



《Python程序设计基础教程（微课版）》

<http://dblab.xmu.edu.cn/post/python>

第10章 基于文件的持久化





提纲

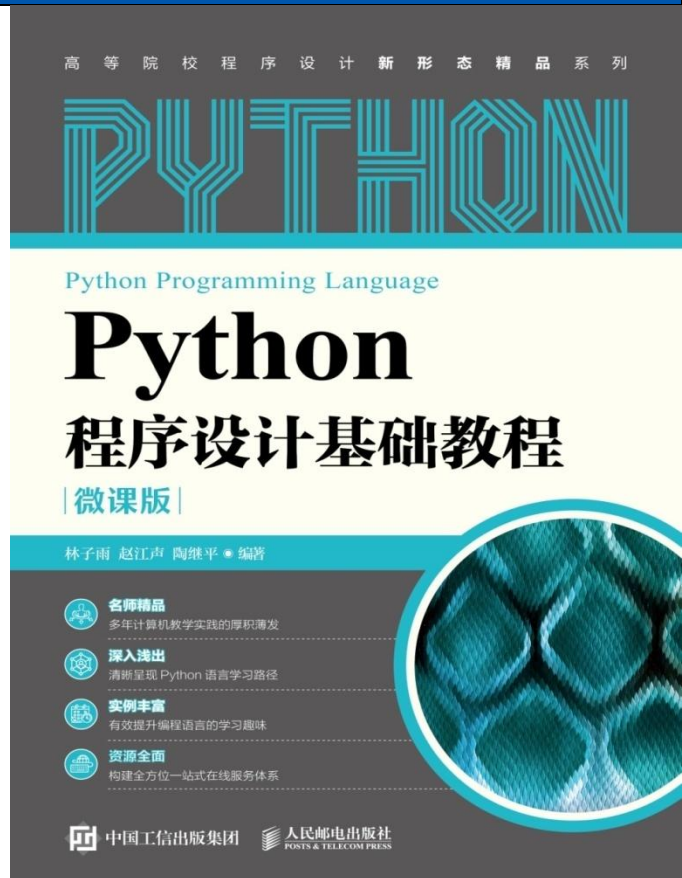
- 10.1 持久化前的准备工作
- 10.2 数据的序列化和反序列化
- 10.3 基于Windows系统的文件和路径
- 10.4 Python对目录的操作
- 10.5 Python对文件的操作
- 10.6 Python对文件内容的操作

本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：
<http://dblab.xmu.edu.cn/post/python>





10.1 持久化前的准备工作

- 在数据持久化前，需要先准备好哪些数据需要持久化。
- 比如，一个RPG游戏可能需要准备好玩家的个人信息、地图上所有怪物的信息、玩家的背包信息等。
- 这些信息需要按照编程者的意愿保存在某段内容空间或者某个变量里。
- 比如，可以用一个字典把这些数据都包含起来。
- 假定系统需要保存一张名片，可能就需要如下所示的字典：

```
>>> vCard = {'firstName': '名', 'lastName': '姓', 'title': '职位',  
'mobilePhoneNumber': '13322223333', 'organization': '公司名',  
'weChatNumber': '微信号'}
```



10.1 持久化前的准备工作

- 经过如此操作，就可以将系统需要持久化的数据整理到一个变量里。
- 之后，就可以将这个名片持久化保存。
- 在使用时，将上述数据导入到微信小程序里的“`wx.addPhoneContact`”函数中，就可以直接将其保存到手机通讯录里。
- 在收集数据时，建议使用最终的的数据结构是“字典”，比如上例中的 **vCard**。这种结构最通用，也最方便，在“字典”中还可以嵌套使用字典、元组或列表等其它数据结果。
- 嵌套的数据结构推荐使用字典、元组或者列表。当然，这只是一个建议，在实际使用过程中，应该按照需求和设计师所给定的方法进行持久化。



10.2 数据序列化和反序列化

数据序列化是将内存里的数据变成一个容易写入文件的形式。那么什么样的形式才是容易写入文件的形式就非常重要了。在现行软件体系中，写入文件的形式主要可以分为“字符流”和“字节流”两种，具体如下：

（1）所谓“字符流”指的是，将数据内容使用字符的形式保存到文件中，在读取过程中，读取到的所有内容都是字符。这是一种非常常见的文件保存形式，比如py文件、记事本文件等，直接打开这些文件可以看到里面所有的内容都是一个个字符。常用的字符流文件包括INI、HTML、XML和JSON等。后面将会介绍JSON格式的序列化方式。



10.2 数据序列化和反序列化

(2) 所谓“字节流”指的是，将数据内容以字节的形式保存到文件中，在读取过程中需要先读取字节，再将字节按照一定的规则转换为可以识别的内容。在计算机系统中，越是底层，这种方法就越常用，因为以这种形式保存的文件直接就是二进制数据。所以，在系统的底层，比如网络数据的发送和接收，都是将数据按照指定格式序列化为“字节流”再进行发送和接收。后面将会介绍如何使用Python模块库里的“pickle”对数据进行序列化。



10.2 数据序列化和反序列化

10.2.1 使用JSON对数据进行序列化和反序列化

10.2.2 使用pickle对数据进行序列化和反序列化

10.2.3 两种序列化方式的对比

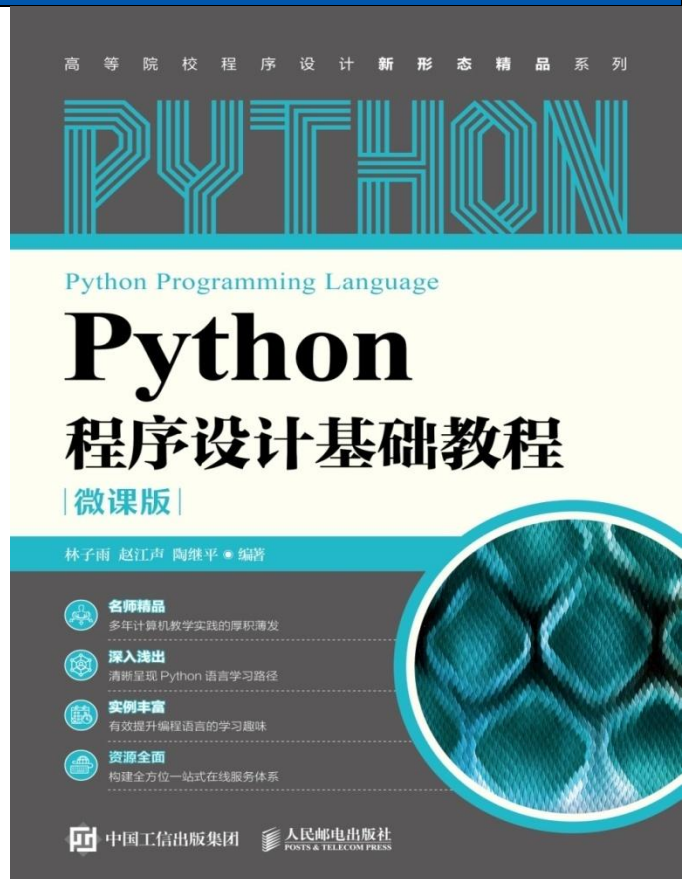
本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：

<http://dblab.xmu.edu.cn/post/python>





10.2.1 使用JSON对数据进行序列化和反序列化

在“字符流”这种形式中，目前最热门的格式应该就是JSON了。同样地，在Python中，也有相应的模块进行处理。首先，看一下使用JSON序列化后的数据：

```
>>> import json
>>> dict = {'name': 'Tom', 'age': 23}
>>> data = json.dumps(dict, indent=4)
>>> print(data)
{
    "name": "Tom",
    "age": 23
}
>>> data = json.dumps(dict)
>>> print(data)
{"name": "Tom", "age": 23}
```




10.2.1 使用JSON对数据进行序列化和反序列化

- 在上面的实例中，在使用JSON时，需要先引用“json”模块。
- 可以看到，JSON序列化之后的结果，同Python里的字典几乎完全一样。
- 这是因为，高级编程软件里，大多数的字典的数据模型都是按照与JSON类似的格式进行设计的。



10.2.1 使用JSON对数据进行序列化和反序列化

但是，二者还是有一定的区别：

- (1) 在Python的字典里，键可以是任何可以哈希的对象，而在JSON里，键必须是一个由双引号包裹的字符串；
- (2) JSON的键是有序的，可重复的。但是，在Python的字典里，键是不可以重复的；
- (3) Python里值可以是元素或者列表，但是JSON里值就只能是数组或者另一个列表；
- (4) Python里值可以是任意编码的任意字符，而JSON的值里如果使用Unicode编码，就必须使用“\uxxxx”的形式书写字符，具体如下所示：

```
>>> dict = {'name': '中文字符', 'age': 23}
>>> print(json.dumps(dict))
{"name": "\u4e2d\u6587\u5b57\u7b26", "age": 23}
```



10.2.1 使用JSON对数据进行序列化和反序列化

(5) 在Python里，使用True、False和None分别表示真、假和空值。在JSON中则是使用true、false和null来表示（这里要注意真和假的大小写），具体如下所示：

```
>>> dict = {'name': '某人', 'boy': True, 'girl': False, 'title': None}
>>> print(json.dumps(dict))
{"name": "\u67d0\u4eba", "boy": true, "girl": false, "title": null}
```



10.2.1 使用JSON对数据进行序列化和反序列化

在序列化时，使用的是**dumps**函数，该函数原型是：

```
json.dumps(obj, *, skipkeys=False, ensure_ascii=True,  
check_circular=True, allow_nan=True, cls=None, indent=None,  
separators=None, default=None, sort_keys=False, **kw)
```



10.2.1 使用JSON对数据进行序列化和反序列化

其中，在使用过程中，值得注意的参数有：

- (1) **obj**: 需要序列化的对象；
- (2) **skipkeys**: 默认为假。如果设为真，那么在序列化时所有不是基本对象 (**str**, **int**, **float**, **bool**, **None**) 的字典键将会被跳过，否则这些不是基本对象的字典键会引发一个异常；
- (3) **indent**: 默认为**None**。将**indent**值设置为一个非负整数或者字符串时，生成的**JSON**字符串将会使用指定数目的空格或者指定字符串对输出内容进行美化。默认使用**None**生成的**JSON**字符串是用于持久化的紧凑字符串，设置了这个值则输出适合用户阅读的美观字符串。
- (4) **sort_keys**: 默认为假。如果设置为真，那么输出的**JSON**字符串的键将会被排序。



10.2.1 使用JSON对数据进行序列化和反序列化

- 其它参数平时一般不会使用到，建议按照默认设置就可以。
- 此函数输出的是一个字符串，这个字符串称为“JSON字符串”或者“JSON”，这个字符串就是之后用来保存到持久化场所的字符串。



10.2.1 使用JSON对数据进行序列化和反序列化

在序列化之后，则需要使用loads函数进行反序列化，实例如下：

```
>>> string = '{"name": "\u67d0\u4eba", "boy": true, "girl": false, "title":  
null}'  
>>> print(json.loads(string))  
{'name': '某人', 'boy': True, 'girl': False, 'title': None}
```

可以看到，反序列化的结果就是原来的对象。loads函数虽然也有很多可选参数，不过一般情况下都用不到，这里不再特别说明。



10.2.2 使用pickle对数据进行序列化和反序列化

- 使用字符流对数据进行序列化后，返回的结果是一个字符串。
- 字符串本身是可以阅读和理解的，并且只要理解了内容之后，就可以对字符串进行修改。
- 每个字符串根据编码占用固定的字节数，但是，其实际表达的含义往往不需要使用如此之多的字节数。比如数字1，只需要1个字节就可以表示。
- 但是，如果使用UTF-8编码的Unicode就需要两个字节来表示。
- 此外，按字符流序列化的结果虽然易读，但却不方便查询。所以，这时需要一种更直接的、更底层的序列化方式，也就是“字节流”。



10.2.2 使用pickle对数据进行序列化和反序列化

在Python里，自定义编码的字节流需要程序员在使用时自行编码实现。不过，Python本身也为程序员们提供了一个基本的模块“**pickle**”来帮助实现对数据的序列化和反序列化，具体实例如下：

```
>>> import pickle
>>> vCard = {'firstName': '名', 'lastName': '姓', 'title': '职位', 'mobilePhoneNumber':
'13322223333', 'organization': '公司名', 'weChatNumber': '微信号'}
>>> bytes = pickle.dumps(vCard)
>>> print(bytes)
b'\x80\x04\x95\x91\x00\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\tfirstName\x94\x8c\x03\
xe5\x90\x8d\x94\x8c\x08lastName\x94\x8c\x03\xe5\xa7\x93\x94\x8c\x05title\x94\x8
c\x06\xe8\x81\x8c\xe4\xbd\x8d\x94\x8c\x11mobilePhoneNumber\x94\x8c\x0b1332
2223333\x94\x8c\x0corganization\x94\x8c\t\xe5\x85\xac\xe5\x8f\xb8\xe5\x90\x8d\x
94\x8c\x0cweChatNumber\x94\x8c\t\xe5\xbe\xae\xe4\xbf\xa1\xe5\x8f\xb7\x94u.'
```



10.2.2 使用pickle对数据进行序列化和反序列化

可以看到，序列化结果是一个无法读取的字节串。这个串的本质是一段二进制代码，所以其本身并不安全。如果恶意攻击者精心构建了一个特殊的字符串，在将其序列化后，这个序列化的结果甚至可以被当成可执行文件直接执行。这个可执行文件很可能就是恶意攻击者的攻击软件，或者具有特殊目的的程序。

所以，在使用**pickle**时，一定要注意，这是一个非常危险的操作，如果你不信任数据的来源，那么无论如何都不应该使用这个函数序列化和反序列化数据。甚至于，在序列化之后，需要使用**MD5**或者**SHA-1**等算法对数据进行签名，在反序列化之前，应当验证这个签名是否正确，以保证数据没有被篡改。再次强调，在反序列化过程中，恶意攻击可以通过反序列化过程直接执行恶意代码。



10.2.2 使用pickle对数据进行序列化和反序列化

使用pickle进行反序列化的具体实例如下：

```
>>> bytes =
```

```
b'\x80\x04\x95\x91\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\tfirstName\x94\x8c\x03\xe5\x90\x8d\x94\x8c\x08lastName\x94\x8c\x03\xe5\xa7\x93\x94\x8c\x05title\x94\x8c\x06\xe8\x81\x8c\xe4\xbd\x8d\x94\x8c\x11mobilePhone  
Number\x94\x8c\x0b13322223333\x94\x8c\x0corganization\x94\x8c\t\xe5\x85\xac\xe5\x8f\xb8\xe5\x90\x8d\x94\x8c\x0cweChatNumber\x94\x8c\t\xe5\xbe\xae\xe4\xbf\xa1\xe5\x8f\xb7\x94u.'
```

```
>>> obj = pickle.loads(bytes)
```

```
>>> print(obj)
```

```
{'firstName': '名', 'lastName': '姓', 'title': '职位', 'mobilePhoneNumber':  
'13322223333', 'organization': '公司名', 'weChatNumber': '微信号'}
```



10.2.3 两种序列化方式的对比

前面介绍了JSON和pickle两种序列化方式，这里对这两种方式进行一个对比，具体如下：

(1) JSON是一个字符串文件，pickle的结果是一个字节流，这使得在文件大小上，保存同样的Unicode字符，pickle会相对较小一些，下面实例演示了二者的文件大小：

```
>>> import json
>>> import pickle
>>> obj = {'firstName': '名', 'lastName': '姓', 'title': '职位', 'mobilePhoneNumber':
'13322223333', 'organization': '公司名', 'weChatNumber': '微信号'}
>>> js = json.dumps(obj)
>>> len(js)
182
>>> pi = pickle.dumps(obj)
>>> len(pi)
156
```



10.2.3 两种序列化方式的对比

(2) 从输出结果上来看，JSON是易读的，pickle是不易读的。但是必须注意，这里所说的易读指的是人类的可读性，而不是指它们的安全性。下面是二者输出结果的对比：

```
>>> obj = {'firstName': '名', 'lastName': '姓', 'title': '职位', 'mobilePhoneNumber':  
'13322223333', 'organization': '公司名', 'weChatNumber': '微信号'}  
>>> js = json.dumps(obj)  
>>> pi = pickle.dumps(obj)  
>>> print(js)  
{"firstName": "\u540d", "lastName": "\u59d3", "title": "\u804c\u4f4d",  
"mobilePhoneNumber": "13322223333", "organization": "\u516c\u53f8\u540d",  
"weChatNumber": "\u5fae\u4fe1\u53f7"}  
>>> print(pi)  
b'\x80\x04\x95\x91\x00\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\tfirstName\x94\x8c\x03\xe5\x90\x8d\x94\x8c\x08lastName\x94\x8c\x03\xe5\xa7\x93\x94\x8c\x05title\x94\x8c\x06\xe8\x81\x8c\xe4\xbd\x8d\x94\x8c\x11mobilePhoneNumber\x94\x8c\x0b13322223333\x94\x8c\x0corganization\x94\x8c\t\xe5\x85\xac\xe5\x8f\xb8\xe5\x90\x8d\x94\x8c\x0cweChatNumber\x94\x8c\t\xe5\xbe\xae\xe4\xbf\xa1\xe5\x8f\xb7\x94u.'
```



10.2.3 两种序列化方式的对比

(3) JSON广泛地应用于各大语言中，而pickle只能用在Python里。JSON是一个应用及其广泛的序列化方式，包括C、C++、Javascript、Java、C#和Python等常见的编程语言都有关于JSON的操作库，但pickle只能用在Python里。从这个角度上来说，如果序列化的结果需要发给其它软件或者系统使用，应当使用JSON。而如果结果只是给Python使用，则可以根据情况使用pickle。

(4) 在默认情况下，JSON不支持自定义类，它只能表示Python内置类型的子集。如果需要序列化一个自定义类，JSON需要将类里的数据整理成一个Python内置类型，比如字典，然后再进行序列化。而pickle则不需要，它可以直接序列化大部分的Python数据类型。



10.2.3 两种序列化方式的对比

（5）尽管**pickle**有各种好处，但是**pickle**的反序列化操作是不安全的，在反序列化过程中，如果不小心碰到了攻击者的恶意代码，那么这些代码有可能被执行。但是，**JSON**的反序列化过程是安全的。

（6）虽然之前再三强调**pickle**的安全性问题，但是，这也需要攻击者精心构建一些恶意代码。如果程序员编写的是一个自己使用的软件，那么为了方便，完全可以使用**pickle**，不应该“因噎废食”。反之，如果程序员写的应用是公开在互联网上的，甚至这些序列化数据需要被另一个用户所使用（比如游戏软件里的存档，可以复制给其它人），那么，这时就必须考虑到**pickle**的安全性问题。



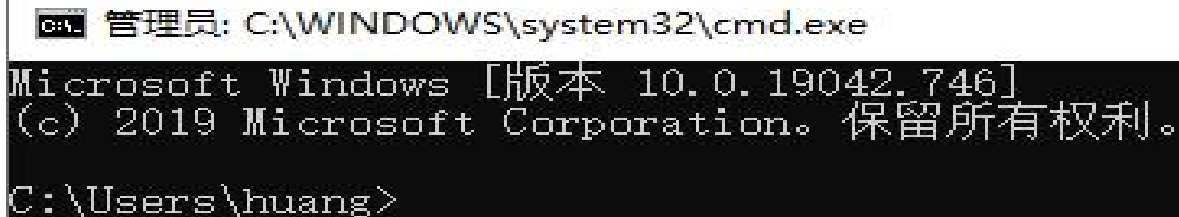
10.3 基于Windows系统的文件和路径

- 在将数据序列化之后，应当将数据保存到持久化场所中。最常见的持久化场所非计算机文件莫属。那么，为了将数据保存到文件中去，应当先认识和理解什么是计算机文件的路径。
- 计算机文件的路径是基于操作系统的，所以在讨论计算机文件的路径时，离开操作系统就没有任何意义。比如，在Unix系统里，CD驱动器的目录是“/dev/cdrom”。而在Windows系统里，CD驱动器则需要分配一个盘符才能访问。这里只介绍Windows系统的路径。



10.3 基于Windows系统的文件和路径

一个操作系统中可以保存若干个文件，这些文件往往保存在不同的目录之下，编程者需要使用“路径”来查询每一个文件所在的位置。在操作系统里，路径可以分为两种，即绝对路径和相对路径。接下来将分别介绍这两种路径。但是，在介绍路径之前，需要先介绍一个关键性的概念——“当前目录”。这个目录如果没有特别说明，指的是现在执行文件所在的目录。在Windows系统的命令行模式下，“当前目录”指的就是光标之前所描述的目录，如图10-1所示，当前目录是“C:\Users\huang”。



```
C:\> 管理员: C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.19042.746]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\huang>
```

图10-1 当前目录



10.3 基于Windows系统的文件和路径

在IDLE里，启动进入IDLE以后默认是当前目录是Python的安装目录。比如，启动进入IDLE以后，可以使用如下语句打印出当前目录的绝对路径：

```
>>> import os
>>> print(os.path.abspath('.'))
C:\Python38
```

如果是在IDLE中执行Python文件，那么当前目录就是Python文件所在的目录，如图10-2所示，当前执行了hello.py，该代码文件所在目录是“C:\Python38\mycode”，因此，当前目录就是“C:\Python38\mycode”。

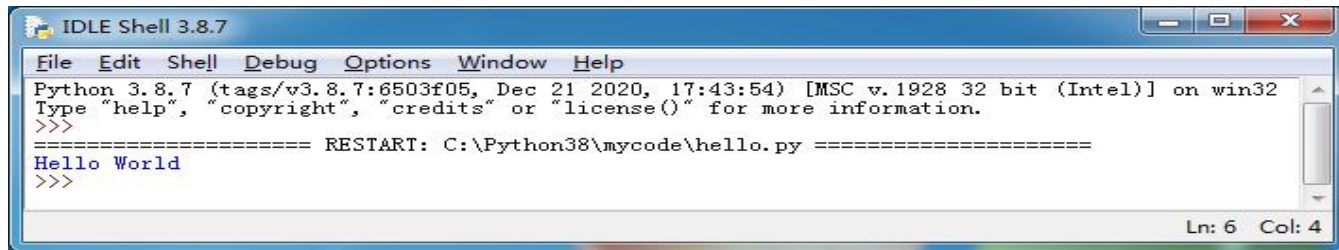


图10-2 执行Python文件



10.3 基于Windows系统的文件和路径

10.3.1 Windows里的绝对路径

10.3.2 Windows里的相对路径

10.3.3 Windows里的环境变量

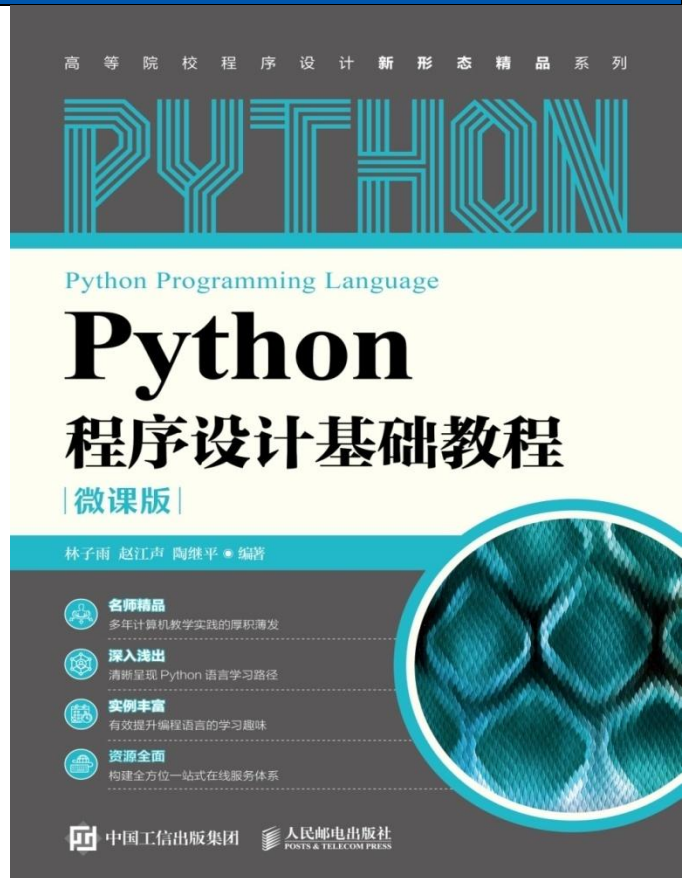
本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：

<http://dblab.xmu.edu.cn/post/python>





10.3.1 Windows里的绝对路径

- 所谓“绝对路径”，指的是无论当前目录是在哪个目录之下，都可以使用这个路径找到编程者所指定的那个文件。
- 在Windows里，文件的绝对路径是从盘符开始的。比如“C:\Windows\System32\kernel64.dll”这个文件路径里，“C:\”就是盘符，它说明这个文件是保存在哪个驱动器里的。需要注意的是，与目录不同，在Python里，盘符必须以“\”结尾，“C:”不是一个合法的盘符，必须要写成“C:\”才可以。



10.3.1 Windows里的绝对路径

- 紧跟在盘符之后的是目录名。目录名按照级别一级级地向下排列，并且以“\”结尾。比如，“C:\Windows\System32\”指的就是C盘下的Windows目录下的System32目录。
- 在实际使用过程中，最后一层目录除非后面跟着文件名，否则不需要加“\”。这是因为，在书写代码时，“\”有转义符的意思，如果书写目录最后追加了“\”，那么就会出现“\\”或者“\'”的情况，这是一个代表着引号的转义符。



10.3.1 Windows里的绝对路径

- 跟在目录名之后的就是文件。文件包含文件名和后缀名（全称为“文件扩展名”，也可以称为“扩展名”）。
- 比如，“kernal64.dll”这个文件的文件名是“kernal64”，后缀名为“.dll”。所谓的“文件重名”，指的是文件的文件名加后缀名全部相同。
- 如果文件名相同，后缀名不同，也是两个不同的文件，比如“python.avi”和“python.mp4”就是两个完全不同的文件。



10.3.1 Windows里的绝对路径

- 在Windows文件系统中，文件名是用来标识文件的名称的，不能使用“/ \ : * " < > | ? ”这9个半角字符。
- 同时，文件名不能超过**255**个字符（一个中文占**2**个字符）。同时，扩展名部分同样也不能超过**255**个字符（一个中文占**2**个字符）。不过一般情况下，后缀名都是使用**3**个字符或者**4**个字符。
- 另外，如果有两个或者两个以上的后缀名，那么只有最后一个后缀名才视作文件的后缀名，前面所有的后缀名都要计算在文件名中。比如有文件“kernal64.dll.bak”，那么此文件的文件名是“kernal64.dll”，后缀名是“.bak”。



10.3.2 Windows里的相对路径

- 所谓“相对路径”，指的是相对某个目录而言一个路径的位置。这里的“某个目录”往往就是指当前目录。
- 假设存在一个如图10-3所示的多级目录，在“C:\”下有3个目录，分别是Python38、Windows和Program Files，在“C:\Python38”目录下有3个目录，分别是mycode、include和libs，在“C:\Python38\mycode”目录下有2个代码文件，分别是hello.py和test.py。

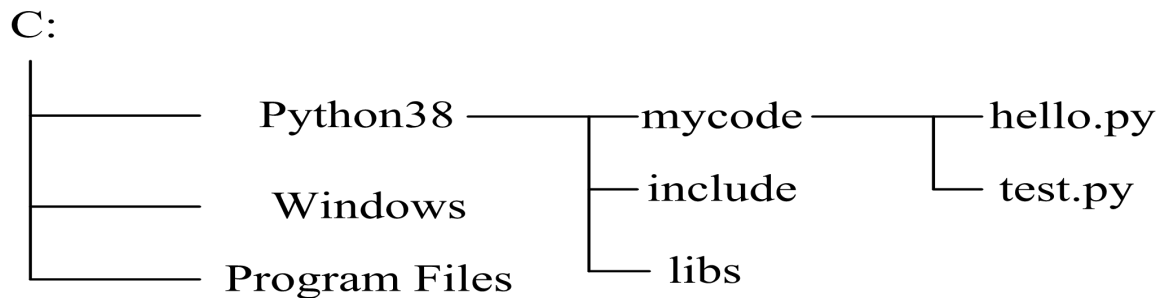


图10-3 一个多级目录实例



10.3.2 Windows里的相对路径

- 现在假设当前目录是“C:\Python38\mycode”，这个目录下有一个文件“hello.py”，这时，采用相对路径的表示方法，这个文件的路径就可以写为“.\hello.py”。
- 在Windows系统里，“.”这个目录指的就是当前目录。
在这里可以看到，如果当前目录不存在，那么相对路径就没有任何意义。
但绝对路径不同，无论当前目录是否存在，绝对路径都可以找到指定地址。
- 不过，幸运的是，在Windows系统中，几乎不可能出现找不到当前目录的情况。



10.3.2 Windows里的相对路径

- 除了可以使用“.”表示当前目录以外，还可以使用“..”表示当前目录的上一级目录。比如，当前目录是“C:\Python38\mycode”，则“..”就表示“C:\Python38”。“..”还可以连用，比如当前目录是“C:\Python38\mycode”，则“..\..”就表示“C:\”。
- 当然，“.”也是可以连用的，不过由于它表示的是当前目录，所以连用没有任何含义。毕竟当前目录的当前目录还是当前目录。



10.3.3 Windows里的环境变量

- 在Windows里，还有一些目录非常特殊，它们在安装完系统后就被指定，但是对于程序开发者来说却并不明确。比如“我的文档”目录，用户可以任意指定“我的文档”的位置，且每个系统一定有一个“我的文档”目录。再比如“Windows”文件夹，它可能在任何一个驱动器下，而且一定存在。这些目录就称为系统的“环境变量”。
- 使用这些目录时，就可以直接使用环境变量名来指代这些目录。比如，系统的安装盘，就可以使用环境变量“%SystemDrive%”来指代，如图10-4所示。

```
C:\Users\huang\AppData\Roaming>cd %SystemDrive%\windows
C:\Windows>
```

图10-4 使用环境变量



10.3.3 Windows里的环境变量

- 夹在两个百分号之间的“**SystemDrive**”就是环境变量的名称，它们可以在系统的环境变量中配置并且使用。使用时，只需要将环境变量名夹在两个百分号里就可以了。
- 同时，环境变量与系统版本是有关的，并不是所有系统都可以使用。比如，在Windows XP里，就不能使用“**%PROGRAMDATA%**”来查找隐藏在“**%SystemDrive%**”下的“**ProgramData**”目录。



10.3.3 Windows里的环境变量

表10-1展示了Windows10下可用的一些环境变量及其示例地址（这里默认将Windows安装在C盘下）。

表10-1 Windows 10中的一些环境变量

环境变量	说明
%HOMEDRIVE%	Windows所在驱动器，比如C:\
%windir%	Windows所在目录，比如C:\WINDOWS
%TEMP%	临时文件保存位置 比如C:\Users\[用户名]\AppData\Local\Temp
%ProgramFiles%	应用程序默认安装目录，比如C:\Program Files
%CommonProgramFiles%	应用程序配置文件存放目录，比如 C:\Program Files\Common Files
%APPDATA%	应用程序数据存放目录 比如C:\Users\[用户名]\AppData\Roaming
%HOMEPATH%	当前登录用户的主目录 比如C:\Users\[用户名]
%HOMEPATH%\desktop	用户桌面所在路径 比如C:\Users\[用户名]\桌面



10.4 Python对目录的操作

10.4.1 获取当前目录

10.4.2 转移到指定目录

10.4.3 新建目录

10.4.4 判断目录是否存在

10.4.5 显示目录内容

10.4.6 判断是目录还是文件

10.4.7 删除目录

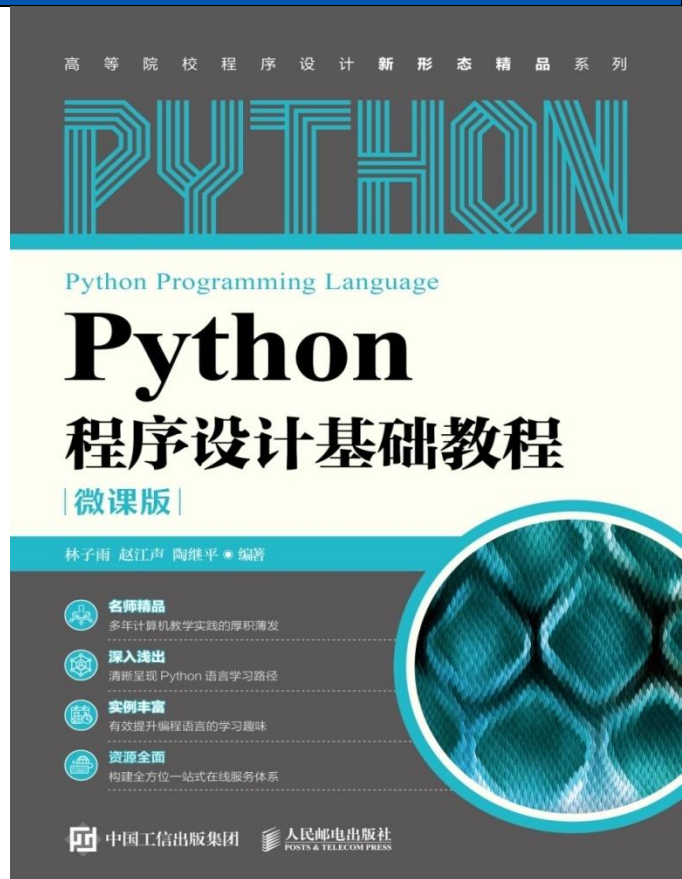
本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：

<http://dblab.xmu.edu.cn/post/python>





10.4.1 获取当前目录

对于Python的目录操作，首先应该了解的就是查询和修改当前目录。在Windows系统的命令行工具中，可以直接查看到当前目录。但是在Python中，需要使用`os.getcwd`函数获取当前目录，具体如下所示：

```
>>> import os  
>>> os.getcwd()  
'C:\\Python38'
```



10.4.2转移到指定目录

在一些情况下，需要将当前目录转移到指定路径，这时就需要使用`os.chdir`，这个函数相当于Windows系统中的“`cd`”命令，其作用是进入某个目录。

具体如下所示：

```
>>> # 假设已经存在 “C:\project” 目录
>>> os.chdir(r"c:\project")
>>> os.getcwd()
'c:\\project'
```

在使用该函数时，推荐使用原始字符串。不然，在书写路径时，所有“`\`”都需要转义，即麻烦又容易出错。



10.4.3新建目录

在修改当前目录时，有可能此目录并不存在，这时就需要使用新建目录的函数，即“**os.makedirs**”。该函数相当于Windows里的“**md**”命令，输入一个参数**path**，如果该目录不存在，则新建目录。在新建目录时，如果目录是嵌套状态，它会从最外层一直新建到最里层，非常方便，具体如下所示：

```
>>> # 在 “c:\project” 目录下创建 “md” 目录
```

```
>>> os.makedirs(r"c:\project\md")
```

```
>>> # 再次在 “c:\project” 目录下创建 “md” 目录
```

```
>>> os.makedirs(r"c:\project\md")
```

```
Traceback (most recent call last):
```

```
File "<pyshell#6>", line 1, in <module>
```

```
    os.makedirs(r"c:\project\md")
```

```
File "C:\Python38\lib\os.py", line 223, in makedirs
```

```
    mkdir(name, mode)
```

```
FileExistsError: [WinError 183] 当文件已存在时，无法创建该文件。: 'c:\\project\\md'
```



10.4.4 判断目录是否存在

从上面实例可以看出，在创建目录时，如果此目录已经存在，那么就无法创建成功，系统会报一个异常。所以在使用过程中，需要先判断目录是否存在，再创建目录。可以使用`os.path.exists`函数判断目录是否存在，具体如下所示：

```
>>> os.path.exists(r"c:\project\md")  
True
```



10.4.5显示目录内容

在了解目录存在后，往往需要知道目录里面有什么内容，这时就需要使用“`os.listdir`”函数，该函数相当于Windows里的“`dir`”命令。为了演示该命令的使用方法，这里在上面实例的基础上，在“`C:\project`”目录下自己手动新建一个没有扩展名的文件“`cd`”，新建一个文本文件`dir.txt`，再新建一个WORD文件`rm.docx`，然后执行如下语句：

```
>>> import os
>>> os.getcwd()
'c:\\project'
>>> os.listdir()
['cd', 'dir.txt', 'md', 'rm.docx']
```



10.4.6判断是目录还是文件

在上面的例子中，需要注意一点，“md”是一个目录，而“cd”是一个没有扩展名的文件。但是，在`listdir`的输出结果中，都是一个没有带点的字符串，很难判断到底是目录还是文件。所以，需要使用`os.path.isdir`和`os.path.isfile`函数来判断到底是目录还是文件，具体如下所示：

```
>>> os.getcwd()
'c:\\project'
>>> os.listdir()
['cd', 'dir.txt', 'md', 'rm.docx']
>>> os.path.isdir("md")
True
>>> os.path.isfile("cd")
True
```



10.4.7 删除目录

- 目录的删除操作需要使用“`os.rmdir`”函数。
- 这个函数与Windows中的“`rm`”命令一样，甚至连限制都是一样的。
- 在各个操作系统中，单纯的删除目录命令是不能删除一个有内容的目录的，即目录里只要有子目录或者文件，那么就不能删除，否则会报错。



10.4.7 删除目录

```
>>> os.getcwd()
'c:\\project'
>>> os.listdir()
['cd', 'dir.txt', 'md', 'rm.docx']
>>> # 在 “md” 目录下新建 “some” 目录
>>> os.makedirs(r"md\\some")
>>> os.listdir()
['cd', 'dir.txt', 'md', 'rm.docx']
>>> os.rmdir("md") # 删除 “md” 目录会返回错误信息，因为 “md” 是非空目录
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    os.rmdir("md")
OSError: [WinError 145] 目录不是空的。: 'md'
>>> os.rmdir(r".\\md\\some") # 删除 “md” 目录下的 “some” 目录
>>> # 这时，“md” 目录是一个空目录，下面就可以成功执行删除操作
>>> os.rmdir("md")
>>> os.listdir()
['cd', 'dir.txt', 'rm.docx']
```



10.4.7 删除目录

当然，除了非空目录不能删除以外，正在使用的目录也不能删除。比如，删除当前目录，肯定也是不能成功的，如下所示：

```
>>> os.rmdir(".")
```

```
Traceback (most recent call last):
```

```
File "<pyshell#1>", line 1, in <module>
```

```
    os.rmdir(".")
```

```
PermissionError: [WinError 32] 另一个程序正在使用此文件，进程无法访问。:'. '
```

在使用时，删除操作一定要做好异常处理，因为在编程时根本不知道这个目录有没有被其它程序占用。



10.4.7 删除目录

此外，在Python中，还提供了一个**shutil**模块，该模块里提供了一些文件和目录的高阶操作，其中就包含了可以级联删除目录的**shutil.rmtree**函数，具体实例如下：

```
>>> os.getcwd()
'c:\\project'
>>> os.makedirs(r"me\some")
>>> import shutil
>>> shutil.rmtree("me")
>>> os.listdir()
['cd', 'dir.txt', 'rm.docx']
```

同样地，**shutil.rmtree**函数在使用前也一定要做好异常处理，因为这个函数比**rmdir**更加有可能失败。



10.5 Python对文件的操作

10.5.1 打开文件

10.5.2 关闭文件

10.5.3 复制文件

10.5.4 重命名文件

10.5.5 删除文件

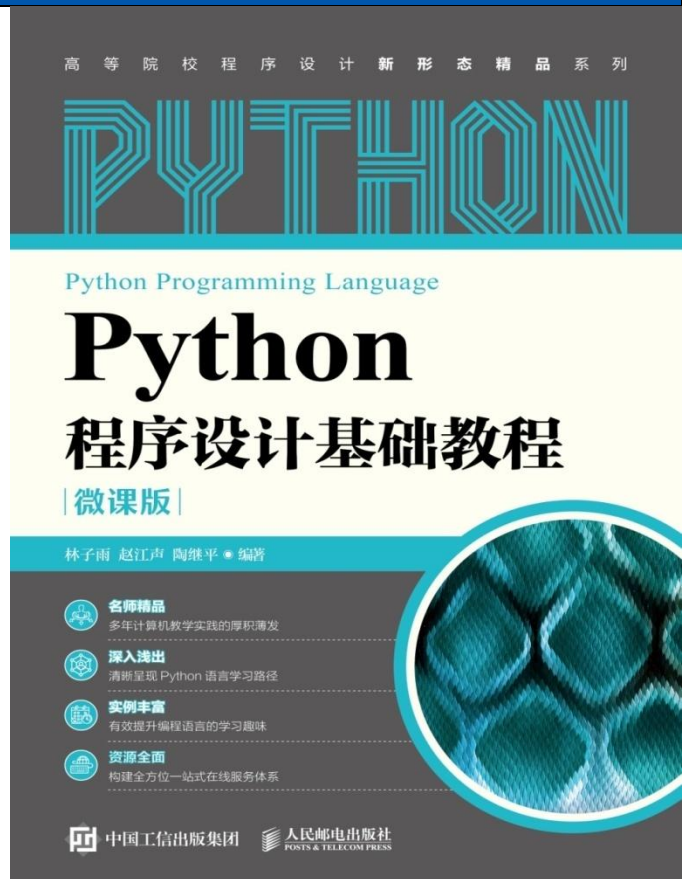
本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：

<http://dblab.xmu.edu.cn/post/python>





10.5.1 打开文件

在Python中，打开文件使用的是`open`函数，该函数不需要引用`os`模块就可以直接调用，它返回一个被称为“文件句柄”的对象，由这个对象来操作文件。只要产生了“文件句柄”，那么在释放这个句柄之前这个文件就被程序占用了，其它程序将无法对该文件进行增加、修改、查询和删除。如图10-5所示，使用Python语句“`f=open("dir.txt")`”打开了一个文件`dir.txt`以后，再在Windows资源管理器中打开`dir.txt`就会报错。

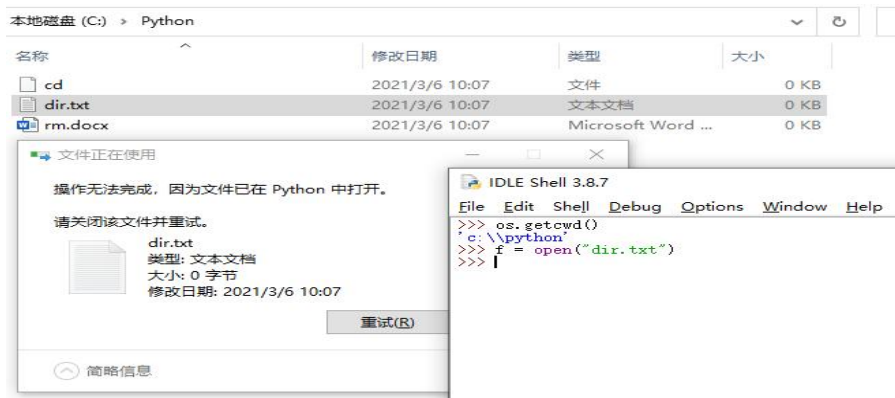


图10-5 文件句柄的占用



10.5.1 打开文件

在使用**open**函数时，不仅可以输入一个路径作为参数，还可以输入对文件的操作模式，表10-2给出了文件的各种操作模式。

表10-2 文件的操作模式

模式	描述
r	以只读的方式打开文件，不能写入
w	以写入的方式打开文件，不能读取信息。如果文件已经存在，清空文件内容，如果文件不存在则新建文件
a	以追加的方式打开文件，如果文件不存在则新建文件，不能读取文件
r+	以读写的方式打开文件，如果文件不存在则报异常
w+	以读写的方式打开文件，文件存在则清除内容，如果不存在则新建
a+	以追加的方式打开文件，可以读写内容
b	以二进制的方式打开文件，该模式只对Windows和DOS生效



10.5.1 打开文件

- 在编程时，一般不会使用到“b”模式。
- “r”、“w”、“a”三种模式与对应的“r+”、“w+”、“a+”比起来，前者只允许读或写，后者同时允许读写，这个需要根据实际需求来处理。
- “a”和“w”相比，使用“w”就相当于把文件清空，再从头开始写。
- 使用“a”则不清空文件，直接把内容加在最后面，需要根据实际情况使用。



10.5.1 打开文件

下面是具体实例：

```
>>> os.chdir(r"c:\python") #假设C盘下面已经存在python目录
```

```
>>> #当前目录中不存在dir.txt
```

```
>>> f = open("dir.txt")
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#6>", line 1, in <module>
```

```
    f = open("dir.txt")
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'dir.txt'
```

```
>>> #自己动手在“c:\python”目录下新建dir.txt文件，然后执行下面操作
```

```
>>> f = open("dir.txt")
```

```
>>> f.read()
```

```
"
```

```
>>> f.close()
```



10.5.1 打开文件

```
>>> f = open("dir.txt", "w")
>>> f.read() #以“w”方式打开文件，不能读取，否则会报错
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    f.read()
io.UnsupportedOperation: not readable
>>> f.write("123")
3
>>> f.close()
>>> #现在文件内容为123
>>> f = open("dir.txt", "w")
>>> f.close()
>>> #上面以“w”方式打开文件，文件内容会被清空
>>> f = open("dir.txt", "w")
>>> f.write("123")
3
>>> f.close()
```



10.5.1 打开文件

```
>>> #现在文件内容为123
>>> f = open("dir.txt", "a")
>>> f.write("456")
3
>>> f.close()
>>> #现在文件内容为123456
```



10.5.2 关闭文件

在打开文件后，一定要记得关闭文件，否则，其它程序就不能再操作文件了。关闭文件使用的是**close**函数。当然，每次都要书写**close**函数会很麻烦，所以可以使用**with**关键字，具体实例如下：

```
>>> with open("dir.txt", "w+") as f:  
        f.write("some text")
```

9

```
>>> f = open("dir.txt", "r")  
>>> f.read()  
'some text'  
>>> f.close()
```

可以看到，使用这种方式操作文件句柄时，只要**with**代码段结束了，句柄就自动释放。



10.5.3 复制文件

在使用`open`函数新建文件和进行处理之前，有可能需要先将某个文件复制一份，这时就需要一个文件的复制操作。在Python中，使用的是`shutil.copy`函数，这个函数的作用相当于Windows系统里的`copy`命令，具体实例如下：

```
>>> import shutil
>>> os.chdir(r"c:\project")
>>> os.getcwd()
'c:\\project'
>>> os.listdir()
['cd', 'dir.txt', 'rm.docx']
>>> shutil.copy("dir.txt", "dir1.txt")
'dir1.txt'
>>> os.listdir()
['cd', 'dir.txt', 'dir1.txt', 'rm.docx']
```



10.5.4 重命名文件

在复制文件后，有可能需要对被复制的文件进行改名，这时就需要使用 `os.rename` 函数。这个函数不仅可以修改文件名，还可以修改目录名。该函数的作用相当于 Windows 系统里的 “`ren`” 命令。具体实例如下：

```
>>> import shutil
>>> os.chdir(r"c:\project")
>>> os.getcwd()
'c:\\project'
>>> os.listdir()
['cd', 'dir.txt', 'dir1.txt', 'rm.docx']
>>> os.rename("dir1.txt", "listdir.txt")
>>> os.listdir()
['cd', 'dir.txt', 'listdir.txt', 'rm.docx']
```



10.5.5删除文件

如果文件已经操作完成需要删除，就需要使用`os.remove`函数，这个函数相当于Windows里的“`del`”命令。不过，在删除时就不需要用户确认了，所以准确地来说，应当是与“`del /q`”命令相同。具体实例如下：

```
>>> import shutil
>>> os.chdir(r"c:\project")
>>> os.getcwd()
'c:\\project'
>>> os.listdir()
['cd', 'dir.txt', 'listdir.txt', 'rm.docx']
>>> os.remove("listdir.txt")
>>> os.listdir()
['cd', 'dir.txt', 'rm.docx']
```



10.5.5删除文件

- 最后，需要特别注意的是，在任何编程语言中，对文件的操作总是会伴随各种各样的异常。
- 所以，无论是哪种操作，都应当事先判断文件是否存在，并且哪怕事先知道文件已经存在，也需要为其加上异常处理代码，因为这些文件有可能已经被别的文件占用。



10.6 Python对文件内容的操作

10.6.1 dump和read函数

10.6.2 write、seek和tell函数

10.6.3 writelines和readlines函数

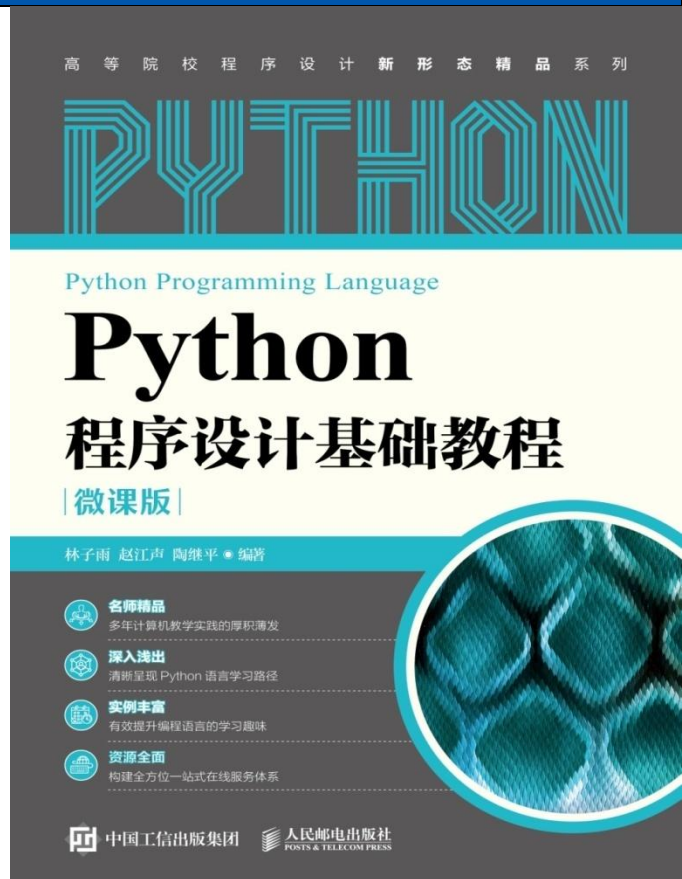
本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：

<http://dblab.xmu.edu.cn/post/python>





10.6.1 dump和read函数

- 在Python中，写入文件内容主要有两种方法：第一种是在序列化的同时将内容写入文件；第二种是使用文件句柄的**write**函数。
- 使用第一种方法时，需要使用序列化时用到的**dumps**函数，这个函数有一个名为**dump**的版本。在**dump**函数中，只需要在第一个参数后补一个文件句柄（当然，这个句柄必须是可写的，也就是说，使用“r”模式打开的不行），就可以直接将内容写入指定文件中，其它参数与**dumps**完全相同。无论是JSON还是pickle，都支持这种写法。



10.6.1 dump和read函数

具体实例如下：

```
>>> os.chdir(r"c:\python")
>>> vCard = {'firstName': '名', 'lastName': '姓', 'title': '职位',
'mobilePhoneNumber': '13322223333', 'organization': '公司名',
'weChatNumber': '微信号'}
>>> import json
>>> with open('vCard.json', 'w+') as f:
    json.dump(vCard, f)
    f.read()
"
```



10.6.1 dump和read函数

- 在上面这个实例中，我们使用“w+”模式打开了一个文件vCard.json，然后，使用语句“`json.dump(vCard,f)`”把数据写入文件，最后可以使用记事本打开查看vCard.json文件。
- 这个实例只演示了JSON的dump函数，pickle也是同理的。
- 当然，读取文件所使用的loads函数也有一个load版本，同样地，只需要输入一个文件句柄，就可以直接将文件的内容进行转换得到结果。



10.6.1 dump和read函数

- 在上面这个实例中，**read**函数用于读取文件内容。从这个实例可以看到，程序在打开文件并且写入之后，直接使用**read**函数读取是没有任何内容的。
- 这是因为，在Python打开文件后，会有一个“文件指针”，使用“r”、“w”、“r+”和“w+”的方式打开时，文件指针会指向文件的开头。在这个实例中，文件使用“w+”的方式打开，本身就会清空文件内容，并且“文件指针”指向了文件的开头，这是一个空位置。
- 然后，**dump**函数向文件里写入数据，在写入1个字符以后，文件指针就向后移动1位，一直到数据写入完成为止。这时，文件指针所在的位置是文件的结尾。这时使用**read**函数读取文件时，自然就读不到任何内容。



10.6.2 write、seek和tell函数

在Python中，写入文件内容的第二种方法是使用文件句柄的**write**函数。使用这种方法时，需要使用任意一种可写的方式（“r+”、“w”、“w+”、“a”或“a+”）打开文件，并且使用**write**函数将内容写入文件。具体如下所示：

```
>>> os.chdir(r"c:\python")
>>> os.getcwd()
'c:\python'
>>> with open('vCard.txt', 'w+') as f:
    f.write("姓名：张某某")
    f.read()
```



10.6.2 write、seek和tell函数

- 同样地，在使用**write**函数写入文件后，文件指针所在的位置也是在文件的结尾。使用**read**函数读取文件时，同样无法读取到任何内容。
- 所以，无论使用上述哪种方法写入文件，如果需要读取内容，就需要使用**seek**函数将文件指针回到开头位置，具体如下所示：

```
>>> os.chdir(r"c:\python")
```

```
>>> os.getcwd()
```

```
'c:\python'
```

```
>>> vCard = {'firstName': '名', 'lastName': '姓', 'title': '职位', 'mobilePhoneNumber':  
'13322223333', 'organization': '公司名', 'weChatNumber': '微信号'}
```

```
>>> with open("vCard.json", "w+") as f:
```

```
    json.dump(vCard, f)
```

```
    f.seek(0)
```

```
    f.read()
```

```
0
```

```
{  
  "firstName": "\u540d",  
  "lastName": "\u59d3",  
  "title": "\u804c\u4f4d",  
  "mobilePhoneNumber":  
    "13322223333",  
  "organization": "\u516c\u53f8\u540d",  
  "weChatNumber":  
    "\u5fae\u4fe1\u53f7"}  
}
```



10.6.2 write、seek和tell函数

- 在这里需要注意，使用**seek**函数后，再使用**write**函数或者**dump**函数写入数据时，是从当前文件指针的位置开始写入的。所以，在充分的考虑和规划之前，不要随意移动文件指针，否则会导致一些意想不到的结果。
- 在使用文件指针时，需要考虑到文件指针的移动肯定不能大于文件的总长度。所以，可以使用“**os.path.getsize**”函数来取文件总长度。此外，也可以在文件句柄中使用**tell**函数获取当前指针位置。



10.6.2 write、seek和tell函数

下面是一个具体实例，读取的是上面实例中生成的vCard.json:

```
>>> with open("vCard.json", "r+") as f:
```

```
    f.read(10)
```

```
    f.tell()
```

```
    f.read()
```

```
    f.tell()
```

```
'{"firstName'
```

```
10
```

```
'e': "\\u540d", "lastName": "\\u59d3", "title": "\\u804c\\u4f4d",
```

```
"mobilePhoneNumber": "13322223333", "organization":
```

```
"\\u516c\\u53f8\\u540d", "weChatNumber": "\\u5fae\\u4fe1\\u53f7"}'
```

```
182
```



10.6.3 writelines和readlines函数

除了上面介绍的read和write函数以外，操作文件内容也可以使用其它的函数。比如，Python还提供了writelines函数（注意这个函数名的line是复数形式的lines），该函数可以向文件中写入一个列表。具体实例如下：

```
>>> os.chdir(r"c:\python")
>>> os.getcwd()
'c:\python'
>>> f = open("t.txt", "w+")
>>> f.writelines(["第一行", "第二行\n", "第三行"])
>>> f.close()
```

这时打开文件“t.txt”，可以看到如下内容：

```
第一行第二行
第三行
```

之所以呈现上面的效果，是因为“第一行”后面没有加上“\n”，所以是不会自动换行的。



10.6.3 writelines和readlines函数

这里需要特别注意的是，虽然这个函数名字叫“**writelines**”，但其实这个函数并不会自动帮编程者加入换行符“**\n**”。该函数的作用仅仅是将一个列表中的所有元素按照顺序写入文件。

既然在Python中有**writelines**函数，当然也会有相对应的读取函数**readlines**。不过，考虑到读取时有时候需要一行一行地读取，所以Python还提供了只读取一行的函数**readline**（注意这个函数的**line**是单数形式，没有**s**）。具体实例如下：

```
>>> f = open("t.txt", "r+")
>>> f.readline()
'第一行第二行\n'
>>> f.seek(0)
0
>>> f.readlines()
['第一行第二行\n', '第三行']
```



10.6.3 writelines和readlines函数

- 上述两个函数在读取时同样会移动文件指针，所以上例中使用了**seek**函数进行重定向，以保证第二次读取也是从头开始读取。
- 这里有非常重要的一点需要注意，在**pickle**中，**dump**和**load**是互为反函数关系，被**dump**序列化的数据原来是什么样子，使用**load**反序列化出来以后的数据就是什么样子。
- 在**JSON**中，**dump**和**load**对于基本数据类型而言同样也是反函数关系。但是，**readlines**和**writelines**并不是互为反函数关系。
- 从上面这个实例可以看到，**readlines**读取的内容是一个只有两个元素的列表['第一行第二行\n', '第三行']，而实际写入的是有3个元素的列表["第一行", "第二行\n", "第三行"]。所以，在使用时需要特别注意。

The background is a solid blue gradient. It features several faint, light-blue silhouettes of people. In the top left, a group of four people is holding hands in a circle. In the top center, a group of seven people is holding hands in a line. On the right side, a person is shown in profile, looking towards the center. In the bottom left, there are two silhouettes of people, one appearing to be looking at the other. The text "Thank You!" is centered in the middle of the image in a white, bold, sans-serif font.

Thank You!