

웹 크롤링 세 갈래의 길

화면에 보이는 것이 코드에도 보이는 것은 아님.

어느 길로 갈지는 개발자도구로 5분 관찰해 정하는 것이 원칙.

3 데이터를 꺼내는 갈래	5 실습 대상 사이트	10 복붙 프롬프트	0 코랩에서 설치할 프로그램
-------------------------	-----------------------	----------------------	---------------------------

강의	일시	주제	담당
제5회 문과코딩 워크숍 · 4일차	2026. 8. 27.(목)	웹 크롤링 — 이론과 실습	이문혁 (경희대 인문사회과학데이터연구소)

오늘의 지도

오늘 만들 결과물은 엑셀 파일 몇 개. 그 파일을 만드는 길이 셋이고, 오늘은 그 셋을 구분하는 눈을 갖는 것이 목표.

TRACK 01 · 정적

보이는 대로 있는 페이지

서버가 완성된 HTML을 통째로 보내 줌. 코드로 받아도 화면과 같은 내용이 들어 있음. requests + BeautifulSoup로 처리함.

TRACK 02 · 숨은 API

값이 따로 배달되는 페이지

처음 받은 HTML은 빈 껍데기이고, 값은 뒤이어 JSON으로 따로 옴. 그 배달 주소를 찾아 직접 부르면 오히려 더 쉬움.

TRACK 03 · 브라우저 자동화

눌러야 나오는 페이지

클릭·스크롤을 해야 값이 나타남. 사람 대신 브라우저를 조종해야 함. 바이브 코딩이 압도적으로 유리한 영역.

한 줄 요약 — 크롤링에서 어려운 것은 코드가 아니라 어느 갈래인지 알아보는 일. 갈래만 정해지면 코드는 AI가 대신 써 줌. 오늘 배우는 것은 그 판별법과, 판별한 다음 AI에게 무엇을 어떻게 넘길지.

I 오늘의 순서

1부

웹을 이루는 것들

크롤링 이야기는 잠시 접어 둬. 웹이 무엇으로 만들어져 있는지부터 봄. IT 용어 · 서버와 클라이언트 · HTML · CSS.

2부

크롤링에 적용하기

1부에서 배운 것을 데이터 수집에 씬. 이름표를 좌표로 쓰는 법 · 개발자도구 · 세 갈래 판별법.

3부

직접 해 보기

코랩으로 두 갈래를 실습하고, 같은 일을 AI에게 시켜 봄. 마지막에 지켜야 할 선까지.

I 오늘 만드는 파일

순서	대상	갈래	강의안	노트북	결과물
A-1	Yes24 베스트셀러	정적	14절	1~5절	yes24_best_260827.xlsx
A-2	멜론 TOP100	정적	14절	6절	melon_top100_260827.xlsx
B-1	메가박스 영화목록	숨은 API	16절	7~9절	megabox_movies_260827.xlsx
B-2	KOBIS 예매율	숨은 API	16절	10절	kobis_realtime_260827.xlsx
—	네이버뉴스	숨은 API	17절	11~14절	검색어_언론사_news_260827.xlsx
C	스크롤이 필요한 페이지	브라우저	18절	—	수집 결과 + 스크립트

주. A·B는 코랩에서, C는 AI로만 함(노트북에 없음). 「강의안」 칸이 이 문서의 몇 절인지, 「노트북」 칸이 실습 노트북의 몇 절인지임 — 두 자료를 나란히 열어 두고 따라오면 됨. 네이버뉴스는 기법보다 «고쳐 가며 쓰는 일»을 배우는 자리.

3부부터는 코랩과 이 문서를 번갈아 봄. 코랩에서 코드를 돌리고, «왜 이렇게 생겼는지»가 궁금하면 위 표의 「강의안」 칸으로 돌아오면 됨.

Part 01

웹을 이루는 것들

여기서는 크롤링을 다루지 않음. 웹페이지가 무엇으로 되어 있고 브라우저가 무슨 일을 하는지만 봄. 이 토대가 있어야 2부의 수집 기법이 요령이 아니라 원리로 이해됨.

알아 둘 IT 용어

오늘 계속 나올 말들. 지금 완벽히 외울 필요는 없고, 나중에 나왔을 때 «아까 그거»라고 떠올릴 수 있으면 충분함.

I 누가 누구에게 — 컴퓨터 사이의 자리

클라이언트 client 서비스를 받아 쓰는 쪽 . 내 노트북, 내 스마트폰, 내 브라우저가 모두 클라이언트. 오늘 쓸 파 이션 코드도 클라이언트가 됨.	서버 server 정보나 서비스를 제공하는 쪽 . 특별한 기계라기보다 그런 역할을 맡은 컴퓨터를 뜻함. 네이 버 서버, 멜론 서버처럼 부름.
프론트엔드 front-end 사용자 눈에 보이는 앞면 을 만드는 영역. 화면 디자인, 버튼, 입력창처럼 사람이 직접 만지 는 부분.	백엔드 back-end 서버 쪽에서 데이터를 꺼내고 응답을 만드는 보이지 않는 뒷면 . 데이터베이스가 여기 있음.

I 무엇을 주고받는가 — 방식과 형식

HTTP · HTTPS protocol 클라이언트와 서버가 대화하는 규칙 . 주소가 https://로 시작하는 것이 이 규칙을 쓴다 는 표시. 뒤의 S는 오가는 내용을 암호로 감싼다는 뜻.	API Application Programming Interface 프로그램끼리 기능을 주고받기로 한 약속 . «이 주소로 이렇게 물으면 이런 형태로 답해 준 다»가 정해져 있음. 사람이 보는 화면 대신 프로그램이 읽을 자료 를 돌려줌.
HTML HyperText Markup Language 사람에게 보여 주려고 만든 형식. 글자에 «이건 제목», «이건 링크» 같은 표지를 붙여 둔 문 서. 5절에서 자세히 봄.	JSON JavaScript Object Notation 프로그램끼리 주고받으려고 만든 형식. 꾸밈이 없고 이름과 값만 짝지어 늘어놓음. 파이썬 의 딕셔너리와 생김새가 거의 같음.

주. HTML과 JSON의 차이가 오늘 배울 세 갈래의 갈림길. 같은 값이 어느 형식으로 오느냐에 따라 쓰는 도구가 달라짐.

I 코드를 다루는 말

라이브러리 library 남이 미리 만들어 둔 기능 꾸러미 . 직접 짜지 않고 가져다 씀. 오늘은 requests(요청 보내기) · BeautifulSoup(HTML 뒤지기) · pandas(표 만들기) 셋을 씀.	임포트 import 그 꾸러미를 지금 쓰겠다고 불러오는 것 . import requests 한 줄이 그 일을 함. 코랩에는 오늘 쓸 라이브러리가 이미 깔려 있어 설치할 것이 없음.
객체 object 값과 기능을 한 덩어리로 묶은 것 . 글자 문치는 그냥 글자지만, 객체로 바꾸면 «여기서 제목만 찾아 줘» 같은 일을 시킬 수 있음.	주석 comment 코드에 남기는 설명 글 . # 뒤에 씀. 컴퓨터는 무시하고 사람만 읽음. 오늘 배포하는 코드는 주석으로 설명을 붙여 두었음.

I 글자와 도구

문자 인코딩 character encoding 사람의 글자를 컴퓨터의 숫자로 바꾸는 규칙 . 보내는 쪽과 읽는 쪽이 다른 규칙을 쓰면 글자가 깨짐. 요즘 표준은 UTF-8 이고, 전 세계 문자를 담은 표를 유니코드 라 부름.	IDE · 디버그 IDE · debug IDE는 코드를 쓰고 실행하는 일을 한자리에서 하게 해 주는 작업 프로그램 임(VS Code 등). 디버그는 문제를 찾아 고치는 일 . 3부 바이브 코딩에서 이 둘을 씀.
--	---

I 웹페이지에 관한 말

URL Uniform Resource Locator 웹 주소. https://... 로 시작하는 그 줄 전체를 가리킴. «어느 서버의 어느 자리»를 적은 것이고, 뒤에 조건을 덧붙일 수도 있음. 3절에서 뜯어봄.	정적 페이지 · 동적 페이지 static · dynamic 페이지가 값을 언제 채우느냐의 차이. 서버가 보내 줄 때 이미 값이 들어 있으면 정적, 받은 뒤에 자바스크립트가 채워 넣으면 동적. 겉보기로는 구분이 안 되고, 오늘 이 구분이 계속 나옴.
--	--

I 오늘 쓰는 도구에 관한 말

구글 코랩 Google Colab 설치 없이 웹브라우저에서 파이썬을 실행하는 곳. 내 컴퓨터에 아무것도 깔지 않아도 되고, 결과를 구글 드라이브에 저장할 수 있음.	셀 cell 코랩에서 코드나 글을 담는 한 칸. 위에서부터 한 칸씩 실행함. 한 셀에서 만든 값은 다음 셀에서도 그대로 쓸 수 있음.
데이터프레임 DataFrame 행과 열이 있는 표 모양의 자료. 엑셀 시트와 거의 같다고 보면 됨. pandas가 만들어 주고, 그대로 엑셀 파일로 저장할 수 있음.	터미널 terminal 글자로 명령을 적어 컴퓨터에게 시키는 입력창. 마우스로 누르는 대신 이름을 적어 실행함. VS Code 안에도 들어 있음.
브라우저 자동화 Playwright · Selenium 사람 대신 브라우저를 조종해 주는 도구. 페이지를 열고 스크롤하고 버튼을 누르는 일을 코드로 시킴. 오늘 세 번째 갈래에서 씬.	헤드리스 headless 브라우저를 창 없이 뒤에서만 돌리는 방식. 빠르지만 무슨 일이 벌어지는지 볼 수 없음. 오늘은 창이 보이게 끄고 씬.

여기 있는 말들은 **오늘 계속 나오는 것**. 지금 다 외울 필요는 없고, 나왔을 때 «아까 그 표에 있던 거»라고 떠올릴 수 있으면 충분함.

웹은 어떻게 작동하는가

주소창에 무언가를 넣고 엔터를 치면 실제로 무슨 일이 벌어지는가. 이것만 알면 나머지는 응용.

요청 · REQUEST

내 쪽에서 서버로

브라우저가 «이 주소의 내용을 주세요»라고 서버에 말을 걸. 이때 나를 소개하는 정보도 함께 보냄 — 어떤 브라우저인지, 어느 페이지에서 왔는지 같은 것.

응답 · RESPONSE

서버에서 내 쪽으로

서버가 요청을 받아 자료를 꺼내 돌려줌. 이때 «성공했다 / 그런 페이지 없다» 같은 처리 결과도 숫자로 함께 붙여 보냄.

I 브라우저가 하는 네 가지 일

01 주소를 해석함

주소창에 넣은 글자에서 «어느 서버의 어느 경로인지»를 읽어 냄.

02 요청을 보냄

그 서버에 접속해 내용을 달라고 함. 이것이 HTTP 요청.

03 글자 뭉치를 받음

서버가 돌려주는 것은 그림이 아니라 **HTML이라는 글자 뭉치**. 이 사실이 오늘 강의의 출발점.

04 보기 좋게 그림

그 글자 뭉치를 해석해 제목은 크게, 링크는 파랗게, 이미지는 이미지로 그려 냄. 우리가 보는 화면은 이 마지막 단계의 결과.

여기가 핵심 — 서버가 보내 주는 것과 우리가 보는 것은 같지 않음. 서버는 **글자 뭉치**를 보내고, 브라우저가 그것을 **그림**으로 만듦. 브라우저를 거치지 않고 글자 뭉치를 그대로 받아 보는 방법도 있음.

I 주소 하나에 담긴 것

```
https://search.naver.com/search.naver?where=news&query=올림픽&sort=1
```

부분	이 예시에서	하는 일
프로토콜	https://	주고받는 규칙. 요즘은 사실상 전부 https임
주소	search.naver.com/search.naver	어느 서버의 어느 경로인지
파라미터	?where=news&query=올림픽&sort=1	물음표 뒤에 조건을 적고 &로 이어 붙임

- 물음표 뒤가 **질문지**. query=올림픽을 query=총선으로 바꾸면 다른 결과가 옴
- 즉 **조건을 주소로 표현**할 수 있고, 그래서 사람이 클릭하지 않아도 조건만 바뀌 가며 반복할 수 있음
- 한글은 주소에 그대로 못 들어가므로 %EC%98%AC... 형태로 바뀜. 이것을 **URL 인코딩**이라 함

이 절에서 배운 **요청 · 응답 · 주소** 셋이 2부에서 코드가 됨. 12절에서 같은 이야기를 파이썬으로 다시 봄.

HTML · CSS · JavaScript

웹페이지 하나는 세 가지 언어가 나눠 맡아 만들어짐. 역할이 완전히 갈리므로 헷갈릴 일이 없음.

HTML

뼈대와 내용

무엇이 제목이고 무엇이 문단이고 무엇이 이미지인지를 정함. 웹페이지의 구조를 만드는 언어.

예 — `<h1>` `<p>` `<div>` `<img>`

CSS

겉모습과 배치

색·크기·여백·위치를 정함. HTML이 만든 뼈대에 **디자인**을 입히는 언어.

예 — `color` `font-size` `margin`

JAVASCRIPT

움직임과 반응

버튼을 누르면 무언가 일어나게 함. 값을 나중에 채워 넣는 일도 함. 웹페이지에 **동작**을 주는 언어.

예 — 버튼 클릭, 더보기, 무한 스크롤

집짓기에 빗대면 **HTML**은 골조, **CSS**는 마감과 도색, **JavaScript**는 문과 조명 같은 설비. 골조만 있어도 집의 구조는 알 수 있고, 도색이 없다고 방의 개수가 달라지지는 않음. 우리가 오늘 가져오려는 것은 방의 개수와 이름, 즉 HTML이 담고 있는 내용.

미리 짚어 둘 것 — 셋 중 **JavaScript**가 오늘의 난이도를 결정함. 값을 처음부터 HTML에 넣어 두는 페이지가 있고, JavaScript가 나중에 채워 넣는 페이지가 있음. 앞의 것은 쉽고 뒤의 것은 한 단계를 더 거쳐야 함. 15절에서 이 갈림길을 다룸.

HTML — 웹페이지의 뼈대

HyperText Markup Language. 이름이 길지만 하는 일은 하나임 — 글자에 «이게 무엇인지»를 표시해 두는 것.

무엇인가

표지를 붙인 문서

그냥 글자만 있으면 컴퓨터는 어디가 제목이고 어디가 본문인지 모름. 그래서 글자마다 표지를 붙여 둬. 그 표지가 태그.

텍스트·이미지·링크·동영상을 브라우저에 보여 주기 위한 규칙의 모음.

왜 마크업 언어라 하는가

계산하지 않고 표시만 함

파이썬 같은 프로그래밍 언어는 계산하고 판단함. HTML은 계산하지 않음. «여기부터 여기까지가 제목이다»라고 표시만 함. 그래서 프로그래밍 언어가 아니라 마크업 언어라 부름.

I 태그와 요소 — 한 줄을 해부해 봄

```
<a href="https://www.khu.ac.kr/" class="gd_name" >경희대학교 </a>
```

■ 태그 이름 — a는 링크라는 뜻 ■ 속성 — 부가 정보. href는 연결 주소, class는 이름표 ■ 내용 — 화면에 실제로 보이는 글자 ■ 종료 태그 — 여기서 끝난다는 표시

TAG

태그

HTML 문서를 이루는 기본 단위. <태그명>으로 열고 </태그명>으로 닫음.
브라우저에게 «이 내용이 무엇인지»를 알려 주는 역할.

ELEMENT

요소

시작 태그부터 종료 태그까지, 그 사이의 내용을 **전부 합친 덩어리**.
앞 쪽 예시에서 <a ...>경희대학교 전체가 요소 하나.

I 요소 안에 요소가 들어감

```
<div>                                ← 바깥 상자
  <h2>오늘의 뉴스</h2>              ← 그 안의 제목
  <p>첫 번째 기사입니다.</p>        ← 그 안의 문단
</div>
```

서랍장 안에 서랍이 있고 서랍 안에 물건이 있는 것과 같음. 이렇게 **겹겹이 들어간 구조**를 중첩이라 부름. 웹페이지는 예외 없이 이 모양이고, 깊은 곳은 열 겹이 넘기도 함. 2부에서는 이 중첩을 타고 내려가게 됨.

여기까지가 **HTML의 전부**. 태그로 표시하고, 속성으로 정보를 붙이고, 겹겹이 넣음. 다음 절에서 실제로 자주 만나는 태그를 봄.

자주 쓰는 태그

태그는 수백 가지지만 실제로 자주 보이는 것은 열 개 남짓. 이것만 알아도 웹페이지 구조가 읽힘.

태그	이름	하는 일	담기는 것
<h1> ... <h6>	제목	제목의 크기와 위계를 나타냄. h1이 가장 큼	제목 글자
<p>	문단	하나의 문단을 묶음	본문 글자
<div>	구역	여러 요소를 덩어리로 묶는 상자 . 줄을 통째로 차지함(블록)	기사 한 건, 상품 한 개 같은 단위
	범위	줄 안에서 일부만 묶음 . 줄을 바꾸지 않음(인라인)	순위, 날짜 같은 짧은 값
<a>	링크	다른 페이지로 연결함	보이는 글자 + href 주소
	이미지	그림을 넣음. 닫는 태그가 없음	src 주소
	목록	여러 항목을 목록으로 늘어놓음	반복되는 항목들
<table> <tr> <td>	표	행과 칸으로 된 표를 만들	표 전체

I div와 span의 차이

BLOCK

<div> — 줄을 통째로 씬

앞뒤로 줄이 바뀜. 그래서 여러 요소를 묶는 **큰 상자** 역할을 함.

```
<div>첫 번째 상자</div>
<div>두 번째 상자</div>
→ 위아래로 쌓임
```

INLINE

 — 줄 안에 머무름

줄을 바꾸지 않음. 문장 가운데 **일부만 집어내는** 데 씬.

```
오늘 순위는 <span>1위</span>임
→ 한 줄 안에 그대로 있음
```

주. div와 span 둘 다 그 자체로는 아무 모양이 없음. 순전히 묶는 용도이고, 모양은 CSS가 입힘. 그래서 웹페이지 HTML을 열면 <div>가 압도적으로 많이 보임.

I 속성 — 태그에 붙는 부가 정보

태그 이름만으로는 부족할 때가 있음. 링크라면 **어디로** 가는지, 이미지라면 **어느 파일**인지를 알려 줘야 함. 그 정보를 **속성**이라 하고 이름="값" 꼴로 시작 태그 안에 씀.

속성	어디에	무엇을
href	<a>	연결될 주소
src		이미지 파일의 주소
alt		그림이 안 뜰 때 대신 보일 설명
id	모든 태그	하나뿐인 이름표 — 8절에서 자세히
class	모든 태그	여럿을 묶는 이름표 — 8절에서 자세히

마지막 두 줄 id와 class가 **오늘 가장 중요한 속성**. 원래는 디자인을 위해 붙이는 이름표인데, 2부에서 이것이 데이터를 찾는 좌표가 됨.

CSS — 웹페이지의 겉모습

Cascading Style Sheets. HTML이 구조와 내용을 정한다면, CSS는 그것이 어떻게 보일지를 정함.

HTML만으로 만든 페이지는 검은 글씨가 흰 배경에 줄줄이 놓인 모습. 여기에 색을 주고 글자 크기를 정하고 여백을 넣고 좌우로 나란히 배치하는 일을 CSS가 함. **표현(CSS)** 과 **구조(HTML)**를 갈라 놓았기 때문에 디자인을 바꿔도 내용은 그대로 둘 수 있음.

I CSS 한 덩어리의 구조

```
a { color: blue; font-size: 16px; }
```

■ 선택자 — 어디에 적용할지 지목 ■ 선언부 — 중괄호로 감싼 부분 전체 ■ 속성 — 무엇을 바꿀지 ■ 값 — 어떻게 바꿀지

읽는 법은 이러함 — «모든 a 태그에 대해, 글자색은 파랑으로, 글자 크기는 16픽셀로 하라.» 선택자가 대상이고 선언부가 지시 내용.

I 네 부분을 나눠 보면

부분	영어	뜻	예
선택자	selector	스타일을 적용할 대상 을 지목하는 부분	a · .title · #main
선언부	declaration block	중괄호 { } 안에 들어가는 지시 내용 전체	{ color: blue; }
속성	property	무엇을 바꿀지 정하는 항목	color · font-size · margin
값	value	그 속성에 어떤 값 을 줄지	blue · 16px · 20px

I 대상을 지목하는 세 가지 방법

태그로 p 모든 <p> 태그. 종류 전체를 한꺼번에 지목함.	CLASS로 .title 이름표가 title인 것 전부. 앞에 마침표를 붙임.	ID로 #main 이름표가 main인 하나. 앞에 우물정을 붙임.
---	--	--

여기서부터가 오늘의 열쇠 — CSS가 디자인을 입히려면 **«어디에 입힐지»**를 먼저 골라내야 함. 그 골라내는 문법이 선택자. 그런데 이 «골라내기»는 디자인에만 쓸 수 있는 것이 아님. 2부에서 이것을 그대로 데이터를 골라내는 데 쓸 것.

id와 class — 이름표

태그 이름만으로는 원하는 것을 꼭 집을 수 없음. 페이지에 <div>가 수백 개이기 때문임. 그래서 이름표를 붙임.

디자이너 입장에서 생각해 보면 이유가 분명해짐. «제목은 크게, 날짜는 회색으로»라고 하려면 **어디가 제목이고 어디가 날짜인지 먼저 구분돼 있어야 함**. 그래서 웹페이지를 만들 때 요소마다 `class="title"`, `class="date"` 같은 이름표를 미리 붙여 둬. 이름표는 디자인을 위해 붙인 것이지만, 결과적으로 **페이지의 모든 부분에 이름이 생긴 셈이 됨**.

	id	class
쓰임	딱 하나를 고유하게 구분할 때	여러 개를 묶어서 다룰 때
중복	한 문서에 같은 이름을 한 번만	같은 이름을 여러 번 반복 가능
CSS 기호	#아이디	.클래스
한 요소에	id는 하나만	class는 여러 개를 동시에 부여 가능
예	<div id="newsct_article">	<div class="itemUnit">
어울리는 대상	기사 본문처럼 페이지에 하나뿐인 영역	목록 항목처럼 같은 모양이 반복되는 단위

I 실제 페이지에서는 이렇게 생겼음

```

<ul id="yesBestList">                ← 이 영역은 페이지에 하나뿐임 → id
  <li>
    <div class="itemUnit">             ← 책 1권. 같은 모양이 반복됨 → class
      <a class="gd_name">책 제목</a>
      <span class="info_auth">지은이</span>
    </div>
  </li>
  <li> ... </li>                      ← 같은 이름표가 24번 반복
</ul>

```

- 바깥 상자에는 id가 붙음 — 베스트셀러 목록 영역은 이 페이지에 하나뿐이기 때문임. `Ctrl+U`로 yesBestList 를 검색하면 실제로 있음
- 안쪽 반복 단위에는 class가 붙음 — 책 24권이 전부 같은 이름표를 나눠 가짐
- 이 구분은 관례이지 강제는 아니나, 실제 웹사이트 대부분이 이 방식을 따름

1부를 여기서 마칩 — 웹페이지는 서버가 보내 준 **글자 덩치(HTML)**이고, 그 안의 요소마다 **이름표(class·id)**가 붙어 있으며, 그 이름표를 지목하는 문법이 **선택자**. 이 세 가지가 2부에서 그대로 도구가 됨.

Part 02

크롤링에 적용하기

1부에서 배운 것을 데이터 수집에 씬. 새로 배우는 문법은 거의 없음. 디자인을 위해 만들어진 이름표와 선택자를 «수집의 좌표»로 바꿔 쓰는 것이 전부.

크롤링이란 무엇인가

브라우저가 늘 하던 일을 파이썬이 대신 하는 것. 특별한 기술이 아니라 자리를 바꾸는 일.

사람이 할 때

브라우저 + 눈 + 손

주소를 넣고, 화면을 보고, 필요한 값을 눈으로 찾아, 엑셀에 손으로 옮겨 적음. 100건이면 100번 반복해야 함.

코드가 할 때

requests + BeautifulSoup + pandas

같은 주소로 요청을 보내고, 받은 글자 덩치에서 이름표를 따라 값을 찾아, 표로 만들어 저장함. 100건이든 1,000건이든 같은 코드.

I 크롤링과 스크래핑 — 자주 섞어 쓰는 두 말

CRAWLING

크롤링 — 돌아다니는 일

링크를 타고 페이지에서 페이지로 옮겨 다니며 문서를 모으는 일에 가까움. 검색엔진이 웹 전체를 훑는 동작이 원래 뜻.

SCRAPING

스크래핑 — 긁어내는 일

한 페이지에서 필요한 값만 골라 뽑는 일. 오늘 하는 작업은 엄밀히는 이쪽에 더 가까움.

주. 실무에서는 둘을 구분 없이 «크롤링»이라 부르는 경우가 대부분이고, 오늘도 그렇게 부름. 다만 다른 자료에서 두 말이 갈라져 나오면 «돌아다니기»와 «뽑아내기»의 차이로 이해하면 됨.

I 오늘의 한 사이클 - 네 단계

01 요청 - 페이지를 받아 옴

`requests.get(주소)` 한 줄이 브라우저의 주소창 역할을 함. 돌아오는 것은 화면이 아니라 **HTML 글자 문치**.

02 파싱 - 뒤질 수 있는 형태로 바꿈

글자 문치 그대로는 «세 번째 div 안의 a 태그»를 찾을 수 없음. BeautifulSoup이 이것을 태그를 따라 이동할 수 있는 **객체**로 바꿔 줌. 이 단계를 파싱(parsing)이라 함.

03 추출 - 이름표를 따라 값을 꺼냄

1부에서 배운 **class.id**와 **선택자**가 여기서 쓰임. 반복되는 덩어리를 전부 찾아 하나씩 돌면서 필요한 값만 꺼냄.

04 저장 - 표로 만들어 파일로 남김

pandas로 행과 열이 있는 표를 만들고 엑셀 파일로 저장함. 여기까지가 한 바퀴.

세 라이브러리가 한 단계씩 맡음 - requests가 받아 오고, BeautifulSoup이 뒤지고, pandas가 표로 만듦. 오늘 나오는 코드는 거의 전부 이 셋 중 하나를 부르는 것.

크롤링에 CSS가 필요한 이유

디자인을 하려고 붙여 둔 이름표가, 그대로 데이터를 찾는 좌표가 됨.

01 웹페이지는 HTML로 뼈대를 만들고 CSS로 디자인을 입힘

1부에서 본 그대로.

02 CSS는 디자인할 부분을 구분하려고 이름표를 붙임

«제목은 크게, 날짜는 회색으로»를 하려면 어디가 제목인지 먼저 갈라 놔야 하기 때문임.

03 그 결과 페이지의 모든 조각에 이름이 생김

`class="title", class="itemUnit" ...` 사람이 보라고 붙인 것이 아닌데도 기계가 읽을 수 있는 지도가 생김.

04 크롤러는 그 이름표를 따라감

`.title`만 골라내면 제목만 모을 수 있음.

I 같은 문법, 다른 목적

디자이너는

이 부분을 이렇게 보이게 하라

```
.gd_name { font-weight: bold; }
```

우리는

이 부분을 가져오라

```
soup.select('.gd_name')
```

지목하는 문법 `.gd_name`은 완전히 같음. 달라지는 것은 **그다음에 무엇을 하느냐**뿐. 그래서 새로 외울 문법이 거의 없음.

뒤집어 말하면 — 사이트를 개편해 디자인이 바뀌면 이름표도 바뀌어 어제까지 돌던 코드가 오늘 0건을 뺌. **17절에서 이 이야기를 다시 함.**

선택자로 지목하기

1부에서 배운 선택자 문법을 파이썬에서 그대로 씀. 표기법이 두 가지 있는데 결과는 같음.

I 선택자 문법 – 이것만 알면 대부분 해결됨

쓰고 싶은 말	선택자	BeautifulSoup
모든 문단	p	soup.select('p')
클래스가 title인 것 전부	.title	soup.select('.title')
아이디가 main인 것 하나	#main	soup.select_one('#main')
클래스가 rank01인 div	div.rank01	soup.select('div.rank01')
그 안에 있는 a	div.rank01 a	soup.select('div.rank01 a')
클래스가 lst50 또는 lst100인 tr	tr.lst50, tr.lst100	soup.select('tr.lst50, tr.lst100')
주. 띄어쓰기는 «그 안에 있는»이라는 뜻이고 쉼표는 «또는»이라는 뜻. 이 둘만 구분하면 됨.		

I 두 가지 표기법 – 같은 일을 하는 두 방식

하는 일	find 계열	select 계열 (CSS 선택자)
여러 개 찾기	find_all('div', class_='itemUnit')	select('div.itemUnit')
하나만 찾기	find('a', class_='gd_name')	select_one('a.gd_name')
중첩 지목	여러 번 이어서 호출	select('div.rank01 a') 한 번
주. 결과는 같음. 오늘 실습에서 Yes24는 find 계열로, 멜론은 select 계열로 다뤄 두 방식을 모두 보게 됨. 중첩이 깊을수록 select가 편함.		

주. `class_`의 밑줄은 오타가 아님. `class`가 파이썬이 이미 쓰고 있는 낱말이라 겹치지 않게 밑줄을 붙인 것.

I 반복되는 덩어리를 훑는 기본형

목록형 페이지는 거의 예외 없이 **같은 모양의 덩어리가 반복됨**. 그 덩어리를 전부 찾아 하나씩 돌면서 그 안에서 값을 꺼내는 것 — 이것이 크롤링 코드의 기본 골격이고, 오늘 나오는 코드는 전부 이 변형.

```
# ① 반복되는 덩어리를 전부 찾음
books = soup.find_all('div', class_='itemUnit')

# ② 하나씩 돌면서
for book in books:
    # ③ 그 덩어리 «안에서» 제목만 꺼냄
    title = book.find('a', class_='gd_name').text
    print(title)
```

자주 하는 실수 — `.text`를 빼면 태그가 통째로 딸려 나옴. `<a class="gd_name">책 제목</a>` 전체가 출력되고 «왜 이상한 게 나오지» 하게 됨. **값만 원하면 `.text`를 붙임.**

요청과 응답의 실제

3절에서 본 요청-응답을 코드로 옮기면 이렇게 됨. 첫 확인은 언제나 상태코드.

```
import requests
html = requests.get(URL)
print(html)          # → <Response [200]>
print(html.text)     # → 받아 온 HTML 전체(글자 뭉치)
```

I 상태코드 — 서버가 붙여 보내는 처리 결과

코드	뜻	크롤링에서의 대처
200	성공	정상. 다음 단계로 진행함
301 / 302	주소가 옮겨짐	대개 자동으로 따라감. 최종 주소를 확인함
403	접근 거부	사람이 아닌 것으로 판단된 상태. User-Agent 헤더를 붙여 봄
404	그런 페이지 없음	URL 오타를 의심함
429	너무 자주 요청함	요청 간격을 늘림. 오늘 반드시 지킬 예의
500	서버 잘못	내 코드 문제가 아님. 잠시 뒤 다시 시도함
★ 주. 상태코드가 200인데 원하는 값이 비어 있다면 코드가 아니라 갈래를 잘못 고른 것(15절 참조). 이 경우가 초보자를 가장 오래 붙잡음.		

I 헤더 — 나를 소개하는 쪽지

요청에 함께 붙여 보내는 부가 정보. 서버는 이것을 보고 요청을 받아 줄지 판단하기도 함.

```
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) ...", # 어떤 브라우저인지
    "Referer": "https://www.melon.com/chart/" # 어느 페이지에서 왔는지
}
res = requests.get(url, headers=headers)
```

- **User-Agent** — 없으면 파이썬임이 드러나고, 일부 사이트는 그것만으로 거절함
- **Referer** — «정상 페이지에서 넘어온 요청»임을 알림. 사이트에 따라 이것을 봄
- 헤더를 붙이는 것은 속임수가 아니라 **브라우저와 같은 조건으로 요청하는 것**

I GET과 POST · 한글이 깨질 때

GET

조건을 주소에 실어 보냄

물음표 뒤에 조건이 그대로 보임. 주소만 복사하면 같은 결과를 다시 받을 수 있음. 오늘 대부분의 실습이 이 방식.

POST

조건을 본문에 담아 보냄

주소에는 조건이 안 보이고 따로 실어 보냄. 조건이 길거나 많을 때 씬. 실습 B-2의 KOBIS가 이 방식.

인코딩

글자가 깨져 보일 때

2절에서 본 문자 인코딩 문제. 보내는 쪽과 읽는 쪽의 규칙이 어긋나면 깨짐.

```
res.encoding = "utf-8"
```

개발자도구 — 화면 뒤를 보는 창

브라우저에 기본으로 들어 있음. 오늘 이걸로 알아낼 것은 딱 하나임 — **화면 주소로 값이 오는가, 아니면 다른 주소에서 오는가.**

여는 법은 **F12** 또는 **Ctrl+Shift+I** (Mac은 **⌘+⌘+I**) 임. 탭이 여럿이지만 오늘 쓰는 것은 두 개뿐.

ELEMENTS 탭

이름표를 확인함

가져오고 싶은 글자에 **오른쪽 클릭 → 검사**. 그 자리의 태그와 이름표(class·id)가 강조 됨. 실습 A에서 쓸 이름표를 여기서 얻음.

NETWORK 탭

값이 어디서 오는지 봄

브라우저가 서버와 주고받은 것이 전부 보임. 화면 주소가 아닌 **다른 주소**가 값을 배달하고 있다면 여기서 드러남. 실습 B가 이것.

I 오늘 하는 판단은 세 갈래뿐

01 **Ctrl+U** 로 원본을 열고 값을 찾아봄

있으면 ① **정적**. 화면 주소를 그대로 받아 이름표로 뜯으면 됨 — 실습 A.

02 없으면 **Network** 탭에서 «오는 자리»를 찾음

다른 주소가 값을 배달하고 있으면 ② **숨은 API**. 그 주소를 직접 부름 — 실습 B.

03 그것도 없으면 눌러야 나오는 것

③ **브라우저 자동화**. 사람 대신 브라우저를 조종함 — 실습 C.

Network 탭을 뒤지는 손놀림은 **화면으로 함께 봄**. 외워야 할 절차가 아니라 **몇 번 해 보면 익는 일**. 지금 가져갈 것은 «화면 주소 말고 다른 주소가 있을 수 있다»는 사실 하나.

Part 03

직접 해 보기

코랩을 열고 하나씩 돌려 봄. 정적으로 두 곳을 해 본 뒤, 코드가 왜 달라지는지 짚고 동적으로 넘어감. 마지막에 같은 일을 AI에게 시켜 봄.

실습 A · 정적 크롤링

화면 주소를 그대로 받아서 이름표로 뜯는 길. 코랩에서 진행함. 정적

I 준비 — 저장할 자리를 먼저 만듦

```
from google.colab import drive
drive.mount('/content/drive')
import os
from datetime import datetime
SAVE_DIR = '/content/drive/MyDrive/mypython'
os.makedirs(SAVE_DIR, exist_ok=True)      # 없으면 FileNotFoundError
TODAY = datetime.now().strftime('%y%m%d') # 파일 이름에 붙일 수집 날짜
```

I A-1 · Yes24 — 코드가 하는 일은 네 가지뿐

01 받아 옴

requests.get(URL) → <Response [200]>이면 성공. 돌아오는 것은 HTML 글자 문치.

02 뜯을 수 있게 바꿈

BeautifulSoup(...). 태그를 따라 이동할 수 있는 형태가 됨.

03 이름표로 골라냄

find_all('div', class_='itemUnit') → 24개. 8절에서 본 그 이름표.

04 표로 만들어 저장함

pd.DataFrame → yes24_best_260827.xlsx

정적 크롤링은 이 네 줄이 전부. 사이트가 바뀌어도 뼈대는 같고, 달라지는 것은 이름표뿐.

I A-2 · 멜론 — 이름표를 여러 겹 따라 내려가기

Yes24는 이름표 하나로 값을 꺼냈음. 멜론은 한 겹 더 들어가야 함. 곡 하나의 정보가 **여러 상자에 나뉘어** 담겨 있기 때문.

```
<tr class="lst50">           ← 곡 1개 = 행 하나
  <span class="rank">1</span>   ← 순위
  <div class="wrap_song_info"> ← 곡 정보를 담은 바깥 상자
    <div class="rank01"><a>곡명</a></div>
    <div class="rank02"><a>아티스트</a></div>
    <div class="rank03"><a>앨범</a></div>
  </div>
</tr>
```

곡명을 꺼내려면 **세 번을 타고 내려가야 함** — wrap_song_info 상자로 들어가서 → rank01 칸을 찾고 → 그 안의 a 태그. 이것을 한 줄로 쓴 것이 아래.

```
div.wrap_song_info div.rank01 a
```

■ 바깥 상자 — 곡 정보 전체 ■ 그 «안에 있는» 곡명 칸 ■ 그 «안에 있는» 링크 — 여기에 곡명이 들어 있음

띄어쓰기가 «그 안에 있는»이라는 뜻. 11절에서 본 그 규칙. 이름표 하나로 안 될 때는 **바깥부터 차례로 짚어 내려가면 됨.**

주. 순위는 span.rank 하나로 꺼냄. 같은 행 안이라도 값마다 깊이가 다를 수 있음.

I A-2 · 코드로 옮기면

```
rows = soup.select("tr.lst50, tr.lst100")      # 행 100개를 전부 찾음

for row in rows:                               # 한 행씩 돌면서
    rank = row.select_one("span.rank").get_text(strip=True)
    title = row.select_one("div.wrap_song_info div.rank01 a").get_text(strip=True)
    ...
```

- select는 여러 개, select_one은 하나 — 행은 여러 개라 select, 한 행 안의 곡명은 하나라 select_one
- row.select_one(...)처럼 **행 안에서** 찾는 것이 중요. soup에서 찾으면 100곡이 뒤섞임.
- get_text(strip=True)는 앞뒤 공백을 떼고 글자만 꺼내는 것
- **헤더 없이 부르면 406으로 거절당함** — 12절에서 본 그 헤더. 실습 코드는 UA와 Referer를 함께 붙임

주.melon_top100_260827.xlsx로 저장한 뒤 **저장한 경로가 메시지로 찍힘**. 이어서 표를 찍어 앞뒤 몇 줄을 눈으로 확인하고 넘어감.

코드가 길어 보이지만 하는 일은 **Yes24와 같음** — 반복되는 덩어리를 찾고, 하나씩 돌면서, 그 안에서 값을 꺼냄. 달라진 것은 «**몇 겹을 타고 내려가느냐**»뿐.

왜 코드가 달라지는가

여기서부터 코드 모양이 바뀔. 문법이 어려워져서가 아니라 **값이 오는 자리가 달라서**.

I 브라우저는 «두 번» 일하고, 파이썬은 «한 번»만 함

브라우저 · 두 단계

받고 → 그 뒤에 채움

① 서버에서 HTML을 받음. ② 그 안의 **자바스크립트를 실행**해서 값을 뒤늦게 채워 넣음.
우리가 보는 화면은 ②까지 끝난 뒤의 모습.

파이썬 · 한 단계

받기만 하고 끝

`requests.get()`은 ①만 함. **자바스크립트를 실행하지 않음**. 그래서 ②에서 채워질 값은 애초에 보이지 않음.

I 그래서 세 갈래로 갈림

TRACK 01

① 정적

받은 글자 뭉치에 값이 이미 들어 있음.

판별	Ctrl+U에 값이 보임
코드	받아서 이름표로 뚝음
오늘 실습	Yes24 · 멜론

TRACK 02

② 숨은 API

값은 뒤이어 따로 배달됨. 그 자리를 직접 부름.

판별	Ctrl+U에 값이 없음
코드	다른 주소를 부름
오늘 실습	메가박스 · KOBIS

TRACK 03

③ 브라우저 자동화

눌러야 나타남. 사람 대신 브라우저를 조종함.

판별	클릭·스크롤해야 나옴
코드	브라우저를 띄움
오늘 실습	AI로 해 봄

판별은 `Ctrl+U` 하나로 함 — 원본을 열고 `Ctrl+F`로 값을 찾아봄. 있으면 ①, 없으면 ②나 ③임. ★ 단 주소가 다르면 답도 다름 — 같은 메가박스라도 메인에는 영화 제목이 원본에 있고 /movie에는 없음.

I 주소가 다르면 코드도 다름 — 오늘 실습의 실제 주소

대상	화면에서 보는 주소	코드가 부르는 주소	갈래
Yes24	yes24.com/.../bestseller	같은 주소	정적
멜론	melon.com/chart/index.htm	같은 주소	정적
메가박스	megabox.co.kr/movie	.../selectMovieList.do	숨은 API
KOBIS	kobis.or.kr/.../findRealTicketList.do	같은 주소에 조건을 실어 보냄	숨은 API
네이버뉴스	search.naver.com/...	s.search.naver.com/.../more	숨은 API

주. KOBIS는 주소가 같지만 그냥 부르면 표가 비어 옴. 조건을 실어야 채워짐. «주소가 다르다»와 «조건을 실어야 한다»는 둘 다 ② 갈래임 — 공통점은 화면을 그대로 받아서는 안 된다는 것.

I 코드는 이렇게 갈림

① 정적

받아서 «뜯음»

```
r = requests.get("화면 주소")
soup = BeautifulSoup(r.text, ...)
soup.select("div.itemUnit")
```

이름표를 알아야 함. 07·08·11절이 여기서 쓰임.

② 숨은 API

부르면 «바로 옴»

```
r = requests.post("진짜 주소", json=조건)
r.json()["movieList"]
```

이름표가 필요 없음. 대신 «어느 주소를 부를지»를 개발자도구로 먼저 알아냄.

어려움의 자리가 서로 다름 — 정적은 «뜯는 것»이 일이고, 숨은 API는 «찾는 것»이 일. 찾기만 하면 뜯을 게 없어 오히려 쉬움.

실습 B · 숨은 API

값이 오는 자리를 직접 부르는 길. 파싱이 아예 필요 없음. 숨은 API

I B-1 · 메가박스 — 판별부터 해 봄

먼저 megabox.co.kr/movie를 그대로 받아 **영화 제목이 한 건도 없음**을 직접 확인함.

```
# ① 화면에 보이는 그 주소 — 목록이 없음
r = requests.get("https://www.megabox.co.kr/movie")
print(len(r.text))          # 100,473 자 — 뭔가 잔뜩 왔는데
print("movie-list-info" in r.text)  # False — 목록은 없음
```

```
# ② 개발자도구에서 찾은 «진짜 배달 주소»
r = requests.get("https://www.megabox.co.kr/on/oh/oha/Movie/selectMovieList.do",
                 params={"currentPage": 1, "recordCountPerPage": 20},
                 headers=headers)
for m in r.json()["movieList"]:
    print(m["movieNm"], m["boxoBokdRt"])  # 제목, 예매율
```

주. 선택자도 태그도 없음 — 키 이름만 알면 됨. 그 이름은 개발자도구에서 확인한 것.

I B-2 · KOBIS — 응답이 JSON이 아니라 «표»인 경우

같은 ② 갈래인데 돌아오는 형태가 다름. 메가박스는 JSON을 주고 KOBIS는 **HTML 표**를 통째로 줌. 그래도 하는 일은 같음.

	메가박스	KOBIS
응답 형태	JSON	HTML 표
꺼내는 법	<code>r.json()</code> 로 딕셔너리	<code>pd.read_html()</code> 로 표 한 방
선택자 필요?	없음(키 이름)	없음(표를 통째로)
주. 둘 다 파싱이 필요 없음. 태그를 따라 내려가는 일은 ① 갈래에서만 함.		

```
res = requests.post(url, data=조건, headers=headers)
res.encoding = "utf-8" # 한글이 깨질 때
df = pd.read_html(StringIO(res.text))[0] # 표를 통째로 읽음
```

확인할 것 — megabox_movies_260827.xlsx와 kobis_realtime_260827.xlsx 두 파일이 생겼는지 봄.

주. 조건에 CSRFToken이라는 값이 들어 있으나 빈 값으로 두어도 됨.

네이버뉴스 — 고치며 쓰는 일

여기서 배울 것은 새 기법이 아님. 크롤링 코드는 «한 번 짜면 끝»이 아니라 «고쳐 가며 쓰는 것»이라는 사실.

무슨 일이 생기나

어제 되던 코드가 오늘 0건

사이트는 개편됨. 특히 큰 포털일수록 자주 바뀜. 이름표(class)가 바뀌면 코드는 그대로 인데 결과만 비어 나옴. 네이버가 대표적.

어떻게 알아채나

★ 0건이 신호

에러가 안남. 문법이 맞으니 파이썬은 조용히 끝냄. 그래서 몇 건 나왔는지 매번 눈으로 보는 것이 유일한 방어.

I 그래서 함수를 미리 만들어 둬

- 노트북의 수집 함수는 120줄 남짓. 읽으라고 준 것이 아님
- 바꾸는 것은 검색어 · 언론사 · 시작일 · 종료일 · 정렬 · 페이지 수 여섯 줄
- 이 함수가 오늘 돌아간다는 것은 오늘 기준으로 손봐 두었다는 뜻

```
query      = '올림픽'      # ← 검색어
media      = 'YTN'         # ← 언론사
date_from  = 20260820      # ← 시작일
...
pages      = 3              # ← 10건 × 3페이지 = 최대 30건
```

고치는 법은 둘 — ① 개발자도구로 지금의 이름표를 다시 봄 ② AI에게 «0건이 나온다. 지금 HTML을 받아서 선택자를 다시 확인하고 고쳐 줘»라고 함. 고칠 자리만 알면 되지 120줄을 읽을 필요는 없음.

AI와 함께 크롤링하기

지금까지는 코드를 직접 읽고 돌렸음. 같은 일을 **한국어로 지시해** 시켜 봄.

I 무엇을 AI에게 맡기나

맡기는 것

코드를 쓰는 일

선택자를 찾고, 헤더를 붙이고, 표로 만들고, 엑셀로 저장하는 일. 브라우저 자동화처럼 코랩이 못 하는 것까지.

내가 하는 것

판단하는 일

어느 사이트를 볼지, 무엇을 가져올지, **결과가 맞는지**. 특히 «몇 건 나왔나»는 사람이 확인 함.

I 준비 — 3일차에서 세팅한 그대로

01 폴더를 엽

VS Code에서 **File > Open Folder** → 다운로드 폴더 안의 mypython. 만들어지는 파일이 전부 여기 쌓임.

02 AI를 부름

Terminal > New Terminal → claude (Codex는 codex, Gemini는 gemini)

03 규칙을 먼저 정함

배포 TXT의 [0-1]을 맨 처음에 한 번 붙여넣음. «선택자를 추측하지 말 것»·«실행해서 결과를 보일 것» 같은 규칙이 들어 있어 그 대화 내내 적용됨. 매번 다시 쓰지 않아도 됨.

주. 코랩의 mypython은 구글 드라이브 안, 여기서 여는 것은 내 컴퓨터 다운로드 폴더 안. 이름만 같고 다른 자리.

I ① 먼저 — 코랩에서 한 것을 그대로 시켜 봄

같은 결과가 나오는지 확인하는 것이 목적. 코랩에서 사람이 찾아 넣은 선택자를 AI가 스스로 찾아내는지 봄. 다섯 곳 전부에 대한 프롬프트가 배포 TXT에 들어 있음.

멜론 재현

TXT [1-2]

https://www.melon.com/chart/index.htm

이 페이지에서 TOP100 차트의 순위, 곡명, 아티스트, 앨범 을 수집해서 melon_top100_260827.xlsx 로 저장해 줘.

끝내지 말고 실행해서 몇 곡이 수집됐는지와 상위 5행, 마지막 5행을 보여 줘.

대상	코랩 결과	AI도 같은 결과가 나오는지
Yes24	24권 6열	TXT [1-1]
멜론	100곡 4열	TXT [1-2]
메가박스	20편	TXT [1-3]
KOBIS	300행 넘게	TXT [1-4]
네이버뉴스	최대 30건	TXT [1-5]

주. 몇 건이 나와야 정상인지를 알고 시작하는 것이 검증 장치. 숫자가 다르면 AI가 어딘가 틀린 것.

여기까지가 «따라 해 보기». 결과가 같다면 다음으로 넘어감 — 이제 내가 하고 싶은 사이트 차례.

I ② ★ 아무 사이트나 되는 프롬프트 — 빈칸 셋만 채움

사이트마다 프롬프트를 새로 만들 필요가 없음. 아래 하나면 됨. 나머지 조건은 [0-1]에 이미 깔려 있으니 < > 안만 바꿔 씀.

범용 크롤링 프롬프트

TXT [2-1]

<여기에 주소를 넣으세요> 이 페이지에서
<가져오고 싶은 것을 적으세요> 를 수집해서
<파일이름>.xlsx 로 저장해 줘.

먼저 이 페이지가 정적인지 동적인지 확인해 줘.

- 정적이면 requests 와 BeautifulSoup 으로,
- 값이 나중에 채워지면 그 값이 오는 주소를 찾아서,
- 클릭이나 스크롤을 해야 나오면 브라우저를 띄워서 해 줘.

끝내지 말고 실행해서 몇 건이 수집됐는지와
상위 5행, 마지막 5행을 보여 줘.

이 하나가 세 갈래를 다 덮음. 어느 갈래인지는 AI가 확인해서 정함 — 우리가 배운 판별을 AI에게 시키는 것. 그래도 «몇 건인지»는 내가 봄.

I ③ 브라우저 자동화 — 코랩이 못 하는 것

- 클릭·스크롤을 해야 값이 나오는 페이지는 코랩에서 못 함 — 화면이 없기 때문. 바이브 코딩에서는 창이 실제로 열림
- 창이 열리는 것을 보는 것이 목적. 한 번 보면 브라우저 자동화가 무엇인지 이해됨
- 처음에 실패해도 그대로 둘 것 — AI가 에러를 읽고 다시 시도하는 과정이 불거리

주. 스크롤 프롬프트는 TXT [2-2]. 수집이 끝이 아님 — 모은 뒤 «더 가져올 것 물어보기»«통계와 그림»은 TXT 3장 [3-1] · [3-2].

법 · 윤리 · 서버 예의

크롤링은 남의 컴퓨터에 부담을 지우는 일. 기술보다 이 감각이 먼저임.

01

먼저 확인함

주소 뒤에 /robots.txt를 붙여 열어 봄. 수집을 막아 둔 경로가 적혀 있음. 이용약관도 함께 봄.

예: <https://www.melon.com/robots.txt>

02

천천히 함

요청 사이에 최소 1초를 씬. 사람이 클릭하는 속도보다 빠를 이유가 없음. 429가 뜨면 이미 늦은 것.

```
import time
time.sleep(1)
```

03

덜 가져옴

필요한 만큼만 가져옴. «일단 다 받아 두자»는 대개 쓰지도 않으면서 상대방에게 부담만 줌.

I 수집보다 재배포가 문제

- **저작권** — 기사 본문은 저작물. 연구·학습을 위해 모아서 분석하는 것과, 그대로 다시 퍼뜨리는 것은 전혀 다른 문제
- **개인정보** — 댓글·작성자 정보에는 개인정보가 섞여 있음. 필요 없으면 처음부터 수집하지 않는 것이 안전함
- **연구 윤리** — 논문에 쓸 자료라면 수집 시점·조건·건수를 기록해 둬. 나중에 재현할 수 없으면 자료로 쓸 수 없음
- **로그인 영역** — 계정으로 들어가야 보이는 것은 이용약관 위반이 되기 쉬움. 오늘 다루지 않는 이유

여럿이 동시에 같은 사이트를 부르면 — 같은 공간에서 나가는 요청은 상대 서버에 **하나의 IP**로 보임. 순식간에 수백 건이 몰려 차단될 수 있음. 오늘 실습에서 반복 횟수를 작게 잡아 둔 이유. 집에서는 늘려도 되나, **그때도 1초는 쉬어야 함**.

오늘의 마무리

Q & A

문의 — 이문혁 (경희대학교 인문사회과학데이터연구소)

`dalvitt@khu.ac.kr`