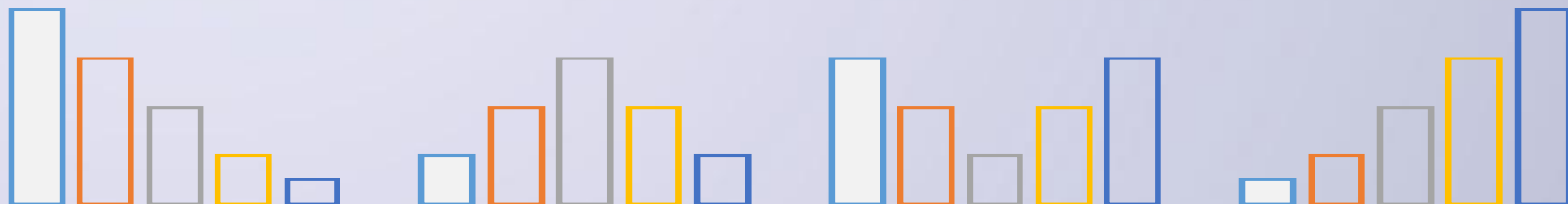


Python语言程序设计

成都信息工程大学区块链产业学院
刘硕


第7章 文件读写和numpy库



目录

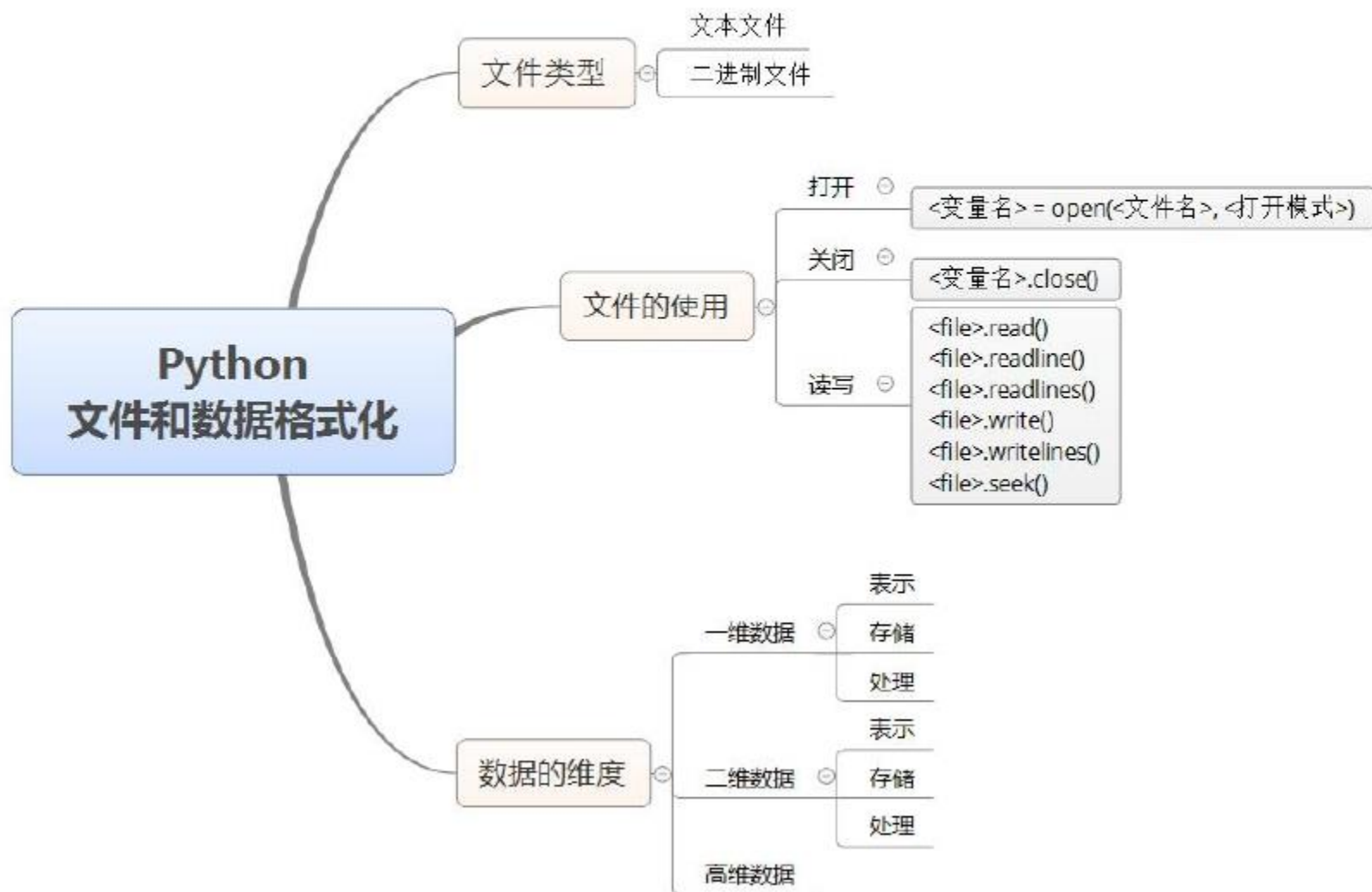
7.1 文件的使用

7.2 numpy库的使用



7.1文件的使用

知识导图

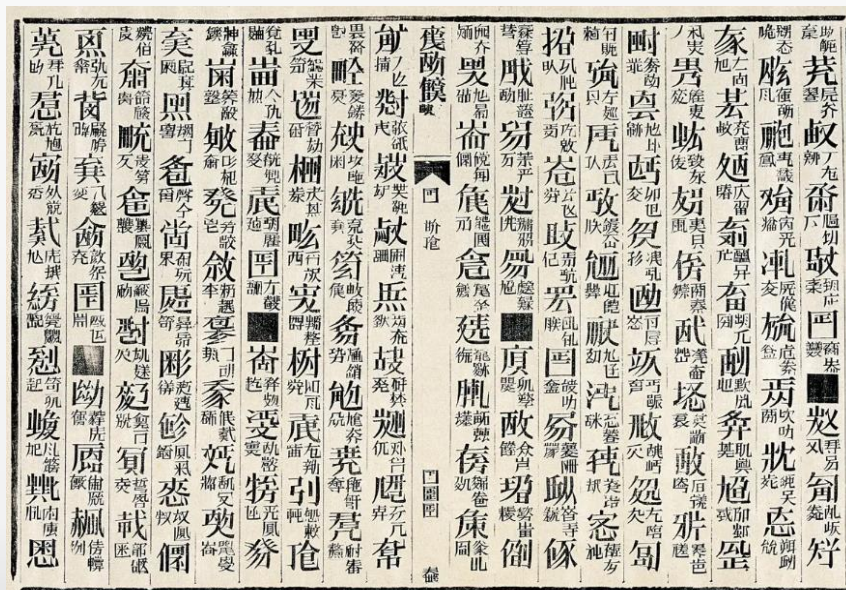




文件的使用

文件

- 文件是存储在辅助存储器上的一组数据序列，可以包含任何数据内容。概念上，文件是数据的集合和抽象。文件包括两种类型：**文本文件和二进制文件。**



文件概述

为什么要引用文件？

许多程序在实现过程中，依赖于把数据保存到变量中，而变量是通过内存单元存储数据的，数据的处理完全由程序控制。当一个程序运行完成或者终止运行，所有变量的值不再保存。

一般的程序都会有数据的输入与输出，如果输入输出的数据量不大，通过键盘和显示器即可解决。当输入输出数据量较大时，就会[受到限制](#)，[带来不便](#)。

[文件是解决上述问题的有效办法](#)。当有大量数据需要处理时，可以通过编辑工具事先建立输入数据的文件，程序运行时将不再从键盘输入，而从指定的文件中读入要处理的数据，实现数据一次输入多次使用。

当有大量数据输出时，可以将其输出到指定的文件保存，不受屏幕大小限制，并且任何时候都可以查看结果文件。一个程序的输出结果还可以作为其他程序的输入，以便进一步加工处理。

文件概述

文件的概念

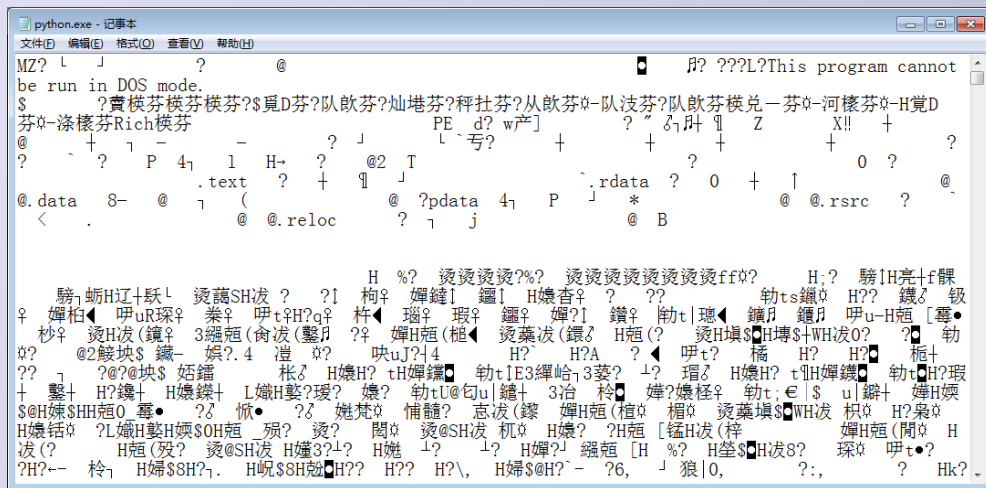
文件是一个存储在外部存储器上信息的集合，可以是文本、图片、程序等任何数据的内容。

文件的分类

文件可分为文本文件和二进制文件。

文本文件一般由单一特点的编码组成，如**UTF-8**编码，内容容易统一展示和阅读。

二进制文件存储的是数据的二进制代码（位0和位1），即将数据在内存中的存储形式复制到文件中。



文件概述

```
>>> bf=open("myfile.txt","rb",encoding='utf-8')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    bf=open("myfile.txt","rb",encoding='utf-8')
ValueError: binary mode doesn't take an encoding argument
...

```

硬盘上有一个文件名为**myfile.txt**，文件中包含一个字符串“**我爱你，中国**”，分别用文本文件方式和二进制文件方式操作，结果如下：

```
>>> tf=open("myfile.txt","r",encoding='utf-8')
>>> print(tf.readline())
我爱你，中国
>>> tf.close()
...

```

```
>>> bf=open("myfile.txt","rb")
>>> print(bf.readline())
b'\xce\xd2\xb0\xae\xc4\xe3\xa3\xac\xd6\xd0\xb9\xfa'
>>> bf.close()

```

文本文件和二进制文件**最主要的区别在于是否有相应的字符编码**。

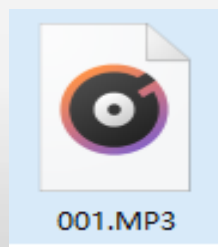
无论文件是按文本文件还是二进制文件创建，都可以用“文本文件方式”和“二进制文件方式”打开，但打开之后的操作有所不同。

采用文本方式读入文件，**文件经过解码形成字符串**，打印出有含义能被识别的字符；

采用二进制方式打开文件，**文件被解析为字节流**，由于存在编码问题，字符串中的一个字符由多个字节表示。

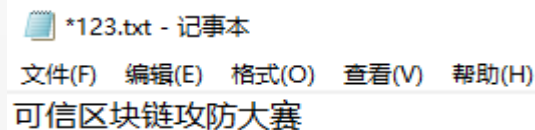
文件的类型

- 文本文件一般由单一特定编码的字符组成，如 UTF-8 编码，内容容易统一展示和阅读。文本文件存储的是常规字符串，由若干文本行组成，通常每行以换行符 '\n' 结尾。
- 二进制文件直接由比特 0 和比特 1 组成，文件内部数据的组织格式与文件用途有关。二进制是信息按照非字符但特定格式形成的文件，例如，png 格式的图片文件、avi 格式的视频文件。



文件的类型

- 二进制文件和文本文件最主要的区别在于是否有统一的字符编码。
- 无论文件创建为文本文件或者二进制文件，都可以用“文本文件方式”和“二进制文件方式”打开，但打开后的操作不同。



```
1 f = open("e:\\liushuo\\123.txt", "rt", encoding="utf-8")    #t表示文本文件方式
2 print(f.readline())
3 f.close()
```

>>>

可信区块链攻防大赛

文件的类型

- 文本文件123.txt，采用二进制方式打开

```
1 f = open("e:\\liushuo\\123.txt", "rb")    #b表示二进制文件方式
2 print(f.readline())
3 f.close()
```

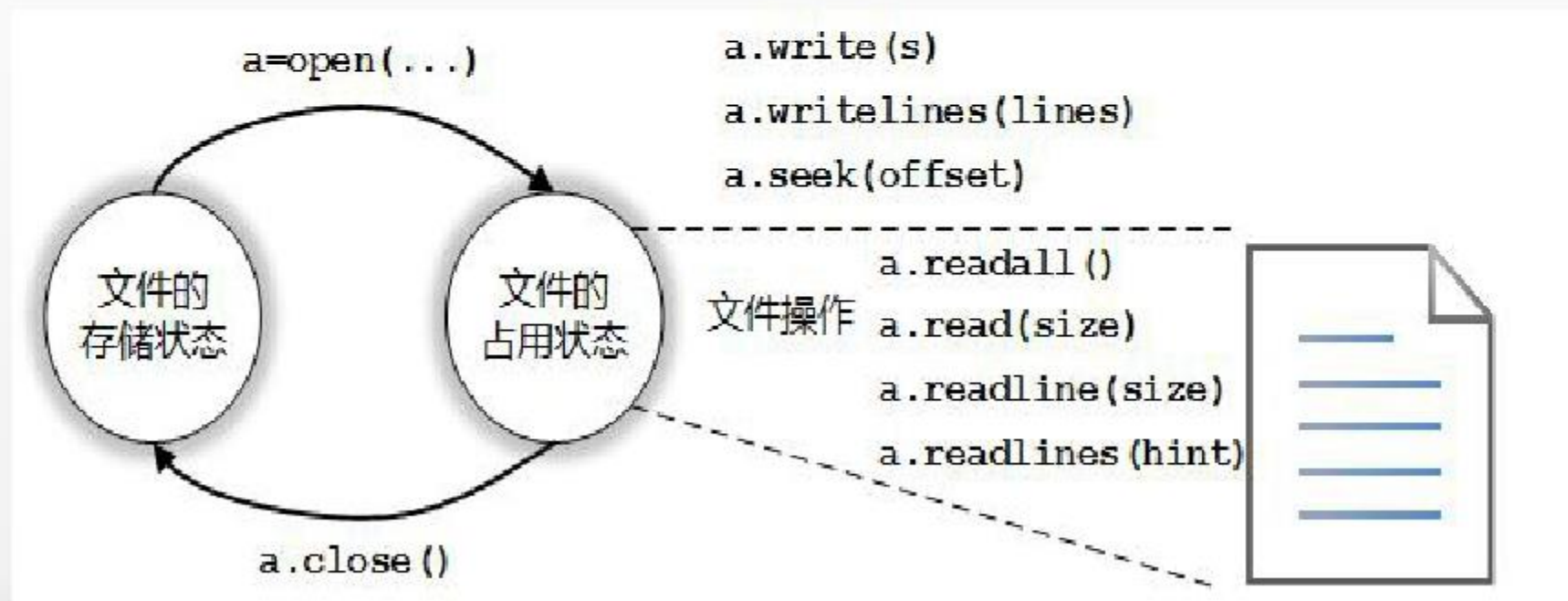
```
>>>
```

```
b'\xe5\x8f\xaf\xe4\xbf\xa1\xe5\x8c\xba\xe5\x9d\x97\xe9\x93
\xbe\xe6\x94\xbb\xe9\x98\xb2\xe5\xa4\xa7\xe8\xb5\x9b'
```

- 采用文本方式读入文件，文件经过编码形成字符串，打印出有含义的字符；采用二进制方式打开文件，文件被解析为字节流。

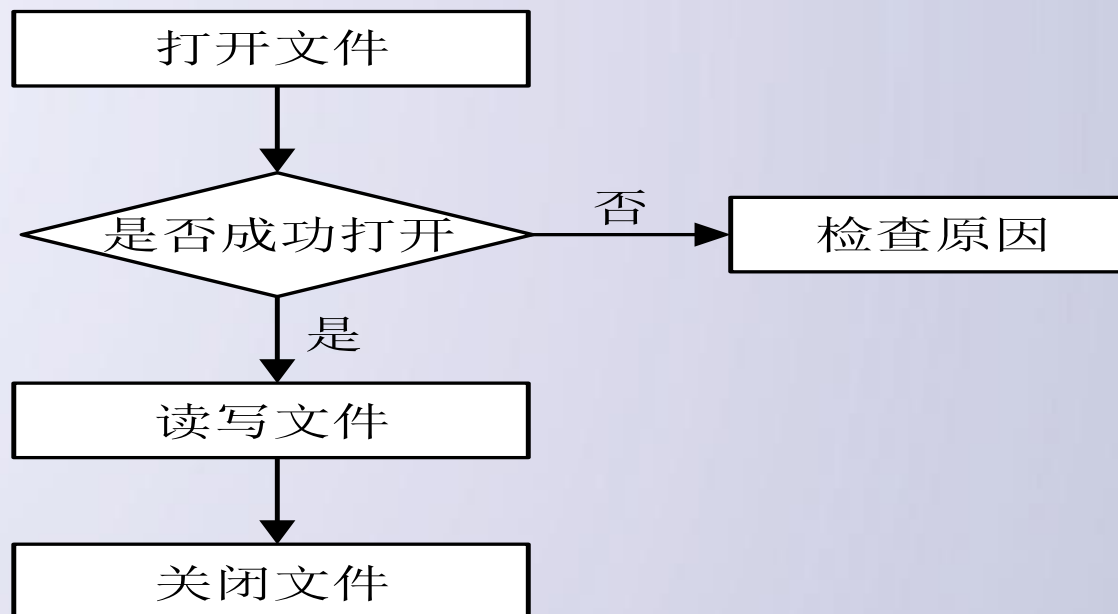
文件的打开和关闭

- Python对文本文件和二进制文件采用统一的操作步骤，即“**打开-操作-关闭**”



打开和关闭文件

打开文件



访问文件时，必须先打开文件。内置函数`open()`可以打开或创建一个文件，并返回一个文件对象。

文件的打开和关闭

- 打开模式使用字符串方式表示，根据字符串定义，单引号或者双引号均可。上述打开模式中，'r'、'w'、'x'、'b'可以和'b'、't'、'+'组合使用，形成既表达读写又表达文件模式的方式。

文件的打开和关闭

- 文件使用结束后要用close()方法关闭，释放文件的使用授权，语法形式如下：

<变量名>.close()



文件的打开和关闭

■ 新建一个文本文件123.txt，其内容为“可信区块链攻防大赛”，保存在目录PATH中，假设此时路径PATH是Windows系统的e盘根目录下的文件夹。打开并关闭该文件的操作过程如下

```
>>>PATH = "e:\\liushuo\\"
>>>f = open(PATH + "123.txt", "rt",encoding="utf-8")
>>>print(f.readline())
可信区块链攻防大赛
>>>f.close()
>>>print(f.readline())
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    print(f.readline())
ValueError: I/O operation on closed file.
```

文件基本操作



■ 文件内容操作三部曲：打开、读写、关闭

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

- ✓ 文件名指定了被打开的文件名称。
- ✓ 打开模式指定了打开文件后的处理方式。
- ✓ 缓冲区指定了读写文件的缓存模式。0表示不缓存，1表示缓存，如大于1则表示缓冲区的大小。默认值是缓存模式。
- ✓ 参数encoding指定对文本进行编码和解码的方式，只适用于文本模式，可以使用Python支持的任何格式，如GBK、utf8、CP936等等。
- ✓ open()函数返回1个文件对象，该对象可以对文件进行各种操作。

打开和关闭文件

`open()`函数的语法格式为:

文件对象=`open(file, mode='r', buffering=-1, encode=None, newline=None)`

参数**file**是待打开的文件名，文件名是字符串类型，可以使用相对路径或者绝对路径表示的文件名。

注意：文件路径中的“\”要用“\\”转义。

例如，要打开c:\python中的大风歌.txt文件，绝对路径文件名须表述成“c:\\python\\大风歌.txt”:

```
>>> fp=open("大风歌.txt","r")
>>> fp=open("c:\\python\\大风歌.txt","r")
```

参数**mode**是打开方式，打开方式也是字符串类型，用于指定打开文件后的操作方式，必须小写。

打开和关闭文件

文件的打开方式

模式	描述	说明
r	以只读的方式打开文件。是默认方式	1.如果文件不存在，则抛出 FileNotFoundError 异常； 2.不清空原内容；“+”模式可同时读出和写入内容 3.文件打开时文件指针指向文件的开头
rb	以只读的方式打开二进制文件	
r+	以读写的方式打开文件	
rb+	以读写的方式打开二进制文件	
w	以只写的方式打开文件	1.如果文件存在，则其内容覆盖；如果文件不存在，则创建新文件 2.清空原文件 3.文件打开时文件指针指向文件的开头
wb	以只写的方式打开二进制文件	
w+	以读写的方式打开文件	
wb+	以读写的方式打开二进制文件	
a	以追加的方式打开文件	1.如果文件存在，不清空原文件，文件指针指向文件的结尾 2.如果文件不存在，则创建新文件 3."a+"模式允许在任意位置读，但只能在文件末尾追加数据
ab	以追加的方式打开二进制文件	
a+	以读写的方式打开文件	
ab+	以读写的方式打开二进制文件	

打开和关闭文件

关闭文件

文件操作完成后，应及时关闭文件。关闭文件的语法格式为：

文件对象.close()

close()方法用于关闭已打开的文件，并释放文件对象所占用的资源。如果再想使用刚才打开的文件，则必须重新打开。

```
>>> fp=open("myfile.txt","r")
>>> print("打开文件的文件名为:",fp.name)
打开文件的文件名为: myfile.txt
>>> print("文件关闭状态:",fp.closed)
文件关闭状态: False
>>> fp.close()
>>> print("文件关闭状态:",fp.closed)
文件关闭状态: True
```

文件基本操作

- 如果执行正常，`open()`函数返回1个可迭代的文件对象，通过该文件对象可以对文件进行读写操作，如果指定文件不存在、访问权限不够、磁盘空间不够或其他原因导致创建文件对象失败则抛出异常。下面的代码分别以读、写方式打开了两个文件并创建了与之对应的文件对象。

```
f1 = open( 'file1.txt', 'r' )
```

```
f2 = open( 'file2.txt', 'w' )
```

- 当对文件内容操作完以后，一定要关闭文件对象，这样才能保证所做的任何修改都确实被保存到文件中。

```
f1.close()
```

文件基本操作

- 需要注意的是，即使写了关闭文件的代码，也无法保证文件一定能够正常关闭。例如，如果在打开文件之后和关闭文件之前发生了错误导致程序崩溃，这时文件就无法正常关闭。在管理文件对象时推荐**with**关键字，可以有效地避免这个问题。

打开和关闭文件

关闭文件

如果在写文件的程序中不调用`close()`方法关闭文件，有时会发生缓冲区中数据不能正确写入磁盘的现象。为了避免这种情况的发生，Python引入了`with`语句来自动调用`close()`方法，其语法格式如下：

```
with open(文件名,访问模式) as 文件对象:  
    <操作>
```

当`with`内部的语句执行完毕后，文件将自动关闭，而不需要显式调用`close()`方法。例如：

#将“myfile.txt”文件中的内容复制到“copy.txt”中

```
>>>with open("myfile.txt","r",encoding="utf-8") as fp1,open("copy.txt","w")  
as fp2:  
    s=fp1.read()  
    fp2.write(s)  
>>> print("文件关闭状态:",fp1.closed)  
文件关闭状态: True  
>>> print("文件关闭状态:",fp2.closed)  
文件关闭状态: True
```

文件的基本操作

1. 文件的读写

文件打开后，可以根据打开文件的模式对文件进行操作。对文件的操作主要是读/写操作。用于读/写文件的方法有：

`read()`

`write()`

`readline()`

`readlines()`

`writelines()`

文件的基本操作

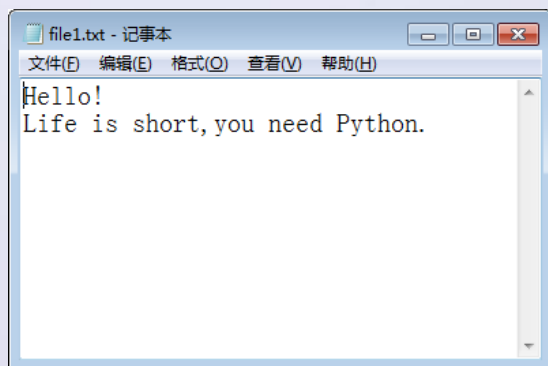
1. 文件的读写

(1) read()方法

read()方法的用法如下：

变量=文件对象.read(size)

例如：当前工作目录中已有文件file1.txt，使用read()方法将其读出。



```
>>> fp=open("file1.txt","r")
>>> fp.read(6)      #从文件中读取6个字符
'Hello!'
>>> fp.read()       #将文件中余下的数据读出，换行符'\n'同时被读出
'\nLife is short,you need Python.\n'
>>> fp.close()
```

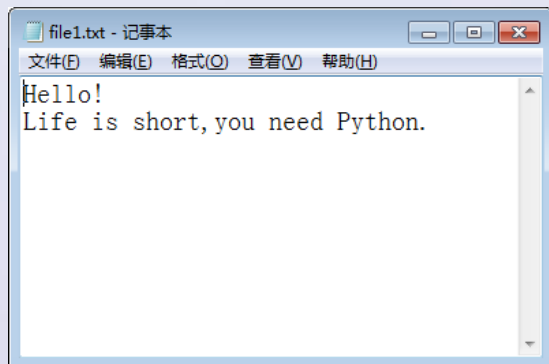
文件的基本操作

1. 文件的读写

(2) readline() 方法

readline() 方法的用法如下：

变量=文件对象.readline(size=-1)



```
>>> fp=open("file1.txt","r")
>>> fp.readline(4)      #从文件中读取4个字符
'Hell'
>>> fp.readline()      #从当前位置到本行末所有
字符读出
'o!\n'
>>> fp.readline()      #读取下一行
'Life is short,you need Python.\n'
>>> fp.readline()      #从文件末尾处读，返回空
串
''
>>> fp.close()
```

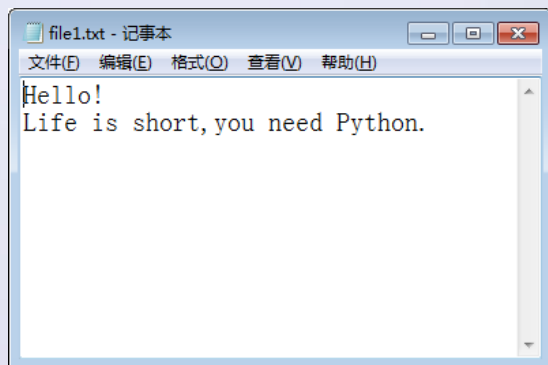
文件的基本操作

1. 文件的读写

(3) readlines()方法

readlines()方法的用法如下：

变量=文件对象.readlines()



```
>>> fp=open("file1.txt","r")
>>> fp.readlines()
['Hello!\n', 'Life is short,you need Python.\n']
>>> fp.readlines()
[]
>>> fp.close()
```

文件的基本操作

例 统计文本文件file1.txt中大写字母出现的次数。

分析：读取文件的所有行，以列表返回，然后遍历列表，统计大写字母的个数。

```
1 fp=open("file1.txt","r")
2 ls=fp.readlines()      #读取文件的所有行，以列表返回
3 c=0
4 for x in ls:            #遍历列表ls
5     print(x.strip())    #输出文件的每一个行（不包括'\n'）
6     for s in x:         #遍历每个列表元素
7         if s.isupper():
8             c=c+1
9     print("文件中大写字母有{}个。".format(c))
10 fp.close()
```

```
fp=open("file1.txt","r")
ls=fp.readlines()      #读取文件的所有行，以列表返回
c=0
for x in ls:            #遍历列表ls
    #print(x.strip())    #输出文件的每一个行（不包括'\n'）
    print(x)
    for s in x:         #遍历每个列表元素
        if s.isupper():
            c=c+1
print("文件中大写字母有{}个。".format(c))
fp.close()
```

Hello!

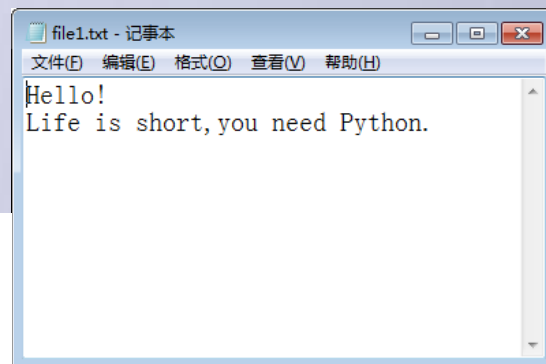
Life is short,you need Python.
文件中大写字母有3个。

运行程序，结果如下：

Hello!

Life is short,you need
Python.

文件中大写字母有3个。



File Edit Format Run Options Window Help

```
fp=open("12.txt","r")
ls=fp.readlines()
for x in ls:           #遍历列表ls
    print(x.strip())   #输出文件的每一个行（不包括'\n'）
    #print(x)
fp.close()
```

```
1
2
3
```

*12.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V)

```
1
2
3
```

File Edit Format Run Options Window Help

```
fp=open("12.txt","r")
ls=fp.readlines()
for x in ls:           #遍历列表ls
    #print(x.strip())   #输出文件的每一个行（不包括'\n'）
    print(x)
fp.close()
```

4.py

```
1
2
3
```

文件的基本操作

(4) write()方法

write()方法的用法如下：

变量=文件对象.write(字符串)

例如：使用write()方法将字符串“Python语言”、“等级考试”、“CUI”写入文件file1.txt的末尾。

>>> fp=open("file1.txt","a") #以追加的方式打开文件

>>> fp.write("Python语言")

8

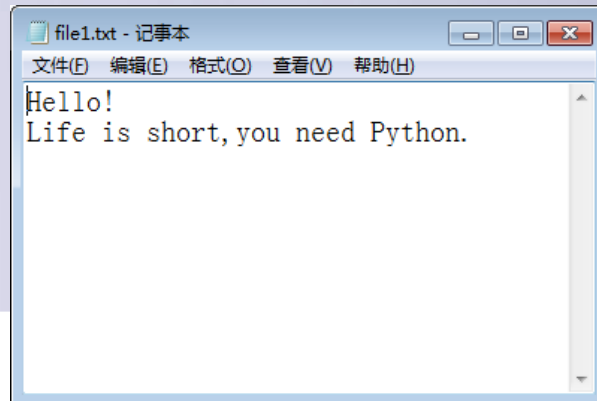
>>> fp.write("等级考试\n")

5

>>> fp.write("CUI\n")

5

>>> fp.close()



*file1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Hello!

Life is short,you need Python.Python语言等级考试
CUI

文件的基本操作

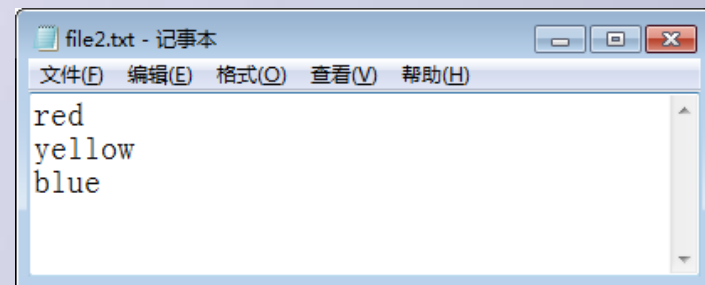
(5) writelines()方法

writelines()方法的用法如下：

变量=文件对象.writelines(列表)

例如：使用writelines()方法将"red"、"yellow"、"blue"以3行的形式写入文件file2.txt中。

```
>>> fp=open("file2.txt","w")
>>>
fp.writelines(["red\n","yellow\n","blue\n"])
>>> fp.close()
```



```
>>> fp=open("file3.txt","w")
>>> fp.writelines(["red","yellow","blue"])
>>> fp.close()
```

file3.txt - 记事本

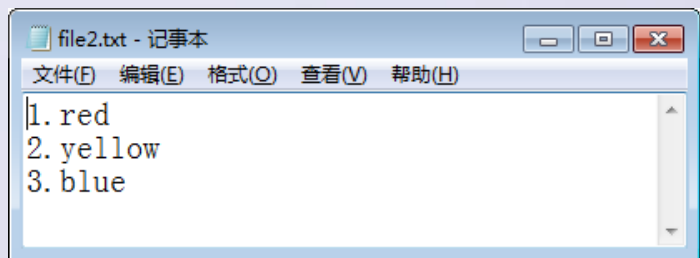
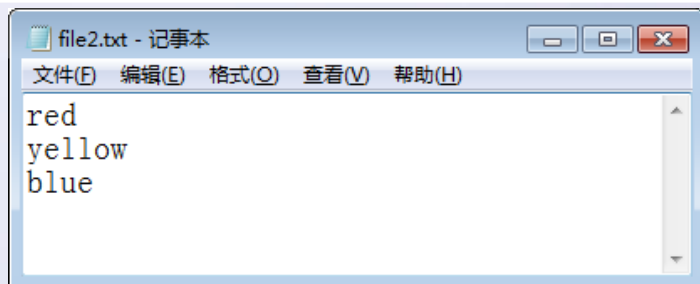
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

redyellowblue

文件的基本操作

例-请将文件file2.txt中的字符串前面加上序号1.,2.,3., 执行后文件中数据如图所示。

分析：使用**readlines**方法将文件**file2.txt**的内容以列表的形式读出，加上序号后再写入文件中。



```
1 fout=open("file2.txt","r")
2 s=fout.readlines()
3 fout.close()
4 for i in range(len(s)):
5     s[i]=str(i+1)+ "." +s[i]
6 fin=open("file2.txt","w")
7 fin.writelines(s)
8 fin.close()
```

文件的基本操作

2. 文件的定位

文件中有一个位置指针，指向当前的读/写位置，读/写一次指针向后移动一次。
Python提供**seek()**方法可以主动调整指针的位置，实现文件的随机访问。
也可使用**tell()**方法获取当前文件指针的位置。

(1) seek()方法

seek()方法的用法如下：

文件对象.**seek(offset,whence=0)**

Python中使用**seek()**方法更改当前文件位置。

文件的基本操作

文件指针的移动是相对的。

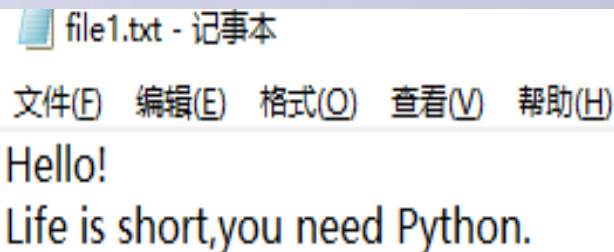
例如：

```
>>> fp=open("file1.txt","r")
...
>>> fp.read()
...
'Hello!\nLife is short,you need Python.\n'
>>> fp.seek(22,0)
...
22
>>> fp.read()
...
'you need Python.\n'
>>>
```

```
>>> fp=open("file1.txt","rb")           #以二进制的方式读取file1.txt文件
>>> fp.read()
b'Hello!\r\nLife is short,you need Python.'  #二进制读取时需要将'\n'转换成'\r\n'
>>> fp.read()                             #从当前位置继续读取文件
b"                                           #读出空串
>>> fp.seek(22,0)                          #从文件开始位置移动22个字节
22
>>> fp.read()                             #从移动后的位置开始读取文件
b'you need Python.'
>>> fp.close()
```

```
>>> fp.seek(1,0)
1
>>> fp.read()
b'ello!\r\nLife is short,you need Python.\r\n'
```

```
>>> fp.closed
False
>>> fp.close()
>>> fp.closed
True
```



file1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Hello!

Life is short,you need Python.

文件的基本操作

(2) tell()方法

tell()方法的用法如下：

文件对象.tell()

Python中使用文件对象的tell()方法返回文件的当前读写位置。其功能是获取文件的当前位置，即相对于文件开始位置字节数。例如：

```
>>> fp=open("file1.txt","rb")
>>> fp.tell()
0
>>> fp.read(8)
b'Hello!\r\n'
>>> fp.tell()
8
>>> fp.read()
b'Life is short,you need Python.'
>>> fp.close()
```

file1.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

Hello!

Life is short,you need Python.

文件基本操作

■ 文件对象常用方法

方法	功能说明
<code>close()</code>	把缓冲区的内容写入文件，同时关闭文件，并释放文件对象
<code>detach()</code>	分离并返回底层的缓冲，底层缓冲被分离后，文件对象不再可用，不允许做任何操作
<code>flush()</code>	把缓冲区的内容写入文件，但不关闭文件
<code>read([size])</code>	从文本文件中读取size个或字符（Python 3.x）的内容作为结果返回，或从二进制文件中读取指定数量的字节并返回，如果省略size则表示读取所有内容
<code>readable()</code>	测试当前文件是否可读
<code>readline()</code>	从文本文件中读取一行内容作为结果返回
<code>readlines()</code>	把文本文件中的每行文本作为一个字符串存入列表中，返回该列表，对于大文件会占用较多内存，不建议使用
<code>seek(offset[, whence])</code>	把文件指针移动到新的位置，offset表示相对于whence的位置。whence为0表示从文件头开始计算，1表示从当前位置开始计算， 2表示从文件尾开始计算 ，默认为0
<code>seekable()</code>	测试当前文件是否支持随机访问，如果文件不支持随机访问，则调用方法 <code>seek()</code> 、 <code>tell()</code> 和 <code>truncate()</code> 时会抛出异常
<code>tell()</code>	返回文件指针的当前位置
<code>truncate([size])</code>	删除从当前指针位置到文件末尾的内容。如果指定了size，则不论指针在什么位置都只留下前size个字节，其余的一律删除
<code>write(s)</code>	把字符串s的内容写入文件
<code>writable()</code>	测试当前文件是否可写
<code>writelines(s)</code>	把字符串列表写入文本文件，不添加换行符

文件的基本操作

例8.3 按字节输出file1.txt文件中的前5个字节和后7个字节。

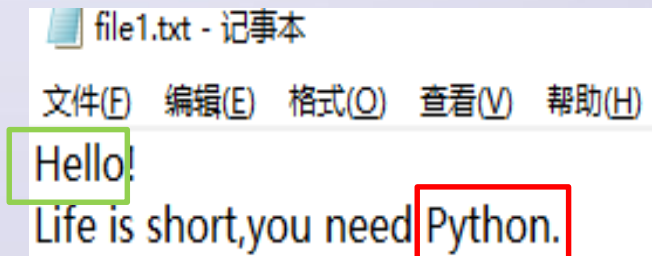
分析：用文件对象的**seek()**方法实现文件指针的移动。

```
1 fp=open("file1.txt","rb")
2 str1=fp.read(5)
3 fp.seek(-7,2)
4 str2=fp.read()
5 print("文件file.txt的前5个字节为{}".format(str1))
6 print("文件file.txt的后7个字节为{}".format(str2))
7 fp.close()
```

运行程序，结果如下：

文件file.txt的前5个字符为b'Hello'

文件file.txt的后7个字符为b'Python.'



7.1 文件基本操作

- 文件对象常用属性

属性	说明
buffer	返回当前文件的缓冲区对象
closed	判断文件是否关闭，若文件已关闭则返回True
fileno	文件号，一般不需要太关心这个数字
mode	返回文件的打开模式
name	返回文件的名称

```
>>> f=open("e:\\liushuo\\123.txt","r",encoding="utf-8")
>>> f.buffer
<_io.BufferedReader name='e:\\liushuo\\123.txt'>
>>> f.closed
False
>>> f.mode
'r'
```




7.1 文件概述-展示的方式不同

二进制文件直接由比特0和比特1组成，没有统一字符编码，文件内部数据的组织格式与文件用途有关。二进制文件和文本文件最主要的区别在于是否有统一的字符编码

无论文件创建为文本文件或者二进制文件，都可以用“文本文件方式”和“二进制文件方式”打开，打开后的操作不同。

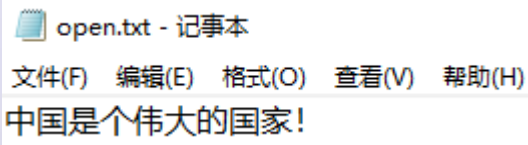
7.1 文件概述-展示方式的不同

微实例7.1：理解文本文件和二进制文件的区别。

微实例7.1

```
1  textFile = open("e:\\liushuo\\open.txt", "rt")  #t 表
2  print(textFile.readline())
3  textFile.close()
4  binFile = open("e:\\liushuo\\open.txt", "rb")  #r 表
5  示二进制文件方式
6  print(binFile.readline())
   binFile.close()
```

```
print(textFile.readline())
UnicodeDecodeError: 'gbk' codec can't decode byte 0xad in position 2: illegal mu
ltibyte sequence
```



encoding="utf-8", 也可以选择编码方式为ASCII等，试着看看不同的编码方式encoding对应的参数



7.1 文件概述

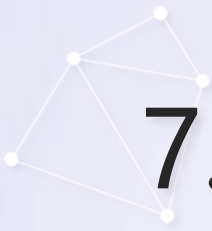
输出结果为：

```
>>>
```

```
中国是个伟大的国家！
```

```
b'\xe4\xb8\xad\xe5\x9b\xbd\xe6\x98\xaf\xe4\xb8\xaa\xe4\xbc\x9f\xe5\xa4\xa7\xe7\x9a\x84\xe5\x9b\xbd\xe5\xae\xb6\xef\xbc\x81\r\n'
```

采用文本方式读入文件，文件经过编码形成字符串，打印出有含义的字符；采用二进制方式打开文件，文件被解析为字节（byte）流。由于存在编码，字符串中的一个字符由2个字节表示。



7.1 文件概述

Python通过解释器内置的open()函数打开一个文件，并实现该文件与一个程序变量的关联，open()函数格式如下：

<变量名> = open(<文件名>, <打开模式>)

open()函数有两个参数：文件名和打开模式。文件名可以是文件的实际名字，也可以是包含完整路径的名字

7.1 文件概述-文件的读写

微实例7.2：文本文件逐行打印

微实例7.2

m7.2PrintFilebyLines.py

```
1 fname = input("请输入要打开的文件: ")
2 fo = open(fname, "r")
3 for line in fo.readlines():
4     print(line)
5 fo.close()
```

中国是个伟大的国家！
我爱我的祖国！

open打开成功后，返回的是一个可迭代的文件

```
fo = open(fname, "r", encoding="utf-8")
for line in fo:
    print(line)
```

File Edit Format Run Options Window Help

```
fname = input("请输入要打开的文件: ")
fo = open(fname, "rt", encoding="utf-8")
for line in fo.readlines():
    print(line)
fo.close()
```

>>>

请输入要打开的文件: e:\\liushuo\\open.txt
中国是个伟大的国家！
我爱我的祖国！

7.1 文件概述-文件的读写

遍历文件的所有行可以直接这样完成

open.txt

中国是个伟大的国家！
我爱我的祖国！

```
1 fname = input("请输入要打开的文件: ")
2 fo = open(fname, "r")
3 for line in fo:
4     print(line)
5 fo.close()
```


```
fname = input("请输入要打开的文件: ")
fo = open(fname, "rt", encoding="utf-8")
# for line in fo.readlines():
print(fo.readlines())
fo.seek(0)
print(' '.join(fo.readline()))
fo.close()
```

请输入要打开的文件: e:\liushuo\open.txt
['中国是个伟大的国家！\n', '我爱我的祖国！\n']
中国是个伟大的国家！

```
fname = input("请输入要打开的文件: ")
fo = open(fname, "r", encoding="utf-8")
print(fo.readlines())
fo.close()
```

>>>


```
=====
请输入要打开的文件: e:\\liushuo\\open.txt
['中国是个伟大的国家！\n', '我爱我的祖国！\n']
```



7.1文件概述-文件的读写

根据打开方式不同可以对文件进行相应的读写操作，Python提供4个常用的文件内容读取方法

方法	含义
<code><file>.readall()</code>	读入整个文件内容，返回一个字符串或字节流*
<code><file>.read(size=-1)</code>	从文件中读入整个文件内容，如果给出参数，读入前size长度的字符串或字节流
<code><file>.readline(size = -1)</code>	从文件中读入一行内容，如果给出参数，读入该行前size长度的字符串或字节流
<code><file>.readlines(hint=-1)</code>	从文件中读入所有行，以每行为元素形成一个列表，如果给出参数，读入hint行



7.1 文件概述-文件的读写

Python提供3个与文件内容写入有关的方法，如表所示。

方法	含义
<code><file>.write(s)</code>	向文件写入一个字符串或字节流
<code><file>.writelines(lines)</code>	将一个元素为字符串的列表写入文件
<code><file>.seek(offset)</code>	改变当前文件操作指针的位置， <code>offset</code> 的值： 0：文件开头； 1：当前位置； 2：文件结尾

7.1 文件概述-文件的读写

微实例7.3

```
1 fname = input("请输入要写入的文件: ")
2 fo = open(fname, "w+")
3 ls = ["唐诗", "宋词", "元曲"]
4 fo.writelines(ls)
5 for line in fo:
6     print(line)
7 fo.close()
```



程序执行结果如下:

```
>>>请输入要写入的文件: e:\liushuo\唐诗宋词.txt
>>>
```

微实例7.3

```
1 fname = input("请输入要写入的文件: ")
2 fo = open(fname, "w+")
3 ls = ["唐诗", "宋词", "元曲"]
4 fo.writelines(ls)
5 fo.seek(0)
6 for line in fo:
7     print(line)
8 fo.close()
```

```
请输入要写入的文件: e:\liushuo\唐诗宋词.txt
唐诗宋词元曲
```

*唐诗宋词.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

唐诗宋词元曲

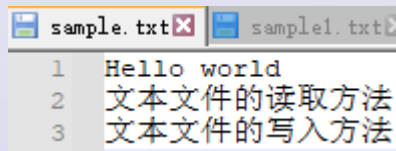
7.1 文本文件操作案例精选

- 例7-1 向文本文件中写入内容，然后再读出。

```
s = 'Hello world\n文本文件的读取方法\n文本文件的写入方法\n'
```

```
with open( 'e:\\liushuo\\sample.txt', 'w') as fp:    #默认使用
    cp936编码
    fp.write(s)
```

```
with open(' e:\\liushuo\\sample.txt ') as fp:    #默认使用
    cp936编码
    print(fp.read())
```



```
sample.txt
1 Hello world
2 文本文件的读取方法
3 文本文件的写入方法
```

7.1 文本文件基本操作

■ 例7-2 读取并显示文本文件的前5个字符。

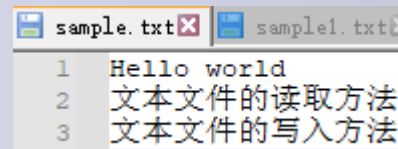
```
f=open('sample.txt','r')
```

```
s=f.read(5)    #读取文件的前5个字符 → 二进制文件就读取前5个字节
```

```
f.close( )
```

```
print('s=',s)
```

```
print('字符串s的长度(字符个数)=' , len(s))
```



Line	Content
1	Hello world
2	文本文件的读取方法
3	文本文件的写入方法

```
s= Hello
字符串s的长度(字符个数)= 5
```

seek(num) :将文件指针定位到文件中指定字节的位置。读取时遇到无法解码的字符会抛出异常。针对于这个函数，我们需要明确的是：其移动单位是字节，不是字符。例如：在一个同时含有中英文的文本文件中，其采用的编码是'gbk'，那么就要注意，一个中文字是一个字符，但是它占两个字节，一个英文字也是一个字符，但是它占一个字节。如果将文件指针定位到一个中文的中间字节位置，肯定是无法解码的，因而会抛出异常。

read([size]) :从文件中读取size字节(二进制文件)或字符(文本文件)的内容作为结果返回,如果省略size,则表示一次性读取所有内容。针对于这个函数，我们需要注意的是：当读取的文件类型不同时，size的单位不一样，二进制文件的读取单位是字节，文本文件的读取单位是字符。并且在。每一次读取成功后，文件指针 会自动向后移动。

7.1 文本文件基本操作

```
>>> s = '中国四川成都CDUIT'  
>>> fp = open(r'e:\liushuo\sample1.txt', 'w')  
>>> fp.write(s)  
11  
>>> fp.close()  
>>> fp = open(r'e:\liushuo\sample1.txt ', 'r')  
>>> print(fp.read(3))  
中国四  
>>> fp.seek(2)  
2  
>>> print(fp.read(1))  
国  
>>> fp.seek(13)  
13  
>>> print(fp.read(1))  
D  
>>> fp.seek(3)  
3  
>>> print(fp.read(1))
```

UnicodeDecodeError: 'gbk' codec can't decode byte 0xfa in position 0: illegal multibyte sequence

7.1 文本文件操作案例精选

- 例读取文本文件**data.txt**（文件中每行存放一个整数）中所有整数，将其按升序排序后再写入文本文件**data_asc.txt**中。

```
with open( 'e:\liushuo\data.txt', 'r') as fp:
    data = fp.readlines()
data = [int(line.strip()) for line in data]
data.sort()
data = [str(i)+'\n' for i in data]
with open( 'e:\liushuo\data_asc.txt', 'w') as fp:
    fp.writelines(data)
```

read readline readlines的比较

read.py

- `f = open('sample.txt', 'r')`
- `print(f.read())`
- `f.close()`

```
py =====  
Hello world  
文本文件的读取方法  
文本文件的写入方法
```

sample.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V)

Hello world
文本文件的读取方法
文本文件的写入方法
|

readline1.py

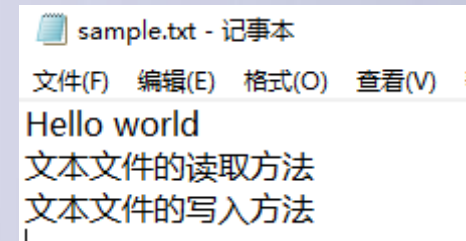
```
f = open('sample.txt', 'r')  
print(f.readline())  
f.close()
```

```
readline1.py =====  
Hello world
```

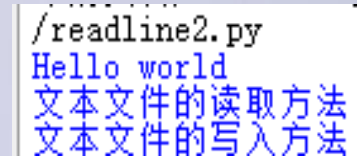
read readline readlines的比较

readline2.py

- `f = open('sample.txt', 'r')`
- `line = f.readline()`
- `while line:`
- `print(line, end="")`
- `line = f.readline()`
- `f.close()`



sample.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
Hello world
文本文件的读取方法
文本文件的写入方法



```
/readline2.py  
Hello world  
文本文件的读取方法  
文本文件的写入方法
```

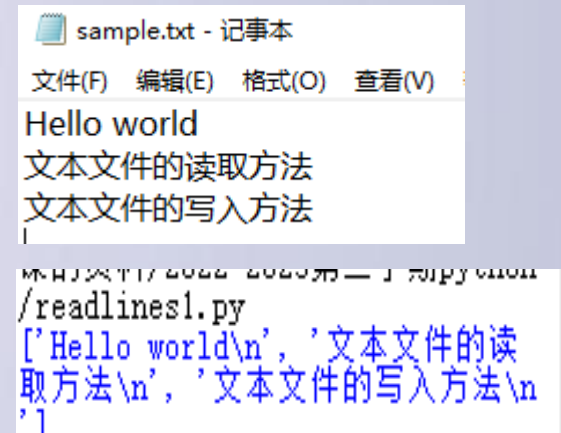
read readline readlines的比较

readlines¹.py

- `f = open('sample.txt', 'r')`
- `print(f.readlines())`
- `f.close()`

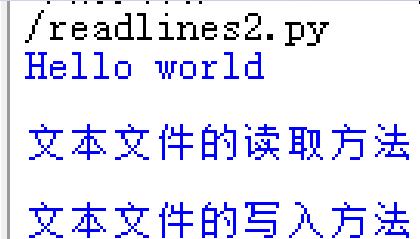
readlines².py

- `f = open('sample.txt', 'r')`
- `for line in f.readlines():`
- `print(line)`
- `f.close()`



```
sample.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
Hello world
文本文件的读取方法
文本文件的写入方法

Python 3.11.7 Shell
/README/2022-2023第二学期python
/readlines1.py
['Hello world\\n', '文本文件的读
取方法\\n', '文本文件的写入方法\\n']
```



```
/readlines2.py
Hello world

文本文件的读取方法
文本文件的写入方法
```




2 numpy 库的使用

NumPy的导入

- 标准的Python 中用列表(list)保存一组值，可以当作数组使用。但由于列表的元素可以是任何对象，因此列表中保存的是对象的指针。对于数值运算来说，这种结构显然比较浪费内存和CPU计算
- Python 提供了array 模块，它和列表不同，能直接保存数值，但是由于它不支持多维数组，也没有各种运算函数，因此也不适合做数值运算。

```
>>> import array
>>> dir(array)
['ArrayType', '__doc__', '__loader__', '__name__', '__package__', '__spec__', '_array_reconstructor',
 'array', 'typecodes']
...>
```

NumPy的导入

- NumPy的诞生弥补了这些不足，Numpy是用于处理含有同种元素的多维数组运算的第三方库。
- NumPy 提供了两种基本的对象：ndarray（n-dimensional array object）和ufunc（universal function object）。
- ndarray(下文统一称之为数组)是存储单一数据类型的多维数组，而ufunc则是能够对数组进行处理的函数。
- 因此，Python 语言的第三方库numpy 得到了迅速发展，至今，numpy 已经成为了科学计算事实上的标准库。

add, all, any, arange, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, corrcoef, cov, cross, cumprod, cumsum, diff, dot, exp, floor, ...

numpy 库概述

- `numpy` 库处理的最基础数据类型是由同种元素构成的多维数组（`ndarray`），简称“数组”。
- 数组中所有元素的类型必须相同，数组中元素可以用整数索引，序号从0开始。`ndarray`类型的维度(`dimensions`)叫做轴(`axes`)，轴的个数叫做秩(`rank`)。一维数组的秩为1，二维数组的秩为2，二维数组相当于由两个一维数组构成。

numpy 库概述

- 由于numpy 库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用numpy 库：

```
>>>import numpy as np
```

- 其中，**as** 保留字与**import**一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。简单说，在程序的后续部分中，**np** 代替**numpy**。

numpy 库常用的创建数组函数

函数	描述
<code>np.array([x,y,z], dtype=int)</code>	从 Python 列表和元组创建数组
<code>np.arange(x,y,i)</code>	创建一个由 x 到 y，以 i 为步长的数组
<code>np.linspace(x,y,n)</code>	创建一个由 x 到 y，等分成 n 个元素的数组
<code>np.indices((m,n))</code>	创建一个 m 行 n 列的矩阵
<code>np.random.rand(m,n)</code>	创建一个 m 行 n 列的随机数组
<code>np.ones((m,n),dtype)</code>	创建一个 m 行 n 列全 1 的数组，dtype 是数据类型

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array((5, 6, 7, 8))
>>> c = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]])
>>> b
array([5, 6, 7, 8])
>>> c
array([[1, 2, 3, 4],
       [4, 5, 6, 7],
       [7, 8, 9, 10]])
>>> c.dtype #数组的元素类型可以通过dtype 属性获得
dtype('int32')
```

```
>>> a.shape #一维数组
(4,)
```

```
>>> c.shape #二维数组其中第0 轴的长度为3，第1 轴的长度为4。
(3, 4)
```

```
>>> c.shape = 4,3 #注意从(3,4)改为(4,3)并不是对数组进行转置，而只是改变每个轴的大小，数组元素在内存中的位置并没有改变：
```

```
>>> c
array([[ 1,  2,  3],
       [ 4,  4,  5],
       [ 6,  7,  7],
       [ 8,  9, 10]])
```

```
>>> b.shape
(4,)
```

ndarray基本概念



- ndarray数组属性

N维数组

- 维度(dimensions)称为轴(axes), 轴的个数称为秩(rank)
- 沿着第0轴和第1轴操作
 - axis = 0 (按列)
 - axis = 1 (按行)

ndarray基本概念

• ndarray数组属性

N维数组

—基本属性

```
>>> x.ndim
2
>>> x.shape
(3, 4)
>>> x.size
12
```



- ndarray.ndim (秩)
- ndarray.shape (维度)
- ndarray.size (元素总个数)
- ndarray.dtype (元素类型)
- ndarray.itemsize (元素字节大小)

ndarray 类的常用属性

- 创建一个简单的数组后，可以查看ndarray 类型有一些基本属性

属性	描述
ndarray.ndim	数组轴的个数，也被称作秩
ndarray.shape	数组在每个维度上大小的整数元组
ndarray.size	数组元素的总个数
ndarray.dtype	数组元素的数据类型，dtype 类型可以用于创建数组中
ndarray.itemsize	数组中每个元素的字节大小
ndarray.data	包含实际数组元素的缓冲区地址
ndarray.flat	数组元素的迭代器

```
>>> x=np.array([1,2,3,4],dtype=float)
>>> x
array([1., 2., 3., 4.])
>>> x.itemsize
8
```

```
>>> x.dtype
dtype('float64')
```

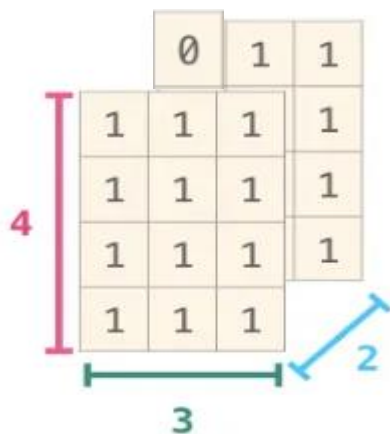
```
>>> y=x.astype('int')
>>> y.itemsize
4
>>> y.dtype
dtype('int32')
>>> z=x.astype('int16')
>>> z.itemsize
2
>>> z1=x.astype('int8')
>>> z1.itemsize
1
```

ndarray 类的常用属性

```
>>>import numpy as np
>>>a = np.ones((4,5))
>>>print(a)
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
>>>a.ndim
2
>>>a.shape
(4,5)
>>>a.dtype
dtype('float64')
```

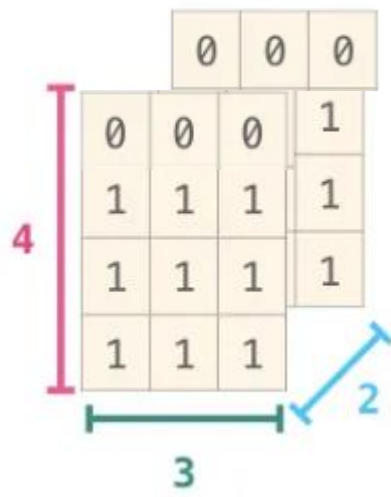
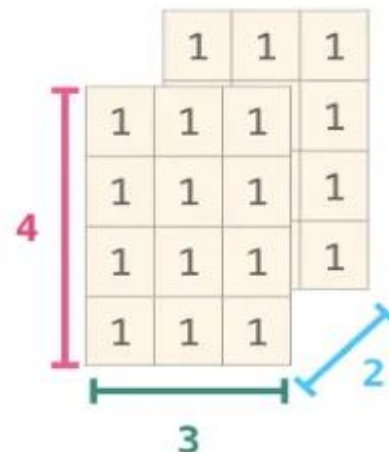
```
y=np.ones((4,3,2))
y[0][0][1]=0
y
array([[[1., 0.],
        [1., 1.],
        [1., 1.]],
       [[1., 1.],
        [1., 1.],
        [1., 1.]],
       [[1., 1.],
        [1., 1.],
        [1., 1.]],
       [[1., 1.],
        [1., 1.],
        [1., 1.]])
```

`np.ones((4,3,2))`



```
y[0][:]=0
y
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],
       [[1., 1.],
        [1., 1.],
        [1., 1.]],
       [[1., 1.],
        [1., 1.],
        [1., 1.]],
       [[1., 1.],
        [1., 1.],
        [1., 1.]])
```

`np.ones((4,3,2))`



ndarray的创建

array()函数

```
>>> import numpy as np
>>> aArray = np.array([1,2,3])
>>> aArray
array([1, 2, 3])
>>> bArray=np.array([(1,2,3),(4,5,6)], dtype
='float32')
>>> bArray
array([[1., 2., 3.],
       [4., 5., 6.]], dtype=float32)
>>> bArray.ndim, bArray.shape, bArray.dtype
(2, (2, 3), dtype('float32'))
```

创建数组

数组的元素类型可以通过**dtype**属性获得。可以通过**dtype**参数在创建时指定元素类型:

```
>>> np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]], dtype=np.float)
array([[ 1.,  2.,  3.,  4.],
       [ 4.,  5.,  6.,  7.],
       [ 7.,  8.,  9., 10.]])
>>> np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]], dtype=np.complex)
array([[ 1.+0.j,  2.+0.j,  3.+0.j,  4.+0.j],
       [ 4.+0.j,  5.+0.j,  6.+0.j,  7.+0.j],
       [ 7.+0.j,  8.+0.j,  9.+0.j, 10.+0.j]])
```

ndarray 类的形态操作方法

方法	描述
<code>ndarray.reshape(n,m)</code>	不改变数组 ndarray，返回一个维度为(n,m)的数组
<code>ndarray.resize(new_shape)</code>	与 <code>reshape()</code> 作用相同，直接修改数组 ndarray
<code>ndarray.swapaxes(ax1, ax2)</code>	将数组 n 个维度中任意两个维度进行调换
<code>ndarray.flatten()</code>	对数组进行降维，返回一个折叠后的一维数组
<code>ndarray.ravel()</code>	作用同 <code>np.flatten()</code> ，但是返回数组的一个视图

```
>>> x=np.array((1, 2, 3, 4))
>>> x
array([1, 2, 3, 4])
>>> y=x.reshape(2, 2)
>>> y
array([[1, 2],
       [3, 4]])
>>> z=x.reshape(4, 1)
>>> z
array([[1],
       [2],
       [3],
       [4]])
>>> z1=x.reshape(4, 2)
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    z1=x.reshape(4, 2)
ValueError: cannot reshape array of size 4 into shape (4,2)
```

```
>>> z2=x.flatten()
>>> z2
array([1, 2, 3, 4])
```

```
x=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
x
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
x.swapaxes(0, 1)
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

```
>>> x=np.array([[1, 2], [3, 4]])
>>> a=x.flatten()
>>> a
array([1, 2, 3, 4])
>>> a[0]=-1
>>> x
array([[1, 2],
       [3, 4]])
>>> b=x.ravel()
>>> b
array([1, 2, 3, 4])
>>> b[0]=-1
>>> x
array([[ -1,  2],
       [ 3,  4]])
```

ndarray 类的形态操作方法

```
>>> aArray = np.arange(1, 17)
>>> aArray
array([ 1,  2,  3, ..., 14, 15, 16])
>>> aArray.resize(4, 4)
>>> aArray
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

ndarray 类的形态操作方法

- 数组在numpy 中被当作对象，可以采用 `<a>.()` 方式调用一些方法。这里给出了改变数组基础形态的操作方法，例如改变和调换数组维度等。其中，`np.flatten()` 函数用于数组降维，相当于平铺数组中数据，该功能在矩阵运算及图像处理中用处很大。

ndarray 类的索引和切片方法

方法	描述
<code>x[i]</code>	索引第 <i>i</i> 个元素
<code>x[-i]</code>	从后向前索引第 <i>i</i> 个元素
<code>x[n:m]</code>	默认步长为 1，从前往后索引，不包含 <i>m</i>
<code>x[-m:-n]</code>	默认步长为 1，从后往前索引，结束位置为 <i>n</i>
<code>x[n,m,i]</code>	指定 <i>i</i> 步长的由 <i>n</i> 到 <i>m</i> 的索引

ndarray 类的索引和切片方法

- 数组切片得到的是原始数组的视图，所有修改都会直接反映到源数组。如果需要得到的ndarray 切片的一份副本，需要进行复制操作，比如`arange[5:8].copy()`

ndarray 类的索引和切片方法

```
a=np.random.rand(5,3)  #生成5x3的数组，用随机数填充
a[2]                    #获得第二行数据
array([0.38760211, 0.45506583, 0.26012577])
a[1:3]                  #获得第一行和第二行的数据
array([[0.67487581, 0.9070003 , 0.03162655],
       [0.38760211, 0.45506583, 0.26012577]])
a[-5:-2:2]             #获得第一行和第三行的数据
array([[0.67715899, 0.07594401, 0.69157123],
       [0.38760211, 0.45506583, 0.26012577]])
a
array([[0.67715899, 0.07594401, 0.69157123],
       [0.67487581, 0.9070003 , 0.03162655],
       [0.38760211, 0.45506583, 0.26012577],
       [0.47401063, 0.22122999, 0.86812716],
       [0.65095999, 0.73188333, 0.91540387]])
```

numpy 库的算术运算函数

函数	描述
<code>np.add(x1, x2 [, y])</code>	$y = x1 + x2$
<code>np.subtract(x1, x2 [, y])</code>	$y = x1 - x2$
<code>np.multiply(x1, x2 [, y])</code>	$y = x1 * x2$
<code>np.divide(x1, x2 [, y])</code>	$y = x1 / x2$
<code>np.floor_divide(x1, x2 [, y])</code>	$y = x1 // x2$, 返回值取整
<code>np.negative(x [,y])</code>	$y = -x$
<code>np.power(x1, x2 [, y])</code>	$y = x1^{**}x2$
<code>np.remainder(x1, x2 [, y])</code>	$y = x1 \% x2$

```
>>> a=np.array([2,3,1])
>>> b=np.array([1,2,3])
>>> c=np.array([5,6,7])
>>> np.add(a,b,c)
array([3, 5, 4])
>>> c
array([3, 5, 4])
>>> np.subtract(a,b,c)
...
array([ 1,  1, -2])
>>> c
...
array([ 1,  1, -2])
...

```

```
import numpy
dir(numpy)
['ALLOW_THREADS', 'AxisError', 'BUFSIZE', 'CLIP', 'ComplexWarning', 'DataSource', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_
IGNORE', 'ERR_LOG', 'ERR_PRINT', 'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVAL
ID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinity', 'MAXDIMS', 'MAY_SHARE_BOUNDS', 'MAY_SHARE_EXACT
', 'ModuleDeprecationWarning', 'NaN', 'NINF', 'NZERO', 'NaN', 'PINF', 'PZERO', 'RAISE', 'RankWarning', 'SHIFT_DIVI
DEBYZERO', 'SHIFT_INVALID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType', 'Tester', 'TooHardError', 'True_
', 'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'VisibleDeprecationWarning', 'WRAP', '_CopyMode', '_NoValue', '_UFUN
C_API', '_NUMPY_SETUP', '__all__', '__builtins__', '__cached__', '__config__', '__deprecated_attrs__', '__dir__
__doc__', '__expired_functions__', '__file__', '__getattr__', '__git_version__', '__loader__', '__name__', '__
package__', '__path__', '__spec__', '__version__', '__add_newdoc_ufunc__', '_distributor_init', '_financial_names', '_
globals_', '_mat', '_pyinstaller_hooks_dir', '_pytesttester', '_version', 'abs', 'absolute', 'add', 'add_docstring
', 'add_newdoc', 'add_newdoc_ufunc', 'all', 'allclose', 'alltrue', 'amax', 'amin', 'angle', 'any', 'append', 'appl
y_along_axis', 'apply_over_axes', 'arange', 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctan
h', 'argmax', 'argmin', 'argpartition', 'argsort', 'argwhere', 'around', 'array', 'array2string', 'array_equal', 'appl
array_equiv', 'array_repr', 'array_split', 'array_str', 'asanyarray', 'asarray', 'asarray_chkfinite', 'ascontiguou
sarray', 'asfarray', 'asfortranarray', 'asmatrix', 'atleast_1d', 'atleast_2d', 'atleast_3d', 'average', 'bartlett'
, 'base_repr', 'binary_repr', 'bincount', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor', 'blackman', '
block', 'bmat', 'bool8', 'bool', 'broadcast', 'broadcast_arrays', 'broadcast_shapes', 'broadcast_to', 'busday_cou
nt', 'busday_offset', 'busdaycalendar', 'byte', 'byte_bounds', 'bytes0', 'bytes_', 'c', 'can_cast', 'cast', 'cbrt
', 'cdouble', 'ceil', 'cfloat', 'char', 'character', 'chararray', 'choose', 'clip', 'clongdouble', 'clongfloat', '
column_stack', 'common_type', 'compare_chararrays', 'compat', 'complex128', 'complex64', 'complex_', 'complexfloat
ing', 'compress', 'concatenate', 'conj', 'conjugate', 'convolve', 'copy', 'copysign', 'copyto', 'core', 'corrcoeff
', 'correlate', 'cos', 'cosh', 'count_nonzero', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'cumproduct', 'c
umsum', 'datetime64', 'datetime_as_string', 'datetime_data', 'deg2rad', 'degrees', 'delete', 'deprecate', 'depreca
te_with_doc', 'diag', 'diag_indices', 'diag_indices_from', 'diagflat', 'diagonal', 'diff', 'digitize', 'disp', 'di
vide', 'divmod', 'dot', 'double', 'dsplit', 'dstack', 'dtype', 'e', 'ediff1d', 'einsum', 'einsum_path', 'emath', 'e
mpty', 'empty_like', 'equal', 'errstate', 'euler_gamma', 'exp', 'exp2', 'expand_dims', 'expm1', 'extract', 'eye', '
fabs', 'fastCopyAndTranspose', 'fft', 'fill_diagonal', 'find_common_type', 'finfo', 'fix', 'flatiter', 'flatnonze
ro', 'flexible', 'flip', 'fliplr', 'flipud', 'float16', 'float32', 'float64', 'float', 'float_power', 'floating', '
floor', 'floor_divide', 'fmax', 'fmin', 'fmod', 'format_float_positional', 'format_float_scientific', 'format_par
ser', 'frexp', 'from_dlpack', 'frombuffer', 'fromfile', 'fromfunction', 'fromiter', 'frompyfunc', 'fromregex', 'fr
```

numpy 库的算术运算函数

- 这些函数中，输出参数`y`可选，如果没有指定，将创建并返回一个新的数组保存计算结果；如果指定参数，则将结果保存到参数中。例如，两个数组相加可以简单地写为`a+b`，而`np.add(a,b,a)`则表示`a+=b`。

```
>>> a=np.array([1,2,3])
>>> b=np.array([4,5,6])
>>> np.add(a,b,a)
array([5, 7, 9])
>>> a
array([5, 7, 9])
```

numpy 库的比较运算函数

函数	符号描述
<code>np.equal(x1, x2 [, y])</code>	<code>y = x1 == x2</code>
<code>np.not_equal(x1, x2 [, y])</code>	<code>y = x1 != x2</code>
<code>np.less(x1, x2, [, y])</code>	<code>y = x1 < x2</code>
<code>np.less_equal(x1, x2, [, y])</code>	<code>y = x1 <= x2</code>
<code>np.greater(x1, x2, [, y])</code>	<code>y = x1 > x2</code>
<code>np.greater_equal(x1, x2, [, y])</code>	<code>y = x1 >= x2</code>
<code>np.where(condition[x,y])</code>	根据给出的条件判断输出 x 还是 y

```
>>> a= np.array([11, 12, 13])
>>> b= np.array([[11, 22, 23],[21, 22, 23]])
>>> print('a不等于b吗? {}'.format(np.not_equal(a, b)))
a不等于b吗? [[False True True]
 [ True True True]]
>>> print('a大于b吗? {}'.format(np.greater(a, b)))
a大于b吗? [[False False]
 [False False False]]
```

```
>>> np.equal(a, b, c)
... array([0, 0, 0])
>>> c
... array([0, 0, 0])
...
>>> np.not_equal(a, b, c)
... array([1, 1, 1])
... 
```

```
>>> a=np.array([2, 3, 1])
...
>>> b=np.array([1, 2, 3])
...
>>> c=np.array([5, 6, 7])
...
>>> d=np.array([True,False,True])
...
>>> np.equal(a, b, c)
... array([0, 0, 0])
>>> np.equal(a, b, d)
... array([False, False, False])
>>> np.equal(a, b)
... array([False, False, False])
... 
```

where函数

```
>>> x = np.random.random((3, 3))
>>> x
array([[0.89311886, 0.4587473, 0.04444645],
       [0.98066, 0.8144072, 0.98489384],
       [0.58542633, 0.21855129, 0.88590604]])
>>> np.where(x <= 0.5, -1, 1)
array([[ 1, -1, -1],
       [ 1,  1,  1],
       [ 1, -1,  1]])
```

numpy 库的比较运算函数

- 其将返回一个布尔数组，它包含两个数组中对应元素值的比较结果，例子如下。
- `where()`函数是三元表达式`x if condition else y`的矢量版本。

```
>>> np.less([1, 2], [2, 2])  
...  
array([ True,  False])  
>>> _.dtype  
...  
dtype('bool')
```

numpy 库的其他运算函数

函数	描述
<code>np.abs(x)</code>	计算基于元素的整形，浮点或复数的绝对值。
<code>np.sqrt(x)</code>	计算每个元素的平方根
<code>np.square(x)</code>	计算每个元素的平方
<code>np.sign(x)</code>	计算每个元素的符号：1(+), 0, -1(-)
<code>np.ceil(x)</code>	计算大于或等于每个元素的最小值
<code>np.floor(x)</code>	计算小于或等于每个元素的最大值
<code>np rint (x[, out])</code>	圆整,取每个元素为最近的整数,保留数据类型
<code>np.exp(x[, out])</code>	计算每个元素指数值
<code>np.log(x), np.log10(x), np.log2(x)</code>	计算自然对数(e),基于 10,2 的对数,log(1 + x)

```
>>> x=np. array(range(-10, 10, 5))
...
>>> x
...
array([-10,  -5,   0,   5])
>>> np. sign(x)
...
array([-1, -1,  0,  1])
...
```


■ 数组支持函数运算

```
>>> x = np.arange(0, 100, 10, dtype=np.float64)
```

```
>>> np.sin(x) #一维数组中所有元素求正弦值
```

```
array([ 0. , -0.54402111,  0.91294525, -0.98803162,  0.74511316,  
       -0.26237485, -0.30481062,  0.77389068, -0.99388865,  
        0.89399666])
```

```
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
>>> np.cos(b) #二维数组中所有元素求余弦值
```

```
array([[ 0.54030231, -0.41614684, -0.9899925 ],  
       [-0.65364362,  0.28366219,  0.96017029],  
       [ 0.75390225, -0.14550003, -0.91113026]])
```

```
>>> np.round(_) #四舍五入
```

```
array([[ 1., -0., -1.,  
        -1.,  0.,  1.,  
         1., -0., -1.]])
```

```
>>> x = np.random.rand(10)    #包含10个随机数的数组
```

```
>>> x = x*10
```

```
>>> x
```

```
array([6.03635335, 3.90542305, 0.05402166,  
       0.97778005, 8.86122047,  
        8.68849771, 8.43456386, 6.10805351,  
       1.01185534, 5.52150462])
```

```
>>> np.floor(x)                #所有元素向下取整
```

```
array([6., 3., 0., 0., 8., 8., 8., 6., 1., 5.])
```

```
>>> np.ceil(x)                #所有元素向上取整
```

```
array([7., 4., 1., 1., 9., 9., 9., 7., 2., 6.])
```

numpy简单应用

```
>>> np.zeros((3,3))
```

#全0二维数组

```
[[ 0.  0.  0.]  
 [ 0.  0.  0.]  
 [ 0.  0.  0.]
```

```
>>> np.zeros((3,1))
```

#全0一维数组

```
array([[ 0.]  
       [ 0.]  
       [ 0.]])
```

```
>>> np.zeros((1,3))
```

```
array([[ 0.,  0.,  0.]])
```

```
>>> np.ones((3,3))
```

#全1二维数组

```
array([[ 1.,  1.,  1.]  
       [ 1.,  1.,  1.]  
       [ 1.,  1.,  1.]])
```

numpy简单应用

```
>>> np.ones((1,3))    #全1一维数组
```

```
array([[ 1.,  1.,  1.]])
```

```
>>> np.identity(3)    #单位矩阵
```

```
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.]])
```

```
>>> np.identity(2)
```

```
array([[ 1.,  0.],  
       [ 0.,  1.]])
```

```
>>> np.empty((3,3))    #空数组，只申请空间而不初始化，元  
素值是不确定的
```

```
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

```
>>> np.empty((3,2))  
array([[0., 0.],  
       [0., 0.],  
       [0., 0.]])  
>>> np.empty((3,1))  
array([[1.23796180e-259],  
       [6.01347002e-154],  
       [1.16312002e+253]])  
... ..
```

numpy简单应用

■ 数组与数组的运算

```
>>> a = np.array((1, 2, 3))
```

```
>>> b = np.array([1, 2, 3], [4, 5, 6], [7, 8, 9])
```

```
>>> c = a * b
```

#数组与数组相乘

```
>>> c
```

#a中的每个元素乘以b中的每一列元素

```
array([[ 1,  4,  9],  
       [ 4, 10, 18],  
       [ 7, 16, 27]])
```

```
>>> c / b
```

#数组之间的除法运算

```
array([[ 1.,  2.,  3.],  
       [ 1.,  2.,  3.],  
       [ 1.,  2.,  3.]])
```

```
>>> c / a
```

```
array([[ 1.,  2.,  3.],  
       [ 4.,  5.,  6.],  
       [ 7.,  8.,  9.]])
```

numpy简单应用

```
>>> a + a
```

```
array([2, 4, 6])
```

```
>>> a * a
```

```
array([1, 4, 9])
```

```
>>> a - a
```

```
array([0, 0, 0])
```

```
>>> a / a
```

```
array([ 1.,  1.,  1.])
```

#数组之间的加法运算

#数组之间的乘法运算

#数组之间的减法运算

#数组之间的除法运算

numpy简单应用

■ 转置

```
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
>>> b
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
>>> b.T
```

#转置

```
array([[1, 4, 7],  
       [2, 5, 8],  
       [3, 6, 9]])
```

```
>>> a = np.array((1, 2, 3, 4))
```

```
>>> a
```

```
array([1, 2, 3, 4])
```

```
>>> a.T
```

#一维数组转置以后和原来是一样的

```
array([1, 2, 3, 4])
```

numpy简单应用

■ 点积

```
>>> a = np.array((5, 6, 7))
>>> b = np.array((6, 6, 6))
>>> a.dot(b)
```

108

```
>>> np.dot(a,b)
```

108

```
>>> c = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
>>> cT = c.T
```

```
>>> c.dot(a)
```

```
array([ 38, 92, 146])
```

```
>>> c[0].dot(a)
```

38

```
>>> c[1].dot(a)
```

92

```
>>> c[2].dot(a)
```

146

#向量内积

```
>>> c
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> cT
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
>>> a
array([5, 6, 7])
```

#二维数组

#转置

#二维数组的每行与一维向量计算内积

#两个一维向量计算内积

numpy简单应用

- 数组元素访问

```
>>> b = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
>>> b
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
>>> b[0]
```

#第0行

```
array([1, 2, 3])
```

```
>>> b[0][0]
```

#第0行第0列的元素值

```
1
```

numpy简单应用

■ 数组多元素同时访问

```
>>> x = np.arange(0,100,10,dtype=float)
```

```
>>> x
```

```
array([ 0., 10., 20., 30., 40., 50., 60., 70., 80., 90.])
```

```
>>> x[[1, 3, 5]]
```

```
array([ 10., 30., 50.])
```

```
>>> x[1,3,5]
...
Traceback (most recent call last):
  File "<pyshell#89>", line 1, in <module>
    x[1,3,5]
IndexError: too many indices for array: array is 1-dimensional, but 3 were indexed
>>> x[1]
...
10.0
```

```
>>> x=np.array([[1, 2, 3], [4, 5, 6]])
>>> x
...
array([[1, 2, 3],
       [4, 5, 6]])
>>> x[1,2]
...
6
```

numpy简单应用

■ 矩阵不同维度上的计算

```
>>> x = np.arange(0,10).reshape(2,5)
```

#创建二维数组

```
>>> x
```

```
array([[0, 1, 2, 3, 4],  
       [5, 6, 7, 8, 9]])
```

```
>>> np.sum(x)
```

#二维数组所有元素求和

```
45
```

```
>>> np.sum(x, axis=0)
```

#二维数组纵向求和

```
array([ 5, 7, 9, 11, 13])
```

```
>>> np.sum(x, axis=1)
```

#二维数组横向求和

```
array([10, 35])
```

```
>>> np.mean(x, axis=0)
```

#二维数组纵向计算算术平均值

```
array([ 2.5, 3.5, 4.5, 5.5, 6.5])
```

```
>>> weight = [0.3, 0.7]
```

#权重

```
>>> np.average(x, axis=0, weights=weight)
```

#二维数组纵向计算加权平均值

```
array([ 3.5, 4.5, 5.5, 6.5, 7.5]) 0*0.3+5*0.7 1*0.3+6*0.7
```

numpy简单应用

```
>>> np.max(x)
```

#所有元素最大值

```
9
```

```
>>> np.max(x, axis=0)
```

#每列元素的最大

值

```
array([5, 6, 7, 8, 9])
```

```
>>> x = np.random.randint(0, 10, size=(3,3))
```

#创建二维数组

```
>>> x
```

```
array([[4, 9, 1],
```

```
       [7, 4, 9],
```

```
       [8, 9, 1]])
```

```
>>> np.std(x)
```

#所有元素标准差

```
3.1544599036840864
```

```
>>> np.std(x, axis=1)
```

#每行元素的标准

差

```
array([3.29983165, 2.05480467, 3.55902608])
```

```
>>> x
```

```
array([[0, 1, 2, 3, 4],
```

```
       [5, 6, 7, 8, 9]])
```

numpy简单应用

```
>>> np.var(x, axis=0)
```

#每列元素的标准

差

```
array([2.88888889, 5.55555556, 14.22222222])
```

```
>>> np.sort(x, axis=0)
```

#纵向排序

```
array([[4, 4, 1],  
       [7, 9, 1],  
       [8, 9, 9]])
```

```
>>> np.sort(x, axis=1)
```

#横向排序

```
array([[1, 4, 9],  
       [4, 7, 9],  
       [1, 8, 9]])
```

```
>>> x  
  
array([[4, 9, 1],  
       [7, 4, 9],  
       [8, 9, 1]])
```

第0轴

第1轴



4	9	1
7	4	9
8	9	1

numpy简单应用

■ 改变数组大小

```
>>> a = np.arange(1, 11, 1)
```

```
>>> a
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
>>> a.shape = 2, 5
```

#改为2行5列

```
>>> a
```

```
array([[ 1,  2,  3,  4,  5],  
       [ 6,  7,  8,  9, 10]])
```

```
>>> a.shape = 5, -1
```

#-1表示自动计算

```
>>> a
```

```
array([[ 1,  2],  
       [ 3,  4],  
       [ 5,  6],  
       [ 7,  8],  
       [ 9, 10]])
```

```
>>> b = a.reshape(2,5)
```

#reshape()方法返回新数组

```
>>> b
```

```
array([[ 1,  2,  3,  4,  5],  
       [ 6,  7,  8,  9, 10]])
```

```
>>> a  
array([1, 3, 5, 7])  
>>> a.shape=2,-1  
>>> a  
array([[1, 3],  
       [5, 7]])  
\>>> |
```

```
>>> a=np.array([1,3,5,7])  
...  
>>> a.shape=3,-1  
...  
Traceback (most recent call last):  
  File "<pyshell#104>", line 1, in <module>  
    a.shape=3,-1  
ValueError: cannot reshape array of size 4 into shape (3,newaxis)
```

numpy简单应用

■ 切片操作

```
>>> a = np.arange(10)
```

```
>>> a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
>>> a[::-1]
```

#反向切片

```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
>>> a[::2]
```

#隔一个取一个元素

素

```
array([0, 2, 4, 6, 8])
```

```
>>> a[:5]
```

#前5个元素

```
array([0, 1, 2, 3, 4])
```

numpy简单应用

```
>>> c = np.arange(25)      #创建数组
>>> c.shape = 5,5          #修改数组大小
>>> c
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
>>> c[0, 2:5]              #第0行中下标[2,5)之间的元素值
array([2, 3, 4])
>>> c[1]                   #第0行所有元素
array([5, 6, 7, 8, 9])
>>> c[2:5, 2:5]            #行下标和列下标都介于[2,5)之间的元素值
array([[12, 13, 14],
       [17, 18, 19],
       [22, 23, 24]])
```

```
>>> c = np.arange(25)
...
>>> c[2,3,4]
...
Traceback (most recent call last):
  File "<pyshell#106>", line 1, in <module>
    c[2,3,4]
IndexError: too many indices for array: array is 1-dimensional, but 3 were indexed
>>> c[[2,3,4]]
...
array([2, 3, 4])
```


numpy简单应用

■ 布尔运算

```
>>> x = np.random.rand(10)          #包含10个随机数的数组
>>> x
array([ 0.56707504, 0.07527513, 0.0149213 , 0.49157657, 0.75404095,
        0.40330683, 0.90158037, 0.36465894, 0.37620859, 0.62250594])
>>> x > 0.5                          #比较数组中每个元素值是否大于0.5
array([ True, False, False, False,  True, False,  True, False, False,  True], dtype=bool)
>>> x[x>0.5]                        #获取数组中大于0.5的元素
array([ 0.56707504, 0.75404095, 0.90158037, 0.62250594])
>>> a = np.array([1, 2, 3])
>>> b = np.array([3, 2, 1])
>>> a > b                            #两个数组中对应位置上的元素比较
array([False, False,  True], dtype=bool)
>>> a[a>b]
array([3])
>>> a == b
array([False,  True, False], dtype=bool)
>>> a[a==b]
array([2])
```

numpy简单应用

■ 去整运算

```
>>> x = np.random.rand(10)*50
```

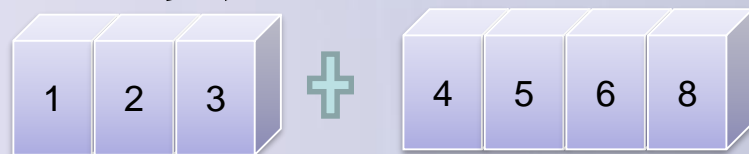
```
>>> x
```

```
array([ 0.69708323, 14.99931488, 15.04431214, 24.60547929,  
       12.12020273, 42.72638176, 16.01128916, 38.91558471,  
       39.6877989 , 21.98678429])
```

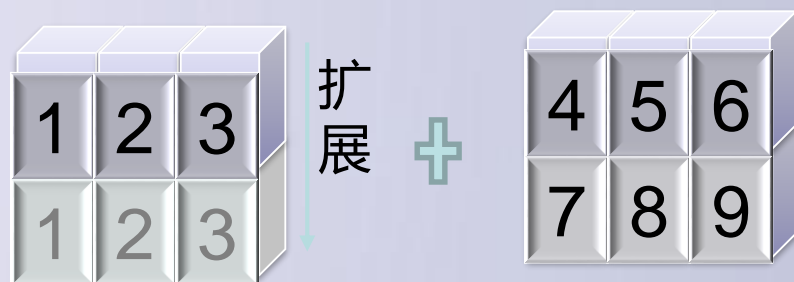
```
>>> np.array([t-int(t) for t in x])
```

```
array([ 0.69708323, 0.99931488, 0.04431214, 0.60547929,  
       0.12020273,  
       0.72638176, 0.01128916, 0.91558471, 0.6877989 , 0.98678429])
```

ndarray的运算



ValueError: non-broadcastable output operand with shape (3) doesn't match the broadcast shape (4,)



=



广播功能

较小的数组会广播到较大数组的大小，使它们的形状兼容

Source

```
>>> a = np.array([1,2,3])
>>> b =
np.array([[4,5,6],[7,8,9]])
>>> a + b
array([[5, 7, 9],
       [8, 10, 12]])
```

numpy简单应用

■ 广播

```
>>> a = np.arange(0,60,10).reshape(-1,1)           #列向量
>>> b = np.arange(0,6)                             #行向量
>>> a
array([[ 0],
       [10],
       [20],
       [30],
       [40],
       [50]])
>>> b
array([0, 1, 2, 3, 4, 5])
```

numpy简单应用

```
>>> a + b
```

#广播

```
array([[ 0,  1,  2,  3,  4,  5],  
       [10, 11, 12, 13, 14, 15],  
       [20, 21, 22, 23, 24, 25],  
       [30, 31, 32, 33, 34, 35],  
       [40, 41, 42, 43, 44, 45],  
       [50, 51, 52, 53, 54, 55]])
```

```
>>> a * b
```

```
array([[ 0,  0,  0,  0,  0,  0],  
       [ 0, 10, 20, 30, 40, 50],  
       [ 0, 20, 40, 60, 80, 100],  
       [ 0, 30, 60, 90, 120, 150],  
       [ 0, 40, 80, 120, 160, 200],  
       [ 0, 50, 100, 150, 200, 250]])
```

```
>>> a
```

```
array([[ 0],  
       [10],  
       [20],  
       [30],  
       [40],  
       [50]])
```

```
>>> b
```

```
array([0, 1, 2, 3, 4, 5])
```

numpy简单应用

■ 分段函数

```
>>> x = np.random.randint(0, 10, size=(1,10))
>>> x
array([[0, 4, 3, 3, 8, 4, 7, 3, 1, 7]])
>>> np.where(x<5, 0, 1)    #小于5的元素值对应0, 其他对应1
array([[0, 0, 0, 0, 1, 0, 1, 0, 0, 1]])
#小于4的元素乘以2, 大于7的元素乘以3, 其他元素变为0
>>> np.piecewise(x,[x<4, x>7],[lambda x:x*2,lambda x:x*3])
array([[ 0,  0,  6,  6, 24,  0,  0,  6,  2,  0]])
```