

03 DOM과 이벤트소개(CLI 실습)



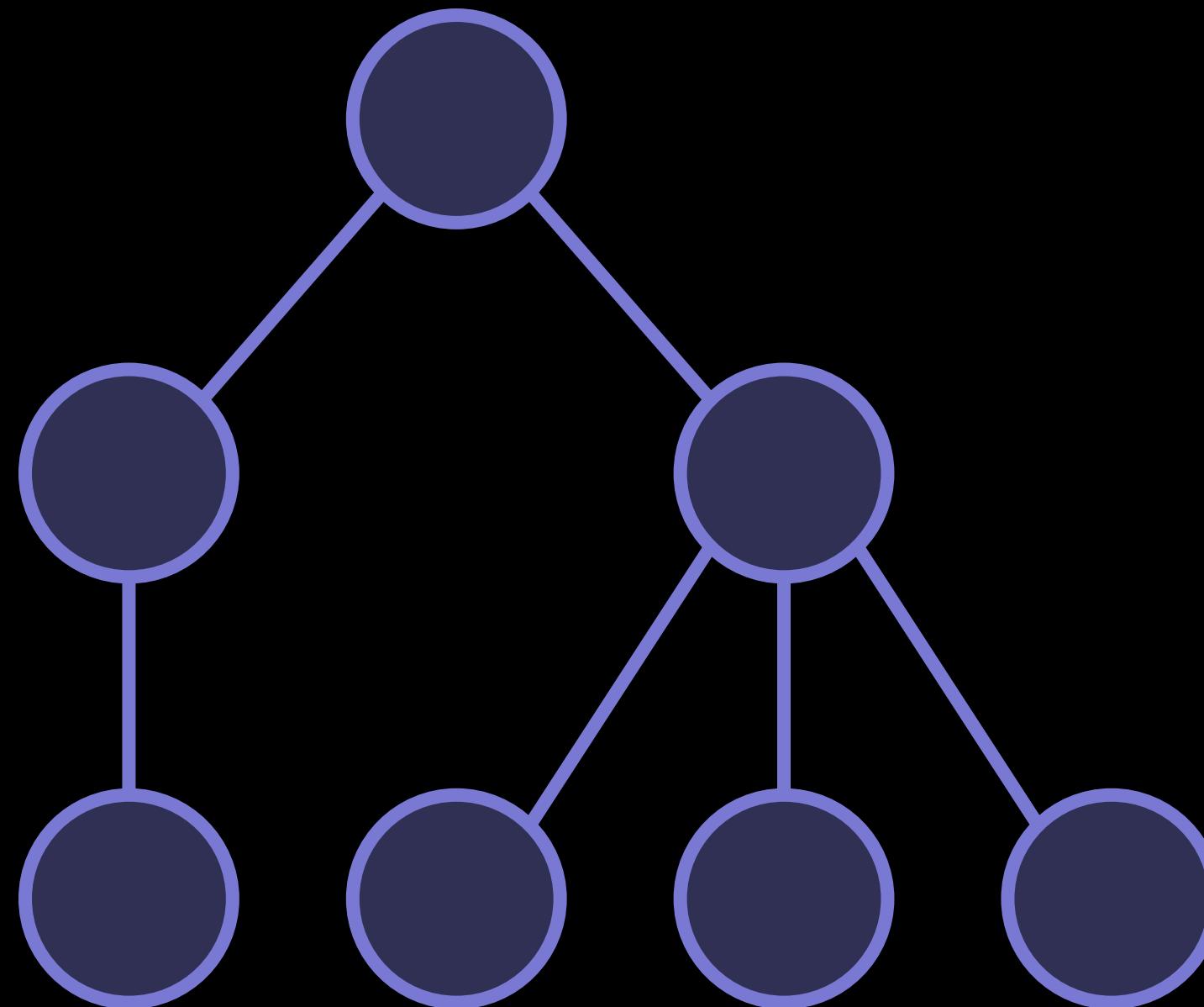
Object Model이란?

어떤 대상을 "객체"로 표현하도록 설계된 규칙 혹은 구조

- 객체 = 데이터 + 행동
- 행동은 "메서드"를 의미

객체라는 개념을 이해해보자

- 속성 (Property)
- 행동 (Method)





Object Model이란?

자동차를 객체로 만든다면?



속성

색상, 속도, 브랜드, 연료량...

행동

Accelerate(), brake(), honk() ...

즉, 자동차라는 개념을 프로그래밍적으로 다룰 수 있게 만드는

"표현 방식" 그 자체를 Object Model이라고 함



Object Model 을 알아보자

- Object Model 을 문서(Document)에 적용하면? (=DOM)

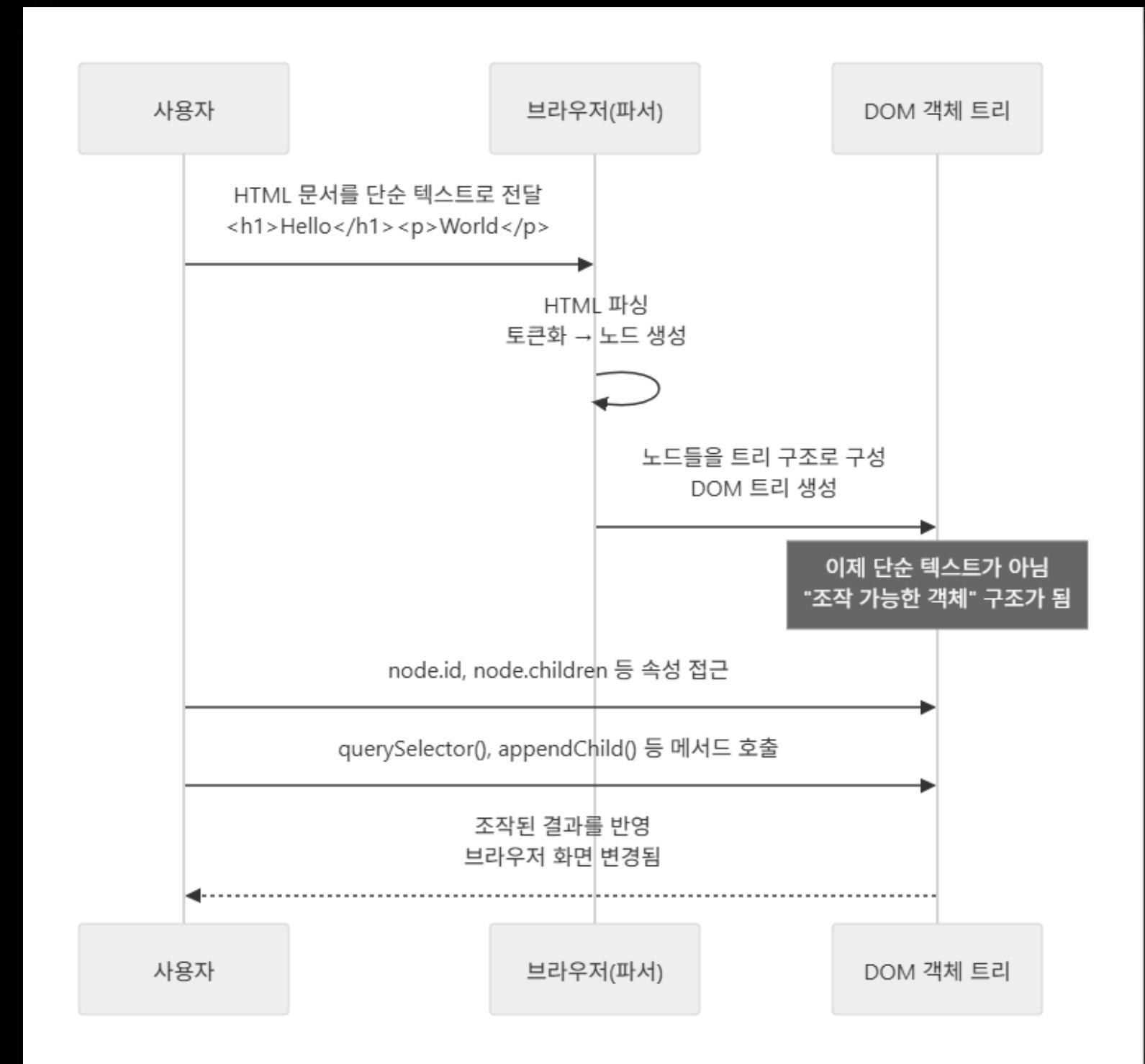
```
<h1>Hello</h1>  
<p>World</p>
```

- 이렇게만 써있으면 그냥 파일이다
 - 이 텍스트는 사람이 읽을 수 있을 뿐 기계가 뭘 해낼 수 있는 상태가 아니다.
 - 단순 텍스트만으로는 브라우저가 그림을 그려낼 수가 없다.
 - 그래서 브라우저 내에 있는 "파서"를 사용해서 "HTML 파싱"을 하게된다.



Object Model 을 알아보자

- HTML 텍스트가 문서 객체 모델(DOM)로 변환되는 자세한 과정입니다.



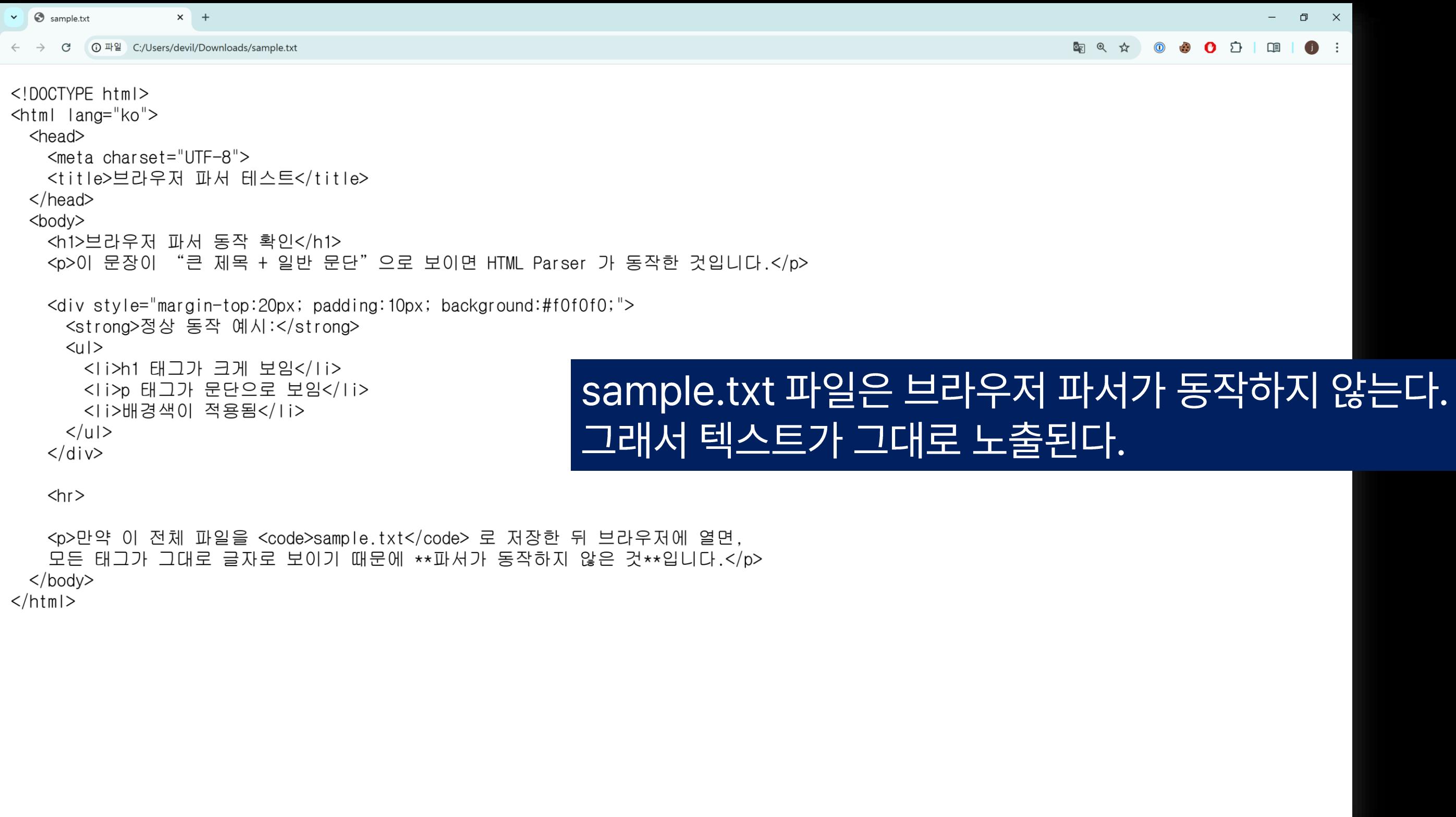


[간단 실습] 브라우저 파서

1. 컴퓨터에 sample.txt 파일을 하나 생성하세요.
2. 컴퓨터에 sample.html 파일을 하나 생성하세요.
3. 이 두개의 파일에 이 [링크](#)에 있는 코드를 넣어서 저장해주세요.
 - 링크가 안보이면? <https://kdt-gitlab.elice.io/-/snippets/11>
4. 이제 각 파일을 브라우저에 드래그 & 드랍해보세요.

[간단 실습] 브라우저 파서

- "sample.txt" 파일은 확장자가 ".txt" 이기 때문에 브라우저 파서가 동작하지 않습니다.



The screenshot shows a web browser window with the title bar "sample.txt". The address bar indicates the file is located at "C:/Users/devil/Downloads/sample.txt". The browser interface includes standard buttons for back, forward, search, and refresh. The main content area displays the following HTML code:

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8">
    <title>브라우저 파서 테스트</title>
  </head>
  <body>
    <h1>브라우저 파서 동작 확인</h1>
    <p>이 문장이 “큰 제목 + 일반 문단”으로 보이면 HTML Parser 가 동작한 것입니다.</p>

    <div style="margin-top:20px; padding:10px; background:#f0f0f0;">
      <strong>정상 동작 예시:</strong>
      <ul>
        <li>h1 태그가 크게 보임</li>
        <li>p 태그가 문단으로 보임</li>
        <li>배경색이 적용됨</li>
      </ul>
    </div>

    <hr>

    <p>만약 이 전체 파일을 <code>sample.txt</code>로 저장한 뒤 브라우저에 열면,<br>
    모든 태그가 그대로 글자로 보이기 때문에 **파서가 동작하지 않은 것**입니다.</p>
  </body>
</html>
```

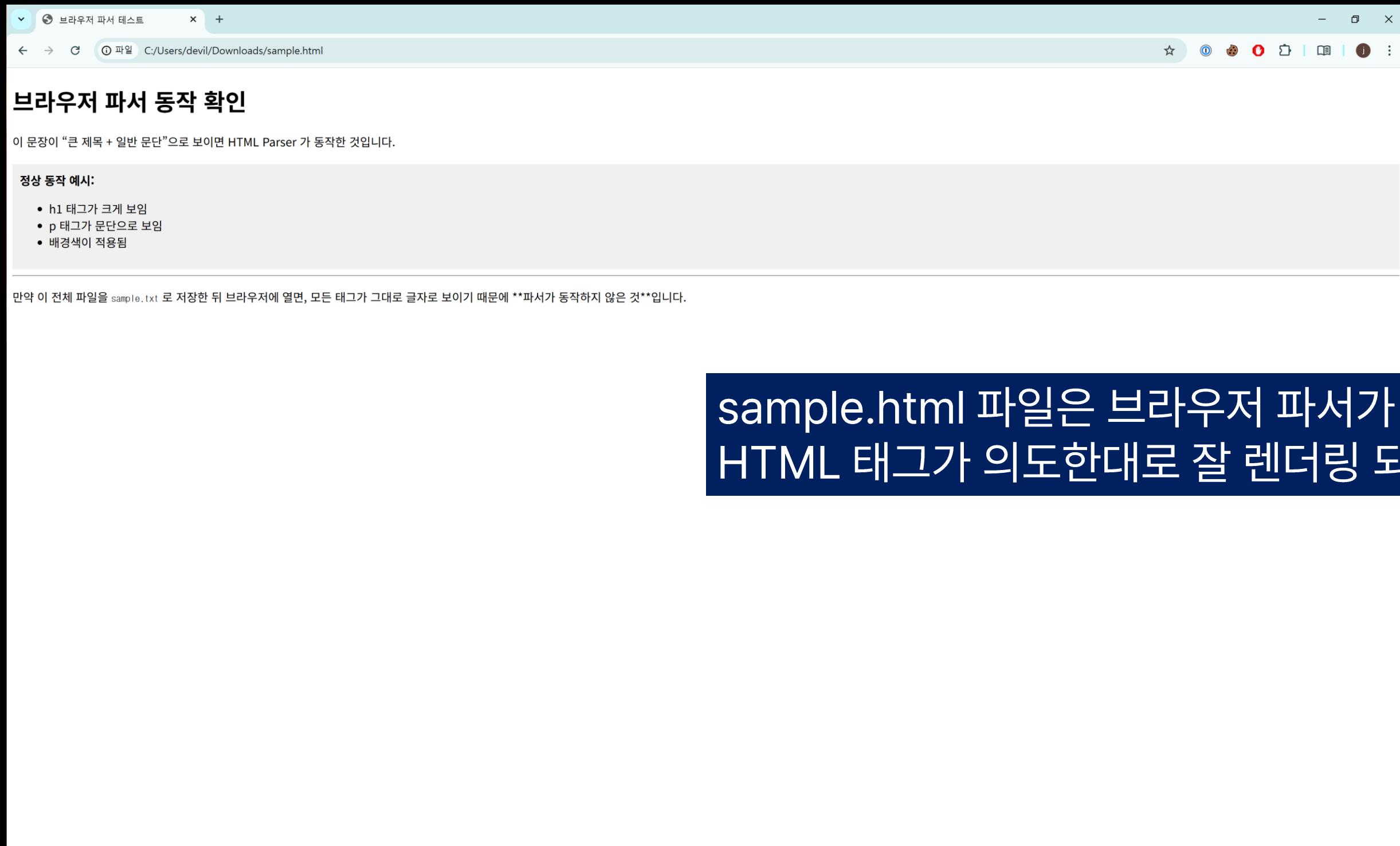
A blue callout box on the right side of the browser window contains the following text:

sample.txt 파일은 브라우저 파서가 동작하지 않는다.
그래서 텍스트가 그대로 노출된다.



[간단 실습] 브라우저 파서

- "sample.html" 파일은 확장자가 ".html" 이기 때문에 브라우저 파서가 동작해서 의도한대로 페이지가 표시되는 것을 볼 수 있습니다



sample.html 파일은 브라우저 파서가 동작한다.
HTML 태그가 의도한대로 잘 렌더링 되는 것을 볼 수 있다.



간단 실습 수행 후 결과 분석

- "sample.html" 문서에 있는 "태그"는 하나하나 모두가 객체이다.
 - 그리고 이 객체 하나하나에는 모두 "속성"과 "행동(=메서드)"가 포함된다.
 - 태그 하나가 감싸고 있는 '자식' 태그는 중첩 태그 (Nested tag) 라고 함
 - 태그 안에 있는 텍스트 내의 문자 역시 객체이다.
- 이 모든 객체는 자바스크립트를 통해 접근할 수 있다.
 - JS로 접근이 된다는 것은 원하는대로 "조작"이 가능하다는 것
- 브라우저 콘솔에서 `document.body` 에 접근할 수 있다
 - 이건 <body> 태그를 자바스크립트 객체로 나타낸 것이다.



[간단 실습] 네이버의 배경화면 색을 바꿔보자

- 이제 다시 원래대로 돌리기 위해서 아래 코드를 실행하자

```
document.body.style.background = "#"; // 배경을 블루색으로 변경하기
```

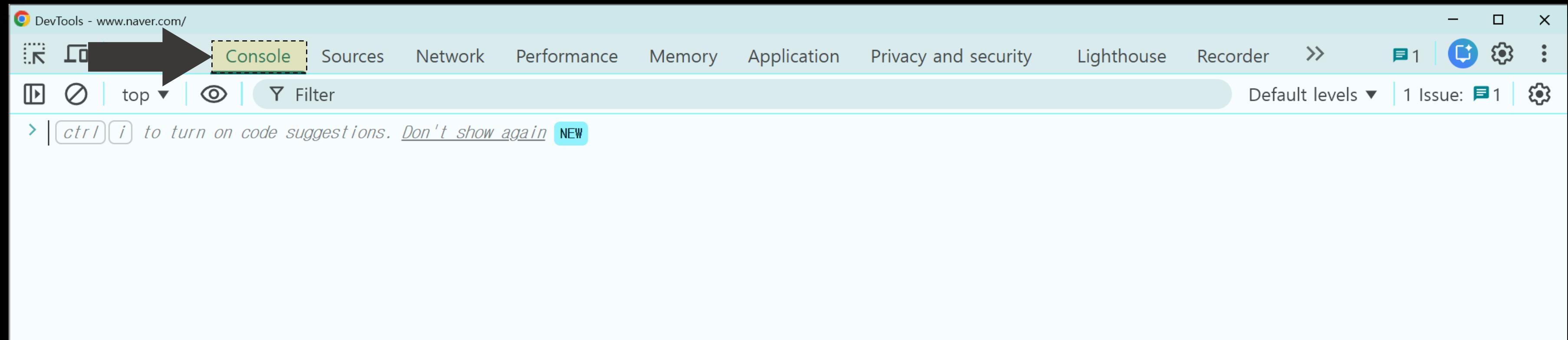
- 위 코드를 실행해서 원래대로 돌려도 되지만 브라우저를 새로고침해서 된다.
- 서버에서 원래 코드를 다시 받아오면 없던 일이 되는 것이다.

- 개발자도구 콘솔탭에서 명령어를 칠 때 방향키를 위 아래로 눌러보자
 - 이전에 작성했던 명령어가 순서대로 나온다.
 - 같은 코드를 여러번 치지 말자



[간단 실습] 네이버의 배경화면 색을 바꿔보자

- 네이버에 접속해서 개발자 도구를 실행
 - 개발자 도구에서 콘솔탭으로 이동



- 콘솔 탭에서 아래 코드를 실행하자
 - 보안상의 이유로 콘솔 탭에서는 코드를 복사 붙여넣기 할 수 없음.
 - 직접 타이핑 해야한다

```
document.body.style.background = 'red'; // 배경을 블루색으로 변경하기
```



간단 실습 후 결과 분석

- 아래 코드를 잘 보자

```
document.body.style.background = 'red'; // 배경을 붉은색으로 변경하기
```

- 분명 `document.body`는 <body> 태그라고 했다.
 - 그리고 JS로 이 태그에 적용되어 있는 '스타일 속성'(style property)을 '조작'한 것이다.
 - 브라우저 안에 DOM 객체 트리가 이미 만들어져 있어서 이게 가능한 것이다.
- `style.background` 외에도 엄청나게 많은 속성이 있다.
 - innerHTML - 해당 노드의 HTML 컨텐츠
 - offsetWidth - 해당 노드의 너비 (픽셀)
 - 등등.. 엄청나게 많은 속성을 JS로 조작 할 수 있다.



DOM 구조에 대해서

- 간단한 문서를 통해 DOM 구조를 알아보자

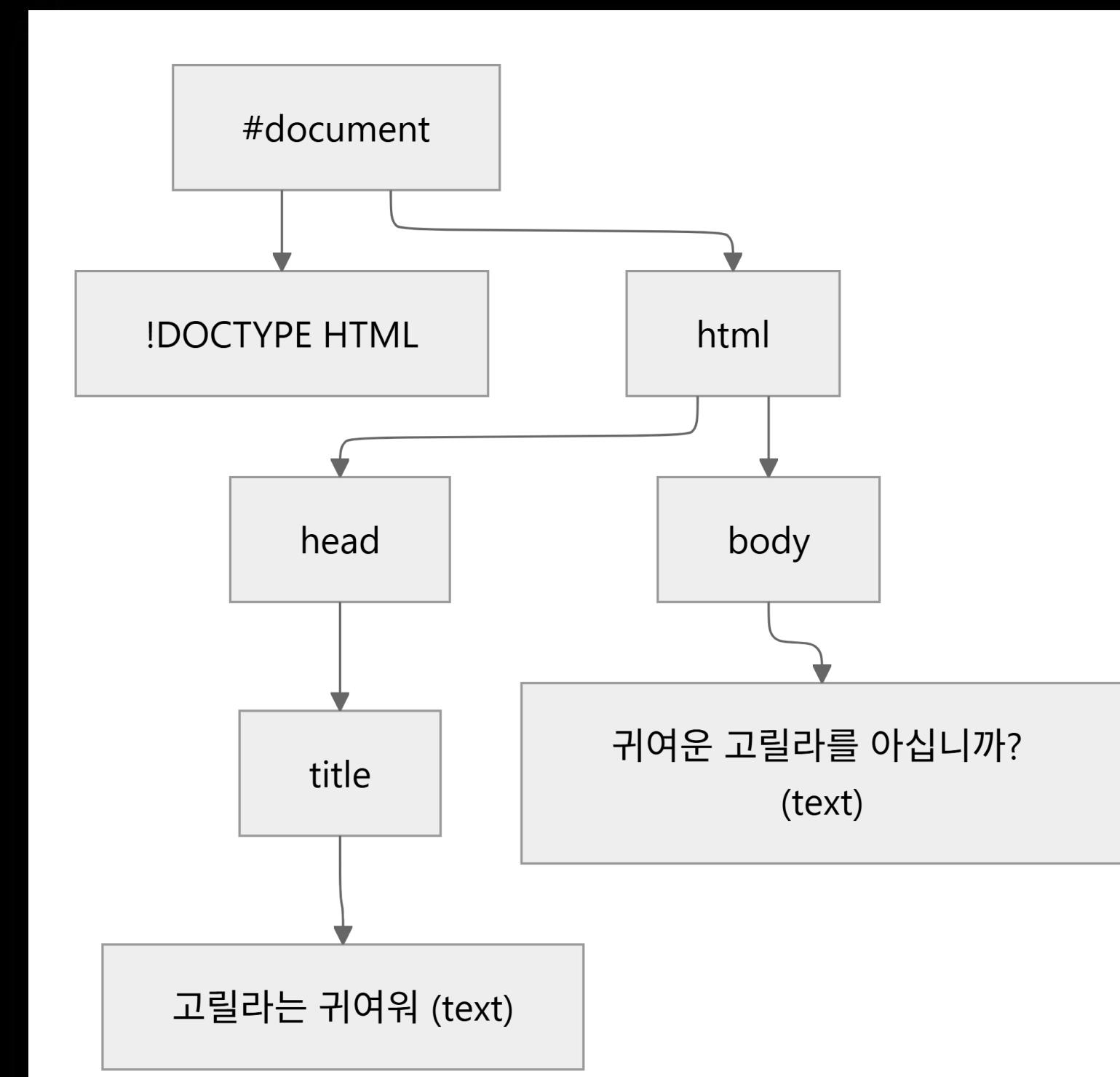
```
<!DOCTYPE HTML>
<html>
  <head>
    <title>고릴라는 귀여워</title>
  </head>
  <body>
    귀여운 고릴라를 아십니까?
  </body>
</html>
```

- 이 HTML을 DOM으로 표현해보면?



DOM 구조에 대해서

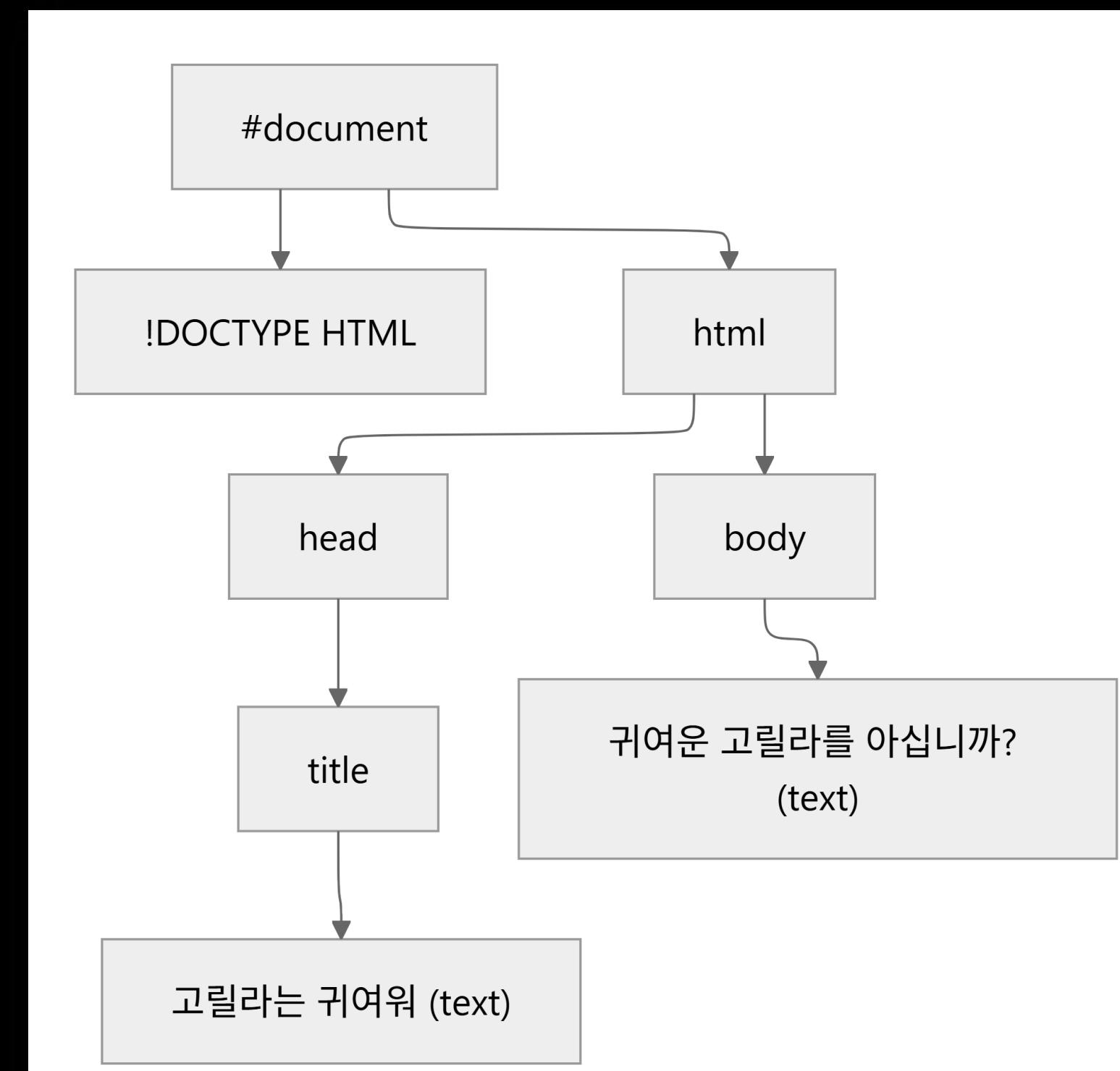
- 브라우저는 HTML을 읽으면 가장 먼저 'document' 객체를 만든다.
 - Javascript의 'document' 변수는 이 루트 노드를 가리킨다.





DOM 구조에 대해서

- DOM은 HTML 문서를 브라우저가 객체(노드)들로 변환한 결과물이다.
 - 즉, HTML(텍스트) → DOM(객체 트리)





DOM 구조에 대해서

- DOM 구조를 자세히 보고 싶다면?
 - <https://software.hixie.ch/utilities/js/live-dom-viewer/>
- 위 링크에 접속해서 이 [링크](#)에 있는 코드를 붙여넣기 해서 DOM을 확인해보자

Live DOM Viewer

Markup to test ([permalink](#), [save](#), [upload](#), [download](#), [hide](#)):

```
<hr>

<p>만약 이 전체 파일을 <code>sample.txt</code>로 저장한 뒤 브라우저에 열면,  
모든 태그가 그대로 글자로 보이기 때문에 **파서가 동작하지 않은 것**입니다.</p>
</body>
</html>
```

DOM view ([hide](#), [refresh](#)):

```
|- DOCTYPE: html
  |- HTML lang="ko"
    |- HEAD
      |- #text:
      |- META charset="UTF-8"
      |- #text:
      |- TITLE
        |- #text: 브라우저 파서 테스트
      |- #text:
    |- BODY
      |- #text:
      |- H1
        |- #text: 브라우저 파서 동작 확인
      |- #text:
      |- P
        |- #text: 이 문장이 “큰 제목 + 일반 문단”으로 보이면 HTML Parser가 동작한 것입니다.
      |- #text:
    |- DIV style="margin-top:20px; padding:10px; background:#f0f0f0;"
```



[간단 실습] 콘솔을 사용해서 DOM 다루기

- 개발자 도구를 사용해서 DOM을 탐색하다보면 자바스크립트를 적용해서 이런저런 값을 테스트 해봐야 하는 경우가 있음
 - DOM에 있는 노드를 즉시 수정하고 브라우저상에서 결과물을 바로 확인해 봐야 하는 경우
 - 이런 경우에서 쓸 수 있는 몇가지 방법에 대한 실습을 해보겠다.
- 우선 앞서 만든 sample.html 파일을 브라우저에 드래그 & 드랍해서 띄우자

브라우저 파서 동작 확인

이 문장이 “큰 제목 + 일반 문단”으로 보이면 HTML Parser 가 동작한 것입니다.

정상 동작 예시:

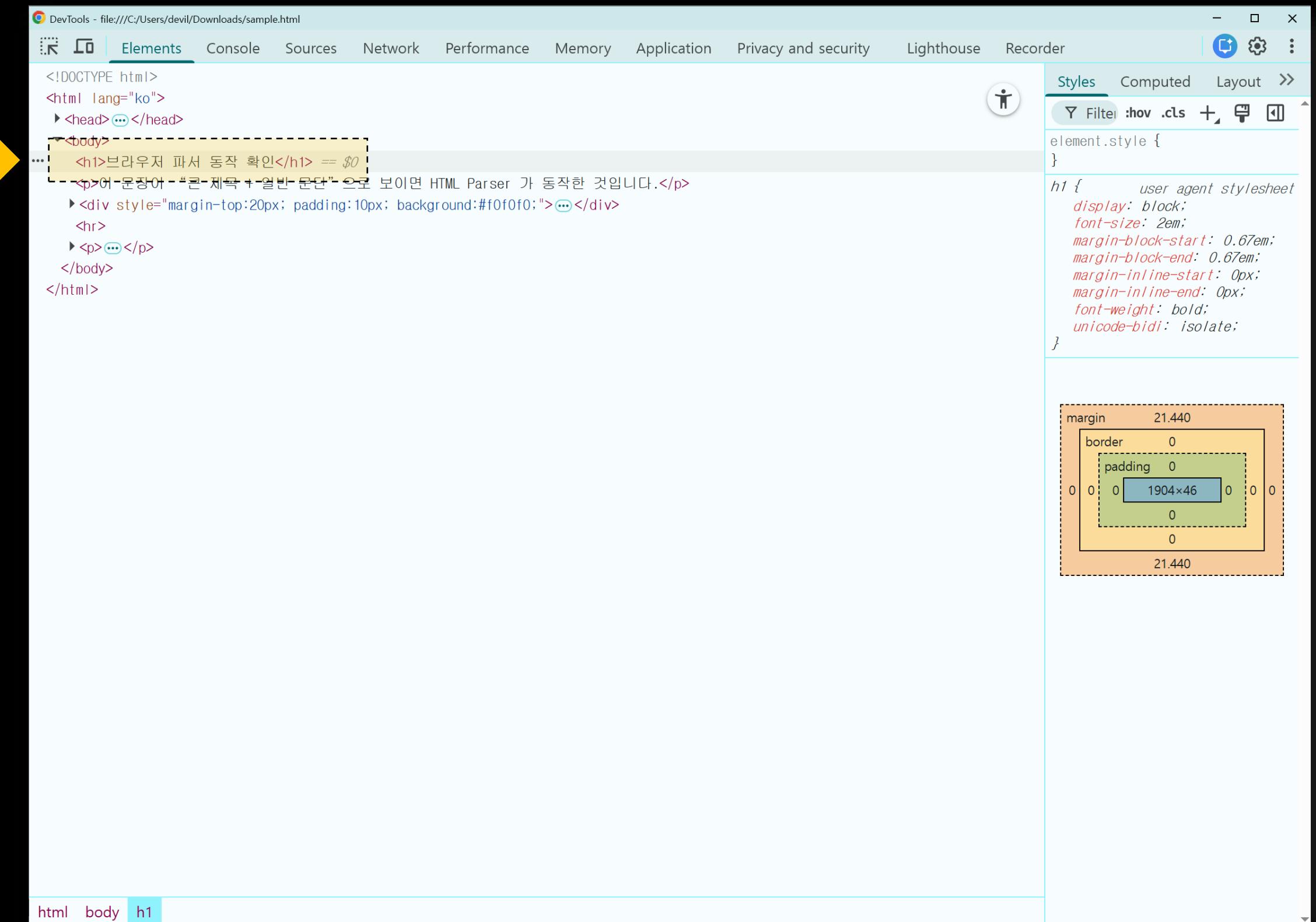
- h1 태그가 크게 보임
- p 태그가 문단으로 보임
- 배경색이 적용됨

만약 이 전체 파일을 `sample.txt` 로 저장한 뒤 브라우저에 열면, 모든 태그가 그대로 글자로 보이기 때문에 **파서가 동작하지 않은 것**입니다.



[간단 실습] 콘솔을 사용해서 DOM 다루기

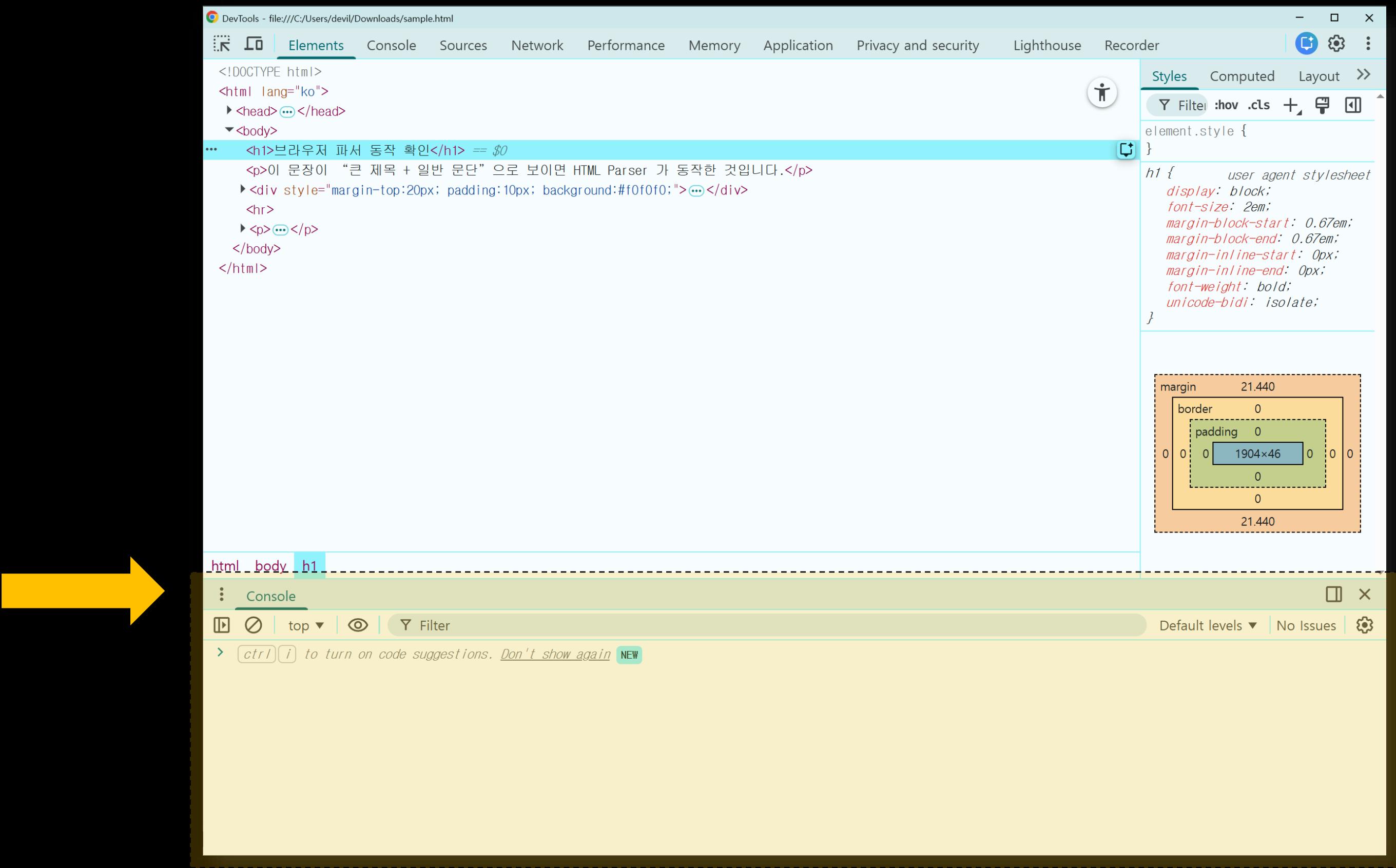
- 개발자 도구를 열어서 Elements 패널에서 `<h1>` 태그를 선택





[간단 실습] 콘솔을 사용해서 DOM 다루기

- ESC 를 눌러서 Elements 채널 아래에 콘솔을 띄우자





[간단 실습] 콘솔을 사용해서 DOM 다루기

- 이제 가장 마지막에 선택했던 요소는 \$0으로 접근 할 수 있다.
- 그 이전에 선택했던 요소는 \$1로 접근할 수 있다.

The screenshot shows the Google Chrome DevTools interface. On the left, the **Elements** tab is active, displaying the HTML structure of a sample.html file. A yellow dashed box highlights the **<body>** element, which contains an **<h1>** header and several **<p>** paragraphs. The **Console** tab on the right is also visible, showing the command `h1` and its output: `동작 확인</h1> == $0`. A large blue callout box points from the highlighted **<body>** element towards this console output, explaining that selecting the last element in the DOM results in the `$0` object. Another callout box at the bottom right provides a general note about testing selected elements.

동작 확인</h1> == \$0
제목 + 일반 문단” 으로 보이면 HTML Pa
n-tc
브라우저 내부적으로 마지막으로 선택한 요소는
자동으로 \$0 오브젝트에 지정된다.

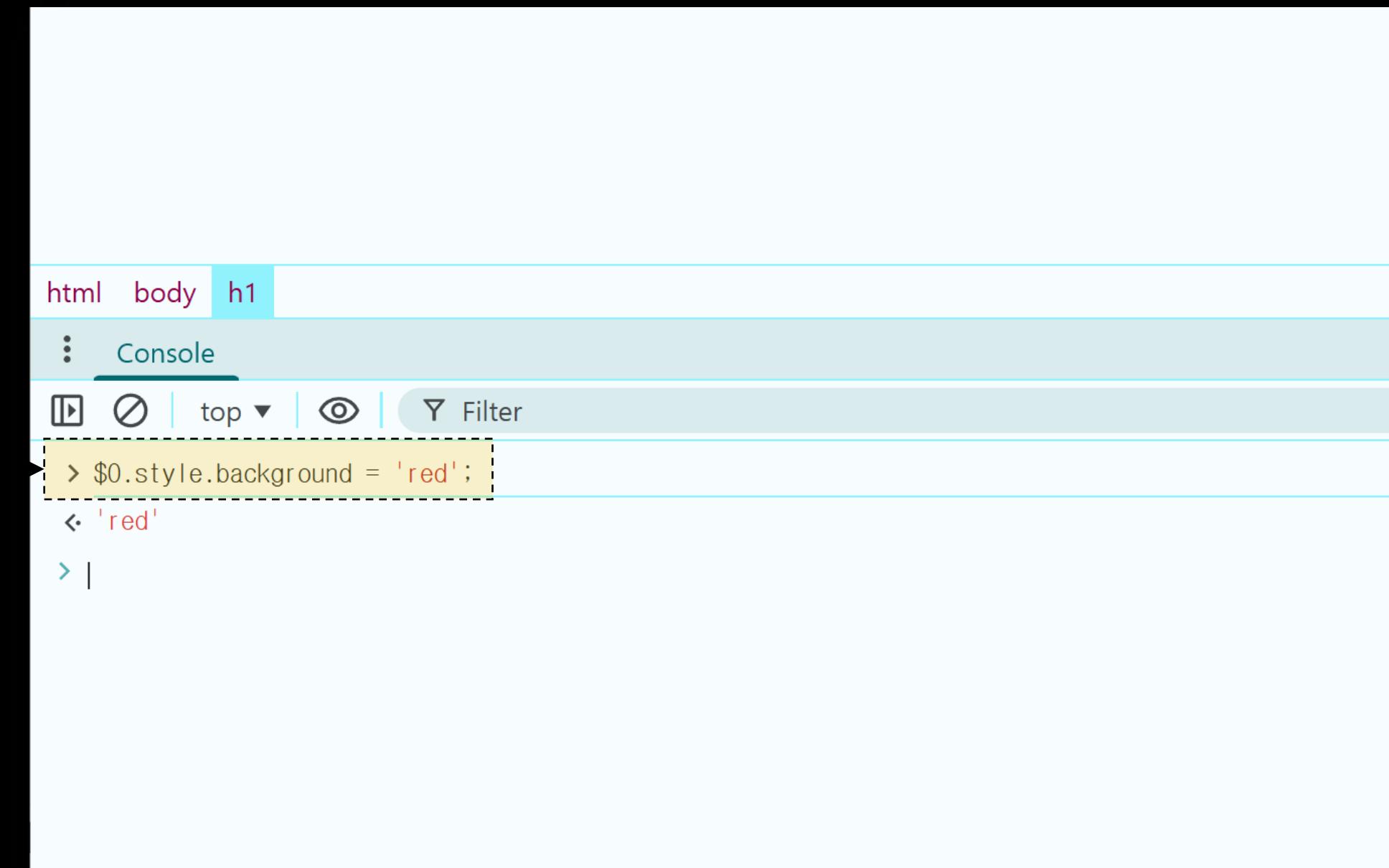
테스트 하고 싶은 객체를 선택해서 \$0 으로 접근하면
손쉽게 Javascript 로 다양한 반응 테스트를 해볼 수 있는 것



[간단 실습] 콘솔을 사용해서 DOM 다루기

- 이제 \$0 객체의 배경색 속성을 바꿔보자
 - <h1> 태그를 선택해서 \$0 객체에 할당 되도록 만들자
 - 그리고 콘솔 창에서 아래 코드를 타이핑 하자
 - 브라우저 보안이슈로 인해 콘솔창에는 코드를 붙여넣기 할 수 없다.
 - 직접 타이핑 하자

```
$0.style.background = 'red'
```





[간단 실습] 콘솔을 사용해서 DOM 다루기

- 그 결과 아래와 같이 DOM Tree 내 객체 중 <h1> 태그가 조작된 것을 확인 할 수 있다.

브라우저 파서 동작 확인

이 문장이 “큰 제목 + 일반 문단”으로 보이면 HTML Parser 가 동작한 것입니다.

정상 동작 예시:

- h1 태그가 크게 보임
- p 태그가 문단으로 보임
- 배경색이 적용됨

만약 이 전체 파일을 sample.txt 로 저장한 뒤 브라우저에 열면, 모든 태그가 그대로 글자로 보이기 때문에 **파서가 동작하지 않은 것**입니다.

|



[간단 실습] 콘솔을 사용해서 DOM 다루기

- 또 다른 방법을 하나 더 실습해보자
- DOM Node를 참조하는 변수가 있는 경우 콘솔에 아래 명령어를 입력해서 Elements 패널에서 해당 요소가 강조 된 것을 확인할 수 있다.

inspect(node)

- 위에서 `node`는 `'\$0'`, `document.body` 같은 DOM 내 node object 를 입력하면 된다.



[간단 실습] 콘솔을 사용해서 DOM 다루기

- `inspect(node)` 를 콘솔에 입력하면 아래와 같이 결과가 나온다

The screenshot shows a browser's developer tools console. The tabs at the top are 'html', 'body', and 'h1'. The 'h1' tab is active. Below the tabs is a toolbar with icons for play/pause, stop, top, and filter. The main area of the console shows the following interaction:

```
> $0.style.background = 'red';
< 'red'
> inspect($0)
< <h1 style="background: red;">브라우저 파서 동작 확인</h1>
```

A yellow arrow points from the text 'inspect 메서드의 인자 값으로 \$0 말고 다른 오브젝트도 넣어서 테스트해보자' to the 'inspect(\$0)' line in the console.

inspect 메서드의 인자 값으로 \$0 말고 다른 오브젝트도 넣어서 테스트해보자

생각해보기

- `document` 객체 하위에 HTML 태그가 모두 저장되어 있다.
- 다른 객체를 inspecting 하려면 document 객체에서 어떻게 접근해야 할까?
- 힌트 : document.querySelector(...), document.querySelectorAll(...)
 - 이 내용은 후반부에 다시 배운다



간단 실습 후 결과 분석

- 이걸 어디에 써먹을 수 있을까?
 - QA 자동화의 가장 큰 고충 중 하나가 바로 이런것이다
 - "내가 작성한 선택자(Selector)가 실제로 올바른 요소를 가리키고 있는가?"
 - 내가 작성한 자동화 코드 중 하나가 아래와 같다고 가정하자.

```
driver.findElement(By.cssSelector(".btn-submit"))
```

- 이 때 `inspect()` 를 사용해서 콘솔에서 먼저 검증함
 - 콘솔창에 ``inspect($('.btn-submit'))`` 을 입력 (\$는 document.querySelector의 단축어)
 - Elements 패널에서 해당 버튼의 HTML 코드가 하이라이트 됨
 - 만약 엉뚱한 곳이 하이라이트되거나 null이 뜣다면 자동화 스크립트에 넣기 전에 선택자를 수정해야 함을 바로 알 수 있다



간단 실습 후 결과 분석

- 'sample.html'에서 콘솔창을 띄워서 `$('strong')`을 실행해보자
- 그러면 해당 DOM 객체가 표시 되는 것을 볼 수 있다.

The screenshot shows the DevTools interface for a file:///C:/Users/devil/Downloads/sample.html page. The Elements tab is active, displaying the page's HTML structure. The Console tab is also visible at the bottom, with a yellow arrow pointing to it. In the Console, the command `$('strong')` is entered, and its result, `정상 동작 예시:`, is shown in a highlighted box. The Styles tab on the right shows the applied CSS rules for the element.

```
<!DOCTYPE html>
...<html lang="ko"> == $0
  > <head>...</head>
  > <body>
    <h1 style="background: red;">브라우저 파서 동작 확인</h1>
    <p>이 문장이 “큰 제목 + 일반 문단”으로 보이면 HTML Parser 가 동작한 것입니다.</p>
    > <div style="margin-top:20px; padding:10px; background:#f0f0f0;" id="jeff">
      <strong>정상 동작 예시:</strong>
      > <ul>...</ul>
    </div>
    <hr>
    > <p>...</p>
  </body>
</html>
```

```
html
::: Console
  ↳ top ▾ | ⚡ Filter
    > $('strong')
    <strong>정상 동작 예시:</strong>
  >
```

Styles Computed Layout >
element.style {
}
html[Attributes Style] {
 -webkit-locale: "ko";
}
:root { user agent stylesheet
 view-transition-name: root;
}
html { user agent stylesheet
 display: block;
}
margin 0



간단 실습 후 결과 분석

- 만약 엉뚱한 선택자를 넣는다면?

The screenshot shows the Google Chrome DevTools interface with the following panels:

- Elements Panel:** Displays the HTML structure of a sample.html file. The code includes an

element with a red background, a element with text about the parser, and a element with styling and an ID of "jeff".
- Console Panel:** Shows JavaScript console output. A yellow arrow points to the last line of the console, which is highlighted with a dashed box and contains the text: `> $('존재하지 않는 선택자')`. This indicates that an invalid selector was used.
- Styles Panel:** Shows the computed styles for various elements, including the root element and the

with id="jeff".



[간단 실습] Console, inspect() 실습

- 이 [링크](#)에 접속해서 HTML 코드를 다운로드 받아주세요
 - 링크가 눌리지 않는다면 아래 URL 을 직접 복사해주세요!
 - <https://kdt-gitlab.elice.io/-/snippets/12>
- 다운로드 받은 코드를 크롬 브라우저에 드래그 드랍해주세요
 - HTML 렌더링 되면 아래와 같은 화면이 됩니다.



QA를 위한 Console inspect() 실습

F12를 눌러 **Console 패널**을 열고 아래 시나리오를 테스트해보세요.

1. 자동화 선택자(Selector) 검증

아래 버튼을 콘솔에서 찾아보세요. (CSS Selector)

로그인 버튼

👉 입력: `inspect($('.btn-submit'))`



[간단 실습] Console, inspect() 실습

- CSS Selector 검증을 해보세요

1. 자동화 선택자(Selector) 검증

아래 버튼을 콘솔에서 찾아보세요. (CSS Selector)

로그인 버튼

👉 입력: `inspect($('.btn-submit'))`



[간단 실습] Console, inspect() 실습

- 제대로 적용 됐다면 아래와 같이 잘 표시가 되는 것을 확인 할 수 있다

The screenshot shows the Chrome DevTools interface. The top part displays the DOM structure with a highlighted element: `html body div.section button#login-btn.btn-submit`. To the right, the CSS panel shows the styles applied to this element, including `cursor: pointer;`, `button { user agent stylesheet }`, and `button { border: 1px solid #ccc; border-radius: 5px; padding: 10px; font-size: 16px; } button:disabled { background-color: #f0f0f0; }`. The bottom part shows the `Console` tab with the following output:

```
<code>👉 입력: inspect($('.btn-submit'))</code>
</div>
▶ <div class="section">...</div>
html body div.section button#login-btn.btn-submit
:
Console
Default levels ▾ No Issues
sample.html?_ijt=k3n…d=RELOAD_ON_SAVE:74
sample.html?_ijt=k3n…d=RELOAD_ON_SAVE:65
③ 이벤트가 연결되어 있습니다! (QA Pass)
> inspect($('.btn-submit'))
<button class="btn-submit" id="login-btn">로그인 버튼</button>
>
```

A yellow arrow points to the `inspect($('.btn-submit'))` command in the console, which is highlighted with a dashed box.



[간단 실습] Console, inspect() 실습

- React, Vue 같은 프레임워크를 사용한다면 div 태그가 중첩되거나 숨어서 안보이는 요소가 제대로 적용됐는지 확인해야 할 때가 있습니다
 - XPath 방식을 사용해서 숨어있는 요소를 손쉽게 찾아 낼 수 있습니다.

2. 복잡한 중첩 요소 (XPath 연습)

👉 여기 숨어있는 텍스트 찾기

특정 텍스트를 포함한 요소를 바로 찾아냅니다.

👉 입력: `inspect($x("//span[contains(text(), '숨어있는')])[0])`



[간단 실습] Console, inspect() 실습

- XPath 를 사용해서 숨어있는 객체도 잘 찾을 수 있게 됩니다.

The screenshot shows the Chrome DevTools Elements tab. The DOM tree on the left displays the following structure:

```
<div>
  <div>
    ...<span class="deep-target"> 여기 숨어있는 텍스트 찾기</span> == $0
  </div>
</div>
```

The element highlighted is ` 여기 숨어있는 텍스트 찾기`. The right panel shows the element's properties and the CSS inheritance chain from `body`.

The bottom section of the DevTools shows the following inspection results:

- window.dataLayer 객체가 로드되었습니다. sample.html?_ijt=k3n...d=RELOAD_ON_SAVE:74
- ③ 이벤트가 연결되어 있습니다! (QA Pass) sample.html?_ijt=k3n...d=RELOAD_ON_SAVE:65

A yellow arrow points to the inspection result for the event connection.

```
> inspect($x("//span[contains(text(), '숨어있는')]")[0])
<span class="deep-target"> 여기 숨어있는 텍스트 찾기</span>
```



[간단 실습] Console, inspect() 실습

- 이벤트가 의도대로 잘 적용 됐는지 확인 할 수 있습니다.

3. 이벤트 리스너 확인

클릭해도 반응 없는 버튼?

버튼에 연결된 기능이 있는지 확인 후 소스로 이동합니다.

- 👉 입력: `getEventListeners(document.getElementById('buggy-btn'))`
- 👉 확인 후: `inspect(document.getElementById('buggy-btn'))`



[간단 실습] Console, inspect() 실습

- 이벤트가 의도대로 적용이 잘 됐다면?
 - 객체 안에 click property 가 존재하는 것을 확인할 수 있습니다.

The screenshot shows the Google Chrome DevTools interface with the following details:

- Elements Tab:** Displays the DOM tree of the page. A specific element is selected, showing its CSS selector as ".section".
- Styles Tab:** Shows the computed styles for the selected element. It includes rules from "element.style" and "sample.html?_ijt=k3n...d=RELOAD_ON_SAVE:8".
- Console Tab:** Displays the JavaScript console output. A yellow arrow points to the third line of code, which shows the result of `getEventListeners` for the element with ID "buggy-btn". The output shows an array containing a single object with a "click" property.

```
3. 이벤트가 연결되어 있습니다! (QA_Pass)
> getEventListeners(document.getElementById('buggy-btn'))
< ▶ {click: Array(1)}
> getEventListeners(document.getElementById('.btn-submit'))
< ▶ {}
```



[간단 실습] Console, inspect() 실습

- Google Tag Manager(GTM)?
 - 유저 이벤트를 추적 할 수 있는 서비스 중 하나
 - 이러한 객체가 제대로 적용이 됐는지 확인 할 수도 있습니다.

4. DataLayer 객체 확인

GTM 등의 데이터가 잘 쌓였는지 객체 형태로 확인합니다.

👉 입력: `inspect(dataLayer)`



[간단 실습] Console, inspect() 실습

- GTM 객체와 같이 유저를 추적하는 이벤트가 제대로 적용 됐다면?
 - 아래와 같이 육안으로 바로 확인 할 수 있다

The screenshot shows the Chrome DevTools interface. On the left, the Elements tab is open, displaying the DOM structure of a sample.html page. The body tab is selected. On the right, the Styles tab shows the CSS styles for the body element, including font-family: sans-serif, padding: 20px, and line-height: 1.6. Below the DOM tree, the Console tab is active, showing the following output:

```
▶ <div class="section">...</div>
▶ <div class="section">...</div>
▶ <script>...</script>
▶ <script>...</script>
</body>
</html>

html  body
```

Console tab:

- Default levels ▾ | No Issues |
- Filter: Filter
- window.dataLayer 객체가 로드되었습니다.
- ③ 이벤트가 연결되어 있습니다! (QA Pass)
- > **inspect(dataLayer)**
- < ▶ (2) [{…}, {…}] ⓘ
 - ▶ 0: {pageCategory: 'signup', visitorType: 'new'}
 - ▶ 1: {event: 'view_item', productId: '12345'}
 - length: 2
 - [[Prototype]]: Array(0)
- >

Sample output from the inspect command:

```
sample.html?_ijt=k3n…d=RELOAD_ON_SAVE:74
sample.html?_ijt=k3n…d=RELOAD_ON_SAVE:65
```



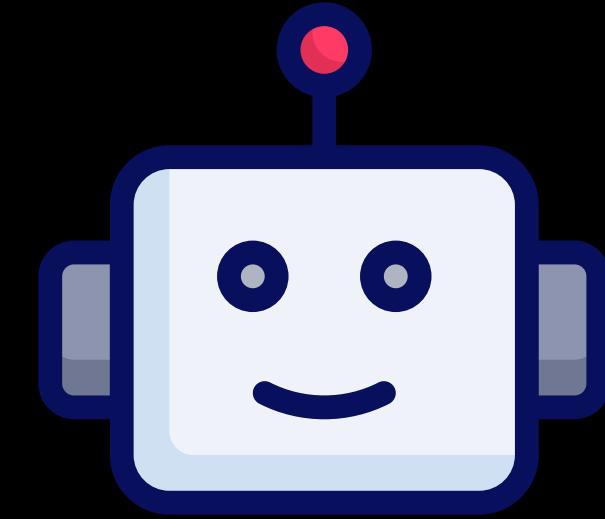
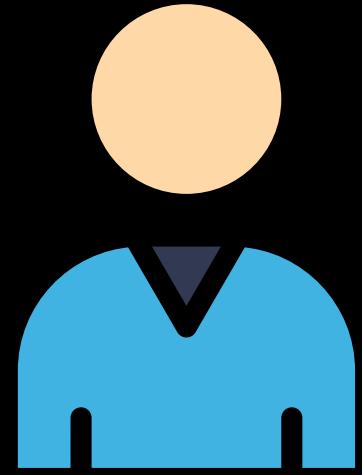
Console, inspect() 실습 정리

- 지금까지 소개해 드린 내용은 디버깅 용도입니다.
- 브라우저의 개발자 도구는 개발을 도와주는 훌륭한 도구입니다.
- 개발자 도구를 사용하면 DOM을 탐색할 수 있고 새로운 것을 적용해 본 후 어떤 변화나 버그가 생기는지 바로 확인할 수 있습니다.
 - 꼭 익숙해질 때까지 반복해서 사용해보시기 바랍니다.
- 이제부터 자바스크립트를 이용해 DOM에 접근하고 수정하는 방법을 살펴보겠습니다.



XPath에 대해서

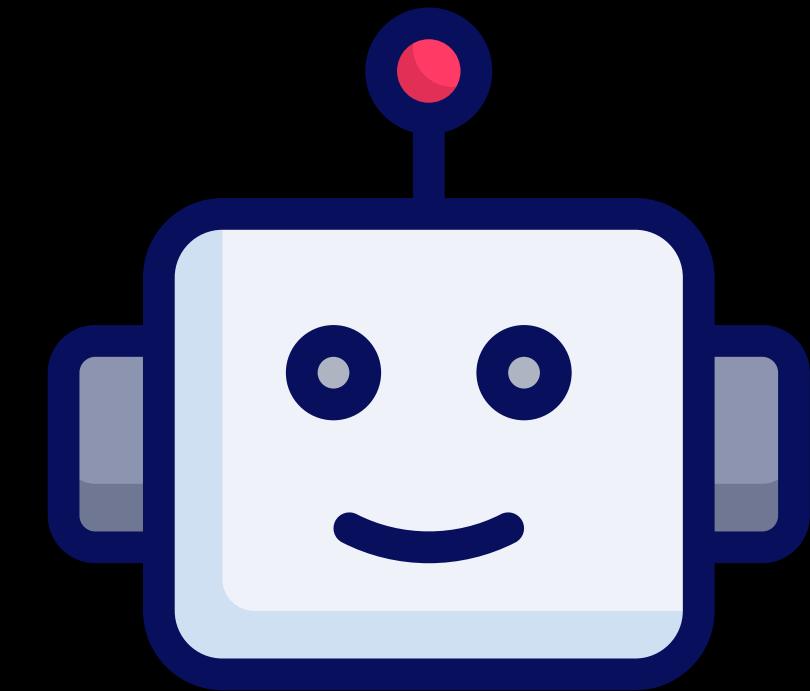
- XPath 란?
 - Selenium 로봇은 눈이 없어서 화면을 볼 수 없음
 - 그래서 우리가 정확한 주소를 알려줘야 요소를 찾아가서 클릭할 수 있음





XPath에 대해서

- 우리는 로봇에게 다음과 같은 형식의 XPath를 전달해주면 된다.



//화면전체/입력창[@아이디='로그인']



XPath 문법

- 딱 3개만 알면 된다!
- 외계어처럼 보이지만 공식은 딱 하나임

//태그명[@속성='값']

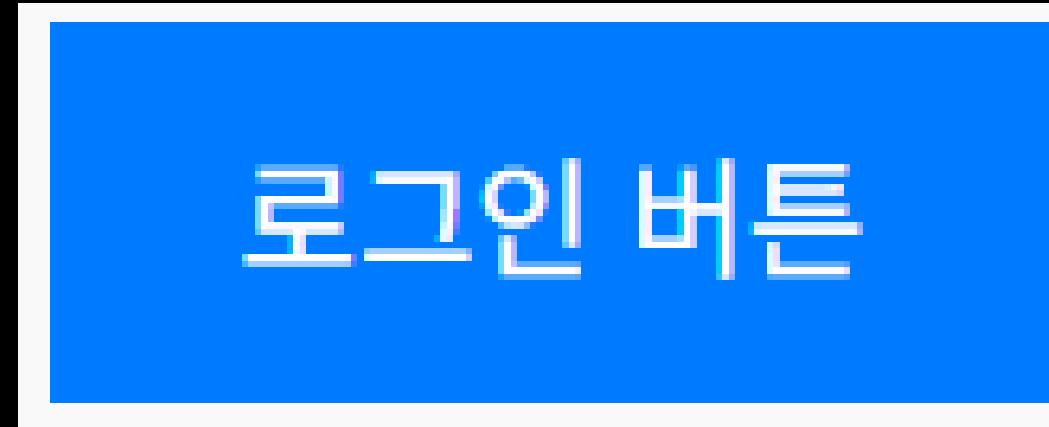
- 이걸 한국어로 풀면 다음과 같다

//어떤 종류의 상자인데[@이름표가='이거인 것'] 찾아줘



XPath 문법

- 이런 HTML(웹페이지 소스)이 있다고 가정해보자



```
<button id="loginBtn">로그인</button>
```



XPath 문법

로그인 버튼

```
<button id="loginBtn">로그인</button>
```

이 문법을 해석해보면 아래와 같다

"//"

- 문서 전체에서 찾아라
- 무조건 이걸로 시작한다고 생각할것!

"button"

- 버튼 태그를 찾아라

"[@id='loginBtn']"

- 근데 id(명찰)가 'loginBtn'이라고 적힌 녀석만!

//button[@id='loginBtn']



XPath 실습

- 기본적인 XPath 문법을 살펴봤다.
- XPath 문법에 대해 더욱 자세한 건이 [링크](#)를 참조
 - 클릭이 안된다면 구글에 "XPath 문법"이라고 검색하면 된다
- 이 [링크](#)에 접속해서 HTML을 파일로 받아주세요
 - 그리고 설명에 나와 있는 10문제를 풀어주세요
 - 링크가 클릭이 안되는 경우 아래 주소를 복사해주세요
 - <https://kdt-gitlab.elice.io/-/snippets/13>



브라우저 이벤트 소개

- 이벤트(event)는 무언가 일어났다는 신호
- 모든 DOM 노드는 이런 신호를 만들어 낸다
 - 참고로 이벤트는 DOM에만 한정되진 않는다.





왜 QA가 이벤트를 알아야 할까?

☞ 문제 상황 (Pain Points)

- "버튼을 눌렀는데 반응이 없어요."(단순 오류 vs 이벤트 미발생)
- Selenium/Cypress 스크립트가 .click()을 수행했는데 미동작
- 특정 브라우저에서만 기능이 동작하지 않음.

💡 해결 능력 (Insight)

- 이벤트 핸들러가 정상적으로 부착되었는지 확인 가능.

Event Interception(가로채기)이나 로딩 지연 이슈 파악.

- 개발자에게 더 구체적인 버그 리포트 제공 가능.



브라우저 이벤트의 3요소



Event Source

"누가?"

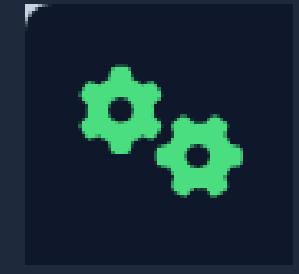
이벤트가 발생한 HTML 요소
(버튼, 인풋 박스, 윈도우 창 등)



Event Type

"무엇을?"

발생한 사건의 종류
(click, scroll, keydown, load 등)



Event Handler

"어떻게?"

이벤트 발생 시 실행될 자바스크립트 함수 (=동작)



[실습] 브라우저 이벤트 감시해보기

monitorEvents ()로 브라우저에서 발생하는 이벤트를 확인해보자

크롬 개발자 도구(F12)는 강력한 이벤트 모니터링 기능을 제공합니다.

별도의 도구 설치 없이 브라우저에서 발생하는 모든 신호를 확인할 수 있습니다.

1. 아무 웹사이트나 접속하여 **F12**를 누른다
2. **Console** 탭으로 이동
3. 우측의 코드를 입력
4. 화면을 클릭하거나 키보드를 눌러본다

```
1 // 원도우에서 발생하는 모든 이벤트 감시
2 monitorEvents(window)
3
4 // // 특정 타입만 감시 (여기선 클릭과 키보드)
5 monitorEvents(window, ["click", "keydown"])
6
7 // 감시 중단
8 unmonitorEvents(window)
```



[실습] 브라우저 이벤트 감시해보기

- 아래와 같이 이벤트가 실시간으로 발생하는 걸 확인할 수 있다



The screenshot shows the Chrome DevTools Console tab with the title 'Console' selected. The interface includes a toolbar with icons for play/pause, stop, and eye, along with a 'Filter' button. To the right, there are buttons for 'Default levels' and '1 Issue' with a count of 1, and a settings gear icon.

The main area displays a list of events monitored on the window:

- > monitorEvents(window)
- < undefined
- devicemotion VM92:1
▶ DeviceMotionEvent {isTrusted: true, acceleration: DeviceMotionEventAcceleration, accelerationIncludingGravity: DeviceMotionEventAcceleration, rotationRate: DeviceMotionEventRotationRate, interval: 16, ...}
- deviceorientation VM92:1
▶ DeviceOrientationEvent {isTrusted: true, alpha: null, beta: null, gamma: null, absolute: false, ...}
- pointerover VM92:1
▶ PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
- mouseover VM92:1
▶ MouseEvent {isTrusted: true, screenX: 118, screenY: 743, clientX: 118, clientY: 656, ...}
- pointermove VM92:1
▶ PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
- mousemove VM92:1
▶ MouseEvent {isTrusted: true, screenX: 118, screenY: 743, clientX: 118, clientY: 656, ...}



이벤트 전파 (Bubbling)

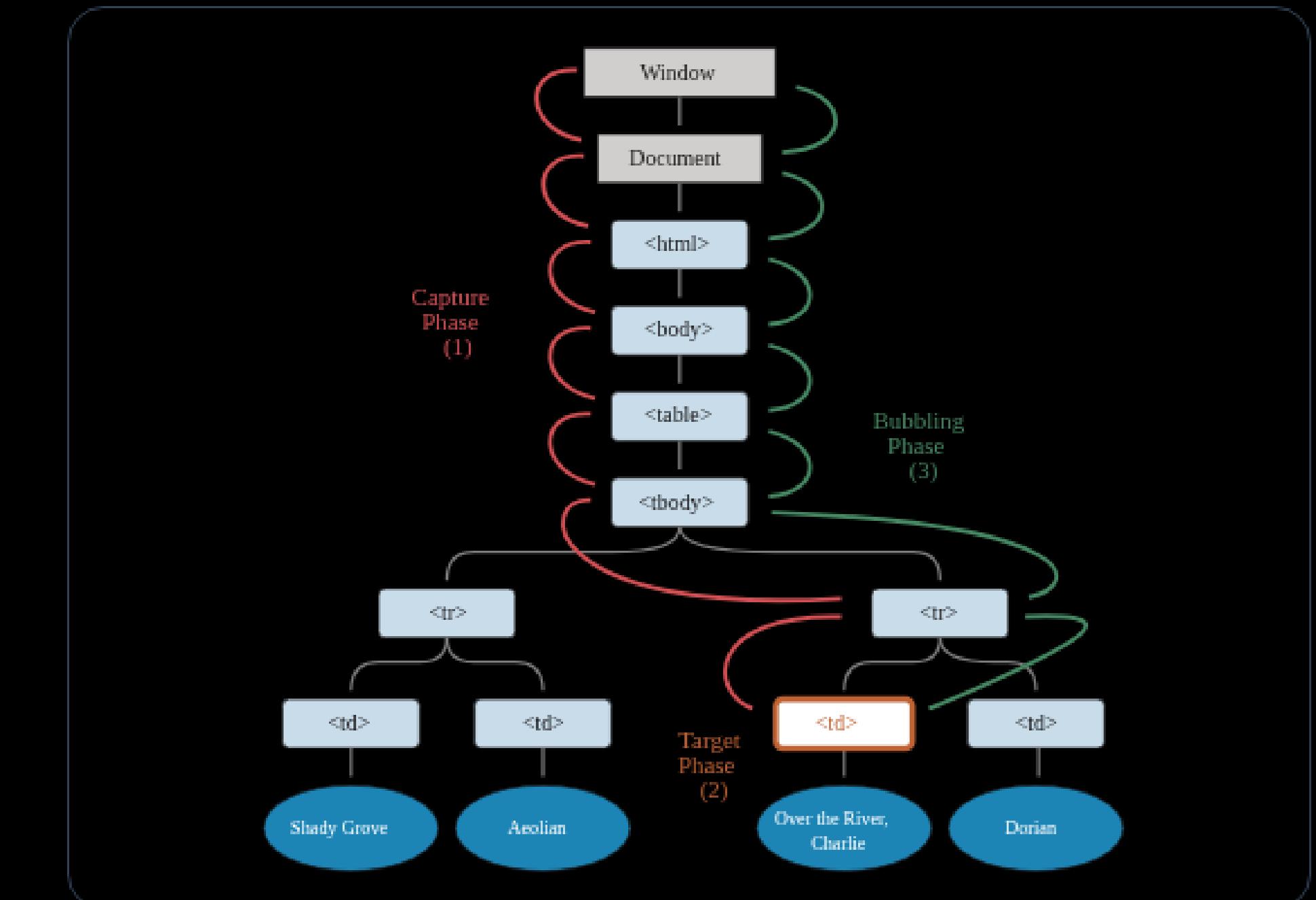
거품처럼 올라가는 이벤트

이벤트는 발생한 지점에서 멈추지 않습니다.

- **Bubbling:** 자식 요소에서 발생한 이벤트가 한 부모 요소로 전파되는 현상. (가장 흔함)
- **Capturing:** 부모 요소에서 자식 요소로 내려가는 현상.

Check Point

"작은 버튼을 눌렀는데, 왜 그 뒤에 있는 큰 박스의 클릭 이벤트까지 같이 실행될까요?" -> 버블링 때문입니다.





[실습] 이벤트 전파 (Bubbling)

- 이 [링크](#)로 접속해서 HTML 파일을 다운로드
- 해당 HTML 파일을 브라우저로 열기
- 렌더링 된 파일에서 가장 안쪽의 노란색 박스를 클릭해보기

이벤트 버블링 클릭 테스트

가장 안쪽의 노란색 박스(**Inner**)를 클릭해보세요.

Outer (할아버지)

Middle (부모)

Inner (자식)

■ OUTER 클릭됨! (currentTarget: outer)
■ MIDDLE 클릭됨! (currentTarget: middle)
■ INNER 클릭됨! (currentTarget: inner)
■ OUTER 클릭됨! (currentTarget: outer)



실습 후 결과 정리

- 가장 안쪽의 노란색 박스만 눌렀는데 Middle, Outer 까지 모두 클릭되는 것을 관찰 할 수 있다.
 - 분명 노란색 박스 하나만 클릭했지만 브라우저는 이를 감싸고 있는 부모 요소들도 클릭된 것으로 간주하여 이벤트를 위로 계속 전달하기 때문이다.
- 버블링을 막으려면?
 - 만약 특정 단계에서 버블링을 멈추고 싶다면 `event.stopPropagation()` 메서드를 사용
 - 예를 들어 Middle에서 전파를 멈추고 싶다면 자바스크립트 코드를 다음과 같이 수정하면 된다.

```
middle.addEventListener('click', function(event) {  
    event.stopPropagation(); // 여기서 버블링 중단!  
    logMessage(` MIDDLE 클릭됨! (Outer로 전파되지 않음)`);  
});
```



실습 후 결과 정리

- 아래 코드를 적용해서 다시 클릭해서 결과를 확인해보자

```
middle.addEventListener('click', function(event) {  
    event.stopPropagation(); // 여기서 버블링 중단!  
    logMessage(` MIDDLE 클릭됨! (Outer로 전파되지 않음)`);  
});
```

- 이렇게 하면 Inner를 클릭했을 때 `Inner -> Middle`까지만 실행된다.
 - Outer는 실행되지 않음!
- 이벤트를 막는 방법은 stopPropagation() 만 있는게 아니다.
 - preventDefault() 라는 것도 존재한다.



실습 후 결과 정리

• stopPropagation() vs preventDefault() 비교 정리

구분	preventDefault()	stopPropagation()
목적	브라우저 고유 기능 차단	이벤트 전파(버블링) 차단
무엇을 막나?	링크 이동, 폼 제출, 텍스트 선택 등	부모 요소의 이벤트 리스너 실행
영향 범위	현재 요소의 행동 에만 영향	DOM 트리 상의 흐름 에 영향
비유	"문은 열지 마" (행동 금지)	"내가 문 열었다고 윗선에 보고하지 마" (보고 금지)

- 보통은 <form> 태그에서 제출 후 페이지가 새로고침 되는 것을 막기 위해서 preventDefault() 메서드를 많이 사용한다.



이벤트 핸들러 (Event Handler)

- 이벤트에 반응하려면 이벤트가 발생했을 때 실행되는 함수인 **핸들러(handler)**를 할당 해야 함
- 핸들러는 사용자의 행동에 어떻게 반응할지를 자바스크립트 코드로 표현한 것
- 핸들러는 여러 가지 방법은 세가지
 - 방법 1: HTML 속성 사용 (권장하지 않음)
 - 방법 2: DOM 속성 사용
 - 방법 3: addEventListener 사용 (권장)



이벤트 핸들러 (Event Handler)

- 방법 1: HTML 속성 사용 (권장하지 않음)

```
<button onclick="alert('클릭!')">클릭하세요</button>
```

- HTML 안의 on<event> 속성에 핸들러를 할당할 수 있음
- input 태그의 onclick 속성에 click 핸들러를 할당하는 것을 살펴보자

```
<input value="클릭해 주세요." onclick="alert('클릭!')" type="button">
```



이벤트 핸들러 (Event Handler)

방법 1: HTML 속성 사용 (권장하지 않음)

- 버튼을 클릭하면 onclick 안의 코드가 실행됨
- 여기서 주의해야 할 것은 속성값 내에서 사용된 따옴표
- 속성값 전체가 큰따옴표로 둘러싸여 있기 때문에 작은 따옴표로 둘러싼다
- 아래 코드를 `onclick="alert("클릭!")"`과 같이 쓰게 된다면 속성값 내부에 또 큰따옴표를 쓰이면서 코드가 작동되지 않게 된다.

```
<input value="클릭해 주세요." onclick="alert('클릭!')" type="button">
```

따라서 긴 코드를 HTML 속성값으로 사용하는 것은 추천하지 않음

- 만약 코드가 길다면 함수를 만들어서 이를 호출하는 방법을 추천!



이벤트 핸들러 (Event Handler)

방법 1: HTML 속성 사용 (권장하지 않음)

- 만약 코드가 길다면 함수를 만들어서 아래와 같이 호출하는 방법을 추천!

```
<script>
    function countRabbits() {
        for(let i=1; i<=3; i++) {
            alert(`토끼 ${i}마리`);
        }
    }
</script>

<input type="button" onclick="countRabbits()" value="토끼를 세봅시다!">
```

HTML 속성은 대·소문자를 구분하지 않음

- ONCLICK, onClick, onCLICK과 모두 동일하게 작동함
- 하지만 속성값은 대개 onclick 같이 소문자로 작성하는게 암묵적 룰이다.



이벤트 핸들러 (Event Handler)

방법 2: DOM 속성 사용

- DOM 프로퍼티 `on<event>`을 사용해도 핸들러를 할당할 수 있음

```
const button = document.querySelector('button');
button.onclick = function() {
    alert('클릭!');
};
```

핸들러를 HTML 속성을 사용해 할당하면, 브라우저는 속성값을 이용해 새로운 함수를 만듭니다. 그리고 생성된 함수를 DOM 프로퍼티에 할당합니다.



이벤트 핸들러 (Event Handler)

방법 3: addEventListener 사용 (권장)

- HTML 속성과 DOM 프로퍼티를 이용한 이벤트 핸들러 할당 방식엔 근본적인 문제가 있다.
- 바로 하나의 이벤트에 복수의 핸들러를 할당할 수 없다는 것
- 버튼을 클릭하면 버튼을 강조하면서 메시지를 보여줘야 하는 것을 구현해야 하는 상황을 생각해 보자
- 당연히 두 개의 이벤트 핸들러가 필요할 것이다.
- 하지만 기존 방법으로는 프로퍼티가 덮어씌워 진다는 문제가 있다.

```
input.onclick = function() { alert(1); }
// ...
input.onclick = function() { alert(2); } // 이전 핸들러를 덮어씀
```



이벤트 핸들러 (Event Handler)

방법 3: addEventListener 사용 (권장)

- 웹 표준에 관여하는 개발자들은 오래전부터 이 문제를 인지하고 있었음
- 그래서 핸들러를 여러 개 할당할 수 있도록 표준을 바꿔야 했음
- 그러면서 addEventListener 와 removeEventListener 라는 특별한 메서드를 이용해 핸들러를 관리하자는 대안이 제시됨.

문법은 다음과 같다.

```
element.addEventListener(event, handler, [options]);
```



이벤트 핸들러 (Event Handler)

방법 3: addEventListener 사용 (권장)

- 다음과 같이 하나의 객체에 여러개의 이벤트 핸들러를 주렁주렁 달 수가 있다

```
<input id="elem" type="button" value="클릭해 주세요.">

<script>
    function handler1() {
        alert('감사합니다!');
    }

    function handler2() {
        alert('다시 한번 감사합니다!');
    }

    elem.onclick = () => alert("안녕하세요.");
    elem.addEventListener("click", handler1); // 감사합니다!
    elem.addEventListener("click", handler2); // 다시 한번 감사합니다!
</script>
```