使用 Python 库获取豆瓣影评

1. 需求分析
2. 内置的 urllib 库与第三方 requests 库的用法差异
3. 使用 requests 库获取豆瓣影评
4. 使用 BeautifulSoup 库解析网页

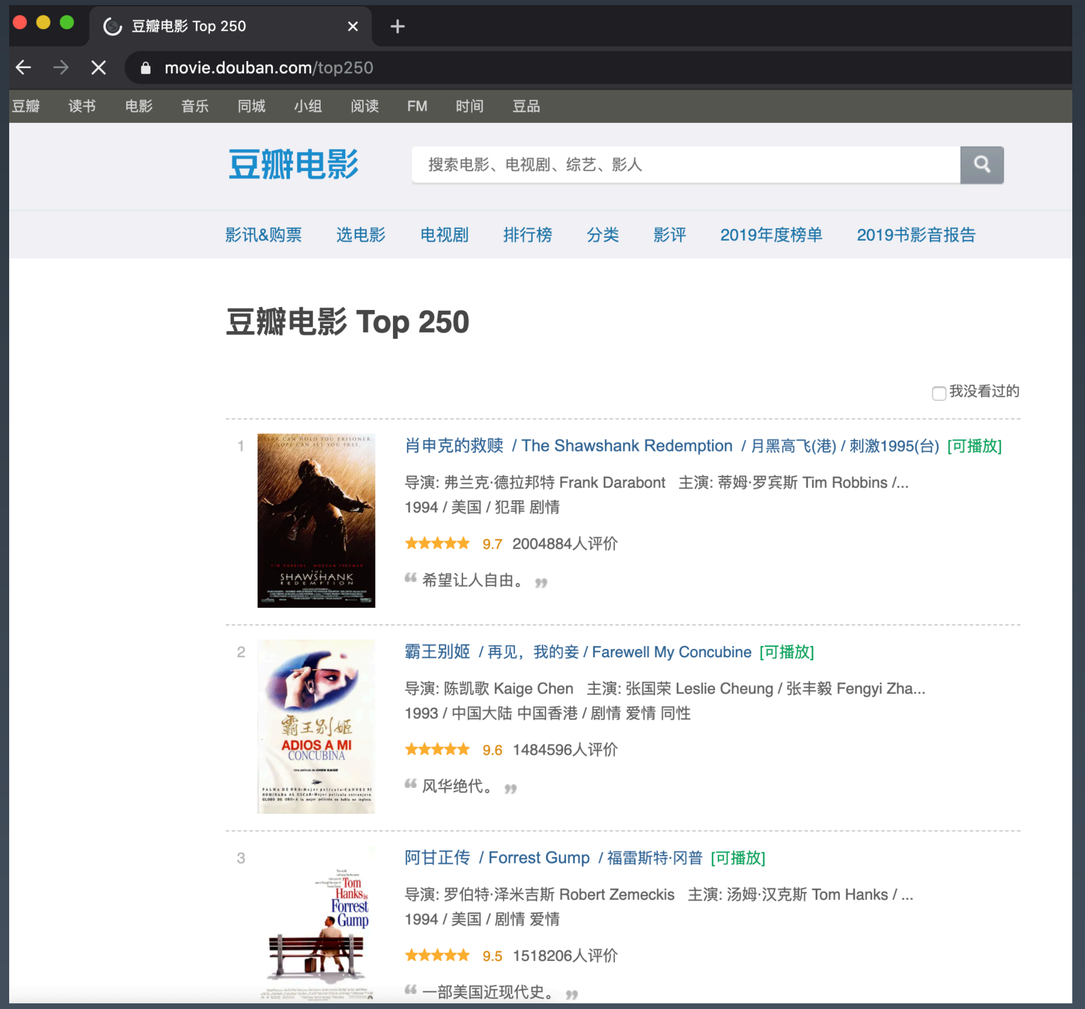
需求分析

获取《豆瓣电影 Top 250》的内容

https://movie.douban.com/top250?start=0

要求：

- 获取电影名称、豆瓣评分、短评数量
- 写入文本文件



内置的 urllib 库与第三方 requests 库的用法差异

内置库与第三方库的差异

第三方库一般需要使用 pip 进行安装
$ pip install requests

urllib 与requests库的文档差异

```
192:test edz$ pip install requests
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting requests
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/1a/70/1
a8b78ced23d7e0f3b187f5cbfab4749523ed65d7c9b1/requests-2.23.0-py2.
l (58 kB)
Requirement already satisfied: idna<3,>=2.5 in /Library/Framework
ork/Versions/3.7/lib/python3.7/site-packages (from requests) (2.9
Requirement already satisfied: certifi>=2017.4.17 in /Library/Fra
framework/Versions/3.7/lib/python3.7/site-packages (from requests
Requirement already satisfied: chardet<4,>=3.0.2 in /Library/Fram
ramework/Versions/3.7/lib/python3.7/site-packages (from requests)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1
ry/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
quests) (1.25.8)
Installing collected packages: requests
Successfully installed requests-2.23.0
```

Python 实战 —— 爬取豆瓣影评

- 使用 requests 库获取豆瓣影评
- 使用 BeautifulSoup 库解析网页
- 将电影名称、上映日期、评分保存至文件
- 运行与调试方法
- 原理解析

关键代码

使用 requests 库获取豆瓣影评

```
user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36'

header = {}

header['user-agent'] = user_agent

response = requests.get(url, headers=header)
```

关键代码

使用 BeautifulSoup 库解析网页

```
# Python 中使用 for in 形式的循环,Python使用缩进来做语句块分隔
for tags in bs_info.find_all('div', attrs={'class': 'pl2'}):
    for atag in tags.find_all('a',):
        # 获取所有链接
        print(atag.get('href'))
        # 获取图书名字
        print(atag.get('title'))
```

Python 的基础语法

1. 赋值
2. 内置数据类型与基本操作
3. 条件与循环
4. 函数
5. 类
6. 鸭子类型

解释和执行

Python 的解释器

Python 交互模式运行

Python 非交互模式运行

赋值

变量的赋值

使用“ = ”进行赋值

注意：不能以关键字做 Python 的变量名

Python 的关键字

False	True	None	class	type	and
def	del	if	elif	else	as
break	continue	for	from	import	in
pass	Not	is	or	return	try
except	while	assert	finally	nonlocal	lambda
raise	with	yield			

基本数据类型

常见的数据类型：

整型、浮点型、字符串、布尔值、空值、列表、元组、 字典、集合。

基本数据类型

数据类型	举例
整型(int)	-1、0、4、
浮点型(float)	5.2、3.14156
布尔值(bool)	True、False
空值	NULL
字符串(str)	'geekbang'、"3.8"
列表(list)	[1, 4, 9, 16, 25]
元组(tuple)	（1, 4, 9, 16, 25）
字典(dict)	{'a':1, 'b':2}
集合	{'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

条件和循环

if 语句

```
if_stmt ::=  "if" expression ":" suite
         ("elif" expression ":" suite)*
         ["else" ":" suite]
```

for 语句

```
for_stmt ::=  "for" target_list "in" expression_list ":" suite
          ["else" ":" suite]
```

while 语句

```
while_stmt ::=  "while" expression ":" suite
           ["else" ":" suite]
```

参考：https://docs.python.org/zh-cn/3.7/reference/compound_stmts.html

推导式

推导式写法：

```
tuple(f'https://movie.douban.com/top250?start={ page * 25 }&filter=' for page in range(10))
```

展开写法：

```
a = []
for page in range(10):
    pages=f'https://movie.douban.com/top250?start={ page * 25 }&filter='
    a.append(pages)
tuple(a)
```

你更喜欢哪种写法？

类

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'

x = MyClass()
```

函数

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

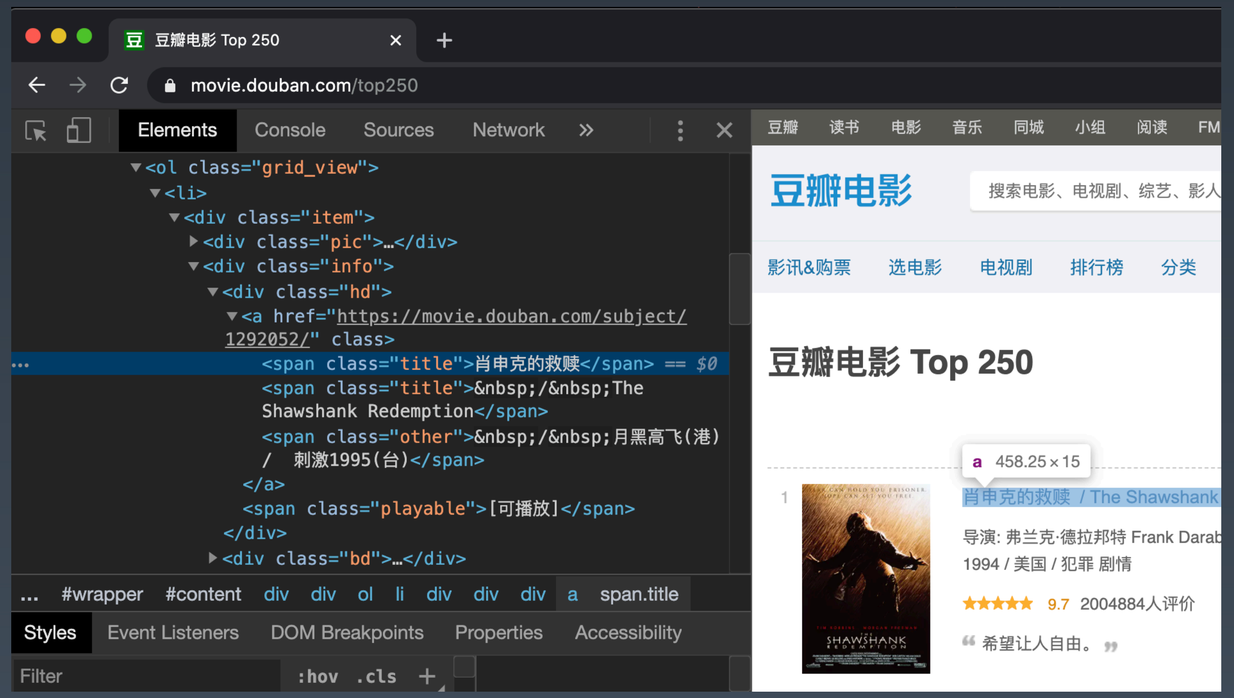
爬虫工程师必备的 HTML 基础

1. HTTP 协议与浏览器的关系
2. HTTP 协议请求与返回头部
3. HTTP 请求方式 get post delete head put
4. HTTP 状态码
5. W3C 标准
6. HTML 常用标签和属性
7. CSS、JavaScript、JSON 简介

浏览器

HTTP 服务端与 HTTP 客户端

浏览器的调试功能

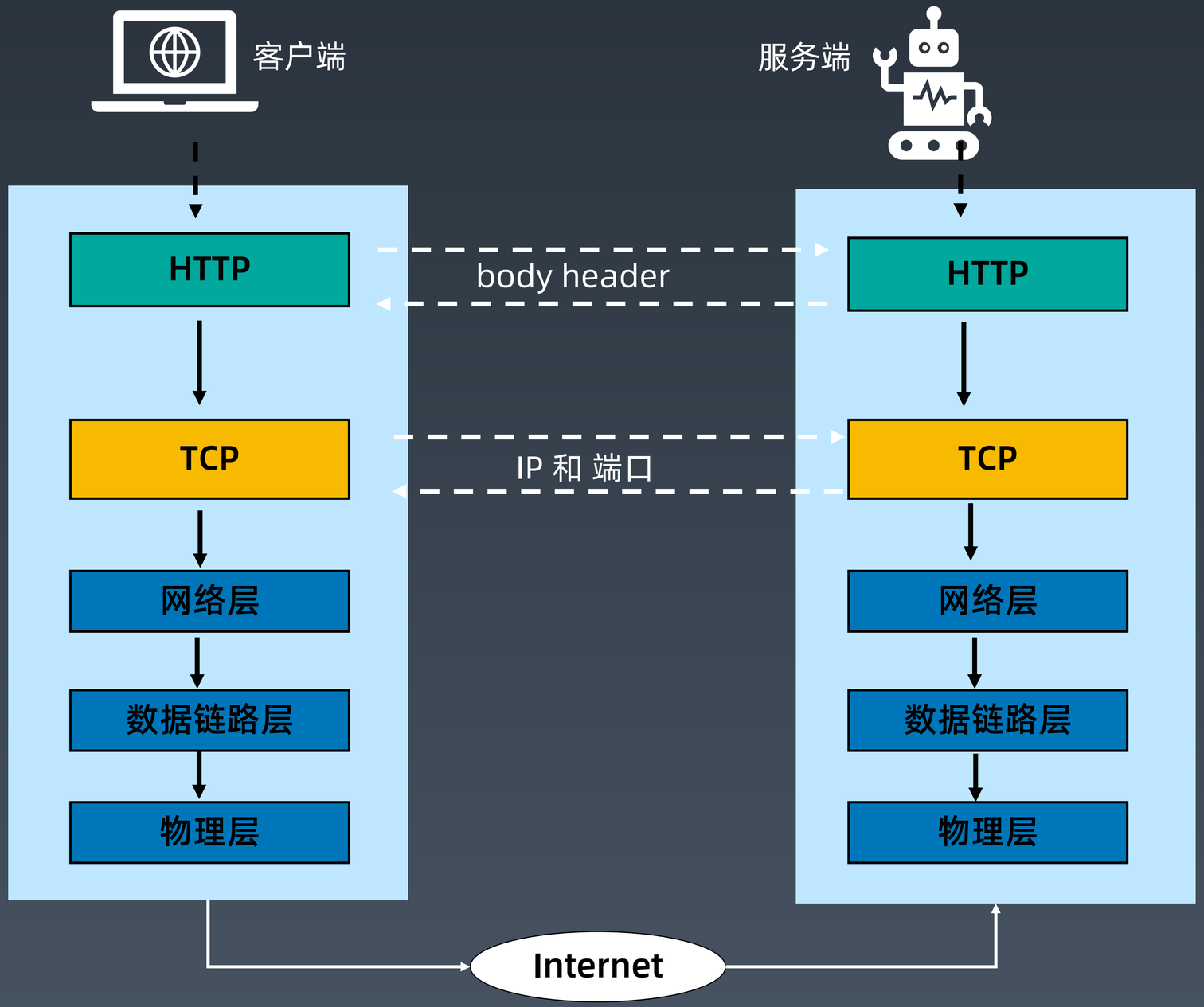


HTTP 协议

HTTP 协议请求与返回头部

HTTP 请求方式 get post delete head put

HTTP 状态码


客户端
服务端
HTTP
body header
HTTP
TCP
IP 和 端口
TCP
网络层
网络层
数据链路层
数据链路层
物理层
物理层
Internet

HTML

W3C 标准

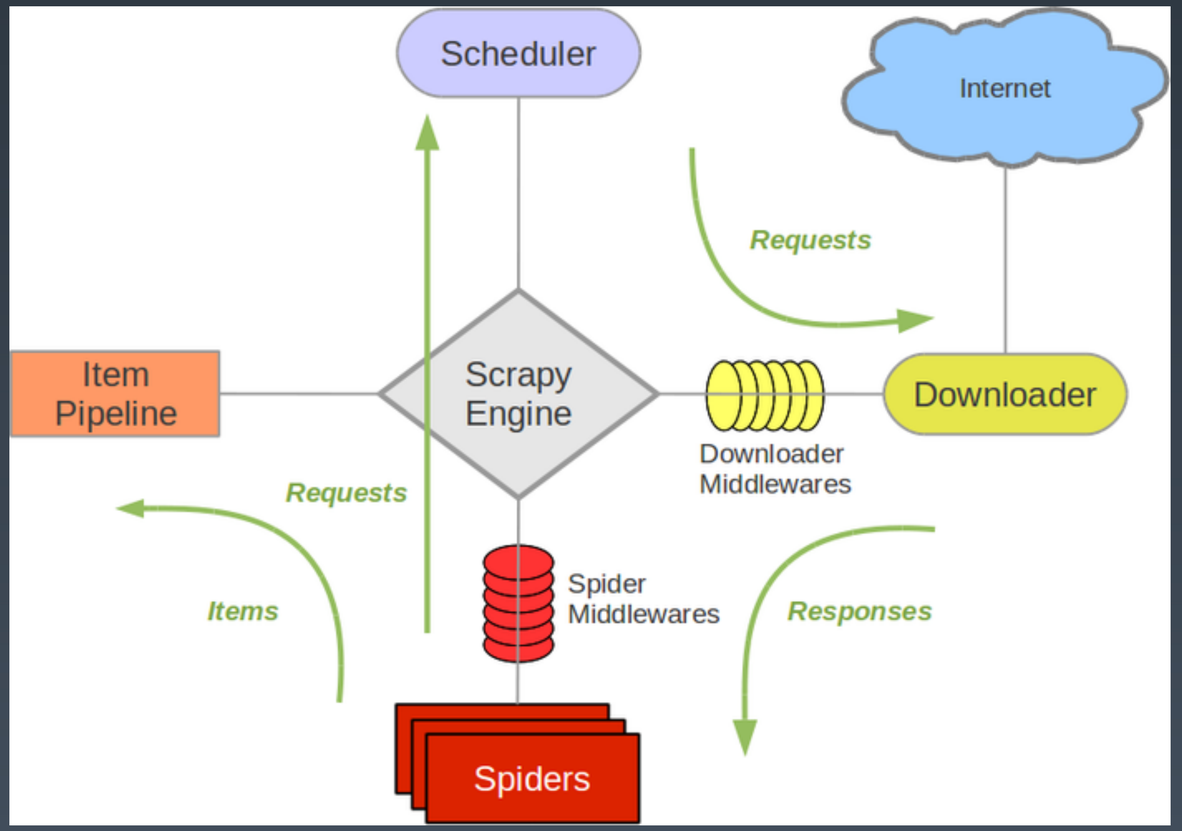
HTML 常用标签和属性

CSS、JavaScript、JSON 简介

Scrapy 核心组件

Scrapy 核心组件	简介
引擎（Engine）	“大脑”，指挥其他组件协同工作。
调度器（Scheduler）	调度器接收引擎发过来的请求，按照先后顺序，压入队列中，同时去除重复的请求。
下载器（Downloader）	下载器用于下载网页内容，并返回给爬虫。
爬虫（Spiders）	用于从特定的网页中提取需要的信息，即所谓的实体（Item） 用户也可以从中提取出链接，让 Scrapy 继续抓取下一个页面。
项目管道（Item Pipelines）	项目管道负责处理爬虫从网页中抽取的实体。 主要的功能是持久化实体、验证实体的有效性、清除不需要的信息等。
下载器中间件（Downloader Middlewares）	
爬虫中间件（Spider Middlewares）	

Scrapy 架构


Scheduler
Internet
Requests
Item
Pipeline
Scrapy
Engine
Downloader
Downloader
Middlewares
Requests
Items
Spider
Middlewares
Responses
Spiders

Scrapy 架构

组件	对应爬虫三大流程	Scrapy 项目是否需要修改
引擎		无需修改，框架已写好
调度器		无需修改，框架已写好
下载器	获取网页（request 库）	无需修改，框架已写好
爬虫器	解析网页（BeautifulSoup 库）	需要
管道	存储数据（存入csv/txt/MySQL 等）	需要
下载器中间件	获取网页 - 个性化部分	一般不用
爬虫器中间件	解析网页 - 个性化部分	一般不用

Scrapy 安装以及目录结构

Scrapy 安装：pip install scrapy

Scrapy 目录结构	
实现爬虫的Python文件	spiders 目录
项目的设置文件	settings.py
项目的配置文件	scrapy.cfg
定义所爬取记录的数据结构	items.py
编写爬虫逻辑	pachong.py
设置保持位置	pipelines.py

Scrapy 实战

实战使用 Scrapy 改写 requests 单线程爬虫

关键代码

requests

```
response = requests.get(myurl,headers=header)

import pandas as pd
movie1 = pd.DataFrame(data = mylist)
```

scrapy

```
yield scrapy.Request(url=url, callback=self.parse)


item = response.meta['item']
item['content'] = content
```

用到的 Python 技巧

推导式

yield 语句

推导式

推导式是构建列表、字典、集合和生成器便捷方式

一个列表推导式的例子:

```
mylist = []

for i in range(1, 11):
    if i > 5:
        mylist.append(i**2)

print(mylist)
```

转换为列表推导式:

```
mylist = [i**2 for i in range(1, 11) if i > 5]
```

推导式

推导式语法：[表达式 for 迭代变量 in 可迭代对象 if 条件]

其他常见用法：

循环嵌套

```
mylist = [str(i)+j for i in range(1, 6) for j in 'ABCDE']
```

字典转换为列表

```
mydict = {'key1' : 'value1', 'key2' : 'value2'}
mylist = [key + ':' + value for key, value in mydict.items()]
print(mylist)
```

字典 key 和 value 互换

```
{value: key for key, value in mydict.items()}
```

推导式

字典推导式

```
mydict = {i: i*i for i in (5, 6, 7)}
print(mydict)
```

集合推导式

```
myset = {i for i in 'HarryPotter' if i not in 'er'}
print(myset)
```

元组推导式要显式使用 tuple(), 不能直接使用()

生成器

```
mygenerator = (i for i in range(0, 11))
print(mygenerator)
print(list(mygenerator))
```

yield

yield 可以作为语句和表达式来使用

yield 和 return 的区别

```
def chain(*iterables):
    for it in iterables:
        yield it

>>> s = 'ABC'
>>> list(chain(s))
['A', 'B', 'C']
```

THANKS!