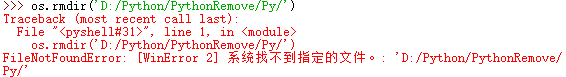
目录下有文件是不能删除目录的，os.rmdir()与os.removedirs()都只能删除空目录（即目录下没有文件），否则会报出错误。



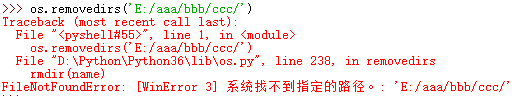
当**删除成功**时，不返回任何值，如



当**目录不存在**时，报错异常，所以在删除目录时，最好也要判断下目录是否存在。



对于os.removedirs()方法，首先也是要判断目录是否存在，否则抛出错误。



**递归删除空目录，当递归到上级目录不是空目录时，自动停止删除，且不会报错**。

要不然，你想想，如果os.removedirs(‘D:/Python/Python36/’)

如果，Python36是空的，就不会报错，Python36是非空的，则报错。

对于Python是不是除了Python36外还有没有其他文件或目录，都无所谓，若有其他文件或目录，则就自动停止，不会删除Python目录，也不会报错。

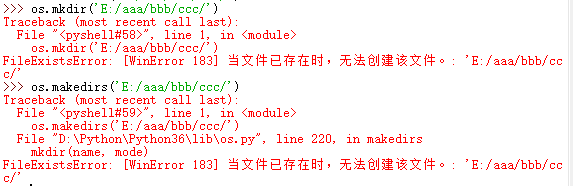
理解：迭代删除的只能是**空目录**，自动停止于**非空目录**。

## 创建目录或文件

### 创建目录：os.mkdir(dirPath)与os.makedirs(dirPath)

当目录已经存在时，调用这两个方法，都会报出错误：

所以，在调用这个方法前，需要判断目录是否存在。



dirPath = 'E:/aaa/bbb/ccc/ddd/';

if not os.path.exists(dirPath):

os.makedirs(dirPath);

### mkdir()只可以创建一级目录，否则报错。



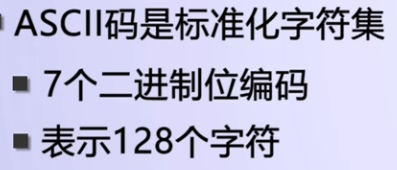


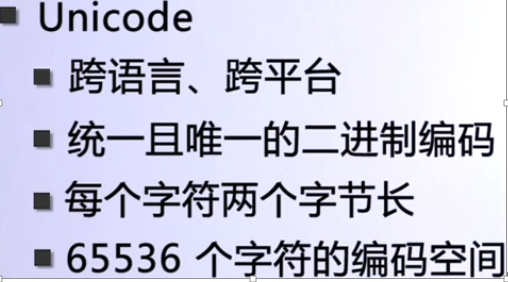
### os.makedirs()可以创建多级目录。

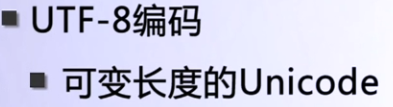
if not os.path.exists(dirPath):

os.makedirs(dirPath);

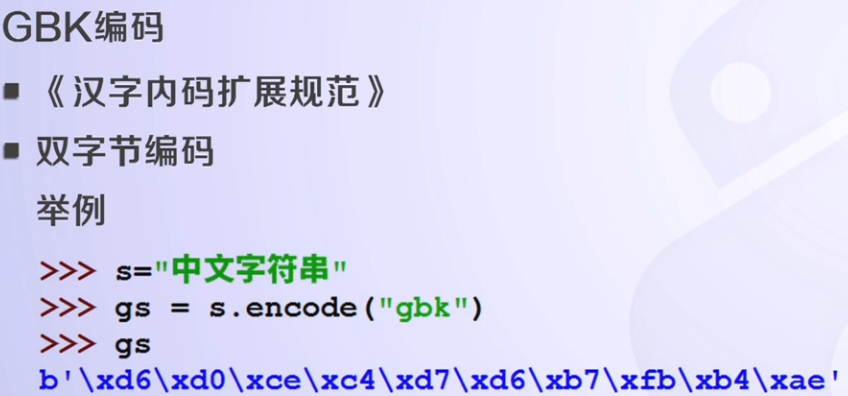
# 常用编码介绍











示例：

str1 = '中国'  
encodeStr = str1.encode('utf-8')  
print(encodeStr)#b'\xe4\xb8\xad\xe5\x9b\xbd'  
str2 = encodeStr.decode('utf-8')  
print(str2)#中国