

前言

学习Python爬虫技术也是一件需要大量实践的事情，因为并不是所有的网站都对爬虫友好，更多的一种情况是网站为了限制爬虫不得不在最小化影响用户体验的前提下对网站访问做出一定的限制，最常见的就是一些网站的注册和登录页面出现的验证码。



12306网站的验证码在很长一段时间内饱受诟病，最初其复杂程度已经影响到了用户交互体验，但是为什么12306没有选择为了用户体验而放弃验证码？

因为验证码就像是一个门槛，它主要针对的并不是人，而是可能含有恶意的计算机程序。

12306网站堪称掌握地表最强开发和维护技术，它每天运行的服务器压力几乎都等同于双十一，高峰时段的操作压力甚至远超双十一十几倍。

这是个提供刚需服务的网站，稳定的重要性在一定程度上是大于用户体验的，如果没有验证码，那么程序就可以轻而易举的执行登录、注册等操作。

像我上一篇文章中提到的自动抢票软件，刷新访问的频率越高，对12306服务器的压力就越大。12306也不能分辨屏幕前坐着的到底是不是人，所以只能采取最简单粗暴的人机识别方法:验证码。

扯远了,不过这也是我们后续会遇到的反爬虫手段之一。

既然是爬虫，当然还是要先确认目标网站，这里推荐一个我认为非常不错的爬虫练习网站：[镀金的天空](#)

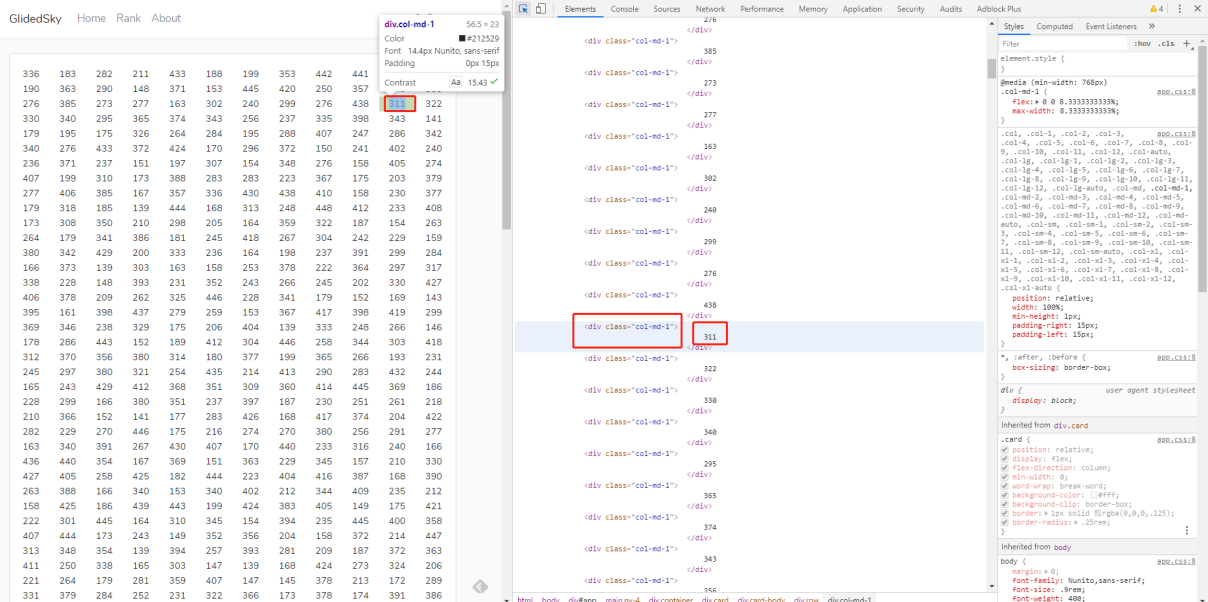
Title
爬虫-基础1
爬虫-基础2
爬虫-IP屏蔽1
爬虫-字体反爬-1
爬虫-CSS反爬
关注微信公众号
爬虫-验证码-1
爬虫-微信基佬群
爬虫-JS加密1
爬虫-雪碧图-1

这个网站中从易到难列出了8中反爬虫的挑战，今天要讲的就是其中的基础1：爬取单页数据并计算数据和

336	183	282	211	433	188	199	353	442	441	214	321
190	363	290	148	371	153	445	420	250	357	240	365
276	385	273	277	163	302	240	299	276	438	311	322
330	340	295	365	374	343	256	237	335	398	343	141
179	195	175	326	264	284	195	288	407	247	286	342
340	276	433	372	424	170	296	372	150	241	402	240
236	371	237	151	197	307	154	348	276	158	405	274
407	199	310	173	388	283	283	223	367	175	203	379
277	406	385	167	357	336	430	438	410	158	230	377
179	318	185	139	444	168	313	248	448	412	233	408
173	308	350	210	298	205	164	359	322	187	154	263
264	179	341	386	181	245	418	267	304	242	229	159
380	342	429	200	333	236	164	198	237	391	299	284
166	373	139	303	163	158	253	378	222	364	297	317
338	228	148	393	231	352	243	266	245	202	330	427
406	378	209	262	325	446	228	341	179	152	169	143
395	161	398	437	279	259	153	367	417	398	419	299
369	346	238	329	175	206	404	139	333	248	266	146
178	286	443	152	189	412	304	446	258	344	303	418
312	370	356	380	314	180	377	199	365	266	193	231
245	297	380	321	254	435	214	413	290	283	432	244
165	243	429	412	368	351	309	360	414	445	369	186
228	299	166	380	351	237	397	187	230	251	261	218
210	366	152	141	177	283	426	168	417	374	204	422
282	229	270	446	175	216	274	270	380	256	291	277
163	340	391	267	430	407	170	440	233	316	240	166
436	440	354	167	369	151	363	229	345	157	210	330
427	405	258	425	182	444	223	404	416	387	168	390
263	388	166	340	153	340	402	212	344	409	235	212
158	425	186	439	443	199	424	383	405	149	175	421
222	301	445	164	310	345	154	394	235	445	400	358
407	444	173	243	149	352	356	204	158	372	214	447
313	348	354	139	394	257	393	281	209	187	372	363
411	250	338	165	303	147	139	168	424	273	324	206
221	264	179	281	359	407	147	145	378	213	172	289
331	379	284	252	231	322	366	173	378	174	391	386

分析页面元素

按下f12调出开发者工具选择我们需要操作的数据，注意这里需要我们记录下数据在html文件中的特征



它是在一个名为col-md-1的div下的数，这个特征为我们之后的数据提取提供了依据。可能有的小伙伴不太懂html代码，这里简单提一下html语法的一些特性：

- 标签通常成对出现且分别用于闭合彼此（和我们之前讲的Python中的代码块相似，需要明确出元素开始和结束的位置）

- 标签通常可命名(本例中的class就是一中定义名称的方式，类似的还有id,作用相当于一个考场有50份题目一样的试卷，但是只有写了你名字的那个才算是你的试卷，同时也是为了定位到具体的数据)
- html代码中不同的标签有不同的定义关键字

HTML 标签

<!-->

<!DOCTYPE>

<a>

<abbr>

<acronym>

<address>

<applet>

<area>

<article>

<aside>

<audio>

<base>

<basefont>

<bdi>

<bdo>

<big>

<blockquote>

<body>

<button>

<canvas>

<caption>

<center>

<cite>

<code>

<col>

<colgroup>

<command>

此元素可告知浏览器其自身是一个 HTML 文档。

<html> 与 </html> 标签限定了文档的开始点和结束点，在它们之间是文档的头部和主体。正如您所了解的那样，文档的头部由 [<head> 标签](#) 定义，而主体由 [<body> 标签](#) 定义。

浏览器支持

IE	Firefox	Chrome	Safari	Opera
				

所有浏览器都支持 <html> 标签。

HTML 与 XHTML 之间的差异

xmlns 属性在 XHTML 中是必需的，但在 HTML 中不是。不过，即使 XHTML 文档中的 <html> 没有使用此属性，W3C 的验证器也不会报错。这是因为 "xmlns=http://www.w3.org/1999/xhtml" 是一个固定值，即使您没有包含它，此值也会被添加到 <html> 标签中。

[有关 xmlns 属性的更多信息。](#)

提示和注释

注释：即使 html 元素是文档的根元素，它也不包含 doctype 元素。doctype 元素必须位于 html 元素之前。

属性

- html中可内嵌css和JavaScript代码，也可以从外部引入它们

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- CSRF Token -->
    <meta name="csrf-token" content="Jk4LWRRfawypn6YRRE1m0ALlegiyNgdVx4Th801">
    <title>GlidedSky</title>
    <!-- Scripts -->
    <script src="https://www.googletagmanager.com/activeview/js/current/osd.js?cb=%2Fr20100101"></script>
    <script type="text/javascript" async src="https://www.google-analytics.com/analytics.js"></script>
    <script src="https://pagead2.googlesyndication.com/pagead/js/r20200316/r20190131/show_ads_impl_fy2019.js" id="google_shimpl"></script>
    <script src="https://hm.baidu.com/hm.js?020fbaa..."></script>
    <script src="http://glidedsky.com/js/app.js"></script>
    <!-- Fonts -->
    <link rel="dns-prefetch" href="//fonts.gstatic.com">
    <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet" type="text/css">
    <!-- Styles -->
    <link href="http://glidedsky.com/css/app.css" rel="stylesheet">
    <script async src="//pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
    <script>...</script>
    <!-- Global site tag (gtag.js) - Google Analytics -->
    <script async src="https://www.googletagmanager.com/gtag/js?id=UA-75859356-3"></script>
    <script>...</script>
    <script>...</script>
    <link rel="preload" href="https://adservice.google.com/adsid/integrator.js?domain=glidedsky.com" as="script">
    <script type="text/javascript" src="https://adservice.google.com/adsid/integrator.js?domain=glidedsky.com"></script>
  </head>
  <body data-feedly-mini="yes">
    <div id="app">
      <nav class="navbar navbar-expand-md navbar-light navbar-laravel">...</nav>
      <main class="py-4">
        ...
      </main>
    </div>
  </body>
</html>
```

内嵌就是直接写在当前html文件中

- html,css,javascript默认代码可直接查看，因此敏感操作不要写在外部js文件中（把敏感信息写在html中也不行）
- html分head、body
- 不同标签可能有不同的属性

这些内容还是需要去仔细的了解一下，本系列的文章基本不会再对html基础方向的问题进行叙述。

程序实现

与基础系列不同，为了尊重站长意愿，同时也为了大家能够更好的上手实践，涉及到glidedsky的文章将不会给出全部代码，但是会尽量讲清楚所涉及到的技术和部分代码。

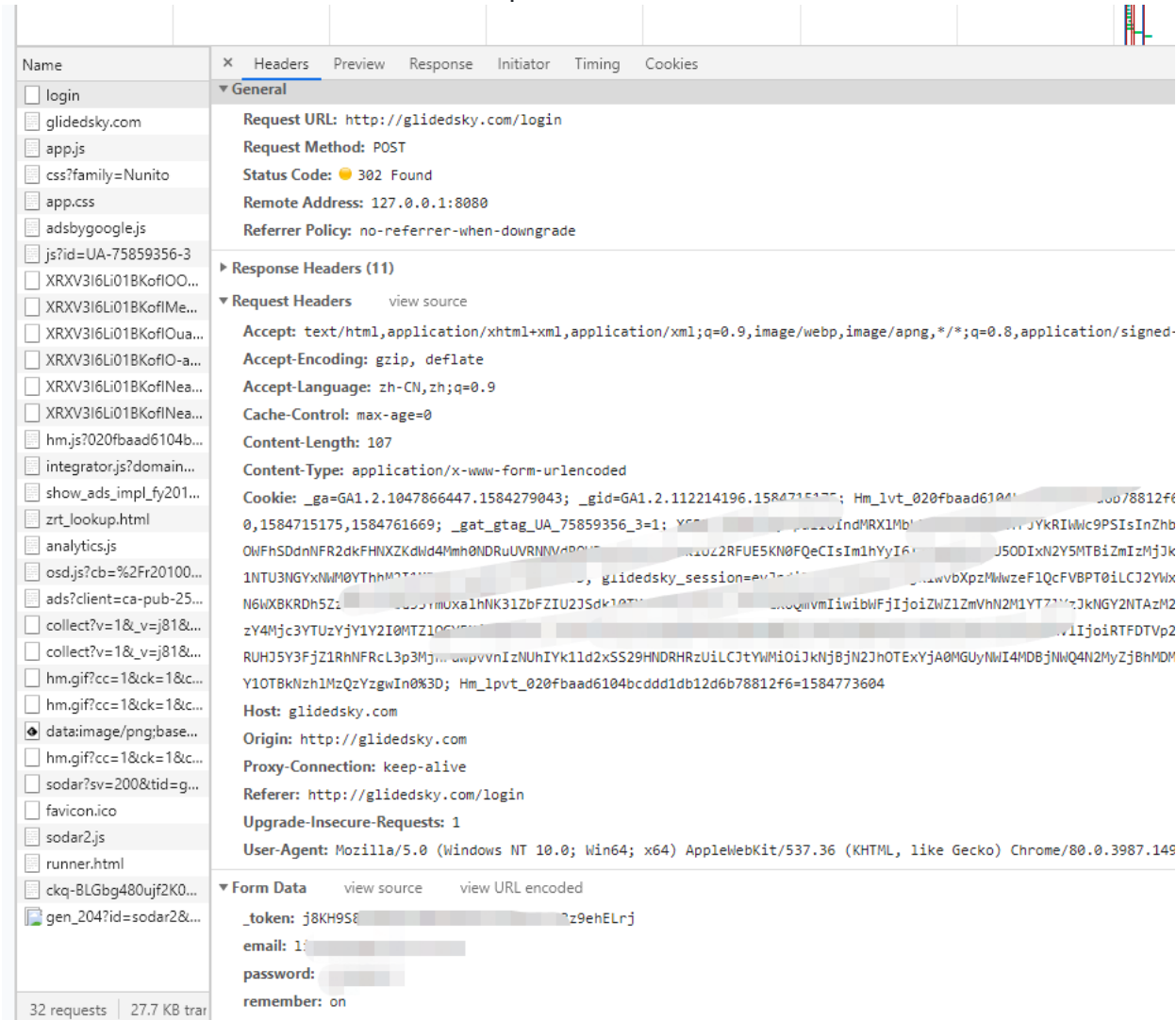
Python获取网页文本

网页访问分GET/POST/DELETE/PUT/OPTION几种不同的类型，最常用的是GET和POST请求。

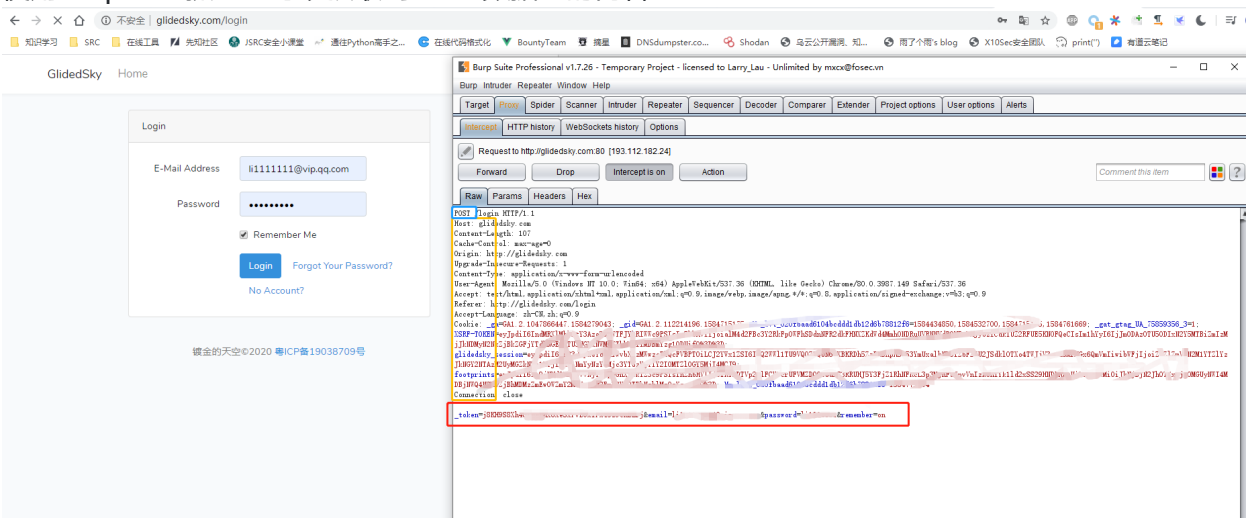
get请求的特点是所有参数及数据都显示在URL链接中，主要用于获取数据，对数据本身并不执行操作，所以我们写爬虫的过程中遇到比较多的还是get请求。

POST请求主要用于向服务器传入数据，POST请求的内容在浏览器页面上是不显示的，需要借助网页抓包工具。

chrome浏览器的f12开发者工具可以查看到post请求及其内容



使用burpsuite抓包也可以获取到POST数据包的内容



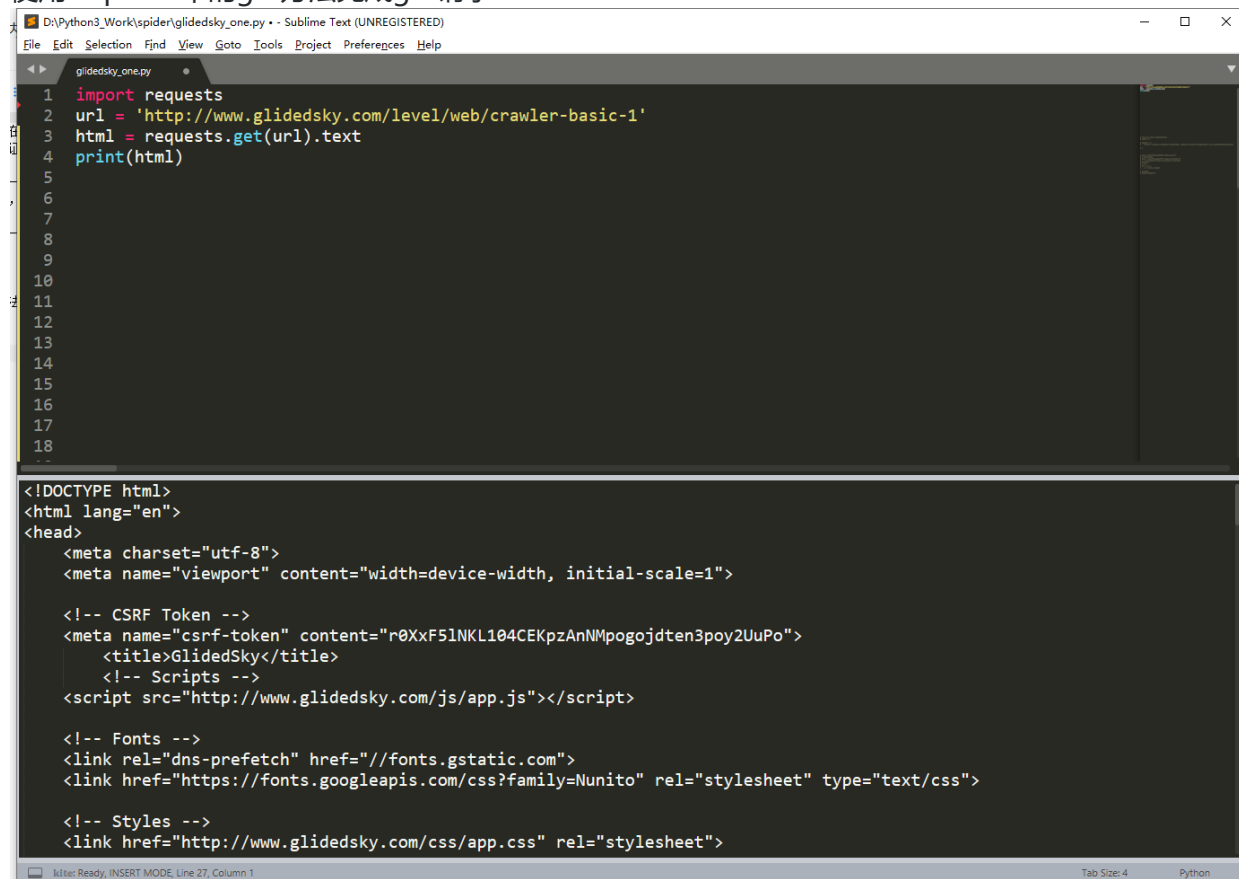
一个请求包通常由请求头和请求主体构成，在这个请求包中，除了POST表单数据之外的数据都是请求头。这里面的数据主要用于和服务器交互，验证数据来源及登录凭证。

首先就是蓝框标注的数据包类型为POST，后一个参数为处理该请求的文件名。然后黄框下有几个属性在写爬虫时需要用到，首先是User-Agent代表访问设备信息和浏览器基本信息。

Cookie，这个属性下保存的信息可以相当于一个凭证，任何人拿到你在这个网站的cookie都可以在不知道你的账号密码的情况下进入登录状态。

在Python中要实现网页访问功能有很多种方法，我比较喜欢使用requests这个库。

使用requests中的get方法完成get请求



The screenshot shows a Sublime Text editor window with a Python script named `glidedsky_one.py` and its output. The script uses the `requests` library to perform a GET request to `http://www.glidedsky.com/level/web/crawler-basic-1'` and prints the response text. The output is the HTML content of the page, including the DOCTYPE, head, meta tags, CSRF token, title, scripts, fonts, and styles.

```
1 import requests
2 url = 'http://www.glidedsky.com/level/web/crawler-basic-1'
3 html = requests.get(url).text
4 print(html)
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <!-- CSRF Token -->
  <meta name="csrf-token" content="r0XxF5lNKL104CEKpzAnNMpogojdten3poy2UuPo">
  <title>GlidedSky</title>
  <!-- Scripts -->
  <script src="http://www.glidedsky.com/js/app.js"></script>

  <!-- Fonts -->
  <link rel="dns-prefetch" href="//fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet" type="text/css">

  <!-- Styles -->
  <link href="http://www.glidedsky.com/css/app.css" rel="stylesheet">
```

注：如果报了这种错误：

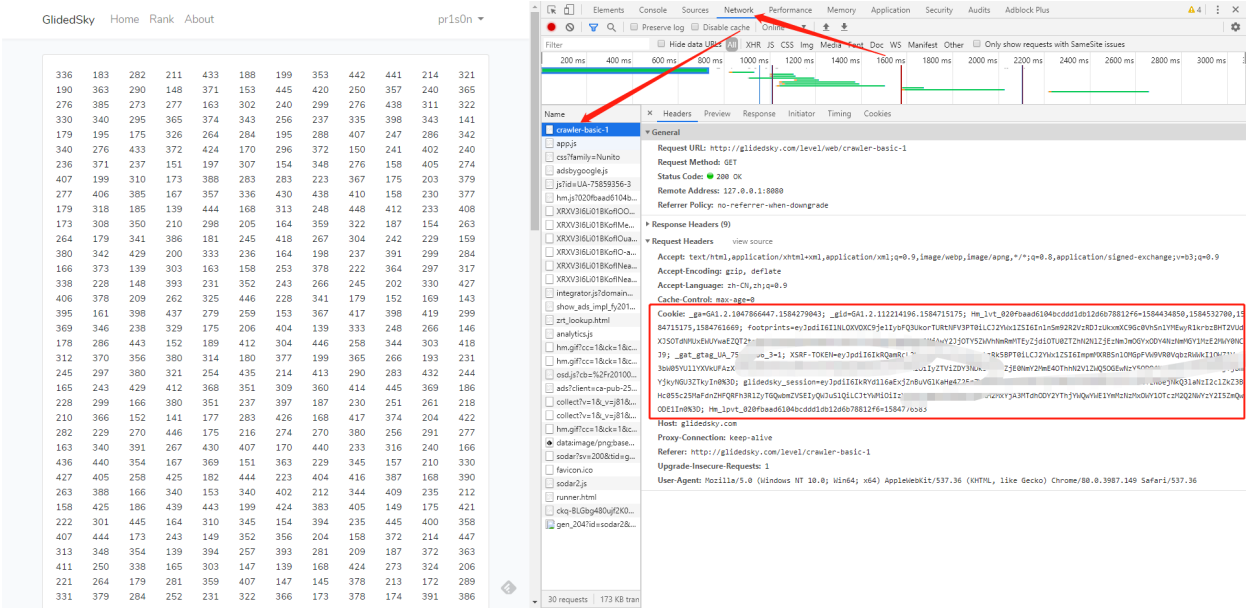
```
glidedsky_basic_two.py x glidedsky_one.py glidedsky_proxy.py x terests.py x
1 import requests
2 import resssss
   File "D:\Python3_Work\spider\glidedsky_one.py", line 2, in
3 url = 'http://www.glidedsky.com/level/web/crawler-basic-1'
4 html = requests.get(url).text
5 print(html)
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

Traceback (most recent call last):
  File "D:\Python3_Work\spider\glidedsky_one.py", line 2, in <module>
    import resssss
ModuleNotFoundError: No module named 'resssss'
[Finished in 0.3s]
```

则说明你的电脑中没有安装requests库，在命令行中使用 `pip install requests` 命令可进行安装。

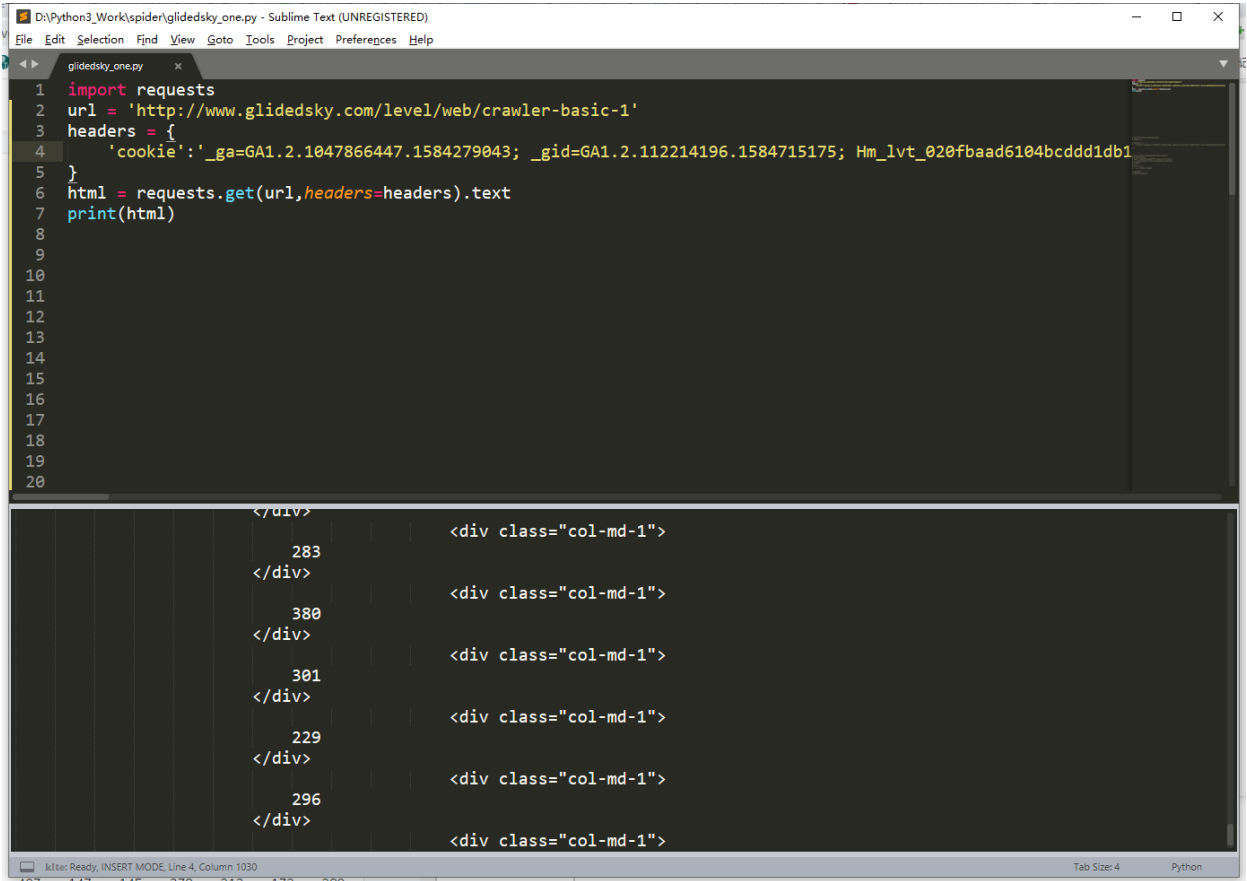
可以看到程序成功返回了网页源码，但是仔细看会发现使用Python获取到的源码中并没有包含我们想要的数

这是因为该网站做了限制，只有登录后在可以看到数据，所以在请求时必须带上cookie才可以。



按下f12，点击network网络选项，找到主要请求的网页，将cookie后的内容全部复制下来。

requests的get()方法允许我们传入一个header参数



成功获取到目标数据

处理获取到的文本数据

接下来先理下思路： 目的：提取所有数字并求和 现已完成：获取到全部文本 接下来要做的：

1. 提取所有数值
2. 将这些数值累加

确实就是这么朴实无华，想要从这么长的文本中准确获取到数值，我们需要用到两个库：BeautifulSoup和re

BeautifulSoup属于外部库，需要在命令行中使用 `pip install BeautifulSoup4` 进行安装。

引入方法：

```
from bs4 import BeautifulSoup
```

re为正则表达式支持库，Python自带，但是同样需要使用import引入

Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库。它能够通过你喜欢的转换器实现惯用的文档导航,查找,修改文档的方式。

这是官方的解释，通俗点说就是它可以帮助你更方便的提取数据。

正则表达式是用来匹配具有某种特征数据的模式，它定义了一些字符代表不同的数据。

虽然普通人真正见到正则表达式本体的机会不多，但是肯定都用过。

例如在注册一个网站时，会要求你填写邮箱和手机号，程序就是通过正则表达式首先判断你输入的邮箱和手机号格式是否正确。

一般域名的规律为“[N级域名].[三级域名].[二级域名.顶级域名]”，比如“qq.com”、“www.qq.com”、“mp.weixin.qq.com”、“12-34.com.cn”，分析可得域名类似“**.**.**.**”组成。

- “*”部分可以表示为[a-zA-Z0-9_]+
- “.”部分可以表示为\.[a-zA-Z0-9_]+
- 多个“.”可以表示为(\.[a-zA-Z0-9_]+)+

综上所述, 域名部分可以表示为 $[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)+$

最终表达式:

由于邮箱的基本格式为“名称@域名”，需要使用“^”匹配邮箱的开始部分，用“\$”匹配邮箱结束部分以保证邮箱前后不能有其他字符，所以最终邮箱的正则表达式为：

$$^{\wedge}[a-zA-Z0-9-+@]{1,64}[a-zA-Z0-9-+]{1,40}(\.[a-zA-Z0-9-+]{1,40}){0,5}\$$$

正则表达式语法同样比较复杂，但是非常有效，非常建议学习。

推荐看这篇文章了解正则表达式: [三十分钟入门正则表达式](#)

理解这两个库的作用之后我们就可以开始使用它们了。

首先使用BeautifulSoup对获取到的网页内容进行处理

```
soup = BeautifulSoup(html, 'html.parser')
```

在处理后的数据中提取出所有class=col-md-1的div的内容

```
nums = soup.find_all('div', class = 'col-md-1')
```

此是我们的数据变为：

```
[<div class="col-md-1">
    336
</div>, <div class="col-md-1">
    183
</div>, <div class="col-md-1">
    282
</div>, <div class="col-md-1">
    211
</div>, <div class="col-md-1">
    433
</div>, <div class="col-md-1">
    188
</div>, <div class="col-md-1">
    199
</div>, <div class="col-md-1">
    353
</div>, <div class="col-md-1">
    442
</div>, <div class="col-md-1">
    441
</div>, <div class="col-md-1">
    214
</div>, <div class="col-md-1">
    334
</div>]
```

注意这里我圈出的细节，这说明经过处理后的数据存储到了一个列表中。

接下来就是把除了数字之外的字符全部剔除

```
x = re.findall(r'\d\d\d',str(nums),re.DOTALL)
```

因为正则表达式是针对字符串的，所以我们需要用str方法将刚获取到的数据转化为字符串类型，findall方法的第一个参数即是匹配三位数字的正则表达式，re.DOTALL表示匹配所有字符，包括换行符等特殊符号。

re模块中的Findall方法，和BeautifulSoup模块的find_all方法完全不一样。

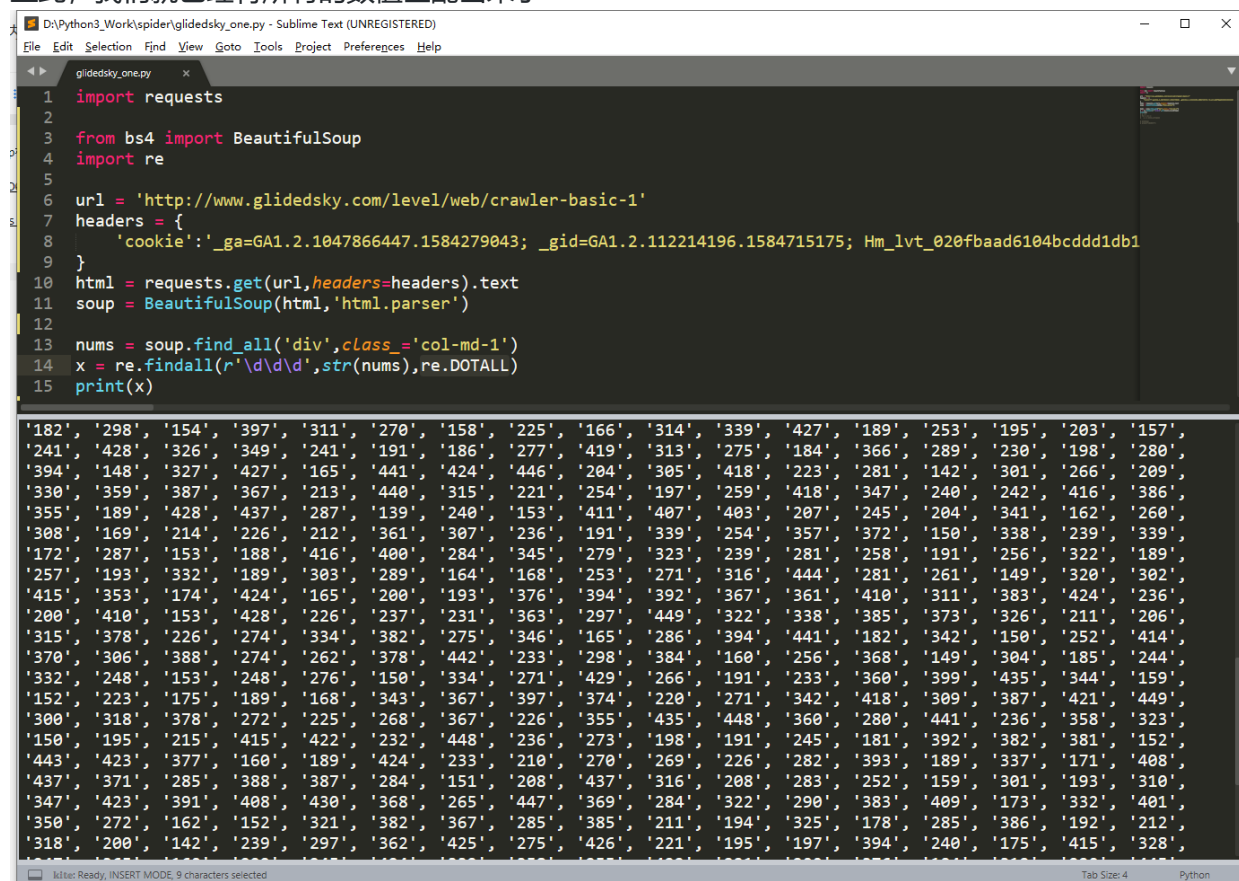
re.**findall**(*pattern, string, flags=0*)

对 *string* 返回一个不重复的 *pattern* 的匹配列表，*string* 从左到右进行扫描，匹配按找到的顺序返回。如果样式里存在一到多个组，就返回一个组合列表；就是一个元组的列表（如果样式里有超过一个组合的话）。空匹配也会包含在结果里。

在 3.7 版更改：非空匹配现在可以在前一个空匹配之后出现了。

Python re模块的[官方文档](#)中详细罗列了各个方法的使用方法和一些简单的正则表达式知识。

至此，我们就已经将所有的数值匹配出来了



The screenshot shows a Sublime Text editor window titled "glidedsky_one.py". The code in the editor is as follows:

```
1 import requests
2
3 from bs4 import BeautifulSoup
4 import re
5
6 url = 'http://www.glidedsky.com/level/web/crawler-basic-1'
7 headers = {
8     'cookie': '_ga=GA1.2.1047866447.1584279043; _gid=GA1.2.112214196.1584715175; Hm_lvt_020fbaad6104bcddd1db1'
9 }
10 html = requests.get(url, headers=headers).text
11 soup = BeautifulSoup(html, 'html.parser')
12
13 nums = soup.find_all('div', class_='col-md-1')
14 x = re.findall(r'\d\d\d\d', str(nums), re.DOTALL)
15 print(x)
```

The output of the script is displayed in the console area at the bottom of the editor. It consists of a large list of four-digit strings, each enclosed in single quotes and separated by commas. The strings are arranged in 20 rows of 10 items each. The first row starts with '182' and the last row ends with '328'.

接下来就可以使用for循环通过下标对这个列表进行遍历，然后将数据累加求和，这部分的代码不再给出。

多尝试，三行代码就能解决。

另外，其实BeautifulSoup和re任意一个模块都可以完成数据筛选工作，我没用的原因是因为我懒。

最后给出最终运行结果

```

D:\Python3_Work\spider\glidedsky_one.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

glidedsky_one.py
1 import requests
2
3 from bs4 import BeautifulSoup
4 import re
5
6 url = 'http://www.glidedsky.com/level/web/crawler-basic-1'
7 headers = {
8     'cookie': '_ga=GA1.2.1047866447.1584279043; _gid=GA1.2.112214196.1584715175; Hm_lvt_020fbaad6104bcddd1db1'
9 }
10 html = requests.get(url, headers=headers, timeout=10).text
11 soup = BeautifulSoup(html, 'html.parser')
12
13 nums = soup.find_all('div', class_='col-md-1')
14 x = re.findall(r'\d\d\d\d', str(nums), re.DOTALL)
15 print(x)

'152', '223', '175', '189', '168', '343', '367', '397', '374', '220', '271', '342', '418', '309', '387', '421', '449',
'300', '318', '378', '272', '225', '268', '367', '226', '355', '435', '448', '360', '280', '441', '236', '358', '323',
'150', '195', '215', '415', '422', '232', '448', '236', '273', '198', '191', '245', '181', '392', '382', '381', '152',
'443', '423', '377', '160', '189', '424', '233', '210', '270', '269', '226', '282', '393', '189', '337', '171', '408',
'437', '371', '285', '388', '387', '284', '151', '208', '437', '316', '208', '283', '252', '159', '301', '193', '310',
'347', '423', '391', '408', '430', '368', '265', '447', '369', '284', '322', '290', '383', '409', '173', '332', '401',
'350', '272', '162', '152', '321', '382', '367', '285', '385', '211', '194', '325', '178', '285', '386', '192', '212',
'318', '200', '142', '239', '297', '362', '425', '275', '426', '221', '195', '197', '394', '240', '175', '415', '328',
'247', '365', '160', '222', '245', '424', '332', '358', '255', '422', '221', '200', '376', '194', '312', '220', '445',
'384', '223', '286', '373', '410', '305', '307', '387', '305', '198', '234', '261', '222', '332', '144', '359', '249',
'345', '350', '354', '322', '257', '283', '407', '269', '329', '319', '187', '442', '268', '333', '228', '214', '360',
'144', '412', '231', '150', '362', '238', '381', '321', '296', '167', '445', '200', '199', '312', '427', '318', '233',
'445', '381', '367', '139', '382', '188', '166', '325', '272', '158', '164', '343', '199', '199', '225', '236', '157',
'379', '231', '329', '364', '157', '148', '382', '328', '302', '173', '208', '188', '351', '363', '308', '262', '283',
'287', '296', '186', '296', '167', '443', '239', '161', '438', '230', '195', '189', '330', '386', '360', '359', '381',
'177', '152', '230', '406', '252', '441', '184', '443', '303', '246', '369', '166', '404', '266', '218', '324', '258',
'278', '276', '449', '348', '141', '152', '153', '235', '219', '204', '268', '392', '232', '400', '303', '328', '259',
'437', '382', '253', '209', '331', '344', '329', '308', '243', '234', '296', '189', '404', '145', '284', '374', '264',
'207', '443', '416', '283', '380', '301', '229', '296', '449', '447']
采集完毕, 结果为: 319
[Finished in 0.7s]

Idle: Ready, INSERT MODE, Line 10, Column 52 - Field 1 of 2
Tab Size: 4 Python
```

爬虫:基础1 Passed

爬虫的目标很简单, 就是拿到想要的数

这里有一个网站, 里面有一些数字。把这些数字的总和, 输入到答案框里面, 即可通过本关。

提交之前要验证邮箱哦~

[待爬取网站](#)

