

# Part 1

## 자바스크립트 기초



코딩 자율학습

제로초의 자바스크립트 입문



# 1장 Hello, JavaScript!

---

1.1 자바스크립트를 시작하기 전에

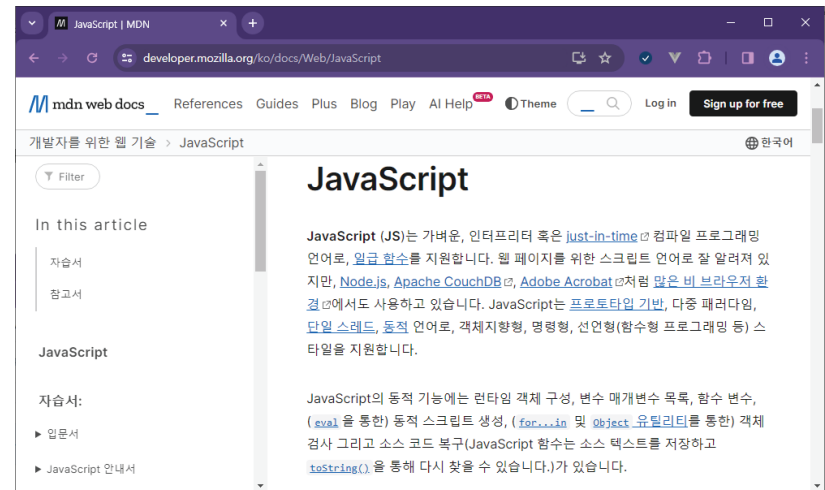
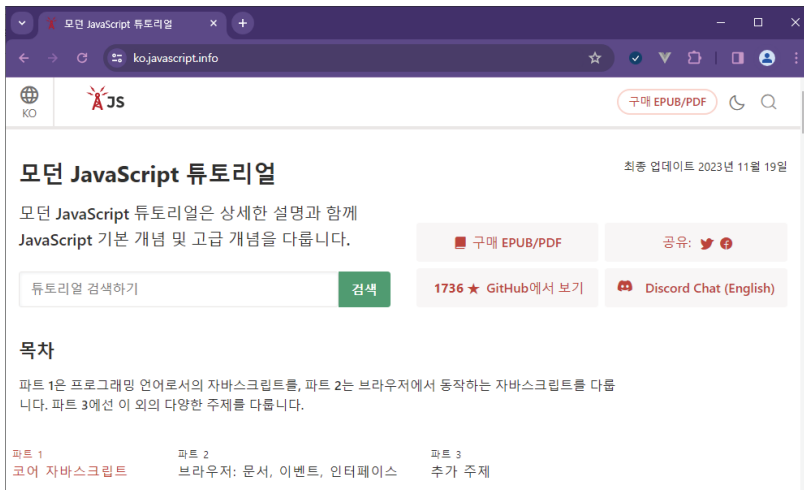
1.2 프로그래밍 사고력 기르기

# 1.1 자바스크립트를 시작하기 전에



## 1.1.1 자바스크립트를 배울 때 도움이 되는 자료

- 모던 자바스크립트 튜토리얼  
<https://ko.javascript.info>
- MDN 웹 문서  
<https://developer.mozilla.org/ko/docs/Web/JavaScript>



# 1.1 자바스크립트를 시작하기 전에



## 1.1.2 Visual Studio Code 설치하기

- VSCode 공식 사이트에서 운영체제에 맞는 버전을 선택해 다운로드  
<https://code.visualstudio.com/download>

The screenshot shows the Visual Studio Code download page. At the top, it says "Download Visual Studio Code" and "Free and built on open source. Integrated Git, debugging and extensions." Below this, there are three main sections: Windows, Linux, and Mac. The Windows section is highlighted with a red box. It shows the Windows logo and a blue button with a download icon and the text "Windows" and "Windows 10, 11". Below the button, there are links for "User Installer", "System Installer", ".zip", and "CLI", each with "x64" and "Arm64" options. The Linux section shows the Tux penguin logo and two blue buttons: ".deb" for "Debian, Ubuntu" and ".rpm" for "Red Hat, Fedora, SUSE". Below these, there are links for ".deb", ".rpm", ".tar.gz", "Snap", and "CLI", each with "x64", "Arm32", and "Arm64" options. The Mac section shows the Apple logo and a blue button with a download icon and the text "Mac" and "macOS 10.15 +". Below the button, there are links for ".zip" and "CLI", each with "Intel chip", "Apple silicon", and "Universal" options.

Visual Studio Code

### Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

**Windows**  
Windows 10, 11

User Installer: x64, Arm64  
System Installer: x64, Arm64  
.zip: x64, Arm64  
CLI: x64, Arm64

**.deb**  
Debian, Ubuntu

**.rpm**  
Red Hat, Fedora, SUSE

.deb: x64, Arm32, Arm64  
.rpm: x64, Arm32, Arm64  
.tar.gz: x64, Arm32, Arm64  
Snap: Snap Store  
CLI: x64, Arm32, Arm64

**Mac**  
macOS 10.15 +

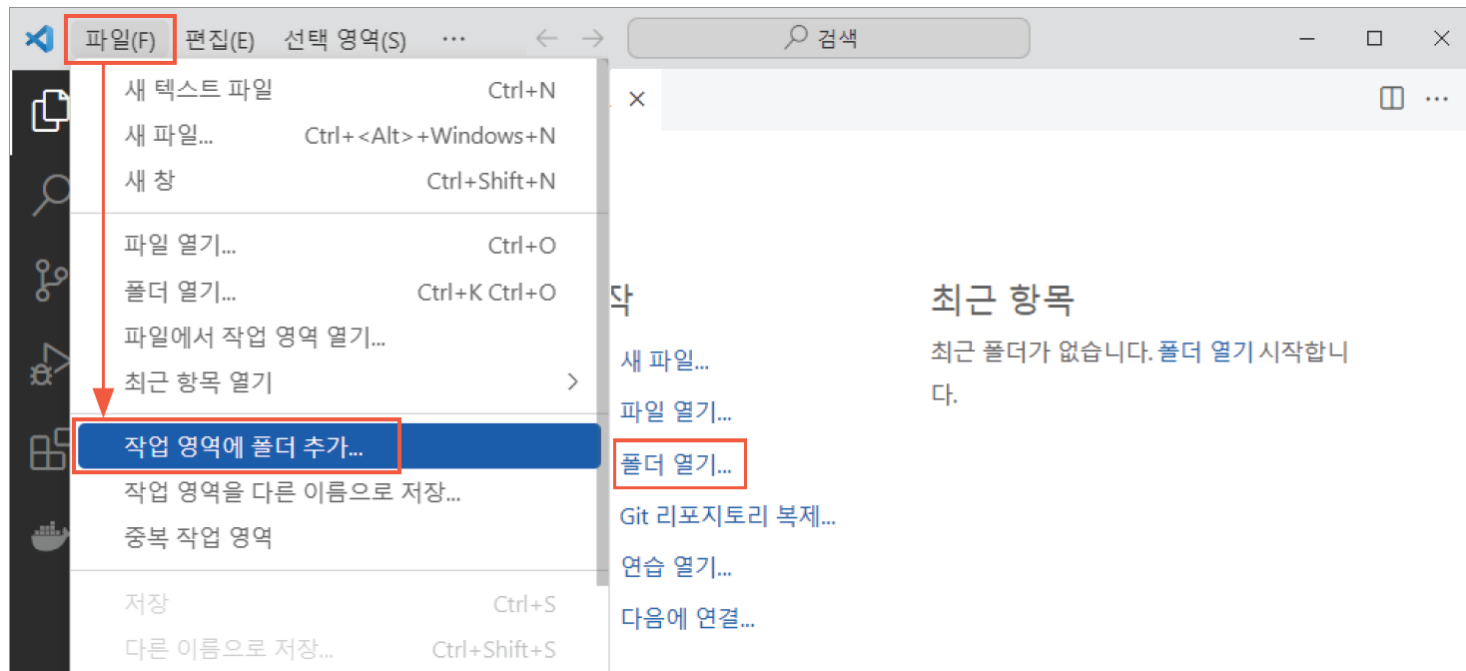
.zip: Intel chip, Apple silicon, Universal  
CLI: Intel chip, Apple silicon

# 1.1 자바스크립트를 시작하기 전에



## 1.1.3 프로젝트 폴더 선택하기

- 설치가 끝나면 VSCode에서 작성한 자바스크립트 소스 파일(스크립트 파일)을 저장할 폴더 지정

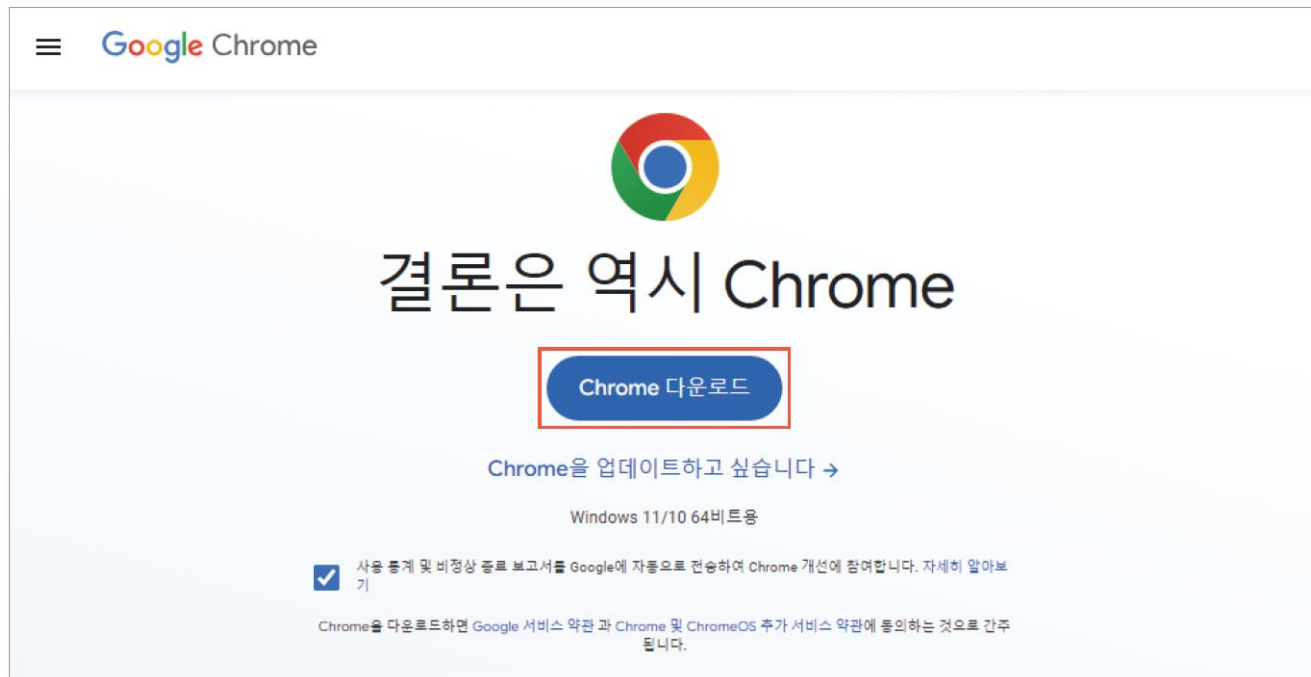


# 1.1 자바스크립트를 시작하기 전에



## 1.1.4 크롬 설치하기

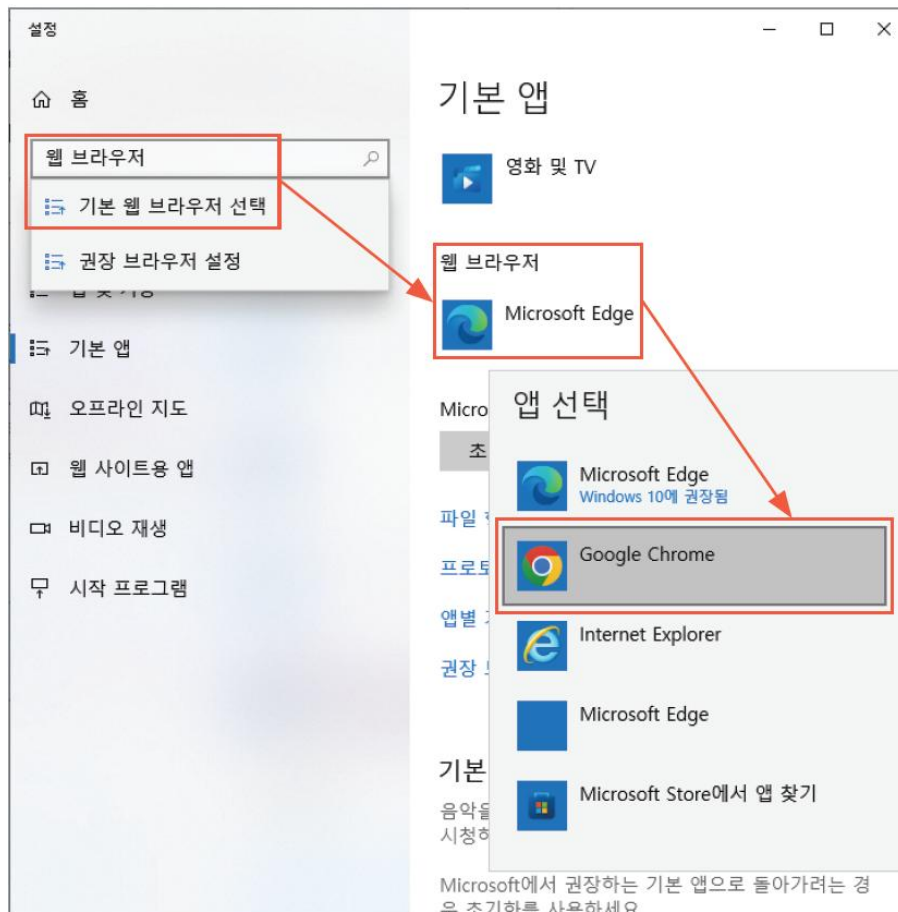
- 크롬 공식 사이트에서 다운로드  
[https://www.google.com/intl/ko\\_ALL/chrome](https://www.google.com/intl/ko_ALL/chrome)



# 1.1 자바스크립트를 시작하기 전에



- 크롬을 기본 웹 브라우저로 설정

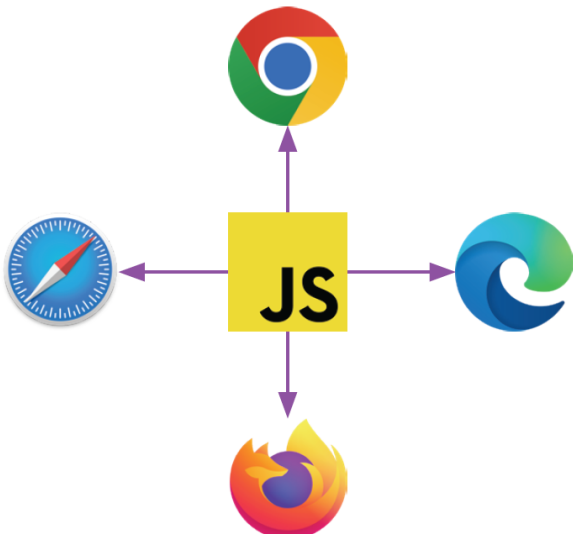


# 1.1 자바스크립트를 시작하기 전에



## 1.1.5 웹 브라우저 콘솔 사용법 익히기

- 자바스크립트 코드 모든 웹 브라우저에서 실행 가능
- 웹 브라우저에서 자바스크립트 코드를 실행할 수 있는 이유  
→ 자바스크립트 코드를 실행하는 자바스크립트 엔진이 내장되어 있기 때문에



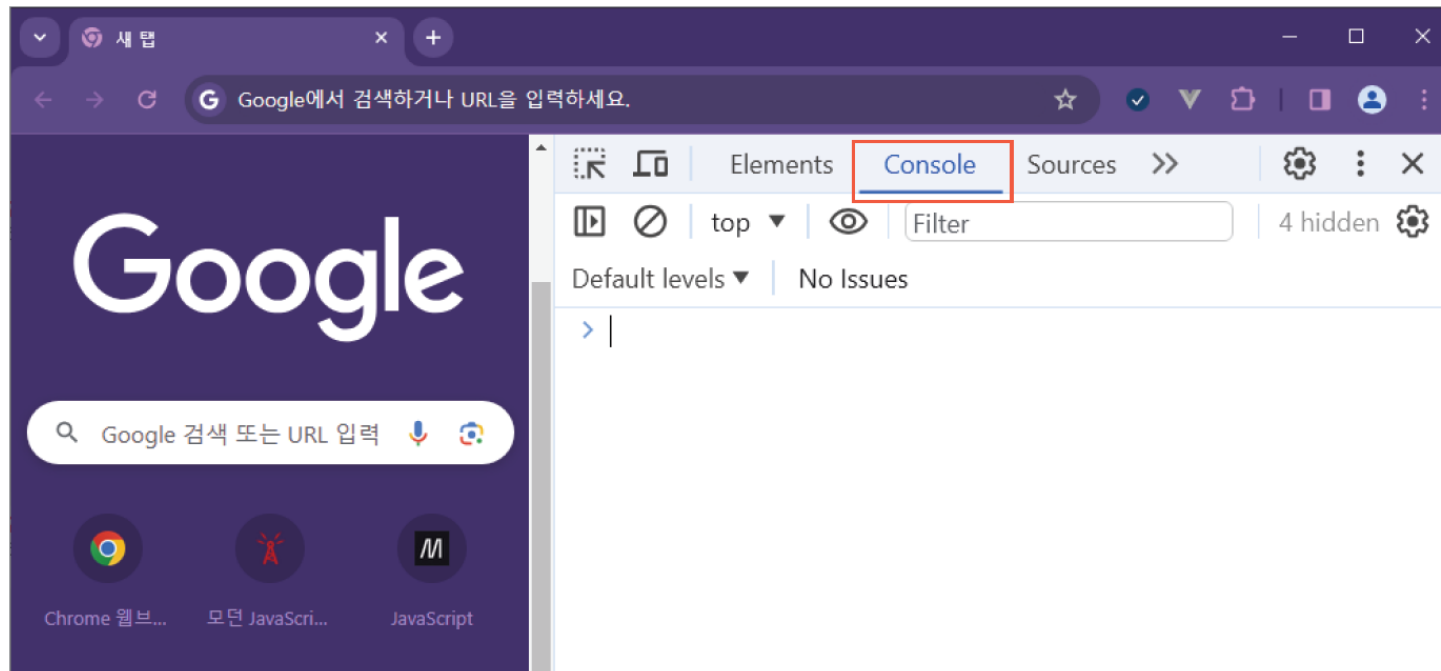
웹 브라우저	자바스크립트 엔진
크롬, 엣지, 오페라	V8
파이어폭스	스파이더몽키
사파리	자바스크립트코어



# 1.1 자바스크립트를 시작하기 전에



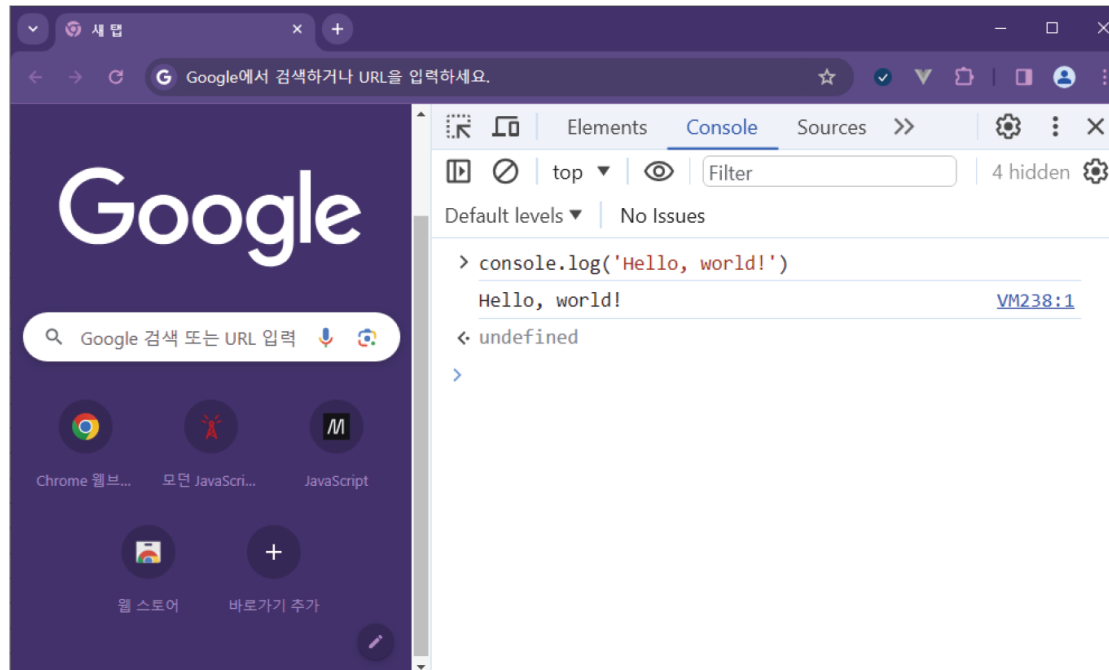
- 크롬 개발자 도구(원: F12, 맥: option + command + I)의 콘솔(Console)에서 자바스크립트 코드의 실행 과정을 볼 수 있음
- 콘솔에 보이는 > 표시를 프롬프트(prompt)라고 함



# 1.1 자바스크립트를 시작하기 전에



- 프롬프트에 코드 입력 후 Enter를 누르면 실행 결과를 확인할 수 있음
- 콘솔에 Hello, world! 출력



# 1.1 자바스크립트를 시작하기 전에

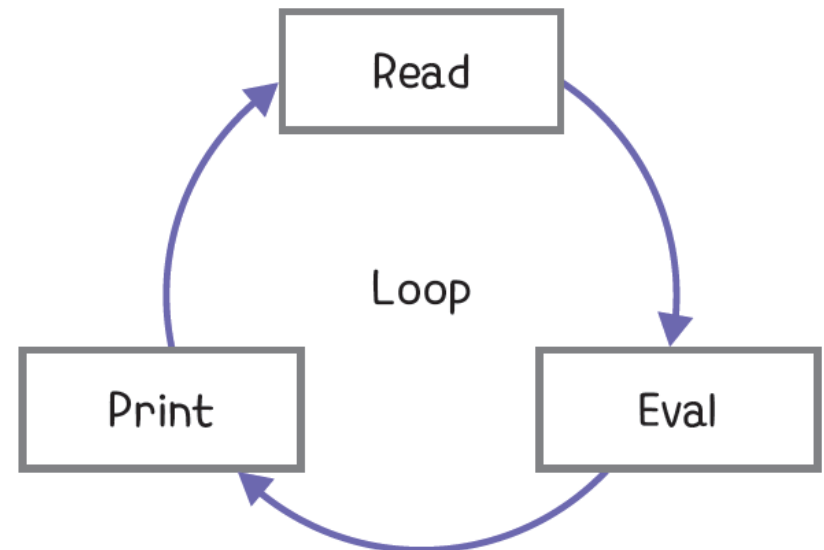


- 인터프리터(interpreter) 방식
  - 코드를 한 덩어리씩 실행해 결과를 출력하는 방식
  - 예: 자바스크립트 등
- 컴파일(compile) 방식
  - 컴퓨터가 이해할 수 있는 언어로 변환하는 과정을 거친 후 한 번에 실행하는 방식
  - 예: C, C++, 자바 등

# 1.1 자바스크립트를 시작하기 전에



- REPL(Read-Eval-Print Loop)
  - 코드를 한 줄씩 입력(Read)받아
  - 이를 평가(Eval)하고
  - 결과를 출력(Print)한 뒤,
  - 다시 프롬프트가 나타나서 새로운 입력을 기다리는 과정을 반복 (Loop)



## 1.2 프로그래밍 사고력 기르기



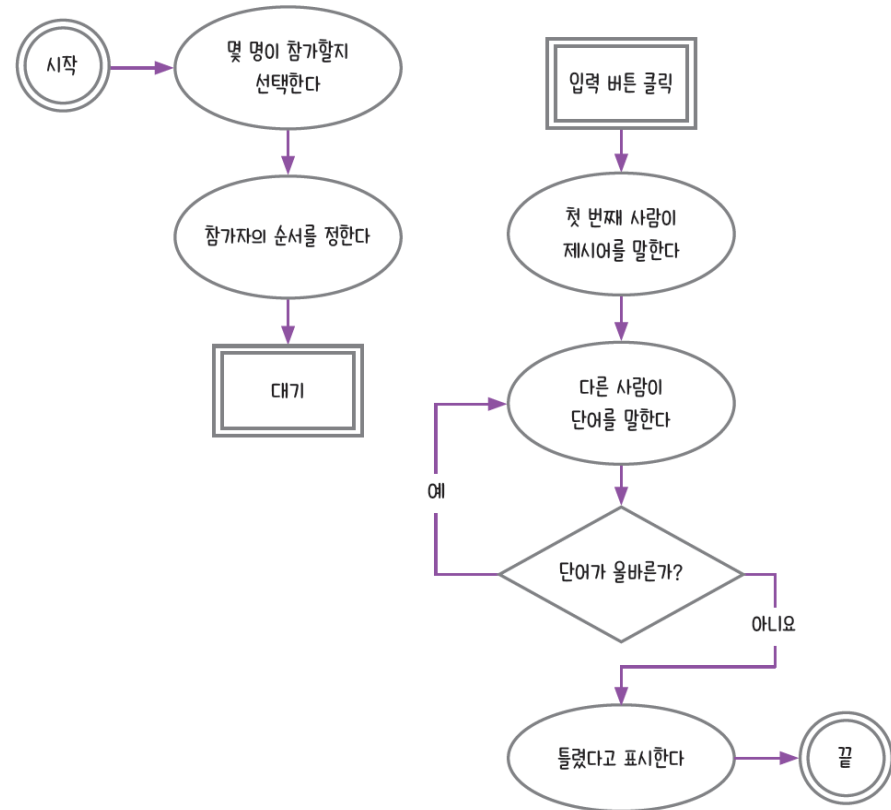
- 프로그래밍 사고력(programming thinking)
  - 프로그램이 수행하길 원하는 행동을 명확한 순서와 절차로 설명하는 것
  - '컴퓨팅 사고력(computational thinking)'이라고도 함

## 1.2 프로그래밍 사고력 기르기



### 1.2.1 프로그래밍 사고력 훈련법



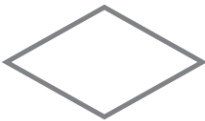


- 순서도(flowchart): 프로그램이 수행하는 명령에 대한 순서와 절차를 도형과 기호를 사용해 도식화한 것



## 1.2 프로그래밍 사고력 기르기



- 이 책에서 순서도에 사용하는 도형과 기호

도형 또는 기호	의미
 두 겹의 원	시작과 끝
 타원	일반 절차
 마름모	판단 절차
 두 겹의 사각형	특수한 상황(대기, 이벤트 발생)
 화살표	다음 절차로 가는 흐름

# 2장 기본 문법 배우기

---

2.1 코드 작성 규칙

2.2 자료형

2.3 변수

2.4 조건문

2.5 반복문

2.6 객체

2.7 클래스



## 2.1 코드 작성 규칙



### 2.1.1 세미콜론

- 세미콜론(;)
  - 붙여도 되고 붙이지 않아도 됨
  - 붙이기를 권장
- 한 줄에 여러 명령을 넣을 때는 세미콜론으로 구분

```
> console.log('Hello, world!'); console.log('Hello, javascript!'); console.log('Hello');
```

## 2.1 코드 작성 규칙



### 2.1.2 주석

- 주석(comment)
  - 사람만 알아볼 수 있도록 설명을 작성한 부분
  - 코드에 영향을 미치지 않음
- 한 줄 주석: // 기호 뒤에 작성

---

```
console.log('Hello, comment!'); // Hello, comment! 출력
```

---

- 여러 줄 주석: /\* \*/ 기호로 감싼, 안쪽에 작성

---

```
/* console.log('Hello, world!');  
console.log('Hello, comment!'); */
```

---

## 2.1 코드 작성 규칙



### 2.1.3 들여쓰기

- 제한 없음
- 단, 이 책에서는 띄어쓰기 2칸으로 통일함
- 규칙적으로 사용하면 코드의 가독성을 높일 수 있음

---

```
if (condition) {  
    console.log('Hello, world!');  
}
```

---

## 2.2 자료형



- 값(value): 프로그램에서 조작할 수 있는 데이터
- 자료형(data type): 값의 종류

### 2.2.1 문자열

- 문자열(string): 문자들이 하나 이상 나열되어 있는 자료형
- 시작과 끝이 따옴표로 감싸진 값
- 시작과 끝을 같은 종류의 따옴표로 감싸야 함
- 연산자(operator): 어떠한 값에 특정 작업을 수행하라는 의미를 나타내는 기호

## 2.2 자료형



- 문자열 안에 따옴표 사용하기
  - 따옴표가 문자열 중간에 들어 있을 때
  - 문자열을 다른 종류의 따옴표로 감싸야 함

↓ ↓ ↓ ↓

'문자열 안에 작은따옴표(')가 있어요. ';      '문자열 안에 작은따옴표(')가 있어요. ';

사람이 인식하는 문자열의 시작과 끝      자바스크립트 엔진이 인식하는 문자열의 시작과 끝

- 이스케이핑(escaping): 기호를 다르게 해석하게 하는 행위
- 예: 백슬래시(\)가 붙은 따옴표는 일반 문자로 해석

---

> "문자열 안에 큰따옴표(\" )가 있어요. ";  
< '문자열 안에 큰따옴표(" )가 있어요. '  
> '문자열 안에 작은따옴표(\')가 있어요. ' ;  
< "문자열 안에 작은따옴표(')가 있어요. "

---

## 2.2 자료형



- 한 문자열을 여러 줄로 표시하기
  - 문자열에서 행갈이할 부분 앞에 \n 문자 넣기

```
> alert('여러 줄에 걸쳐\n표시됩니다.');
```

chrome://new-tab-page 내용:

여러 줄에 걸쳐  
표시됩니다.

확인

## 2.2 자료형



- 템플릿 리터럴 사용하기
  - 템플릿 리터럴(template literal): 백틱(`, backtick 또는 backquote)으로 감싸진 문자열
  - 백틱 문자열을 사용하면 \n 문자를 사용하지 않아도 행갈이 할 수 있음

> alert(`여러 줄에 걸쳐  
표시됩니다.

줄을 더 늘려 볼까요?`);

< undefined

chrome://new-tab-page 내용:

여러 줄에 걸쳐  
표시됩니다.

줄을 더 늘려 볼까요?

확인

## 2.2 자료형



- 문자열 합치기
  - 두 문자열 사이에 + 기호를 두면 양쪽 문자열이 하나로 합쳐  
진다

---

> '문자열이 긴 경우에는 문자열을 ' + '나눈 뒤 다시 합칩니다.';

< '문자열이 긴 경우에는 문자열을 나눈 뒤 다시 합칩니다.'

---



## 2.2 자료형



### 2.2.2 숫자

- 숫자(number): 따옴표 없이 숫자 그대로 작성

```
5;
```

- 따옴표로 감쌀 경우 문자열이 됨

<pre>&gt; typeof 5;</pre>	<pre>&gt; typeof '5';</pre>
<pre>&lt; 'number'</pre>	<pre>&lt; 'string'</pre>

- 지수 표기법(exponential notation)을 사용할 수 있음

```
5e4; // 5 * 104(10000) = 50000(+는 생략 가능)
```

```
5e+4; // 5 * 104(10000) = 50000
```

```
5e-3; // 5 * 10-3(1/1000) = 0.005
```

## 2.2 자료형



- 문자열을 숫자로 바꾸기
  - parseInt()와 Number() 함수 사용
  - parseInt() 함수: 문자열을 정수로만 바꿈
  - 문자열을 실수로 바꾸고 싶으면 parseFloat() 함수 또는 Number() 함수 사용
- NaN
  - NaN: Not a Number(숫자가 아님)
  - 주의: 이름과는 다르게 숫자

## 2.2 자료형



- 산술 연산자 사용하기
  - $+$ ,  $-$ ,  $*$ ,  $/$  등의 기호를 사용
  - $\%$  연산자: 나눗셈의 나머지를 구하기
  - $**$  연산자: 숫자를 거듭제곱
- 무한 값
  - Infinity: 무한 값
  - - Infinity: 음수 무한 값. 음수를 0으로 나누는 경우
  - NaN: 무한끼리 연산할 때(연산이 성립하지 않는다는 의미)

## 2.2 자료형



- 문자와 숫자 더하기
  - 형 변환(type casting): 값의 자료형이 바뀌는 현상 또는 바꾸는 행위
  - + 연산자를 사용할 때는 숫자보다 문자열이 우선시
  - 예: 문자열 '1'과 숫자 0을 더하면 숫자 1이 아니라 문자열 '10'이 나옴
  - - 연산자를 사용할 때는 다른 자료형이 먼저 숫자로 형 변환된 후 연산
  - 예: 문자열 '9'는 숫자 9로 형 변환되고, 여기서 5를 빼기 때문에 숫자 4가 나옴

## 2.2 자료형



- 연산자 우선순위 이해하기
  - 연산자 우선순위가 높을수록 먼저 계산
  - 같은 우선순위의 연산자가 여러 개 나오면 먼저 나온 순서대로 계산

우선순위	연산자(심표로 구분)	우선순위	연산자(심표로 구분)
19	()(그룹화)	9	==, !=, ===, !==
18	., [], new, ()(함수 호출), ?.	8	&
17	new(인수 없이)	7	^
16	++(후위), --(후위)	6	!
15	!, ~, +(단항), -(단항), ++(전위), --(전위), typeof, void, delete, await	5	&&
14	**	4	, ??
13	*, /, %	3	? : (삼항 연산자)
12	+(다항), -(다항)	2	=, +=, -=, **=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=,  =, &&=,   =, ??=, yield, yield*
11	<<, >>, >>>		
10	<, <=, >, >=, in, instanceof	1	, (심표)

## 2.2 자료형



- 실수 연산 시 주의할 점
  - 부동소수점 문제: 10진법으로 계산한 결과와 차이가 발생
  - 원인: 2진법으로 실수를 표현하면 무한 반복되는 실수가 있어서 어쩔 수 없이 근사값으로 저장
  - 해결 방법: 실수를 정수로 바꿔 계산하고 마지막에 다시 실수로 바꾸기

## 2.2 자료형



### 2.2.3 불 값

- 불 값(boolean): 참(true)과 거짓(false)을 나타내는 자료형
- 불 값 표현하기
  - 따옴표로 감싸지 않고 true와 false를 입력

---

```
> typeof true;  
< 'boolean'
```

---

## 2.2 자료형



- 비교 연산자 사용하기
  - >: 왼쪽 값이 오른쪽 값보다 크다(초과)
  - <: 왼쪽 값이 오른쪽 값보다 작다(미만)
  - >=: 크거나 같다(이상)
  - <=: 작거나 같다(이하)
  - ==: 양쪽 값이 같은지 비교
  - !=: 양쪽 값이 다른지 비교

> 5 > 3;	> 5 >= 5;
< true	> true
> 5 < 3;	> 5 <= 4;
< false	> false



## 2.2 자료형



- ==와 ===의 다른 점
  - ==: 값이 같은지 비교. 자료형이 달라도 값이 같으면 true
  - ===: 값뿐 아니라 자료형까지 같은지 비교. 자료형까지 같을 때만 true

## 2.2 자료형



- 논리 연산자 사용하기
  - && 연산자: '그리고'를 의미  
왼쪽 식과 오른쪽 식이 모두 true여야 true
  - || 연산자: '또는'을 의미  
왼쪽 식이나 오른쪽 식 둘 중 하나라도 true면 true
- 논리 연산자 사용 시 유의할 점
  - && 연산자: 앞에 나오는 값이 참인 값이면 뒤에 나오는 값 결과, 앞에 나오는 값이 거짓인 값이면 앞에 나오는 값이 결과
  - || 연산자: 앞에 나오는 값이 참인 값이면 앞에 나오는 값 결과, 앞에 나오는 값이 거짓인 값이면 뒤에 나오는 값이 결과
  - ?? 연산자(널 병합 연산자, nullish coalescing operator): 앞에 나오는 값이 null이나 undefined면 뒤에 나오는 값이 결과, null도 undefined도 아니면 앞에 나오는 값이 결과

## 2.2 자료형



### 2.2.4 빈 값 사용하기

- undefined
  - 빈 값(비어 있음을 의미)
  - 값이자 자료형
  - 보통 반환할 결과 값이 없을 때 기본 값으로 사용
- null
  - 빈 값
  - undefined와 같지 않음

---

```
> undefined === null;  
< false
```

---

## 2.3 변수



- 변수(variable): 값을 저장하고 저장한 값을 불러올 수 있게 하는 것
- 변수 선언(declaration): 변수를 만드는 행위

### 2.3.1 let으로 변수 선언하기

- 변수를 선언하는 방법
  - let(또는 const, var) 다음에 선언하려는 변수명(변수의 이름) 적고 그 뒤에 대입(assignment, 할당) 연산자 = 입력, = 연산자 뒤에는 변수에 저장할 식 입력

형식

`let` 변수명 = 식;

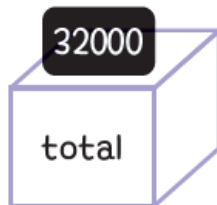
## 2.3 변수



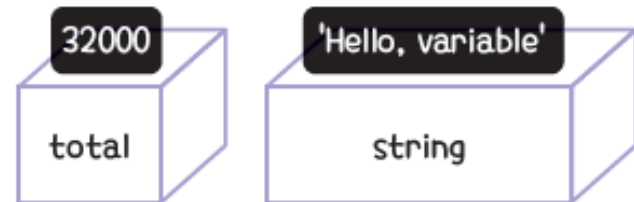
- 선언문: let으로 시작하는 명령
- 초기화(initialization): 변수를 선언함과 동시에 값을 대입하는 행위

```
> let total = 5000 + 8000 + 10000 + 9000;  
< undefined
```

```
> let string = 'Hello, variable';  
< undefined  
> string;  
< 'Hello, variable'
```



메모리



메모리

## 2.3 변수



### 2.3.2 변수명 짓기

- 변수명: 변수의 값이 무엇인지 알려 주는 역할을 하므로 자세하게 짓기
- 변수명의 제약 사항
  - 특수문자는 \$와 \_만 사용할 수 있다.
  - 숫자로 시작해서는 안 된다.
  - 예약어(reserved word)는 변수명으로 사용할 수 없다.
  - 위의 제약 사항을 어기지 않는다면 한글, 한자, 유니코드도 가능

## 2.3 변수



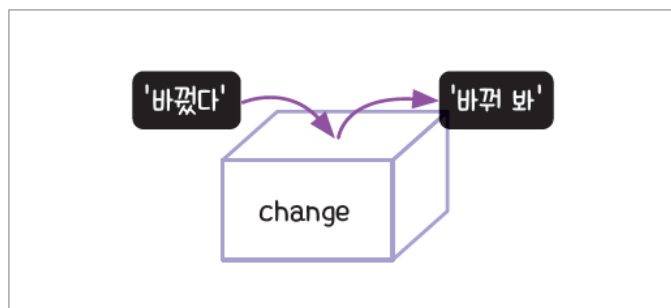
### 2.3.3 변수의 값 수정하기

- 변수에 넣은 값을 바꿀 때: 대입 연산자를 사용해 새로운 값 입력

```
> let change = '바꿔 봐';  
< undefined
```

```
> change = '바꿨다';  
< '바꿨다'
```

```
> change;  
< '바꿨다'
```



메모리

- 변수에 넣은 값을 비울 때: undefined를 대입하거나 null을 대입

## 2.3 변수



### 2.3.4 변수 활용하기

- 변수는 계산된 값을 저장할 때
- 코드 중복을 줄일 때

### 2.3.5 const로 상수 선언하기

- const: 상수(constant)의 줄임말
- 한번 값을 대입하면 다른 값을 대입할 수 없다는 특성 때문에 상수 선언 시 초기화(선언과 동시에 값을 대입하는 것)하지 않으면 에러 발생

```
> const value = '상수입니다.';
```

```
< undefined
```

```
> value = '바꿀 수 없습니다.';
```

```
Uncaught TypeError: Assignment to constant variable.
```



## 2.3 변수



### 2.3.6 var 알아보기

- var: 변수(variable)의 줄임말
- 변수문(variable statement): var로 변수를 선언한 문장
- 기존에 선언한 변수를 다시 선언해도 에러가 발생하지 않음
- 예약어에 사용하는 단어를 변수명으로 사용할 수 있음
- var 대신 let을 사용하면 에러가 발생해 해당 이름을 변수명으로 사용하지 못하게 막음

## 2.4 조건문



- 조건문: 주어진 조건에 따라 코드를 실행하거나 실행하지 않는 문

### 2.4.1 조건문의 기본 형식

- 조건문을 나타내는 예약어인 if 뒤에 나오는 소괄호 안에 조건(식)을 넣고, 다음 줄에 실행문을 넣기

형식    **if** (조건식)  
         실행문

형식    **if** (조건식) {  
         실행식1;  
         실행식2;  
         실행식3;  
         }    실행문

## 2.4 조건문



- 조건문은 조건식과 실행문으로 구분
- 조건식이 참인 값이면 실행문이 실행되고, 거짓인 값이면 실행문이 실행되지 않음
- 실행문에는 여러 개의 식을 넣을 수 있음
- 실행문의 식이 둘 이상이면 식들을 중괄호로 감싸기

## 2.4 조건문



### 2.4.2 else를 사용해 두 방향으로 분기하기

- if 문 뒤에 else를 붙이고 실행문 입력
- else 문도 실행문에 식을 여러 개 넣을 수 있음
- 식이 하나인 경우 중괄호 생략 가능

## 2.4 조건문



### 2.4.3 else if를 사용해 여러 방향으로 분기하기

- if 문과 else 문 사이에 else if를 적고 그 뒤에 해당하는 조건식과 실행문 추가
- else if 문 뒤에 else 문은 생략 가능

형식

if (조건식)

실행문

else if (조건식)

실행문

else

실행문

형식

if (조건식)

실행문

else if (조건식)

실행문

else if (조건식)

실행문

## 2.4 조건문



- else if 문 실행문에 식을 여러 개 넣을 수 있음
- 식이 하나인 경우에는 중괄호 생략 가능
- if 문과 else 문 사이에 else if 문을 원하는 만큼 넣을 수 있음
- if 문과 else if 문만 사용해도 되고, if 문과 else 문을 사용해도 됨
- else if 문이나 else 문은 단독으로 사용할 수 없음

## 2.4 조건문



### 2.4.4 중첩 if 문 사용하기

- if, else, else if 문 안에 다시 조건문을 넣을 수 있음
- 중첩 if 문을 if-else if-else 문으로 변환해 코드의 가독성을 높이길 권장

## 2.4 조건문



### 2.4.5 switch 문으로 분기하기

형식    `switch` (조건식) {  
          `case` 비교 조건식:  
            실행문  
          }

- `switch` 문에는 조건식 2 개 사용
- `switch` 옆에 있는 소괄호의 조건식 값이 `case`의 비교 조건식 값과 일치(==)하면 해당하는 실행문이 실행



## 2.4 조건문



- 보통 조건식에 변수를 넣고, 비교 조건식에는 변수와 비교할 값을 넣음
- case를 여러 번 사용해 여러 방향으로 분기
- break 문을 사용해 case 문을 빠져나옴
- default를 사용해 어떤 것도 일치하지 않을 때 실행하는 case 문을 만들



**형식** 조건식 ? 참일 때 실행되는 식 : 거짓일 때 실행되는 식

- 
- ① condition1 ? condition2 ? '둘 다 참' : 'condition1만 참' : 'condition1이 거짓';
- ② 참인 경우 거짓인 경우
- ① condition1 ? 'condition1이 참' : condition2 ? 'condition2가 참' : '둘 다 거짓';
- ② 참인 경우 거짓인 경우

## 2.5 반복문



### 2.5.1 while 문으로 반복해서 출력하기

형식

`while` (조건식)  
실행문

형식

```
while (조건식) {  
    실행식1;  
    실행식2;  
    실행식3;  
}
```

- `while` 문: 조건식이 참일 동안 반복해서 실행문 실행

## 2.5 반복문



### 2.5.2 for 문으로 반복해서 출력하기

형식    **for** (시작; 조건식; 종료식)  
         실행문

- for 문 실행 순서

```
for (①let i = 0; ②i < 100; ④i++) {  
  ③console.log('Hello, for!');  
} // ②가 참이면 ②~④ 반복
```

- 시작이 먼저 실행되고, 그다음 조건식, 실행문, 종료식 순서로 실행
- 실행이 1번 끝나면 다시 조건식 검사
- 조건식이 참이라면 실행문과 종료식을 반복 실행

## 2.5 반복문



- for 문과 while 문의 실행 순서 비교

```
for (let i = 0; i < 100; i++) {  
  console.log('Hello, for!');  
} // ②가 참이면 ②~④ 반복
```

```
let i = 0;  
while (i < 100) {  
  console.log('Hello, while!');  
  i++;  
}
```

## 2.5 반복문



### 2.5.3 1부터 100까지 출력하기

- `i`가 0부터 시작하므로 `console.log(i + 1)`을 해야만 1부터 100까지 출력된다.

### 2.5.4 `break` 문으로 반복문 멈추기

- `break` 문: 반복문을 중간에 멈춰야 할 때 사용

### 2.5.5 `continue` 문으로 실행 건너뛰기

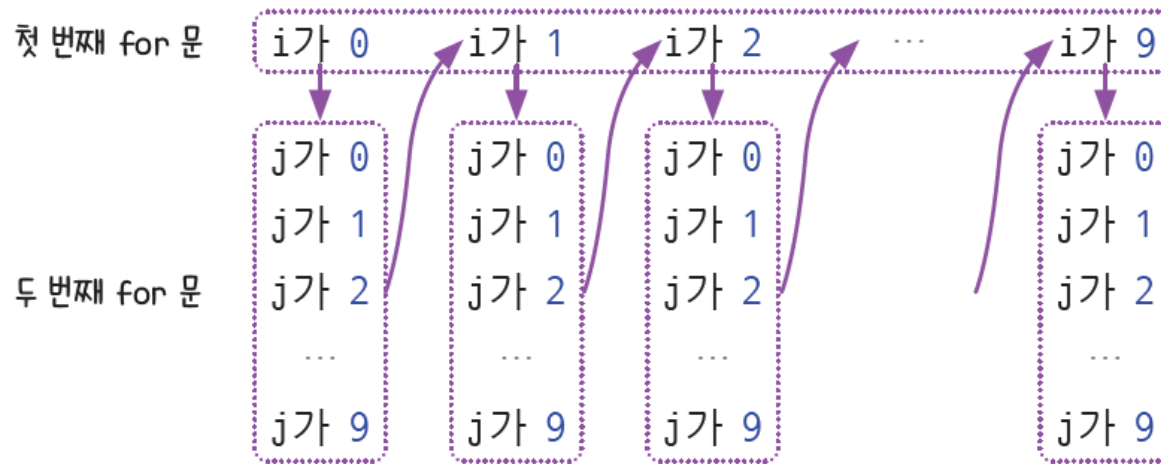
- `continue` 문: 반복문이 특정 조건에서만 실행되게 할 때 사용

## 2.5 반복문



### 2.5.6 중첩 반복문 사용하기

- 중첩 반복문: 반복문 안에 반복문이 들어 있는 경우
- 중첩 반복문의 실행 순서

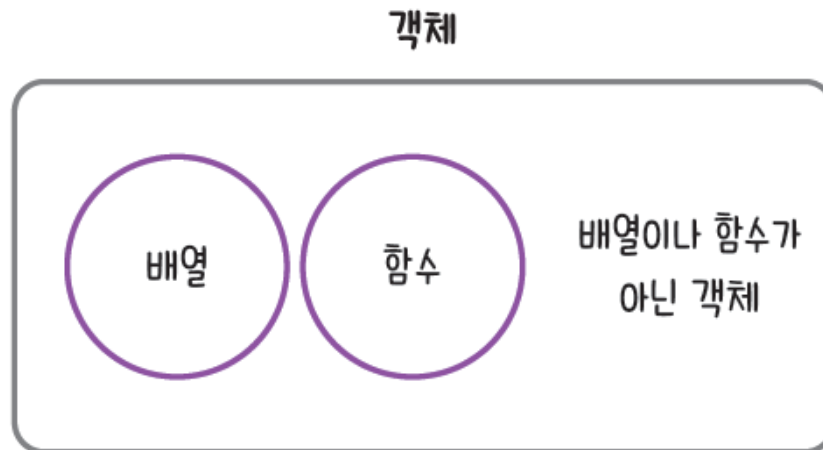


- 중첩 반복문 사용 시 중첩 횟수가 증가할수록 코드 복잡함
- 실무에서는 대부분 이중 이나 삼중 반복문 정도만 사용

## 2.6 객체



- 객체(object): 다양한 값을 모아 둔 또 다른 값
- 객체의 종류



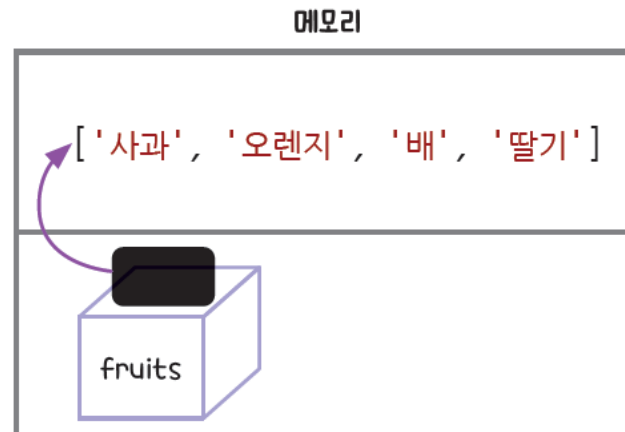


## 2.6 객체



### 2.6.1 배열

- 배열 생성하기
  - 대괄호([], 배열 리터럴)로 값들을 한꺼번에 감싸고, 각 값은 쉼표로 구분
  - 인덱스(index): 값의 자릿수로, 0부터 시작
  - 이차원 배열: 배열 안에 배열이 있는 경우
  - 요소(element): 배열의 값
  - 메모리에서 배열의 구조



## 2.6 객체



- 배열의 요소 개수 구하기
  - 배열 이름 뒤에 `.length` 붙이기
  - 빈 값도 유효한 값이므로 요소 개수를 셀 때 포함
  - 요소를 찾는 방법: 인덱스 사용하기, `at()` 사용하기
- 배열에 요소 추가하기
  - 원하는 배열의 인덱스에 값 대입
  - `unshift()`: 배열의 맨 앞에 새로운 요소를 추가할 때
  - `push()`: 배열에 값을 추가할 때

## 2.6 객체



- 배열의 요소 수정하기
  - 원하는 인덱스에 바꿀 값 넣기
- 배열에서 요소 삭제하기
  - `pop()`: 마지막 요소를 삭제할 때
  - `shift()`: 첫 번째 요소를 삭제할 때
  - `splice()`: 중간 요소를 삭제할 때
  - `splice(시작 인덱스, 삭제할 요소 개수)`로 작성하며, 숫자를 하나만 넣으면 배열 끝까지 삭제

## 2.6 객체



- 배열에서 요소 찾기
  - `includes()`: 주어진 값이 배열에 존재하는지 확인할 때 사용, 있으면 `true`, 없으면 `false`
  - `indexOf()`: 검색하려는 값이 어느 인덱스에 위치하는지 앞에서부터 찾을 때
  - `lastIndexOf()`: 검색하려는 값이 어느 인덱스에 위치하는지 뒤에서부터 찾을 때
  - `indexOf()`와 `lastIndexOf()` 사용 시 주의할 점 : 값뿐 아니라 자료형도 일치해야 함

## 2.6 객체



- 배열 자르고 합치기
  - `slice()`: 시작 인덱스부터 종료 인덱스까지 배열을 잘라 새 배열 만들기

**형식**    `배열.slice(<시작 인덱스>, <종료 인덱스>)`

- `concat()`: 두 배열을 합쳐 하나의 새로운 배열로 만들기

**형식**    `배열.concat(값1, 값2, ...)`

## 2.6 객체



- 배열과 비슷한 문자열의 특징
  - 문자열은 문자를 여러 개 이어 놓은 것이어서 이를 각각 분리할 수도 있음

형식 문자열[자릿수]

- 배열과 마찬가지로 0부터 시작
- 문자열 길이 구하기: 문자열 뒤에 .length

형식 문자열.length

- 문자열의 마지막 문자 구하기: 문자열[문자열.length - 1] 또는 at(-1)
- indexOf(), includes(), lastIndexOf()도 사용할 수 있다.

## 2.6 객체



- `join()`: 배열을 문자열로 만들 때
  - 소괄호 안에 아무 값이 없으면 배열의 요소를 쉼표로 합치기
  - 문자열이 있으면 해당 문자열을 요소 사이사이에 넣어 하나의 문자열로 만들기
- `split()`: 문자열을 배열로 만들 때
  - 소괄호 안에 값이 없으면 문자열이 배열의 첫 번째 요소가 됨
  - 소괄호에 빈 문자열을 넣으면 대상 문자열이 전부 개별 문자로 쪼개져 각각 배열의 요소가 됨
  - 소괄호 안에 넣은 문자열이 대상 문자열에 있으면 해당 문자열을 기준으로 대상 문자열을 나눔

## 2.6 객체



- 배열 반복하기
  - 배열은 값들을 나열한 것이라 반복문과 함께 사용하는 경우가 많음
  - while 문, for 문 모두 사용 가능
- 이차원 배열
  - 배열의 요소로 배열이 들어 있을 때
  - 이차원 배열의 대표적인 예

	A
1	1
2	2
3	3



## 2.6 객체



- flat()과 fill()
  - flat(): 배열의 차원을 한 단계 낮추기
  - fill(): 빈 배열의 요소를 특정 값으로 미리 채우기
- Set으로 중복 요소 제거하기
  - Set은 배열과 달리 중복을 허용하지 않음
  - 배열뿐 아니라 문자열 중복도 제거
  - size: Set의 요소 개수를 구할 때
  - Array.from(): Set을 배열로 바꿀 때

## 2.6 객체



### 2.6.2 함수

- 함수(function): 특정한 작업을 수행하는 코드
- 함수 선언하기
  - 함수 선언(declare): 함수를 만드는 행위
  - 익명 함수(anonymous function): 이름이 없는 함수

형식

```
function() {};  
() => {};
```

## 2.6 객체



- 함수 선언문(function declaration statement): 함수를 상수에 대입하지 않고 function 뒤에 함수의 이름을 넣는 방식

형식    `function` 이름() { 실행문 }

- 함수 표현식(function expression): 함수를 상수나 변수에 대입하는 방식

형식    이름 = `function`() { 실행문 }

- 화살표 함수(arrow function): 화살표 기호를 사용한 함수

형식    `() => { 실행문 }`  
          // 또는  
          `() => 반환식`

## 2.6 객체



- 함수 호출하기
  - 함수 호출(call): 함수를 사용하는 행위
  - 함수를 선언한 후 함수 이름 뒤에 ()를 붙여 호출하면 함수가

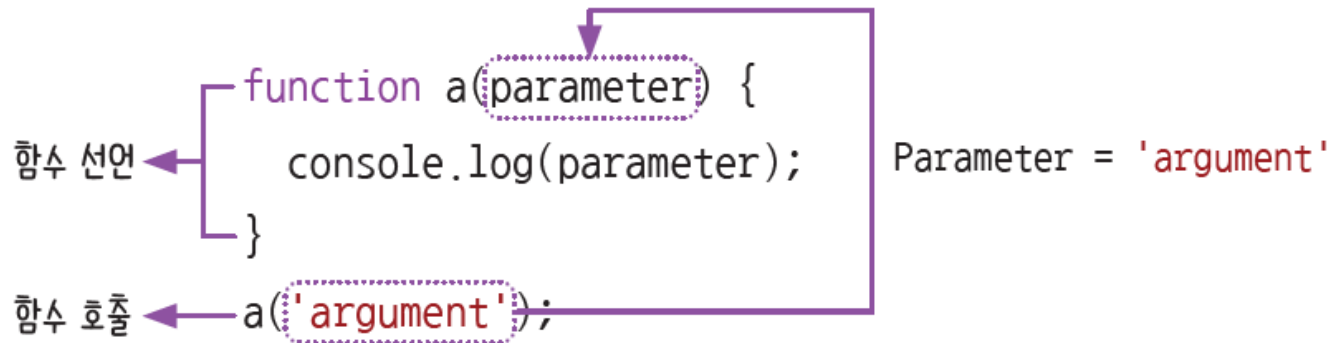
```
function a() {}  
a();  
< undefined
```

- return 문으로 반환값 지정하기
  - 반환값(return value): 함수를 호출하면 나오는 결과 값

## 2.6 객체



- 매개변수와 인수 사용하기
  - 인수(argument): 함수를 호출할 때 넣은 값
  - 매개변수(parameter): 함수를 선언할 때 사용한 변수
  - 매개변수와 인수의 관계



## 2.6 객체



- 다른 변수 사용하기
  - 순수 함수(pure function): 자신의 매개변수나 내부 변수(또는 상수)만 사용하는 함수
- 고차 함수 사용하기
  - 고차 함수(high order function): 함수를 만드는 함수
  - 함수 호출을 반환값으로 대체해 보기

```
const func = () => {  
  return () => {  
    console.log('hello');  
  };  
};  
  
const innerFunc = func();
```

함수 호출 부분을  
반환값으로 바꿔서  
생각해 보기

```
const innerFunc = () => {  
  console.log('hello');  
};
```

## 2.6 객체



### 2.6.3 객체 리터럴

- 객체 생성하기
  - 속성(property): 중괄호로 묶인 정보. 속성 이름과 속성 값으로 구분
  - 객체 리터럴(object literal): 중괄호를 사용해 객체를 표현하는 것

형식 {  
    <속성 이름>: <속성 값>,  
}

형식 {  
    <속성1 이름>: <속성1 값>,  
    <속성2 이름>: <속성2 값>,  
    <속성3 이름>: <속성3 값>,  
}

## 2.6 객체



- 객체 속성에 접근하기
  - 속성 이름을 통해 속성 값에 접근
  - 마침표(.) 사용 시: 변수.속성
  - 대괄호([]) 사용 시: 변수['속성']
- 객체 속성을 추가/수정/삭제하기
  - 추가 시: 변수.속성 = 값;
  - 수정 시: 변수.속성 = 값;
  - 삭제 시: delete 변수.속성;



## 2.6 객체



- 메서드 이해하기
  - 메서드(method): 객체의 속성 값으로 함수가 들어갔을 때 해당 속성을 메서드라고 함
- 객체 간 비교하기
  - 문자열, 불 값, null, undefined를 비교하면 모두 true 반환
  - NaN만 false 반환

## 2.6 객체



- 중첩된 객체와 옵셔널 체이닝 연산자
  - 객체 안에도 객체가 들어 갈 수 있음
  - 속성에 접근할 때 마침표와 대괄호를 조합 사용 가능
  - 변수.속성.['속성'], 변수['속성'].속성 형태로도 가능
  - ?. 연산자(옵셔널 체이닝, optional chaining operator): 존재하지 않는 속성에 접근할 때 에러가 발생하는 것을 막아 줌

---

```
zerocho.girlfriend?.name;
```

```
< undefined
```

```
zerocho.name?.first;
```

```
< '현영'
```

---

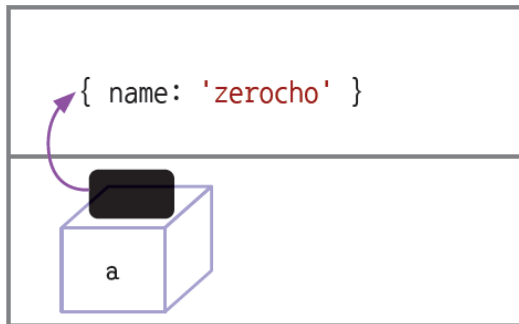
## 2.6 객체



- 참조와 복사

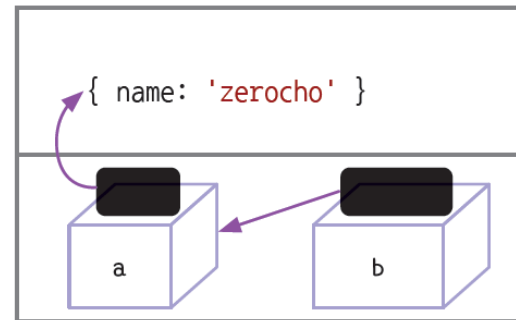
① 변수 a가 객체를 가리킴

메모리



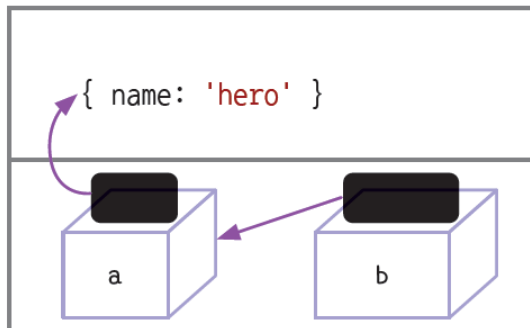
② 변수 b가 변수 a를 가리킴

메모리



③ 변수 a의 name 속성을 바꾸면 변수 b도 바뀐다

메모리

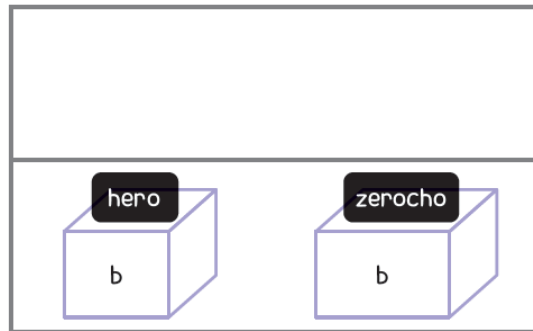


→ 이럴 때 변수 a와 b가 같은 객체를 참조하고 있다 또는 변수 a와 b 그리고 객체 간에 참조 관계가 있다고 표현

## 2.6 객체



- 객체가 아닌 값을 대입한 경우: 참조 관계가 생기지 않는 상황



- 복사(copy): 어떤 값을 다른 변수에 대입할 때 기존 값과 참조 관계가 끊기는 것
- 객체가 아닌 값은 애초부터 참조 관계가 없으므로 그냥 복사만 됨

## 2.6 객체



그림 2-29 배열 내부에 객체가 있을 때 메모리 구조

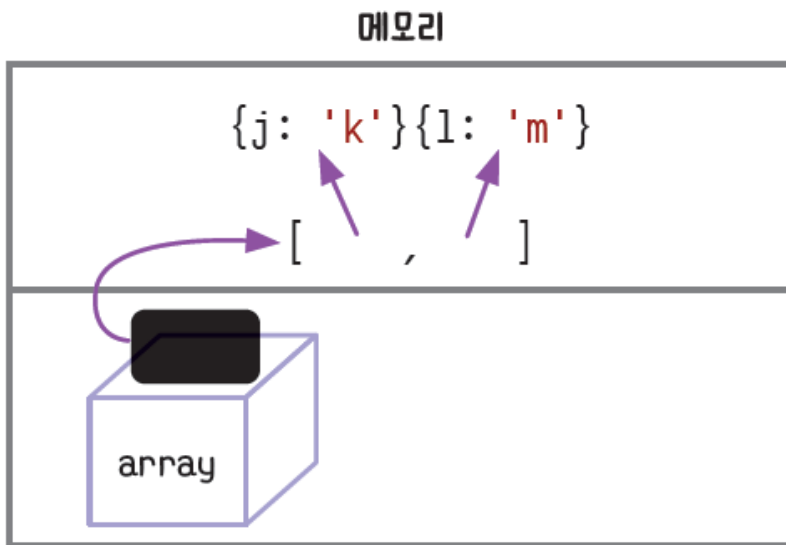
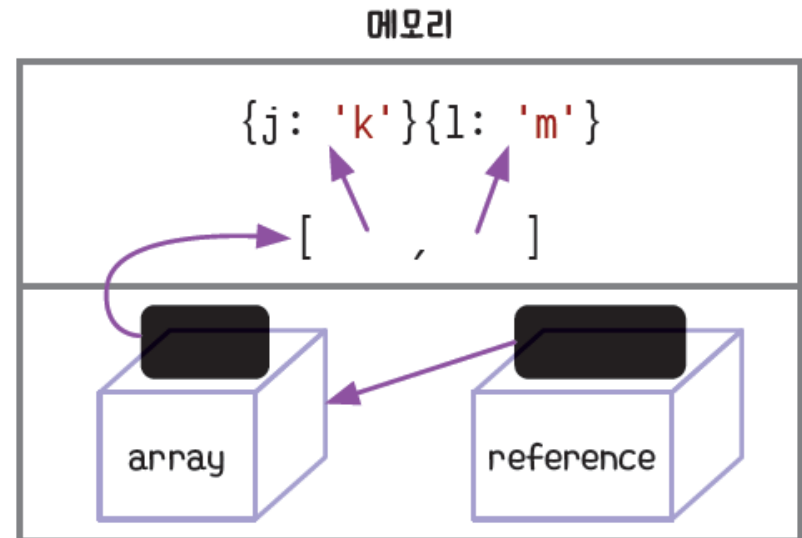


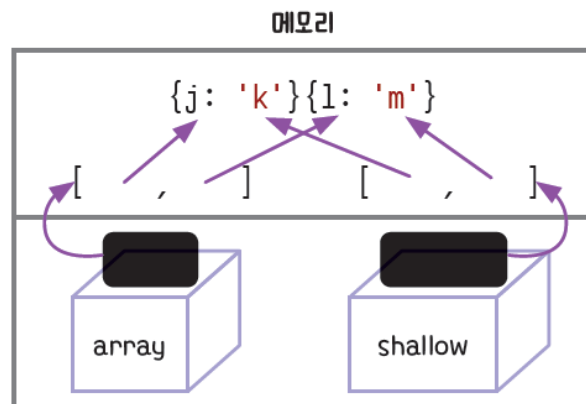
그림 2-30 배열 내부에 객체가 있는 배열을 참조할 때 메모리 구조



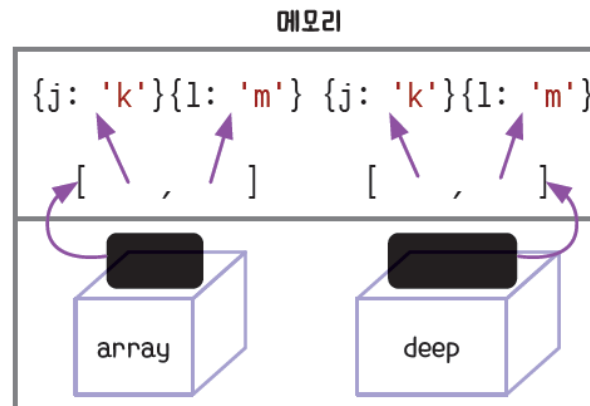
## 2.6 객체



- 얕은 복사(shallow copy): 외부 객체만 복사되고 내부 객체는 참조 관계를 유지하는 복사
- 얕은 복사 시... 연산자(스프레드 문법, spread syntax) 사용
- 깊은 복사(deep copy): 내부 객체까지 참조 관계가 끊기면서 복사 되는 것



얕은 복사 시 메모리 구조



깊은 복사 시 메모리 구조

## 2.6 객체



- 구조분해 할당(destructuring assignment)
  - 객체의 속성 이름과 대입하는 변수명이 같을 때 다음과 같이 줄여서 쓸 수 있음

---

```
const person = { name: '제로초' };  
const name = person.name;  
const { name } = person; // 앞 줄과 같은 의미  
name; // '제로초'
```

---

- 유사 배열 객체(array-like object)
  - 배열 모양을 한 객체
  - 배열이 아니므로 배열 메서드를 사용할 수 없음

## 2.6 객체



### 2.6.4 함수를 인수로 받는 배열 메서드

- forEach()와 map()
  - forEach(): for 문을 사용하지 않고도 반복문 수행 가능

형식    배열.forEach(함수);

- forEach()메서드의 인수

```
const arr = [1, 5, 4, 2];  
arr.forEach((number, index) => {  
  console.log(number, index);  
});
```



arr	1일 때	5일 때	4일 때	2일 때
number	1	5	4	2
index	0	1	2	3



## 2.6 객체



- 콜백 함수(callback function): 다른 메서드에 인수로 넣었을 때 실행되는 함수
- 메서드에 콜백 함수 전달 원리



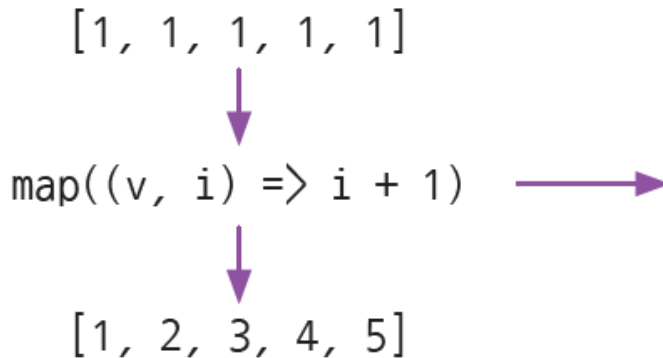
## 2.6 객체



- `map()`: 배열 요소들을 일대일로 짝지어서 다른 값으로 변환해 새로운 배열을 반환

형식    `배열.map(<콜백 함수>);`

- `map()` 메서드의 작동 방식



<b>v</b>	1	1	1	1	1
<b>i</b>	0	1	2	3	4
<b>i + 1</b>	1	2	3	4	5

## 2.6 객체



- find(), findIndex(), filter()
  - find(): 콜백 함수의 반환값이 true인 요소를 찾는 메서드
  - findIndex(): 찾은 요소의 인덱스를 반환하고, 찾지 못했다면 -1을 반환하는 메서드
  - filter(): 콜백 함수의 반환값이 true가 되는 모든 요소를 찾아 결과를 배열로 반환하는 메서드

형식

```
배열.find(<콜백 함수>);  
배열.findIndex(<콜백 함수>);  
배열.filter(<콜백 함수>);
```

## 2.6 객체



- sort()
  - 비교 함수의 반환값에 따라 배열을 정렬하는 메서드

형식    배열.sort(<비교 함수>);

- 비교 함수의 형식

형식    (a, b) => 반환값

## 2.6 객체



- reduce()
  - 배열에 있는 반복 메서드의 일종
  - 배열의 요소들을 하나의 값으로 합침
  - 초기 값이 없으면 배열의 첫 번째 요소가 초기 값이 됨

**형식** 배열.reduce((〈누적 값〉, 〈현재 값〉) => {  
    return 〈새로운 누적 값〉;  
}, 〈초기 값〉);

## 2.6 객체



- every()와 some()
  - every(): 하나라도 조건을 만족하지 않는 요소(조건 함수가 false를 반환)를 찾으면 반복 중단
  - some(): 하나라도 조건을 만족하는 요소(조건 함수가 true를 반환)를 찾으면 반복 중단

형식    배열.every(<조건 함수>);  
         배열.some(<조건 함수>);

## 2.7 클래스



- 클래스(class): 객체를 생성하기 위한 템플릿(서식)

### 2.7.1 함수로 객체를 생성하는 방법

- 공장 함수(factory function): 객체를 반환하는 함수
- 새로운 객체가 필요하면 그때마다 함수 호출

---

```
function createMonster(name, hp, att) {  
  return { name, hp, att };  
}  
  
const monster1 = createMonster('슬라임', 25, 10);  
const monster2 = createMonster('슬라임', 26, 9);  
const monster3 = createMonster('슬라임', 25, 11);
```

---

## 2.7 클래스



- 생성자 함수(constructor function): new를 붙여 호출하는 함수
- new를 붙여 호출할 때마다 새로운 객체 생성

---

```
function Monster(name, hp, att) {  
  this.name = name;  
  this.hp = hp;  
  this.att = att;  
}  
const monster1 = new Monster('슬라임', 25, 10);  
const monster2 = new Monster('슬라임', 26, 9);  
const monster3 = new Monster('슬라임', 25, 11);
```

---



## 2.7 클래스



- 프로토타입(prototype): 생성자 함수로 생성한 객체가 공유하는 속성
- 생성자 함수는 prototype 속성 안에 추가해야 메서드를 재사용할 수 있음

```
function Monster(name, hp, att) {  
  this.name = name;  
  this.hp = hp;  
  this.att = att;  
}  
Monster.prototype.attack = function(monster) {  
  monster.hp -= this.att;  
};  
const monster1 = new Monster('슬라임', 25, 10);  
const monster2 = new Monster('슬라임', 26, 9);  
monster1.attack === monster2.attack; // true
```

## 2.7 클래스



### 2.7.2 this 이해하기

- this: 상황에 따라 값이 달라짐(기본으로 window 객체를 가리킴)
  - 객체 메서드로 this를 사용하면 this는 해당 객체를 가리킴
  - 함수의 this는 bind() 메서드를 사용해 값을 바꿀 수 있음
  - 생성자 함수를 호출할 때 new를 붙이면 this는 생성자 함수가 새로 생성하는 객체가 됨

## 2.7 클래스



### 2.7.3 클래스로 객체를 생성하는 방법

- class 예약어로 클래스 선언
- 생성자 함수 이름을 클래스 이름으로 넣음
- 매개변수를 포함한 기존 함수의 코드는 constructor() 메서드 안에 넣음

```
형식  class <클래스 이름> {  
        constructor(매개변수1, 매개변수2, ...) {  
            // 생성자 함수 내용  
        }  
    }
```

## 2.7 클래스



### 2.7.4 클래스 상속하기

- 상속: 부모 클래스에서 공통부분을 가져와 사용하는 것
- 부모 클래스: 복수의 클래스에서 공통되는 부분만 추려 만든 클래스
- 자식 클래스: 부모 클래스에서 공통부분을 가져와 사용하는(상속 받는) 클래스

형식

```
class <자식 클래스> extends <부모 클래스> {  
    constructor(매개변수1, 매개변수2, ...) {  
        super(인수1, 인수2 ...); // 부모 클래스의 생성자 호출  
        this.매개변수 = 값; // 자식 클래스만의 속성  
    }  
    메서드() { // 부모 클래스의 메서드만 호출하면 생략 가능  
        super.메서드(); // 부모 클래스의 메서드 호출  
        // 부모 클래스 메서드 호출 이후의 동작  
    }  
    메서드(매개변수1, 매개변수2, ...) {  
        // 자식 클래스만의 동작  
    }  
}
```

# 3장 심화 문법 배우기

---

3.1 비동기와 타이머

3.2 스코프와 클로저

3.3 호출 스택과 이벤트 루프

3.4 프로미스와 `async/await`

## 3.1 비동기와 타이머



- 동기(synchronous): 앞선 작업이 완전히 끝난 후에 다음 작업이 실행되는 것
- 비동기(asynchronous): 동기가 아님, 즉 앞선 작업이 끝나지 않았는데도 다음 작업이 실행되는 것



## 3.1 비동기와 타이머

### 3.1.1 setTimeout()

- setTimeout() 함수: 지정한 시간 뒤에 코드 실행

형식    `setTimeout(함수, 밀리초);`

### 3.1.2 setInterval()

- setInterval() 함수: 지정한 시간마다 주기적으로 지정한 함수 실행

형식    `setInterval(함수, 밀리초);`

## 3.1 비동기와 타이머



### 3.1.3 clearTimeout()과 clearInterval()

- clearTimeout() 함수: setTimeout() 함수의 실행을 취소
- clearInterval() 함수: setInterval() 함수의 실행을 취소

형식

```
const 아이디 = setTimeout(함수, 밀리초);
```

```
clearTimeout(아이디);
```

```
const 아이디 = setInterval(함수, 밀리초);
```

```
clearInterval(아이디);
```



## 3.2 스코프와 클로저



### 3.2.1 블록 스코프와 함수 스코프

- 스코프(scope): 범위
- 함수 스코프: 함수를 경계로 접근 가능 여부가 달라지는 것
  - 함수만 신경 씀
  - 함수가 끝날 때 함수 내부의 변수도 같이 사라짐
  - var
- 블록 스코프: 블록(중괄호, {})을 경계로 접근 가능 여부가 달라지는 것
  - 블록을 신경 씀
  - 블록이 끝날 때 내부의 변수도 같이 사라짐
  - let, const

## 3.2 스코프와 클로저



### 3.2.2 클로저와 정적 스코프

- 클로저(closure): 외부 값에 접근하는 함수
- 정적 스코프(lexical scope): 함수가 선언된 위치에 따라 접근할 수 있는 값이 달라지는 것
- 동적 스코프(dynamic scope): 호출된 위치에 따라 접근할 수 있는 값이 달라지는 것

## 3.2 스코프와 클로저



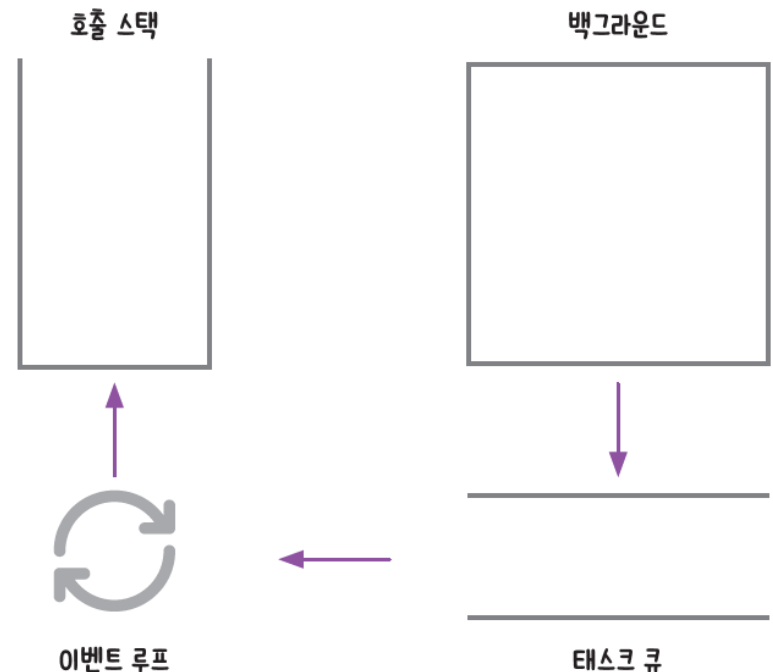
### 3.2.3 let과 var를 사용한 결과가 다른 이유

- var를 쓸 때: 클로저가 실행될 때 이미 i 변수의 값이 달라짐
- let을 쓸 때: 블록별로 i 변수의 값이 고정
- 스코프를 알아 두어야 할 때
  - setTimeout() 같은 비동기 함수와 반복문, var가 만났을 때
  - switch 문을 사용할 때

## 3.3 호출 스택과 이벤트 루프



- 호출 스택(call stack): 동기 담당
- 이벤트 루프(event loop): 비동기 담당
- 백그라운드(background): 타이머를 처리하고 이벤트 리스너를 저장하는 공간
- 태스크 큐(task queue): 실행될 콜백 함수들이 대기하고 있는 공간

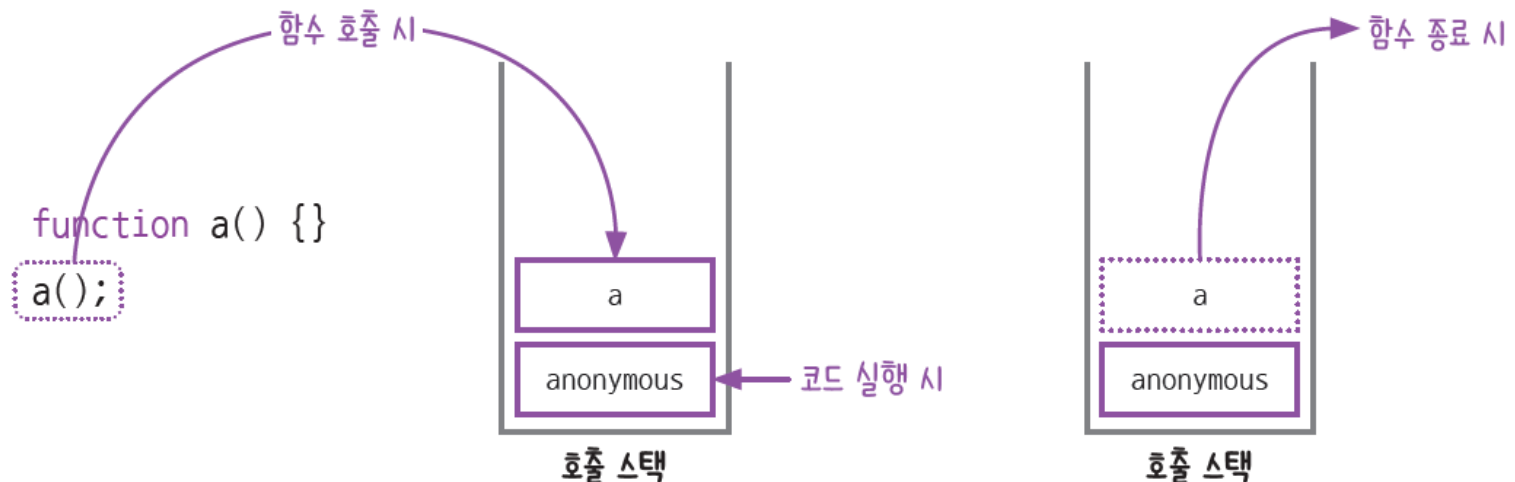


## 3.3 호출 스택과 이벤트 루프



### 3.3.1 호출 스택

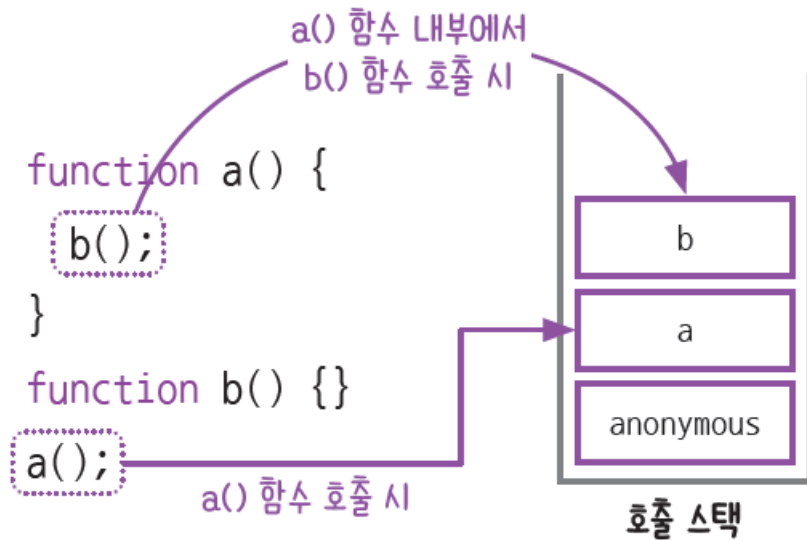
- 스택(stack): 바닥이 막히고 천장은 뚫린 하나의 통처럼, 물건을 넣을 때는 한 개씩 쌓아 넣고 물건을 뺄 때는 마지막에 넣은 것부터 먼저 빠지는 구조
- a() 함수의 호출과 종료 시 호출 스택 구조



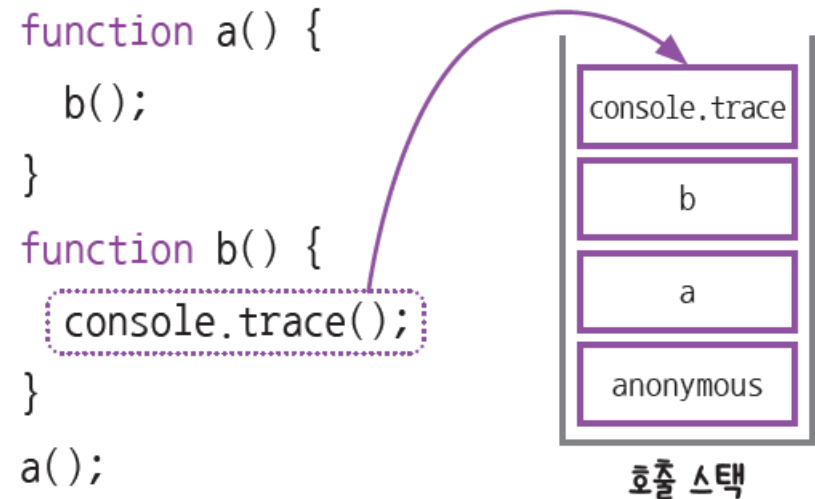
## 3.3 호출 스택과 이벤트 루프



- a() 함수 내부에서 b() 함수 호출 시



- b() 함수 내부에서 console.trace() 호출 시



## 3.3 호출 스택과 이벤트 루프



### 3.3.2 이벤트 루프

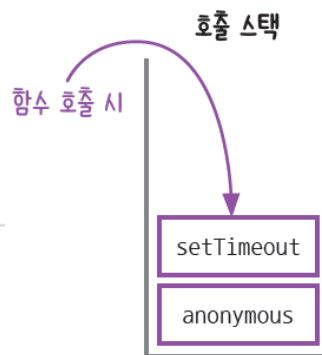
- 이벤트 루프: 호출 스택이 비어 있을 때 태스크 큐에서 호출 스택으로 함수를 이동시키는 역할을 함
- setTimeout() 함수 호출 시

```
const timerId = setTimeout(() => {  
  console.log('0초 뒤에 실행됩니다.');
```

---

```
}, 0);  
console.log('내가 먼저');
```

---



이벤트 루프

백그라운드

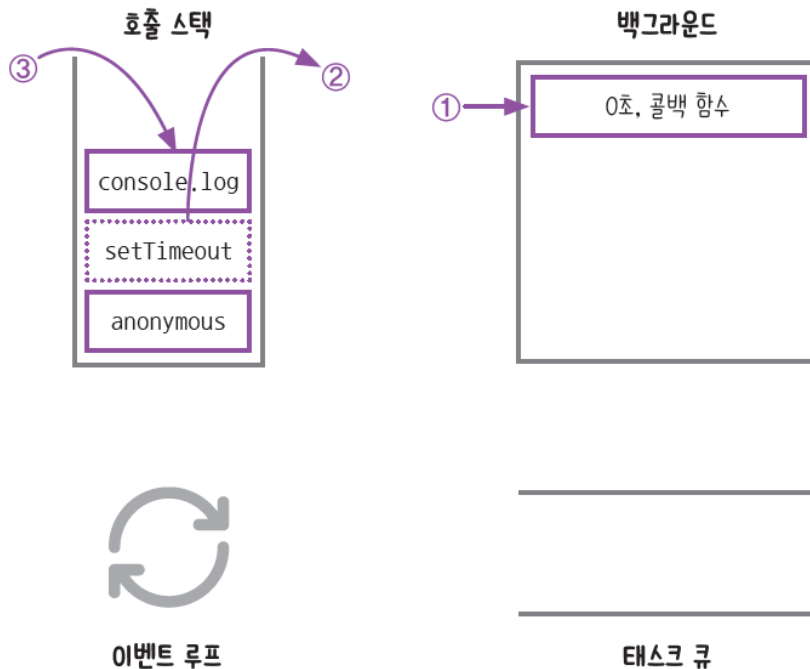


태스크 큐

## 3.3 호출 스택과 이벤트 루프



- setTimeout() 함수 종료 시



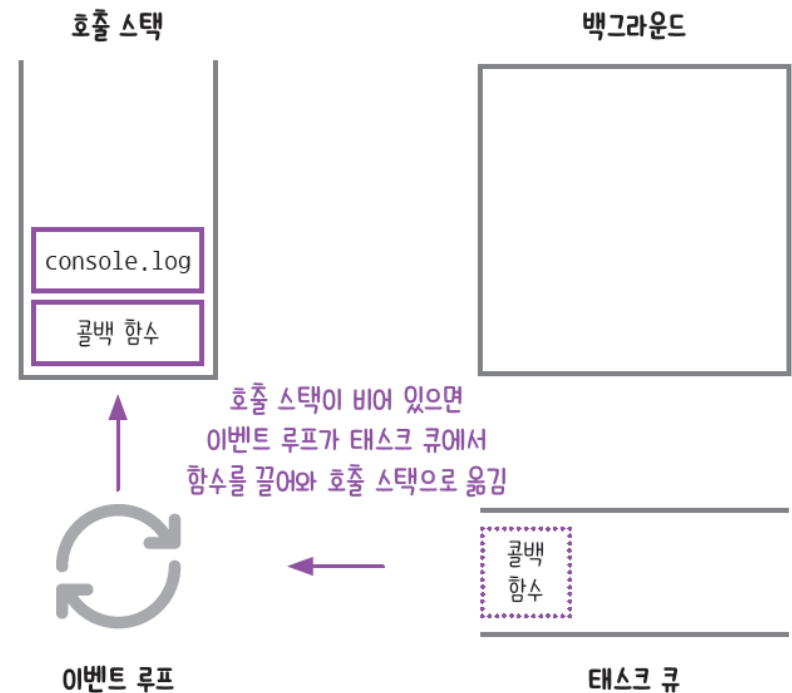
- ① `setTimeout()` 함수는 콜백 함수를 배경라운드로 보내고 종료
- ② `setTimeout()` 함수는 종료되면서 호출 스택을 빠져나감
- ③ 다음 줄의 `console.log('내가 먼저');`가 실행됨



## 3.3 호출 스택과 이벤트 루프



- 이벤트 루프는 호출 스택이 비어 있을 때 움직임
- 호출 스택이 비어 있으면 이벤트 루프는 태스크 큐에서 함수를 하나 끌어와 호출 스택으로 보냄
- 큐는 먼저 들어온 함수가 먼저 빠져나감
- 콜백 함수가 호출 스택으로 보내지면 실행

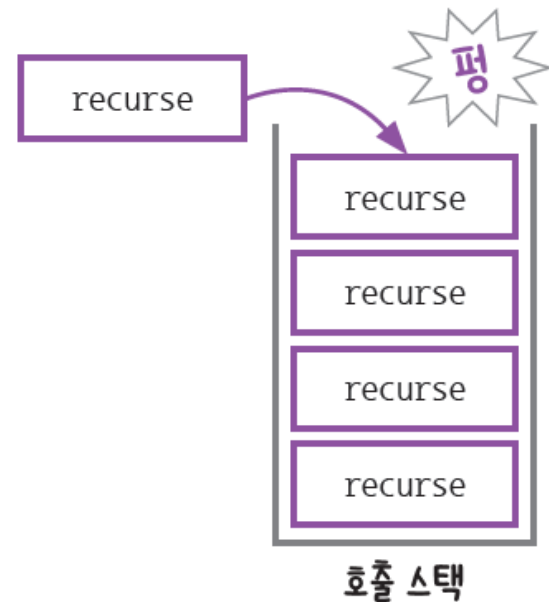


## 3.3 호출 스택과 이벤트 루프



### 3.3.3 재귀 함수

- 재귀 함수(recursive function): 어떤 함수가 내부에서 자기 자신을 다시 호출하는 함수
- 재귀 함수 호출 시 호출 스택의 최대 크기 초과 문제 발생



## 3.4 프로미스와 async/await



### 3.4.1 프로미스

- 프로미스: Promise라는 클래스를 사용하는 문법
- new를 붙여 Promise 클래스를 호출하면 프로미스 객체를 생성, 인수로 콜백 함수를 넣음
- 콜백 함수의 매개변수는 resolve()와 reject() 함수
- 콜백 함수 내부에서 resolve()나 reject() 함수 중 하나를 호출해야 함
- resolve() 함수 호출 프로미스 성공, reject() 함수 호출 프로미스 실패, 둘 다 호출하면 먼저 호출한 함수만 유효

형식 `const <프로미스 객체> = new Promise((resolve, reject) => {  
 resolve(); // 프로미스 성공  
 // 또는  
 reject(); // 프로미스 실패  
});`

## 3.4 프로미스와 async/await



### 3.4.1 프로미스

- 프로미스 객체에는 then() 메서드나 catch() 메서드를 붙일 수 있음
- 두 메서드도 인수로 콜백 함수를 넣음
- then()의 콜백 함수는 resolve() 함수를 호출할 때 실행
- catch()의 콜백 함수는 reject() 함수를 호출할 때 실행
- resolve()의 인수로 전달한 값은 then() 콜백 함수의 매개변수로 전달
- reject()의 인수로 전달한 값은 catch() 콜백 함수의 매개변수로 전달

형식

```
<프로미스 객체>.then(<콜백 함수>);  
// 또는  
<프로미스 객체>.catch(<콜백 함수>);
```

## 3.4 프로미스와 async/await



- resolve() 호출 시 then() 실행, reject() 호출 시 catch() 실행

```
const p1 = new Promise((resolve, reject) => {  
  resolve('success');  
});  
p1.then((data) => console.log(data));
```

Diagram illustrating the execution flow for a resolved promise:

- The `resolve('success')` call in the promise constructor is highlighted with a dotted box.
- An arrow points from this box to the `(data)` parameter in the `then` method.
- Another arrow points from the `(data)` parameter to the `data` argument in the `console.log(data)` call.
- The value `'success'` is shown below the `console.log` call, indicating the data passed to the log function.

```
const p2 = new Promise((resolve, reject) => {  
  reject('error');  
});  
p2.catch((error) => console.log(error));
```

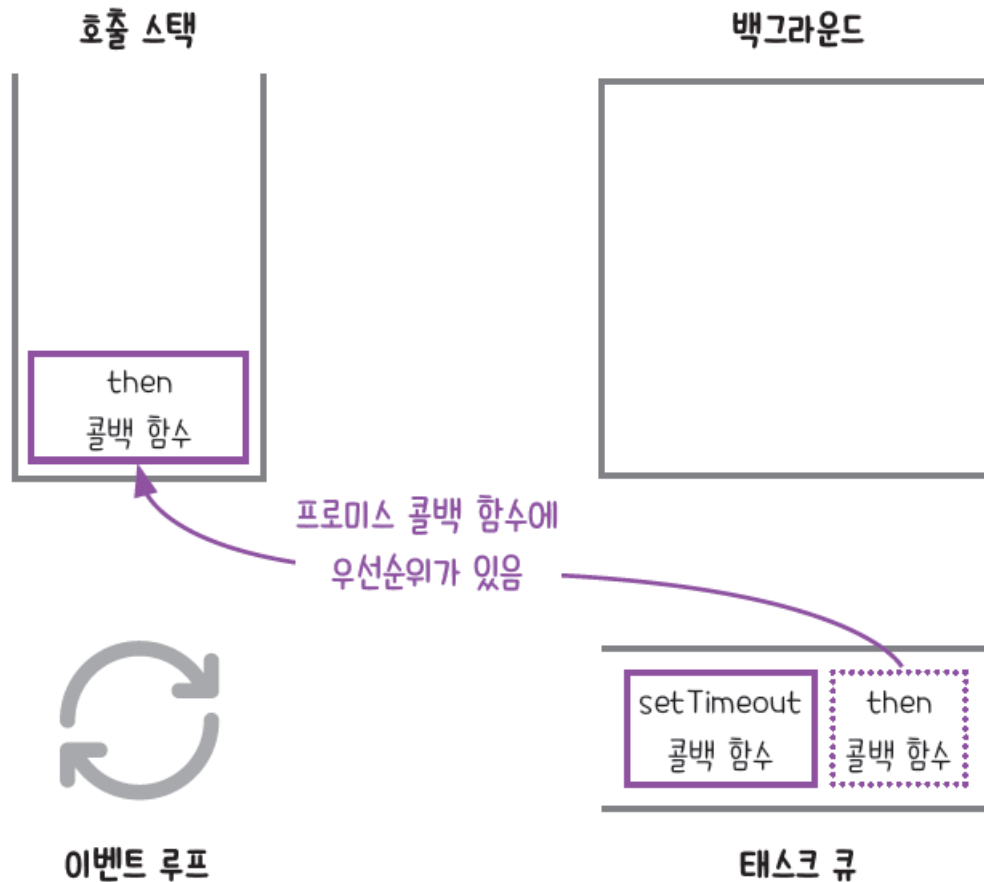
Diagram illustrating the execution flow for a rejected promise:

- The `reject('error')` call in the promise constructor is highlighted with a dotted box.
- An arrow points from this box to the `(error)` parameter in the `catch` method.
- Another arrow points from the `(error)` parameter to the `error` argument in the `console.log(error)` call.
- The value `'error'` is shown below the `console.log` call, indicating the data passed to the log function.

## 3.4 프로미스와 async/await



- 프로미스 사용 시 호출 스택과 이벤트 루프



## 3.4 프로미스와 async/await



### 3.4.2 async/await

- await: 프로미스인 비동기 코드를 순서대로 실행하게 함

### 3.4.3 try-catch 문으로 에러 처리하기

- try-catch 문으로 감싸 에러 처리
- catch 문의 error는 사용하지 않는 경우 생략할 수 있음
- finally 문을 추가할 수 있음

# 4장 HTML과 DOM 조작하기

---

4.1 HTML 파일 생성하기

4.2 DOM 사용하기

4.3 이벤트와 이벤트 리스너

4.4 다양한 DOM 속성

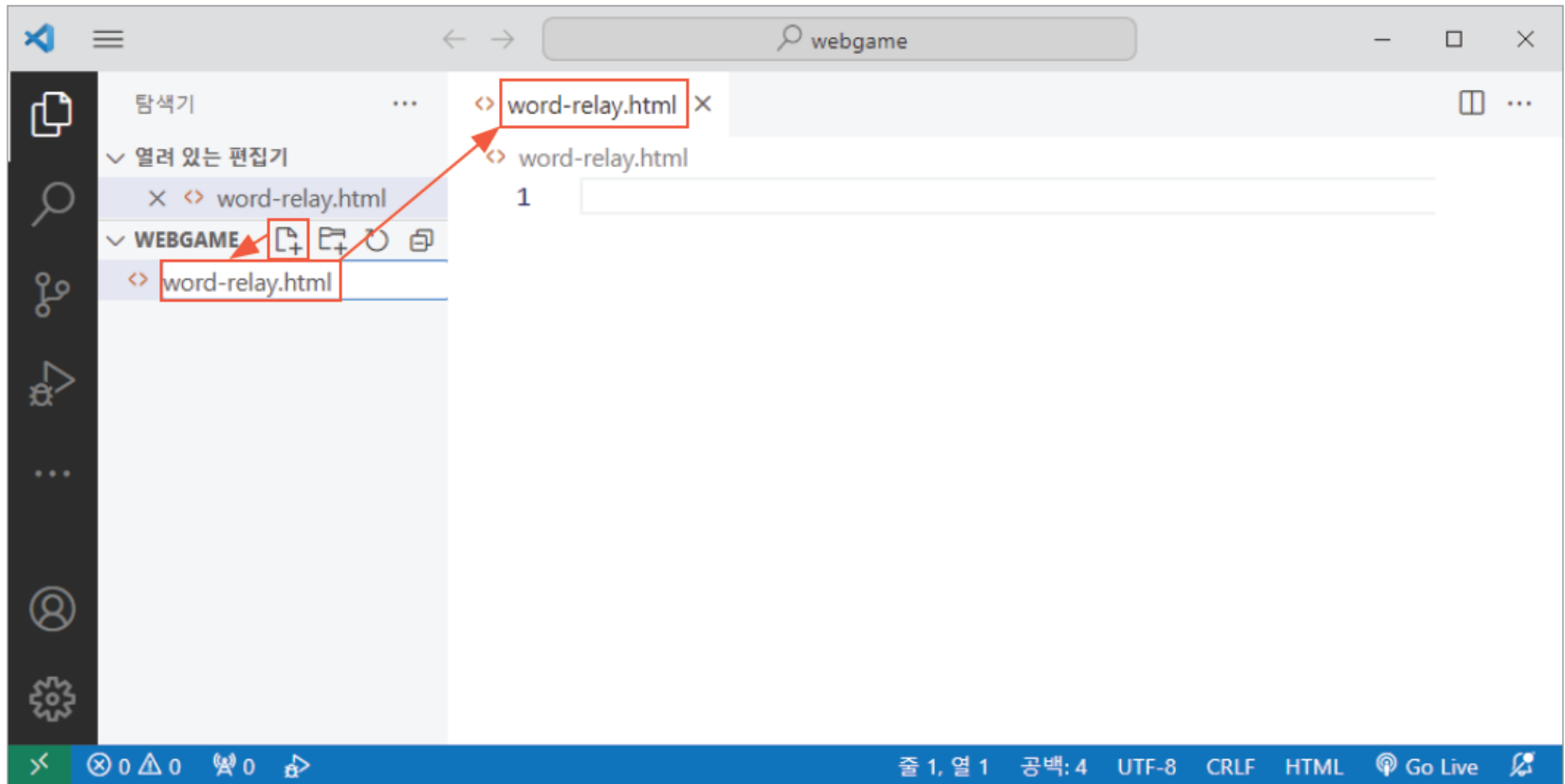
4.5 window 객체



# 4.1 HTML 파일 생성하기



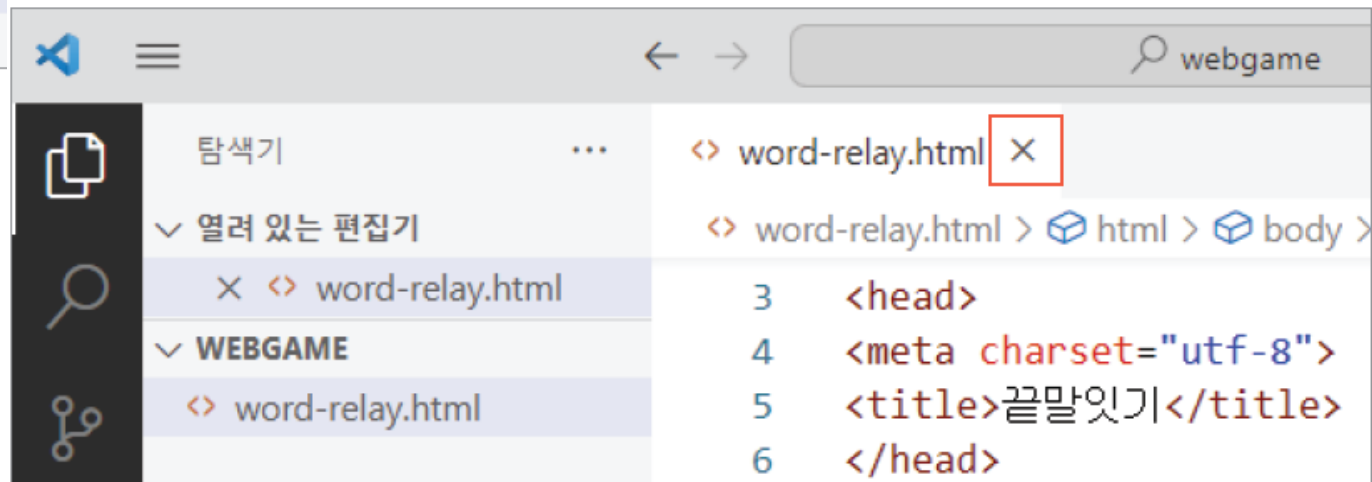
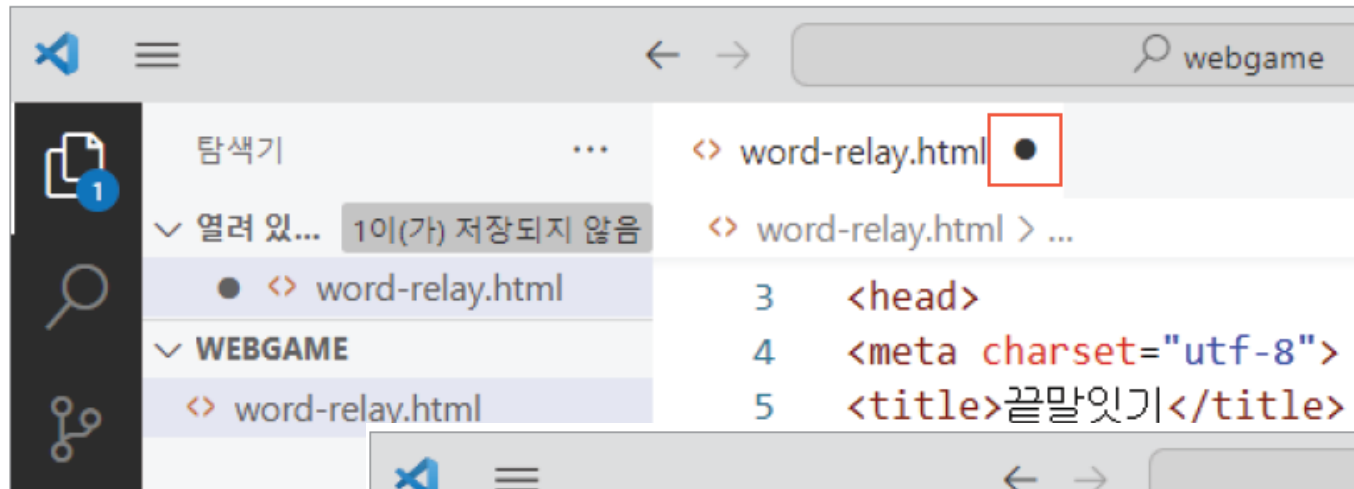
- VSCode에서 새 파일 만들기



## 4.1 HTML 파일 생성하기



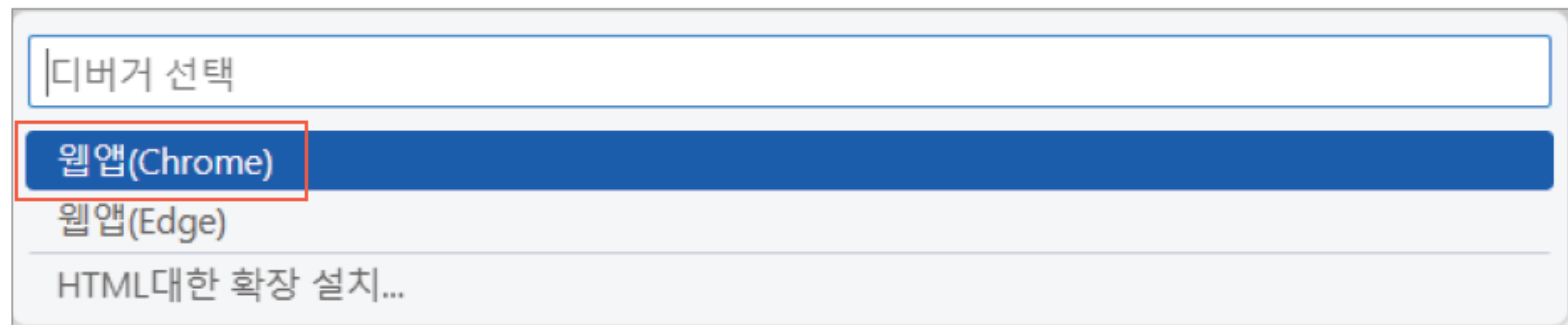
- HTML 파일 작성 후 저장하기



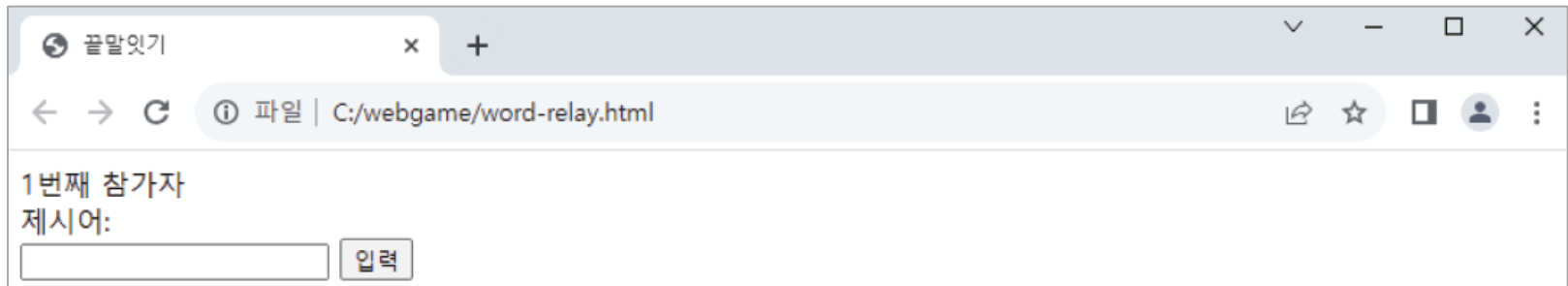
## 4.1 HTML 파일 생성하기



- 파일을 저장한 후 F5 를 누르면 디버거를 선택할 수 있음
- 웹앱(Chrome) 선택



- 크롬에서 HTML 파일이 실행됨



## 4.2 DOM 사용하기

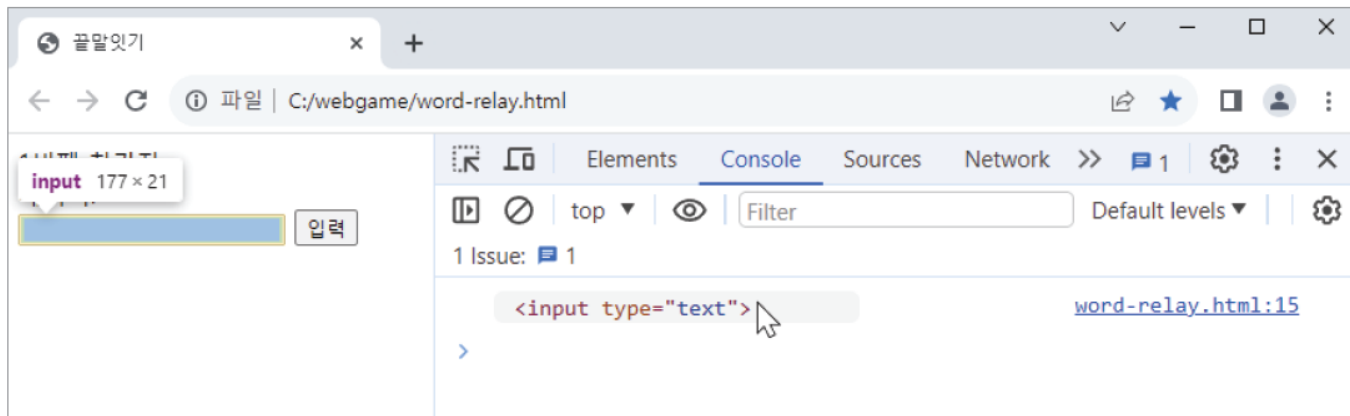


### 4.2.1 선택자 사용하기

- 선택자(selector): HTML 태그를 가져오게(선택하게) 도와주는 문자열
- document 객체: 브라우저에 열려 있는 HTML 문서를 나타냄

형식 `document.querySelector('선택자')`

- 선택자에 input 태그를 넣은 결과

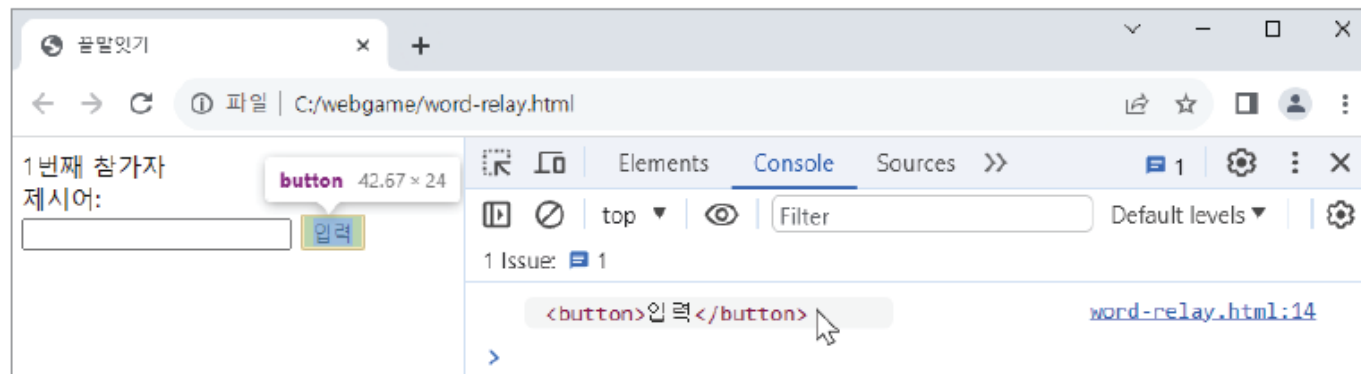


## 4.2 DOM 사용하기



- DOM(Document Object Model): 웹 브라우저가 웹 페이지의 HTML을 자바스크립트 객체로 구성해 둔 것
- DOM은 document 객체를 통해 접근 및 조작할 수 있음
- 선택자에 button 태그를 넣으면 button 태그에 접근할 수 있음

```
<script>  
const $button = document.querySelector('button');  
console.log($button);  
</script>
```



## 4.2 DOM 사용하기



- 여러 태그를 한꺼번에 선택할 때는 `document.querySelectorAll()` 메서드 사용

```
<button>입력</button>
```

```
<button>버튼2</button>
```

```
<button>버튼3</button>
```

```
<script>
```

```
const $$buttons = document.querySelectorAll('button');
```

```
console.log($$buttons);
```

```
</script>
```

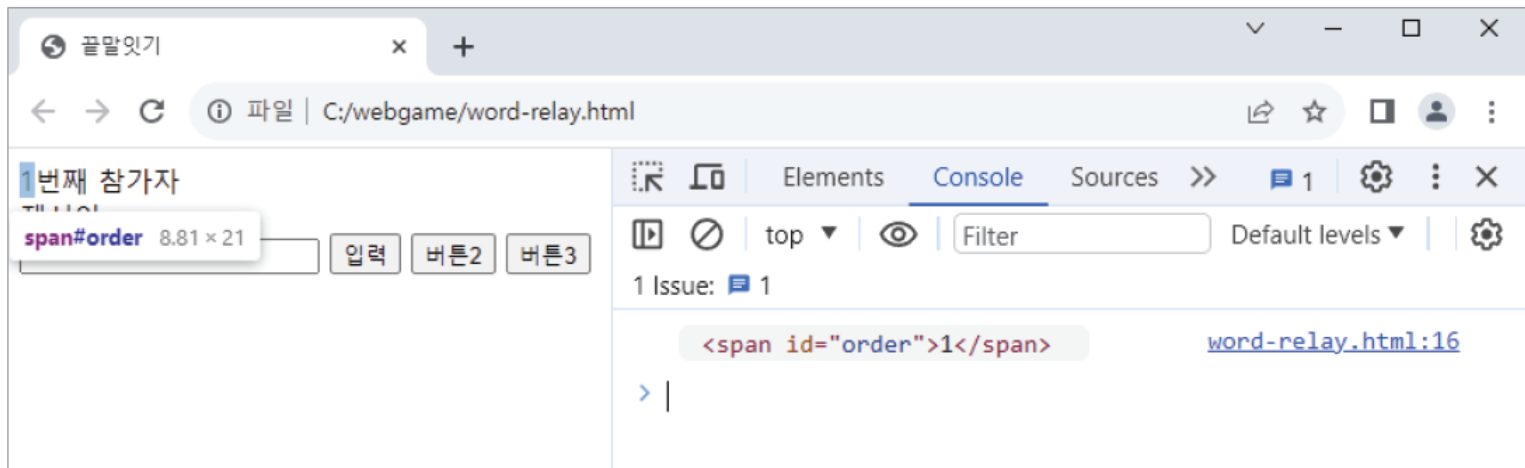
## 4.2 DOM 사용하기



- id 속성으로 특정 태그 선택하기
  - 여러 개의 태그 중에서 특정 태그만 선택할 때 사용

형식 `document.querySelector('#<id 속성 값>')`

- id 속성의 값을 선택자로 사용한 결과



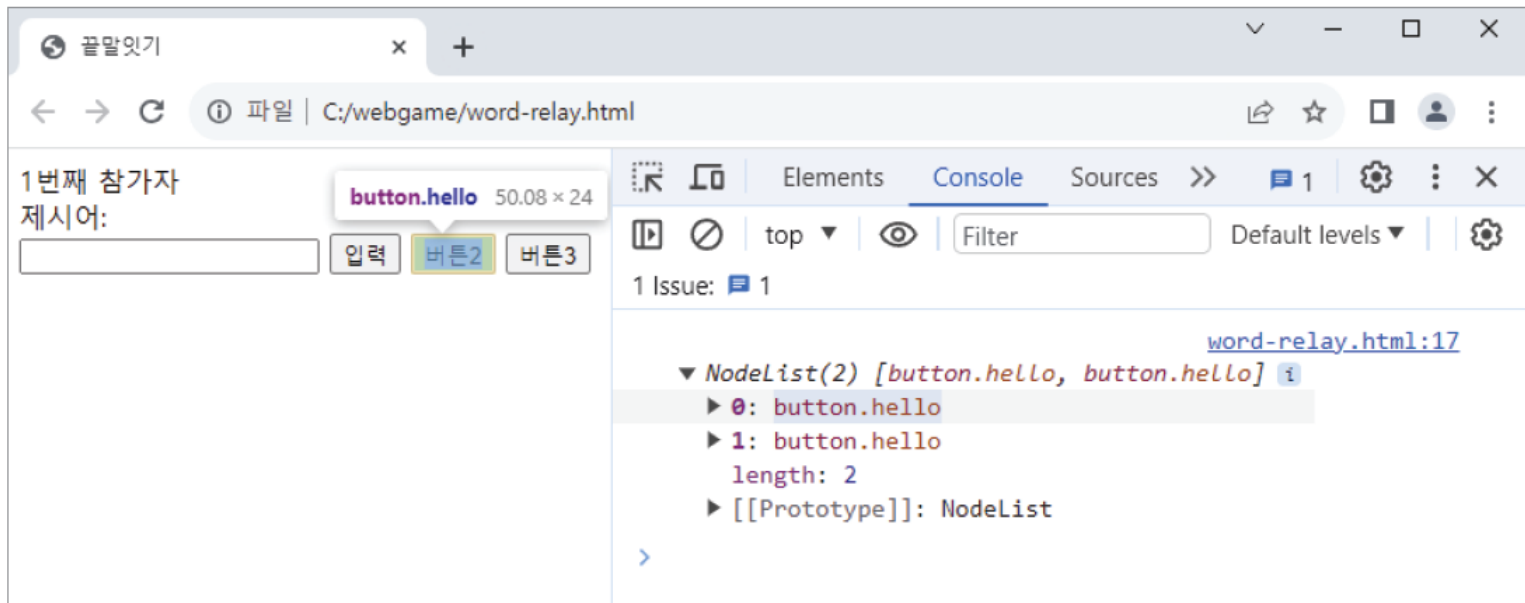
## 4.2 DOM 사용하기



- class 속성으로 여러 태그 선택하기
  - 여러 태그 중에서 골라 선택할 때 사용

형식 `document.querySelectorAll('.<class 속성 값>')`

- class 속성 값을 선택자로 사용한 결과





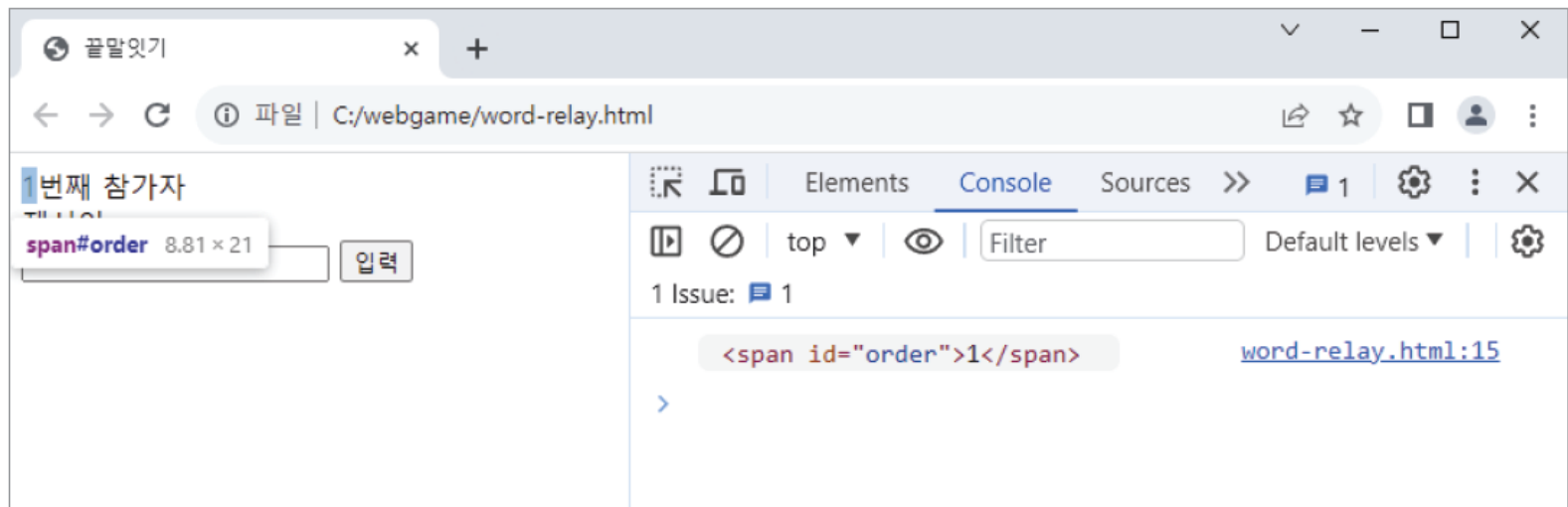
## 4.2 DOM 사용하기



- 태그 안의 태그 선택하기
  - 다른 태그 안에 들어 있는 태그를 선택할 때 사용

형식 `document.querySelector('선택자 내부선택자 내부선택자 ...')`

- 선택자가 여러 개일 때



## 4.2 DOM 사용하기

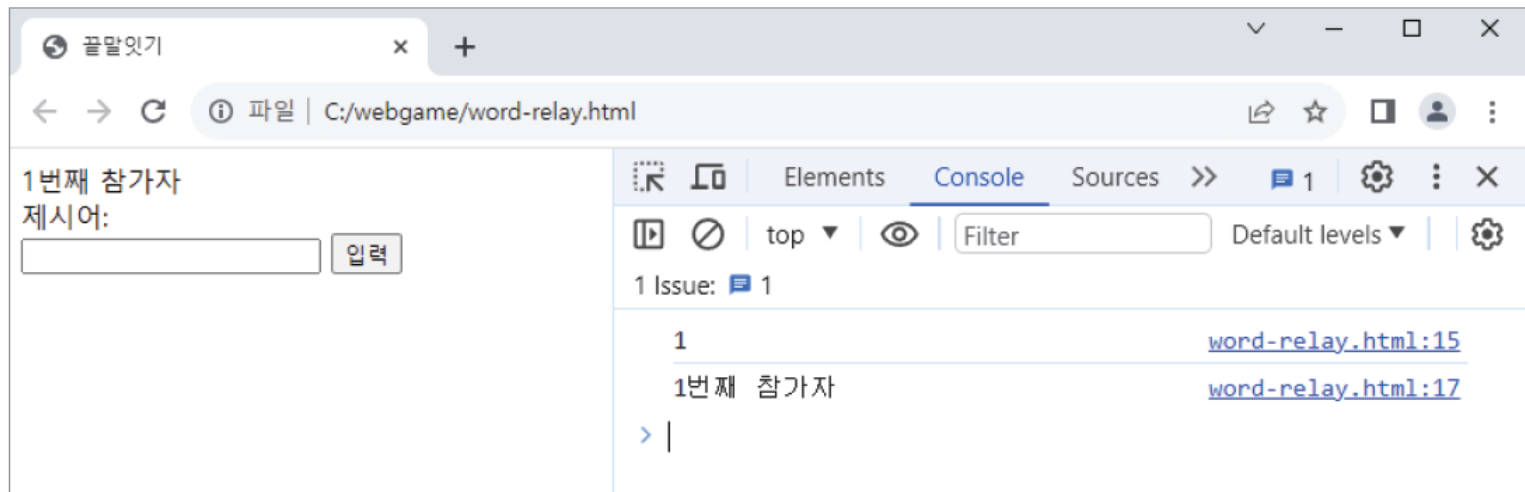


### 4.2.2 태그의 값에 접근하기

- 텍스트와 태그 가져오기
  - 태그 내부의 문자열을 가져올 때

**형식** 태그.textContent // 태그 내부의 문자열을 가져옴

- textContent 사용 결과



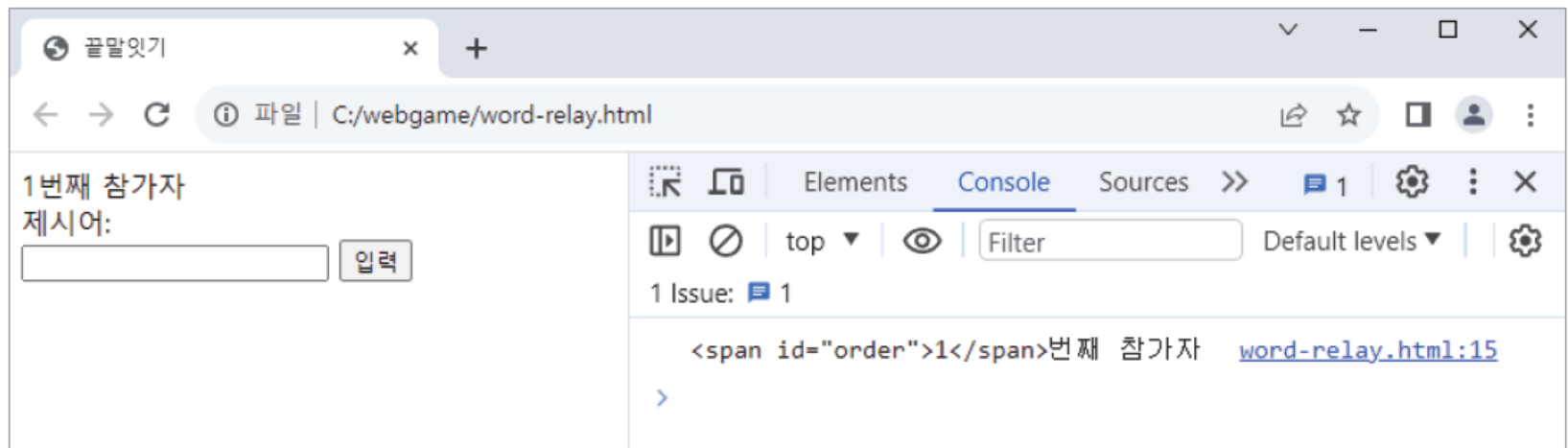
## 4.2 DOM 사용하기



- 내부의 태그까지 전부 가져올 때

**형식**    태그.innerHTML // 태그 내부의 HTML 태그를 포함한 문자열을 가져옴

- innerHTML 사용 결과



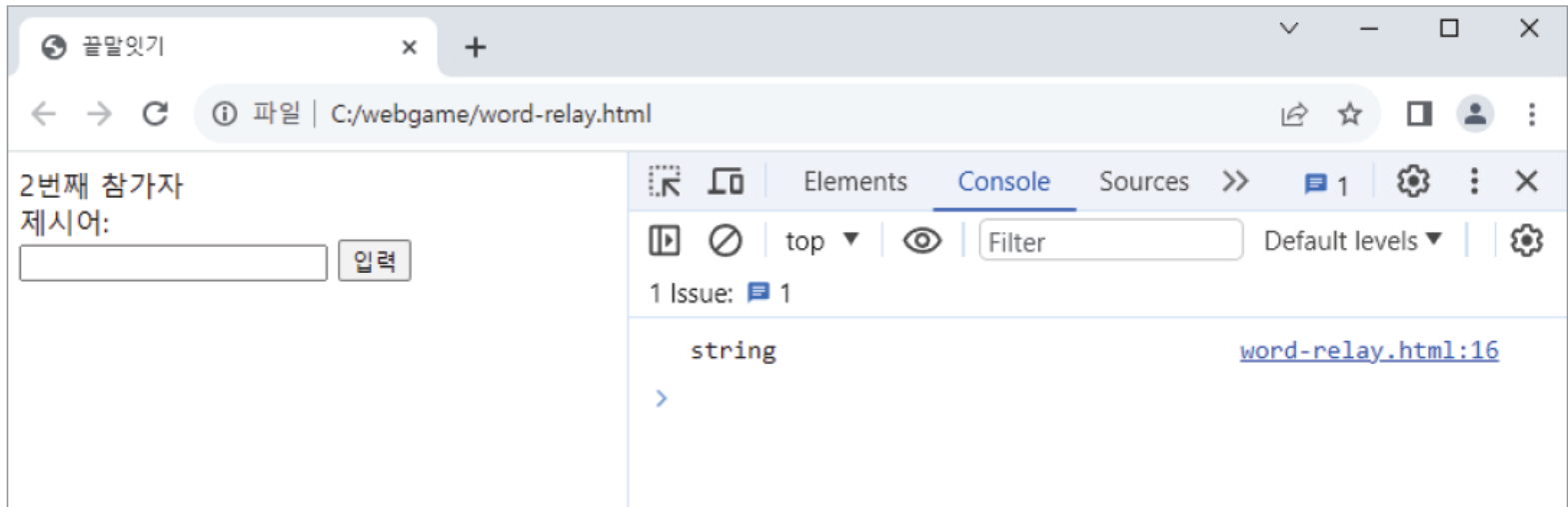
## 4.2 DOM 사용하기



- 텍스트와 태그 변경하기
  - 태그 내부 텍스트의 값을 수정할 때

**형식** 태그.textContent = 값; // 태그 내부의 문자열을 해당 값으로 설정함

- textContent로 텍스트 수정한 결과



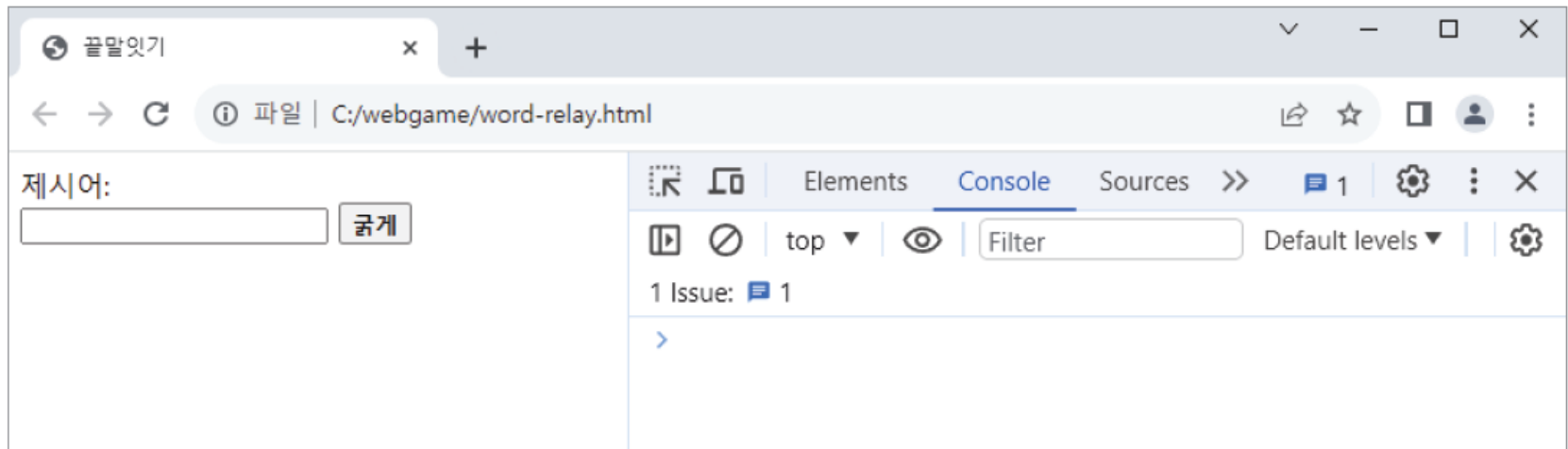
## 4.2 DOM 사용하기



- 태그 자체를 수정할 때

**형식**    `태그.innerHTML = 값; // 태그 내부의 태그를 해당 값으로 설정함`

- innerHTML로 태그를 수정한 결과



## 4.2 DOM 사용하기



- 입력 태그의 값 가져와 변경하기
  - 태그 내부의 값을 선택할 때는 `textContent`를 사용하지만, 입력 태그만 `value`를 사용
  - 대표적인 입력 태그: `input`, `select`, `textarea`

형식 `<입력 태그>.value` // 입력창의 값을 가져옴  
`<입력 태그>.value = 값;` // 입력창에 값을 넣음

- 입력 태그를 선택할 때 `focus()` 메서드 사용

형식 `<입력 태그>.focus()` // 입력창을 하이라이트함

## 4.3 이벤트와 이벤트 리스너



- 이벤트(event): 사용자가 태그와 상호작용할 때 발생
- 이벤트 리스너(event listener): 자바스크립트가 HTML에서 발생하는 이벤트를 감지할 수 있게 함

### 4.3.1 이벤트 리스너 추가하기

- `addEventListener()`: 태그에 이벤트 리스너를 다는 메서드

형식

`태그.addEventListener('〈이벤트 이름〉', 〈이벤트 리스너〉)`

## 4.3 이벤트와 이벤트 리스너



- script 태그 내부에 이벤트가 발생하면 이를 감지할 onClickButton() 함수를 만들고 버튼 태그의 클릭 이벤트에 연결
- 태그는 document.querySelector('button')으로 선택
- 선택한 태그에 addEventListener() 메서드를 사용해 이벤트 리스너를 담
- 이벤트 이름은 click, 버튼을 클릭하면 onClickButton() 함수 실행

```
<script>
```

```
const onClickButton = () => {  
  console.log('버튼 클릭');  
};
```

```
const $button = document.querySelector('button');  
$button.addEventListener('click', onClickButton);
```

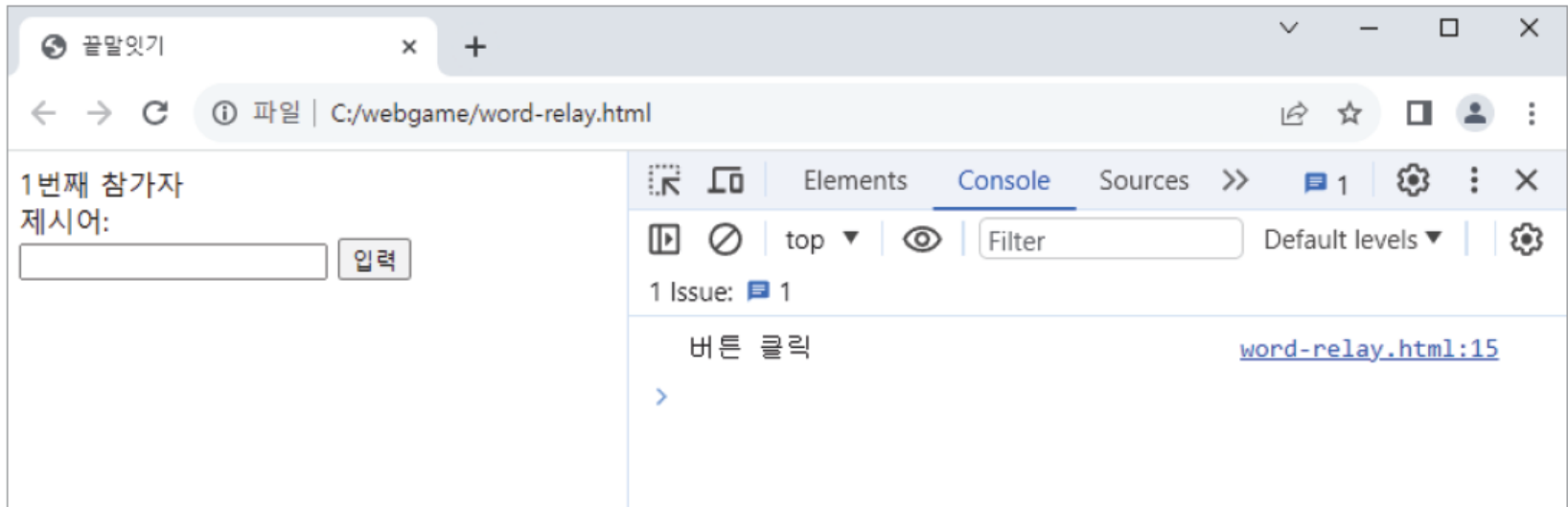
```
</script>
```



## 4.3 이벤트와 이벤트 리스너



### ■ 버튼 클릭 이벤트 실행결과

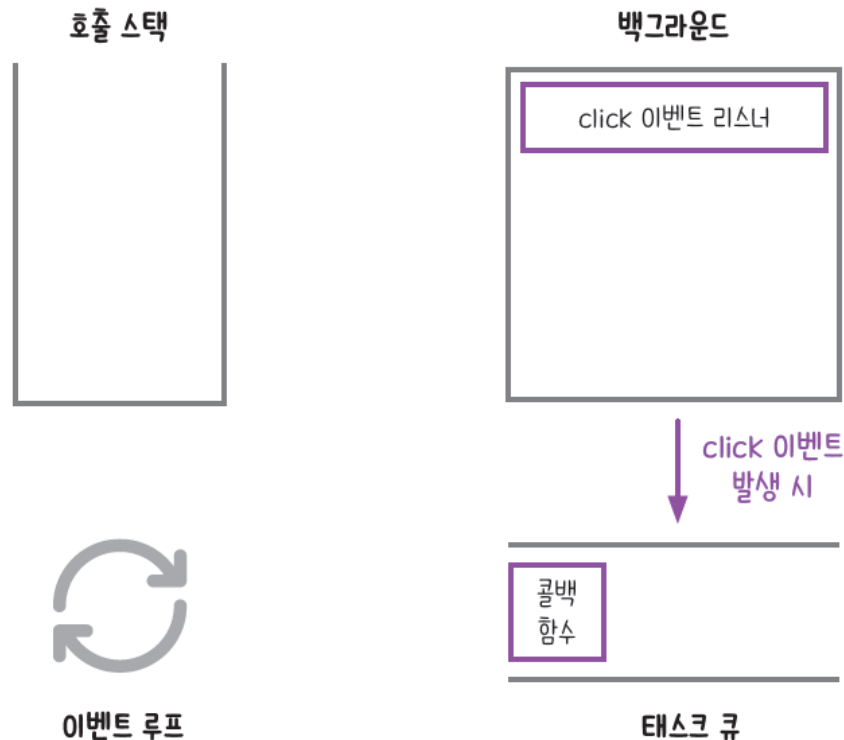


The screenshot shows a web browser window with a single tab titled '끝말잇기'. The address bar displays the file path 'C:/webgame/word-relay.html'. The page content includes the text '1번째 참가자 제시어:' followed by an empty text input field and a button labeled '입력'. The browser's developer tools are open, with the 'Console' tab selected. The console shows a single log entry: '버튼 클릭' (Button Click) at 'word-relay.html:15'. The console interface includes tabs for 'Elements', 'Console', and 'Sources', along with filters and a 'Filter' input field.

## 4.3 이벤트와 이벤트 리스너



- 이벤트 리스너는 대표적인 비동기 함수
- 이벤트 리스너 추가 시 호출 스택과 이벤트 루프



- `addEventListener()` 메서드를 실행하는 순간 백그라운드에서 이벤트 리스너 등록
- 이벤트 리스너가 존재하는 동안에는 전체 코드는 종료되지 않음
- 준비하고 있다가 click 이벤트가 발생하는 순간 이벤트 리스너의 콜백 함수를 태스크 큐로 보냄

## 4.3 이벤트와 이벤트 리스너



### 4.3.2 이벤트 리스너 제거하기

- 이벤트 리스너는 `removeEventListener()` 메서드로 제거
- 연결한 함수와 제거하는 함수가 같아야 함

형식

```
function 함수() {}  
태그.addEventListener('이벤트', 함수)  
태그.removeEventListener('이벤트', 함수)
```

## 4.3 이벤트와 이벤트 리스너



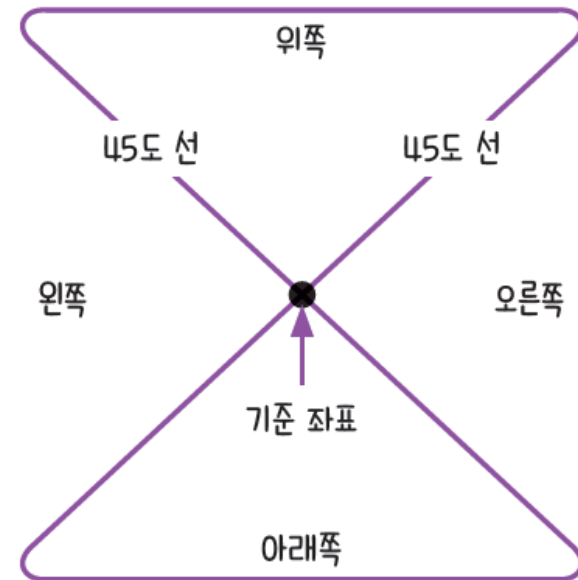
### 4.3.3 키보드와 마우스 이벤트

- 키보드 이벤트
  - 키보드의 키를 눌렀다(keydown) 뗐을 때(keyup) 발생하는 이벤트
  - 키보드를 누르고 있으면 keydown 이벤트가 계속 호출되므로 키보드에서 손을 뗄 때 발생하는 keyup 이벤트를 기준으로 코드를 작성
- 마우스 이벤트
  - 마우스를 움직이면 mousemove 이벤트가 발생하고, 오른쪽이나 왼쪽 버튼을 클릭하면 mousedown, 클릭했다가 뗄 때는 mouseup 이벤트가 발생
  - 이벤트 리스너의 매개변수인 event 안에는 화면 좌표 등의 정보가 담겨 있음

## 4.3 이벤트와 이벤트 리스너



- 마우스 이벤트의 속성에서 x, y 좌표를 얻어 마우스의 위치 변화를 알아낼 수 있음
- 마우스의 이동 방향 판단 기준
  - ✓ 기준 좌표를, 마우스를 클릭한 시작점으로 설정하고, 마우스 버튼에서 손을 떼는 지점을 끝점으로 함
  - ✓ 시작점과 끝점이 이루는 각도로 방향 판단
  - ✓ 각도가  $\pm 45$ 도보다 작으면 각각 왼쪽과 오른쪽이 되고,  $\pm 45$ 도보다 크면 각각 위쪽과 아래쪽이 됨

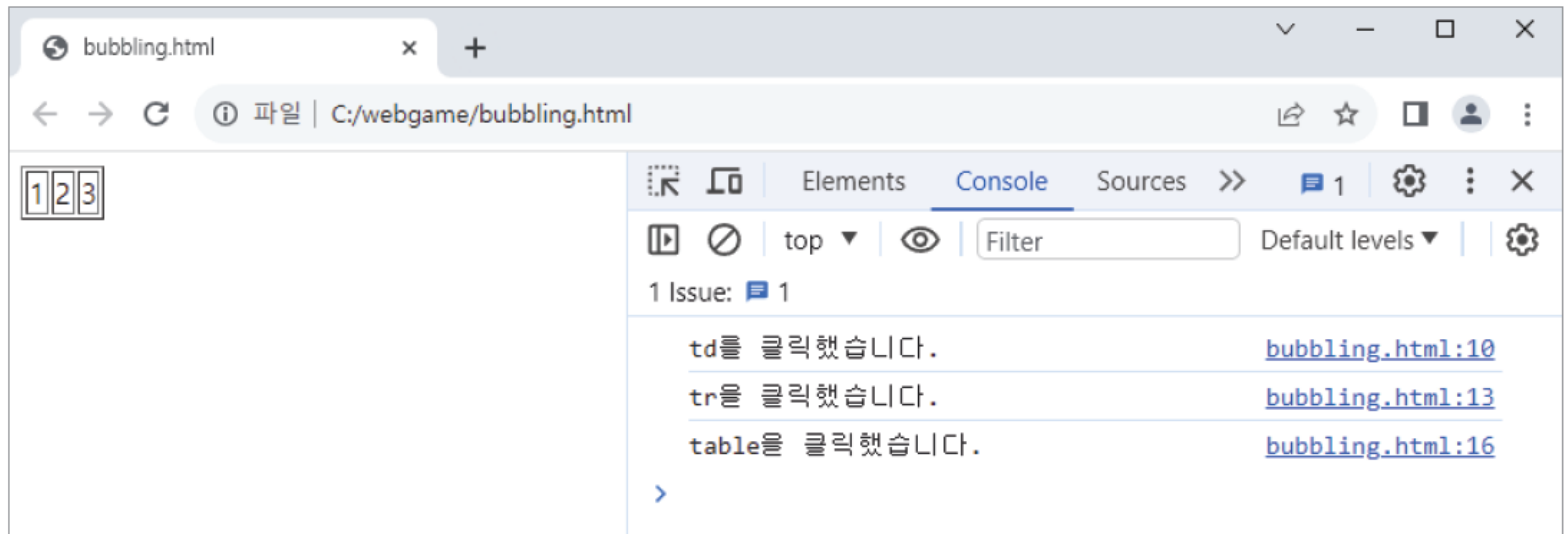


## 4.3 이벤트와 이벤트 리스너



### 4.3.4 이벤트 버블링과 캡처링

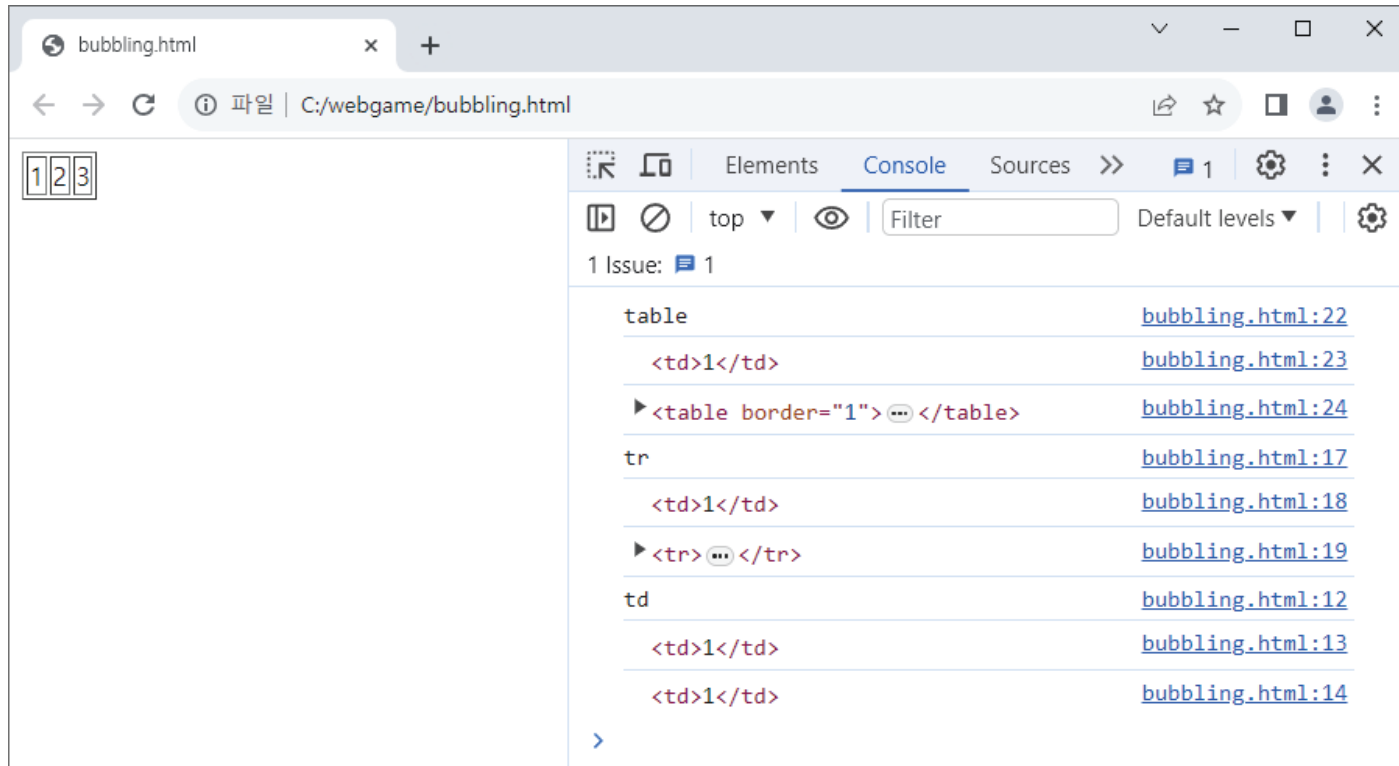
- 이벤트 버블링(event bubbling)
  - 이벤트가 발생할 때 부모 태그에도 동일한 이벤트가 발생하는 현상
  - 이벤트 버블링 현상 예



## 4.3 이벤트와 이벤트 리스너



- 이벤트 캡처링(event capturing)
  - 이벤트가 자식 태그로 전파되어 내려가는 현상
  - 이벤트 캡처링 예



## 4.4 다양한 DOM 속성



### 4.4.1 태그 속성 다루기

- DOM을 통해 자바스크립트로 태그 속성을 다룰 수 있음

형식

태그.속성 // 조회 시 사용

태그.속성 = 값; // 수정 시 사용

- class 속성은 자바스크립트의 class와 헷갈리지 않게 className 사용

형식

태그.className = '클래스1 클래스2 ...';



## 4.4 다양한 DOM 속성



- classList 객체
  - 기존 클래스에 새로운 클래스를 추가하거나 삭제하려면 태그.classList 사용
  - 태그에 붙은 클래스를 조작하는 여러 메서드를 제공
  - contains() 메서드: 태그에 해당 클래스가 존재하는지 확인할 때

형식    태그.classList.contains('클래스')

- add(), replace(), remove() 메서드: 클래스 추가/수정/제거

형식    태그.classList.add('클래스1', '클래스2', ...) // 추가  
          태그.classList.replace('<기존 클래스>', '<수정 클래스>') // 수정  
          태그.classList.remove('클래스1', '클래스2', ...) // 제거

## 4.4 다양한 DOM 속성



- style 속성
  - 자바스크립트에서 태그의 CSS를 변경할 수 있음
  - style 속성을 사용해 태그에 CSS를 적용하는 방식인 인라인 스타일(inline style) 사용

형식    태그.style.<CSS 속성> = '값';

## 4.4 다양한 DOM 속성



### 4.4.2 부모와 자식 태그 찾기

- 현재 태그의 부모 태그를 찾고 싶을 때: parentNode 속성 사용
- 자식 태그를 찾고 싶을 때: children 속성 사용

<script>

```
console.log(document.querySelector('td').parentNode); // tr 태그  
console.log(document.querySelector('tr').children); // [td, td, td]
```

</script>

## 4.4 다양한 DOM 속성



### 4.4.3 새로운 태그 만들기

- document.createElement(): 태그를 만드는 메서드
- document.createTextNode(): 텍스트를 만드는 메서드
- append(), appendChild(): 다른 태그 내부에 만든 태그를 추가하는 메서드

형식

```
<부모 태그>.appendChild(<자식 태그>)
```

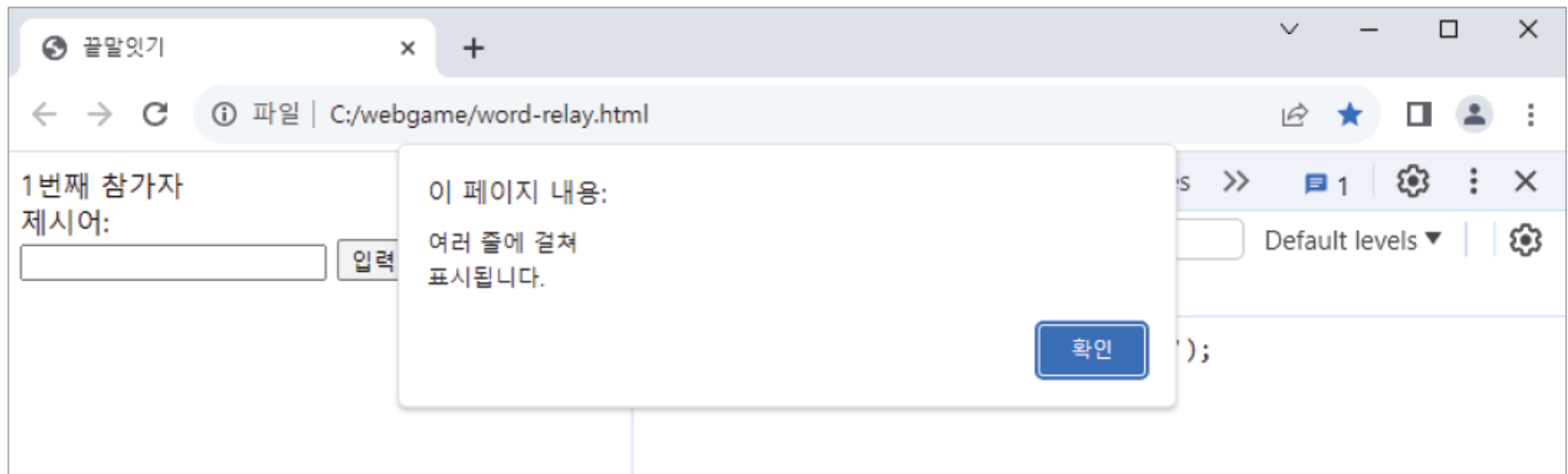
```
<부모 태그>.append(<자식 태그1>, <자식 태그2>, ...)
```

## 4.5 window 객체



### 4.5.1 대화상자 사용하기

- 대화상자(dialog): alert(), prompt(), confirm() 등의 메서드로 띄우는 창
- alert(): 알림창 또는 경고창



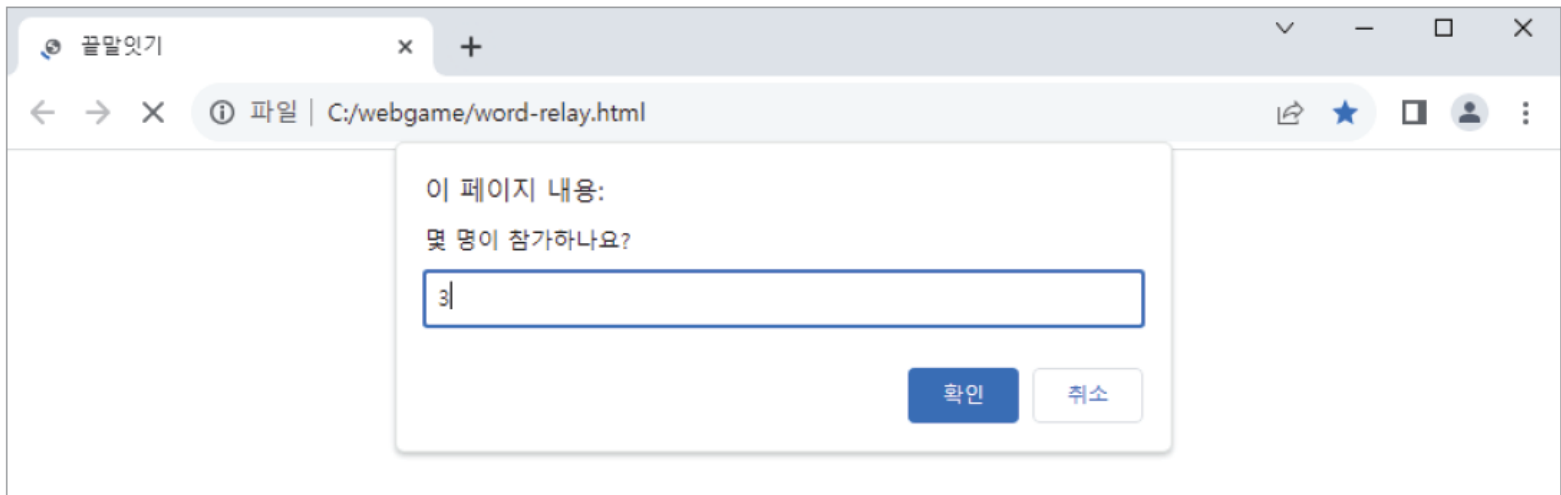
## 4.5 window 객체



- prompt(): 입력창
  - 사용자가 직접 프로그램에 값을 전달할 때 사용

형식 `prompt('사용자에게 표시할 메시지')`

- prompt() 실행결과



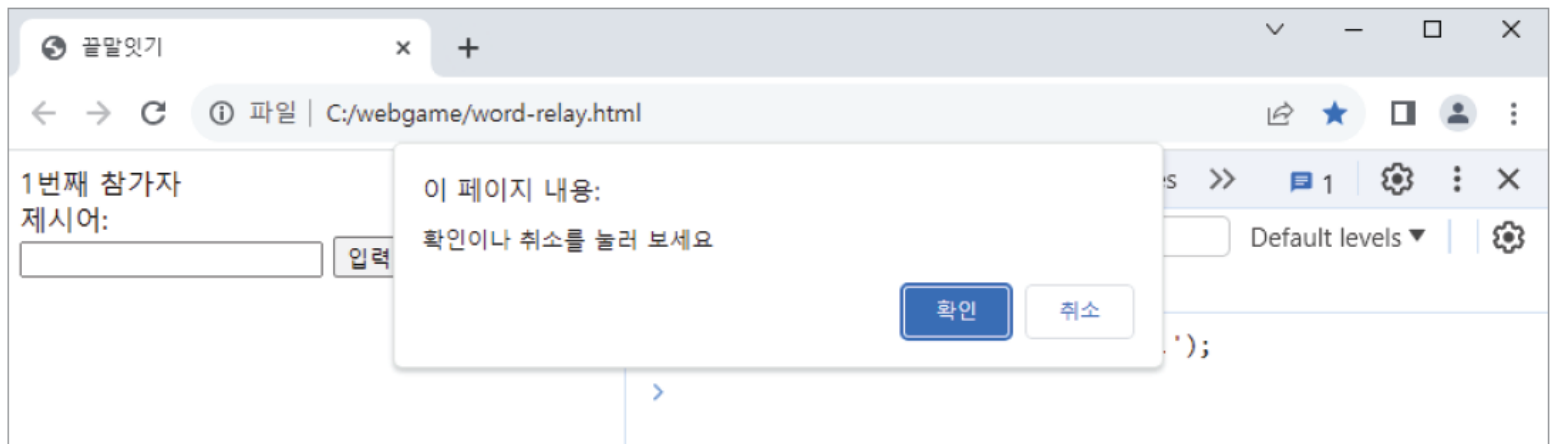
## 4.5 window 객체



- confirm(): 확인장
  - 사용자에게 의사를 물어볼 때 사용

형식 `confirm('사용자에게 표시할 메시지')`

- confirm() 실행결과



## 4.5 window 객체



### 4.5.2 Math 객체

- Math 객체: 수학에 사용하는 다양한 메서드가 제공
  - `Math.ceil()`, `Math.round()`, `Math.floor()`: 올림, 반올림, 내림할 때 사용
  - `Math.max()`, `Math.min()`, `Math.sqrt()`: 최댓값, 최솟값, 제곱근을 구할 때 사용
  - `Math.random()`: 무작위 숫자를 생성할 때 사용



## 4.5 window 객체



### 4.5.3 Date 생성자 함수

- Date 생성자 함수: 날짜 계산을 할 때 사용

형식

```
const <날짜 객체> = new Date(연, 월, 일, 시, 분, 초, 밀리초);  
const <날짜 객체> = new Date(타임스탬프);
```

- setFullYear(), setMonth(), setDate(), setHours(), setMinutes(), setSeconds(), setMilliseconds(): 연, 월, 일, 시, 분, 초, 밀리초를 수정할 때 사용
- getFullYear(), getMonth(), getDate(), getHours(), getMinutes(), getSeconds(), getMilliseconds(): 현재 객체의 연, 월, 일, 시, 분, 초, 밀리초를 가져올 때 사용
- getDay(): 요일을 구할 때 사용