楼+之Python实战第10期 (/courses/1190)

文件处理

文件处理概述

文件处理是项目开发中使数据持久化和获取配置的主要方式。我们在平时的工作中接触到各种文件类型，比如 word、视频、txt文本等。Python 中提供了很方便读写文件的函数，可以很方便的处理各种类型的文件。

在本实验中，我们将学习如何使用 Python 进行下列的文件处理：

1. 输入与输出
2. 打开与关闭文件
3. 读取文件内容
4. 写入文件
5. pickle 和 JSON 序列化
6. CSV 文件的读写

知识点

- 输入与输出
- 打开与关闭文件
- 读取与写入文件
- pickle 序列化
- JSON 序列化
- CSV 文件读写方法

输入与输出

在计算机程序中常常会听人提到一个叫做 `I/O` 的词，其实它是指 `Input/Output` ，也就是输入和输出。通常情况下输入设备包括鼠标、键盘、摄像头、麦克风等等，而输出设备就是显示屏、扬声器、耳机等等。在这里要讨论的是最常见的输入和输出设备，也就是键盘输入和显示屏输出。

在 Python 3 中，一般从终端获取用户的输入使用的是 `input()` 函数，括号中可以写入提示信息，用户的输入都是字符串类型，这个函数主要用于动态获取用户的输入：

```
>>> a = input("please input your name:")  # 括号中写入提示信息
please input your name:jack
>>> a
'jack'  # 用户的输入都是字符串类型
>>> b = input("please input a num:")
please input a num:3
>>> b
'3'
>>> int(b)   # 用户输入的数字是一个字符串类型，通过 int() 函数可以将其转变为整型
3
>>>
```

而 Python 中最常见的输出就是 `print()` 函数，这个函数用于在显示屏中进行打印。

```
>>> print("hello")   # 字符串
hello
>>> print(1)   # 整型
1
>>> print(1+2)   # 表达式
3
>>> print("hello","shiyanlou")   # 两个字符串之间使用逗号隔开可以形成字符串拼接，逗号的位置会用一个空格表示
hello shiyanlou
>>> a = 1
>>> a
1
>>> str(a)   # srt() 函数可以将一个整型转换为字符串类型
'1'
>>>
```

有的时候打印的内容中一小部分数据会经常变动，这个时候就可以使用到格式化输出。格式化输出常用的有两种方式，一种是 `%+字母` 的形式，另一种是使用 `format()` 函数。

`%+字母` 表示这个数据在字符串中的显示格式，常用的有： %d整数 %s字符串 %f浮点数（%.2f四舍五入显示两位，%.3f四舍五入显示三位）

```
>>> print("%s is %d years old"%('Jack',8))
Jack is 8 years old
>>> print("Apple's price is %.2f"%3.278)
Apple's price is 3.28
>>>
```

更推荐的是使用 `format()` 函数进行格式化。在字符串中使用大括号 `{}` 表示一个数据，大括号中的 0 表示 `format()` 函数中的第一个数据。大括号中的 :.2f 表示四舍五入显示两位。

楼+之Python实战第10期 (/courses/1190)

```
>>> print("{0} is {1} years old".format('Jack',8))
Jack is 8 years old
>>> print("Apple's price is {:.2f}".format(3.278))
Apple's price is 3.28
>>>
```

练习题：输入数据并求和

本节将考察之前学习的输入与输出的知识点，请按照题目要求完成，点击 下一步 系统将自动检测完成结果。

在 /home/shiyanlou 目录下创建代码文件 sumtest.py ：

```
cd /home/shiyanlou/
touch sumtest.py
```

在代码中需要满足以下要求：

- 使用 input() 获取用户输入的三个整数值
- 对这三个整数值使用加法进行求和并打印输出
- 打印时需要使用 format() 函数

程序运行示例：

```
shiyanlou:~/ $ python3 sumtest.py
please enter the first num:1
please enter the second num:2
please enter the third num:3
a is 1,b is 2,c is 3,a+b+c=6
```

程序完成后，点击 下一步 ，系统将自动检测完成结果。

打开与关闭

我们使用 open() 函数打开文件。这个函数将返回一个文件对象，我们对文件的读写都将使用这个对象。

open() 函数需要两个参数，第一个参数是文件路径或文件名，第二个是文件的打开模式。模式通常是下面这样的：

- "r" ，以只读模式打开，你只能读取文件但不能编辑/删除文件的任何内容
- "w" ，以写入模式打开，如果文件存在将会删除里面的所有内容，然后打开这个文件进行写入
- "a" ，以追加模式打开，写入到文件中的任何数据将自动添加到末尾

- "b" ，以二进制的方式打开

楼+之Python实战第10期 (/courses/1190)

默认的模式为只读模式，也就是说如果你不提供任何模式， open() 函数将会以只读模式打开文件。

我们将实验打开一个文件，进入 Python 交互模式打开这个文件：

```
>>> file = open('/etc/protocols')
>>> type(file)
<class '_io.TextIOWrapper'>
>>> file
<_io.TextIOWrapper name='/etc/protocols' mode='r' encoding='UTF-8'>
```

从上面的信息，可以看到打开的文件 /etc/protocols ，打开的模式为只读模式 r ，编码方式为 UTF-8。

当程序执行结束后，我们需要关闭文件，如果不关闭会持续占用一些内存和操作系统资源，不关闭的文件也有可能造成数据丢失。我们使用方法 close() 完成这个操作，重复关闭不会有任何影响：

```
>>> file.close()
>>> file
<_io.TextIOWrapper name='/etc/protocols' mode='r' encoding='UTF-8'>
>>> file.close()
```

在实际情况中，我们应该尝试使用 with 语句处理文件对象，它会在文件用完后自动关闭，就算发生异常也没关系。它是 try-finally 块的简写：

```
>>> with open('/etc/protocols') as file:
...     count = 0
...     for line in file:
...         count += 1
...     print(count)
...
64
```

这个程序中我们将打印文件的行数，可以看到代码块中并没有使用 close，但当代码执行到 with 代码块之外时，文件会被自动关闭。

文件打开关闭操作视频：

0:00 / 4:08

读取文件内容

此处我们先用 `echo` 命令创建一个文件，这个文件的内容包括两行：

```
I love python
lou+ at shiyanlou.com
```

创建的文件路径为 `/home/shiyanlou/test.txt` ，创建的命令：

```
$ echo 'I love Python' > /home/shiyanlou/test.txt
$ echo 'lou+ at shiyanlou.com' >> /home/shiyanlou/test.txt
$ cat test.txt
I love Python
lou+ at shiyanlou.com
```

使用 `read()` 可以一次性读取整个文件的内容到字符串：

```
>>> filename = '/home/shiyanlou/test.txt'
>>> file = open(filename)
>>> file.read()
'I love Python\nlou+ at shiyanlou.com\n'
>>> file.close()
```

当然也可以使用 with，避免忘记了 close：

```
>>> filename = '/home/shiyanlou/test.txt'
>>> with open(filename) as file:
...     file.read()
...
```

项目开发中，我们需要谨慎使用 `read()` 读取整个文件，因为有可能你的系统内存并不足够存储整个文件的内容。当 `read()` 执行后，再次执行将不会有任何内容的输出。

在处理文本文件的时候，我们通常会采用逐行处理， `readline()` 就是用来每次读取文件的一行， `readlines()` 可以读取所有行，但不同于 `read()` ，这个函数返回的是一个列表，列表中每个元素都是对应文本文件中一行内容的字符串：

```
>>> filename = '/home/shiyanlou/test.txt'
>>> file = open(filename)
>>> file.readline()
'I love Python\n'
>>> file.readline()
'lou+ at shiyanlou.com\n'
>>> file.close()
```

由于使用了两次 `readline()` 再次使用 `readlines()` 将不会有任何输出，所以我们需要再次重新打开文件：

```
>>> file = open(filename)
>>> file.readlines()
['I love Python\n', 'lou+ at shiyanlou.com\n']
>>> file.close()
```

你可以 for 循环遍历文件对象来读取文件中的每一行：

```
>>> file = open(filename)
>>> for x in file:
...     print(x, end = '')
...
I love Python
lou+ at shiyanlou.com
>>> file.close()
```

实现文件内容统计程序

让我们写一个程序，这个程序接受用户输入的字符串作为将要读取的文件的文件名，并且在屏幕上打印文件行数和文件内容，程序写入 `/home/shiyanlou/test.py` ：

```
#!/usr/bin/env python3

filename = input("Enter the file name: ")

with open(filename) as file:
    count = 0
    for line in file:
        count += 1
        print(line)
    print('Lines:', count)
```

运行程序：

```
$ cd /home/shiyanlou
$ chmod +x test.py
$ ./test.py
Enter the file name: /home/shiyanlou/test.txt
I love Python

lou+ at shiyanlou.com

Lines: 2
```

读取文件操作视频：

楼+之Python实战第10期 (/courses/1190)

0:00 / 5:29

写入文件

最常用的写入文件的方法是 `write()` ，让我们通过 `write()` 方法打开一个文件然后我们随便写入一些文本。

```
>>> filename = '/home/shiyanlou/test.txt'
>>> with open(filename, 'w') as file:
...     file.write('testline1')
...     file.write('testline2')
...
9
9
```

这个程序中，我们将文件以 'w' 模式打开，然后写入两段内容，执行过程中输出的两个 9 分别表示两次写入的字符数量。

查看刚刚写入完成的文件内容，是否已经变化，仍然使用 `readlines()` 查看：

现在读取我们刚刚创建的文件。

```
>>> with open(filename, 'r') as file:
...     print(file.readlines())
...
['testline1testline2']
```

这时候发现文件中原来的内容已经被完全覆盖了，并且写入的内容占了一行（因为没有写入换行符的原因）。

如果我们想向文件中增加内容如何操作呢？可以使用 `'a'` 追加模式打开文件：

```
>>> filename = '/home/shiyanlou/test.txt'
>>> with open(filename, 'a') as file:
...     file.write('testline3')
...     file.write('testline4')
...
9
9
```

再次读取，可以看到新增加的字符串附加到了原来的内容后面：

```
>>> with open('/home/shiyanlou/1') as file:
...     print(file.readlines())
...
['testline1testline2testline3testline4']
```

楼+之Python实战第10期(/courses/1190)

写入文件操作视频：

0:00 / 3:10

拷贝文件示例

在这个例子里我们拷贝给定的文本文件到另一个给定的文本文件，实现类似 Linux 上的 cp 命令，这个程序可以接受两个参数，第一个参数为要拷贝的文件，第二个参数为拷贝后的新文件的路径，代码如下，请理解每一行的代码：

```
#!/usr/bin/env python3

import sys

def copy_file(src, dst):
    with open(src, 'r') as src_file:
        with open(dst, 'w') as dst_file:
            dst_file.write(src_file.read())

if __name__ == '__main__':
    if len(sys.argv) == 3:
        copy_file(sys.argv[1], sys.argv[2])
    else:
        print("Parameter Error")
        print(sys.argv[0], "srcfile dstfile")
        sys.exit(-1)
    sys.exit(0)
```

拷贝文件操作视频：

0:00 / 3:33

你可以看到我们在这里使用了一个新模块 `sys`。`sys.argv` 包含所有命令行参数。

楼+之Python实战第10期 (/courses/1190)

`sys.argv` 的第一个值 `sys.argv[0]` 是程序自身的名字，下面这个程序打印命令行参数：

```
#!/usr/bin/env python3

import sys
print("Program:", sys.argv[0])
print("Parameters:")
for i, x  in enumerate(sys.argv):
    if (i == 0):
        continue
    print(i, x)
```

运行程序：

```
$ ./argvtest.py lou+ shiyanlou.com
Program: ./argvtest.py
Parameters:
1 lou+
2 shiyanlou.com
```

这里我们用到了一个新函数 `enumerate(iterableobject)`，在列表中循环时，索引位置和对应值可以使用它同时得到。这里在参数列表中使用 continue 去除了 `sys.argv[0]` 程序自身的名字。

- 拓展阅读：文件与 IO (https://python3-cookbook.readthedocs.io/zh_CN/latest/chapters/p05_files_and_io.html)

练习题：按要求读写文件

本节将考察基本的文件读写知识点，请按照题目要求完成，点击 `下一步` 系统将自动检测完成结果。

在 `/home/shiyanlou` 目录下有一个 `shiyanlou.txt` 文件，内容如下：

```
Shiyanlou
Louplus
I love python
Hello world
```

要求在 `/home/shiyanlou/copyfile.py` 文件中写入代码实现：

- 读取 `shiyanlou.txt` 文件中的内容，并将其写入 `/home/shiyanlou/shiyanlou_copy.txt` 文件中
- `copyfile.py` 文件代码中必须使用 `readlines()` 方法
- `shiyanlou_copy.txt` 文件中的内容格式必须如下所示（即每一行前面加上对应的行数）：

楼+之Python实战第10期 (/courses/1190)

```
1 Shiyanlou
2 Louplus
3 I love python
4 Hello world
```

程序完成后，点击 下一步 ，系统将自动检测完成结果。

pickle 和 JSON 序列化

如果我们想用文本文件保存一个 Python 对象怎么操作？

这里就涉及到序列化的问题，序列化指的是将内存中的对象转化为可以存储的格式。Python 中最常用两种方式进行序列化：

1. `pickle` 模块
2. JSON 格式

pickle

我们首先通过一个实例将 Python 的一个字典存入到文件中并读取出来恢复成字典对象，这个过程中用的就是 pickle 模块：

```
>>> import pickle
>>> courses = { 1:'Linux', 2:'Vim', 3:'Git'}
>>> with open('./courses.data', 'wb') as file:
...     pickle.dump(courses, file)
...
>>> with open('./courses.data', 'rb') as file:
...     new_courses = pickle.load(file)
...
>>> new_courses
{1: 'Linux', 2: 'Vim', 3: 'Git'}
>>> type(new_courses)
<class 'dict'>
>>>
```

注意写入和读取文件都需要使用 `b` 二进制模式。

最终我们写入文件并读取后仍然可以恢复到原来的字典对象。如果只是想将对象序列化成一个字节流，那可以使用 `pickle.dumps(obj)` 。

- 拓展阅读：序列化 Python 对象 (https://python3-cookbook.readthedocs.io/zh_CN/latest/c05/p21_serializing_python_objects.html?highlight=pickle)

JSON

楼+之Python实战第10期 (/courses/1190)

JSON(JavaScript Object Notation, JS 对象标记)是一种轻量级的数据交换格式。

JSON 格式在互联网应用开发中应用非常广泛，可以作为不同的服务组件之间进行数据传递的格式。在互联网应用提供的各种 API 接口返回值基本都是 JSON 格式。

Python 也提供了 json 模块支持 JSON 序列化。JSON 支持字典和列表两种数据类型的序列化，方法一样。下面以字典为例进行讲解。 `dumps` 和 `dump` 方法将数据写入文件：

```
>>> import json  # 引入模块
>>> courses = {1: 'Linux', 2: 'Git', 3: 'Vim'}  # 字典对象
>>> json.dumps(courses)  # dumps 方法将字典转换成 json 格式的字符串
'{"1": "Linux", "2": "Git", "3": "Vim"}'
>>> with open('courses.json', 'w') as f:
...     f.write(json.dumps(courses))  # json 格式的字符串可直接写入 json 文件
...
38
>>> with open('courses1.json', 'w') as f:
...     json.dump(courses, f)  # dump 方法可将字典进行序列化并存入文件
...
```

在终端查看 `courses.json` 和 `courses1.json` ，两个文件的内容一样。 `loads` 和 `load` 方法读取 json 文件：

```
>>> json_str = json.dumps(courses)
>>> json_str
'{"1": "Linux", "2": "Git", "3": "Vim"}'
>>> data1 = json.loads(json_str)  # loads 方法将 json 字符串反序列化为字典 / 列表
>>> data1
{'1': 'Linux', '2': 'Git', '3': 'Vim'}
>>> with open('courses.json', 'r') as f:  # 'r' 参数为默认参数，可省略不写
...     data2 = json.load(f)  # load 方法将 json 文件反序列化为字典 / 列表
...
>>> data2
{'1': 'Linux', '2': 'Git', '3': 'Vim'}
```

- 拓展阅读：读写 JSON 数据 (https://python3-cookbook.readthedocs.io/zh_CN/latest/c06/p02_read-write_json_data.html?highlight=json)

练习题：使用 JSON 序列化数据到文件中

使用前面学到的 JSON 序列化的方法将下面的列表对象保存到文件 `/tmp/jsontest.json` 中。

该数据为两个示例用户的部分学习数据，需要序列化到指定的文件中：

楼+之Python实战第10期 (/courses/1190)

```
    'user_id': 1000,
    'name': 'Shiyan',
    'pass': 10,
    'study_time': 50,
},
{
    'user_id': 2000,
    'name': 'Lou',
    'pass': 15,
    'study_time': 171,
}]
```

完成后点击 下一步 ，系统将自动检测完成结果。

CSV 文件读写

csv 即 Comma Separate Values（逗号分隔值）的缩写，顾名思义，文件内容是由逗号进行分隔的一列一列的数据，具有编辑方便、可视化效果优良的特点。 Python 的 csv 模块专门用于处理 CSV 文件，下面举例来说明。

首先，我们创建一个 test.csv 文件，将以下内容写入并保存：

```
Symbol,Price,Date,Time,Change,Volume
"AA",39.48,"6/11/2007","9:36am",-0.18,181800
"AIG",71.38,"6/11/2007","9:36am",-0.15,195500
"AXP",62.58,"6/11/2007","9:36am",-0.46,935000
"BA",98.31,"6/11/2007","9:36am",+0.12,104800
"C",53.08,"6/11/2007","9:36am",-0.25,360900
"CAT",78.29,"6/11/2007","9:36am",-0.23,225400
```

现在，我们来尝试使用 csv 模块读写文件。

- reader 方法读取数据：

```
# 读取文件，用 reader() 读取的结果是个迭代器，文件关闭后此迭代器无法读取数据
# 迭代器的概念会在下一节实验中讲到，这里只需记住命令的用法即可
>>> import csv   # 引入模块
>>> with open('test.csv') as f:
...     data = csv.reader(f)   # reader 方法读取文件
...
>>> from collections.abc import Iterator   # 引入 Iterator 类
>>> isinstance(data, Iterator)   # 判断 data 的数据类型是否为迭代器
True
>>> next(data)   # 正如前面的注释所说，csv.reader 方法的处理结果为迭代器，且文件关闭后无法读取数据
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

既然如此，就在文件关闭前对 `csv.reader` 的结果进行一些处理：

```
# 可以把读取到的数据转换成 tuple 或 list
# tuple 或 list 里每个元素都是列表，每个子列表里的元素都是字符串
>>> with open('test.csv') as f:
...     data = list(csv.reader(f))   # list 方法将迭代器进行迭代并将结果转换为列表数据
...
>>> type(data)   # data 的数据类型变成列表
<class 'list'>
>>> for i in data:   # 列表内每个元素也都是列表，对应 CSV 文件中的一行数据
...     print(i)
...
['Symbol', 'Price', 'Date', 'Time', 'Change', 'Volume']
['AA', '39.48', '6/11/2007', '9:36am', '-0.18', '181800']
['AIG', '71.38', '6/11/2007', '9:36am', '-0.15', '195500']
['AXP', '62.58', '6/11/2007', '9:36am', '-0.46', '935000']
['BA', '98.31', '6/11/2007', '9:36am', '+0.12', '104800']
['C', '53.08', '6/11/2007', '9:36am', '-0.25', '360900']
['CAT', '78.29', '6/11/2007', '9:36am', '-0.23', '225400']
```

- `writer` 方法写入数据：

```
# csv.writer(f) 是一个具有 writerow 和 writerows 方法的 _csv.writer 类的实例
# 如果只写入一行，则使用 writerow 方法
# 由于 data 是读取多行文件的结果，所以下面使用是 writerows 方法写入数据
>>> with open('test_w.csv', 'w') as f:
...     csv.writer(f).writerows(data)
...
```

此时按 `Ctrl + C` 退出交互解释器，回到终端。执行 `cat test_w.csv` 即可看到文件内容与 `test.csv` 相同。

在 Windows 系统中使用 Python 代码写入 csv 文件会出现空行，加个参数 `newline=''` 即可解决:

```
>>> with open('文件实战第10期', newline='') as f:
...     csv.writer(f).writerows(data)
...
```

楼+之Python实战第10期 (/courses/1190)

总结

本实验我们学习了文件的打开与读写，也了解了序列化及文件路径操作的基本用法。包括以下的知识点：

- 输入与输出
- 打开与关闭文件
- 读取文件内容
- 写入文件
- pickle 和 JSON 序列化
- CSV 文件的读写

这些内容在实际的项目开发中非常常用，文件的打开关闭与读写是每种编程语言都会涉及到的内容。互联网项目中，JSON 序列化是作为 Web Service API 提供接口连接不同服务的常用方法，尤其必须要理解为什么要序列化。

拓展阅读

关于实验 Python 读取和写入数据，你可以继续阅读：

- 《Python 官方文档 - 文件读写》(http://www.pythondoc.com/pythontutorial3/inputoutput.html#tut-files)
- 《用 Python 的输入输出功能读取和写入数据》(https://www.ibm.com/developerworks/cn/opensource/os-python8/index.html)

JSON 是存储和交换文本信息的语法，更多关于 JSON 的介绍，可以浏览：

- 《W3school：JSON 教程》 (http://www.w3school.com.cn/json/index.asp)

上一节：面向对象编程 (/courses/1190/labs/8523/document)

下一节：挑战：工资计算器读写数据文件 (/courses/1190/labs/8525/document)