웹 스크래핑 - request 라이브

requests 라이브러리

- `requests`는 파이썬에서 HTTP 요청을 쉽게 보낼 수 있게 해주는 라이브러리
- GET, POST, PUT, DELETE 등의 HTTP 메소드를 간단하게 실행 가능
- **requests 라이브러리를 사용하여 실제 운영 중인 웹 사이트에서 웹 스크래핑 진행할 것**

requests

Python HTTP for Humans.

https://pypi.org/project/requests/

- 공식 문서

Requests: HTTP for Humans™ — Requests 2.31.0 documentation

Requests is an elegant and simple HTTP library for Python, built for human beings.

https://requests.readthedocs.io/en/latest/

- 설치

```
pip install requests
```

URL에 파라미터 전달하기

- Requests를 사용하면 이 인자들을 문자열의 딕셔너리로 제공할 수 있으며, params 키워드 인자를 사용
- 예를 들어, httpbin.org/get에 key1=value1 및 key2=value2를 전달하고자 한다면 다음 코드를 사용

```
payload = {'key1': 'value1', 'key2': 'value2'} r = requests.get('https://h
ttpbin.org/get', params=payload)
```

- URL이 올바르게 인코딩되었는지 URL을 출력하여 확인

```
print(r.url) # https://httpbin.org/get?key2=value2&key1=value1
```

- 값으로 리스트를 전달 가능
- 리스트를 사용하면 한 키에 여러 값을 효율적으로 전달할 수 있으며, 서버 측에서는 이를 배열이나 리스트로 인식하여 처리 가능

```
payload = {'key1': 'value1', 'key2': ['value2', 'value3']} r = requests.ge
t('https://httpbin.org/get', params=payload) print(r.url) # https://httpbi
n.org/get?key1=value1&key2=value2&key2=value3
```

Response Content

- 서버의 응답 내용 읽기 가능
- GitHub 타임라인 읽기
- Requests는 서버에서 오는 내용을 자동으로 디코딩

```
import requests r = requests.get('https://api.github.com/events') r.text #
{"repository":{"open_issues":0,"url":"https://github.com/...
```

r.content

- `r.content`는 서버로부터 받은 응답의 본문을 바이트(byte) 형태로 제공합니다. 이 속성은 이미지나 PDF 파일 같은 이진 데이터를 다룰 때 주로 사용됩니다.
- 바이트 형태로 데이터를 받기 때문에, 텍스트 데이터에 대해서는 인코딩을 수동으로 처리해야 할 수도 있습니다.

r.text

- `r.text`는 `r.content`를 기반으로 하며, Requests가 자동으로 인코딩을 처리하여 문자열 형태로 변환한 응답의 본문 제공
- r.text 속성은 텍스트 기반의 데이터(예: HTML, JSON)를 다룰 때 유용
- Requests는 HTTP 헤더에서 `Content-Type`을 보고 인코딩을 추측하여 적절히 디코딩

JSON Response Content

- JSON 데이터를 다룰 경우를 위한 내장 JSON 디코더 존재
- JSON 디코딩이 실패할 경우, `r.json()`은 예외 발생

```
import requests r = requests.get('https://api.github.com/events') r.json()
# [{'repository': {'open_issues': 0, 'url': 'https://github.com/...
```

r.json() & r.text 비교

- `r.json()` 메소드는 JSON 형식의 응답을 파이썬의 딕셔너리로 자동 변환
- 이 메소드는 JSON 데이터를 파싱하여 `키-값 쌍의 구조`를 가진 파이썬 객체로 만들어 주기 때문에, 딕셔너리의 키를 이용하여 직접 접근 가능

```
payload = {'key1': 'value1', 'key2': 'value2'} r = requests.get('https://h
ttpbin.org/get', params=payload) print(r.json()['args']) # 출력: {'key1':
'value1', 'key2': 'value2'}
```

- `r.text`는 응답을 일반 텍스트 형태로 제공
- 이 텍스트는 JSON 형식의 문자열일 수 있지만, 파이썬의 문자열 객체이므로 JSON 객체처럼 키로 직접 접근할 수 없음

```
payload = {'key1': 'value1', 'key2': 'value2'} r = requests.get('https://h
ttpbin.org/get', params=payload) print(r.text['args']) # TypeError: string
indices must be integers, not 'str'
```

- 위 코드에서 `r.text['args']`는 `TypeError`를 발생시킵니다. 문자열 인덱싱은 정수를 사용해야 하며, 문자열을 딕셔너리처럼 키로 접근할 수 없기 때문
- 문자열 데이터를 딕셔너리로 사용하려면 먼저 JSON 파싱 과정을 거쳐야 합니다. 이를 위해서는 `json.loads(r.text)`를 사용하여 문자열을 파이썬 객체로 변환해야 합니다.

결론

- `r.json()`은 JSON 형식의 응답을 쉽게 딕셔너리로 변환하고 바로 사용할 수 있게 해 주는 반면, `r.text`는 단순한 문자열 데이터를 반환하므로 JSON 데이터를 처리하려면 추가적인 파싱 과정이 필요
- 따라서 JSON 형식의 데이터를 다루는 경우 `r.json()`을 사용하는 것이 훨씬 편리하고 직관적

Custom Headers

- 요청에 HTTP 헤더를 추가하고 싶다면, 단순히 딕셔너리를 `headers` 파라미터로 전달
- 예를 들어, 이전 예제에서는 user-agent를 지정하지 않음

```
url = 'https://api.github.com/some/endpoint' headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.6367.118 Safari/537.36'} r = requests.get(url, headers=headers) # 왼쪽 headers는 requests.get() 함수에 대한 파라미터 이름 # 오른쪽 headers는 사용자가 정의한 변수 이름
```

User-Agent 헤더

User-Agent

- 요청을 보내는 클라이언트(브라우저나 다른 도구)의 정보 식별
- 웹 서버는 이 정보를 사용하여 다양한 브라우저나 클라이언트에 맞게 최적화된 콘텐츠를 제공
- 웹 스크래핑을 할 때 `User-Agent`를 설정하면, 요청이 실제 사용자의 브라우저에서 발생한 것처럼 보이게 할 수 있어, 접근 제한이나 차단을 피할 확률을 높일 수 있음
- 참고로, 네이버는 엄격해서 헤더 정보가 반드시 필요

select를 활용하여 zdnet의 주요 정보 가져오기

```
import requests from bs4 import BeautifulSoup url = "https://zdnet.co.kr/" header_info = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36'} r = requests.get(url, headers=header_info, verify=False) soup = BeautifulSoup(r.text, 'html.parser') links = soup.select("body > div.contentWrapper > div > div.left_cont > div.news1_box > div.news_list > div > div.assetText > a > h4")
```

네이버 지식IN의 결과 값을 가져오기

```
import requests from bs4 import BeautifulSoup keyword = input("키워드 입력: ") url = f"https://kin.naver.com/search/list.naver?query={keyword}" #https://kin.naver.com/search/list.naver?query=%ED%8C%8C%EC%9D%B4%EC%8D%AC header_info = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36'} r = requests.get(url, headers=header_info, verify=False) soup = BeautifulSoup(r.text, 'html.parser') titles = soup.select("#s_content > div.section > ul > li > dl > dt > a") dates = soup.select("#s_content > div.section > ul > li > dl > dd.txt_inline") for title, date in zip(titles, dates): print(f"질문 : {title.text}") print(f"날짜: {date.text}")
```

9시 25분까지!! 진행

https://www.malware-traffic-analysis.net/2023/index.html 를 크롤링 해서 1. 제목 주소를 가져오시오. 2. 링크 정보를 전체 **URL** 형식으로 출력하세요. (https://www.malware-traffic-analysis.net 로 시작) 3. 결과 값을 txt 파일로 저장하세요.!! (후에 추가) 4. 결과 txt 파일을 하루에 한번씩 메일로 보낸다.

```
import requests from bs4 import BeautifulSoup url = "https://www.malware-traffic-analysis.net/2023/index.html" header_info = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36'} r = requests.get(url, headers=header_info, verify=False) soup = BeautifulSoup(r.text, 'html.parser') tags = soup.select("#main_content > div.content > ul > li > a.main_menu") results = [] for tag in tags: link_text = tag.text link_href = f"https://www.malware-traffic-analysis.net/2023/{tag.get('href')}" results.append(f"{link_text}\n{link_href}\n") print(results) with open('malwares.txt', 'w', encoding='utf-8') as file: for result in results: file.write(result)
```

```
#pip install feedparser pandas import feedparser import pandas as pd url =
"https://www.dailysecu.com/rss/allArticle.xml" feed =
feedparser.parse(url) titles = [] links = [] descriptions = [] authors =
[] pubDates = [] for entry in feed.entries: titles.append(entry.title)
links.append(entry.link) descriptions.append(entry.description)
authors.append(entry.author) pubDates.append(entry.published) data =
{'Title': titles, "Link": links, "Description": descriptions, "Author":
authors, "PubDate":pubDates} df = pd.DataFrame(data)
df.to_excel('dailysecu.xlsx', index=False) print("파일이 생성")
```