گزارش کار تکلیف شماره 4

در این تکلیف، با استفاده از کتابخانه Scrapy، به خزش در یک سایت خبری پرداخته ایم.

- **گروه 3:**
  - دینا امیدوار طهرانی
  - نسیم فانی
  - محسن محمودزاده
  - مریم واقعی

راه اندازی:

1. نصب : pip install Scrapy
2. ایجاد پروژه: scrapy startproject <project_name>
3. ایجاد اولین spider: scrapy genspider <spiderName> <URL>
4. مقدار allowed_domains و start_urls به ترتیب دامنه هایی که خزنده مجاز به پیمایش هست و آدرس های آغازین هست که هر کدام میتواند چندین مقدار داشته باشند و به صورت پیش فرض آدرسی است که هنگام ایجاد اسپایدر وارد کرده بودیم.
5. متد اصلی parse هست و بعد از دانلود هر یک از url های موجود در start_urls اجرا خواهد شد، همانطور که مشخص هست یکی از پارامتر های این متد response هست و همان response ای است که در shell به آن دسترسی داشتیم و میتوانیم از طریق آن و استفاده از انتخابگر ها به محتوای مورد نظرمان دست یابیم.

کلاس FactchecksSpider شامل سه تابع است که نحوه کار هر یک را شرح می‌دهیم:

1. parse

در این تابع عمل خزش صفحات انجام می‌شود و لینک‌های داخل هر صفحه (از طریق شناسایی تگ a در html) استخراج می‌شوند.

برای استخراج النمنت‌های html می‌توان از دو روش استفاده کرد: 1- با کمک xpath 2- با کمک css

در این جا ما از روش دوم استفاده کرده ایم.

لینک‌های استخراج شده از داخل صفحات در آرایه‌ی links نگهداری می‌شود. با تابع urljoin آدرس کامل لینک را بدست می‌آوریم. اگر لینک تکراری نباشد(قبلا در url_set وجود نداشته باشد) ، آن را به url_set اضافه می‌کنیم و لینک را به scrapy.Request می‌دهیم که تابع parse_fact_details را اجرا کند. این تابع در بخش 2 توضیح داده شده است.

اگر در سایت صفحه بعد (next page) داشتیم و در url_set وجود نداشت (تکراری نباشد) آن را به url_set اضافه کرده و مجددا تابع parse را فراخوانی می‌کنیم تا در آن صفحه عمل خزش و استخراج لینک‌ها انجام شود.

این عمل آن قدر تکرار می‌شود تا در نهایت به صفحه‌ی آخر برسیم.

```
def parse(self, response):
    """
    This method crawls the pages and extracts:
    1. links to the articles
    2. link to next page
    """
    links = response.css(
        "body > div.theme-content > div > div > main > div > div.media-list > article.media-wrapper > a::attr(href)"
    ).extract()
    next_page = response.css(".btn-next::attr(href)").extract()
    for link in links:
        link = response.urljoin(link)
        if link not in self.url_set:
            self.url_set.add(link)
            yield scrapy.Request(
                url=link, headers=self.headers, callback=self.parse_fact_details
            )
    if next_page:
        if next_page[0] not in self.url_set:
            self.url_set.add(next_page[0])
            yield scrapy.Request(
                url=next_page[0], headers=self.headers, callback=self.parse
            )
    pass
```

2. parse_fact_details

این تابع به طور کلی برای parse کردن صفحات مربوط به هر article (خبر) است.

در ابتدا تمام لینک هایی که در content آن صفحه وجود دارد را داخل متغیری به نام links_in_content می ریزیم. سپس در یک حلقه‌ی for بررسی می‌کند که اگر آن لینک با http://www.snopes.com/fact-check/ که url سایت مربوطه است شروع شده بود و در url_set (از ابتدا هر url ای که parse می‌کنیم را به این url_set اضافه می‌کنیم) هم نبود (یعنی تکراری نبوده است) ، آن را parse می‌کند. سپس آن را به url_set اضافه می‌کند و برای آن Request ای می‌دهد.

```
def parse_fact_details(self, response):
    """This method crawls and extracts the details of each article."""

    links_in_content = response.css("div.content:nth-child(2) a::attr(href)").extract()
    for link in links_in_content:
        if link.startswith("https://www.snopes.com/fact-check/") and (link not in self.url_set):
            self.url_set.add(link)
            yield scrapy.Request(
                url=link, headers=FactchecksSpider.headers, callback=self.parse_fact_details
            )
```

هم چنین در بعضی صفحات خبرهای سایت مربوطه بخشی به نام cliam وجود داشت، که حاوی محتوای نسبتا مهمی در مورد آن خبر بود. پس این بخش را هم با response.css(".claim > p :nth-child(1) ::text") گرفته و در متغیر آرایه ای به نام claim_list می‌ریزیم. سپس از آنجایی که بعضی از صفحات خبری cliam را ندارند، ابتدا بررسی می‌کنیم که آیا این لیست claim_list طولی دارد یا خیر(یعنی عضوی دارد یا نه) که اگر true برگرداند به معنی آن است که آن صفحه cliam دارد، پس آن را در متغیر claim می‌ریزیم.

بعد از آن به سراغ متن داخل content می‌رویم. در ابتدا کل متن div آن content مان را با تمام html ها گرفته و در لیست content_body_list می‌ریزیم. سپس از آنجایی که خبرهای صفحات آخر سایت snope که مدت زیادی از publish آن ها گذشته است حاوی content ای نیستند، بررسی می‌کنیم که آیا این content_body_list طولی دارد یا خیر که اگر true برگرداند، content_body_list[0] که دارای content ما است را داخل متغیر content_body می‌ریزیم(پس content_body اکنون یک string است که tag ها html داخلش هستند). حال با استفاده از تابع remove_html_tags که در ادامه آمده است، از regex ها استفاده کرده و tag ها را حذف می‌کنیم.

```
claim_list = response.css(".claim > p:nth-child(1) ::text").extract()
claim = ''
if len(claim_list):
    claim = claim_list[0]

content = ''
content_body_list = response.css("div.content:nth-child(2)").extract()
if len(content_body_list):
    content_body = content_body_list[0]
    content = self.remove_html_tags(content_body)
```

در این سایت، بخش زیادی از جزئیات مورد نظر ما درون شیء snopesPageData قرار گرفته است. به همین منظور یک الگو (pattern) برای استخراج این شئ نوشته‌ایم و با کمک css selector از داخل script اولین جایی که با این الگو match می‌شود را پیدا می‌کنیم. String بدست آمده را با کمک کتابخانه‌ی json که در ابتدا import شده است، به json تبدیل می‌کنیم.

فایل items.py یک کلاس به نام FactCheckItem دارد که در آن می‌توان برای item فیلدهای مختلفی تعریف کرد و آن‌ها را مقداردهی نمود.. یک شئ از FactCheckItem به نام item ایجاد می‌کنیم و با کمک چند if اطلاعات مورد نظرمان را استخراج کرده و در item مقداردهی می‌کنیم.

مثالی از snopesPageData در یکی از articleها :

```
var snopesPageData = {"url":"https:\/\/www.snopes.com\/fact-check\/best-bathroom-in-
america\/","template":"single","content-type":"Fact
Check","post_id":"285253","author":"65","author_name":"Madison
Dapcevich","category":"Entertainment","rating":"True","title":"With a price tag this high,
Bancroft Park flushed away its competition. ","date_published":"22 November
2020","date_updated":""};
```

```
    pattern = r'\bvar\s+snopesPageData\s*=\s*(\{.*?\})\s*;\s*\n'
    json_data = response.css("script::text").re_first(pattern)
    json_res = json.loads(json_data)
    title_list = response.css("h1.title::text").extract()
    title = title_list[0]
    item = FactCheckItem()

    if title:
        item["title"] = title
    if "url" in json_res:
        item["url"] = json_res["url"]
    if "date_published" in json_res:
        item["date_published"] = json_res["date_published"]
    if "rating" in json_res:
        item["rating"] = json_res["rating"]
    if "author_name" in json_res:
        item["author_name"] = json_res["author_name"]
    if "category" in json_res:
        item["category"] = json_res["category"]
    if "tags" in json_res:
        item["tags"] = json_res["tags"]
    item["claim"] = claim
    item["content"] = content
    yield item
    pass
```

```python
class FactCheckItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    title = scrapy.Field()
    url = scrapy.Field()
    content = scrapy.Field()
    date_published = scrapy.Field()
    rating = scrapy.Field()
    author_name = scrapy.Field()
    category = scrapy.Field()
    claim = scrapy.Field()
    tags = scrapy.Field()
    pass
```

3. remove_html_tags

در این متد ما از regex ها استفاده کردیم. اولین regex، مربوط به تگ script است. ما می‌خواهیم که هم خود تگ script و هم محتویات داخلش به طور کلی حذف شوند پس از clean_script = re.compile('<script[^>]*>[\s\S]*?</script>') استفاده می‌کنیم. هم چنین زیر نویس عکس ها را نیز در اینجا با استفاده از دستوری مشابه قبل حذف می‌کنیم(خود tag ها با تمام محتویات آن حذف می‌شود). تگiframe به همراه تمام محتویات آن مروبط به فیلم ها است ، پس آن را هم تماما حذف میکنیم. در آخر هم یک regex برای clean_all_tags داریم که برای بقیه tag هایی که به جا مانده است و می‌خواهیم فقط tag آن ها حذف شود و نه محتویاتش از آن استفاده می‌کنیم(شامل p و ...) و همچنین حذف تگ‌هایی مانند img که محتوای آن نیازی نداریم.

```python
def remove_html_tags(self, text):
    """This method removes HTML tags from a string"""
    clean_script = re.compile('<script[^>]*>[\s\S]*?</script>')
    clean_image_caption = re.compile('<figcaption[^>]*>[\s\S]*?</figcaption>')
    clean_iframe = re.compile('<iframe[^>]*>[\s\S]*?</iframe>')
    clean_all_tags = re.compile('<.*?>')
```

پس ابتدا clean_script را برروی text اعمال کرده و بعد clean_image_caption و بعد هم clean_iframe را. سپس \n و \t هایی که در متن وجود داشت و در هنگام ایجاد content باعث ایجاد خط های خالی زیاد و ... در content شده بود را حذف می‌کنیم(در هنگام حذف \n ، از آنجایی که ممکن است با حذف خط های خالی ، بین دو کلمه‌ی جدا ، فاصله را حذف کنیم و باعث چسبیدن آن ها بهم شویم، پس یک space می‌گذاریم تا مانع آن شویم).

\xa0 را هم که مربوط به non_breaking space است را با space جایگزین می‌کنیم. در آخر هم این text را بر می گردانیم.

```
    # remove script tag with its content
    text = re.sub(clean_script, '', text)
    # remove caption of an image with its content
    text = re.sub(clean_image_caption, '', text)
    text = re.sub(clean_iframe, '', text)
    text = re.sub("[\n]+", " ", text)
    text = re.sub("[\t]+", "", text)
    # \xa0 is actually non-breaking space in Latin1 (ISO 8859-1), also chr(160).
    # You should replace it with a space.
    text = re.sub("[\xa0]+", " ", text)
    return re.sub(clean_all_tags, '', text)
```

4. گرفتن خروجی

در آخر هم برای اجرا کد و گرفتن خروجی و ذخیره‌ی آن در محیط ترمینال با استفاده از scrapy crawl command ، فایل FactChecks -o output.json را ایجاد می‌کنیم. این فایل output که فایل خام خروجی است و در آن تمام خروجی های مورد نیاز در مورد هر صفحه خبری را در یک خط نوشته شده است را برای اینکه خروجی خواناتری داشته باشیم در main ابتدا باز کرده و data ی آن را می‌خوانیم و سپس بااستفاده از متد dumps کتابخانه json ، data ی همین فایل output را به آن داده و به آن فرم و شکل مرتب تری می‌دهیم. سپس آن را در فایل خروجی prettified_output.json می ریزیم.

```
file = open('./snopes/output.json', mode='r')
data = json.load(file)
dest_data = json.dumps(data, indent=1)
dest_file = open('prettified_output.json', 'w')
dest_file.write(dest_data)
file.close()
dest_file.close()
```

نمونه کد خروجی خام : (اینترها صرفا جهت نمایش بهتر در داکیومنت زده شده‌اند و در فایل اصلی اطلاعات هر لینک در یک خط نوشته شده است)


```
{"title": "Officer Pops Open Woman\u2019s Trunk, Lets Her Go After What He Sees?", "url": "https://www.snopes.com/fact-check/officer-pops-open-womans-trunk-lets-her-go-after-what-he-sees/", "date_published": "22 November 2020", "rating": "Mostly False", "author_name": "David Mikkelson", "category": "Viral Phenomena", "claim": "A police officer stopped a driver and then let her go after finding something shocking or surprising in her trunk.", "content": " Many different clickbait and arbitrage websites have published articles about a seemingly recent incident in which a motorist was pulled over by a police officer and then let go after the officer \u2014 who was being furtively recording by the driver \u2014 popped her trunk and found something alarming or surprising there:
The implication of all these misleading and incomplete headlines was that the officer found something so shocking or unusual in the trunk of the woman\u2019s car (an explosive device? biohazardous material?) that he let the driver go for his own safety (without even calling for backup), or that the motorist recorded him engaging in some unlawful act (such as stealing something from her trunk) and thus he didn\u2019t dare cite her.         In fact, the underlying incident described in these articles took place back in 2017, and it was not shocking at all but rather mundane (although somewhat heartwarming). As the driver in this tale, Chy-Niece Thacker posted on her Facebook page, while on her way to a job interview she was stopped by Officer Jenkins of the Henrico County Police because the brake lights in her car were not functioning. Thacker was upset at this circumstance because, she said, she\u2019d just had the brake lights on that vehicle replaced a month earlier. Jenkins asked Thacker to pop the trunk so he could inspect the brake lights, then instructed her to \u201cpop the hood to check the relay box\u201d:     After spending about half an hour tinkering with her car to see if he could get the lights working again, Jenkins had Thacker turn on her vehicle\u2019s hazard lights and then escorted her to the closet mechanic to get them repaired. Although he could have done so, Jenkins declined to cite Thacker for non-functioning brake lights, telling her that \u201che cared more about her safety than giving her a ticket\u201d: \u201cWe \u2019ve seen so many stories where a traffic stop has turned into the death [of an officer], and it\u2019s given people a bad taste in their mouth. I think this one officer helping can set a trend,\u201d Thacker said. Contrary to the implications of clickbait headlines, the police officer in this story didn\u2019t let a motorist off because of what he found in the trunk of her car, nor because his interaction with her had been recorded. In fact, the officer in this incident was not recorded in either an audio or a video sense \u2014 the driver merely took some photographs of him as he attempted to repair the brake lights on her vehicle. "},
```


نمونه کد خروجی prettified :


```
{
 "title": "Officer Pops Open Woman\u2019s Trunk, Lets Her Go After What He Sees?",
 "url": "https://www.snopes.com/fact-check/officer-pops-open-womans-trunk-lets-her-go-after-what-he-sees/",
 "date_published": "22 November 2020",
 "rating": "Mostly False",
 "author_name": "David Mikkelson",
 "category": "Viral Phenomena",
 "claim": "A police officer stopped a driver and then let her go after finding something shocking or surprising in her trunk.",
 "content": " Many different clickbait and arbitrage websites have published articles about a seemingly recent incident in which a mo
},
```


5. بخش های نمره اضافه انجام شده

- نام نویسنده خبر(author_name)
- موضوع خبر(category)
- tags
- حل خطای 403 : در طی عمل خزش پس از مدتی از سمت سرور خطای 403 دریافت می‌شد و عمل خزش بعد از حدود 90 صفحه متوقف می‌شد، که این به دلیل آن بود که درخواست های زیادی در لحظه توسط یک user agent داده می‌شد. برای حل این مشکل 2 راه حل را بررسی کرده که راه حل دوم نتیجه بخش بود.

  راه اول: روش این راه استفاده از random user agent بود که در آن یک لیست شامل تعداد زیادی user agent در نظر گرفته و هربار به صورت رندوم توسط یکی از این user agent ها درخواست می‌دادیم. این روش باعث شد تا تعداد صفحات بیشتری نسبت به قبل(تا 413 صفحه) خزش شوند ولی برای سایت snope که دارای 1280 صفحه بود و پیوسته بر تعداد آن هم اضافه می‌شد کارساز نبود.

  راه دوم: پس از راه دوم که ایجاد یک delay به اندازه‌ی 0.25 ثانیه بین هر درخواست بود استفاده کردیم.DOWNLOAD_DELAY بصورت پیش فرض در settings برابر 0 است (یعنی هر pageای به آن می‌رسید سریع

برای آن ریکوست می‌داد و این باعث می‌شد تا تعداد ریکوست ها به صورت نمایی رشد کند ولی پس از ایجاد 0.25 ثانیه تاخیر بین هر درخواست، این مشکل برطرف شد و تمام صفحات را خزش کرد)

- بخش claim : در بعضی صفحات خبرهای سایت مربوطه بخشی به نام cliam وجود داشت، که حاوی محتوای نسبتا مهمی در مورد آن خبر بود. این بخش را با response.css(".claim > p :nth-child(1) ::text") گرفته و در متغیر آرایه ای به نام claim_list می‌ریزیم. (بعضی از صفحات خبری cliam را ندارند)
- ذخیره خروجی تولید شده توسط خزنده با استفاده از دستور scrapy crawl FactChecks -o output.json