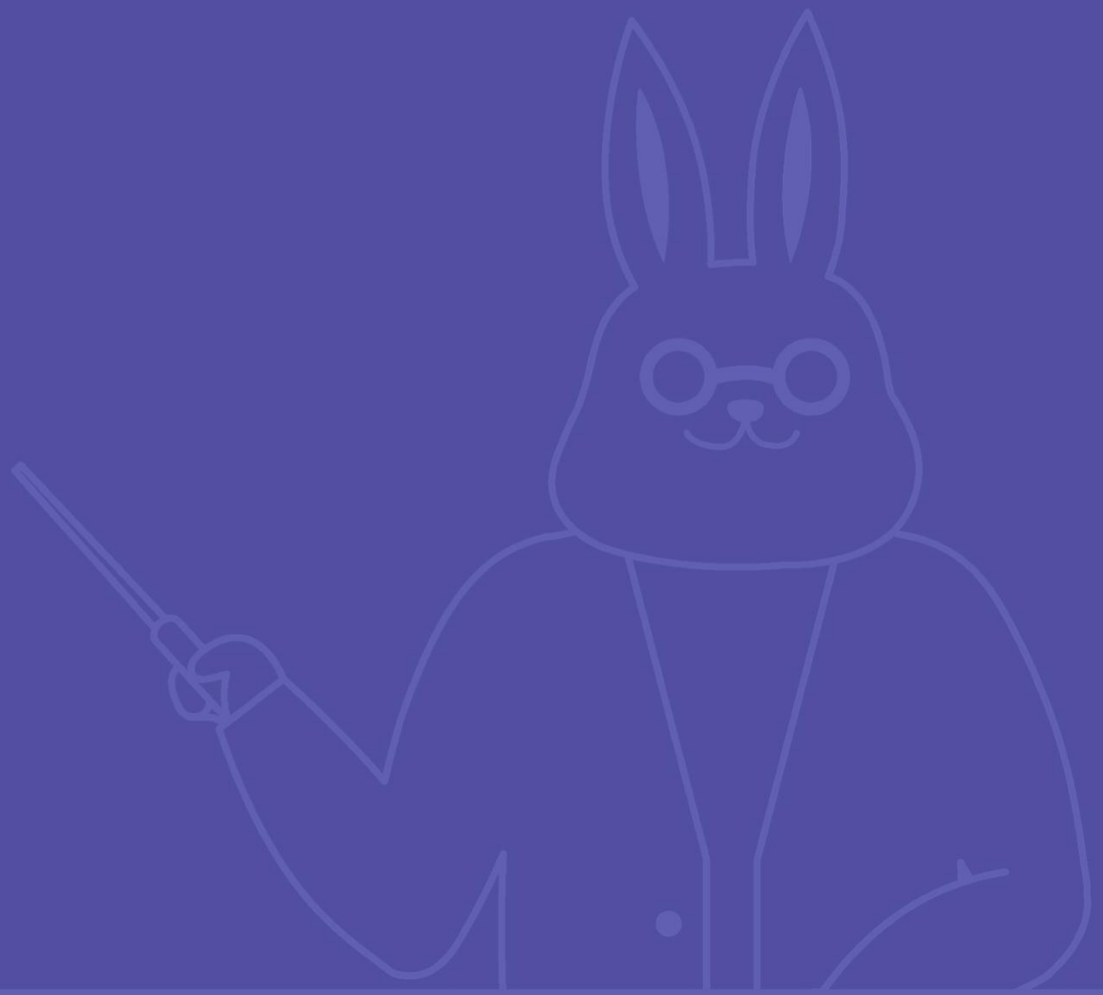
자바스크립트

03 자바스크립트 시작하기



🎯 수강목표

1. JAVASCRIPT가 무엇인지 알 수 있다.

javascript가 어떤 것이고, 왜 사용하는지 이해한다.
또한 javascript의 특성을 이해한다.

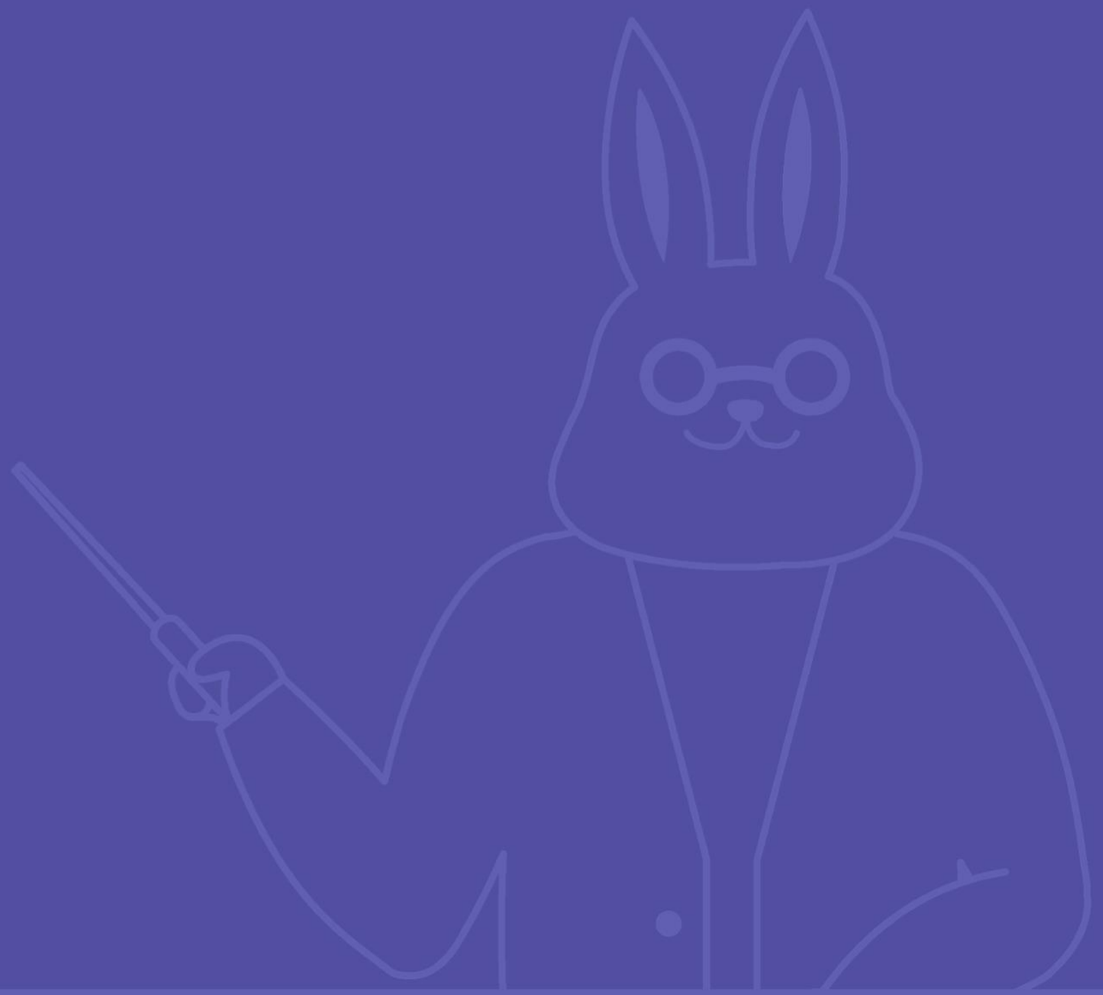
2. 동적인 웹사이트를 만들 수 있다.

javascript를 활용해서 클릭 이벤트 등 다양한 동적 기능을 추가한다.

목차

01

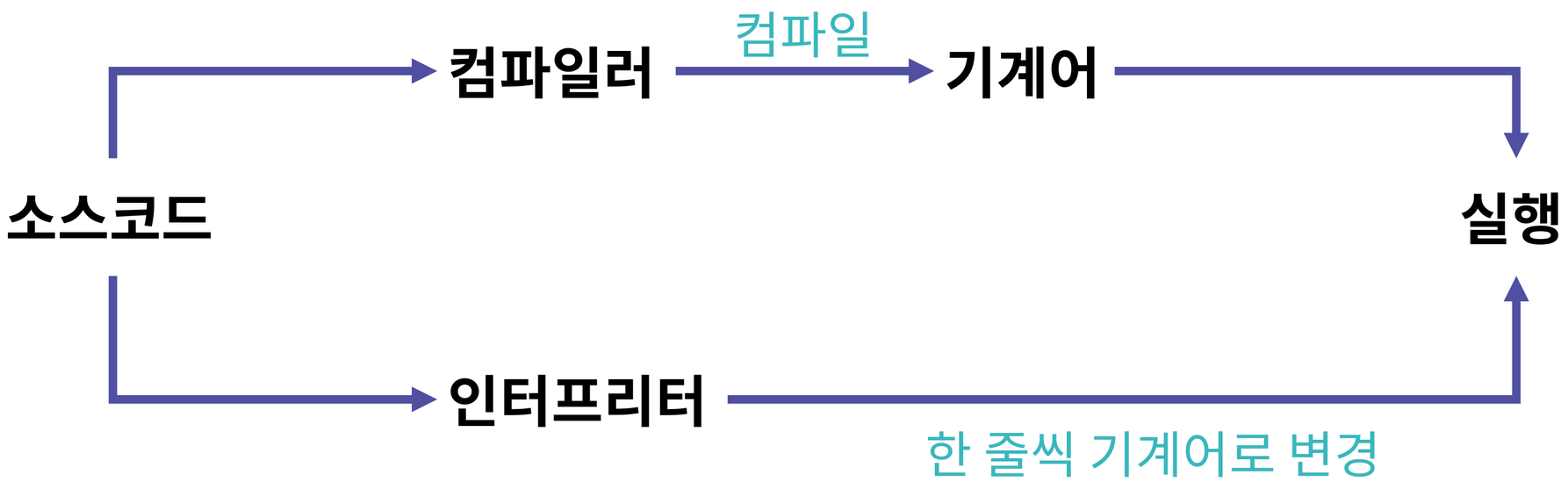
javascript 란?



자바스크립트는 어떤 걸까?

- 웹에서 동작에 대한 부분을 다룰 수 있음
- **객체** 기반 스크립트 언어
- **인터프리터** 언어

인터프리터(interpreter)?



ES (ECMAScript)

- Ecma International?

  정보 통신에 대한 표준을 제정하는 비영리 표준화 기구

- ECMA-262

  범용 목적의 스크립트 언어에 대한 명세

- ECMAScript?

  ECMA-262 기술 규격에 의해 정의된 범용 스크립트 언어

- 그러면 JavaScript는?

  ECMAScript 사양을 준수하는 범용 스크립트 언어

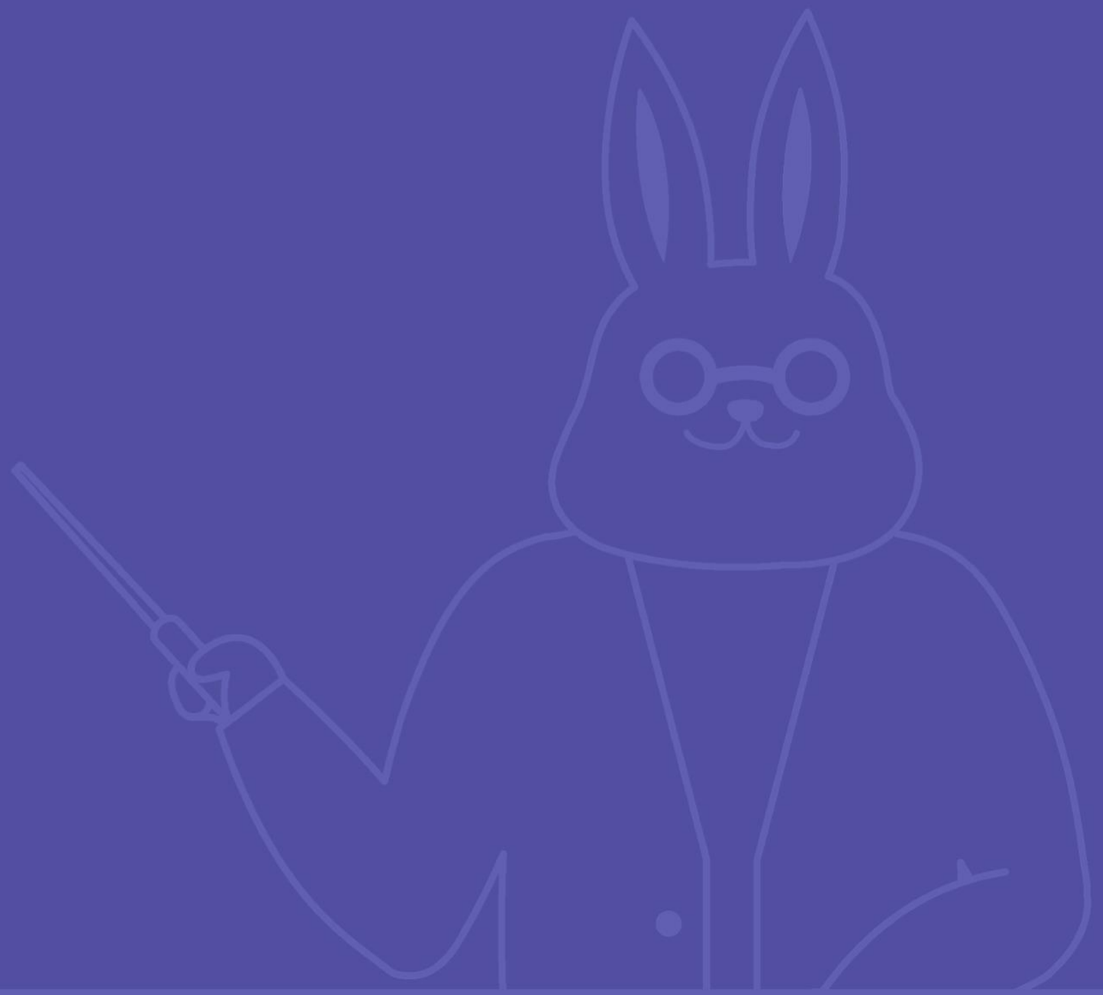
ES (ECMAScript)

판	출판일	이름	이전 판과의 차이점
1	1997년 6월		초판
2	1998년 6월		ISO/IEC 16262 국제 표준과 완전히 동일한 규격을 적용하기 위한 변경.
3	1999년 12월		강력한 정규 표현식, 향상된 문자열 처리, 새로운 제어문 , try/catch 예외 처리, 엄격한 오류 정의, 수치형 출력의 포매팅 등.
4	버려짐		4번째 판은 언어에 얽힌 정치적 차이로 인해 버려졌다. 이 판을 작업 가운데 일부는 5번째 판을 이루는 기본이 되고 다른 일부는 ECMA스크립트의 기본을 이루고 있다.
5	2009년 12월		더 철저한 오류 검사를 제공하고 오류 경향이 있는 구조를 피하는 하부집합인 "strict mode"를 추가한다. 3번째 판의 규격에 있던 수많은 애매한 부분을 명확히 한다.[3]
5.1	2011년 6월		ECMA스크립트 표준의 제 5.1판은 ISO/IEC 16262:2011 국제 표준 제3판과 함께 한다.
6	2015년 6월	ECMAScript 2015 (ES2015)	6판에는 클래스와 모듈 같은 복잡한 응용 프로그램을 작성하기 위한 새로운 문법이 추가되었다. 하지만 이러한 문법의 의미는 5판의 strict mode와 같은 방법으로 정의된다. 이 판은 "ECMAScript Harmony" 혹은 "ES6 Harmony" 등으로 불리기도 한다.
7	2016년 6월	ECMAScript 2016 (ES2016)	제곱연산자 추가, Array.prototype.includes
8	2017년 6월	ECMAScript 2017 (ES2017)	함수 표현식의 인자에서 trailing commas 허용, Object values/entries 메소드, async/await 등.
9	2018년 6월	ECMAScript 2018 (ES2018)	Promise.finally, Async iteration, object rest/spread property 등.
10	2019년 6월	ECMAScript 2019 (ES2019)	Object.fromEntries, flat, flatMap, Symbol.description, optional catch 등.

출처 : 위키백과

02

javascript 사용법



3가지 사용 방법

- 인라인(inline) 방식
- 내부(internal) 방식
- 외부(external) 방식

인라인(inline) 방식

예제 1

```
<input
    type="button"
    onclick="alert('example!');
>
```

- 정해진 속성 값 안에 입력
- 짧은 내용 작성할 때 사용
- 주로 함수 호출하는 방식을 이용

내부(internal) 방식

예제 2

```
<script>
    console.log("example!");
</script>
```

- script 태그 내에 작성
- 어디에도 넣을 수 있음
- 코드가 읽힐 때 실행되므로, 위치가 중요함

외부(external) 방식

예제 3

```
<script src="main.js"></script>

// main.js
console.log("example!");
```

- script 태그의 속성으로 파일명 기재
- 태그 내용은 기입하지 않음
- **순서에 주의**할 것

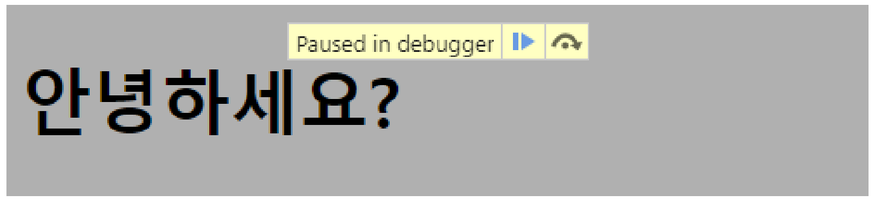
디버깅 방법

예제 4

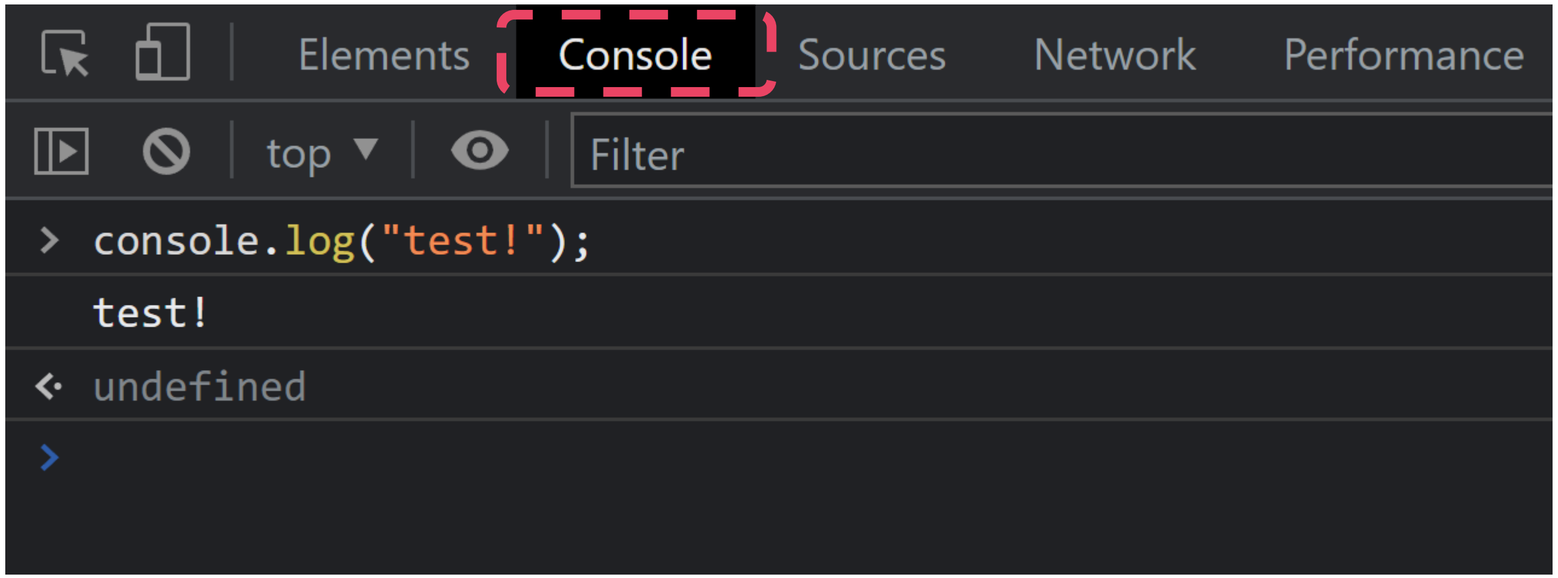
```
<body>
    <h1>안녕하세요?</h1>
    <script>
        debugger;
        console.log("example!");
        document.write("example");
    </script>
</body>
```

- **console**은 브라우저의 **디버깅 콘솔**을 의미

- debugger가 써 있는 부분에서 멈출 수 있음
- 개발자 도구를 사용해야 함

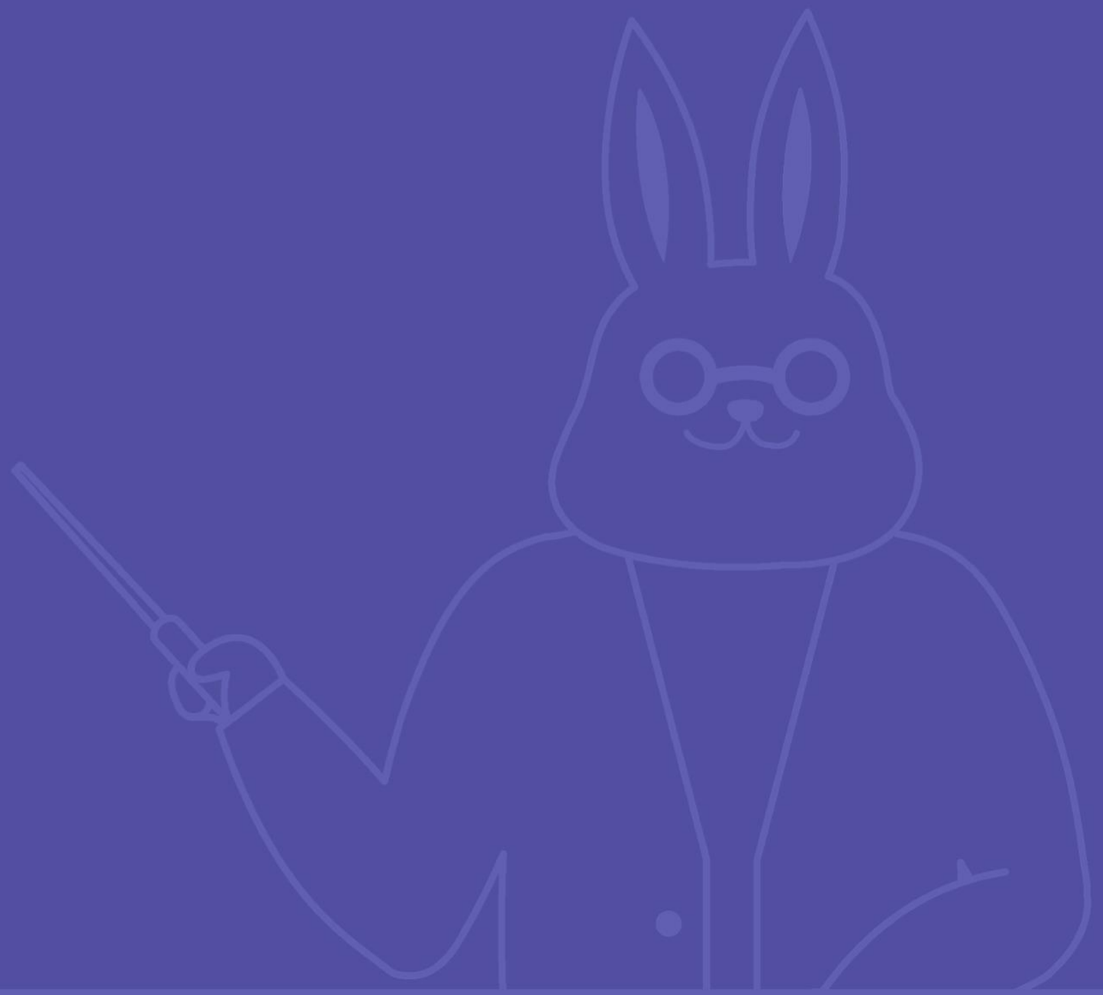


디버깅 방법


Elements
Console
Sources
Network
Performance
top
Filter
console.log("test!");
test!
undefined

03

변수 이해하기



변수의 선언, 할당, 참조

예제 5

```
var ex1;              // 선언 및 초기화
ex1 = 10;             // 할당
console.log(ex1); // 참조
```

var, let, const

- **var** : 변수 선언
- **let** : (ES6에서부터) 변수 선언
- **const** : (ES6에서부터) 변하지 않는 변수 선언 (상수)

var, let 차이 (Hoisting) – 간단히만 살펴보자

- **호이스팅?**

  코드 실행 전, **변수/함수 선언**이 해당 스코프의 최상단으로 끌어올려진 것 같은 **현상**

예제 6

```
var ex1 = 10;
var ex2 = 30;
var ex3 = "호이스팅";
```

예제 7

```
var ex1;
var ex2;
var ex3;

ex1 = 10;
ex2 = 30;
ex3 = "호이스팅";
```

var, let 차이 (Hoisting) – 간단히만 살펴보자

예제 8

```
console.log(ex1); // undefine
var ex1 = 10;
console.log(ex1);

console.log(ex2); // error 발생
let ex2 = 20;
console.log(ex2);
```

Identifier **Memory**

문자열

예제 9

```
let title = "이력서";
let content = '안녕하세요?';

const ES_VERSION = 6;

// es6부터 사용 가능
let tmp = `템플릿 리터럴은 \
ES${ES_VERSION}부터 사용 \
가능합니다.`;
```

- 문자열은 " 혹은 ' 로 감싸서 사용
- 여러 줄 정의 시 \ 으로 줄바꿈 가능

  \ 뒤에는 아무 것도 없어야함 (공백포함)

- **템플릿 리터럴(Template Literals)**
  - ` 으로 감싸주어 사용
  - ${ } 으로 변수 값을 직접 집어넣을 수 있음
  - \ 을 사용하지 않아도 줄바꿈을 인식함

문자열 – escape character

코드	출력
`\XXX`	8진수 Latin-1 문자
`\'`	작은따옴표
`\"`	큰따옴표
`\\`	역슬래시
`\n`	개행
`\r`	캐리지 리턴
`\v`	세로 탭
`\t`	탭
`\b`	백 스페이스
`\f`	폼 피드
`\uXXXX`	유니코드 코드포인트
`\u{X}` ... `\u{XXXXXX}`	유니코드 코드포인트
`\xXX`	Latin-1 문자

```
let str;
str = "큰따옴표(\")";
str = '작은따옴표(\')';
str = "탭('\t')";
```

함수

예제 10

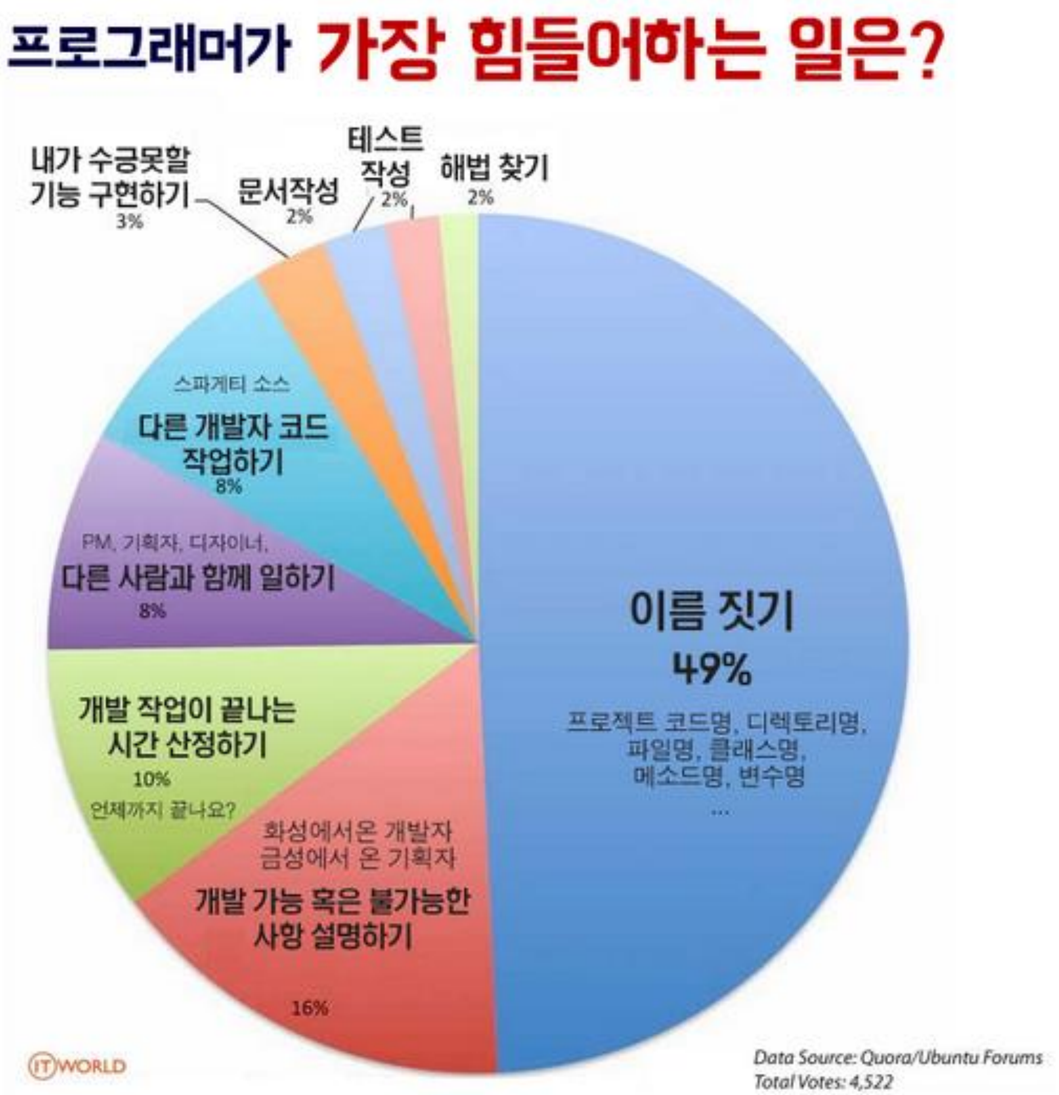
```
// 1. 함수 선언
function insertText(text) {
    document.write(text);
    return "ok";
}

// 2. 함수 표현식
let insertText = function(text) {
    document.write(text);
    return "ok";
}

let res = insertText("안녕하세요?");
```

- 코드를 블럭화 할 수 있음
- 의미의 구분 / 반복되는 코드의 구분

- **함수 선언**을 통해 어디서든 사용할 수 있음
- **함수 표현식**은 변수에 함수를 담을 수 있음

- **return**을 통해 **함수 실행 후** 값을 넘겨 받을 수 있음

변수명에 대한 이야기



출처 : Quora

변수명에 대한 이야기

```
/* Bad */
const page_count = 5
const shouldUpdate = true

/* Good */
const pageCount = 5
const shouldUpdate = true

/* Good as well */
const page_count = 5
const should_update = true
```

```
/* Bad */
const a = 5 // "a" could mean anything
const isPaginatable = a > 10 // "Paginatable" sounds extremely unnatural
const shouldPaginatize = a > 10 // Made up verbs are so much fun!

/* Good */
const postCount = 5
const hasPagination = postCount > 10
const shouldPaginate = postCount > 10 // alternatively
```

출처 : https://github.com/kettanaito/naming-cheatsheet

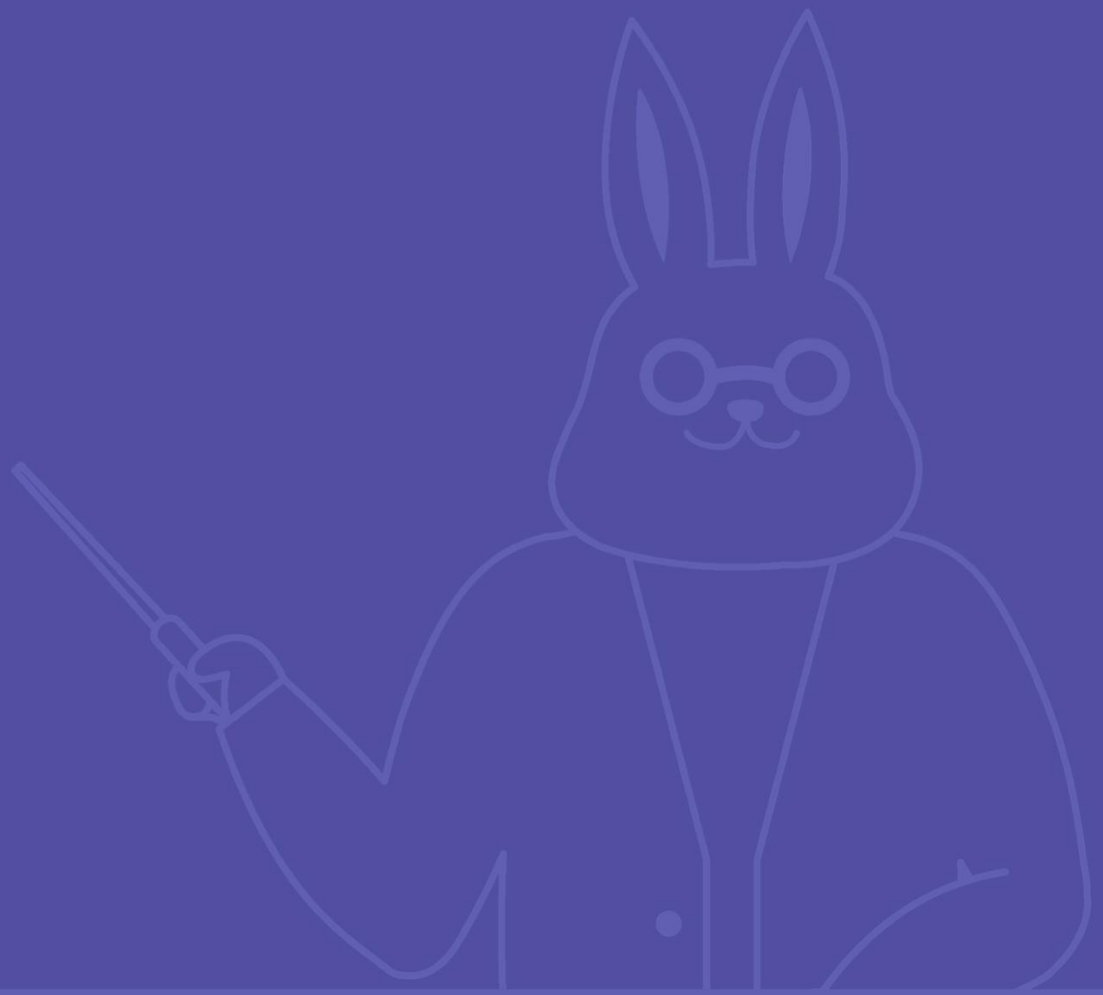
변수명에 대한 이야기

Name	Prefix	Action (A)	High context (HC)	Low context (LC)
`getUser`		`get`	`User`	
`getUserMessages`		`get`	`User`	`Messages`
`handleClickOutside`		`handle`	`Click`	`Outside`
`shouldDisplayMessage`	`should`	`Display`	`Message`	

출처 : https://github.com/kettanaito/naming-cheatsheet

04

조건문과 반복문



조건문 (if – else)

예제 11

```
let visitCnt = 5;
let memberLevel = "";
if (visitCnt >= 100) {
   memberLevel = "gold";
} else if (visitCnt >= 30) {
    memberLevel = "silver";
} else {
    memberLevel = "bronze";
}
console.log(memberLevel);
```

- 조건을 여러 가지로 나눌 수 있음 (**else if**)
- 앞 조건을 충족하지 못하면 다음 조건을 검사하는 형태

- else 부분을 통해서 **기본 값을 지정**
- **조건이 실행되지 않았을 경우를 고려**

반복문 (for / while)

예제 12

```
for (let i = 1; i < 10; i++) {
    console.log(i * i);
}

let i = 1;
while (i < 10) {
    console.log(i*i);
    i++;
}
```

- 반복 작업이 필요한 경우 사용
- 상황에 맞춰서 for / while 중 선택
- 더 다양한 반복문이 존재함
- 우선 가장 기본인 여기까지만 알아두자!

반복문 (for / while)

예제 13

```
for (let i = 1; i < 10; i++) {
    if (i == 3) continue;

    console.log(i * i);
}

for (let i = 1; i < 10; i++) {
    if (i == 3) break;

    console.log(i * i);
}
```

- **continue** :

  해당 지점에서 다음 반복으로 바로 넘어감

- **break** :

  해당 지점에서 바로 반복문을 종료

간단한 실습을 해보자

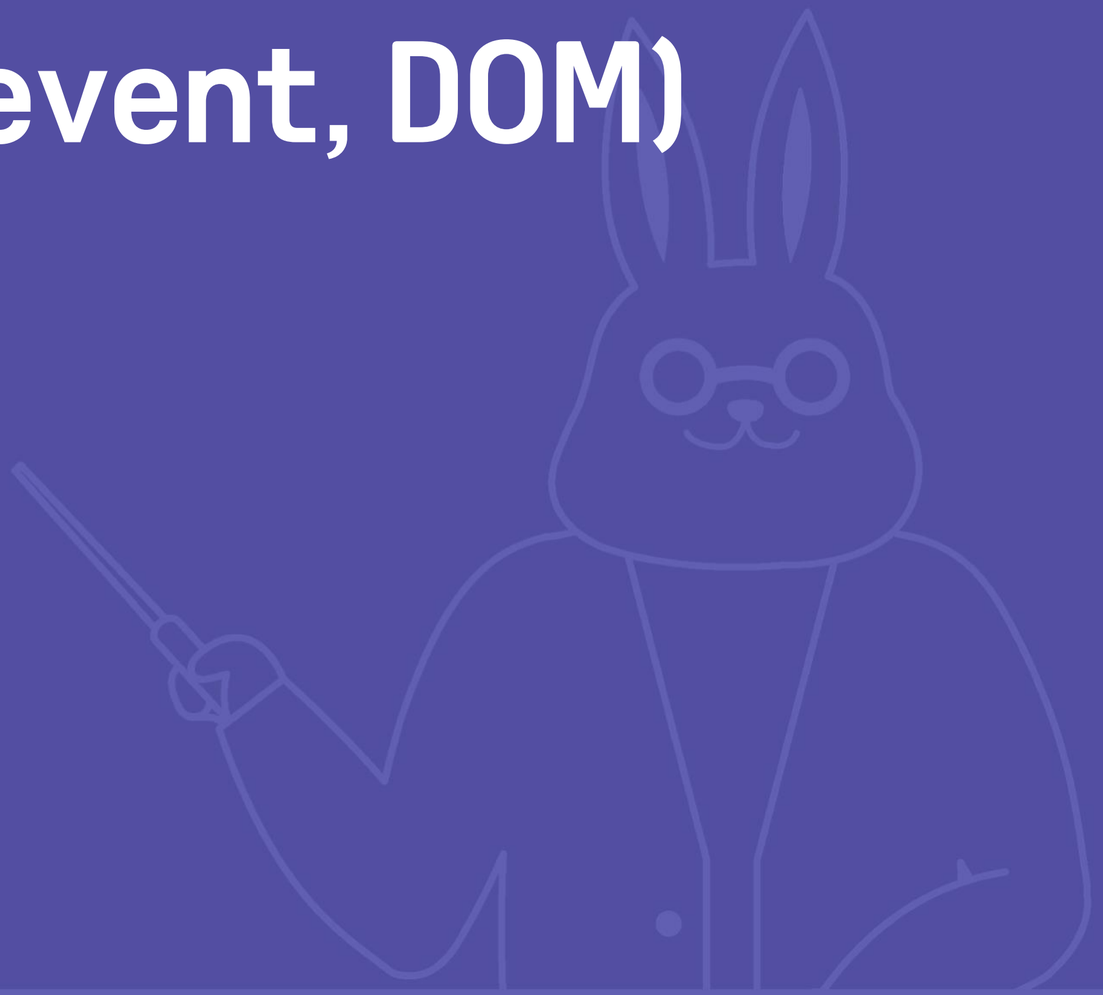
- 구구단을 콘솔에 출력
- 단, 4단과 7단은 출력하지 않음

- **for 문**과 **if 문**을 사용

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
```

05

실습을 통해 이해하기 (event, DOM)



실습 결과물을 먼저 살펴보자

클릭하면

onclick

예제 14

```
// index.html
<input
    type="button"
    onclick="handleClick();"
>

// main.js
function handleClick() {
    alert("클릭되었습니다.");
}
```

- onclick은 이미 정의되어 있는 속성
- 프로퍼티(property)로서 정의되어 있음

- 해당 요소가 **클릭되었을 경우** 속성 값에 있는 **javascript가 실행** 됨

- 개발자 도구에서 event 확인 가능

다양한 이벤트

- **onclick** : 해당 요소가 **클릭**되었을 경우
- **oncontextmenu** : 해당 요소를 **오른쪽 클릭**했을 경우 (메뉴가 나오기 전 이벤트 발생)
- **onchange** : 해당 요소의 **값이 변경**되었을 경우
- **onmouseover** : 해당 요소에 **마우스가 올라**갔을 경우
- **onkeydown** : 해당 요소에서 **키보드가 눌렸**을 경우
- **onfocus** : 해당 요소에 **포커스가 이동** 됐을 경우

event 다루기

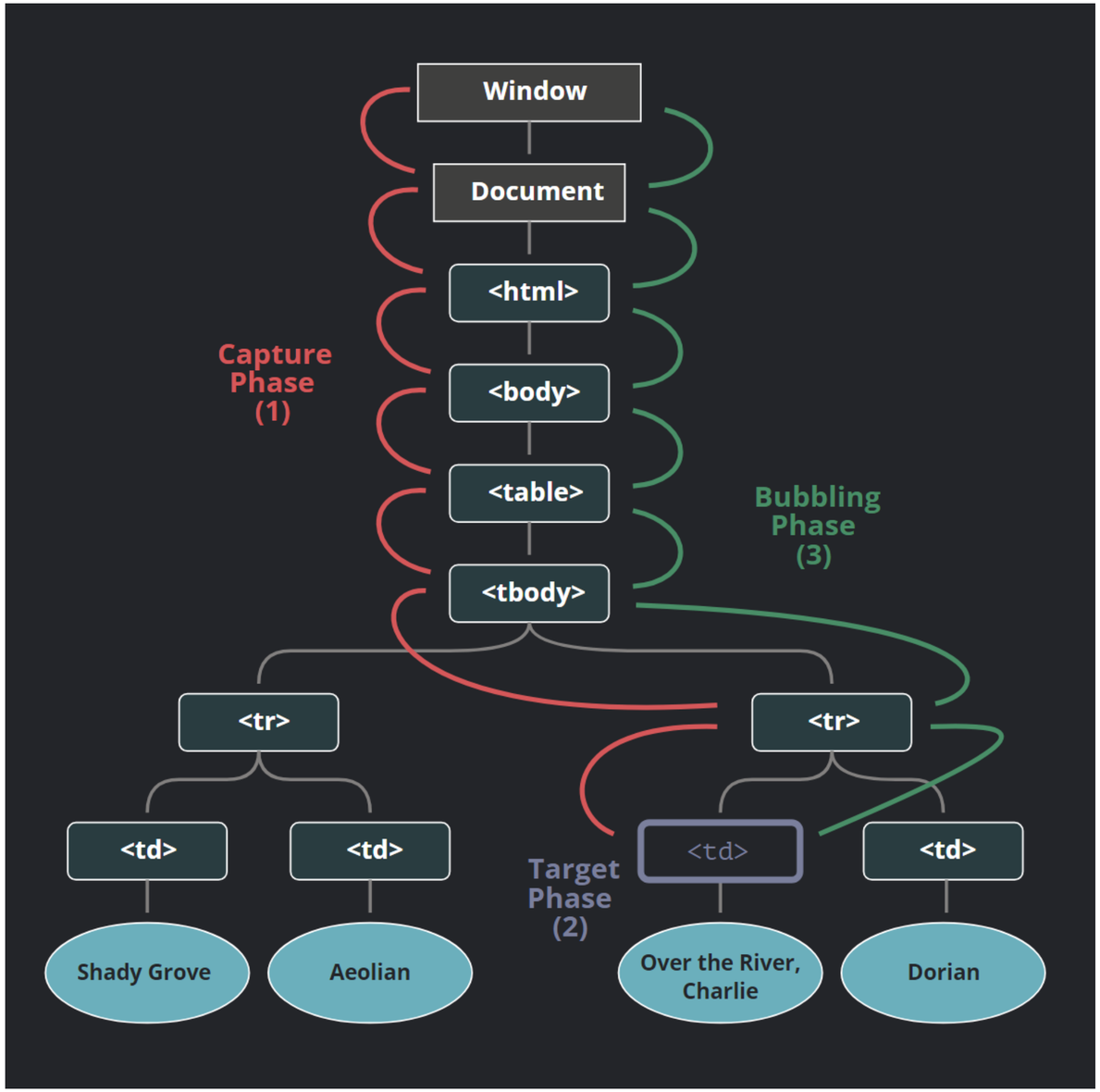
예제 15

```
<input id="target" type="button" onclick="handleClick();" />
<script>
    let target = document.getElementById("target");
    // property 방식
    target.onclick = function () {
        handleClick();
    }
    // addEventListener 방식 (중복설정 가능)
    target.addEventListener("click", function () {
        handleClick();
    });
</script>
```

event 캡처링과 버블링

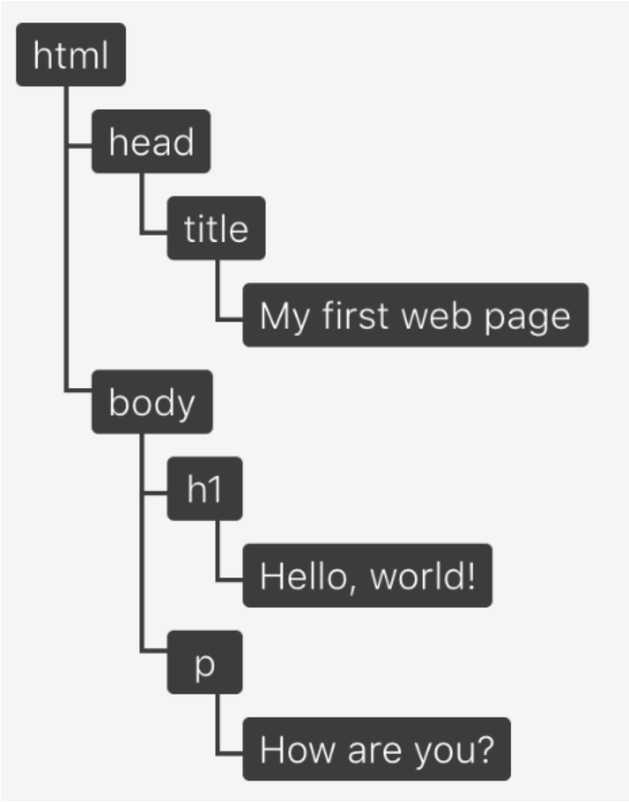
예제 16

```
<div onclick="alert('div');">
    <p onclick="alert('p')">
        안녕하세요?
    </p>
</div>
```



DOM (Document Object Model)

script에서 다루기 쉽게 문서를 객체화 한 것



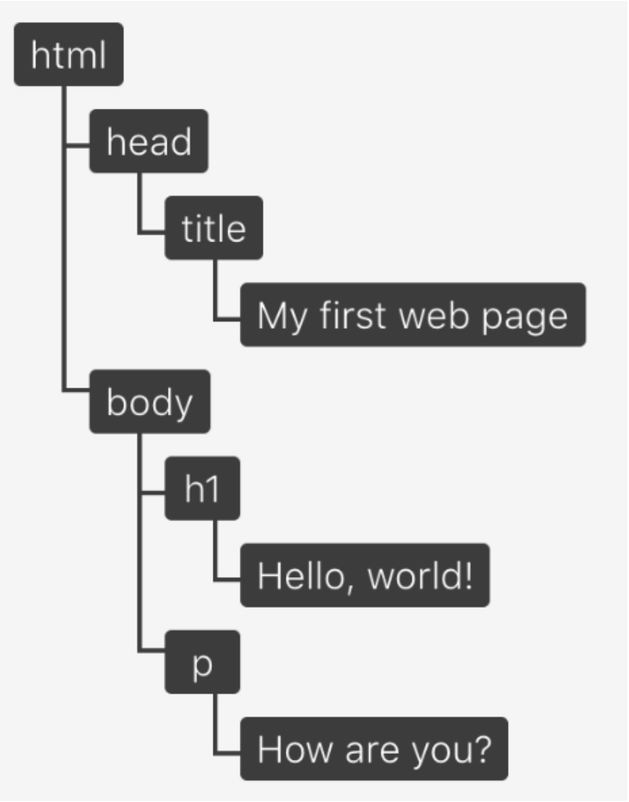
- 각 태그를 **노드(Node)**의 개념으로 변경
- 각 노드로 이루어진 모습을 **트리(Tree)**라고 부름
- 즉, 왼쪽 이미지는 **DOM Tree**

- DOM은 문서와 스크립트 사이의 징검다리 역할
- DOM은 **각 브라우저로 부터 구현**됨

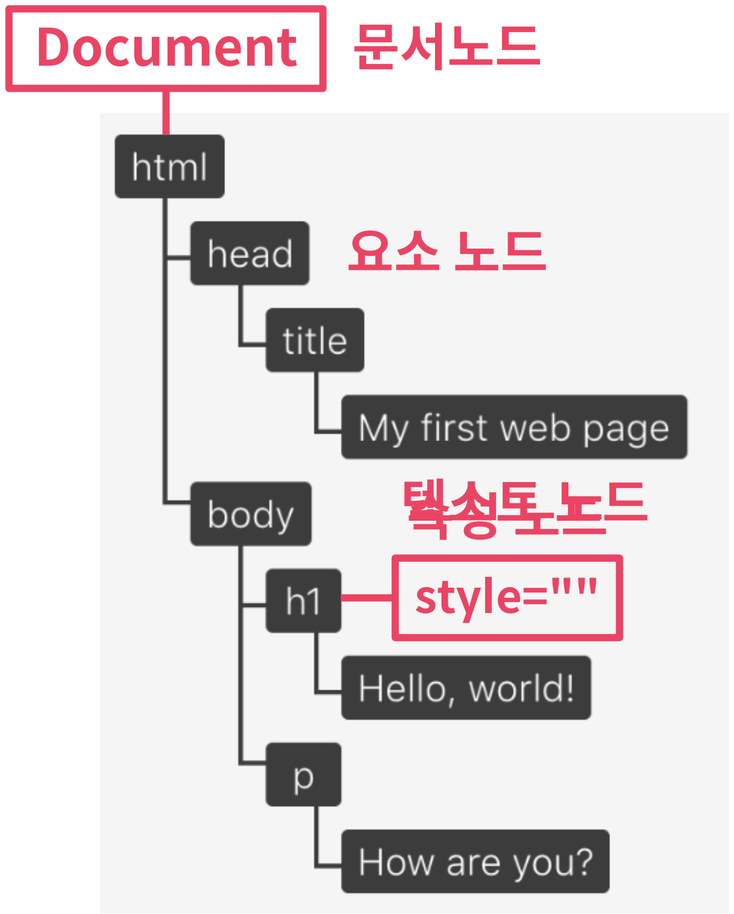
DOM (Document Object Model)

예제 17

```
<html>
    <head>
        <title>
            My first web page
        </title>
    </head>
    <body>
        <h1>Hello, world!</h1>
        <p>How are you?</p>
    </body>
</html>
```



노드의 종류



- **문서 노드 (document node)** : 문서 전체를 나타내는 노드
- **요소 노드 (element node)** : 모든 HTML 요소가 요소 노드

  (유일하게 속성노드를 가질 수 있음)
- **속성 노드 (attribute node)** : 모든 HTML 요소의 속성
- **텍스트 노드 (text node)** : 모든 HTML의 텍스트는 텍스트 노드
- **주석 노드 (comment node)**

DOM 요소 접근

예제 18

```
let ex1 = document.getElementById("ex1");
let ex2 = document.getElementsByClassName("sub");
let ex3 = document.getElementsByTagName("div");
let ex4 = document.getElementsByName("checkOption");

// CSS Selector를 사용해서 노드 선택
let ex5 = document.querySelector("#ex5");
let ex6 = document.querySelectorAll("div.sub");
```

- 다양한 함수를 통해서 각 요소에 접근이 가능
- 상황에 맞춰서 골라 사용하면 됨

DOM 요소 생성

예제 19

```
let newElem = document.createElement("div");
let newText = document.createTextNode("안녕하세요?");

newElem.appendChild(newText); // 자식 노드로 텍스트 노드를 추가

let ex1 = document.getElementById("ex1");
ex1.appendChild(newElem); // ex1 요소 아래에 새로 만든 div 요소를 추가
```

- 없던 요소를 추가할 수 있음
- 반대로 제거를 위한 **"removeChild"**도 있음

DOM 요소 수정

예제 20

```
let ex1 = document.getElementById("ex1");
ex1.setAttribute("value", "값을 수정합니다.");
ex1.removeAttribute("value");
console.log(ex1.innerText);
console.log(ex1.innerHTML);
ex1.innerHTML = "<h1>안녕하세요?</h1>";
ex1.style.color = "blue";
```

- 다양한 방법으로 요소를 변경할 수 있음
- 하나씩 다 해보도록 하자

그럼 이제 이 동작을 한번 구현해보자!

클릭하면

DOM을 활용해서 1. create 방식 / 2. innerHTML 방식

클릭하면 Modal이 새로 생김

연락처

TEL

070-4633-2015

WEB

https://elice.io

E-MAIL

contact@elice.io