

第八章 文件操作(IO 技术)

讲师：高 淇

一个完整的程序一般都包括数据的存储和读取，我们在前面写的程序数据都没有进行实际的存储，因此 python 解释器执行完数据就消失了。实际开发中，我们经常需要从外部存储介质（硬盘、光盘、U 盘等）读取数据，或者将程序产生的数据存储到文件中，实现“持久化”保存。

有基础的同学知道，很多软件系统是将数据存储的数据库中；数据库实际也是基于文件形式存储的，本章我们就学习文件的相关操作。

文本文件和二进制文件

按文件中数据组织形式，我们把文件分为文本文件和二进制文件两大类。

1. 文本文件

文本文件存储的是普通“字符”文本，python 默认为 unicode 字符集（两个字节表示一个字符，最多可以表示：65536 个），可以使用记事本程序打开。但是，像 word 软件编辑的文档不是文本文件。

2. 二进制文件

二进制文件把数据内容用“字节”进行存储，无法用记事本打开。必须使用专用的软件解码。常见的有：MP4 视频文件、MP3 音频文件、JPG 图片、doc 文档等等。

文件操作相关模块概述

Python 标准库中，如下是文件操作相关的模块，我们会陆续给大家介绍。

名称	说明
io 模块	文件流的输入和输出操作 input output
os 模块	基本操作系统功能，包括文件操作
glob 模块	查找符合特定规则的文件路径名
fnmatch 模块	使用模式来匹配文件路径名
fileinput 模块	处理多个输入文件
filecmp 模块	用于文件的比较
cvs 模块	用于 csv 文件处理
pickle 和 cPickle	用于序列化和反序列化
xml 包	用于 XML 数据处理
bz2、gzip、zipfile、zlib、tarfile	用于处理压缩和解压缩文件（分别对应不同的算法）

创建文件对象 open()

open()函数用于创建文件对象，基本语法格式如下：

open(文件名[,打开方式])

如果只是文件名，代表在当前目录下的文件。文件名可以录入全路径，比如：D:\a\b.txt。

为了减少 “\” 的输入，可以使用原始字符串：r “d:\b.txt”。示例如下：

```
f = open(r"d:\b.txt", "w")
```

打开方式有如下几种：

模式	描述
r	读 read 模式
w	写 write 模式。如果文件不存在则创建；如果文件存在，则重写新内容；
a	追加 append 模式。如果文件不存在则创建；如果文件存在，则在文件末尾追加内容
b	二进制 binary 模式（可与其他模式组合使用）
+	读、写模式（可与其他模式组合使用）

文本文件对象和二进制文件对象的创建：

如果我们没有增加模式 “b”，则默认创建的是文本文件对象，处理的基本单元是 “字符”。如果是二进制模式 “b”，则创建的是二进制文件对象，处理的基本单元是 “字节”。

文本文件的写入

基本的文件写入操作

文本文件的写入一般就是三个步骤：

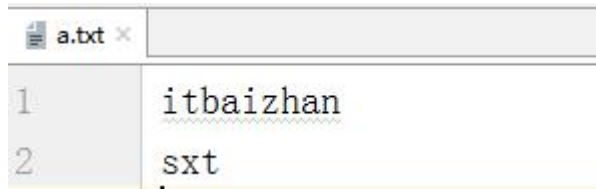
1. 创建文件对象
2. 写入数据
3. 关闭文件对象

我们首先创建一个小程序，体验一下文本文件的写入操作。

【操作】文本写入操作简单测试

```
f = open(r"a.txt","a")
s = "itbaizhan\nsxt\n"
f.write(s)
f.close()
```

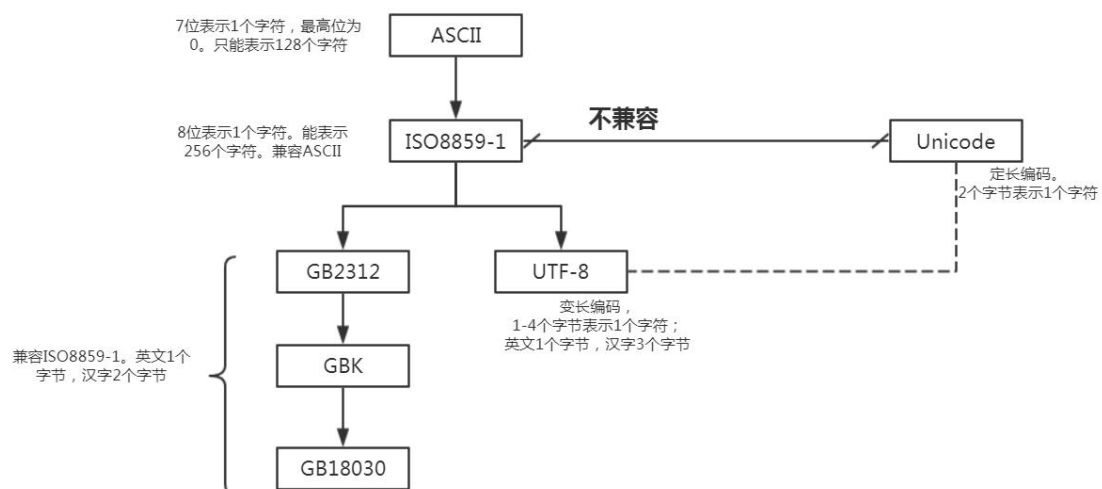
执行结果：



常用编码介绍

在操作文本文件时，经常会操作中文，这时候就经常会碰到乱码问题。为了让大家有能力解决中文乱码问题，这里简单介绍一下各种编码之间的关系。

常用编码之间的关系如下：



ASCII

全称为 American Standard Code for Information Interchange，美国信息交换标准代码，这是世界上最早最通用的单字节编码系统，主要用来显示现代英语及其他西欧语言。

ASCII 码用 7 位表示，只能表示 128 个字符。只定义了 $2^7=128$ 个字符，用 7bit 即可完全编码，而一字节 8bit 的容量是 256，所以一字节 ASCII 的编码最高位总是 0。

0 ~ 31 表示控制字符如回车、退格、删除等；32 ~ 126 表示打印字符即可以通过键盘输入并且能显示出来的字符；其中 48 ~ 57 为 0 到 9 十个阿拉伯数字，65 ~ 90 为 26 个大写英文字母，97 ~ 122 号为 26 个小写英文字母，其余为一些标点符号、运算符号等，具体可以参考 ASCII 标准表（大家自行百度，不在此赘述）。

ISO8859-1

ISO-8859-1 又称 Latin-1，是一个 8 位单字节字符集，它把 ASCII 的最高位也利用起来，并兼容了 ASCII，新增的空间是 128，但它并没有完全用完。

在 ASCII 编码之上又增加了西欧语言、希腊语、泰语、阿拉伯语、希伯来语对应的文字符号，它是向下兼容 ASCII 编码

GB2312,GBK,GB18030

•GB2312

GB2312 全称为信息交换用汉字编码字符集，是中国于 1980 年发布，主要

用于计算机系统汉字处理。GB2312 主要收录了 6763 个汉字、682 个符号。

GB2312 覆盖了汉字的大部分使用率,但不能处理像古汉语等特殊的罕用字,所以后来出现了像 GBK、GB18030 这种编码。

GB2312 完全兼容 ISO8859-1。

•GBK

全称为 Chinese Internal Code Specification ,即汉字内码扩展规范 ,于 1995 年制定。它主要是扩展了 GB2312 ,在它的基础上又加了更多的汉字 ,它一共收录了 21003 个汉字

•GB18030

现在最新的内码字集于 2000 年发布 ,并于 2001 年强制执行 ,包含了中国大部分少数民族的语言字符 ,收录汉字数超过 70000 余个。

它主要采用单字节、双字节、四字节对字符编码 ,它是向下兼容 GB2312 和 GBK 的 ,虽然是我国的强制使用标准 ,但在实际生产中很少用到 ,用得最多的反而是 GBK 和 GB2312

Unicode

Unicode 编码设计成了固定两个字节 ,所有的字符都用 16 位($2^{16}=65536$)表示 ,包括之前只占 8 位的英文字符等 ,所以会造成空间的浪费 ,UNICODE 在很长的一段时间内都没有得到推广应用。

Unicode 完全重新设计，不兼容 iso8859-1，也不兼容任何其他编码。

UTF-8

对于英文字母，unicode 也需要两个字节来表示。所以 unicode 不利于传输和存储。因此而产生了 UTF 编码，UTF-8 全称是 (8-bit Unicode Transformation Format)。

UTF 编码兼容 iso8859-1 编码，同时也可以用来表示所有语言的字符，不过，UTF 编码是不定长编码，每一个字符的长度从 1-4 个字节不等。其中，英文字母都是用一个字节表示，而汉字使用三个字节。

【老鸟建议】一般项目都会使用 UTF-8。unicode 中虽然汉字是两个字节，UTF-8 中汉字是 3 个字节。但是互联网中一个网页也包含了大量的英文字母，这些英文字母只占用 1 个字节，整体占用空间，UTF-8 仍然优于 Unicode。

中文乱码问题

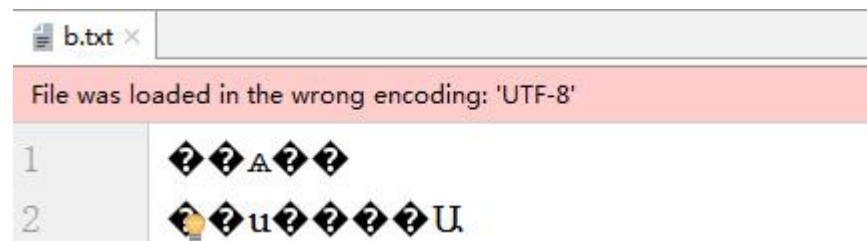
windows 操作系统默认的编码是 GBK，Linux 操作系统默认的编码是 UTF-8。当我们用 open() 时，调用的是操作系统打开的文件，默认的编码是 GBK。

【示例】中文字符文件，乱码出现测试

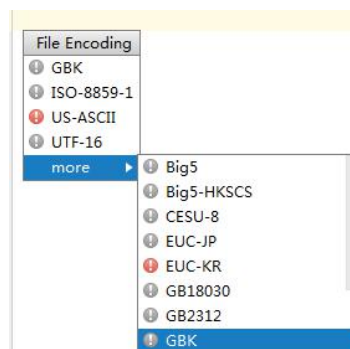
```
#测试写入中文  
f = open(r"b.txt", "w")
```

```
f.write("尚学堂\n 百战程序员\n")  
  
f.close()
```

运行结果（Linux 环境中不存在这个问题）：



我们在文件编辑区单击右键，选择 FileEncoding，选择 GBK 即可：



再选择 Reload，文件即显示正常。



【示例】通过指定文件编码解决中文乱码问题

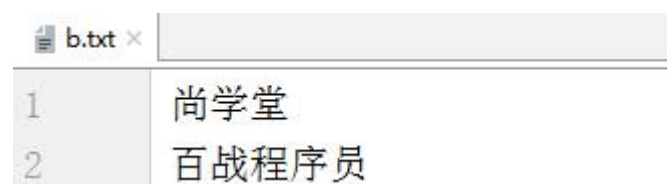

```
#测试写入中文

f = open(r"b.txt","w",encoding="utf-8")

f.write("尚学堂\n 百战程序员\n")

f.close()
```

运行结果：



	b.txt
1	尚学堂
2	百战程序员

write()/writelines()写入数据

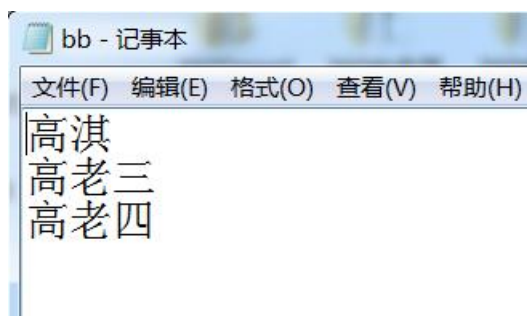
write(a)：把字符串 a 写入到文件中

writelines(b)：把字符串列表写入文件中，不添加换行符

【操作】添加字符串列表数据到文件中

```
f = open(r"d:\bb.txt","w",encoding="utf-8")
s = ["高淇\n","高老三\n","高老四\n"]
f.writelines(s)
f.close()
```

执行结果：



close()关闭文件流

由于文件底层是由操作系统控制，所以我们打开的文件对象必须显式调用 `close()` 方法关闭文件对象。当调用 `close()` 方法时，首先会把缓冲区数据写入文件(也可以直接调用 `flush()` 方法)，再关闭文件，释放文件对象。

为了确保打开的文件对象正常关闭，一般结合异常机制的 `finally` 或者 `with` 关键字实现无论何种情况都能关闭打开的文件对象。

【操作】结合异常机制 `finally` 确保关闭文件对象

```
try:

    f = open(r"my01.txt", "a")

    str = "gaoqi"

    f.write(str)

except BaseException as e:

    print(e)

finally:
```

```
f.close()
```

with 语句(上下文管理器)

with 关键字（上下文管理器）可以自动管理上下文资源，不论什么原因跳出 with 块，都能确保文件正确的关闭，并且可以在代码块执行完毕后自动还原进入该代码块时的现场。

【操作】使用 with 管理文件写入操作

```
s = ["高淇\n", "高三\n", "高考五\n"]  
with open(r"d:\bb.txt", "w") as f:  
    f.writelines(s)
```

文本文件的读取

文件的读取一般使用如下三个方法：

1. read([size])

从文件中读取 size 个字符，并作为结果返回。如果没有 size 参数，则读取整个文件。

读取到文件末尾，会返回空字符串。

2. readline()

读取一行内容作为结果返回。读取到文件末尾，会返回空字符串。

3. readlines()

文本文件中，每一行作为一个字符串存入列表中，返回该列表

【操作】 读取一个文件前 4 个字符

```
with open(r"bb","r",encoding="utf-8") as f:
    print(f.read(4))
```

【操作】文件较小，一次将文件内容读入到程序中

```
with open(r"d:\bb.txt","r") as f:
    print(f.read())
```

【操作】按行读取一个文件

```
with open(r"bb.txt","r") as f:
    while True:
        fragment = f.readline()
        if not fragment:
            break
        else:
            print(fragment,end="")
```

【操作】使用迭代器（每次返回一行）读取文本文件

```
with open(r"d:\bb.txt","r") as f:
    for a in f:
        print(a,end="")
```

【操作】为文本文件每一行的末尾增加行号

```
with open("e.txt","r",encoding="utf-8") as f:
    lines = f.readlines()
```

```
lines = [ line.rstrip()+" #" +str(index+1)+"\n" for index,line in
enumerate(lines)] #推导式生成列表

with open("e.txt","w",encoding="utf-8") as f:

    f.writelines(lines)
```

执行前文件内容：

我 love u!

尚学堂

百战程序员

执行程序后文件内容：

我 love u! #1

尚学堂 #2

百战程序员 #3

二进制文件的读取和写入

二进制文件的处理流程和文本文件流程一致。首先还是要创建文件对象，不过，我们需要指定二进制模式，从而创建出二进制文件对象。例如：

```
f = open(r"d:\a.txt", 'wb')    #可写的、重写模式的二进制文件对象

f = open(r"d:\a.txt", 'ab')    #可写的、追加模式的二进制文件对象

f = open(r"d:\a.txt", 'rb')    #可读的二进制文件对象
```

创建好二进制文件对象后，仍然可以使用 write()、read()实现文件的读写操作。

【操作】 读取图片文件，实现文件的拷贝

```
with open('aa.gif', 'rb') as f:

    with open('aa_copy.gif', 'wb') as w:

        for line in f.readlines():

            w.write(line)

print('图片拷贝完成！')
```

文件对象的常用属性和方法

文件对象封装了文件相关的操作。在前面我们学习了通过文件对象对文件进行读写操作。本节我们详细列出文件对象的常用属性和方法，并进行说明。

文件对象的属性

属性	说明
----	----

name	返回文件的名字
mode	返回文件的打开模式
closed	若文件被关闭则返回 True

文件对象的打开模式

模式	说明
r	读模式
w	写模式
a	追加模式
b	二进制模式（可与其他模式组合）
+	读写模式（可以其他模式组合）

文件对象的常用方法

方法名	说明
read([size])	从文件中读取 size 个字节或字符的内容返回。若省略[size]，则读取到文件末尾，即一次读取文件所有内容
readline()	从文本文件中读取一行内容
readlines()	把文本文件中每一行都作为独立的字符串对象，并将这些对象放入列表返回
write(str)	将字符串 str 内容写入文件
writelines(s)	将字符串列表 s 写入文件文件，不添加换行符
seek(offset	把文件指针移动到新的位置，offset 表示相对于 whence 的多少个

[whence])	字节的偏移量； offset : off 为正往结束方向移动，为负往开始方向移动 whence 不同的值代表不同含义： 0: 从文件头开始计算（默认值） 1：从当前位置开始计算 2：从文件尾开始计算
tell()	返回文件指针的当前位置
truncate([size])	不论指针在什么位置，只留下指针前 size 个字节的内容，其余全部删除； 如果没有传入 size，则当指针当前位置到文件末尾内容全部删除
flush()	把缓冲区的内容写入文件，但不关闭文件
close()	把缓冲区内容写入文件，同时关闭文件，释放文件对象相关资源

文件任意位置操作

【示例】seek()移动文件指针示例

```
with open("e.txt", "r", encoding="utf-8") as f:  
    print("文件名是：{0}".format(f.name))  
    print(f.tell())
```



```
print("读取的内容 : {0}".format(str(f.readline())))\n\nprint(f.tell())\n\nf.seek(0,0)\n\nprint("读取的内容 : {0}".format(str(f.readline())))
```

使用 pickle 序列化

Python 中，一切皆对象，对象本质上就是一个“存储数据的内存块”。有时候，我们需要将“内存块的数据”保存到硬盘上，或者通过网络传输到其他的计算机上。这时候，就需要“对象的序列化和反序列化”。对象的序列化机制广泛的应用在分布式、并行系统上。

序列化指的是：将对象转化成“串行化”数据形式，存储到硬盘或通过网络传输到其他地方。反序列化是指相反的过程，将读取到的“串行化数据”转化成对象。

我们可以使用 pickle 模块中的函数，实现序列化和反序列操作。

序列化我们使用：

`pickle.dump(obj, file)` `obj` 就是要被序列化的对象，`file` 指的是存储的文件

`pickle.load(file)` 从 `file` 读取数据，反序列化成对象

【操作】将对象序列化到文件中

```
import pickle\nwith open(r"d:\data.dat", "wb") as f:
```

```
a1 = "高淇"
a2 = 234
a3 = [20,30,40]

pickle.dump(a1,f)
pickle.dump(a2, f)
pickle.dump(a3, f)
```

【操作】将获得的数据反序列化对象

```
import pickle
with open(r"d:\data.dat","rb") as f:
    a1 = pickle.load(f)
    a2 = pickle.load(f)
    a3 = pickle.load(f)
    print(a1)
    print(a2)
    print(a3)
```

执行结果：

高淇

234

[20, 30, 40]

CSV 文件的操作

csv(Comma Separated Values)是逗号分隔符文本格式，常用于数据交换、Excel

文件和数据库数据的导入和导出。与 Excel 文件不同，CSV 文件中：

值没有类型，所有值都是字符串

不能指定字体颜色等样式

不能指定单元格的宽高，不能合并单元格

没有多个工作表

不能嵌入图像图表

Python 标准库的模块 csv 提供了读取和写入 csv 格式文件的对象。

我们在 excel 中建立一个简单的表格：

姓名	年龄	工作	薪水
高淇	18	程序员	50000
高老三	19	测试工程师	20000
高老五	20	人工智能开发	50000

另存为"csv(逗号分隔)"，我们打开查看这个 csv 文件内容：

姓名,年龄,工作,薪水

高淇,18,程序员,50000

高老三,19,测试工程师,20000

高老五,20,人工智能开发,50000

csv.reader 对象和 csv 文件读取

【操作】csv.reader 对象用于从 csv 文件读取数据

```
import csv
with open(r"d:\a.csv") as a:
    a_csv = csv.reader(a)      #创建 csv 对象,它是一个包含所有数据的列表,每一行为一个元素
    headers = next(a_csv)     #获得列表对象,包含标题行的信息
    print(headers)
    for row in a_csv:         #循环打印各行内容
        print(row)
```

执行结果：

['姓名', '年龄', '工作', '薪水']

```
['高淇', '18', '程序员', '50000']
```

```
['高老三', '19', '测试工程师', '20000']
```

```
['高老五', '20', '人工智能开发', '50000']
```

csv.writer 对象和 csv 文件写入

【操作】csv.writer 对象写一个 csv 文件

```
import csv

headers = ["工号", "姓名", "年龄", "地址", "月薪"]
rows = [("1001", "高淇", 18, "西三旗 1 号院", "50000"), ("1002", "高八", 19, "西三旗 1 号院", "30000")]

with open(r"d:\b.csv", "w") as b:
    b_csv = csv.writer(b)      #创建 csv 对象
    b_csv.writerow(headers)    #写入一行 (标题)
    b_csv.writerows(rows)      #写入多行 (数据)
```

执行结果：

工号,姓名,年龄,地址,月薪

1001,高淇,18,西三旗 1 号院,50000

1002,高八,19,西三旗 1 号院,30000

os 和 os.path 模块

os 模块可以帮助我们直接对操作系统进行操作。我们可以直接调用操作系统的可执行文件、命令，直接操作文件、目录等等。在系统运维的核心基础。

os 模块-调用操作系统命令

•os.system 可以帮助我们直接调用系统的命令

【示例】os.system 调用 windows 系统的记事本程序

```
import os  
  
os.system("notepad.exe")
```

【示例】os.system 调用 windows 系统中 ping 命令

```
import os  
  
os.system("ping www.baidu.com")
```

运行结果：

正在 Ping www.a.shifen.com [111.206.223.206] 具有 32 字节的数据:

来自 111.206.223.206 的回复: 字节=32 时间=9ms TTL=56

来自 111.206.223.206 的回复: 字节=32 时间=7ms TTL=56

来自 111.206.223.206 的回复: 字节=32 时间=6ms TTL=56

来自 111.206.223.206 的回复: 字节=32 时间=9ms TTL=56

111.206.223.206 的 Ping 统计信息:

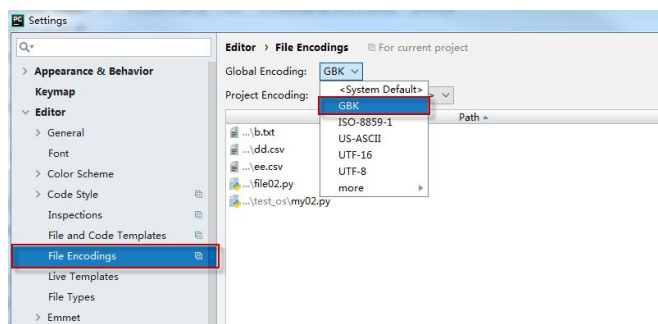
数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),

往返行程的估计时间(以毫秒为单位):

最短 = 6ms, 最长 = 9ms, 平均 = 7ms

【注】Linux 是命令行操作更容易, 我们可以通过 os.system 可以更加容易的调用相关的命令;

【注】控制台输出中文可能会有乱码问题, 可以在 file-->setting 中设置:



•os.startfile：直接调用可执行文件

【示例】运行安装好的微信

```
import os  
  
os.startfile(r"C:\Program Files (x86)\Tencent\WeChat\WeChat.exe")
```

运行结果：



os 模块-文件和目录操作

我们可以通过前面讲的文件对象实现对于文件内容的读写操作。如果，还需要对文件和

目录做其他操作，可以使用 `os` 和 `os.path` 模块。

`os` 模块下常用操作文件的方法

方法名	描述
<code>remove(path)</code>	删除指定的文件
<code>rename(src,dest)</code>	重命名文件或目录
<code>stat(path)</code>	返回文件的所有属性
<code>listdir(path)</code>	返回 <code>path</code> 目录下的文件和目录列表

`os` 模块下关于目录操作的相关方法，汇总如下：

方法名	描述
<code>mkdir(path)</code>	创建目录
<code>makedirs(path1/path2/path3/...)</code>	创建多级目录
<code>rmdir(path)</code>	删除目录
<code>removedirs(path1/path2...)</code>	删除多级目录
<code>getcwd()</code>	返回当前工作目录：current work dir
<code>chdir(path)</code>	把 <code>path</code> 设为当前工作目录
<code>walk()</code>	遍历目录树
<code>sep</code>	当前操作系统所使用的路径分隔符

【示例】os 模块：创建、删除目录、获取文件信息等

```
#coding=utf-8

#测试 os 模块中，关于文件和目录的操作

import os

#####获取文件和文件夹相关的信息#####

print(os.name)    #windows->nt    linux 和 unix->posix

print(os.sep)     #windows->\    linux 和 unix->/

print(repr(os.linesep))  #windows->\r\n    linux-->\n

print(os.stat("my02.py"))

#####关于工作目录的操作#####

#print(os.getcwd())

#os.chdir("d:")    #改变当前的工作目录为：d:盘根目录

#os.mkdir("书籍")

#####创建目录、创建多级目录、删除#####

#os.mkdir("书籍")

#os.rmdir("书籍")  #相对路径都是相对于当前的工作目录
```



```
#os.makedirs("电影/港台/周星驰")

#os.removedirs("电影/港台/周星驰")    #只能删除空目录

#os.makedirs("../音乐/香港/刘德华")    #../指的是上一级目录

#os.rename("电影","movie")

dirs = os.listdir("movie")

print(dirs)

#####
```

os.path 模块

os.path 模块提供了目录相关（路径判断、路径切分、路径连接、文件夹遍历）的操作

方法	描述
isabs(path)	判断 path 是否绝对路径
isdir(path)	判断 path 是否为目录
isfile(path)	判断 path 是否为文件

exists(path)	判断指定路径的文件是否存在
getsize(filename)	返回文件的大小
abspath(path)	返回绝对路径
dirname(p)	返回目录的路径
getatime(filename)	返回文件的最后访问时间
getmtime(filename)	返回文件的最后修改时间
walk(top,func,arg)	递归方式遍历目录
join(path,*paths)	连接多个 path
split(path)	对路径进行分割，以列表形式返回
splittext(path)	从路径中分割文件的扩展名

【示例】测试 os.path 中常用方法

```
#测试 os.path 常用方法

import os

import os.path

#####获得目录、文件基本信息#####

print(os.path.isabs("d:/a.txt"))    #是否绝对路径

print(os.path.isdir("d:/a.txt"))    #是否目录

print(os.path.isfile("d:/a.txt"))   #是否文件

print(os.path.exists("a.txt"))      #文件是否存在
```

```
print(os.path.getsize("a.txt"))    #文件大小

print(os.path.abspath("a.txt"))    #输出绝对路径

print(os.path.dirname("d:/a.txt"))  #输出所在目录


#####获得创建时间、访问时间、最后修改时间#####

print(os.path.getctime("a.txt"))    #返回创建时间

print(os.path.getatime("a.txt"))    #返回最后访问时间

print(os.path.getmtime("a.txt"))    #返回最后修改时间


#####对路径进行分割、连接操作#####

path = os.path.abspath("a.txt")     #返回绝对路径

print(os.path.split(path))          #返回元组：目录、文件

('C:\\Users\\Administrator\\PycharmProjects\\mypro_io\\test_os', 'a.txt')

print(os.path.splitext(path))       #返回元组：路径、扩展名

('C:\\Users\\Administrator\\PycharmProjects\\mypro_io\\test_os\\a',
'.txt')

print(os.path.join("aa", "bb", "cc")) #返回路径：aa/bb/cc
```

【示例】列出指定目录下所有的.py 文件，并输出文件名

```
#coding=utf-8

#列出指定目录下所有的.py 文件，并输出文件名

import os

import os.path

path = os.getcwd()

file_list = os.listdir(path) #列出子目录和子文件

for filename in file_list:

    pos = filename.rfind(".")

    if filename[pos+1:]=="py":

        print(filename,end="\t")

print("#####")

file_list2 = [filename for filename in os.listdir(path) if

filename.endswith(".py")]

for filename in file_list2:

    print(filename,end="\t")
```

walk()递归遍历所有文件和目录

os.walk()方法：

返回一个 3 个元素的元组 , (dirpath, dirnames, filenames),

dirpath : 要列出指定目录的路径

dirnames : 目录下的所有文件夹

filenames : 目录下的所有文件

【示例】使用 walk()递归遍历所有文件和目录

```
#coding=utf-8

import os

all_files = []

path = os.getcwd()
list_files = os.walk(path)

for dirpath,dirnames,filenames in list_files:

    for dir in dirnames:

        all_files.append(os.path.join(dirpath,dir))

    for name in filenames:

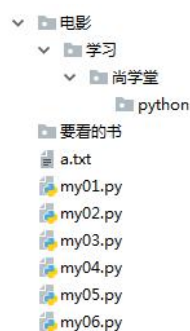
        all_files.append(os.path.join(dirpath,name))

#打印子目录和子文件
```

```
for file in all_files:
```

```
    print(file)
```

目录结构是：



运行结果：

```
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\要看的书
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\a.txt
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my01.py
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my02.py
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my03.py
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my04.py
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my05.py
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my06.py
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影\学习
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影\学习\尚学堂
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影\学习\尚学堂\python
```

shutil 模块(拷贝和压缩)

shutil 模块是 python 标准库中提供的，主要用来做文件和文件夹的拷贝、移动、删除等；还可以做文件和文件夹的压缩、解压缩操作。

os 模块提供了对目录或文件的一般操作。shutil 模块作为补充，提供了移动、复制、压缩、解压等操作，这些 os 模块都没有提供。

【示例】实现文件的拷贝

```
import shutil
```

```
#copy 文件内容
```

```
shutil.copyfile("1.txt", "1_copy.txt")
```

【示例】实现递归的拷贝文件夹内容(使用 shutil 模块)

```
import shutil
```

```
#"音乐"文件夹不存在才能用。
```

```
shutil.copytree("电影/学习", "音乐
```

```
", ignore=shutil.ignore_patterns("*.html", "*.htm"))
```

将文件夹“电影/学习”下面的内容拷贝到文件夹“音乐”下。拷贝时忽略所有的 html 和 htm 文件。运行结果如下：



【示例】实现将文件夹所有内容压缩(使用 shutil 模块)

```
import shutil
```

```
import zipfile
```

```
#将"电影/学习"文件夹下所有内容压缩到"音乐 2"文件夹下生成 movie.zip
```

```
#shutil.make_archive("音乐 2/movie", "zip", "电影/学习")
```

```
#压缩:将指定的多个文件压缩到一个 zip 文件
```

```
#z = zipfile.ZipFile("a.zip", "w")

#z.write("1.txt")

#z.write("2.txt")

#z.close()
```

【示例】实现将压缩包解压缩到指定文件夹(使用 shutil 模块)

```
import shutil

import zipfile

#解压缩：

z2 = zipfile.ZipFile("a.zip", "r")

z2.extractall("d:/") #设置解压的地址

z2.close()
```

递归算法

递归是一种常见的解决问题的方法，即把问题逐渐简单化。递归的基本思想就是“**自己调用自己**”，一个使用递归技术的方法将会直接或者间接的调用自己。

利用递归可以用简单的程序来解决一些复杂的问题。比如：斐波那契数列的计算、汉诺塔、快排等问题。

递归结构包括两个部分：

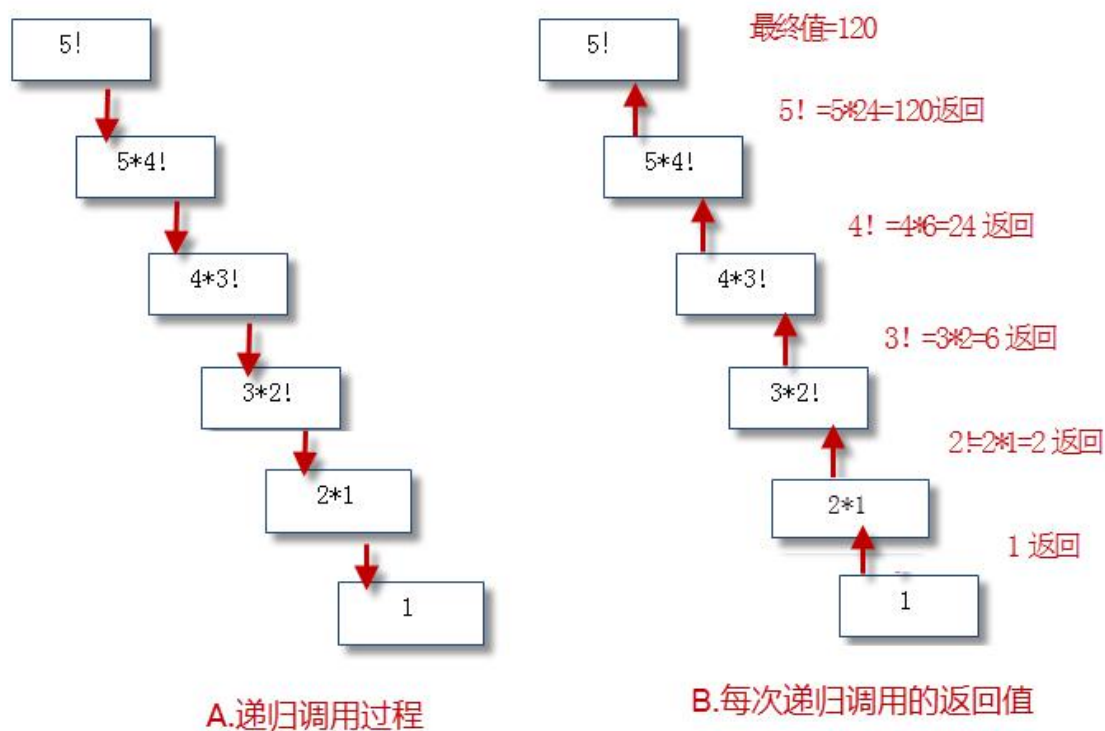
- **定义递归头。**解答：什么时候不调用自身方法。如果没有头，将陷入死循环，也就是递归的结束条件。

□ **递归体**。解答：什么时候需要调用自身方法。

【示例 3-22】使用递归求 n!

```
#coding=utf-8  
  
#测试递归  
  
def factorial(n):  
    if n==1:  
        return 1  
    else:  
        return n*factorial(n-1)  
  
a = factorial(10)  
print(a)
```

执行过程如图所示：



递归原理分析图

递归的缺陷

简单的程序是递归的优点之一。但是递归调用会占用大量的系统堆栈，内存耗用多，在递归调用层次多时速度要比循环慢的多，所以在使用递归时要慎重。

【示例】使用递归算法遍历目录下所有文件

```
import os

allfile = []

def getFiles(path, level):

    childFiles = os.listdir(path)

    for file in childFiles:
```

```
filepath = os.path.join(path,file)

if os.path.isdir(filepath):

    getFiles(filepath,level+1)

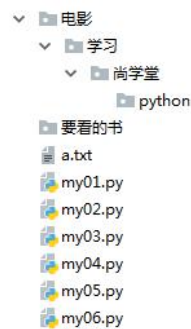
allfile.append("\t"*level+filepath)


getFiles(os.getcwd(),0)


for f in reversed(allfile):

    print(f)
```

目录结构：



运行结果：

```
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\要看的书
C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影
  C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影\学习
    C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影\学习\尚学堂
      C:\Users\Administrator\PycharmProjects\mypro_io\test_os\电影\学习\尚学堂
        \python
          C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my07.py
          C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my06.py
          C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my05.py
```

C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my04.py

C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my03.py

C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my02.py

C:\Users\Administrator\PycharmProjects\mypro_io\test_os\my01.py

C:\Users\Administrator\PycharmProjects\mypro_io\test_os\a.txt