

第五章文件处理

本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务端进行重写。学完之后，你所收获的不仅仅是精通一门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

一 文件读写基本操作

1. 文件操作的工作流程

文件在我们的计算机上随处可见，当我们需要永久保存数据的时候就会用到文件，文件是由计算机操作系统来提供的，那么自然也就受操作系统的控制。如下图所示，一套完整的计算机系统主要由三部分构成：

1. 应用程序
2. 操作系统
3. 计算机底层硬件

capable of executing. The rest of the software runs in **user mode**, in which only a subset of the machine instructions is available. In particular, those instructions that affect control of the machine or do **I/O** (Input/Output) are forbidden to user-mode programs. We will come back to the difference between kernel mode and user mode repeatedly throughout this book. It plays a crucial role in how operating systems work.

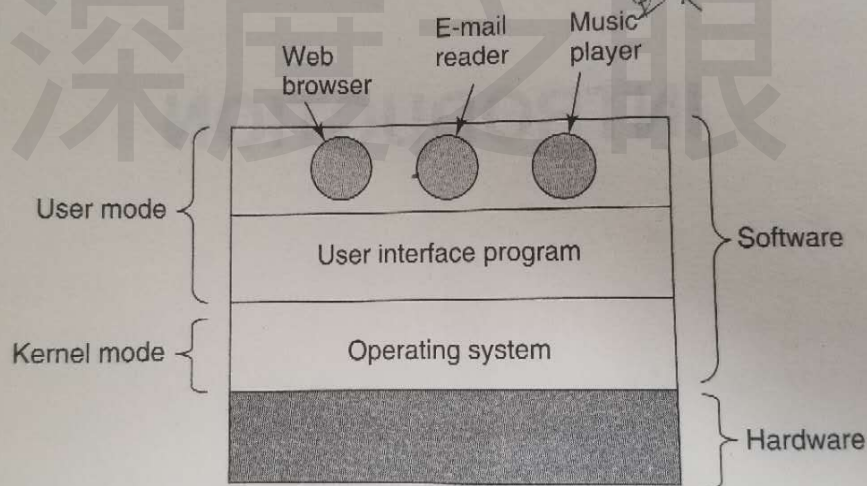


Figure 1-1. Where the operating system fits in.

The user interface program, shell or GUI, is the lowest level of user-mode software, and allows the user to start other programs, such as a Web browser, email reader, or music player. These programs too make use of the operating system.

如果应用程序需要操作硬件，必须先要发指令给操作系统，通过操作系统来帮应用程序完成对机器硬件的操作。如果应用程序需要把自己产生的数据永久保存起来（应用程序产生的数据原本是在内存中），那么就是把它写入硬盘，这时应用程序要通过操作系统提供的接口来控制硬件，如果应用程序需要读取文件内容，同样是向操作系统发起请求，最后由操作系统返回文件内容，这之间的过程如下图所示：

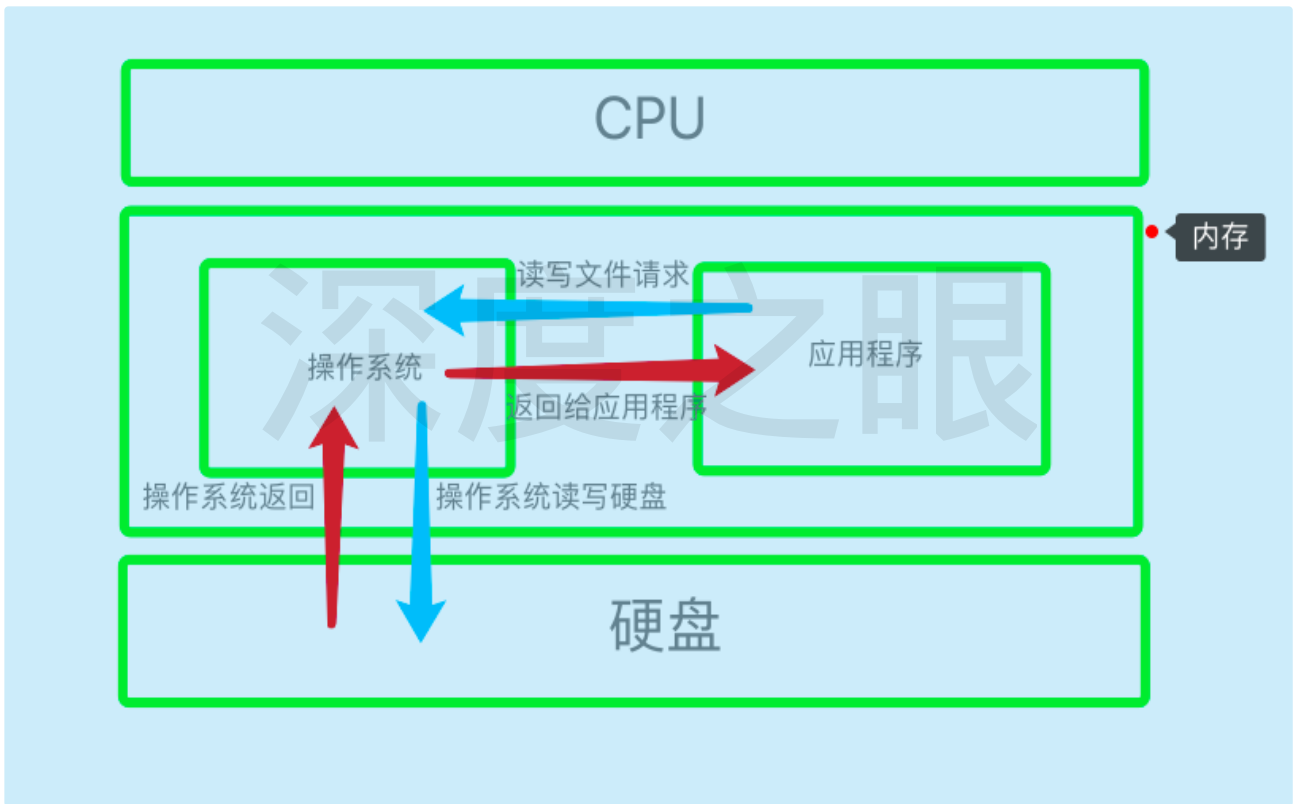
深度之眼

作者：马一特

作者：马一特

马一特

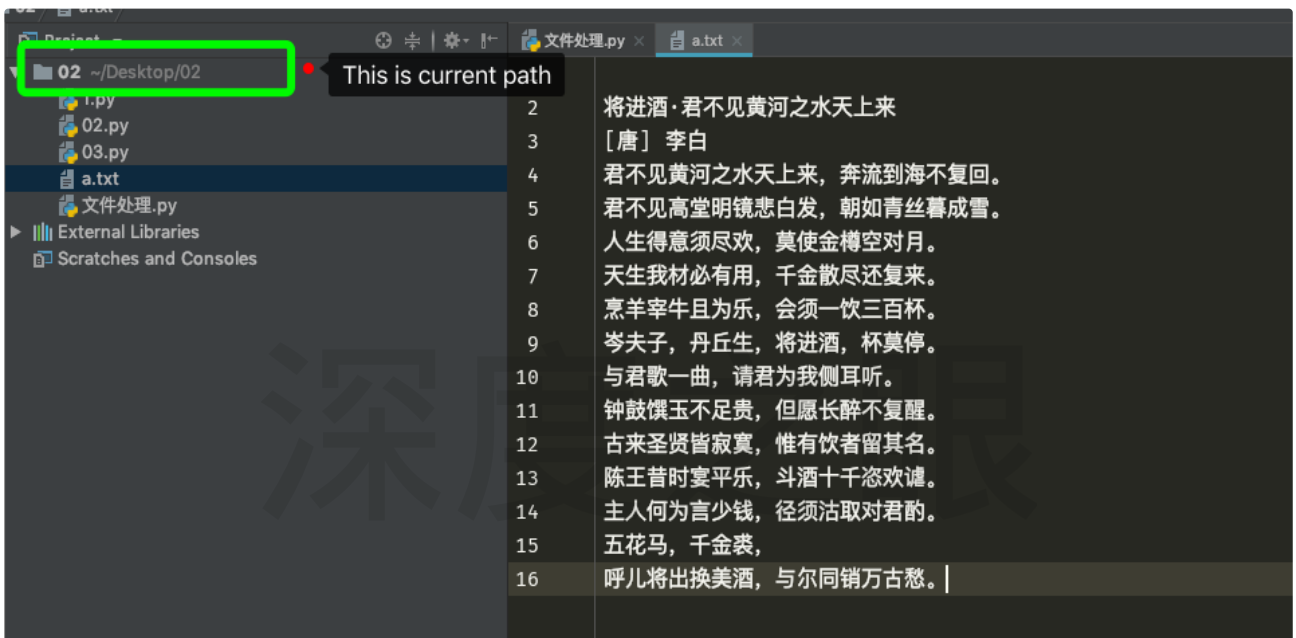
马一特



接下来我们要讲解的就是应用程序如何调用操作系统提供的接口来读取和写入文件。

2. 文件操作的基本形式

我们事先在当前路径下准备好一个文件a.txt，如下图所示：



接下来我们在 "文件处理.py" 文件内开始写打开读取文件的操作，在Python中必然会有一个功能或者接口来打开文件，这个接口就是open，使用参数如下图所示：

```

2  """
3  open打开文件需要3个参数，
4  打开文件之后会有一个返回值，
5  读写操作就是对这个返回值进行操作
6  f = open('文件的路径',mode='打开文件的模式',encoding='操作文件的字符编码')
7  """
8  # a.txt不加任何前缀，默认在当前路径下，"r"模式是只读模式
9  # 如果打开的文件路径是绝对路径，就会有"\"，右斜杠有转译的意思，你会看到颜色代码颜色已经发生变化
10 f = open('C:\n\c\d\a.txt',mode='r',encoding='utf-8')
11
12 # 这时只需要在路径前面加一个"r"，就不会转译，"r"全称是raw string，指的是原生字符串
13 fx = open(r'C:\n\c\d\a.txt',mode='r',encoding='utf-8')
14

```

文件读取操作代码示例如下：

```

"""
open打开文件需要3个参数，
打开文件之后会有一个返回值，
读写操作就是对这个返回值进行操作
f = open('文件的路径',mode='打开文件的模式',encoding='操作文件的字符编码')
"""

"""
open是向操作系统发请求，会占用操作系统资源，这个资源不会自动回收
返回值就是应用程序拿到的变量，应用程序的变量Python解释器会自动帮你回收
"""

# 1 打开文件
f = open(r'a.txt',mode='r',encoding='utf-8')# 关于open()函数中的mode参数可参见二.2
"""
对于应用程序来说上面这行代码与我写一个"f = 1"没有本质区别，
你不需要再执行"del f"，因为解释器会自动帮你清理这个应用程序资源，
但是，打开的文件占用了操作系统的资源，这不会自动回收，所以必须人为显式的进行回收，
f.close()即可，注意其中del f一定要发生在f.close()之后，否则就会导致操作系统打开的文件还没有关
白白占用资源
"""

# 2 读取文件
data = f.read()
print(data)

# 3 关闭文件，清理操作系统打开文件的资源
f.close()

```

```
print(f) # 应用程序的资源还在
# f.read() # 文件关闭不能再进行读取
```

打开一个文件其实是占用了两部分资源，分别是操作系统资源和应用程序资源，应用程序资源会自动由Python解释器来回收，而操作系统打开文件的资源并不会立即回收，操作系统每打开一个文件其实会有一个编号，每个编号与应用程序向操作系统发起文件操作请求的编号一一对应，这个编号称为文件描述符，操作系统的文件描述符编号是有限的，所以，当服务器高并发的时候，由于打开文件个数非常多，因而还没来得及关闭，那么服务器就卡了，返回给用的的结果就是用户的客户端卡了。

有的时候关闭文件的操作总是会被遗忘，我们有一个使用 "with" 来操作文件的方式，它是一个上下文的操作，会帮你自动的关闭文件，代码示例如下：

```
# as 指的是赋值
with open('a.txt', 'r', encoding='utf-8') as f:
    data = f.read()
    print(data)
```

除此之外，“with”可以连续打开多个文件，代码示例如下：

```
with open('a1.txt', 'r', encoding='utf-8') as f1, \
    open('a2.txt', 'r', encoding='utf-8') as f2:
    data1 = f1.read()
    data2 = f2.read()
```

二 默认打开文件的参数说明

1. 文件打开的字符编码

如果不指定字符编码，默认打开文件的字符编码与操作系统相匹配：

- Windows系统（中国大陆用户）：gbk
- Linux系统：utf-8
- MacOS：utf-8

在不指定字符编码的情况下，MacOS系统示例代码：

```
# 文件保存的以utf-8编码保存，与本机默认编码一致
with open('a.txt', 'r', ) as f:
    data = f.read()
    print(data)
```

2. 文件的打开模式

文件默认的打开模式是“t”模式，指的是文本模式，这意味着在该模式下无法打开图片，视频和音频等文件，因为这些是以二进制格式存储的，文本模式是以字符形式存储的。

操作文件的模式有三种，分别是：“r”，“w”和“a”模式，“r”是只读，“w”是只写，“a”是指追加，默认操作文件大模式是“r”模式，所以默认文件的打开模式是“rt”模式，对于操作文本文件，“t”模式必须与操作文件的三种模式连用，很多时候你看到的，这个“t”经常会省略不写，这是可以的。

三 文本模式打开文件的操作

在文件处理中，我们需要打开一个文件，打开文件有若干种方式，有时我们只希望读取文件内容而不想修改，那么为了防止失误性的修改文件，可使用'r'模式打开文件，这样你就不能对文件进行任何修改啦，每一种模式都有其存在的意义，大家用多了就明白了。

1. 操作文件“r”模式

全部读取使用read，代码示例如下：

```
f = open('a.txt', mode='r', encoding='utf-8') # “r”模式下，如果文件不存在会报错
# f.write('哈哈') #抛出异常，不能写
print(f.readable()) # 判断是否可读
print('=====>1')
print(f.read()) # 全部读取
print('=====>2')
# 读文件会有一个光标移动，第一次读完了，光标移至末尾，第二次读无内容
print(f.read())
f.close()
```

一行一行读文件内容使用readline，代码示例如下：

```
f = open('a.txt', mode='r', encoding='utf-8')
# readline指的是一行一行读文件
print(f.readline(), end='') # 文件中有换行，print也自带换行，指定end参数去掉默认换行
print(f.readline(), end='')
print(f.readline(), end='')
f.close()
```

全部读取文件内容，存入列表，每行内容为列表的一个元素使用readlines，代码示例如下：

```
f = open('a.txt', mode='r', encoding='utf-8')

print(f.readlines())
f.close()
```

readlines可以加数字作为参数，但是他不是指的行数，而是字节数，所以我们一般不用，如需逐行打印文件内容常用readlines与for循环连用，代码示例如下：

```
# 如果文件内容比较少的时候，以下两种方式都可以

with open('a.txt') as f:
    # 当文件很大时，f.readlines()结果是一个很大的列表在内存中，机器就卡了
    for line in f.readlines():
        print(line, end='')

# 推荐使用这种方式
with open('a.txt') as f: # f是一个可迭代对象，就像老母鸡会下蛋一样
    for line in f:
        # 文件内容很大时，使用这种方式每次内存中只有一行内容
        print(line, end='')
```

2. 操作文件“w”模式

注意：在“w”只写模式下，当文件存在时，就会清空该文件，代码示例如下：


```
f = open(r'a.txt', mode='w', encoding='utf-8') # 默认是wt
f.write('第一行\n') # 需要自己添加“\n”来换行
f.write('第二行\n')
f.close()
```

当文件不存在时，就会创建空文档，代码示例如下：

```
f = open(r'a1.txt', mode='w', encoding='utf-8') # 默认是wt
f.write('第一行\n')
f.write('第二行\n')
f.close()
```

只写模式常用的方法：

```
f = open(r'a1.txt', mode='w', encoding='utf-8') # 默认是wt

f.writable()
# writelines指的是可以放一个列表或者元组，里面可以有多行内容，需要自己加换行符
f.writelines(['111111\n', '222222\n', '333333\n'])
# 下面这样代码与上面写的结果一样
# f.write('aaaaaa\nbbbbbbb\ncccccc\n')
f.close()
```

3. 操作文件“a”模式

“a”模式指的是只追加写，当文件不存在时，创建空文件；当文件存在时，光标直接移至文件末尾，所以，我们在记录日志的时候都会使用“a”模式，代码示例如下：

```
f = open('access.log', mode='a', encoding='utf-8')
print(f.writable())
print(f.readable())
f.write('555555555555\n')
f.close()
```


四 二进制模式打开文件的操作

1. “b” 模式基本介绍

“b” 模式指的是文件打开的模式为“b” 模式，它与“t” 模式类似，不能单独使用，必须以“rb”，“wb” 或者“ab” 模式来使用，“b” 模式读写都是以bytes为单位进行的，所以可以理解为“b” 模式就是二进制模式。对于普通文本来说是以字符的形式保存的，但是对于图片，视频或者音频等等这些文件则是以二进制形式保存的，所以“t” 模式无法读取，代码及报错示例如下：

```
119     with open('01.jpg', 'r', encoding='utf-8') as f:
120         f.read()
121
/Users/albert/anaconda3/bin/python /Users/albert/Desktop/02/文件处理.py
Traceback (most recent call last):
  File "/Users/albert/Desktop/02/文件处理.py", line 120, in <module>
    f.read()
  File "/Users/albert/anaconda3/lib/python3.7/codecs.py", line 322, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start byte
```

图片文件并不是以字符编码保存的，而是以JPG这个格式保存成了二进制形式，与字符编码没有关系，所以我们以文本模式处理文件是不可行的。应该以二进制模式打开文件，这时不需要指定字符编码，正确的打开方式请看如下代码示例：

```
# b模式下一定不能指定encoding参数
with open('01.jpg', 'rb', ) as f:
    data = f.read()
    print(data)
```

强调：

- 1、与t模式类似，不能单独使用，必须是rb，wb，ab
- 2、b模式下读写都是以bytes单位的
- 3、b模式下一定不能指定encoding参数，不指定encoding参数默认为二进制

4、b模式（二进制）可以读取文字，图片，视频，什么文件都可以（因为所有类型的数据都要以二进制形式存在硬盘中）

2. 操作文件的“rb”模式

需要说明的一点是，“b”模式也可以读取文本文件，字符的底层都是以二进制形式存储的，只不过你在使用“t”模式读取文本文件的时候open帮你把二进制转成了能够看懂的文本，这是“t”模式的便利之处，但是它有局限性，只能操作文本文件，而“b”模式具有统一性，任何文件底层存储原理都是二进制，这也就是意味着“b”模式可以操作任何文件，代码示例如下：

```
with open('01.jpg', 'rb', ) as f1, open('a.txt', 'rb') as f2:
    img = f1.read()
    text = f2.read()
    print(text.decode('utf-8')) # 把bytes转化成utf-8
```

3. 操作文件的“wb”模式

“wb”模式也是操作文件“w”模式的一种，当文件存在时，就会清空该文件，当文件不存在时，就会创建空文件，比如当你需要对一个文档重写时就会用到。代码示例如下：

```
# wb模式写入
with open('a.txt', 'wb') as f:
    msg = '你好，世界'
    f.write(msg.encode('utf-8')) # 指定写入文件的字符编码

# rb模式读取
with open('a.txt', 'rb') as f:
    data = f.read()
    print(data)
    print(type(data))
    print(data.decode('utf-8')) # 指定读取文件的字符编码
```

4. 操作文件的“ab”模式

“ab”模式指的是以二进制形式追加写，与操作文件的“a”模式同理，代码示例如下：

```
with open('a.txt', 'ab') as f:
    msg = '\n世界：你也好，小鬼'
    f.write(msg.encode('utf-8')) # 指定写入文件的字符编码
```

补充：

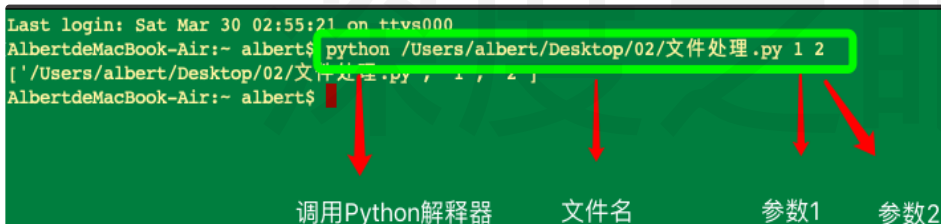
在这这里讲一个小模块的部分用法，是sys模块，它的使用说明请看下面代码示例和截图：

```
import sys # 首先导入这个模块

list_test = sys.argv # 它的返回值是一个列表
print(list_test)
```

注：小模块是python大牛已经写好的轮子，里面可能有你在特定场景下需要的一些功能性函数，你就不需要自己写函数啦，调它的库再调它的函数就完事啦~

接下来我们在终端环境下执行以下命令（注意参数1前后的空格）：



The screenshot shows a terminal window with the following text:

```
Last login: Sat Mar 30 02:55:21 on ttvs000
AlbertdeMacBook-Air:~ albert$ python /Users/albert/Desktop/02/文件处理.py 1 2
['/Users/albert/Desktop/02/文件处理.py', '1', '2']
AlbertdeMacBook-Air:~ albert$
```

Annotations with red arrows point to parts of the command:

- 调用Python解释器 (Call Python interpreter) points to `python`.
- 文件名 (File name) points to `/Users/albert/Desktop/02/文件处理.py`.
- 参数1 (Parameter 1) points to `1`.
- 参数2 (Parameter 2) points to `2`.

你可以看到在终端执行打印的结果就是一个列表，第一个值是文件路径，第二个和第三个值分别是两个参数。

5. 操作文件的其他模式（了解）

一般来说，你遇到的操作文件的模式都是只读或者只写，但是也有可读可写的模式，这类模式了解即可，说明和代码示例如下：

```
# "r+t"模式，或者写成"r+"模式，指的是可读可写
with open('a.txt', 'r+t', encoding='utf-8') as f:
    print(f.readable())
    print(f.writable())

# "w+r"或者"w+"模式，可读可写
with open('a.txt', 'w+t', encoding='utf-8') as f:
    print(f.readable())
    print(f.writable())

# "a+t"或者"a+"，可读可追加写
with open('a.txt', 'a+t', encoding='utf-8') as f:
    print(f.readable())
    print(f.writable())

# "U"模式，通用换行符，已废弃，无需了解
# with open('a.txt', 'U', encoding='utf-8') as f:
#     print(f.readable())
#     print(f.writable())
```

“r+”与“w+”，“a+”模式都是可读可写，他们的区别在于不改变自身原本操作文件的形式，即“r+”模式下，当文件不存在会保存，“w+”或者“a+”模式当文件不存在会创建新文件，当文件存在会清空文件。

五 文件内光标移动

在打开文件时会有一个光标移动，我们使用seek这个方法，可以实现光标的移动，代码示例如下：

```
with open('a.txt', 'r') as f:
    f.seek(9) # 这个参数指的是偏移量，以字节为单位
    data = f.read()
    print(data)
```

注意：光标移动只能是从左往右移动，seek可以加两个参数，第一个参数就是上面代码中的参数，你如果只传一个参数也就是指的这个参数，第二个参数默认是0，指的是光标在文件开头位置开始移动，除了0之外，只能接收1或者2作为参数，1表示从当前位置开始移动，2表示从文件末尾开始移动，其中1和2必须在“b”模式下进行，0可以在“t”模式或者“b”下都能运行，但是无论哪种模式，都是以bytes为单位进行的，代码示例如下：

```
with open('a.txt', 'rb') as f:
    f.seek(3, 0) # 移动三个字节，也就是utf-8编码下一个中文字符
    print(f.tell()) # 当前光标位置，以字节为单位
    f.seek(3, 1) # 从当前位置向右偏移3个字节然后再读取文件内容
    # f.seek(3, 2) # 只能读取动态数据新添加的内容
    data = f.read()
    print(data.decode('utf-8'))

with open('a.txt', 'r') as f:
    f.seek(3, 0)
    print(f.read())
```

补充：

在这里补充一个小模块的用法，是os模块，它的使用说明请看下面代码示例：

```
import os # 首先导入这个模块

os.rename('a.txt', 'b.txt') # 修改文件名，两个参数分别为源文件名和目标文件名
os.remove('a1.txt') # 删除a1.txt文件，这个参数指的是文件路径
```

作者：马一特

作者：马一特