

06 | Python “黑箱”：输入与输出

2019-05-22 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 11:06 大小 10.17M



你好，我是景霄。


世纪之交的论坛上曾有一句流行语：在互联网上，没人知道你是一条狗。互联网刚刚兴起时，一根网线链接到你家，信息通过这条高速线缆直达你的屏幕，你通过键盘飞速回应朋友的消息，信息再次通过网线飞入错综复杂的虚拟世界，再进入朋友家。抽象来看，一台台的电脑就是一个个黑箱，黑箱有了输入和输出，就拥有了图灵机运作的必要条件。

Python 程序也是一个黑箱：通过输入流将数据送达，通过输出流将处理后的数据送出，可能 Python 解释器后面藏了一个人，还是一个史莱哲林？No one cares。

好了废话不多说，今天我们就由浅及深讲讲 Python 的输入和输出。

输入输出基础


最简单直接的输入来自键盘操作，比如下面这个例子。

 复制代码

```
1 name = input('your name:')
2 gender = input('you are a boy?(y/n)')
3
4 ##### 输入 #####
5 your name:Jack
6 you are a boy?
7
8 welcome_str = 'Welcome to the matrix {prefix} {name}.'
9 welcome_dic = {
10     'prefix': 'Mr.' if gender == 'y' else 'Mrs',
11     'name': name
12 }
13
14 print('authorizing...')
15 print(welcome_str.format(**welcome_dic))
16
17 ##### 输出 #####
18 authorizing...
19 Welcome to the matrix Mr. Jack.
```

`input()` 函数暂停程序运行，同时等待键盘输入；直到回车被按下，函数的参数即为提示语，输入的类型永远是字符串型（`str`）。注意，初学者在这里很容易犯错，下面的例子我会讲到。`print()` 函数则接受字符串、数字、字典、列表甚至一些自定义类的输出。

我们再来看下面这个例子。

 复制代码

```
1 a = input()
2 1
3 b = input()
4 2
5
6 print('a + b = {}'.format(a + b))
7 ##### 输出 #####
8 a + b = 12
9 print('type of a is {}, type of b is {}'.format(type(a), type(b)))
10 ##### 输出 #####
11 type of a is <class 'str'>, type of b is <class 'str'>
```

```
12 print('a + b = {}'.format(int(a) + int(b)))
13 ##### 输出 #####
14 a + b = 3
```

这里注意，把 str 强制转换为 int 请用 int()，转为浮点数请用 float()。而在生产环境中使用强制转换时，请记得加上 try except（即错误和异常处理，专栏后面文章会讲到）。

Python 对 int 类型没有最大限制（相比之下，C++ 的 int 最大为 2147483647，超过这个数字会产生溢出），但是对 float 类型依然有精度限制。这些特点，除了在一些算法竞赛中要注意，在生产环境中也要时刻提防，避免因为对边界条件判断不清而造成 bug 甚至 Oday（危重安全漏洞）。


我们回望一下币圈。2018 年 4 月 23 日中午 11 点 30 分左右，BEC 代币智能合约被黑客攻击。黑客利用数据溢出的漏洞，攻击与美图合作的公司美链 BEC 的智能合约，成功地向两个地址转出了天量级别的 BEC 代币，导致市场上的海量 BEC 被抛售，该数字货币的价值也几近归零，给 BEC 市场交易带来了毁灭性的打击。

由此可见，虽然输入输出和类型处理事情简单，但我们一定要慎之又慎。毕竟相当比例的安全漏洞，都来自随意的 I/O 处理。

文件输入输出

命令行的输入输出，只是 Python 交互的最基本方式，适用一些简单小程序的交互。而生产级别的 Python 代码，大部分 I/O 则来自于文件、网络、其他进程的消息等等。

接下来，我们来详细分析一个文本文件读写。假设我们有一个文本文件 in.txt，内容如下：

 复制代码


```
1 I have a dream that my four little children will one day live in a nation where they wi
2
3 I have a dream that one day down in Alabama, with its vicious racists, . . . one day ri
4
5 I have a dream that one day every valley shall be exalted, every hill and mountain shal
6
7 This is our hope. . . With this faith we will be able to hew out of the mountain of des
8
9 And when this happens, and when we allow freedom ring, when we let it ring from every v:
```

好，让我们来做简单的 NLP（自然语言处理）任务。如果你对此不太了解也没有影响，我会带你一步步完成这个任务。

首先，我们要清楚 NLP 任务的基本步骤，也就是下面的四步：

1. 读取文件；
2. 去除所有标点符号和换行符，并把所有大写变成小写；
3. 合并相同的词，统计每个词出现的频率，并按照词频从大到小排序；
4. 将结果按行输出到文件 out.txt。

你可以自己先思考一下，用 Python 如何解决这个问题。这里，我也给出了我的代码，并附有详细的注释。我们一起来看下这段代码。

 复制代码

```
1 import re
2
3 # 你不用太关心这个函数
4 def parse(text):
5     # 使用正则表达式去除标点符号和换行符
6     text = re.sub(r'^\w ', ' ', text)
7
8     # 转为小写
9     text = text.lower()
10
11     # 生成所有单词的列表
12     word_list = text.split(' ')
13
14     # 去除空白单词
15     word_list = filter(None, word_list)
16
17     # 生成单词和词频的字典
18     word_cnt = {}
19     for word in word_list:
20         if word not in word_cnt:
21             word_cnt[word] = 0
22             word_cnt[word] += 1
23
24     # 按照词频排序
25     sorted_word_cnt = sorted(word_cnt.items(), key=lambda kv: kv[1], reverse=True)
26
27     return sorted_word_cnt
28
29 with open('in.txt', 'r') as fin:
```

```
30     text = fin.read()
31
32 word_and_freq = parse(text)
33
34 with open('out.txt', 'w') as fout:
35     for word, freq in word_and_freq:
36         fout.write('{} {} \n'.format(word, freq))
37
38 ##### 输出（省略较长的中间结果）#####
39
40 and 15
41 be 13
42 will 11
43 to 11
44 the 10
45 of 10
46 a 8
47 we 8
48 day 6
49
50 ...
51
52 old 1
53 negro 1
54 spiritual 1
55 thank 1
56 god 1
57 almighty 1
58 are 1
```

你不用太关心 `parse()` 函数的具体实现，你只需要知道，它做的事情是把输入的 `text` 字符串，转化为我们需要的排序后的词频统计。而 `sorted_word_cnt` 则是一个二元组的列表 (list of tuples) 。

首先我们需要先了解一下，计算机中文件访问的基础知识。事实上，计算机内核 (kernel) 对文件的处理相对比较复杂，涉及到内核模式、虚拟文件系统、锁和指针等一系列概念，这些内容我不会深入讲解，我只说一些基础但足够使用的知识。

我们先要用 `open()` 函数拿到文件的指针。其中，第一个参数指定文件位置（相对位置或者绝对位置）；第二个参数，如果是 `'r'` 表示读取，如果是 `'w'` 则表示写入，当然也可以用 `'rw'`，表示读写都要。`a` 则是一个不太常用（但也很有用）的参数，表示追加 (append)，这样打开的文件，如果需要写入，会从原始文件的最末尾开始写入。

这里我插一句，在 Facebook 的工作中，代码权限管理非常重要。如果你只需要读取文件，就不要请求写入权限。这样在某种程度上可以降低 bug 对整个系统带来的风险。

好，回到我们的话题。在拿到指针后，我们可以通过 `read()` 函数，来读取文件的全部内容。代码 `text = fin.read()`，即表示把文件所有内容读取到内存中，并赋值给变量 `text`。这么做自然也是有利有弊：

优点是方便，接下来我们可以很方便地调用 `parse` 函数进行分析；

缺点是如果文件过大，一次性读取可能造成内存崩溃。

这时，我们可以给 `read` 指定参数 `size`，用来表示读取的最大长度。还可以通过 `readline()` 函数，每次读取一行，这种做法常用于数据挖掘（Data Mining）中的数据清洗，在写一些小的程序时非常轻便。如果每行之间没有关联，这种做法也可以降低内存的压力。而 `write()` 函数，可以把参数中的字符串输出到文件中，也很容易理解。

这里我需要简单提一下 `with` 语句（后文会详细讲到）。`open()` 函数对应于 `close()` 函数，也就是说，如果你打开了文件，在完成读取任务后，就应该立刻关掉它。而如果你使用了 `with` 语句，就不需要显式调用 `close()`。在 `with` 的语境下任务执行完毕后，`close()` 函数会被自动调用，代码也简洁很多。

最后需要注意的是，所有 I/O 都应该进行错误处理。因为 I/O 操作可能会有各种各样的情况出现，而一个健壮（robust）的程序，需要能应对各种情况的发生，而不应该崩溃（故意设计的情况除外）。

JSON 序列化与实战

最后，我来讲一个和实际应用很贴近的知识点。

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，它的设计意图是把所有事情都用设计的字符串来表示，这样既方便在互联网上传递信息，也方便人进行阅读（相比一些 binary 的协议）。JSON 在当今互联网中应用非常广泛，也是每一个用 Python 程序员应当熟练掌握的技能点。

设想一个情景，你要向交易所购买一定数额的股票。那么，你需要提交股票代码、方向（买入 / 卖出）、订单类型（市价 / 限价）、价格（如果是限价单）、数量等一系列参数，而

这些数据里，有字符串，有整数，有浮点数，甚至还有布尔型变量，全部混在一起并不方便交易所解包。

那该怎么办呢？

其实，我们要讲的 JSON，正能解决这个场景。你可以把它简单地理解为两种黑箱：

第一种，输入这些杂七杂八的信息，比如 Python 字典，输出一个字符串；

第二种，输入这个字符串，可以输出包含原始信息的 Python 字典。

具体代码如下：

 复制代码

```
1 import json
2
3 params = {
4     'symbol': '123456',
5     'type': 'limit',
6     'price': 123.4,
7     'amount': 23
8 }
9
10 params_str = json.dumps(params)
11
12 print('after json serialization')
13 print('type of params_str = {}, params_str = {}'.format(type(params_str), params_str))
14
15 original_params = json.loads(params_str)
16
17 print('after json deserialization')
18 print('type of original_params = {}, original_params = {}'.format(type(original_params), original_params))
19
20 ##### 输出 #####
21
22 after json serialization
23 type of params_str = <class 'str'>, params_str = {'symbol': '123456', 'type': 'limit',
24 after json deserialization
25 type of original_params = <class 'dict'>, original_params = {'symbol': '123456', 'type'
```

其中，


`json.dumps()` 这个函数，接受 Python 的基本数据类型，然后将其序列化为 string；
而 `json.loads()` 这个函数，接受一个合法字符串，然后将其反序列化为 Python 的基本数据类型。

是不是很简单呢？

不过还是那句话，请记得加上错误处理。不然，哪怕只是给 `json.loads()` 发送了一个非法字符串，而你没有 catch 到，程序就会崩溃了。

到这一步，你可能会想，如果我要输出字符串到文件，或者从文件中读取 JSON 字符串，又该怎么办呢？

是的，你仍然可以使用上面提到的 `open()` 和 `read()/write()`，先将字符串读取 / 输出到内存，再进行 JSON 编码 / 解码，当然这有点麻烦。

 复制代码

```
1 import json
2
3 params = {
4     'symbol': '123456',
5     'type': 'limit',
6     'price': 123.4,
7     'amount': 23
8 }
9
10 with open('params.json', 'w') as fout:
11     params_str = json.dump(params, fout)
12
13 with open('params.json', 'r') as fin:
14     original_params = json.load(fin)
15
16 print('after json deserialization')
17 print('type of original_params = {}, original_params = {}'.format(type(original_params)
18
19 ##### 输出 #####
20
21 after json deserialization
22 type of original_params = <class 'dict'>, original_params = {'symbol': '123456', 'type'
```


这样，我们就简单清晰地实现了读写 JSON 字符串的过程。当开发一个第三方应用程序时，你可以通过 JSON 将用户的个人配置输出到文件，方便下次程序启动时自动读取。这也是现在普遍运用的成熟做法。

那么 JSON 是唯一的选择吗？显然不是，它只是轻量级应用中最方便的选择之一。据我所知，在 Google，有类似的工具叫做 Protocol Buffer，当然，Google 已经完全开源了这个工具，你可以自己了解一下使用方法。

相比于 JSON，它的优点是生成优化后的二进制文件，因此性能更好。但与此同时，生成的二进制序列，是不能直接阅读的。它在 TensorFlow 等很多对性能有要求的系统中都有广泛的应用。

总结

这节课，我们主要学习了 Python 的普通 I/O 和文件 I/O，同时了解了 JSON 序列化的基本知识，并通过具体的例子进一步掌握。再次强调一下需要注意的几点：

I/O 操作需谨慎，一定要进行充分的错误处理，并细心编码，防止出现编码漏洞；

编码时，对内存占用和磁盘占用要有充分的估计，这样在出错时可以更容易找到原因；

JSON 序列化是很方便的工具，要结合实战多多练习；

代码尽量简洁、清晰，哪怕是初学阶段，也要有一颗当元帅的心。

思考题

最后，我给你留了两道思考题。

第一问：你能否把 NLP 例子中的 word count 实现一遍？不过这次，in.txt 可能非常非常大（意味着你不能一次读取到内存中），而 output.txt 不会很大（意味着重复的单词数量很多）。

提示：你可能需要每次读取一定长度的字符串，进行处理，然后再读取下一次的。但是如果单纯按照长度划分，你可能会把一个单词隔断开，所以需要细心处理这种边界情况。

第二问：你应该使用过类似百度网盘、Dropbox 等网盘，但是它们可能空间有限（比如 5GB）。如果有一天，你计划把家里的 100GB 数据传送到公司，可惜你没带 U 盘，于是

你想了一个主意：

每次从家里向 Dropbox 网盘写入不超过 5GB 的数据，而公司电脑一旦侦测到新数据，就立即拷贝到本地，然后删除网盘上的数据。等家里电脑侦测到本次数据全部传入公司电脑后，再进行下一次写入，直到所有数据都传输过去。

根据这个想法，你计划在家写一个 `server.py`，在公司写一个 `client.py` 来实现这个需求。

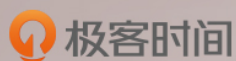
提示：我们假设每个文件都不超过 5GB。

你可以通过写入一个控制文件（`config.json`）来同步状态。不过，要小心设计状态，这里有可能产生 `race condition`。

你也可以通过直接侦测文件是否产生，或者是否被删除来同步状态，这是最简单的做法。

不要担心难度问题，尽情写下你的思考，最终代码我也会为你准备好。

欢迎在留言区写下你的答案，也欢迎你把这篇文章转给你的同事、朋友，一起在思考中学习。



Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

上一篇 05 | 深入浅出字符串

下一篇 07 | 修炼基本功：条件与循环

精选留言 (78)

写留言



Jingxiao 置顶

2019-05-26

13

思考题第二题：（省略了一些异常处理，后面会讲到）

```
server.py
```

```
# 我们假设 server 电脑上的所有的文件都在 BASR_DIR 中，为了简化不考虑文件夹结构，  
网盘的路径在 NET_DIR
```

...

展开



Jingxiao 置顶

2019-05-26

11

思考题第一题：

```
import re
```

```
CHUNK_SIZE = 100 # 这个数表示一次最多读取的字符长度...
```

展开



不瘦到140...

2019-05-22

15

```
from collections import defaultdict
```

```
import re
```

```
f = open("ini.txt", mode="r", encoding="utf-8")
```

```
d = defaultdict(int)...
```

展开

作者回复:



逆光飞翔

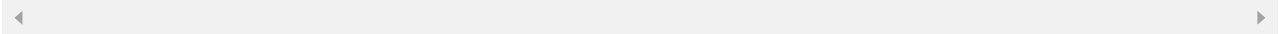
2019-05-22

👍 7

老师，为什么filter (none, list) 可以过滤空值，不是保留空值嘛

展开 ▾

作者回复: filter(None, Iterable) 是一种容易出错的用法，这里不止过滤空字符串，还能过滤 0, None, 空列表等值。这里的 None，严格意义上等于 lambda x: x, 是一个 callable。



Geek_59f23...

2019-05-22

👍 5

第一题:

1、使用defaultdict初始化计数器更方便更快，不用再多做一步in判断，parse函数只需返回filter对象。

2、读取大文件时使用for循环遍历迭代器，不占用内存空间，生成一行处理一行，就此例...

展开 ▾



Python高...

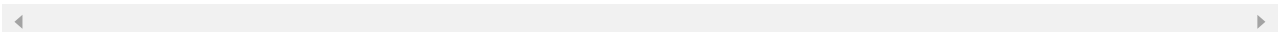
2019-05-22

👍 5

第一问:

```
with open("in.txt", "rt") as f:
    for line in f:
        Counter.update(line)
```

作者回复: 想到 Counter 很棒，但是这样注意这样不行哦，Counter 初始化输入 str 会把 str 视作一个容器，最后 Counter 里存的全是单个字符和它的次数，而不是单词。



lmingzhi

2019-05-22

👍 4

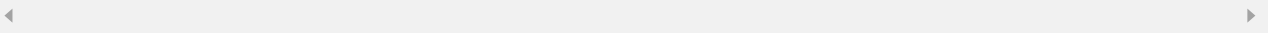
```
# 第一题， 修改parse函数，使其可以更新word_cnt
import re
def parse(text, word_cnt):
```

转为小写

```
text = text.lower() ...
```

展开 ▾

作者回复: ㄟ



許敲敲

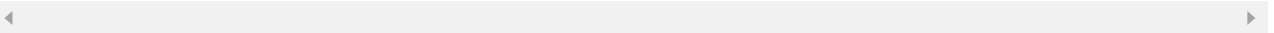
2019-05-22

👍 4

这门课太值了 哈哈 我以前学到的真的toy python

展开 ▾

作者回复: 谢谢，加油！



Lone

2019-05-22

👍 3

第一题打卡

```
import re
```

```
def parse(text, word_cnt):
```

```
    text = text.lower()...
```

展开 ▾



人间乐园

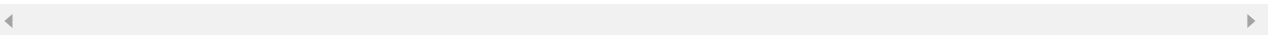
2019-05-22

👍 3

第一道，for line in fin 读取单行，使用 result = yield line 进行双向传递，直接把 line 给计数器，先判断 line 结尾处，如果是单词或者半个单词，则返回 result 给生成器，拼接到下一个 line 前，如果是 None 则不拼接，继续生成这个 line。

展开 ▾

作者回复: 很棒的思路，但是注意如果原始文件只有一行呢？你想说的是读取一定长度的字符串吧





Tango
2019-05-22



第七天打卡。

展开 ▾

作者回复: ㇏



.....
2019-05-23



```
from collections import defaultdict
import re
```

```
d = defaultdict(int)
```

...

展开 ▾



zxq
2019-05-22



老师，想请问一下python关于线程管理机制，举个例子来说：我有两个程序read.py和write.py，一个是负责读取info.txt内容，而另一个是将数据写入这个txt。由于我不知道两个python程序会不会同时被调用，假设调用write.py写入数据时，尚未释放txt的端口，这时候read.py被调用来读取数据，这时候就有冲突了。python如何处理这样的机制呢？如何优化这两个py文件，可不可以通过调用端口占用状态来解决呢？

展开 ▾



Fergus
2019-05-26



#Ask 1

NLP 自然语言处理

1.假设文件足够大，不能一次性读入内存

2.注意边界处理，如果单纯按数量读取，可能会将单词拆断...

展开 ▾



code2



2019-05-23

看到一片短文，摘录如下：

数据科学家是“比软件工程师更擅长统计学，比统计学家更擅长软件工程的人”。许多数据科学家都具有统计学背景，但是在软件工程方面的经验甚少。我是一名资深数据科学家，在Stackoverflow的python编程方面排名前1%，并与许多（初级）数据科学家共事。以下是我经常看到的10大常见错误，本文将为你相关解决方案： ...

展开 ▾



进击的菜鸟...

2019-05-23

👍 1

第一题：

```
from collections import defaultdict
import re
```

...

展开 ▾



GentleCP

2019-05-23

👍 1

老师github链接还没放出吗

展开 ▾

作者回复: 周末左右会给出答案，这几天你们可以思考一下哦



John Si

2019-05-23

👍 1

一個十年前讀CS畢業的人，因害怕失敗而放棄編程，沒有從事編程相關工作。現在後悔不已，所以報讀了老師的課程，在聽了老師幾堂課後發現編程原來沒自己想像那麼難，失敗也不是甚麼大問題。但要跟上進度,還是有點吃力，但我會努力學習並追上各同學的程度。老師提出的思考題其實我並沒有想法，可能是我水平太低了，請問老師能否提點一下呢？謝謝！

展开 ▾



宝仔

2019-05-23

👍 1

word_cnt = dict()! 老师你之前不是说过: word_cnt = {} 这样初始化字典不是效率更高吗? 为什么你代码里用了函数初始化字典



倾

2019-05-22

👍 1

之前用python写过项目代码, 不过是半路出家, 更多的是去网上找自己用到的东西, 现在又在用go写项目代码, 不过python值得好好学习, 不过只能在空闲的时候好好琢磨一下了。

展开 ∨