第3章

文件与数据的存储

Python 以简洁见长，在其他语言中比较复杂的文件读写和数据 IO，在 Python 中由于比较简单的语法和丰富的类库而显得尤为方便。本章将从最简单的文本文件的读写出发，重点介绍 CSV 文件的读写和操作数据库，同时介绍一些其他形式的数据的存储方式。

3.1 Python 中的文件

3.1.1 基本的文件读写

谈到 Python 中的文件读写，总会使人想到“open”关键字，其最基本的操作如下面的示例：

```
# 最朴素的open()方法
f = open('filename.text','r')
# 做点事情
f.close()
```

```
# 使用with,在语句块结束时会自动关闭
with open('t1.text','rt') as f: # r代表read,t代表text,一般"t"为默认,可省略
  content = f.read()

with open('t1.txt','rt') as f:
  for line in f:
    print(line)
with open('t2.txt', 'wt') as f:
  f.write(content) # 写入

append_str = 'append'
with open('t2.text','at') as f:
  # 在已有内容上追加写入,如果使用"w",则已有内容会被清除
  f.write(append_str)
# 文件的读写操作默认使用系统编码,一般为utf8
# 使用encoding设置编码方式
with open('t2.txt', 'wt',encoding = 'ascii') as f:
  f.write(content)
# 编码错误总是很烦人,如果用户觉得有必要暂时忽略,可以如下
with open('t2.txt', 'wt',errors = 'ignore') as f: # 忽略错误的字符
  f.write(content) # 写入
with open('t2.txt', 'wt',errors = 'replace') as f: # 替换错误的字符
  f.write(content) # 写入

# 重定向print()函数的输出
with open('redirect.txt', 'wt') as f:
  print('your text', file = f)

# 读写字节数据,例如图片、音频
with open('filename.bin', 'rb') as f:
  data = f.read()

with open('filename.bin', 'wb') as f:
  f.write(b'Hello World')

# 从字节数据中读写文本(字符串),需要使用编码和解码
with open('filename.bin', 'rb') as f:
   text = f.read(20).decode('utf-8')

with open('filename.bin', 'wb') as f:
  f.write('Hello World'.encode('utf-8'))
```

用户不难发现,在open()的参数中,第一个是文件路径,第二个是模式字符(串),代表了不同的文件打开方式,比较常用的是"r"(代表读)、"w"(代表写)、"a"(代表写,并追加内容),"w"和"a"经常引起混淆,其区别在于,如果用"w"模式打开一个已存在的文件,会清空文件里的内容数据,重新写入新的内容,如果用"a",不会清空原有数

据，而是继续追加写入内容。对模式字符（串）的详细解释见图 3-1。

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newline mode (deprecated)

图 3-1　open()函数定义中的模式字符

在一个文件（路径）被打开后，用户就拥有了一个 file 对象（在其他一些语言中常被称为句柄），这个对象也拥有自己的一些属性：

```
f = open('h1.html','r')
print(f.name)                # 文件名，h1.html
print(f.closed)              # 是否关闭，False
print(f.encoding)            # 编码方式，US - ASCII
f.close()
print(f.closed)              # True
```

当然，除了最简单的 read()和 write()方法以外，还有一些其他的方法：

```
# t1.txt 的内容
# line 1
# line 2: cat
# line 3: dog
#
# line 5

with open('t1.txt','r') as f1:
  # 返回是否可读
  print(f1.readable())           # True
  # 返回是否可写
  print(f1.writable())           # False
  # 逐行读取
  print(f1.readline())           # line 1
  print(f1.readline())           # line 2: cat
  # 读取多行到列表中
  print(f1.readlines())          # ['line 3: dog\n', '\n', 'line 5']
  # 返回文件指针的当前位置
  print(f1.tell())               # 38
  print(f1.read())               # 指针在末尾，因此没有读取到内容
  f1.seek(0)                     # 重设指针
```

```
    # 重新读取多行
    print(f1.readlines()) # ['line 1\n', 'line 2: cat\n', 'line 3: dog\n', '\n', 'line 5']

with open('t1.txt','a + ') as f1:
    f1.write('new line')
    f1.writelines(['a','b','c'])        # 根据列表写入
    f1.flush()                          # 立刻写入,实际上是清空 IO 缓存
```

3.1.2 序列化

Python 程序运行时,其变量(对象)都保存在内存中,一般把“将对象的状态信息转换为可以存储或传输的形式的过程”称为(对象的)序列化。通过序列化,用户可以在磁盘上存储这些信息,或者通过网络来传输,并最终通过反序列化过程重新读入内存(可以是另外一个计算机的内存)且使用。在 Python 中主要使用 pickle 模块来实现序列化和反序列化。下面就是一个序列化的小例子:

```
import pickle
l1 = [1,3,5,7]
with open('l1.pkl','wb') as f1:
    pickle.dump(l1,f1)                  # 序列化

with open('l1.pkl','rb') as f2:
    l2 = pickle.load(f2)
    print(l2)                           # [1, 3, 5, 7]
```

在 pickle 模块的使用中还存在一些细节,比如 dump()和 dumps()两个方法的区别在于 dumps()将对象存储为一个字符串,与之相对应,可以使用 loads()来恢复(反序列化)该对象。从某种意义上说,Python 对象都可以通过这种方式来存储、加载,不过有一些对象比较特殊,无法进行序列化,例如进程对象、网络连接对象等。

3.2 字符串

字符串是 Python 中最常用的数据类型,Python 为字符串操作提供了很多有用的内建函数(方法),下面介绍几种常用的方法。

- str.capitalize():返回一个以大写字母开头,其他都小写的字符串。

- str.count(str, beg=0, end=len(string))：返回 str 在 string 里面出现的次数，如果 beg(开始)或者 end(结束)被设置，则返回指定范围内 str 出现的次数。
- str.endswith(obj, beg=0, end=len(string))：判断一个字符串是否以参数 obj 结束，如果 beg 或者 end 指定，则只检查指定的范围。其返回布尔值。
- str.find()：检测 str 是否包含在 string 中，这个方法与 str.index()方法类似，不同之处在于 str.index()如果没有找到会返回异常。
- str.format()：格式化字符串。
- str.decode()：以 encoding 指定的编码格式解码。
- str.encode()：以 encoding 指定的编码格式编码。
- str.join()：以 str 作为分隔符，把参数中所有的元素的字符串表示合并为一个新的字符串，要求参数是 iterable。
- str.partition(string)：从 string 出现的第一个位置起，把字符串 str 分成一个 3 元素的元组。
- str.replace(str1,str2)：将 str 中的 str1 替换为 str2，这个方法还能够指定替换次数，十分方便。
- str.split(str1="", num=str.count(str1))：以 str1 为分隔符对 str 进行切片，这个函数容易让人联想到 re 模块中的 re.split()方法(见第 2 章的相关内容)，前者可以视为后者的弱化版。
- str.strip()：去掉 str 左、右两侧的空格。

这里通过一段代码演示上面函数的功能：

```
s1 = 'mike'
s2  = 'miKE'
print(s1.capitalize())                                    # Mike
print(s2.capitalize())                                    # Mike
s1 = 'aaabb'
print(s1.count('a'))                                      # 3
print(s1.count('a',2,len(s1)))                            # 1
print(s1.endswith('bb'))                                  # True
print(s1.startswith('aa'))                                # True
cities_str = ['Beijing','Shanghai','Nanjing','Shenzhen']
print([cityname for cityname in cities_str if cityname.startswith(('S','N'))])# 比较复杂
                                                                              #的用法
```

```
# ['Shanghai', 'Nanjing', 'Shenzhen']

print(s1.find('aa'))                                    # 0
print(s1.index('aa'))                                   # 0
print(s1.find('c'))                                     # -1
# print(s1.index('c'))                                  # 值错误

print('There are some cities: ' + ', '.join(cities_str))
# There are some cities: Beijing, Shanghai, Nanjing, Shenzhen
print(s1.partition('b'))                                # ('aaa', 'b', 'b')
print(s1.replace('b','c',1))                            # aaacb
print(s1.replace('b','c',2))                            # aaacc
print(s1.replace('b','c'))                              # aaacc
print(s2.split('K'))                                    # ['mi', 'E']

s3 = '  a abc c '
print(s3.strip())                                       # 'a abc c'
print(s3.lstrip())                                      # 'a abc c '
print(s3.rstrip())                                      # '  a abc c'
# 最常见的format()的使用方法
print('{} is a {}'.format('He','Boy'))                  # He is a Boy
# 指明参数编号
print('{1} is a {0}'.format('Boy','He'))                # He is a Boy
# 使用参数名
print('{who} is a {what}'.format(who = 'He',what = 'boy'))  # He is a boy

print(s2.lower())                                       # mike
print(s2.upper())                                 # MIKE,注意该方法与capitalize()不同
```

除了这些方法以外,Python 的字符串还支持其他一些实用方法。另外,如果要对字符串进行操作,正则表达式往往会成为十分重要的配套工具,关于正则表达式的内容可参考第 2 章和附录 A。

3.3 Python 与图片

3.3.1 PIL 与 Pillow

PIL(Python Image Library)是 Python 中用于图片、图像的基础工具,而 Pillow 可以认为是基于 PIL 的一个变体(正式说法是“分支”),在某些场合,PIL 和 Pillow 可以当成同义词使用,因此这里主要介绍一下 Pillow。在这之前,如果用户没有安装

Pillow，记得要先通过 pip 安装。Pillow 的主要模块是“Image”，其中的 Image 类是比较常用的：

```
from PIL import Image, ImageFilter

# 打开图像文件
img = Image.open('cat.jpeg')
img.show()                                          # 查看图像
print(img.size)                                     # 图像尺寸，输出(289, 174)
print(img.format)                                   # 图像(文件)格式，输出JPEG
w,h = img.size
# 缩放
img.thumbnail((w//2, h//2))
# 保存缩放后的图像
img.save('thumbnail.jpg', 'JPEG')

img.transpose(Image.ROTATE_90).save('r90.jpg')      # 旋转90°
img.transpose(Image.FLIP_LEFT_RIGHT).save('l2r.jpg')  # 左右翻转

img.filter(ImageFilter.DETAIL).save('detail.jpg')   # 不同的滤镜
img.filter(ImageFilter.BLUR).save('blur.jpg')

img.crop((0,0,w//2,h//2)).save('crop.jpg')          # 根据参数指定的区域裁剪图像

# 创建新图片
img2 = Image.new("RGBA",(500,500),(255,255,0))
img2.save("new.png","PNG")                          # 创建一张500 × 500的纯色图片

img2.paste(img,(10,10))                             # 将img粘贴到指定位置
img2.save('combine.png')
```

上面代码的运行结果见下面的几张图片，图 3-2 是缩放前后的图片对比，图 3-3 是翻转、旋转后的图片效果，图 3-4 是 BLUR 后的效果（模糊效果），图片的粘贴效果可见图 3-5。

图 3-2　缩放前后的图片对比

图 3-3　翻转、旋转后的图片

图 3-4　BLUR 后的图片

图 3-5　粘贴后的图片

在实际使用中，PIL 的 Image.save()方法常用来做图片格式的相互转换，而缩放等方法也十分实用。在网页抓取中，当用户遇到需要保存较小的图片时，可以先进行缩放处理再存储。

3.3.2　Python 与 OpenCV 简介

与基本的 PIL 相比，OpenCV 更像是一把瑞士军刀。cv2 模块则是比较新的接口版本。OpenCV 的全称是 Open Source Computer Vision Library，它基于 C/C++语言，但经过包装后可在 Java 和 Python 等其他语言中使用。OpenCV 由英特尔公司发起，可以在商业和学术领域免费、开源使用，2009 年后的 OpenCV 2.0 版本是目前比较常见的版本。目前已经出现了 OpenCV3 版本，但 OpenCV 2.0 仍旧受到广泛欢迎。由于免费、开源、功能丰富，并且跨平台易于移植，OpenCV 已经成为目前计算机

视觉编程与图像处理方面最重要的工具之一。图 3-6 是 OpenCV 的官方站点。

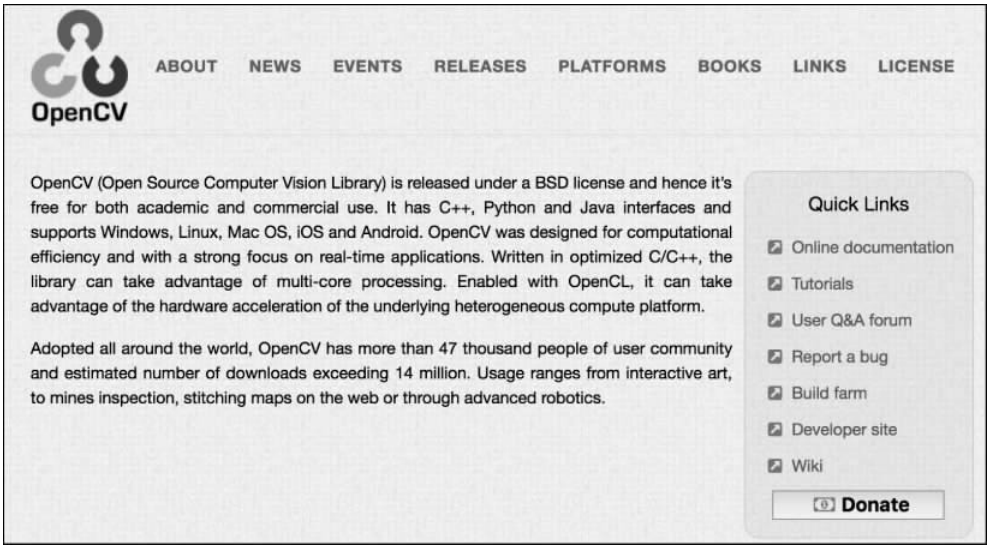


图 3-6　OpenCV 的官方站点

如果要在 Python 中使用 cv2 模块，需要先在计算机上安装 OpenCV 包。其实它在 Windows 系统上的安装并没有想象中那么复杂，将从下载网址（https://opencv.org/releases.html）中下载对应的 OpenCV 包解压，然后将“C:/opencv/build/python/2.7”下的 cv2.pyd 文件复制到“C:/Python27/lib/site-packages”即可。

在 Mac 系统上，则可以使用包管理工具 homebrew 进行快速安装，如图 3-7 所示。

```
==> Summary
🍺  /usr/local/Cellar/sqlite/3.23.1: 11 files, 3MB
==> Installing opencv dependency: xz
==> Downloading https://homebrew.bintray.com/bottles/xz-5.2.3.high_sierra.bottle
######################################################################## 100.0%
==> Pouring xz-5.2.3.high_sierra.bottle.tar.gz
🍺  /usr/local/Cellar/xz/5.2.3: 92 files, 1.4MB
==> Installing opencv dependency: python
```

图 3-7　homebrew 安装 OpenCV 的过程

使用下面的命令安装 homebrew：/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"。

安装成功后，使用命令 brew update 和 brew install opencv 即可一键安装。除了 OpenCV 以外，Redis、MySQL、OpenSSL 等也可以使用这种方法安装。

最终在 Python 中导入 cv2，查看当前版本，安装成功：

```
>>> cv2.__version__
'3.4.0'
```

由于 OpenCV 已经是比较专业的图像处理工具包，这里对 OpenCV 的具体使用就不详细介绍了，在开发时如果用户需要用到 OpenCV，可以随时在官方站点（https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html）中找到相应的说明。

3.4 CSV 文件

3.4.1 CSV 简介

CSV 的全称是 Comma Separated Values（逗号分隔值），CSV 文件以纯文本形式存储表格数据（数字和文本）。CSV 文件由任意数目的记录组成，记录之间以某种换行符（一般是制表符或者逗号）分隔，每条记录中是一些字段。在进行网络抓取时，用户难免会遇到 CSV 文件数据，而且由于 CSV 的设计简单，在很多时候使用 CSV 保存数据（数据有可能是原生的网页数据，也可能是已经经过爬虫程序处理后的结果）十分方便。

3.4.2 CSV 的读写

Python 的 CSV 面向的是本地的 CSV 文件，如果用户需要读取网络资源中的 CSV，为了让用户在网络中遇到的数据也能被 CSV 以本地文件的形式打开，可以先把它下载到本地，然后定位文件路径，作为本地文件打开；如果用户只需要读取一次，并不想真正保存这个文件（就像验证码图片那样，可见第 5 章的相关内容），可以在读取操作结束后用代码删除文件。除此之外，用户也可以直接把网络上的 CSV 文件当成一个字符串来读，转换成一个 StringIO 对象后就能够作为文件来操作了。

【提示】 IO 是 Input/Output 的简写，意为输入/输出，StringIO 就是在内存中读写字符串。StringIO 针对的是字符串（文本），如果还要操作字节，可以使用 BytesIO。

使用 StringIO 的优点在于，这种读写是在内存中完成的（本地文件则是从硬盘读取），因此用户不需要先把 CSV 文件保存到本地。例 3-1 是一个直接获取网上的 CSV 文件并读取打印的例子。

【例 3-1】 获取在线 CSV 文件并读取。

```
from urllib.request import urlopen
from io import StringIO
import csv

data = urlopen("https://raw.githubusercontent.com/jasonong/List-of-US-States/master/states.csv").read().decode()
dataFile = StringIO(data)
dictReader = csv.DictReader(dataFile)
print(dictReader.fieldnames)

for row in dictReader:
  print(row)
```

运行结果为：

```
['State', 'Abbreviation']
{'Abbreviation': 'AL', 'State': 'Alabama'}
{'Abbreviation': 'AK', 'State': 'Alaska'}
…
{'Abbreviation': 'NY', 'State': 'New York'}
{'Abbreviation': 'NC', 'State': 'North Carolina'}
{'Abbreviation': 'ND', 'State': 'North Dakota'}
{'Abbreviation': 'OH', 'State': 'Ohio'}
{'Abbreviation': 'OK', 'State': 'Oklahoma'}
{'Abbreviation': 'OR', 'State': 'Oregon'}
…
```

这里需要说明一下 DictReader()，DictReader()将 CSV 的每一行作为一个 dict 返回，而 reader()则把每一行作为一个列表返回，使用 reader()时的输出是这样的：

```
['State', 'Abbreviation']
…
['California', 'CA']
['Colorado', 'CO']
['Connecticut', 'CT']
['Delaware', 'DE']
['District of Columbia', 'DC']
['Florida', 'FL']
```

```
['Georgia', 'GA']
...
```

用户根据自己的需要选用读取形式即可。

写入和读取是反向操作，下面的例子展示了如何写入数据到 CSV：

```
import csv

res_list = [['A','B','C'],[1,2,3],[4,5,6],[7,8,9]]
with open('SAMPLE.csv', "a") as csv_file:
  writer = csv.writer(csv_file, delimiter = ',')
  for line in res_list:
    writer.writerow(line)
```

打开 SAMPLE.csv 的内容：

```
A,B,C
1,2,3
4,5,6
```

writer()与上文的 reader()是相对应的，这里需要说明的是 writerow()方法和 writerows()方法。writerow()顾名思义就是写入一行，接收一个可迭代对象作为参数；writerows()直观地说等于多个 writerow()，因此上面的代码与下面是等效的：

```
res_list = [['A','B','C'],[1,2,3],[4,5,6],[7,8,9]]
with open('SAMPLE.csv', "a") as csv_file:
  writer = csv.writer(csv_file, delimiter = ',')
  writer.writerows(res_list)
```

如果说 writerow()会把列表中的每个元素作为一列写入 CSV 的一行中，writerows()就是把列表中的每个列表作为一行再写入。所以如果用户误用了 writerows()，可能会导致让人啼笑皆非的错误：

```
res_list = ['I WILL BE ','THERE','FOR YOU']
with open('SAMPLE.csv', "a") as csv_file:
  writer = csv.writer(csv_file, delimiter = ',')
  writer.writerows(res_list)
```

这里由于“I WILL BE”是一个字符串，而 str 在 Python 中是 iterable(可迭代对象)，所以这样写入，最终的结果为(逗号为分隔符)：

```
I, ,W,I,L,L, ,B,E,
T,H,E,R,E
F,O,R, ,Y,O,U
```

如果CSV要写入数值，那么也会报错，即“csv. Error: iterable expected, not int”。

当然，在读取作为网络资源的CSV文件时，除了StringIO以外，还可以先下载到本地读取后再删除（对于只需要读取一次的情况而言）。另外，XLS作为电子表格（使用Office Excel编辑）也常作为CSV的替代文件格式出现，处理XLS可以使用openpyxl模块，其设计和操作与CSV类似。

3.5　使用数据库

在Python中使用数据库（主要是关系型数据库）是一件非常方便的事情，因为一般都能找到对应的经过包装的API库，这些库的存在极大地提高了用户编写程序的效率。一般而言，用户只需要编写SQL语句并通过相应的模块API执行就可以完成数据库的读写了。

3.5.1　使用MySQL

在Python中进行数据库操作需要通过特定的程序模块（API）来实现，其基本逻辑是首先导入接口模块，然后通过设置数据库名、用户、密码等信息来连接数据库，接着执行数据库操作（可以通过直接执行SQL语句等方式），最后关闭与数据库的连接。由于MySQL是比较简单且常用的轻量型数据库，下面先用PyMySQL模块来介绍在Python中如何使用MySQL。

【提示】 PyMySQL是Python 3.x版本中用于连接MySQL服务器的一个库，在Python 2.x版本中使用的是mysqldb。PyMySQL是基于Python开发的MySQL驱动接口，在Python 3.x中非常常用。

首先确保在本地计算机上已经成功开启了MySQL服务（如果还未安装MySQL，需要先进行安装，可以在“https://dev.mysql.com/downloads/installer/”下载MySQL官方安装程序），之后使用pip install pymysql安装该模块。在上面的准备完成后，创建一个名为“DB”的数据库和一个名为“scraper1”的用户，密码设为

“password”：

```
CREATE DATABASE DB;
GRANT ALL PRIVILEGES ON *.'DB' TO 'scraper1'@'localhost' IDENTIFIED BY 'password';
```

接着创建一个名为“users”的表：

```
USE DB;
CREATE TABLE 'users' (
    'id' int(11) NOT NULL AUTO_INCREMENT,
    'email' varchar(255) COLLATE utf8_bin NOT NULL,
    'password' varchar(255) COLLATE utf8_bin NOT NULL,
    PRIMARY KEY ('id')
) ENGINE = InnoDB DEFAULT CHARSET = utf8 COLLATE = utf8_bin
AUTO_INCREMENT = 1;
```

现在有了一个空表，使用 PyMySQL 进行操作，见例 3-2。

【例 3-2】 使用 PyMySQL。

```
import pymysql.cursors
# Connect to the database
connection = pymysql.connect(host = 'localhost',
                             user = 'scraper1',
                             password = 'password',
                             db = 'DB',
                             charset = 'utf8mb4',
                             cursorclass = pymysql.cursors.DictCursor)
try:
    with connection.cursor() as cursor:
        sql = "INSERT INTO 'users' ('email', 'password') VALUES (%s, %s)"
        cursor.execute(sql, ('example@example.org', 'password'))

    connection.commit()

    with connection.cursor() as cursor:
        sql = "SELECT 'id', 'password' FROM 'users' WHERE 'email' = %s"
        cursor.execute(sql, ('example@example.org',))
        result = cursor.fetchone()
        print(result)
finally:
    connection.close()
```

在这段代码中，首先通过 pymysql.connect()函数进行了连接配置并打开了数据库连接；在 try 代码块中打开了当前 connection 的 cursor()(游标)，并通过 cursor 执

行了特定的 SQL 插入语句；commit()方法将提交当前的操作，之后再次通过 cursor 实现对刚才插入数据的查询；最后在 finally 语句块中关闭了当前数据库连接。

本程序的输出为：

```
{'id': 1, 'password': 'password'}
```

考虑到在执行 SQL 语句时可能发生错误，可以将程序写成下面的形式：

```
try:
  ...
except:
  connection.rollback()
finally:
  ...
```

rollback()方法将回滚操作。

3.5.2　使用 SQLite3

SQLite3 是一种小巧、易用的轻量型关系型数据库系统，在 Python 中内置了 sqlite3 模块用于和 SQLite3 数据库进行交互，首先使用 PyCharm 创建一个名为“new-sqlite3”的 SQLite3 数据源，如图 3-8 所示。

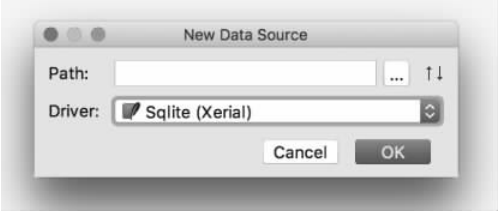


图 3-8　在 PyCharm 中新建 SQLite3 数据源

然后使用 sqlite3(此处的 sqlite3 指的是 Python 中的模块)进行建表操作，与前面对 MySQL 的操作类似：

```
import sqlite3
conn = sqlite3.connect('new - sqlite3')
print("Opened database successfully")
cur = conn.cursor()
cur.execute(
```

```
  '''CREATE TABLE users
       (ID INT PRIMARY KEY      NOT NULL,
       NAME            TEXT     NOT NULL,
       AGE             INT      NOT NULL,
       GENDER         TEXT,
       SALARY          REAL);'''
)
print("Table created successfully")
conn.commit()
conn.close()
```

接着在 users 表中插入两条测试数据，可以看到，sqlite3 模块与 pymysql 模块的函数名非常相像：

```
conn = sqlite3.connect('new - sqlite3')
c = conn.cursor()

c.execute(
  '''INSERT INTO users (id,name,age,gender,salary)
      VALUES (1, 'Mike', 32, 'Male', 20000);''')
c.execute(
  '''INSERT INTO users (id,name,age,gender,salary)
      VALUES (2, 'Julia', 25, 'Female', 15000);''')
conn.commit()
print("Records created successfully")
conn.close()
```

最后进行读取操作，确认两条数据已经被插入：

```
conn = sqlite3.connect('new - sqlite3')
c = conn.cursor()
cursor = c.execute("SELECT id, name, salary  FROM users")
for row in cursor:
  print(row)
conn.close()
# 输出
# (1, 'Mike', 20000.0)
# (2, 'Julia', 15000.0)
```

UPDATE、DELETE 等操作，只需要更改对应的 SQL 语句即可，除了 SQL 语句变化以外，整体的使用方法是一致的。

需要说明的是，在 Python 中通过 API 执行 SQL 语句往往需要使用通配符，遗憾的是，不同的数据库类型使用的通配符可能并不一样，比如在 SQLite3 中使用“?”，而

在 MySQL 中使用“%s”。虽然看上去像是对 SQL 语句的字符串进行格式化(调用 format()方法),但是这并非一回事。另外,在一切操作完毕后不要忘了通过 close()关闭数据库连接。

3.5.3　使用 SQLAlchemy

有时候,为了进行数据库操作,用户还需要一个比底层 SQL 语句更高级的接口,即 ORM(对象关系映射)接口。SQLAlchemy 这样的库(见图 3-9)能够满足这样的需求,使得用户可以在隐藏底层 SQL 的情况下实现各种数据库的操作。所谓 ORM,大概的意思就是在数据表与对象之间建立对应关系,这样用户得以通过纯 Python 语句来表示 SQL 语句,从而进行数据库操作。

图 3-9　SQLAlchemy 的 logo

除了 SQLAlchemy 以外,Python 中的 SQLObject 和 peewee 等也是 ORM 工具。值得一提的是,虽然 SQLAlchemy 是 ORM 工具,但也支持传统的基于底层 SQL 语句的操作。

使用 SQLAlchemy 进行建表以及增/删/改/查:

```
import pymysql
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine, Column, Integer, String, func
from sqlalchemy.orm import sessionmaker

pymysql.install_as_MySQLdb()  # 如果没有这个语句,在导入 SQLAlchemy 时可能会报错
Base = declarative_base()

class Test(Base):
  __tablename__ = 'Test'
  id = Column('id', Integer, primary_key = True, autoincrement = True)
  name = Column('name', String(50))
  age = Column('age', Integer)


engine = create_engine(
  "mysql://scraper1:password@localhost:3306/DjangoBS",
)
```

```
db_ses = sessionmaker(bind = engine)
session = db_ses()

Base.metadata.create_all(engine)

# 插入数据
user1 = Test(name = 'Mike', age = 16)
user2 = Test(name = 'Linda', age = 31)
user3 = Test(name = 'Milanda', age = 5)
session.add(user1)
session.add(user2)
session.add(user3)
session.commit()

# 修改数据,使用merge()方法(如果存在则修改数据,如果不存在则插入数据)
user1.name = 'Bob'
session.merge(user1)

# 与上面等效的修改方式
session.query(Test).filter(Test.name == 'Bob').update({'name': 'Chloe'})
# 删除数据
session.query(Test).filter(Test.id == 3).delete() # 删除Milanda
# 查询数据
users = session.query(Test)
print([user.name for user in users])

# 按条件查询
user = session.query(Test).filter(Test.age < 20).first()
print(user.name)

# 在结果中进行统计
user_count = session.query(Test.name).order_by(Test.name).count()
avg_age = session.query(func.avg(Test.age)).first()
sum_age = session.query(func.sum(Test.age)).first()
print(user_count)
print(avg_age)
print(sum_age)

session.close()
```

上面程序的输出为：

```
['Chloe', 'Linda']
Chloe
2
(Decimal('23.5000'),)
```

```
(Decimal('47'),)
```

除此之外，在 SQLAlchemy 中还有其他一些常用的函数方法和功能，对于更多内容，用户可以参考 SQLAlchemy 的官方文档。上面的代码演示的 ORM 操作实际上为数据库提供了更高级的封装，用户在编写类似的程序时往往能获得更好的体验。

3.5.4　使用 Redis

简单地说，Redis 是一个开源的键值对存储数据库，因为不同于关系型数据库，往往也被称为数据结构服务器。Redis 是基于内存的，但可以将存储在内存的键值对数据持久化到硬盘。使用 Redis 最主要的好处就在于可以避免写入不必要的临时数据，也免去了对临时数据进行扫描或者删除的麻烦，并最终改善程序的性能。Redis 可以存储键与 5 种不同数据结构类型之间的映射，分别是 STRING（字符串）、LIST（列表）、SET（集合）、HASH（散列）和 ZSET（有序集合）。为了在 Python 中使用 Redis API，用户可以安装 redis 模块，其基本用法如下：

```
import redis

red = redis.Redis(host = 'localhost', port = 6379, db = 0)
red.set('name', 'Jackson')
print(red.get('name'))          # b'Jackson'
print(red.keys())               # [b'name']
print(red.dbsize())             # 1
```

redis 模块使用连接池来管理对一个 Redis Server 的所有连接，这样就避免了每次建立、释放连接的开销。默认每个 Redis 实例都会维护一个自己的连接池。用户可以直接建立一个连接池，这样可以实现多个 Redis 实例共享一个连接池：

```
import redis
# 使用连接池
pool = redis.ConnectionPool(host = 'localhost', port = 6379)

r = redis.Redis(connection_pool = pool)
r.set('Shanghai', 'Pudong')
print(r.get('Shanghai')) # b'Pudong'
```

通过 set()方法设置过期时间：

```
import time
r.set('Shenzhen','Luohu',ex=5)                    # ex 表示过期时间(按秒)
print(r.get('Shenzhen'))                          # b'Luohu'
time.sleep(5)
print(r.get('Shenzhen'))                          # None
```

批量设置与读取：

```
r.mset(Beijing='Haidian',Chengdu='Qingyang',Tianjin='Nankai')       # 批量
print(r.mget('Beijing','Chengdu','Tianjin'))     # [b'Haidian', b'Qingyang', b'Nankai']
```

除了上面这些最基本的操作以外，Redis 还提供了丰富的 API 供开发者与 Redis 数据库交互，由于本节只是简单地介绍一下 Python 中的数据库，这里对此就不赘述了。

3.6 其他类型的文档

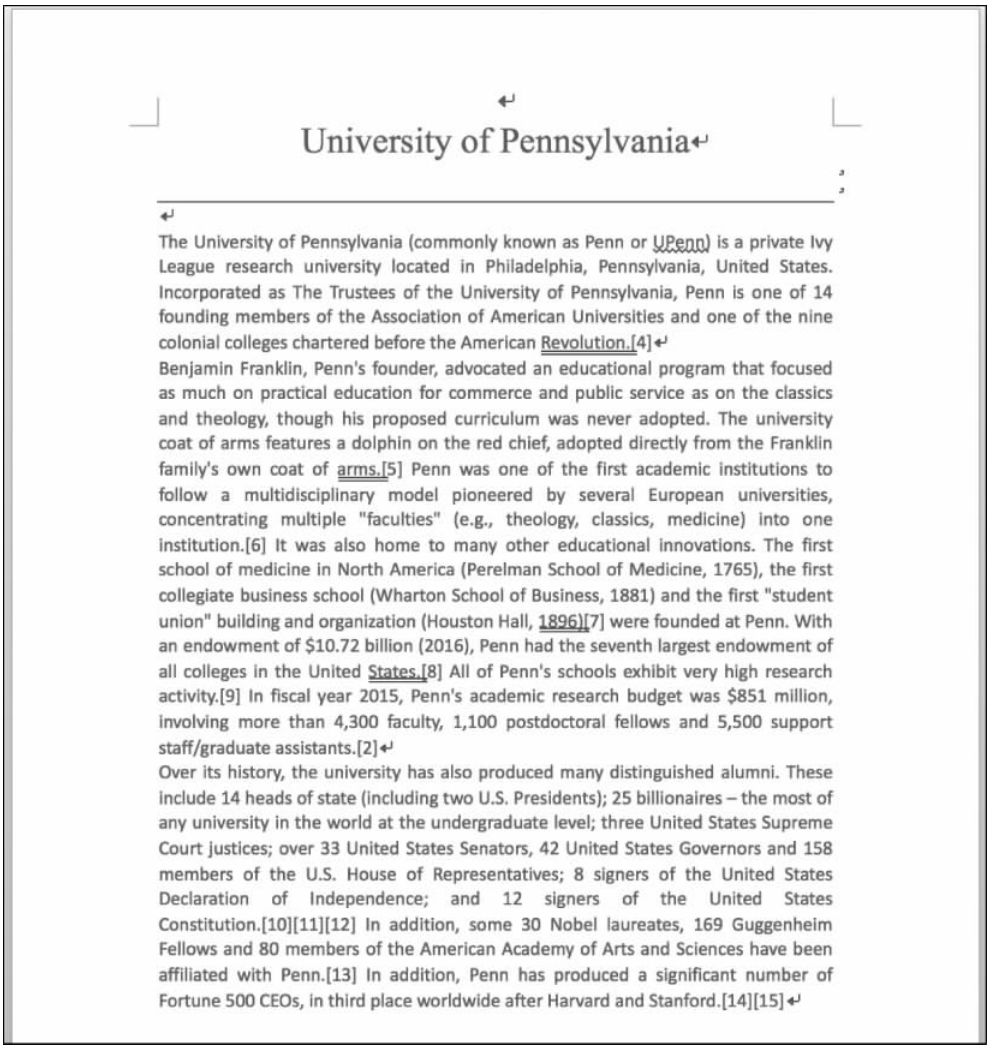
除了一些常见的文件格式以外，用户有时候还需要处理一些相对比较特殊的文档类型文件。首先来试着读取.docx 文件(.doc 与.docx 是 Microsoft Word 程序的文档格式)，这里以一个内容为 University of Pennsylvania 的维基百科的 Word 文档为例，图 3-10 是该文件中的内容。

如果要读取这样的.docx 文件，用户必须先下载、安装 python-docx 模块，仍然使用 pip 或者 PyCharm IDE 进行安装。之后通过该模块进行文件操作：

```
import docx
from docx import Document
from pprint import pprint

def getText(filename):
  doc = docx.Document(filename)
  fullText = []
  for para in doc.paragraphs:
    fullText.append(para.text)
  return fullText

pprint(getText('sample.docx'))
```



University of Pennsylvania

The University of Pennsylvania (commonly known as Penn or UPenn) is a private Ivy League research university located in Philadelphia, Pennsylvania, United States. Incorporated as The Trustees of the University of Pennsylvania, Penn is one of 14 founding members of the Association of American Universities and one of the nine colonial colleges chartered before the American Revolution.[4]

Benjamin Franklin, Penn's founder, advocated an educational program that focused as much on practical education for commerce and public service as on the classics and theology, though his proposed curriculum was never adopted. The university coat of arms features a dolphin on the red chief, adopted directly from the Franklin family's own coat of arms.[5] Penn was one of the first academic institutions to follow a multidisciplinary model pioneered by several European universities, concentrating multiple "faculties" (e.g., theology, classics, medicine) into one institution.[6] It was also home to many other educational innovations. The first school of medicine in North America (Perelman School of Medicine, 1765), the first collegiate business school (Wharton School of Business, 1881) and the first "student union" building and organization (Houston Hall, 1896)[7] were founded at Penn. With an endowment of $10.72 billion (2016), Penn had the seventh largest endowment of all colleges in the United States.[8] All of Penn's schools exhibit very high research activity.[9] In fiscal year 2015, Penn's academic research budget was $851 million, involving more than 4,300 faculty, 1,100 postdoctoral fellows and 5,500 support staff/graduate assistants.[2]

Over its history, the university has also produced many distinguished alumni. These include 14 heads of state (including two U.S. Presidents); 25 billionaires – the most of any university in the world at the undergraduate level; three United States Supreme Court justices; over 33 United States Senators, 42 United States Governors and 158 members of the U.S. House of Representatives; 8 signers of the United States Declaration of Independence; and 12 signers of the United States Constitution.[10][11][12] In addition, some 30 Nobel laureates, 169 Guggenheim Fellows and 80 members of the American Academy of Arts and Sciences have been affiliated with Penn.[13] In addition, Penn has produced a significant number of Fortune 500 CEOs, in third place worldwide after Harvard and Stanford.[14][15]

图 3-10　Word 文档的内容

上面程序的输出为：

```
...
"Benjamin Franklin, Penn's founder, advocated an educational program that "
'focused as much on practical education for commerce and public service as on '
'the classics and theology, though his proposed curriculum was never adopted. '
'The university coat of arms features a dolphin on the red chief,adopted '
"directly from the Franklin family's own coat of arms.[5] Penn was one of the "
'first academic institutions to follow a multidisciplinary model pioneered by '
...
```

除了读取.docx 文档以外，python-docx 模块还支持直接创建文档：

```
import docx
from docx import Document

document = Document()

document.add_heading('This is Title', 0)          # 添加标题,例如"Doc Title @zyang"

p = document.add_paragraph('A plain paragraph ')  # 添加段落,例如"Doc Paragraph @zyang"
p.add_run(' bold text ').bold = True              # 添加格式文字
p.add_run(' italic text ').italic = True

document.add_heading('Heading 1', level = 1)
document.add_paragraph('Intense quote', style = 'IntenseQuote')

document.add_paragraph(                           # 无序列表
    'unordered list 1', style = 'ListBullet'
)
for i in range(3):
  document.add_paragraph(                         # 有序列表
    'ordered list {}'.format(i), style = 'ListNumber'
  )

document.add_picture('cat.jpeg')                  # 添加图片

table = document.add_table(rows = 1, cols = 2)    # 设置表
hdr_cells = table.rows[0].cells
hdr_cells[0].text = 'name'                        # 设置列名
hdr_cells[1].text = 'gender'
d = [dict(name = 'Bob',gender = 'male'),dict(name = 'Linda',gender = 'female')]
for item in d:                                    # 添加表中的内容
    row_cells = table.add_row().cells
    row_cells[0].text = str(item['name'])
    row_cells[1].text = str(item['gender'])

document.add_page_break()                         # 添加分页

document.save('demo1.docx')                       # 保存到路径
```

使用 Office Word 软件打开 demo1.docx,效果如图 3-11 所示。

除了.doc 文件以外,在采集网络信息时用户还可能会遇到处理 PDF 文件的需求(在某些场合尤其常见,例如下载 slide 或者 paper 时)。在 Python 中有对应的库来操作 PDF 文件,这里使用 PyPDF2 来解决这个需求(使用 pip install PyPDF2 即可安装)。

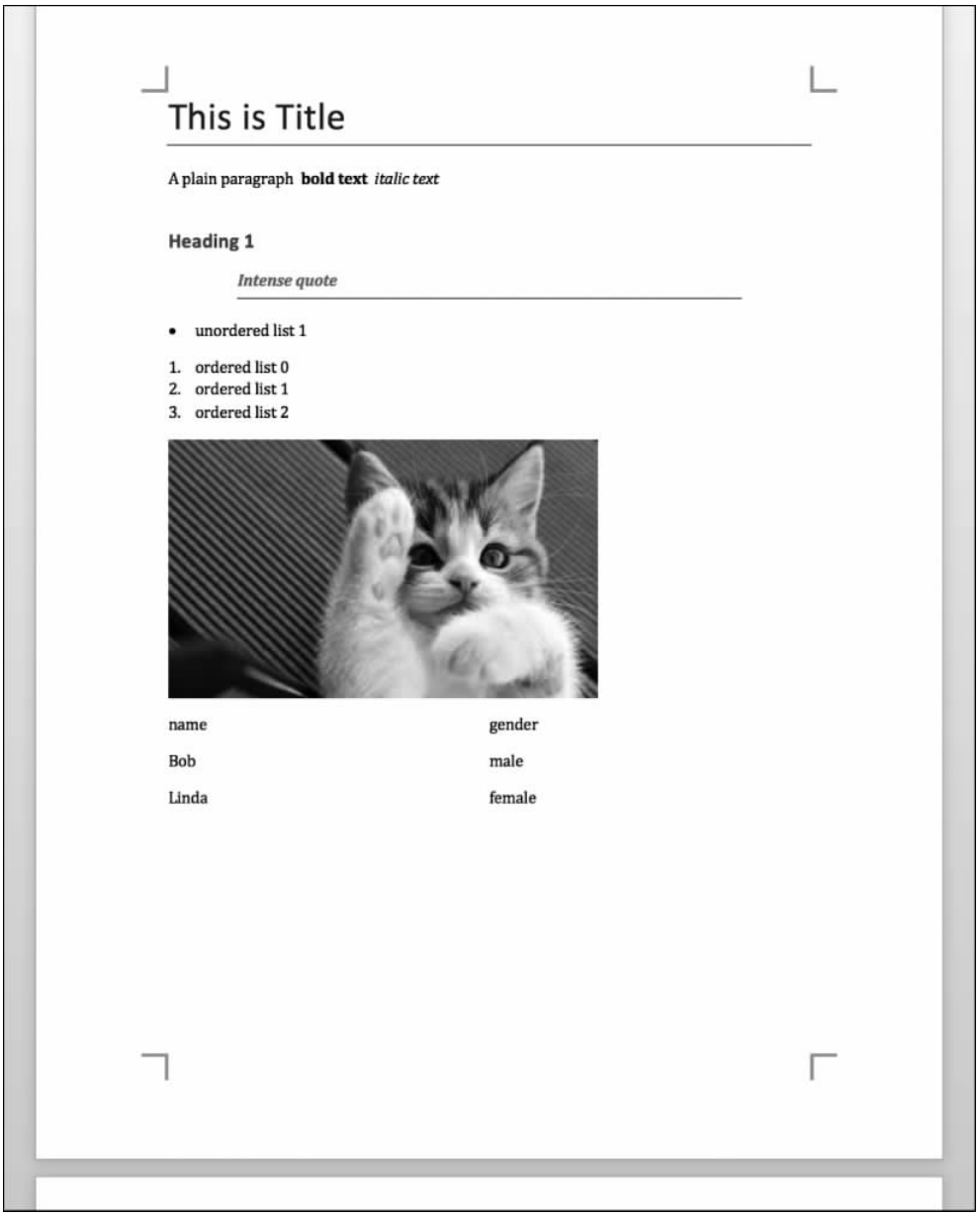
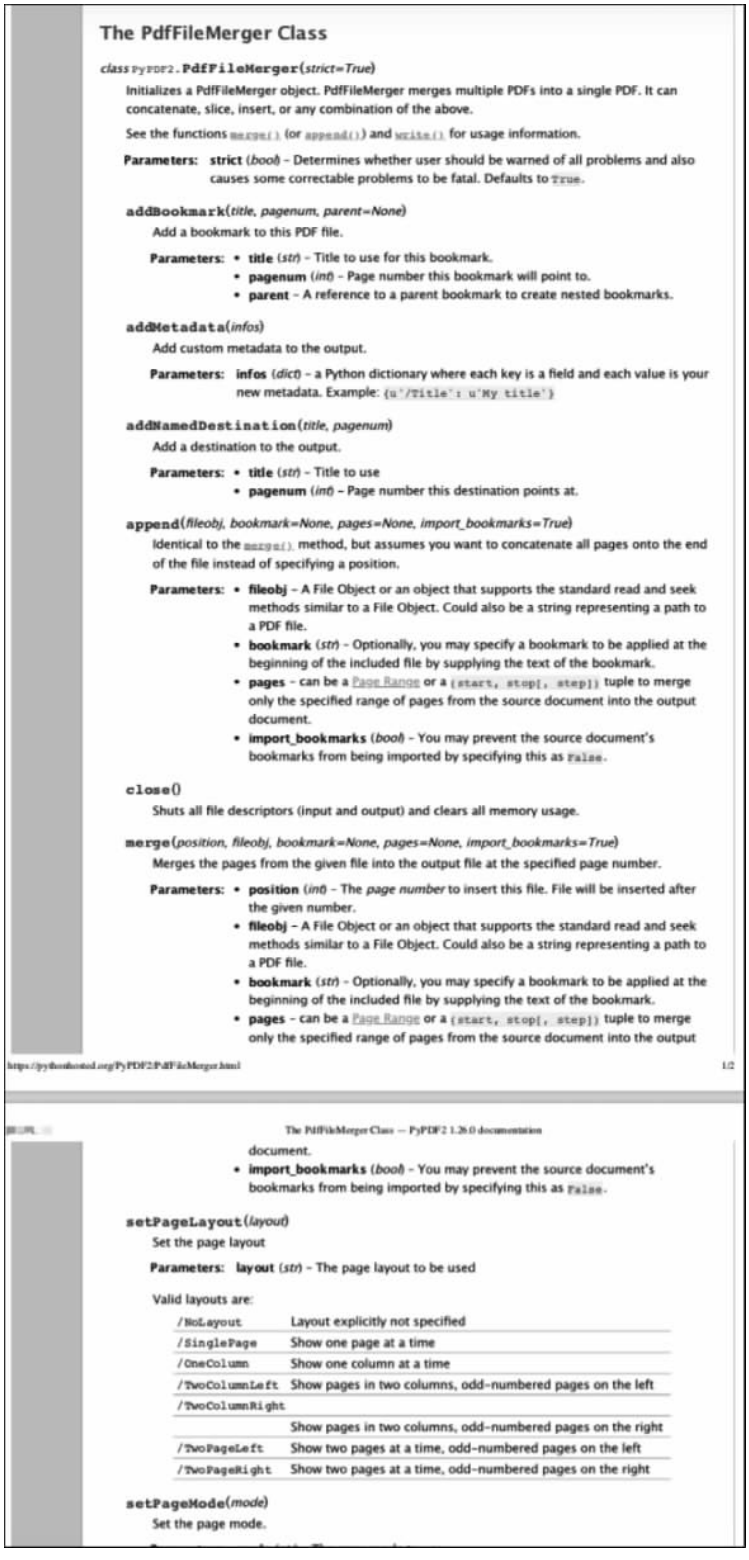


图 3-11　新建文档的内容

首先可以通过浏览器的打印页面方式生成一个内容为网页的 PDF 文件，此处将“https://pythonhosted.org/PyPDF2/PdfFileMerger.html”这个地址的网页内容保存在 raw.pdf 中，如图 3-12 所示。

接着使用 PyPDF2 进行简单的 PDF 页码粘贴与 PDF 合并操作：



The PdfFileMerger Class

class PyPDF2.**PdfFileMerger**(*strict=True*)

Initializes a PdfFileMerger object. PdfFileMerger merges multiple PDFs into a single PDF. It can concatenate, slice, insert, or any combination of the above.

See the functions `merge()` (or `append()`) and `write()` for usage information.

Parameters: **strict** (*bool*) – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to `True`.

addBookmark(*title, pagenum, parent=None*)

Add a bookmark to this PDF file.

Parameters:
- **title** (*str*) – Title to use for this bookmark.
- **pagenum** (*int*) – Page number this bookmark will point to.
- **parent** – A reference to a parent bookmark to create nested bookmarks.

addMetadata(*infos*)

Add custom metadata to the output.

Parameters: **infos** (*dict*) – a Python dictionary where each key is a field and each value is your new metadata. Example: `{u'/Title': u'My title'}`

addNamedDestination(*title, pagenum*)

Add a destination to the output.

Parameters:
- **title** (*str*) – Title to use
- **pagenum** (*int*) – Page number this destination points at.

append(*fileobj, bookmark=None, pages=None, import_bookmarks=True*)

Identical to the `merge()` method, but assumes you want to concatenate all pages onto the end of the file instead of specifying a position.

Parameters:
- **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.
- **bookmark** (*str*) – Optionally, you may specify a bookmark to be applied at the beginning of the included file by supplying the text of the bookmark.
- **pages** – can be a Page Range or a `(start, stop[, step])` tuple to merge only the specified range of pages from the source document into the output document.
- **import_bookmarks** (*bool*) – You may prevent the source document's bookmarks from being imported by specifying this as `False`.

close()

Shuts all file descriptors (input and output) and clears all memory usage.

merge(*position, fileobj, bookmark=None, pages=None, import_bookmarks=True*)

Merges the pages from the given file into the output file at the specified page number.

Parameters:
- **position** (*int*) – The *page number* to insert this file. File will be inserted after the given number.
- **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.
- **bookmark** (*str*) – Optionally, you may specify a bookmark to be applied at the beginning of the included file by supplying the text of the bookmark.
- **pages** – can be a Page Range or a `(start, stop[, step])` tuple to merge only the specified range of pages from the source document into the output

https://pythonhosted.org/PyPDF2/PdfFileMerger.html 1/2

The PdfFileMerger Class — PyPDF2 1.26.0 documentation

document.
- **import_bookmarks** (*bool*) – You may prevent the source document's bookmarks from being imported by specifying this as `False`.

setPageLayout(*layout*)

Set the page layout

Parameters: **layout** (*str*) – The page layout to be used

Valid layouts are:

`/NoLayout`	Layout explicitly not specified
`/SinglePage`	Show one page at a time
`/OneColumn`	Show one column at a time
`/TwoColumnLeft`	Show pages in two columns, odd-numbered pages on the left
`/TwoColumnRight`	Show pages in two columns, odd-numbered pages on the right
`/TwoPageLeft`	Show two pages at a time, odd-numbered pages on the left
`/TwoPageRight`	Show two pages at a time, odd-numbered pages on the right

setPageMode(*mode*)

Set the page mode.

图 3-12　raw.pdf 的内容

```
from PyPDF2 import PdfFileReader, PdfFileWriter
raw_pdf = 'raw.pdf'
out_pdf = 'out.pdf'

# PdfFileReader 对象
pdf_input = PdfFileReader(open(raw_pdf, 'rb'))

page_num = pdf_input.getNumPages()              # 页数,输出 2
print(page_num)
print(pdf_input.getDocumentInfo())              # 文档信息
# 输出{'/Creator': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3 ) AppleWebKit/537.36
(KHTML, like Gecko)
#   Chrome/65.0.3325.181 Safari/537.36', '/Producer': 'Skia/PDF m65', '/CreationDate':
# "D:20180425142439+00'00'", '/ModDate': "D:20180425142439+00'00'"}

# 返回一个 PageObject
pages_from_raw = [pdf_input.getPage(i) for i in range(2)]
# raw.pdf 共两页,这里取出这两页

# 获取一个 PdfFileWriter 对象
pdf_output = PdfFileWriter()
# 将一个 PageObject 添加到PdfFileWriter 中
for page in pages_from_raw:
  pdf_output.addPage(page)
# 输出到文件中
pdf_output.write(open(out_pdf, 'wb'))

from PyPDF2 import PdfFileMerger, PdfFileReader
# 合并两个 PDF 文件
merger = PdfFileMerger()
merger.append(PdfFileReader(open('out.pdf', 'rb')))
merger.append(PdfFileReader(open('raw.pdf', 'rb')))
merger.write("output_merge.pdf")
```

最后打开 output_merge.pdf,发现已经成功地合并了 out.pdf 与 raw.pdf,由于 out.pdf 是 raw.pdf 中两页的完全复制,所以最终的效果是 raw.pdf 的两页内容的重复(共 4 页,见图 3-13)。

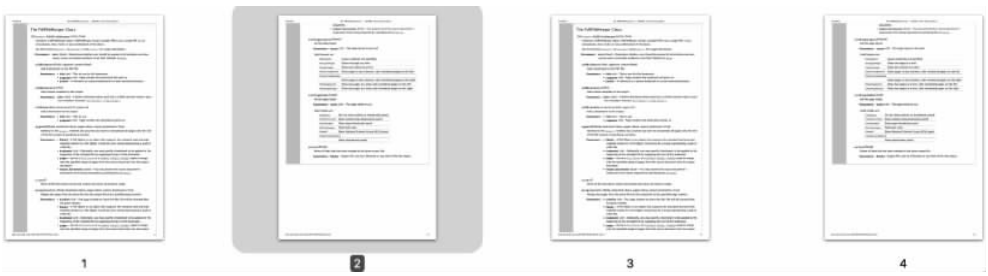


图 3-13　output_merge.pdf 文件的内容

3.7 本章小结

在本章中主要讨论了 Python 与各种文件的一些操作，首先介绍了最基本的文件打开与读写操作，之后通过图片文件以及 CSV、DOCX、PDF 等格式的文件展示了 Python 中文件处理的丰富功能。本章还系统性地介绍了一些数据库交互的方法，其中有关 MySQL 和 Redis 的部分对爬虫程序的编写尤为重要。