# 文件操作

# 学习目标



- 掌握文件的基本操作
- 掌握二进制文件的处理
- 了解文件的系统操作
- 了解常见文件格式的读写

# 文件分类

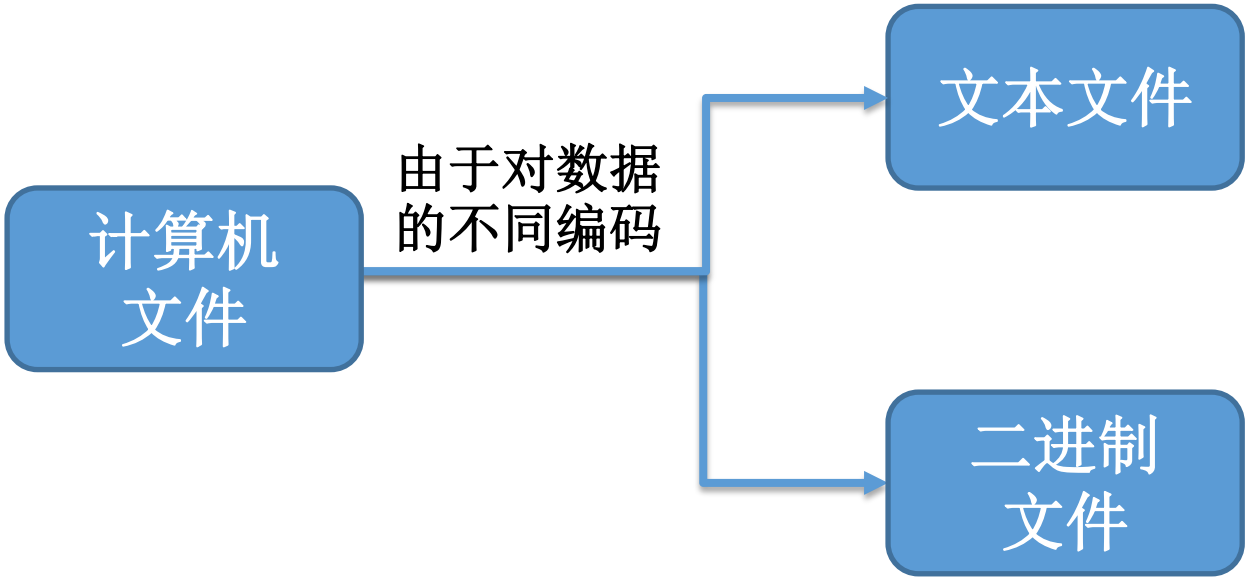


按文件中数据的组织形式把文件分为文本文件和二进制文件两类。

文本文件：存储的是常规字符串，由若干文本行组成，每行以换行符“\n”结尾。可以使用文本编辑器进行显示、编辑并且能够直接阅读。如网页文件、记事本文件、程序源代码文件等。

二进制文件：存储的是字节串bytes，需要使用专门的软件进行解码后才能读取、显示、修改或执行。如图形图像文件、音视频文件、可执行文件、数据库文件等。

# 文件分类




文本文件可以用普通文本编辑器直接查看，通常是以字符串形式存在的字母、汉字和数字等。

二进制文件则呈现了各种形态，包括字处理文件、可执行文件、图像文件、音视频文件等。

# 打开文件



- 向（从）一个文件写（读）数据之前，需要先创建一个和物理文件相关的文件对象，然后通过该文件对象对文件内容进行读取、写入、删除、修改等操作，最后关闭并保存文件内容。
- Python内置的open()函数可以按指定的模式打开指定的文件并创建文件对象。打开一个文件并创建文件对象的语法如下。

  file_object = open(file, mode='r', buffering=-1, encoding)

  open函数打开文件file，返回一个指向文件file的文件对象file_object。

# 打开文件



- **file_object = open(file, mode='r', buffering=-1, encoding)**

参数的说明如下：

**filename**：要访问的文件名称。 **file**是一个包含文件所在路径及文件名称的字符串值，如**'c:\\User\\test.txt'**。

**mode**：指定打开文件后的处理方式：只读，写入，追加等。这个参数是非强制的，默认文件访问模式为只读**(r)**。

**buffering**：**0**表示不缓存，**1**表示缓存。大于**1**表示缓冲区的大小。**-1**表示缓冲区的大小为系统默认值。

**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。

# 打开文件



- **Python 使用open()方法打开一个文件，并返回一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。如果文件不存在、访问权限不够、磁盘空间不足或其它原因导致创建文件对象失败，open()函数就会抛出一个IOError的错误，并且给出错误码和详细的信息。**

# 文件读写



- 磁盘满、无法写入，打开文件要读取但文件不存在，或者文件路径错误。这些是文件读写过程中随时可能遇到的问题。
- 凡是涉及文件输入输出的操作，这类问题在程序设计时是必然要考虑的因素，否则程序的设计并不完整和严谨。

# 文件对象



## ➢ 文件打开模式

打开模式定义了打开文件后的处理方式，如只读、读写、追加等。

| 模 式 | 功能描述 |
| --- | --- |
| r | 只读模式 |
| w | 写模式，若存在则覆盖原有内容 |
| a | 追加模式 |
| r+ | 读写，不创建 |
| w+ | 读写，若不存在则新建 |
| a+ | 追加模式但可读 |
| rt, wt | 默认为文本方式，相当于r, w |
| rb, wb | 读写二进制文件 |

# 打开文件



文件打开的不同模式：

| 模式 | 描述 |
| --- | --- |
| r | 以只读方式打开文件，文件的指针放在文件的开头。这是默认模式，可省略。 |
| r+ | 以读写格式打开一个文件，文件指针放在文件的开头。 |
| w | 以写入格式打开一个文件，如果该文件已存在，则将其覆盖。如果该文件不存在，则创建新文件。 |
| w+ | 以读写格式打开一个文件，如果该文件已存在则将其覆盖。如果该文件不存在，则创建新文件。 |
| a | 以追加格式打开一个文件，如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| a+ | 以读写格式打开一个文件，如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。 |

# 打开文件



一个文件被打开后，返回一个文件对象，通过文件对象可以得到有关该文件的各种信息。文件对象的属性如表所示：

| 属性 | 描述 |
| --- | --- |
| **closed** | 判断文件是否关闭，如果文件已被关闭，返回**True**；否则返回**False** |
| **mode** | 返回被打开文件的访问模式 |
| **name** | 返回文件的名称 |

# 打开文件



一个文件被打开后，返回一个文件对象，通过文件对象可以得到有关该文件的各种信息。文件对象的属性举例如下：

In [29]:
```
file_object=open('c:\\test\\scores.txt', 'r')
print('文件名: ', file_object.name)
print('是否已关闭 : ',file_object.closed)
print('访问模式 : ', file_object.mode)
```

```
文件名:  c:\test\scores.txt
是否已关闭 :  False
访问模式 :  r
```

# 文件对象



文件对象的常用方法如表所示：

| 方法 | 功能说明 |
| --- | --- |
| close() | 刷新缓冲区里还没写入的信息，并关闭该文件 |
| flush() | 刷新文件内部缓冲，把内部缓冲区的数据立刻写入文件，但不关闭文件 |
| next() | 返回文件下一行 |
| read([size]) | 从文件的开始位置读取指定的size个字符数，如果未给定则读取所有 |
| readline() | 读取整行，包括"\n"字符 |
| readlines() | 把文本文件中的每行文本作为一个字符串存入列表中并返回该列表 |

# 文件对象



文件对象的常用方法如表所示：

| 方法 | 功能说明 |
| --- | --- |
| seek(offset[,from]) | 用于移动文件读取指针到指定位置，offset为需要移动的字节数；from指定从哪个位置开始移动，默认值为0，0代表从文件开头开始，1代表从当前位置开始，2代表从文件末尾开始。 |
| tell() | 方法返回文件的当前位置，即文件指针当前位置 |
| truncate( [ size ]) | 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个字符，其余的删除 |
| write(str ) | 把字符串str的内容写入文件中，没有返回值。由于缓冲，字符串内容可能没有加入到实际的文件中，直到调用 flush() 或close()方法被调用。 |
| writelines( [str ]) | 用于向文件中写入字符串序列 |
| writable() | 测试当前文件是否可写 |
| readable() | 测试当前文件是否可读 |

# 文件对象



文件的关闭：

- 当对文件内容操作完成以后，必须关闭文件，这样能够保证所做的修改得到保存。
- close() 方法用于关闭一个已打开的文件，可以将缓冲的数据写入文件，然后关闭文件。关闭后的文件不能再进行读写操作，否则会触发 ValueError 错误。close()方法允许调用多次。
- 而flush()方法将缓冲的数据写入文件，但是不关闭文件。

# 文件对象

- 使用上下文管理语句with可以自动管理资源，在代码块执行完毕后自动还原进入该代码块之前的现场或上下文。
  - with语句的语法如下：

```
with context_expr [as var]:
    with语句块
```

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

# 文件对象



文件的关闭：

- 使用with语句的好处：
- 使用with自动关闭资源，可以在代码块执行完毕后还原进入该代码块时的现场。
- 不论何种原因跳出with块，不论是否发生异常，总能保证文件被正确关闭，资源被正确释放。

# 文件对象



写文件：

当一个文件以“写”的方式打开后，可以使用write()方法和writelines()方法，将字符串写入文本文件。

➢1. write()方法：

write(s) 方法用于向一个打开的文件中写入指定的字符串。在文件关闭前或缓冲区刷新前，字符串内容存储在缓冲区中，这时在文件中是看不到写入的内容的。需要重点注意的是，write()方法不会在字符串的结尾添加换行符“\n”。

write()方法语法格式如下：fileObject.write(str)

参数str：要写入文件的字符串。

返回值：返回的是写入的字符长度。

在操作文件时，每调用一次write()方法，写入的数据就会追加到文件末尾。

# 文件对象



写文件：

1. write()方法

➢程序运行后，会在程序当前的路径下，生成一个名为test.txt文件，打开该文件，可以看到数据成功被写入。

➢注意：当向文件写入数据时，如果文件不存在，那么系统会自动创建一个文件并写入数据。如果文件存在，那么会清空原来文件的数据，重新写入新数据。

In [35]:
```
fp= open("test.txt", "w")                    #以只写方式打开文本文件
fp.write("My name is Guido van Rossum!\n")
fp.write("I invented the Python programming language!\n")
fp.write("I love Python!\n")
fp.close()
```

# 文件对象



写文件：

2. writelines()方法

➢writelines()方法把字符串列表写入文本文件，不添加换行符“\n”。

In [43]:
```
data = ['My ','name ','is ','John!\n','How ','are ','you!']    #生成要写入的列表内容
with open("data_desc.txt","w") as fp:
    fp.writelines(data)
with open("data_desc.txt","r") as fp:
    for line in fp:
        print(line)
```

```
My name is John!

How are you!
```

# 文件对象



读文件：

当一个文件被打开后，可使用三种方式从文件中读取数据：read()、readline()、readlines()。

- read([size])：从文件读取指定的size个字符数，如果未给定则读取所有。
- readline()：读取整行。
- readlines()：把文本文件中的每行文本作为一个字符串存入列表中并返回该列表。

注意：打开一个已经存在的文件，读取正常，当试图打开一个不存在的文件，系统会给出错误信息。

# 文件对象



读文件：

Python文件对象提供了三个“读”方法： read()、readline()和 readlines()。每种方法可以接受一个变量以限制每次读取的数据数量。

1. read()方法

read()方法从文件当前位置起读取size个字符串，若无参数size，则表示读取至文件结束为止。如果使用多次，那么后面读取的数据是从上次读完后的位置开始。

read()语法结构如下：fileObject.read(size)

参数size ：从文件中读取的字符数。如果没有指定字符数，那么就表示读取文件的全部内容。返回值：返回从字符串中读取的字符内容。

# 文件对象



读文件：

1. read()方法

In [44]:

```
fp = open("test.txt", "r")          #只读方式打开文本文件
content = fp.read(10)               #读取10个字节数据
print(content)
```

```
My name is
```

# 文件对象



读文件：

2. readline()方法

该方法每次读出一行内容，该方法读取时占用内存小，比较适合大文件，该方法返回一个字符串对象。

readline()语法：fileObject. readline()

返回值：返回读取的字符串。

# 文件对象



## 读文件：readline()方法案例：

```
fp= open("test1.txt", "r")     #只读方式打开文件
line = fp.readline()
print("读取第一行:%s" % (line))
print("-----------华丽的分割线-------------")
while line:                          #循环读取每一行
    print(line)
    line = fp.readline()
fp.close()
print( "文件", fp.name, "已经成功分行读出！")
```

```
读取第一行:My name is Guido van Rossum!

-----------华丽的分割线-------------
My name is Guido van Rossum!

I invented the Python programming language!

I love Python!
文件 test1.txt 已经成功分行读出！
```

# 文件对象



读文件：

3. readlines()方法

readlines()方法读取文件的所有行，保存在一个列表中，每行作为一个元素，但读取大文件会比较占内存。该列表可以由 Python 的 for... in ... 结构进行处理。

readlines()语法：fileObject.readlines()

返回值：此方法返回包含所有行的列表。

# 文件对象

软件学院 School of Software

## 读文件：readlines()方法案例：

In [46]:

```
fp= open("test.txt", "r")    #只读方式打开文件
lines = fp.readlines()
print("行的数据类型:", type(lines))
print(("列表形式存放每一行: %s" %(lines)))
print("----------------------分割线----------------")
for line in lines: #依次读取每行
    line = line.strip()      #去掉每行头尾空白符，包括'\n'
    print("读取的数据为: %s" % (line))
fp.close()
print( "文件", f.name, "已经成功把所有行读出！")
```

```
行的数据类型: <class 'list'>
列表形式存放每一行: ['My name is Guido van Rossum!\n', 'I invented the Python programming language!\n', 'I love Python!\n']
----------------------分割线----------------
读取的数据为: My name is Guido van Rossum!
读取的数据为: I invented the Python programming language!
读取的数据为: I love Python!
文件 test.txt 已经成功把所有行读出!
```

# 文件对象



读文件：用readlines()方法读取一段中文诗词并打印。

- Windows平台下open()函数在打开文件的时候缺省的编码（encoding）为gbk(cp936)，并不是UTF-8，因此在打开文件的时候应指定编码为UTF-8，否则读取文件会出现错误。

In [48]:

```
fname = r'C:\test\shi.txt'
with open(fname,'r' , encoding='utf-8') as f:
    lines = f.readlines()
    for line in lines: print(line.rstrip())
```

```
日照香炉生紫烟
遥看瀑布挂前川
飞流直下三千尺
疑是银河落九天
```

# 文件对象



读文件：

4. 从文件逐行读取数据：

In [47]:
```
with open('test.txt') as f:
    for line in f:
        print(line,end='')
```

```
My name is Guido van Rossum!
I invented the Python programming language!
I love Python!
```

# 文件对象



文件指针的定位：

- 文件对象的tell()方法返回文件的当前位置，即文件指针当前位置。
- 使用文件对象的read()方法读取文件之后，文件指针到达文件的末尾，如果再来一次read()将会发现读取的是空内容，如果想再次读取全部内容，或读取文件中的某行字符，必须将文件指针移动到文件开始或某行开始，这可通过文件对象的seek()方法来实现，其语法格式如下：

```
seek(offset[,whence])
```

说明：用于移动文件读取指针到指定位置，offset为需要移动的字节数；whence指定从哪个位置开始移动，默认值为0，0代表从文件开头开始，1代表从当前位置开始，2代表从文件末尾开始。

**注意：Python3不允许非二进制打开的文件，相对于文件末尾的定位**

# 文件对象



- 下面我们尝试在文件后附加一段文本的同时，还进行文件内容的读取。选取之前的已写入部分生成的test.txt。
- 此时的运行结果显示s为空串，而通过tell()方法查看文件指针，可以看出，文件的指针在该文件的末尾。

In [9]:
```
with open('test.txt','a+') as f:
    s = f.read(5)
    print('in file -> ', s)
    point=f.tell()
    print(point)
```

```
in file ->
91
```

# 文件对象



- 重改程序后。

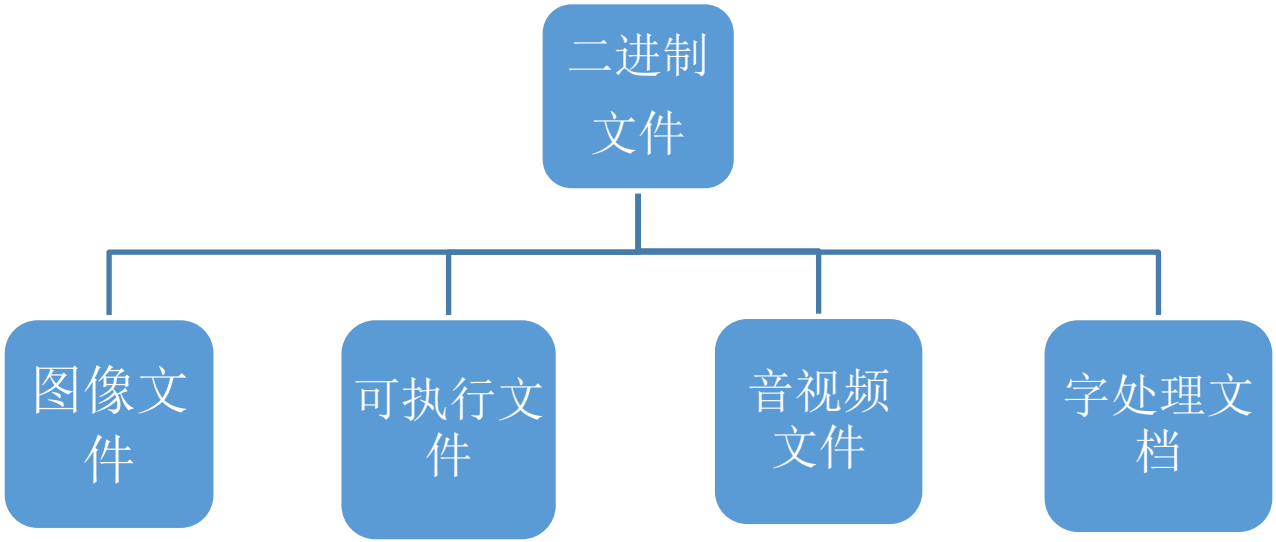
In [22]:

```
with open('test.txt','a+') as f:
    f.write('\n Python is a programming language. \n')
    f.flush()       # 清空缓冲区，确保数据保存到文件
    f.seek(0)       # 将文件指针转移到文件首部
    s = f.readlines()
    for line in s:      # 逐行读取文件数据
        print(line)
```

```
My name is Guido van Rossum!

I invented the Python programming language!

I love Python!


 Python is a programming language.
```

## ➢ 二进制文件

二进制文件包括图像文件、可执行文件、音视频文件、字处理文档等，不能使用记事本或其他文本编辑软件直接读写。



# 读写二进制数据



## ➢ 二进制文件的读写

二进制文件读写的是bytes字节串。运行下面程序，由于写入的是一个字符串，非字节串，系统会抛出异常。

In [23]:
```
with open("test.bt","wb") as fp:
    fp.write("abcd")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-23-1097f6556c11> in <module>
      1 with open("test.bt","wb") as fp:
----> 2     fp.write("abcd")

TypeError: a bytes-like object is required, not 'str'
```

# 读写二进制数据



## ➢ 读写二进制文件

表 2-7-2 二进制文件模式

| 文件使用方式 | 意 义 |
| --- | --- |
| rb | 只读打开一个二进制文件，只允许读数据。如文件存在，则打开后可以顺序读；如文件不存在，则打开失败。 |
| wb | 只写打开或建立一个二进制文件，只允许写数据。如文件不存在，则建立一个空文件；如文件已经存在，则把原文件内容清空。 |
| ab | 追加打开一个文本文件，并在文件末尾写数据。如文件不存在，则建立一个空文件；如文件已经存在，则把原文件打开，并保持原内容不变，文件位置指针指向末尾，新写入的数据追加在文件末尾。 |

# 读写二进制数据



## ➢ 读写二进制文件

➢ 因为二进制文件是字节流，因此二进制文件在打开读写时不用指定编码，也不存在readline,readlines读一行或者多行的操作函数。一般二进制文件使用read函数读取，使用write函数写入。

**1. 写二进制文件**

如果要把二进制数据data写入文件，则使用write函数，格式：

文件对象.write(data)

# 读写二进制数据



## ➢ 读写二进制文件

**2. 读二进制文件**

**文件对象.read()**

**文件对象.read(n)**

- **如果不指定要读取的字符数n，使用read()读，则读到整个文件的内容。**
- **如果使用read(n)指定要读取的字符数，要么就按要求读取n个字符；如果要读n个字符，而文件没有那么多字符，那么就读取所有文件内容。**

# 读写二进制数据



## ➢ 读写二进制文件例子

In [22]:

```
def writeFile():
        fobj=open('abc.txt','wb')
        fobj.write('Python文件'.encode())
        fobj.close()

def readFile():
        fobj=open('abc.txt','rb')
        data=fobj.read()
        print(data.decode())
        fobj.close()

writeFile()
readFile()
```

```
Python文件
```

# 读写二进制数据



## ➢ 读写二进制文件例子

In [43]:
```
with open('myModule.py','rb') as f:
    data=f.read()
    print(data.decode())
```

```
def myMin(a,b):
    c=a
    if a>b:
        c=b
        return c
def myMax(a,b):
    c=a
    if a<b:
        c=b
        return c
```

# 读写二进制数据



## ➢ 读写二进制文件

- 二进制文件读写的是bytes字节串。示例如下：

```
with open("test.bt","wb") as fp:
    fp.write("abcd")        #产生异常，需要转换成bytes
```

运行该程序，由于写入的是一个字符串，非字节串，系统会抛出异常，信息如下：

```
TypeError        Traceback (most recent call last)
TypeError: a bytes-like object is required, not 'str'
```

# 读写二进制数据



## ➢ 读写二进制文件

- 修改程序如下：

In [12]:

```
with open("test.bt","wb+") as fp:
    fp.write(bytes("我爱中国".encode("utf-8")))      #转换成使用utf-8编码
    fp.seek(0)                                      #文件指针定位到开头
    b = fp.read().decode("utf-8")                   #解码方式和编码方式要一致
    print(b)
```

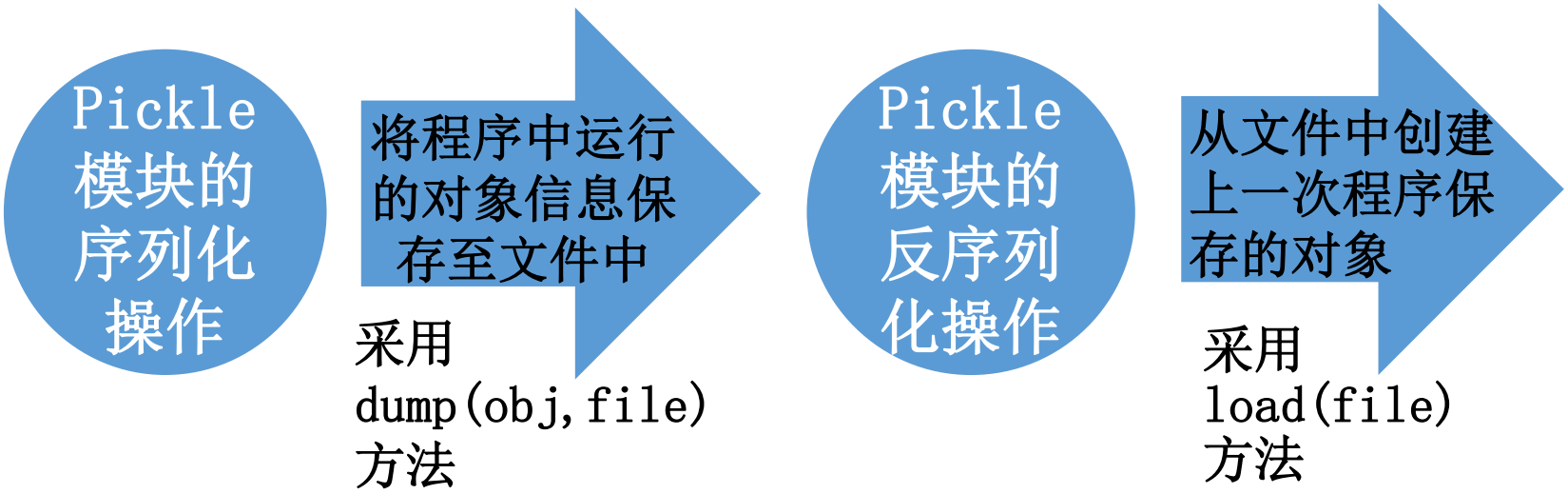
```
我爱中国
```

- 可以看出如果直接用文本文件或二进制文件格式存储Python中的各种对象，通常需要进行繁琐的转换。

# 读写二进制数据

Python程序在内存中的数据一般是放置在列表、元组、字典等各类对象之中。当进行文件保存或网络处理时，不能直接送入这些对象本身，必须将这些对象进行序列化，以转化为字节码才能进行处理。

pickle是一个常用且效率较高的二进制文件序列化模块，属于python语言的标准组件，不需要单独安装。



# 读写二进制数据



## ➢ 用pickle模块读写二进制文件例子。

In [14]:

```
import pickle       #导入pickle模块
name = "张三"
age = 20
scores = [65,70,76,80]
with open("test.bt","wb+") as fp:    #以读写方式打开二进制文件
    pickle.dump(name,fp)         #写入文件
    pickle.dump(age,fp)
    pickle.dump(scores,fp)
    fp.seek(0)           #将文件指针移动到文件开头
    print(fp.read())     #读出文件的全部内容，返回一个字节串
    fp.seek(0)
    name = pickle.load(fp)    #读取文件
    age = pickle.load(fp)
    scores = pickle.load(fp)
    print(name,";",age,";",scores)
```

```
b'\x80\x04\x95\n\x00\x00\x00\x00\x00\x00\x8c\x06\xe5\xbc\xa0\xe4\xb8\x89\x94.\x80\x04K\x14.\x80\x04\x95\r\x00\x00\x00\x00\x00\x00\x00]\x94(KAKFKLKPe.'
张三 ; 20 ; [65, 70, 76, 80]
```

# 读写二进制数据



## ➢ 用pickle模块读写二进制文件例子。

在当前目录下生成一个student.dat的文件。

In [15]:

```
import pickle
students = []          # 建立一个学生列表
students.append({'学号':'01', '姓名':'王五', '平时':80, '实验':60, '期末': 74})
students.append({'学号':'02', '姓名':'张三', '平时':68, '实验':83, '期末': 79})
students.append({'学号':'03', '姓名':'李四', '平时':73, '实验':75, '期末': 85})
students.append({'学号':'04', '姓名':'孙六', '平时':87, '实验':69, '期末': 63})
ratio = [0.3, 0.2, 0.5] # 平时成绩占比0.3, 实验占比0.2, 期末占比0.5

with open('student.dat','wb') as f:
    pickle.dump(students,f)
    pickle.dump(ratio,f)
```

# 读写二进制数据



## ➢ 用pickle模块读写二进制文件例子。

读取学生成绩文件，并计算出总评成绩 。

In [16]:

```
import pickle
with open('student.dat','rb') as f:
    students = pickle.load(f)
    ratio = pickle.load(f)
    for s in students:
        s['总评'] = s['平时']*ratio[0]+s['实验']*ratio[1]+s['期末']*ratio[2]
        print(s)
```

```
{'学号': '01', '姓名': '王五', '平时': 80, '实验': 60, '期末': 74, '总评': 73.0}
{'学号': '02', '姓名': '张三', '平时': 68, '实验': 83, '期末': 79, '总评': 76.5}
{'学号': '03', '姓名': '李四', '平时': 73, '实验': 75, '期末': 85, '总评': 79.4}
{'学号': '04', '姓名': '孙六', '平时': 87, '实验': 69, '期末': 63, '总评': 71.4}
```

## os、os.path与shutil模块

os模块除了提供使用操作系统功能和访问文件系统的简便方法外，还提供了大量文件和目录操作的方法。

### 1、文件的重命名

os.rename() 方法用于重命名文件或目录，如果dst是一个存在的目录, 将抛出OSError。

rename()方法的语法格式如下：os.rename(src, dst)；参数：src 是要修改的目录名， dst 是修改后的目录名，返回值：该方法没有返回值

### 2、文件的删除

os.remove() 方法用于删除指定路径的文件。如果指定的路径是一个目录，将抛出OSError。remove()方法语法格式：os.remove(path)

参数：path 是要移除的文件路径

返回值：该方法没有返回值

# 文件系统操作



- **os、os.path与shutil模块**
- **下面通过一个案例来演示rename()和remove()方法的应用。**

In [2]:

```
import os           #导入os包
print("目录为: %s"%os.listdir(os.getcwd()))     #列出当前目录下的文件和子目录
os.rename("test.txt","test1.txt")     #重命名文件
print("重命名成功！")
print("重命名后目录为: %s"%os.listdir(os.getcwd()))
os.remove("test1.txt")
print("删除成功！")
print("删除后目录为: %s"%os.listdir(os.getcwd()))
```

```
目录为: ['.ipynb_checkpoints', '1.txt', 'a.txt', 'bar.ipynb', 'chapter2.ipynb', 'charpter2.i
unction.ipynb', 'module.ipynb', 'myModule.ipynb', 'myModule.py', 'test.bt', 'test.txt', 'Unt
pynb', 'Untitled3.ipynb', 'Untitled4.ipynb', 'Untitled5.ipynb', 'Untitled6.ipynb', 'Untitled
重命名成功!
重命名后目录为: ['.ipynb_checkpoints', '1.txt', 'a.txt', 'bar.ipynb', 'chapter2.ipynb', 'cha
txt', 'function.ipynb', 'module.ipynb', 'myModule.ipynb', 'myModule.py', 'test.bt', 'test1.t
titled2.ipynb', 'Untitled3.ipynb', 'Untitled4.ipynb', 'Untitled5.ipynb', 'Untitled6.ipynb',
che__']
删除成功!
删除后目录为: ['.ipynb_checkpoints', '1.txt', 'a.txt', 'bar.ipynb', 'chapter2.ipynb', 'charp
t', 'function.ipynb', 'module.ipynb', 'myModule.ipynb', 'myModule.py', 'test.bt', 'Untitled.
'Untitled3.ipynb', 'Untitled4.ipynb', 'Untitled5.ipynb', 'Untitled6.ipynb', 'Untitled7.ipynb
```

## os、os.path与shutil模块

**3、判断是否是文件**

**os.path.isfile(path)方法判断path是否是一个文件，返回值是True或者False。**

**4、文件的复制**

**shutil.copy(src.dst)方法将文件src复制到文件或目录dst中，该函数返回目标文件名。**

**5、检查文件是否存在**

**os.path.exists(path)方法用于检查文件的存在性，返回一个布尔值。**

**6、获取绝对路径名**

**os.path.abspath(path)方法返回path的绝对路径名。**

# 文件系统操作



- **os、os.path与shutil模块**
- 示例如下：

In [5]:

```
import os,shutil
if not os.path.exists(r".\1.py"):        #当前目录下1.py文件不存在
    with open(r".\1.py","w") as fp:      #创建1.py文件
        fp.write("print('hello world!')\n")
filename = shutil.copy(r".\1.py","c:\\test")     #复制1.py到c:\test目录下
print(os.path.abspath("1.py"))                   #打印1.py文件所在的绝对路径
```

```
C:\2020-2021(2)\bigdata\jupyter code\1.py
```

# os与os.path模块



os模块提供了文件系统和文件级操作的使用方法，如下表所示

## os模块的常用文件操作方法

| 方 法 | 功能说明 |
| --- | --- |
| **access(path, mode)** | 按照mode制定的权限访问文件 |
| **open((path, flags, mode=511)** | 按mode指定的权限打开文件，默认权限为可读、可写、可执行 |
| **chmod(path, mode, *, dir_fd=None)** | 改变文件的访问权限 |
| **remove(path)** | 删除指定的文件 |
| **rename(src, dst)** | 重命名文件或目录 |

# os与os.path模块



| | |
|---|---|
| **stat(path)** | 返回文件的所有属性 |
| **fstat(path)** | 返回打开的文件的所有属性 |
| **startfile(filepath[, operation])** | 使用关联的应用程序打开指定文件 |
| **mkdir(path, mode=511)** | 创建目录 |
| **makedirs(path1/path2... , mode= 511)** | 创建多级目录 |
| **rmdir(path)** | 删除目录 |
| **removedirs(path1/path2…)** | 删除多级目录 |
| **listdir( path)** | 返回指定目录下的文件和目录信息 |
| **getcwd()** | 返回当前工作目录 |
| **get_exec_path()** | 返回可执行文件的搜索路径 |
| **chdir(path)** | 把path设为当前工作目录 |
| **walk(top, topdown=True)** | 遍历目录树,该方法返回一个元组,包括3个元素：所有路径名、所有目录列表与文件列表 |
| **sep** | 当前操作系统所使用的路径分隔符 |
| **extsep** | 当前操作系统所使用的文件扩展名分隔符 |

# os与os.path模块



os.path模块提供了用于路径判断、切分、连接以及文件夹遍历的方法，如下表所示

- os.path模块的常用文件操作方法

| 方法 | 功能说明 |
|---|---|
| abspath(path) | 返回绝对路径 |
| dirname(p) | 返回目录的路径 |
| exists(path) | 判断文件是否存在 |
| getatime(filename) | 返回文件的最后访问时间 |
| getctime(filename) | 返回文件的创建时间 |
| getmtime(filename) | 返回文件的最后修改时间 |
| getsize(filename) | 返回文件的大小 |
| isabs(path) | 判断path是否为绝对路径 |
| isdir(path) | 判断path是否为目录 |
| isfile(path) | 判断path是否为文件 |
| join(path,*paths) | 连接两个或多个path |
| split(path) | 对路径进行分割,以列表形式返回 |
| splitext(path) | 从路径中分割文件的扩展名 |
| splitdrive(path) | 从路径中分割驱动器的名称 |

# 文件系统操作



## os、os.path与shutil模块

在实际开发中，有时需要用程序的方式文件夹进行一定的操作。比如创建、删除、显示目录内容等，可以通过os和os.path模块提供的方法来完成。

1、创建文件夹

os.mkdir(path)方法用于创建目录，目录存在时会抛出FileExistsError异常。

2、获取当前目录

os.getcwd()返回当前工作目录。

3、改变默认目录

os.chdir(path)改变当前工作目录。

4、获取目录内容

os.listdir(path)返回path指定的目录下包含的文件或子目录的名字列表。

## os、os.path与shutil模块

5、删除目录

os.rmdir(path)删除path指定的目录，如果目录非空，则抛出一个OSError异常。

6、判断是否为目录

os.path.isdir(path)方法用于判断path是否为目录，返回一个布尔值。

7、连接多个目录

os.path.join(path,*paths)方法连接两个或多个path，形成一个完整的目录。

8、分割路径

os.path.split(path)方法对路径进行分割，以元组方式进行返回；os.path.splitext(path)方法从路径中分割文件的扩展名；os.path.splitdrive(path)从路径中分割驱动器名称。

9、获取路径

os.path.abspath(path)方法返回path的绝对路径；os.path.dirname(path)返回path的路径名部分。

# 文件系统操作



## os、os.path与shutil模块

下面通过一个例子，演示文件夹的相关操作，该例子编写一个批量修改文件和目录名的小程序。程序实现文件和目录名前加上Python-前缀。

In [6]:

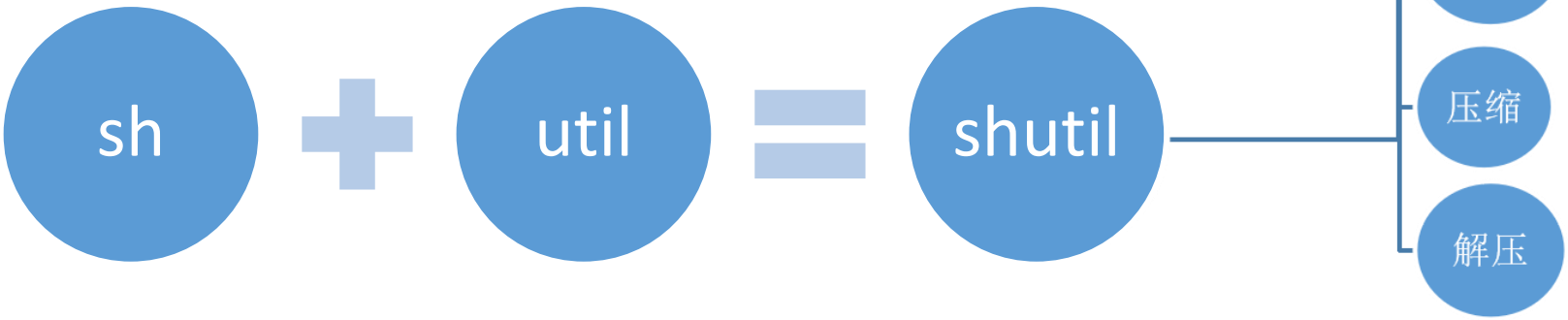
```
import os                              #导入os模块
folderName = './test/'
dirList = os.listdir(folderName)       # 获取指定路径下所有文件和子目录的名字
for name in dirList:                   # 遍历输出所有文件和子目录的名字
    print("修改前文件名：",name)
    newName = 'Python-' + name
    print("修改后文件名：",newName)
    os.rename(os.path.join(folderName,name), os.path.join(folderName,newName))
```

```
修改前文件名： Untitled.ipynb
修改后文件名： Python-Untitled.ipynb
修改前文件名： Untitled1.ipynb
修改后文件名： Python-Untitled1.ipynb
修改前文件名： Untitled2.ipynb
修改后文件名： Python-Untitled2.ipynb
修改前文件名： Untitled3.ipynb
修改后文件名： Python-Untitled3.ipynb
修改前文件名： Untitled4.ipynb
修改后文件名： Python-Untitled4.ipynb
修改前文件名： Untitled5.ipynb
修改后文件名： Python-Untitled5.ipynb
```

# shutil模块

shutil模块也属于Python标准库，是对os模块的补充。

Shutil模块可以与os模块配合使用，基本可以完成一般的文件系统功能。



## 使用shutil操作文件与文件夹

**shutil模块拥有许多文件（夹）操作的功能，包括复制、移动、重命名、删除、压缩包处理等。**

**（1）shutil.copyfileobj (fsrc, fdst) 将文件内容从源fsrc文件复制到fdst文件中去，前提是目标文件fdst具备可写权限。fsrc、fdst参数是打开的文件对象。**

**（2）shutil.copy(fsrc, destination)将fsrc文件复制到destination 文件夹中，两个参数都是字符串格式。如果destination是一个文件名称，那么它会被用来当作复制后的文件名称，即等于“复制 + 重命名”。**

# 文件系统操作



## 使用shutil操作文件与文件夹

（3）shutil.copytree(source, destination)复制整个文件夹，将source文件夹中的所有内容复制到destination中，包括 source里面的文件、子文件夹都会被复制过去。两个参数都是字符串格式。

（4）shutil.move(source, destination)将source文件或文件夹移动到destination中。返回值是移动后文件的绝对路径字符串。如果destination 指向一个文件夹，那么source文件将被移动到destination中，并且保持其原有名字。如果source指向一个文件，destination 指向一个文件，那么source文件将被移动并重命名。

# 文件系统操作

软件学院 School of Software

## 使用shutil操作文件与文件夹

举例如下：

In [12]:
```
import shutil, os
os.chdir("c:\\test")     # 进入文件所在目录
shutil.copy("scores.txt","sample1.txt")    # 成功复制
```

Out[12]: 'sample1.txt'

# 读写常见文件格式

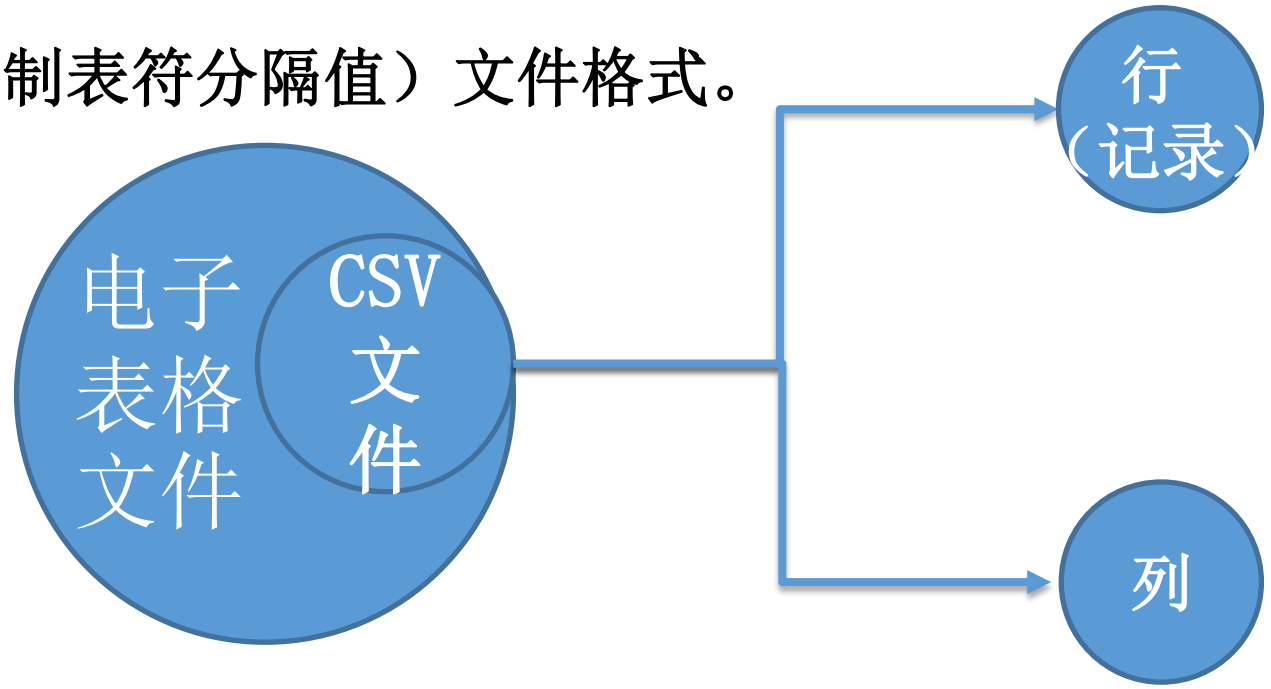
- **CSV文件**
- **Excel文件**
- **JSON文件**

# 7.4.1 CSV文件

- CSV格式属于电子表格文件，其中数据存储在单元格内。每个单元格按照行和列结构进行组织。

  CSV中的每一行代表一个观察，通常称为一条记录。每个记录可以包含一个或多个由逗号分隔的字段。如果文件中不使用逗号分隔，而是使用制表符进行分隔，这样的文件格式称为TSV（制表符分隔值）文件格式。

- TSV（制表符分隔值）文件格式。



# 文件系统操作



## csv文件的读取和写入

- csv (comma separated values，逗号分隔值)文件是一种用来存储表格数据（数字和文本）的纯文本格式文件，文档的内容是由“,”分隔的一列列的数据构成，它可以被导入各种电子表格和数据库中。纯文本意味着该文件是一个字符序列。
- 在csv文件中，数据“栏”(数据所在列，相当于数据库的字段)以逗号分隔，可允许程序通过读取文件为数据重新创建正确的栏结构(如把两个数据栏的数据组合在一起)，并在每次遇到逗号时开始新的一栏。
- csv文件由任意数目的记录组成，记录间以某种换行符分隔，一行即为数据表的一行；每条记录由字段组成，字段间的分隔符最常见的是逗号或制表符。
- 可使用Word、记事本、Excel等方式打开csv文件。

# 文件系统操作



## csv文件的读取和写入

- 创建csv文件的方法有很多，最常用的方法是用电子表格创建，如Microsoft Excel。在Microsoft Excel中，选择“文件”>“另存为”，然后在“文件类型”下拉选择框中选择 "CSV (逗号分隔)(*.csv)"，然后点击保存即创建了一个csv格式的文件。
- Python的csv模块提供了多种读取和写入csv格式文件的方法。

本节基于consumer.csv文件，其内容为：

客户年龄,平均每次消费金额,平均消费周期

23,318,10

22,147,13

24,172,17

27,194,57

## csv文件的读取

**csv.reader()用来读取csv文件，其语法格式如下：**

```
csv.reader(csvfile, dialect='excel')
```

**参数说明：**

**csvfile：可以是文件(file)对象或者列表(list)对象，如果csvfile是文件对象，要求该文件要以newline=''的方式打开，否则两行之间会空一行。**

**dialect：编码风格，默认为excel的风格，也就是用逗号（,）分隔，dialect方式也支持自定义，通过调用register_dialect方法来注册。**

# 文件系统操作



## csv文件的读取和写入

# 文件系统操作



## csv文件的读取例子

In [14]:

```
import csv,shutil, os
os.chdir("C:/2020-2021(2)/bigdata/jupyter code")     # 进入文件所在目录
with open('consumer.csv',newline='') as csvfile:
    spamreader = csv.reader(csvfile)      # 返回的是迭代类型
    for row in spamreader:
        print(', '.join(row))      #以逗号连接各字段
    csvfile.seek(0)              #文件指针移动到文件开始
    for row in spamreader:
        print(row)
```

```
客户年龄 , 平均每次消费金额 , 平均消费周期
23, 318, 10
22, 147, 13
24, 172, 17
27, 194, 57
['客户年龄 ', '平均每次消费金额 ', '平均消费周期']
['23', '318', '10']
['22', '147', '13']
['24', '172', '17']
['27', '194', '57']
```

# 文件系统操作



## csv文件的写入

**csv.writer()用来写入csv文件，其语法格式如下：**

**csv.writer(csvfile, dialect='excel', **fmtparams)**

**说明：csv.writer()返回一个csv.writer对象，使用csv.writer对象可将用户的数据写入该csv.writer对象所对应的文件里。**

**csvfile：可以是文件(file)对象或者列表(list)对象。**

**dialect：编码风格，默认为excel的风格，也就是用逗号（,）分隔，dialect方式也支持自定义，通过调用register_dialect方法来注册。**

**csv.writer()所生成的csv.writer文件对象支持以下写入csv文件的方法：**

**writerow(row)：写入一行数据**

**writerows(rows)：写入多行数据**

# 文件系统操作



## csv文件的写入

# 文件系统操作



## csv文件的写入

**csv.writer()用来写入csv文件，其语法格式如下：**

**csv.writer(csvfile, dialect='excel', **fmtparams)**

**说明：csv.writer()返回一个csv.writer对象，使用csv.writer对象可将用户的数据写入该csv.writer对象所对应的文件里。**

**csvfile：可以是文件(file)对象或者列表(list)对象。**

**dialect：编码风格，默认为excel的风格，也就是用逗号（,）分隔，dialect方式也支持自定义，通过调用register_dialect方法来注册。**

**csv.writer()所生成的csv.writer文件对象支持以下写入csv文件的方法：**

**writerow(row)：写入一行数据**

**writerows(rows)：写入多行数据**

# 文件系统操作



## csv文件的写入例子

In [15]:

```
import csv,shutil, os
os.chdir("C:/2020-2021(2)/bigdata/jupyter code")    # 进入文件所在目录
with open('consumer.csv', 'w', newline='') as csvfile:
    #写入的数据将覆盖consumer.csv文件
    spamwriter = csv.writer(csvfile)        #生成csv.writer文件对象
    spamwriter.writerow(['55','555','55'])   #写入一行数据
    spamwriter.writerows([('35','355','35'),('18','188','18')])
with open('consumer.csv',newline='') as csvfile:   #重新打开文件
    spamreader = csv.reader(csvfile)
    for row in spamreader:   #输出数据文件
        print(row)
```

```
['55', '555', '55']
['35', '355', '35']
['18', '188', '18']
```

# 文件系统操作



## csv文件的写入例子

In [20]:
```
import csv
def csv_write(path, data):
    with open(path,'w',encoding='utf-8',newline='') as f:
        writer = csv.writer(f,dialect='excel')
        for row in data:
            writer.writerow(row)
    return True
data = [
    ['姓名','年龄', '身高（cm）','体重（kg)'],
    ['张三', 38, '176cm','75'],
    ['李四', 25, '160cm', '46'],
    ['王五', 28, '170cm','62']
]
csv_write('persons.csv',data)
```

Out[20]: True

## 使用csv.DictReader()读取csv文件

很多情况下，读取csv数据时，往往先把csv文件中的数据读成字典的形式，即为读出的每条记录中的数据添加一个说明性的关键字，这样便于理解。为此，csv库提供了能直接将csv文件读取为字典的函数：DictReader()，也有相应的将字典写入csv文件的函数DictWriter()。

- csv.DictReader()的语法格式如下：

```
csv.DictReader(csvfile, fieldnames=None, dialect='excel')
```

说明：csv.DictReader()返回一个csv.DictReader对象，可以将读取的信息映射为字典，其关键字由可选参数fieldnames来指定。

- ✓csvfile：可以是文件(file)对象或者列表(list)对象。
- ✓fieldnames：是一个序列，用于为输出的数据指定字典关键字，如果没有指定，则以第一行的各字段名作为字典关键字。
- ✓dialect：编码风格，默认为excel的风格。

# 文件系统操作



## 使用csv.DictReader()读取csv文件

In [16]:

```
import csv
with open('consumer.csv', 'r') as csvfile:
    dict_reader = csv.DictReader(csvfile)
    for row in dict_reader:
        print(row)
```

```
{'55': '35', '555': '355'}
{'55': '18', '555': '188'}
```

## 使用csv.DictWriter()写入csv文件

如果需要将字典形式的记录数据写入csv文件，则可以使用csv.DictWriter()来实现，其语法格式如下：

```
csv.DictWriter(csvfile, fieldnames, dialect='excel')
```

说明：csv.DictWriter ()返回一个csv.DictWriter对象，该对象的操作方法与csv. writer对象的操作方法类似。参数csvfile、fieldnames和dialect的含义与DictReader()函数种的参数类似。

# 文件系统操作

软件学院 School of Software

## 使用csv.DictWriter()写入csv文件

In [18]:
```
import csv
dict_record = [{'客户年龄': 23, '平均每次消费金额': 318, '平均消费周期': 10}, {'客户年龄': 22, '平均每次消费金额': 147, '平均消费周期': 13
keys = ['客户年龄','平均每次消费金额','平均消费周期']
with open('consumer1.csv', 'w+',newline='') as csvfile:
    dictwriter = csv.DictWriter(csvfile, fieldnames=keys)
    #若直接写入会导致没有数据名，先执行writeheader()将文件头写入
    # writeheader()没有参数，建立对象dictwriter时已设定fieldnames
    dictwriter.writeheader()
    for item in dict_record:
        dictwriter.writerow(item)
```

In [19]:
```
import csv
print("以csv.DictReader() 读取consumer1.csv: ")
with open('consumer1.csv', 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row)
```

```
以csv.DictReader() 读取consumer1.csv:
{'客户年龄': '23', '平均每次消费金额': '318', '平均消费周期': '10'}
{'客户年龄': '22', '平均每次消费金额': '147', '平均消费周期': '13'}
```

# 文件系统操作



## csv文件的格式化参数

创建csv.reader或csv.writer对象时，可以指定csv文件格式化参数。csv文件格式化参数包括以下几项：

- delimiter：单字词，默认值为','，用来分割字段。
- doublequote：如果为True（默认值），字符串中的双引号用""表示，若为False，使用转义字符escapechar指定的字符。
- escapechar：转义字符，一个单字串，当quoting被设置成QUOTE_NONE、doublequote被设置成False，被writer用来转义delimiter。
- lineterminator：被writer用来换行，默认为'\r\n'。
- quotechar：单字串，用于包含特殊符号的引用字段，默认值为‘"’。

# 文件系统操作

## 自定义dialect

**dialect用来指定csv文件的编码风格，默认为excel的风格，也就是用逗号“,”分隔，dialect支持自定义，即通过调用register_dialect方法来注册csv文件的编码风格，其语法格式如下：**

**csv.register_dialect(name[, dialect], **fmtparams)**

**说明：这个函数是用来自定义dialect的。**

**name：所自定义的dialect的名字，比如默认的是'excel'，你可以定义成'mydialect'。**

**delimiter：分隔符，默认的是逗号。**

**fmtparams：用于指定特定格式，以覆盖dialect中的格式。**

# 文件系统操作



## 自定义dialect

In [21]:

```
import csv
'''自定义了一个命名为mydialect的dialect，参数只设置了delimiter和quoting这两个，其他的仍然采用默认值，其中以':'为分隔符'''
csv.register_dialect('mydialect', delimiter=':', quoting= csv.QUOTE_ALL)
#quoting用于指定使用双引号的规则，QUOTE_ALL(全部)
with open('consumer.csv', newline='') as f:
    spamreader = csv.reader(f,dialect= 'mydialect')
    for row in spamreader:
        print(row)
```

```
['55,555,55']
['35,355,35']
['18,188,18']
```

# Excel文件



- Excel是常见的电子表格文件，常用openpyxl模块读写Excel表格中的数据。
- 在安装jupyter note时已安装好了openpyxl两个模块。
- 查看这两个模块时，可在编辑器中输入并运行：conda list

```
In [28]: conda list
```

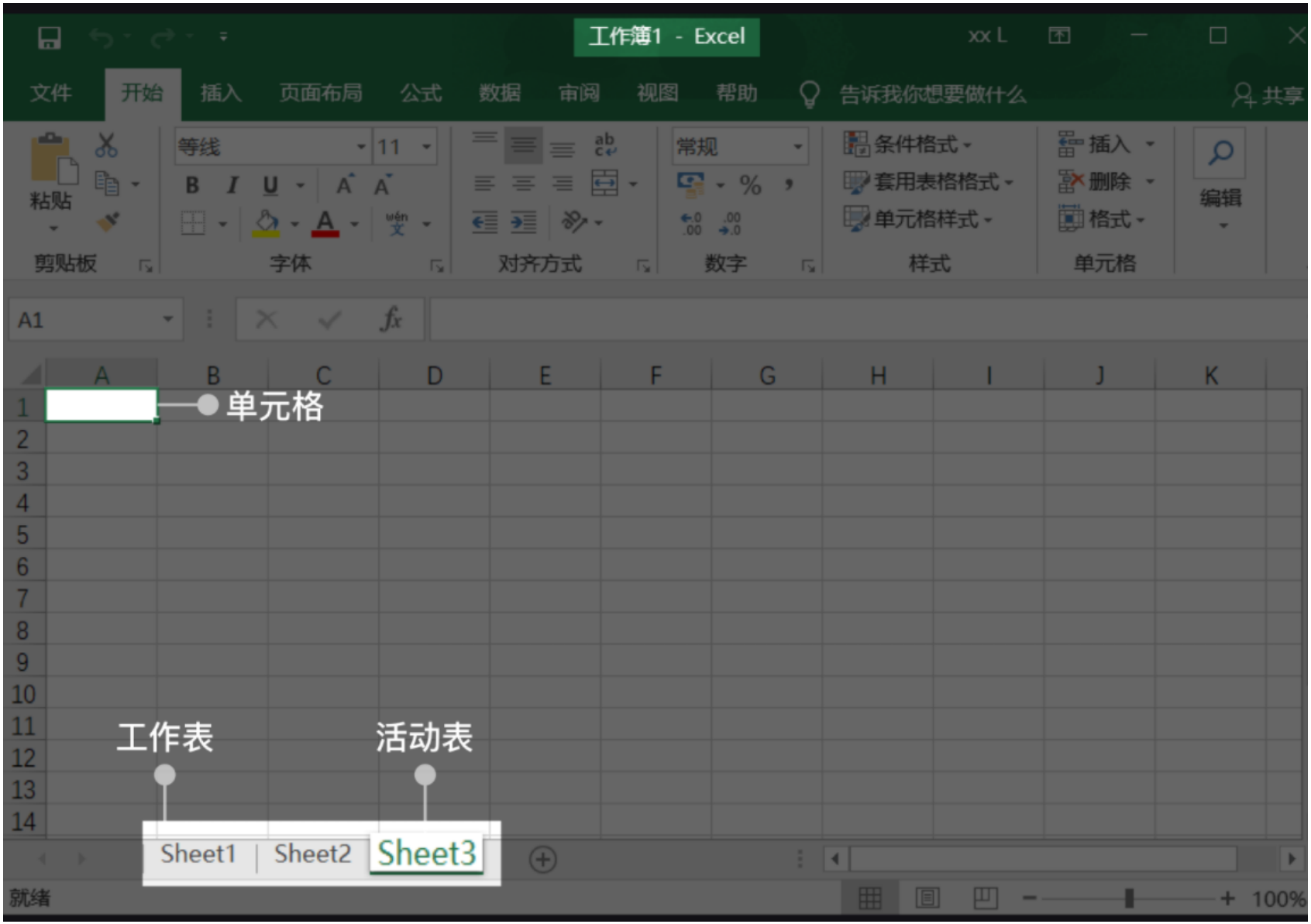
```
numba                     0.51.2           py38hf9181ef_1
numexpr                   2.7.1            py38h25d0782_0
numpy                     1.19.2           py38hadc3359_0
numpy-base                1.19.2           py38ha3acd2a_0
numpydoc                  1.1.0              pyhd3eb1b0_1
olefile                   0.46                     py_0
openpyxl                  3.0.5                    py_0
openssl                   1.1.1h               he774522_0
packaging                 20.4                     py_0
pandas                    1.1.3            py38ha925a31_0
pandoc                    2.11                 h9490d1a_0
pandocfilters             1.4.3            py38haa95532_1
paramiko                  2.7.2                    py_0
parso                     0.7.0                    py_0
partd                     1.1.0                    py_0
```

# Excel文件



- 一个Excel文档也称为一个工作簿（workbook），每个工作簿里可以有多个工作表（worksheet），当前打开的工作表又叫活动表。
- 每个工作表里有行和列，特定的行与列相交的方格称为单元格（cell）。比如下图第A列和第1行相交的方格我们可以直接表示为A1单元格。

# Excel文件


工作簿1 - Excel
文件
开始
插入
页面布局
公式
数据
审阅
视图
帮助
告诉我你想要做什么
共享
粘贴
剪贴板
等线
11
字体
对齐方式
常规
数字
条件格式
套用表格格式
单元格样式
样式
插入
删除
格式
单元格
编辑
A1
单元格
工作表
活动表
Sheet1
Sheet2
Sheet3
就绪
100%

# Excel文件



- 在进行Excel文件读写时，应首先进行工作簿（'workbook'）的获取，然后从工作簿中处理表单（'sheet'）。进入表单处理环节以后，即可按行或者列进行数据的读写，或者是按单元格的方式处理数据。

# Excel文件



- 写入Excel文件的例子：

In [46]:

```
import openpyxl
wb=openpyxl.Workbook()
sheet=wb.active
sheet.title='student'
rows= [['学号','姓名','班级'],['201301','张三','14班'],['201302','李四','14班']]
for i in rows:
    sheet.append(i)
print(rows)
wb.save('student.xlsx')
```

```
[['学号', '姓名', '班级'], ['201301', '张三', '14班'], ['201302', '李四', '14班']]
```

# Excel文件



- 读取Excel文件的步骤：

# Excel文件



## 读取Excel文件的例子：

In [11]:

```
import openpyxl
wb = openpyxl.load_workbook('student1.xlsx')
sheet = wb['student']
columns = sheet.max_column
rows = sheet.max_row
row_data = []
for i in range(1,rows +1) :
    for j in range(1, columns + 1):
        cell_value = sheet.cell(i,j).value
        row_data.append(cell_value)
print(row_data)
```

```
['学号', '姓名', '班级', '201301', '张三', '14班', '201302', '李四', '14班']
```

## JSON文件

JSON是一种使用广泛的轻量数据格式，它可以将JavaScript对象中表示的一组数据转换为字符串，常用于数据的存储和交换。从数据格式来看，Python中的字典类型与JSON数据格式很接近。

以下左侧为字典数据，右侧为JSON数据：

```
d = {
        'a': 123,
        'b’: {
                        'x': ['A', 'B', 'C']
        },
        'c': True,
        'd': None
}
```

```
d = {
        "a": 123,
        "b": {
                        "x": ["A", "B", "C"]
        },
        "c": true,
        "d": null
}
```

# JSON文件



- 由此可见，**Python字典类型与JSON类型的数据在形式上相近，但也有一些区别，如Python中的字符串允许单引号和双引号，而JSON数据中要求必须是双引号，同时二者在布尔类型、空值的处理方面等也有区别。Python标准库中的json模块可以直接将Python数据类型转化为JSON。**

# JSON文件



- Python数据类型与JSON数据类型的相互转换。

In [19]:

```
import json
d = {}; d['a'] = 123;
d['b'] = {'x': ['A', 'B', 'C']}
d['c'] = True; d['d'] = None
print(d)
json_str = json.dumps(d)          # 转换成JSON字符串
print(json_str)
e = json.loads(json_str)          # 将JSON转化为Python字典类型
print(e)
```

```
{'a': 123, 'b': {'x': ['A', 'B', 'C']}, 'c': True, 'd': None}
{"a": 123, "b": {"x": ["A", "B", "C"]}, "c": true, "d": null}
{'a': 123, 'b': {'x': ['A', 'B', 'C']}, 'c': True, 'd': None}
```

# 总结



- pickle模块可用于对二进制数据进行序列化和反序列化
- os、os.path、shutil的文件系统操作方法
- CSV文件、Excel文件和JSON文件的读写方法