



第7章 文件与数据格式化

授课老师：刘国旭

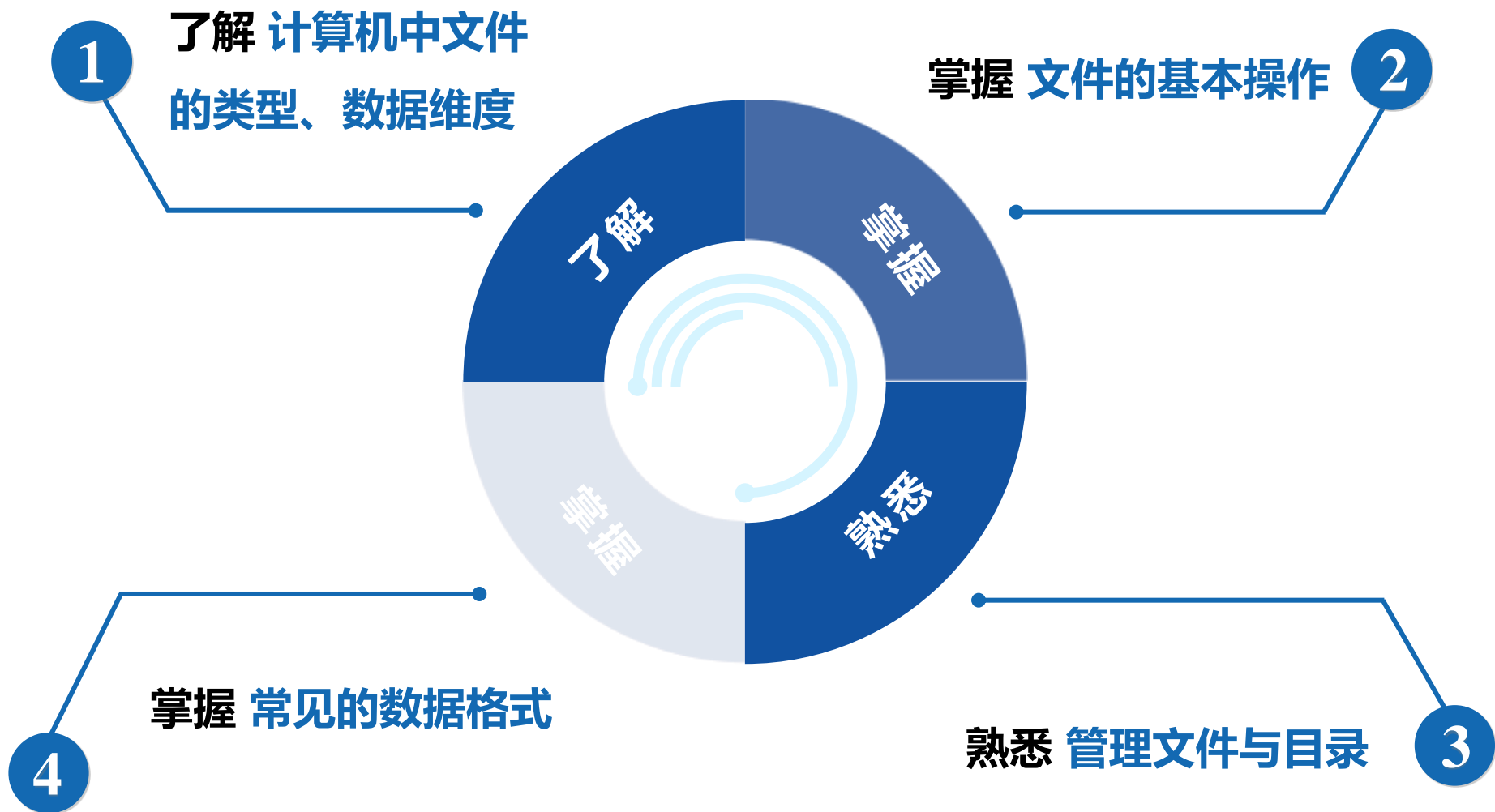
潍坊科技学院



- 文件概述
- 文件的基本操作
- 文件与目录管理
- 数据维度
- 数据格式化



学习目标





目录页



潍坊科技学院
Weifang University of Science and Technology



- 7.1** 文件概述
- 7.2** 文件的基础操作
- 7.3** 文件与目录管理
- 7.4** 实训案例
- 7.5** 数据维度与数据格式化



目录页



潍坊科技学院
Weifang University of Science and Technology



7.1 文件概述

7.2 文件的基础操作

7.3 文件与目录管理

7.4 实训案例

7.5 数据维度与数据格式化



7.1 文件概述



文件标识

- 文件标识的**意义**：找到计算机中**唯一确定的文件**。
- 文件标识的**组成**：文件路径、文件名主干、文件扩展名。

D:\itcast\chapter10\example.dat

路径

文件名主干 扩展名

- 操作系统以文件为单位对数据进行管理。



7.1 文件概述



文件类型

根据数据的逻辑存储结构，人们将计算机中的文件分为**文本文件**和**二进制文件**。

- 文本文件：专门存储**文本字符**数据。
- 二进制文件：不能直接使用文字处理程序正常读写，必须先了解其结构和序列化规则，再设计正确的反序列化规则，才能正确获取文件信息。
- 二进制文件和文本文件这两种类型的划分**基于数据逻辑存储结构**而非物理存储结构，计算机中的数据在物理层面都以二进制形式存储。



多学一招：标准文件



标准文件

Python的sys模块中定义了3个标准文件，分别为：

- **stdin**（标准输入文件）。标准输入文件对应输入设备，如键盘。
- **stdout**（标准输出文件）。
- **stderr**（标准错误文件）。标准输出文件和标准错误文件对应输出设备，如显示器。

在解释器中导入sys模块后，便可对标准文件进行操作。



多学一招：标准文件



潍坊科技学院
Weifang University of Science and Technology

标准文件

```
import sys  
file = sys.stdout  
file.write("hello")
```

示例



目录页



潍坊科技学院
Weifang University of Science and Technology



7.1 文件概述

7.2 文件的基础操作

7.3 文件与目录管理

7.4 实训案例

7.5 数据维度与数据格式化



7.2 文件的基本操作



文件的**打开**、**关闭**与**读写**是文件的基础操作，任何更复杂的文件操作都离不开这些操作。





7.2.1 文件的打开与关闭



1. 打开文件

内置函数open()用于打开文件，该方法的声明如下：

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

【参数】

- **file**: 文件的路径。
- **mode**: 设置文件的打开模式，取值有r、w、a。
- **buffering**: 设置访问文件的缓冲方式。取值为0或1。
- r: 以只读方式打开文件 (mode参数的默认值)。
- w: 以只写方式打开文件。
- a: 以追加方式打开文件。
- b: 以二进制形式打开文件。
- +: 以更新的方式打开文件 (可读可写)



7.2.1 文件的打开与关闭



1. 打开文件

打开模式	名称	描述
r/rb	只读模式	以只读的形式打开文本文件/二进制文件，若文件不存在或无法找到，文件打开失败
w/wb	只写模式	以只写的形式打开文本文件/二进制文件，若文件已存在，则重写文件，否则创建新文件
a/ab	追加模式	以只写的形式打开文本文件/二进制文件，只允许在该文件末尾追加数据，若文件不存在，则创建新文件
r+/rb+	读取（更新）模式	以读/写的形式打开文本文件/二进制文件，若文件不存在，文件打开失败
w+/wb+	写入（更新）模式	以读/写的形式打开文本文件/二进制文件，若文件已存在，则重写文件
a+/ab+	追加（更新）模式	以读/写的形式打开文本/二进制文件，只允许在文件末尾添加数据，若文件不存在，则创建新文件



7.2.1 文件的打开与关闭



1. 打开文件

内置函数`open()`用于打开文件，该方法的声明如下：

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

【返回值】

- 若`open()`函数调用成功，返回一个文件对象。

```
file1 = open('E:\\a.txt')           # 以只读方式打开E盘的文本文件a.txt
file2 = open('b.txt', 'w')         # 以只写方式打开当前目录的文本文件b.txt
file3 = open('c.txt', 'w+')        # 以读/写方式打开文本文件c.txt
file4 = open('d.txt', 'wb+')       # 以读/写方式打开二进制文件d.txt
```

示例



7.2.1 文件的打开与关闭



1. 打开文件

- 若待打开的文件不存在，**文件打开失败**，程序会**抛出异常**，并打印错误信息。

示例

```
FileNotFoundError          Traceback (most recent call last)
<ipython-input-5-23b0bb5a2ffc> in <module>
----> 1 file1 = open("b.txt")

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



7.2.1 文件的打开与关闭



2. 关闭文件

Python可通过`close()`方法关闭文件，也可以使用`with`语句实现文件的自动关闭。

(1) `close()`方法

- `close()`方法是文件对象的内置方法。

```
file.close()
```

示例



7.2.1 文件的打开与关闭



2.关闭文件

(2) with语句

- with语句可**预定义清理**操作，以实现文件的自动关闭。

```
with open('a.txt') as f:  
    pass
```

示例



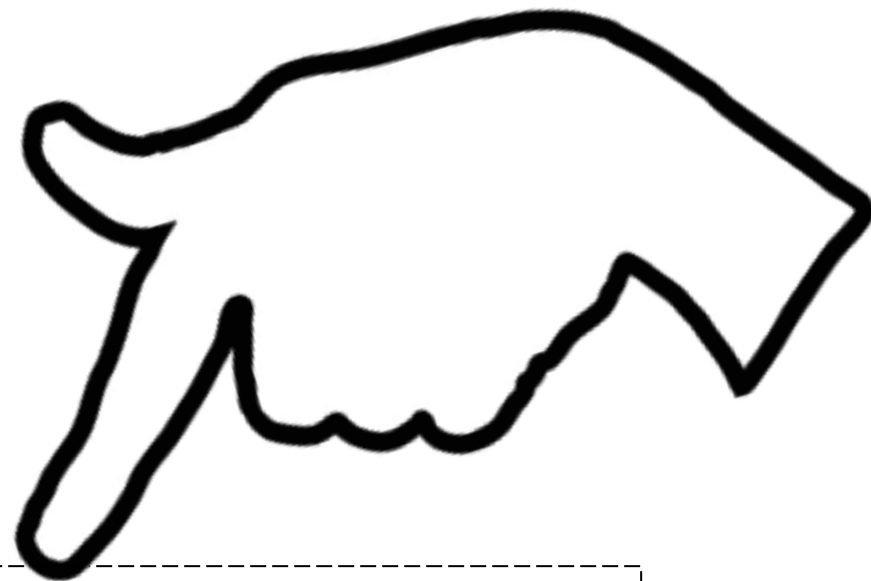
思考：为什么要及时关闭文件？



思考：
为什么要及时关闭文件？



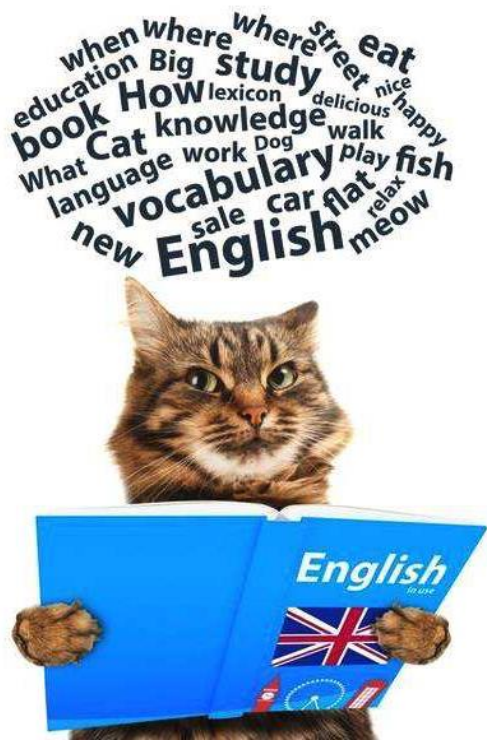
思考：为什么要及时关闭文件？



- ✓ 计算机中可打开的**文件数量**是有限
- ✓ 打开的文件**占用系统资源**
- ✓ 若程序因异常关闭，可能产生**数据丢失**



7.2.2 文件的读写



Python提供了一系列读写文件的方法，包括读取文件的`read()`、`readline()`、`readlines()`方法和写文件的`write()`、`writelines()`方法，下面结合这些方法分别介绍如何读写文件。



7.2.2 文件的读写



1. 读取文件——read()方法

read()方法可以从指定文件中**读取指定字节的数据**，其语法格式如下：

```
read(n=-1)
```

```
with open('file.txt', mode='r') as f:
```

```
    print(f.read(2))
```

```
    print(f.read())
```

示例

```
# 读取两个字节的数据
```

```
# 读取剩余的全部数据
```



7.2.2 文件的读写



1. 读取文件——readline()方法

readline()方法可以从指定文件中**读取一行数据**，其语法格式如下：

readline()

```
with open('file.txt', mode='r', encoding='utf-8') as f:
```

```
    print(f.readline())
```

```
    print(f.readline())
```

示例



7.2.2 文件的读写



1. 读取文件——readline()方法

readlines()方法可以一次读取文件中的**所有数据**，若读取成功，该方法会返回一个列表，文件中的每一行对应列表中的一个元素。语法格式如下：

`readlines(hint=-1)`

- **hint**: 单位为字节，用于控制要读取的行数如果行中数据的总大小超出了hint字节，readlines()不会再读取更多的行。

```
with open('file.txt', mode='r', encoding='utf-8') as f:  
    print(f.readlines())
```

示例

使用readlines()方法读取数据



7.2.2 文件的读写



1. 读取文件

- `read()` (参数缺省时) 和 `readlines()` 方法都可一次读取文件中的所有数据, 但因为计算机的内存是有限的, 若文件较大, `read()` 和 `readlines()` 的一次读取便会耗尽系统内存, 所以这两种操作都不够安全。
- 为了保证 **读取安全**, 通常多次调用 `read()` 方法, **每次读取 `size` 字节的数据**。



7.2.2 文件的读写



2. 写文件——write()方法

write()方法可以将指定字符串写入文件，其语法格式如下：

```
write(data)
```

以上格式中的参数data表示要写入文件的数据，若数据写入成功，write()方法会返回本次写入文件的数据的字节数。

```
string = "Here we are all, by day; by night."           # 字符串
with open('write_file.txt', mode='w', encoding='utf-8') as f:
    size = f.write(string)                               # 写入字符串
    print(size)                                         # 打印字节数
```

示例



7.2.2 文件的读写



2.写文件——writelines()方法

writelines()方法用于将行列表写入文件，其语法格式如下：

`writelines(lines)`

- 以上格式中的参数lines表示要写入文件中的数据，该参数可以是一个字符串或者字符串列表。
- 若写入文件的数据在文件中需要换行，需要显式指定换行符。

```
string = "Here we are all, by day;\nby night we're hurl'd By dreams,  
each one into a several world."  
with open('write_file.txt', mode='w', encoding='utf-8') as f:  
    f.writelines(string)
```

示例



多学一招：字符与编码



文本文件支持多种**编码方式**，不同编码方式下字符与字节的对应关系不同，常见的编码方式以及字符与字节的对应关系如表所示。

编码方式	语言	字符数	字节数
ASCII	中文	1	2
	英文	1	1
UTF-8	中文	1	3
	英文	1	1
Unicode	中文	1	2
	英文	1	2
GBK	中文	1	2
	英文	1	1



7.2.3 文件的定位读写



7.2.2节使用read()方法读取了文件file.txt，结合代码与程序运行结果进行分析，可以发现read()方法**第1次**读取了2个字符，**第2次**从第3个字符“e”开始**读取了**剩余字符。



7.2.3 文件的定位读写



- 在文件的一次打开与关闭之间进行的读写操作是连续的，程序总是从上次读写的位置继续向下进行读写操作。
- 每个文件对象都有一个称为“文件读写位置”的属性，该属性会记录当前读写的位置。
- 文件读写位置默认为0，即在文件首部。





7.2.3 文件的定位读写



Python提供了一些获取与修改文件读写位置的方法，以实现文件的定位读写。

- **tell()**: 获取文件当前的读写位置。
- **seek()**: 控制文件的读写位置。





7.2.3 文件的定位读写



1.tell()方法

tell()方法用于获取文件**当前的读写位置**，以操作文件file.txt为例，tell()的用法如下：

```
with open('file.txt') as f:
```

```
    print(f.tell())
```

```
    print(f.read(5))
```

```
    print(f.tell())
```

```
# 获取文件读写位置
```

```
# 利用read()方法移动文件读写位置
```

```
# 再次获取文件读写位置
```

示例



7.2.3 文件的定位读写




2.seek()方法

Python提供了seek()方法，使用该方法可**控制文件的读写位置**，实现文件的随机读写。seek()方法的语法格式如下：

`seek(offset, from)`

- offset: 表示偏移量，即读写位置需要移动的字节数。
- from: 用于指定文件的读写位置，该参数的取值为0、1、2。

seek()方法调用成功后会返回当前读写位置。

- 
- 0: 表示文件开头。
 - 1: 表示使用当前读写位置。
 - 2: 表示文件末尾。



7.2.3 文件的定位读写



2.seek()方法

```
with open('file.txt') as f:
```

```
    print(f.tell())
```

```
    print(f.read(5))
```

```
    print(f.tell())
```

```
# 获取文件读写位置
```

```
# 利用read()方法移动文件读写位置
```

```
# 再次获取文件读写位置
```

示例



7.2.3 文件的定位读写



2.seek()方法

在Python3中，若打开的是文本文件，那么seek()方法只允许相对于文件开头移动文件位置，若在**参数from值为1、2**的情况下对文本文件**进行位移操作**，将会产生错误。

```
with open('file.txt') as f:
```

```
    f.seek(5,0)
```

字节

```
    f.seek(3,1)
```

示例

相对文件开头移动5

```
-----  
UnsupportedOperation                                Traceback (most recent call last)  
<ipython-input-1-ab11f67330be> in <module>  
      1 with open('file.txt') as f:  
      2     f.seek(5,0)  
----> 3     f.seek(3,1)  
  
UnsupportedOperation: can't do nonzero cur-relative seeks
```

相对文件开头移动5字节



7.2.3 文件的定位读写



2.seek()方法

若要相对当前读写位置或文件末尾进行位移操作，需以**二进制形式**打开文件。

```
with open('file.txt','rb') as f:
```

```
    f.seek(5,0)
```

```
    f.seek(3,1)
```

示例



目录页



潍坊科技学院
Weifang University of Science and Technology



7.1 文件概述

7.2 文件的基础操作

7.3 文件与目录管理

7.4 实训案例

7.5 数据维度与数据格式化



7.3 文件与目录管理



对于用户而言，**文件和目录**以不同的形式展现，但对计算机而言，目录是文件属性信息集合，它本质上也是一种文件。



7.3 文件与目录管理



os模块中定义了与文件操作相关的函数，利用这些函数可以实现删除文件、文件重命名、创建/删除目录、获取当前目录、更改默认目录与获取目录列表等操作。



7.3 文件与目录管理



管理文件与目录

- **删除**文件——`os.remove(文件名)`
- 文件**重命名**——`os.rename(原文件名,新文件名)`
- **创建/删除**目录——`os.mkdir(目录名)/os.rmdir(目录名)`
- 获取**当前目录**——`os.getcwd()`
- 更改**默认目录**——`os.chdir(路径名)`
- 获取**目录列表**——`os.listdir(目录/路径)`



目录页



潍坊科技学院
Weifang University of Science and Technology



7.1 文件概述

7.2 文件的基础操作

7.3 文件与目录管理

7.4 实训案例

7.5 数据维度与数据格式化



7.4.1 信息安全策略——文件备份



当下是信息时代，信息在当今社会占据的地位不言而喻，信息安全更是当前人类重视的问题之一。人类考虑从传输和存储两方面保障信息的安全，备份是在存储工作中保障信息安全的有效方式。本案例要求编写程序，实现一个具有**备份文件与文件夹**功能的备份工具。



7.4.2 用户账户管理



某些网站要求访问者在访问网站内容之前必须先进行登录，若用户没有该网站的账号，则需要先进行注册。用户注册完账号后，网站的服务器会保存账号信息，以便用户下次访问网站时网站可根据保存的信息验证用户的身份。为保障账户安全，用户可时常修改密码；若后续用户不再使用网站，可以选择注销账户。

本案例要求实现包含**用户注册、登录、修改密码和注销功能**的用户账户管理程序（要求程序使用文件存储用户的账户信息）。



目录页



潍坊科技学院
Weifang University of Science and Technology



7.1 文件概述

7.2 文件的基础操作

7.3 文件与目录管理

7.4 实训案例

7.5 数据维度与数据格式化

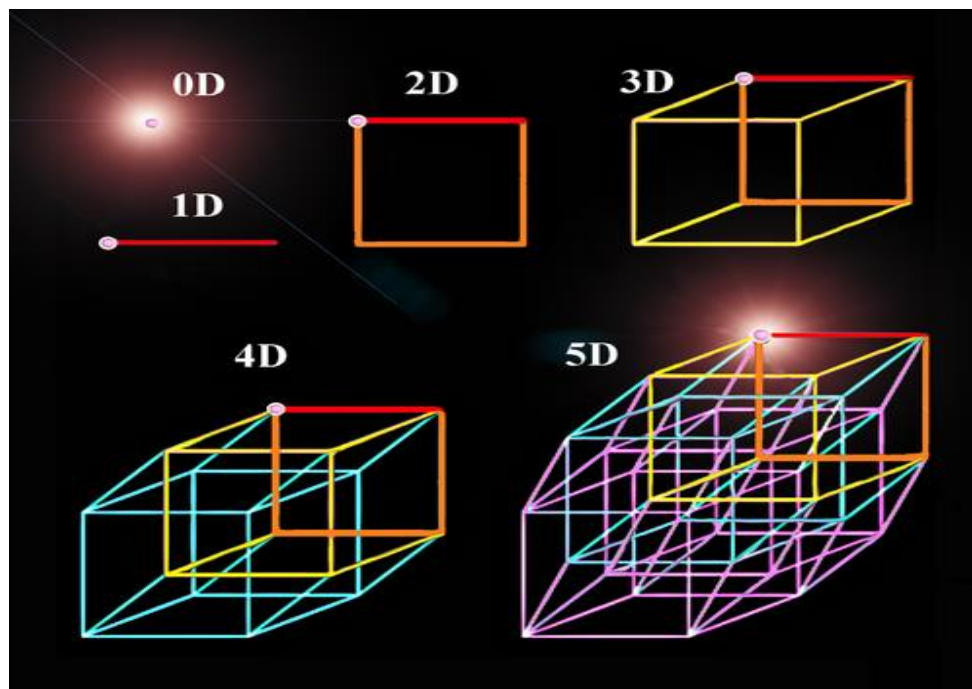


7.5 数据维度与数据格式化



维度

从广义上讲，维度是与事物“有联系”的概念的数量，根据“有联系”的概念的数量，事物可分为不同维度。





7.5.1 基于维度的数据分类



基于维度的数据分类

根据组织数据时与数据有联系的参数的数量，数据可分为**一维数据**、**二维数据**和**多维数据**。

一维数据

具有对等关系的一组线性数据，如：

- 一维列表
- 一维元组
- 集合

二维数据

二维数据关联参数的数量为2，如：

- 矩阵
- 二维数组
- 二维列表
- 二维元组

多维数据

利用键值对等简单的二元关系展示数据间的复杂结构，如：

- 字典



7.5.1 基于维度的数据分类



示例

一维数据

成都,杭州,重庆,武汉,苏州,西安,天津,南京,郑州,长沙,沈阳,青岛,宁波,东莞,无锡

二维数据

姓名	语文	数学	英语	理综
刘婧	124	137	145	260
张华	116	143	139	263
邢昭林	120	130	148	255
鞠依依	115	145	131	240
黄丽萍	123	108	121	235
赵越	132	100	112	210



7.5.1 基于维度的数据分类



示例

多维数据

“高三一班考试成绩”:[

```
{“姓名”:“刘婧”,  
“语文”:“124”,  
“数学”:“137”,  
“英语”:“145”,  
“理综”:“260”};  
{“姓名”:“张华”,  
“语文”:“116”,  
“数学”:“143”,  
“英语”:“139”,  
“理综”:“263”};  
.....
```

]



7.5.2 一二维数据的存储与读写



一二维数据的存储与读写

1. 数据存储

一维数据呈线性排列，一般用特殊字符分隔，例如：

- 使用空格分隔：成都 杭州 重庆 武汉 苏州 西安 天津
- 使用逗号分隔：成都,杭州,重庆,武汉,苏州,西安,天津
- 使用&分隔：成都&杭州&重庆&武汉&苏州&西安&天津

一维数据的存储需要**注意**以下几点：

- 同一文件或同组文件一般使用同一分隔符分隔。
- 分隔数据的分隔符不应出现在数据中。
- 分隔符为英文半角符号，一般不使用中文符号作为分隔符。



7.5.2 一二维数据的存储与读写



一二维数据的存储与读写

1. 数据存储

- 二维数据可视为多条一维数据的集合，当二维数据只有一个元素时，这个二维数据就是一维数据。
- CSV (Comma-Separated Values, 逗号分隔值) 是国际上通用的一二维数据存储格式。

CSV格式规范:

- 以纯文本形式存储表格数据
- 文件的每一行对应表格中的一条数据记录
- 每条记录由一个或多个字段组成
- 字段之间使用逗号 (英文、半角) 分隔



7.5.2 一二维数据的存储与读写



一二维数据的存储与读写

1. 数据存储

CSV也称字符分隔值，具体示例如下：

姓名,语文,数学,英语,理综

刘婧,124,137,145,260

张华,116,143,139,263

邢昭林,120,130,148,255

鞠依依,115,145,131,240

黄丽萍,123,108,121,235

赵越,132,100,112,210



7.5.2 一二维数据的存储与读写



一二维数据的存储与读写

2.数据读取

- Windows平台中CSV文件的后缀名为.csv，可通过Office Excel或记事本打开
- Python在程序中读取.csv文件后会以**二维列表形式**存储其中内容

```
csv_file = open('score.csv')
lines = []
for line in csv_file:
    line = line.replace('\n','')
    lines.append(line.split(','))
print(lines)
csv_file.close()
```

示例



7.5.2 一二维数据的存储与读写



一二维数据的存储与读写

3.数据写入

将一、二维数据写入文件中，即按照数据的组织形式，在文件中**添加新的数据**。

```
...  
for line in lines:  
    print(line)  
    file_new.write(','.join(line)+'\n')  
csv_file.close()  
file_new.close()
```

示例



7.5.3 多维数据的格式化



多维数据的格式化

为了直观地表示多维数据，也为了便于组织和操作，三维及以上的多维数据统一采用键值对的形式进行格式化。

网络平台上传递的数据大多是高维数据，**JSON**是网络中常见的高维数据格式。JSON格式的数据遵循以下语法规则：



- 数据存储**键值对** (key:value) 中，例如“姓名”：“张华”。
- 数据的字段由**逗号分隔**，例如“姓名”：“张华”，“语文”：“116”。
- 一个**花括号**保存一个JSON对象，例如{“姓名”：“张华”，“语文”：“116”}。
- 一个**方括号**保存一个数组，例如[{"姓名”：“张华”，“语文”：“116”}]。



7.5.3 多维数据的格式化



示例:

“高三二班考试成绩”:[

```
{“姓名”:“陈诚”,  
“语文”:“124”,  
“数学”:“127”,  
“英语”:“145”,  
“理综”:“259”};  
{“姓名”:“黄思”,  
“语文”:“116”,  
“数学”:“143”,  
“英语”:“119”,  
“理综”:“273”};
```

.....

]



7.5.3 多维数据的格式化



示例： 将学生成绩以XML格式存储

<高三二班考试成绩>

<姓名>陈诚</姓名><语文>124</语文><数学>127<数学/><英语>145<英语/><理综>259<理综/>

<姓名>黄思</姓名><语文>116</语文><数学>143<数学/><英语>119<英语/><理综>273<理综/>

.....

</高三二班考试成绩>



7.5.3 多维数据的格式化



JSON模块——json

利用json模块的dumps()函数和loads()函数可以实现Python对象和JSON数据之间的转换，这两个函数的具体功能如表所示。

函数	功能
dumps()	对Python对象进行转码，将其转化为JSON字符串
loads()	将JSON字符串解析为Python对象



7.5.3 多维数据的格式化



Python对象与JSON数据转化时的类型对照表

Python对象	JSON数据
dict	object
list,tuple	array
str,unicode	string
int,long,float	number
True	true
False	false
None	null



7.5.3 多维数据的格式化



1.dumps()函数

示例：使用dumps()函数对Python对象进行转码。

```
>>> import json
>>> pyobj = [[1, 2, 3], 345, 23.12, 'qwe', {'key1': (1,2,3),
'key2': (2,3,4)}, True, False, None]
>>> jsonstr = json.dumps(pyobj)
>>> print(jsonstr)
>>> [[1, 2, 3], 345, 23.12, "qwe", {"key1": [1, 2, 3], "key2": [2, 3,
4]}, true, false, null]
```



7.5.3 多维数据的格式化



2.loads()函数

示例：使用loads()函数将JSON数据转换为符合Python语法要求的数据类型。

```
>>> import json
>>> jsonstr = [[1, 2, 3], 345, 23.12, "qwe", {"key1": [1, 2, 3], "key2": [2, 3, 4]}, true, false, null]
>>> pydata = json.loads(jsonstr)
>>> print(pydata)
[[1, 2, 3], 345, 23.12, 'qwe', {'key1': [1, 2, 3], 'key2': [2, 3, 4]}, True, False, None]
```



7.6 本章小结



本章主要介绍了**文件与数据格式化**相关的知识，包括计算机中文件的**定义**、文件的基本**操作**、文件与**目录管理**、数据**维度**与数据格式化。通过本章的学习，希望读者能了解计算机中文件的意义、熟练地读取和管理文件，并掌握常见的数据组织形式。