I. 각 항목별 주 사용 목적 정리

1. request(라이브러리): 정보(웹페이지)를 요청&가져오기
2. BeaurifulSoup(라이브러리): HTML을 파싱해서 태그별로 접근하기. 정보를 찾고 추출하기
3. json(내장 모듈): 키-값 형식의 데이터 형식으로, 웹과 앱이 데이터를 주고받을 때 표준처럼 쓰이는 형식
4. Selenium(라이브러리): 사람이 웹 브라우저에서 하는 동작을, 코드로 자동으로 시키는 도구
5. 크롬개발자 도구: 수집할 페이지를 분석하는데 사용. 웹 페이지의 HTML, CSS, JavaScript 코드를 검사하고 수정할 수 있으며, 네트워크 요청 응답 내용 분석, 성능 분석, 콘솔 로그 등 다양한 기능을 제공

- 위의 라이브러리(패키지)는 별도 pip 설치 필요

II. request 모듈

1. 개요

> 웹 페이지에 `요청(request)`을 보내서, `응답(response)`을 받아오는 파이썬 라이브러리

- 웹 사이트 주소(URL)에 접속해서
- HTML 문서, 이미지, JSON 등의 내용을 코드로 받아옴
- get/post 요청 방식을 모두 지원하며 요청시 설정해야 하는 헤더정보, 쿠키정보 설정등 HTTP요청을 위한 모든 기능을 지원한다.

> HTTP 요청 방식(HTTP Method)
>
> - HTTP 프로토콜은 클라이언트가 서버에 요청하는 목적에 따라 다음과 같은 방식을 정의한다.
>   - HTTP 통신규약: WEB 프로토콜. 즉 웹브라우저는 HTTP 규약에 맞춰 만들어진 것이다.
>   - GET, POST, PUT, PATCH, DELETE, HEADER, OPTIONS, TRACE, CONECT 등이 있다.
>   - GET: 기본 요청방식으로 서버가 가진 데이터를 `요청`한다. (RETRIEVE)
>   - POST: 클라이언트의 데이터를 서버에 `전송(저장)`한다. (CREATE)

- 설치
  - `pip install requests`

2. requests 코딩 패턴

0. 통상의 인터넷 브라우저 절차

- url를 적어서 엔터치면 서버로 해당 화면을 요청하는 서류를 보낸다.
- 서류를 받은 서버는 그에맞는 서류를 보내 응답한다
- 서류를 받으면 그걸 토대로 화면에 뿌린다.
- 이런 과정을 request로 진행함
- 정확히는 문서를 가져오는 것 까지 request가 하고, 문서를 보여주고 그 안에서 뭔가를 찾기 시작하는것이 BeautifulSoup

1. requests의 get()/post() 함수를 이용해 url을 넣어 서버 요청한다.

2. request 모듈은 요청한 문서를 제공하는 것까지하고 우리는 res라는 변수에 담는다.

3. 응답받은 내용(일반적으로 HTML 페이지)을 처리.

```
- text(HTML)은 BeautifulSoup를 이용해 원하는 내용을 추출한다.
- binary 파일의 경우 파일출력을 이용해 local에 저장
```

3. 요청 함수

- HTTP 요청 방식에 따라 두개 함수를 사용.

(1) get() vs post()

구분	GET 방식	POST 방식
목적	클라이언트가 **서버의 자원을 요청** (데이터 받아오기)	클라이언트가 **자신의 데이터를 서버로 전송**
동작 방식	URL에 파라미터를 붙여서 요청 (`?q=value`)	데이터는 요청 **본문(body)**에 담겨 전송
사용 예시	검색, 기사 열람, 목록 보기 등	로그인, 회원가입, 설문 제출 등
브라우저 동작	**주소창에 직접 입력**하거나, **링크 클릭**으로도 가능	폼(form) 제출 또는 `requests.post()` 같은 코드로 수행
보안성	URL에 노출 → 상대적으로 보안 낮음	URL에 노출되지 않음 → **개인정보 전송에 적합**
캐싱 여부	✅ 브라우저가 캐싱함	❌ 기본적으로 캐싱되지 않음

- 캐싱? 한 번 받아온 데이터를 임시로 저장해두었다가, 다음에 다시 요청할 때 더 빠르게 가져오는 기술

(2) 각 방식의 주요 매개변수

- 추가적인 정보가 필요하면 이하의 매개변수를 넣어 요청한다.

1) GET 방식 요청

- **주요 매개변수**
  - params: 요청파라미터를 dictionary로 전달
  - headers: HTTP 요청 header를 dictionary로 전달
    - 'User-Agent': 요청시 요청의 주체를 알리며 요청함
    - 'Referer': 요청시 이전에 무슨 페이지에서 왔는지 알리며 요청함
    - 크롤링을 하기 위해 필요한 header 정보는 웹브라우저의 개발자 도구를 이용해 확인한다. (Network 탭에서 확인)
    - 사이트 쪽에서는 User-Agent 또는 Referer를 설정하여 크롤링을 막기도한다.
  - cookies: 쿠키정보를 전달
- **반환값(Return Value)**
  - Response: 응답결과

2) POST 방식 요청

- **주요 매개변수**
  - datas : 요청파라미터를 dictionary로 전달
  - files : 업로드할 파일을 dictionary로 전달
    - key: 이름, value: 파일과 연결된 InputStream(TextIOWrapper) # key의 "이름"은 정해진대로 적어야한다.
    - 즉. 해당 부분을 검색했을때 name="XXXXX" 부분에서 XXXXX를 그대로 보내야함
  - headers: HTTP 요청 header를 dictionary로 전달
    - 'User-Agent', 'Referer' 등 헤더 설정
  - cookies: 쿠키정보를 전달
- **반환값(Return Value)**
  - Response: 응답결과

(3) 요청 파라미터(Request Parameter)

1) 개념

- 서버가 일하기 위해 클라이언트로 부터 받아야 하는 값들
- 클라이언트가 서버에게 일 시킬때 추가적인 정보등을 주어 원하는 결과를 얻고자 하는것
- `name=value` 형식으로 클라이언트가 전달한다. 여러개일 경우 `&`로 연결해서 전송됨 (ex: page=1&keyword=test)

- ex. 검색엔진에 검색시, 원하는 키워드를 입력하여 같이 보냄. 로그인시 id,pw를 입력하여

2) GET 요청 시

1)내용

- **URL 뒤에 붙는 쿼리스트링(Query String)** 을 통해 전달됨
- querystring: URL 뒤에 붙여서 전송하는 요청파라미터를 말한다.

2. 형식
   i) URL 뒤에 ?를 붙이고 그 뒤에 요청파라미터를 붙여 구성한다.
   ii) &를 기준으로 나눠보면 =으로 짝지어져있다. name=value
   (`?` 가 url과 요청파라미터를 구분하는 구분자로 사용된다.)

> ```
> https://search.naver.com/search.naver?query=python&page=1
> ```

3. requests.get() 요청시 요청파라미터 전달하는 두가지 방법
   i) URL 뒤에 직접 붙이기
   `→ url?name=value&name2=value2`
   ii) params 매개변수에 딕셔너리로 전달

> ```
> requests.get(url, params={"query": "python", "page": 1})
> ```

3) POST 요청 시

1. 내용

- Post 요청시 요청정보의 body에 넣어 전달
- requests.post() 요청시에는 dictionary로 구성한 뒤 매개변수 datas에 전달한다.
- 입력한 ID, PW등이 url에 적어서 전달되면 안되므로, get과는 다른 방식으로 전달하는 것(url에 노출되지 않음)

2. 전달 방법

```
requests.post(url, data={"id": "abc", "pw": "1234"})
```

(4) HTTP 요청 헤더

클라이언트(나)와 서버(웹사이트) 가 서로 요청하고 응답할 때 주고받는 부가 정보
말하자면, "이 요청은 누가, 어떻게, 어떤 걸 원해서 보냈는지"를 담은 명함 같은 메타정보

> HTTP 요청시 웹브라우저가 client의 여러 부가적인 정보들을 Key-Value 쌍 형식으로 전달한다.
>
> - accept: 클라이언트가 처리가능한 content 타입 (Mime-type 형식으로 전달)

- accept-language: 클라이언트가 지원하는 언어(ex: ko, en-US)
- host: 요청한 host
- user-agent: 웹브라우저 종류

4. Response객체 - 응답데이터

- get()/post() 의 요청에 대한 서버의 응답 결과를 Response 클래스의 객체에 담아 반환한다.
  - Response객체의 속성(attribute)들을 이용해 서버가 응답결과를 조회할 수있다.
- **주요 속성(Attribute)**
  - **url** - 응답한 서버의 url
  - **status_code** - HTTP 응답 상태코드
  - **headers** - 응답 header 정보로 dictionary로 반환한다.
  - **응답 결과 데이터 조회**
    - **text** - 응답내용(html을 str로 반환)
    - **content** - 응답내용(응답결과가 binary-image, 동영상등-일 경우 사용하며 bytes 타입으로 반환한다.)
    - **json()** - 응답 결과가 JSON 인 경우 dictionary로 변환해서 반환한다.

> ### Python json 모듈
>
> JSON 형식 문자열을 다루는 파이썬 표준 모듈
> JSON? key-value 형태 또는 배열 형태의 text(dict꼴의 str)
>
> - json.loads(json문자열)
>   - JSON 형식 문자열을 dictionary로 변환
> - json.dumps(dictionary)
>   - dictionary를 JSON 형식 문자열로 변환

> ### HTTP 응답 상태코드
>
> 서버의 응답 결과를 나타내는 세 자리 숫자 코드이다. 이 코드를 통해 요청이 성공적으로 처리되었는지, 오류가 발생했는지, 아니면 다른 조치가 필요한지 등을 클라이언트에게 알려준다.
> 응답한 결과에 대해 제대로된 응답이 진행된것 인지 판단하는 번호
> 크롤링시에는, 200번대가 성공인 것만 알면 된다.

III. BeautifulSoup 모듈

1. 개요

> Markup 언어 parsing 라이브러리

- HTML이나 XML 문서 내에서 원하는 정보를 가져오기 위한 파이썬 라이브러리. 페이지를 갖고오는게 아니라 가져온 문서에서 내가 원하는 정보를 갖다줌
- 그럼 가져오는것은? request
- 설치
  - beautifulsoup4 설치
    `pip install beautifulsoup4`
  - lxml 설치(html/xml parser)
    `pip install lxml`

(1) 코딩 패턴

> 1. 조회할 HTML내용을 전달하여 BeautifulSoup 객체 생성
> 2. BeautifulSoup 객체의 메소드들을 이용해 문서내에서 필요한 정보 조회
>   - 태그이름과 태그 속성으로 조회
>   - css selector를 이용해 조회
>   - . 표기법을 이용한 탐색(Tree 구조 순서대로 탐색)

(2) 파싱(Parsing)이란?

무언가를 읽고, 분석해서 구조화하는 과정
특히, **문자열 데이터(HTML, JSON, 코드 등)** 를 **컴퓨터가 이해할 수 있는 구조(트리, 딕셔너리 등)** 로 바꾸는 과정을 말함

예를 들어:

```
<h1>Hello</h1>
```

→ 이걸 그냥 문자열로 보면:

```
<h1>Hello</h1>
```

하지만 파싱하면:

```
태그: h1
내용: Hello
```

처럼 구조화된 데이터로 바뀌는 것을 말함.
파서는? 파싱을 하는 프로그램

2. BeautifulSoup 객체 생성

- BeautifulSoup(html str [, 파서])
  - 매개변수
    1. 정보를 조회할 html을 string으로 전달. BeautifulSoup은 request로 가져온 문서(html)에서 무언갈 찾는 역할
    2. 파서
       - html.parser(기본파서)
       - lxml : 매우 빠르다. html, xml 파싱 가능(xml 파싱은 lxml만 가능)
         - 사용시 install 필요
         - `conda install lxml`
         - `pip install lxml`

3. 문서내에서 원하는 정보 검색하는 방법

(1) Tag 객체

- 하나의 태그(element)에 대한 정보를 다루는 객체.
  - BeautifulSoup 조회 메소드들의 **조회결과의 반환타입.**
  - 조회 함수들이 찾은 Element가 하나일 경우 **Tag 객체를, 여러개일 경우 Tag 객체들을 담은 List(ResultSet)** 를 반환한다.
  - Tag 객체는 찾은 정보를 제공하는 메소드와 Attribute를 가지고 있다. 또 찾은 Tag가 하위 element를 가질 경우 찾을 수 있는 조회 메소드를 제공한다.
  - ex)

```
<a href="dsafdsa">      <!--속성="값"-->
에버랜드                <!--내용-->
</a>
```

- 위와 같은경우, 하나의 태그 객체 안에 속성, 값, 내용이 인스턴스 변수로서 들어가있다. 그 인스턴스변수를 추출하고 싶으면 그에 맞는 메서드를 쓰면됨
- 주요 속성/메소드
  - **태그의 속성값 조회**
    - tag객체.get('속성명') 또는 tag객체['속성명']
    - ex) a.get('href') 또는 a['href']
  - **태그내 text값 조회**
    - tag객체.get_text() 또는 tag객체.text
    - ex) a.get_text() 또는 a.text

- **contents 속성**
  - 조회한 태그의 모든 자식 요소들을 리스트로 반환
  - ex) child_list = a.contents

(2) 조회 함수

- **태그의 이름으로 조회**
  - find_all(): 해당하는거 모두 조회. 부모든 자식이든 해당하는 태그는 모두 조회. 자료구조 형태로 조회된다.
  - find(): 태그 객체 하나만 조회

- **css selector를 이용해 조회**
  - **select(selector='css셀렉터')** : css 셀렉터와 일치하는 tag들을 모두 자료구조에 넣어 반환한다.
  - **select_one(selector='css셀렉터')** : css 셀렉터와 일치하는 tag를 하나만 반환한다. 일치하는 것이 여러개일 경우 첫번째 것 하나만 반환한다.

- `.` **표기법(dot notation)**
  - dom tree 구조의 계층 순서대로 조회
  - 위의 두방식으로 찾은 tag를 기준으로 그 주위의 element 들을 찾을 때 사용

(3) 태그의 이름으로 조회

- **find_all**(name=태그명, attrs={속성명:속성값, ..})
  - 이름의 모든 태그 element들을 리스트에 담아 반환.
  - 여러 이름의 태그를 조회할 경우 List에 태그명들을 묶어서 전달한다.
  - 태그의 attribute 조건으로만 조회할 경우 name을 생략한다.
- **find**(name=태그명, attrs={속성명:속성값})
  - 이름의 태그중 첫번째 태그 element를 반환.

IV. BeautifulSoup 모듈

1. 개요

> 웹 브라우저 제어 도구

- 원래는 웹 어플리케이션 자동 테스트를 위한 목적으로 만들어진 프레임워크
- 웹브라우저에서 새로고침하든, 뒤로가기하든 이런 제어를 위한 코드를 짜놓은 것
- **requests 모듈의 한계**
  - Javascript를 이용한 AJAX 기법의 비동기적 요청 처리 페이지 크롤링이 힘들다.
  - 원래 html은 뭔가 요청이 가해졌을때, 항상 전체 페이지가 reroad되는 것, 하지만 javescript는 웹페이지 중 일부만 없앨 수 있는 언어(ex스팸메일함 비우기)
  - (스크롤해서 내렸을때 새로 나오는 정보들에 대한 크롤링이 어려움)
  - 로그인 후 요청이 가능한 페이지들에 대한 크롤링이 번거롭다.
  - Selenium을 활용하면 이 두가지 모두 쉽게 처리할 수 있다.
- **Selenium 단점**
  - 속도가 느림
- **설치**
  - `pip install selenium`

2. Driver

- 웹브라우저를 제어하는 프로그램으로 웹 브라우저별로 제공된다.
- Selenium 패키지의 Driver객체를 이용해 제어하게 된다.
- 설치: DriverManager 이용
- `pip install webdriver-manager`
- 이걸 설치하면 드라이버 알아서 선택해줌

3. 다운 받은 Driver이용해 WebDriver생성

- WebDriver를 생성하면 웹브라우저가 실행 되며 생성된 웹브라우저를 WebDriver를 사용해서 컨트롤한다.
- 페이지 이동
  - `WebDriver.get("이동할 URL 주소")`
- Web browser 끄기
  - `WebDriver.close()`

(1) WebDriver 주요 속성/메소드

- **page_source** : 현재 페이지의 html 소스를 반환
  - page_source로 html을 받아서 BeautifulSoup으로 크롤링할 원소를 찾을 수 있다.
- **get_screenshot_as_file(파일경로)**

    - 현재 웹브라우저 화면을 지정한 캡처해서 지정한 파일 경로에 저장한다.
- **set_window_size(width, height)**
    - 웹브라우저 윈도우 크기 조정
- **maximize_window()**
    - 웹브라우저 화면 최대 크기로 만들기.
- **get_window_size()**
    - 웹브라우저 윈도우 크기 조회. (width, height)
- **execute_script("자바스크립트코드")**
    - 문자열로 전달한 **javascript 코드**를 실행시킨다.
- **quit()**, **close()**
    - 웹브라우저를 종료한다.

(2) Page의 Element 조회 메소드

- BeautifulSoup을 이용하지 않고 셀레늄 자체 parser를 이용할 수 있다.
- **find_element()**: 조건을 만족하는 첫번째 요소를 반환한다.
    - 매개변수
        - **by**: 검색방식
            - **By.ID**
            - **By.CLASS_NAME**
            - **By.TAG_NAME**
            - **By.CSS_SELECTOR**
            - **By.XPATH**
            - **By.LINK_TEXT**
            - **By.PARTIAL_LINK_TEXT**
        - **value**: str - 검색조건
    - 반환타입: **WebElement**
- **find_elements()**: 조건을 만족하는 모든 요소를 찾는다.
    - 매개변수: find_element()와 동일
    - 반환타입
        - **list of WebElement**

(3) WebElement (조회결과) 메소드 / 속성

- 메소드
    - **get_attribute('속성명')**: 태그의 속성값 조회

    - **send_keys("문자열")**: 입력폼에 문자열 값을 입력.
    - **click()**: element를 클릭
    - **submit()**: element가 Form인 경우 폼 전송
    - **clear()**: element가 입력폼인 경우 텍스트를 지운다.
    - 위 조회 메소드들 : 하위의 elements들 조회
- 속성
    - **text**: 태그내의 텍스트
    - **tag_name**: 태그이름

4. 브라우저의 headless 모드를 이용.

- Headless 브라우저
    - 브라우저의 창을 띄우지 않고 실제 브라우저와 동일하게 동작하도록 하는 방식
    - 화면을 띄우다 보니 메모리를 잡아먹는다. 속도를 높이기 위함.
    - CLI(command line interface) 기반의 OS (리눅스 서버)를 지원하기 위한 브라우저. 화면을 띄울수 없는 애한테 띄우라고하니 에러가 나기 때문
    - 크롬은 버전 60부터 headless 모드 지원
- selenium에서 headless 모드
    - webdriver options에 headless 설정

5. 대기하기

(1) Explicit Wait(명시적 대기)

- 특정 조건/상황을 만족할 때 까지 대기
- `WebDriverWait(browser, 초).until(expected_contition)` 구문 사용
- expected_contition:내가 찾는 태그 넣기.
- '초'는 최대 시간

(2) Implicit Wait(암시적 대기)

- 현재 페이지에 없는 element나 elememt들이 loading 되기를 설정한 시간만큼 기다린다.
- 설정한 시간 이내에 elements가 loading되면 대기가 종료된다.
- `implicit_wait(초)` 구문 사용
- 페이지가 로딩되는 시간을 '초'까지 기다린다.
- 한번 설정하면 설정된 WebDriver가 close될때 까지 그 설정이 유지된다.

(3) 예시

1. 암시적 대기

```
browser.implicit_wait(5)
# 페이지 로딩(dom tree완성)될 때까지 최대 5초간 기다린다.
# 로딩이 되면 5초가 되지 않아도 대기를 끝낸다.
```

2. 명시적 대기

```
from selenium.webdriver.support import expected_conditions as EC

...

try:
    # element가 반환될 때 까지 최대 10초 기다린다.
    element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "myDynamicElement"))
    )
finally:
    driver.quit()
```

6. 무한 스크롤

- javascript 에서 현재 페이지의 높이(scroll pane(scroll bar가 움직이는 공간)의 길이)
  - `document.documentElement.scrollHeight`
- scroll bar를 이동
  - `window.scrollTo(가로 스크롤바를 이동시킬 위치:정수,
    세로 스코롤바를 이동시킬 위치:정수)
- 현재 페이지의 높이 height를 구해서 그것의 맨 밑으로 스크롤바를 내린다> 그러면 현재 페이지의 길이가 길어진다.> 이전의 페이지 길이와 현재 페이지의 길이가 같은지 확인> 다르다면 다시 스크롤을 맨 아래로 내려준다.> 또다시 이전의 페이지 길이와 현재 페이지 길이 같은지 확인> 이전 페이지 길이와 현재 페이지 길이가 같아질 때까지 진행> 같아지면 stop
- 셀레늄엔 이런 기능이 없어서 자바스크립트 코드로 진행한다.

V. OPEN API

> 누구나 사용할 수 있도록 공개된 API (Application Programming Interface)
> 즉, 외부 개발자나 사용자도 접근 가능한 인터페이스

Open API는 말 그대로 공개된 프로그래밍 인터페이스로, 외부 개발자나 사용자가 특정 서비스나 애플리케이션에 접근하여 서비스를 받을 수 있도록 공개된 API이다.
데이터 추가, 삭제, 수정, 조회 등의 서비스를 제공하는 것을 open api라고함.

1. 정의

Open API는 애플리케이션 개발자가 공개된 API를 사용해 다른 서비스와 애플리케이션을 연동할 수 있도록 만든 인터페이스이다.
일반적으로 RESTful API 형식으로 서비스 한다.

2. 특징

- 공개성: 누구나 접근할 수 있으며, 문서화가 잘 되어 있어 사용자가 쉽게 활용할 수 있음.
- 표준화: 대부분 표준화된 HTTP 프로토콜과 JSON, XML 형식을 사용.
- 보안성: API 키나 OAuth 같은 인증 방식으로 보안을 유지.

3. 사용 사례

다양한 기업, 공공기관에서 다양한 서비스를 오픈 api로 제공한다.

- 공공데이터 포털: 행정안전부에서 서비스하는 정부, 공공기관, 지자체 등이 보유한 데이터를 개방하고 제공하는 플랫폼.
- 구글 맵 API: 외부 애플리케이션에서 구글 맵을 활용할 수 있게 해주는 대표적인 Open API.
- 트위터 API: 트위터(X) 데이터를 외부에서 가져오거나 포스팅할 수 있도록 제공.
- 네이버 개발자 오픈 API: 네이버의 다양한 서비스를 제공. (검색, 검색어 트랜드 조회, 캘린더 등)

4. 공공데이터 포털 데이터 조회

- 서비스를 받기위한 API key를 신청한다. 로그인과는 다름. 데이터 요청시 api key를 제출한다!
- 데이터를 조회 등을 하기 위한 가이드가 데이터 제공자마다 각자 있다.
- 가이드에 따라 요청방식, 요청 URL, 전달 값을 맞춰 요청한다.
- 사용자마다 유일한 값으로 key를 갖는 것(임시 id같은 개념)
- ID와는 다르게 쉽게 삭제하고 다시 발급받을 수 있다.
- 검색어 입력후 오픈API 활용 신청

5. 요청해야하는 url 만들기

1. 상세설명에 들어가면 base url이 있다, 거기 하단에 있는 API 목록에서 원하는 api를 뒤에 붙여서 주소를 만든다. 아래처럼 만들어짐
2. http://apis.data.go.kr/1613000/BUSINESS_CAR/T_OD_BUSINESS_CAR_BRN_INFO
3. https로 하면 안됨!!

4. 하단에는 미리보기가 가능하도록 해주는데, 요청할 api 누르면 하단에 적어줘야할게 나온다 그걸 양식에 맞게 params를 적으면 됨

5. 예제

```
import requests
import json


url='http://apis.data.go.kr/1613000/BUSINESS_CAR/T_OD_BUSINESS_CAR_BRN_INFO'
with open('api_key.json', 'rt') as fr: # apikey 적어놓은 파일 읽어들이기
    key_dict = json.load(fr) #json 텍스트 - > dict: load()
    # json : 위에서 rt로 읽어들인 텍스트의 문자열을 dict으로 바꿔준다.
    # apikey를 git push 하지 않도록 .gitignore에 적어둬야 한다.


key = key_dict['apikey']
params ={'serviceKey' : key,
         'pageNo' : '1',
         'numOfRows' : '3',
         'resultType' : 'json',
         'crtr_yr' : '2023',
         'mtrpl_lcgv_nm' : '서울'
        }
# key 자리에 apikey 그냥 넣어도되는데, 모듈을 사용해서 변수를 넣도록 한다.
# 나중에 git 등에 올렸을때 apikey 유출되면 1억 물수있다.
# 요구하던 항목들을 dict 형태로 적어서 거기에 맞게 적어야한다.

response = requests.get(url, params=params)
if response.status_code == 200: #응답이 정상적으로 오면
    result = response.json() # json 형태로 받는 것
    print(type(result))
    from pprint import pprint
    pprint(len(result['response']['body']["items"]["item"]))
    pprint(result['response']['body']["items"]["item"][0])
```