



廈門大學
XIAMEN UNIVERSITY



高校大数据课程

《大数据导论》课程作业实验报告

任务

1

2022.12.3

黄勔 22920212204392

目录

一、	实验准备	3
(1)	任务目标	3
(2)	开发环境	3
(3)	准备工作	3
	✓ Python 3 的安装	3
	✓ 请求库的安装	4
	✓ 解析库的安装	5
二、	实验过程	6
(1)	抓取分析	6
(2)	打开页面	7
(3)	属性提取	7
(4)	写入文件	8
(5)	分页提取	8
(6)	整合代码	8
三、	实验结果与重现	10
(1)	实验结果	10
(2)	重现方法	11
四、	实验问题与总结	11
(1)	实验问题	11
(2)	总结	11

一、实验准备

(1) 任务目标

本次任务我选择了网站“Quotes to Scrape” (<https://quotes.toscrape.com/>)编写 Python 程序，爬取网页中的名言数据，保存到一个文本文件中。

选择这个网站的原因是首先这个网站的结构较为简单，利于初学者阅读 HTML 并快速上手编程；其次该网站的数据较为丰富，有大量的名言文本，便于我们抓取；最后这个网站的设计初衷就是让爬虫访问这个站点（而不访问 Quotes 主站），不会让爬虫造成主站的拥堵，有利于互联网生态的维护。

本次任务提交的文件包含本实验报告、代码文件（.py 格式）和采集到的数据文件，文件均包含在本目录，本实验报告主要叙述了我在分析、练习与实战编写爬虫的思路过程。

(2) 开发环境

由于任务 1 的实验要求较为简单，在此简要的叙述我的编程环境：

- 操作系统：Windows 10 64 位 版本 21H2
- 编程环境：Python 3.9.6
- 编辑器：VS Code

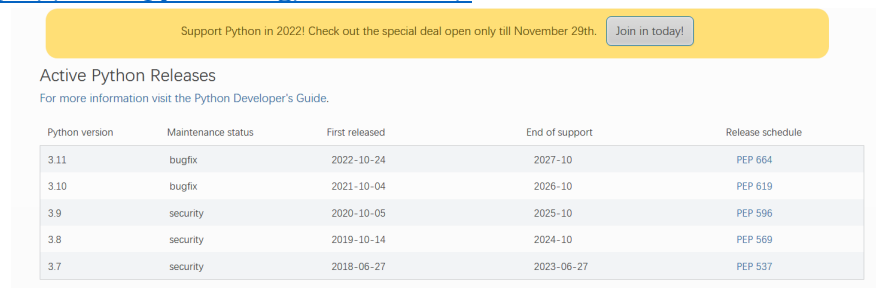
(3) 准备工作

✓ Python 3 的安装

既然要使用 Python 3 开发爬虫，那么第一步一定是安装 Python 3.在此我选择了使用下载安装包安装的形式。

首先到官方网站下载 Python3 的安装包：

<https://www.python.org/downloads/>.



Support Python in 2022! Check out the special deal open only till November 29th. [Join in today!](#)

Active Python Releases
For more information visit the [Python Developer's Guide](#).

Python version	Maintenance status	First released	End of support	Release schedule
3.11	bugfix	2022-10-24	2027-10	PEP 664
3.10	bugfix	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537

图 1 官网下载界面

下载完成之后，直接双击 Python 安装包，然后通过图形界面安装，接着设置 Python 的安装路径，

完成后将 Python 3 和 Python 3 的 Scripts 目录配置到环境变量。我安装 Python 3 路径为 D:\Program Files，从资源管理器中打开该路径。

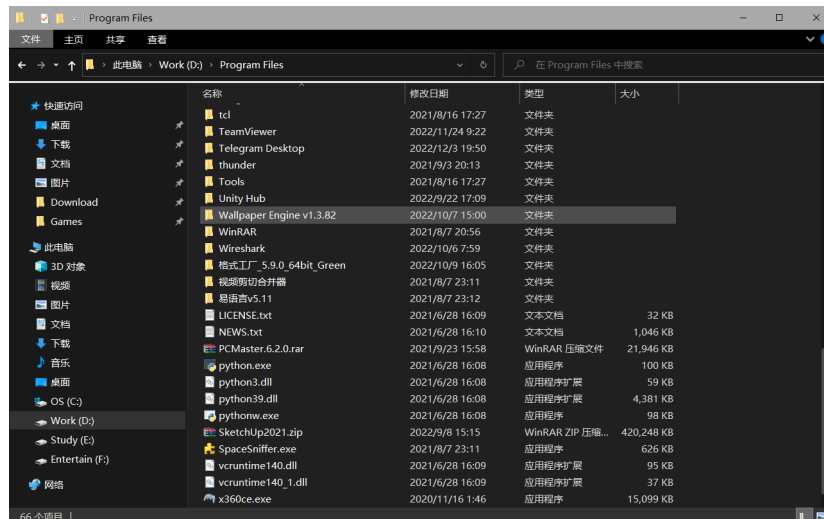


图 2 安装目录界面

将路径复制下来，在“计算机” - “属性” - “高级系统设置”中，点击“环境变量”，在系统 Path 变量中添加 Python 3 的安装目录。

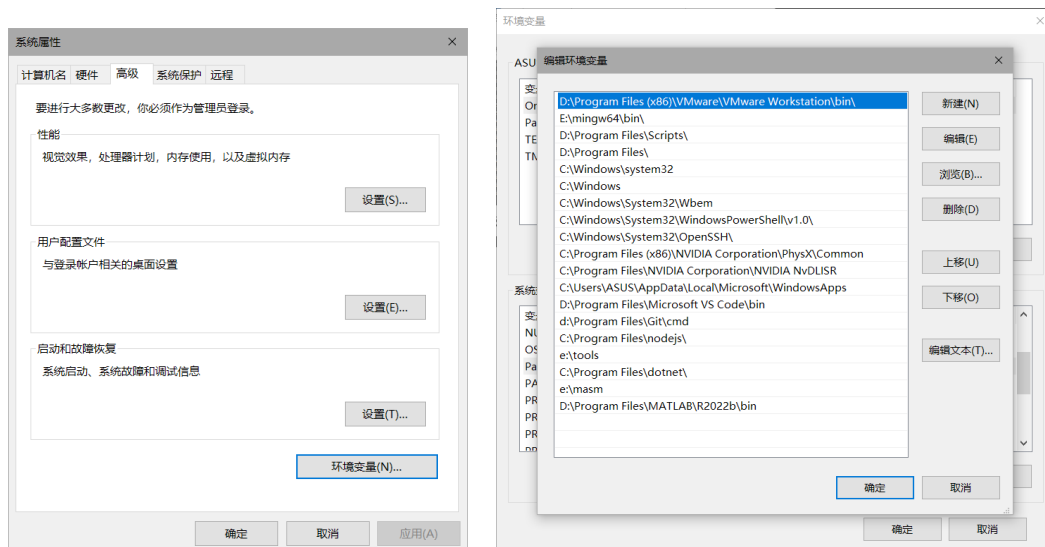


图 3 配置环境变量

完成后,我们可以通过命令行测试安装是否成功。在“开始”菜单中搜索 cmd, 找到命令提示符, 此时就进入命令行模式了。输入 python, 测试一下能否成功调用 Python, 测试结果如图所示。

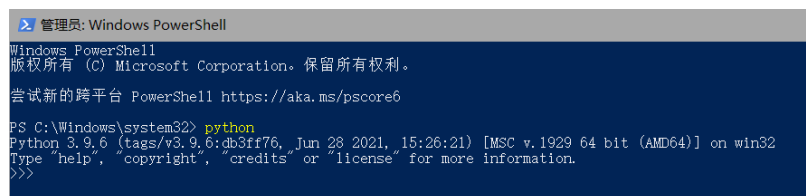


图 4 测试安装结果

✓ 请求库的安装

requests 的安装: requests 请求库运用十分普遍在此不多做介绍。在这里选择 pip 安装, 运行 pip install requests 即可。

selenium 的安装: selenium 是一个自动化测试工具, 利用它可以驱动浏览

器执行特定的操作，如点击、下拉等，在我以往的使用过程中发现，这种抓取方式往往是最有效的且最便捷的。安装方法同上。

ChromeDriver 的安装：selenium 库需要配合浏览器使用，在此我选用 ChromeDriver 驱动配置。

首先，下载 Chrome 浏览器，方法有很多，在此不再赘述。

随后安装 ChromeDriver。因为只有安装 ChromeDriver,才能驱动 Chrome 浏览器完成相应的操作。在此给出下载地址：

<https://chromedriverstorage.googleapis.com/index.htm>

随后查看 chrome 版本，点击 Chrome 菜单“帮助”→“关于 Google Chrome”，即可查看 Chrome 的版本号，如图所示。



图 5 Chrome 版本页面

找好对应的版本号后，随后到 ChromeDriver 镜像站下载对应的安装包即可：

<https://chromedriver.storage.googleapis.com/index.html>

Index of /107.0.5304.62/

	Name	Last modified	Size	ETag
	Parent Directory		-	
	chromedriver_linux64.zip	2022-10-25 12:20:56	7.26MB	90d3353f17fcbd755626d528e94a1d9a
	chromedriver_mac64.zip	2022-10-25 12:20:57	8.41MB	652c969a3b8d47e7fa9518d90b411fba
	chromedriver_mac_arm64.zip	2022-10-25 12:20:59	7.72MB	dba9920d41a8ec9fb847326ae0f68200
	chromedriver_win32.zip	2022-10-25 12:21:00	6.46MB	a5040d2731fe174c9a7b026edb3fe271
	notes.txt	2022-10-25 12:21:05	0.00MB	936b74dab32b11adddaffb0a624d9894a

图 6 Driver 镜像站页面

下载后将下载的目录添加环境变量配置即可。

配置完成后，就可以在命令行下直接执行 chromedriver 命令了，输入后的结果如下，这样就可以利用 Chrome 来做网页抓取了。

```
PS C:\Windows\system32> chromedriver
Starting ChromeDriver 105.0.5195.52 (412c95e518836d8a7d97250d62b29c2ae6a26a85-refs/branch-heads/5195@{#853}) on port 9515
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
```

图 7 测试 Driver 配置结果

✓ 解析库的安装

Beautiful Soup 是 Python 的一个 HTML 或 XML 的解析库，我们可以用它来方便地从网页中提取数据。它拥有强大的 API 和多样的解析方式，在此处利用 `pip install beautifulsoup4` 即可安装。

```
PS C:\Users\ASUS> pip install beautifulsoup4
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.11.1-py3-none-any.whl (128 kB)
    128.2/128.2 kB 269.2 kB/s eta 0:00:00
```

图 8 配置 BeautifulSoup

二、实验过程

(1) 抓取分析

Quotes to Scrape 这个网站大致有 10 页谚语，如图所示。

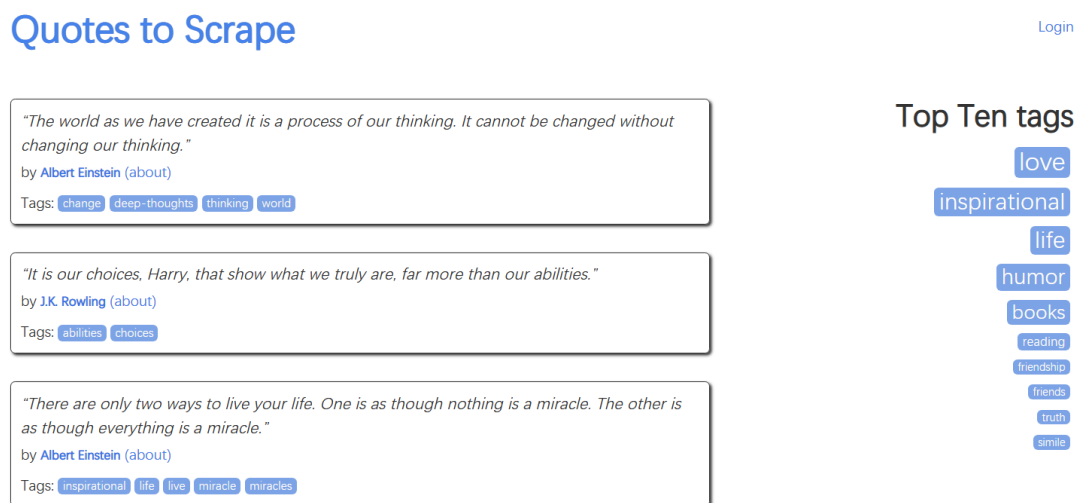


图 9 目标网站 Quotes to Scrape

排名第一的是名言：“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.” 页面中有效的信息还有 Tags 和作者，在任务 1 中我仅选取了名言进行爬取。

将网页滚动到最下方，可以发现有分页的按钮，直接点击下一页，观察变化。

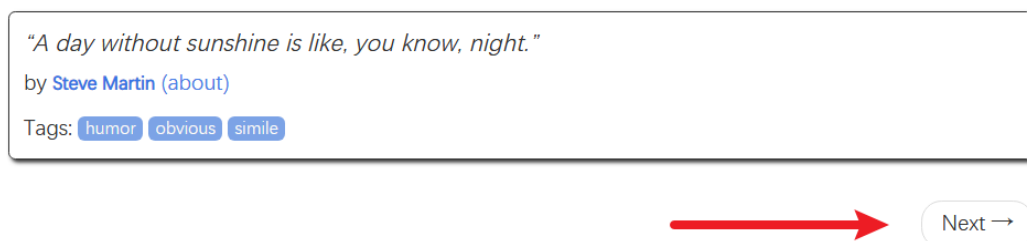
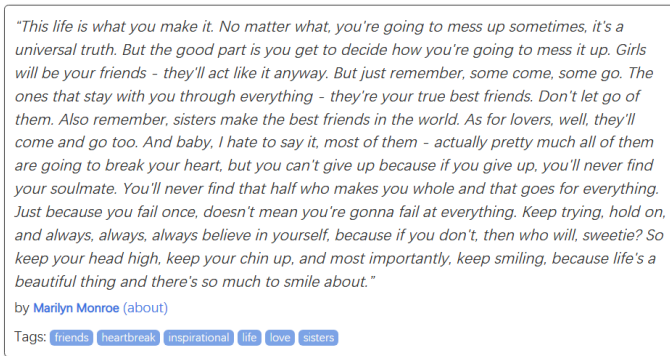


图 10 下一页按钮

此时网页便会跳转到下一页。

Quotes to Scrape

Login



Top Ten tags

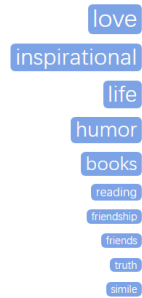


图 11 下一页界面

所以我的思路便是利用 selenium 和 beautifulsoup，大致思路是使用 webdriver 获取“下一页”按钮，获取每一页源码，提取所要的谚语。

(2) 打开页面

接下来用代码实现这个过程。首先抓取第一页的内容。我实现了 page_request() 方法，并给它传入 driver 和 url 参数。然后将抓取的页面结果返回，再通过 main() 方法调用。初步代码实现如下：

```
7 def page_request(driver: webdriver, url):
8     # 打开目标网址首页
9     driver.get(url)

43 def main():
44     url = 'http://quotes.toscrape.com/'
45     # 加载驱动
46     chromedriver = webdriver.Chrome() #在此我已经配置了driver的路径在环境变量，若未配置需要在括号中填入driver路径
47     quotesoup = BeautifulSoup(chromedriver.page_source, 'lxml')
48     page_request(chromedriver, url)
```

(3) 属性提取

打开 Chrome 的开发者工具，找到谚语所在的位置

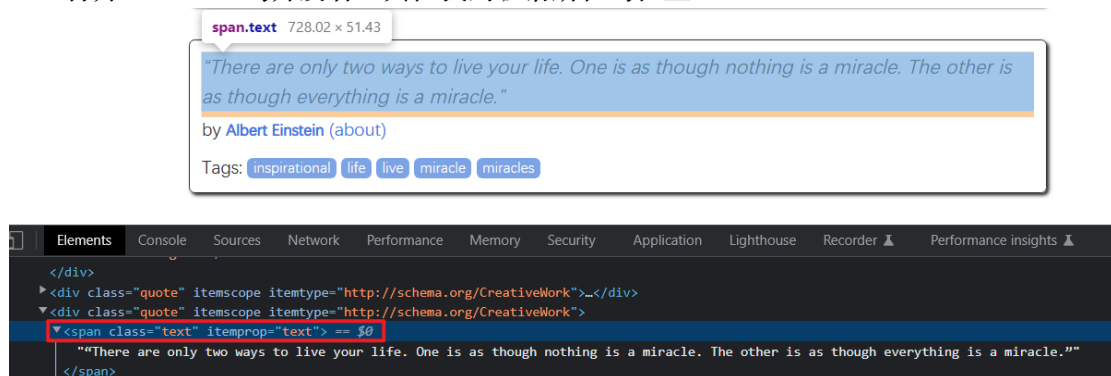


图 12 谚语所在位置

我这里发现谚语是在 span 下 class 属性位“text”的文本，由此即可通过指定属性匹配谚语，编写代码如下：

```

22 def page_parse(driver,soup):
23     i = 10
24     j = 1
25     while i > 0:
26         try:
27             soup = BeautifulSoup(driver.page_source, 'lxml')
28             # 找到并获取第一页的谚语位置span集合:items, 点击下一页之后会变成下一页的谚语集合
29             items = soup.find_all('span', class_='text')
30             # 打印获取到第一页的谚语
31             for item in items:
32                 save_txt('谚语'+str(j)+':')
33                 save_txt(item.text)
34                 j += 1
35             next_request(driver)
36             i -= 1
37             # 停顿1秒, 页面观察点击下一页的效果
38             time.sleep(1)
39         except:
40             return None

```

(4) 写入文件

写入文件较为简单, 在此我实现了 save_txt()方法:

```

16 def save_txt(info):
17     print(info)
18     with open('result.txt', 'a', encoding='utf-8') as f:
19         f.write(info + '\n')

```

(5) 分页提取

找到下一页按钮 Next

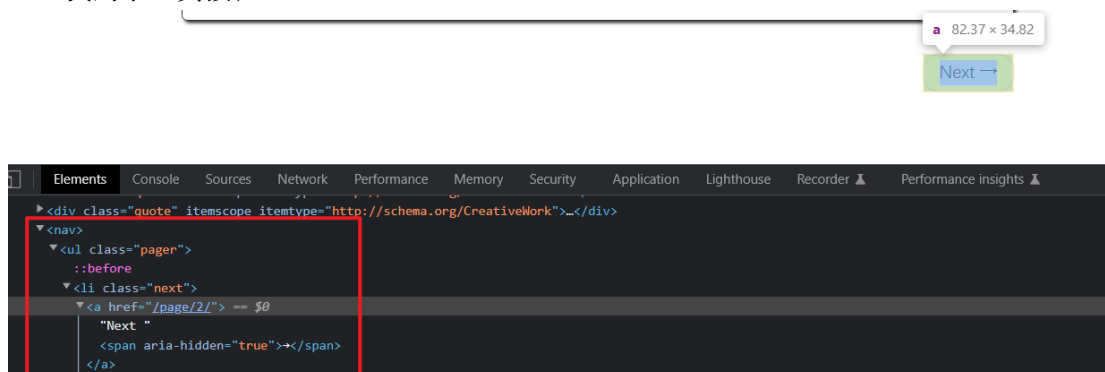


图 13 下一页按钮元素

此时通过 selenium 模拟点击 Next 按钮即可跳转到下一页, 在此我实现了 next_request()方法让网页进入下一页:

```

11 def next_request(driver: webdriver):
12     # 获取下一页next按钮
13     elem = driver.find_element(By.XPATH,'//ul[@class="pager"]/li[@class="next"]/a')
14     elem.click()

```

(6) 整合代码

通过函数整合与 main 函数的编写, 我得到了完整的代码:

```

from bs4 import BeautifulSoup
from selenium import webdriver

```



```

from selenium.webdriver.common.by import By
import time

def page_request(driver: webdriver,url):
    # 打开目标网址首页
    driver.get(url)

def next_request(driver: webdriver):
    # 获取下一页 next 按钮
    elem =
driver.find_element(By.XPATH,'//ul[@class="pager"]/li[@class="next"]/a'
)
    elem.click()

def save_txt(info):
    print(info)
    with open('result.txt', 'a', encoding='utf-8') as f:
        f.write(info + '\n')

def page_parse(driver,soup):
    i = 10
    j = 1
    while i > 0:
        try:
            soup = BeautifulSoup(driver.page_source, 'lxml')
            # 找到并获取第一页的谚语位置 span 集合:items, 点击下一页之后会变成
            下一页的谚语集合
            items = soup.find_all('span', class_='text')
            # 打印获取到第一页的谚语
            for item in items:
                save_txt('谚语'+str(j)+':')
                save_txt(item.text)
                j += 1
            next_request(driver)
            i -= 1
            # 停顿 1 秒, 页面观察点击下一页的效果
            time.sleep(1)
        except:
            return None

def main():
    url = 'http://quotes.toscrape.com/'

```

```
# 加载驱动
chromedriver = webdriver.Chrome() #在此我已经配置了 driver 的路径在环境变量，若未配置需要在括号中填入 driver 路径
quotesoup = BeautifulSoup(chromedriver.page_source, 'lxml')
page_request(chromedriver,url)
page_parse(chromedriver,quotesoup)

if __name__ == '__main__':
    print("*****开始爬取 quotes 数据*****")
    main()
    print("*****爬取完成*****")
```

三、实验结果与重现

(1) 实验结果

运行过程如下图：

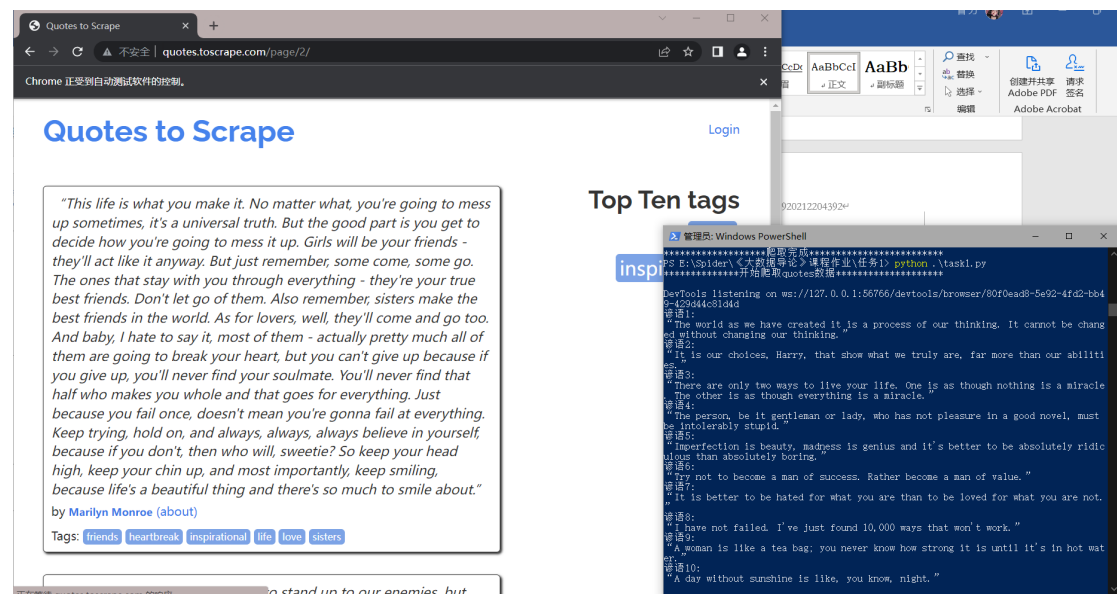


图 14 运行界面截图

目录下生成了“result.txt”即为 10 页的谚语数据。

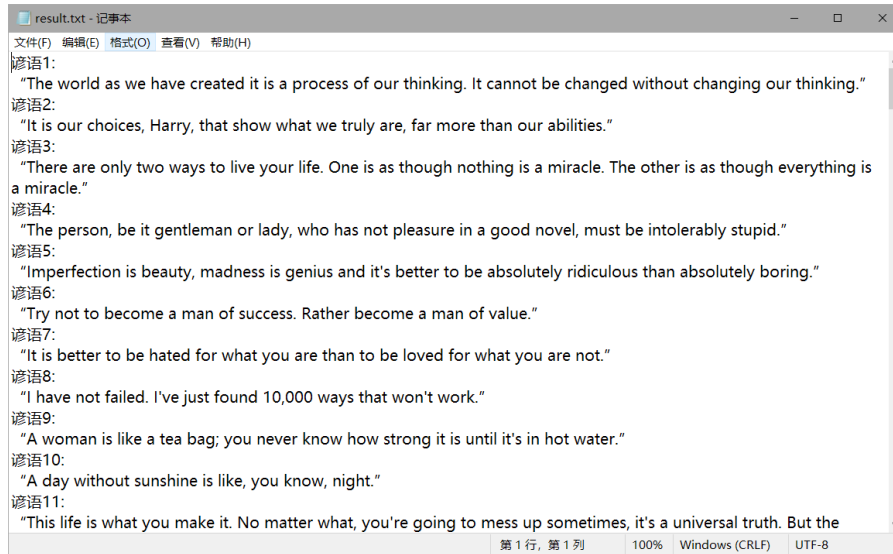


图 15 结果截图

(2) 重现方法

运行与本实验报告同目录下的“task1.py”即可（需要按照“准备工作”中的步骤安装相关库与 chromedriver）

运行成功后便能在目录下看到“result.txt”即为爬取到的数据。

四、实验问题与总结

(1) 实验问题

- 网页成功跳转，也没有抛出错误但没有成功获取到数据
原因是我初期编写代码时将 soup 传进子函数解析网址，但实际上 soup 没有正确返回，最后我将 soup 放在 page_parse()函数中，成功解决了问题。

(2) 总结

在这一次实验中我对爬虫有了更实际化的认识，在实验过程中虽然遇到了许多困难，但是通过一步步解决问题，我也对具体实现获取网页数据的方法有了更深刻的认识，并对设计爬虫的流程有了更多的思考，这让我受益匪浅；在下一个任务中我需要继续研究，还要很多需要研究和实践，这是一次很有意义的实验！