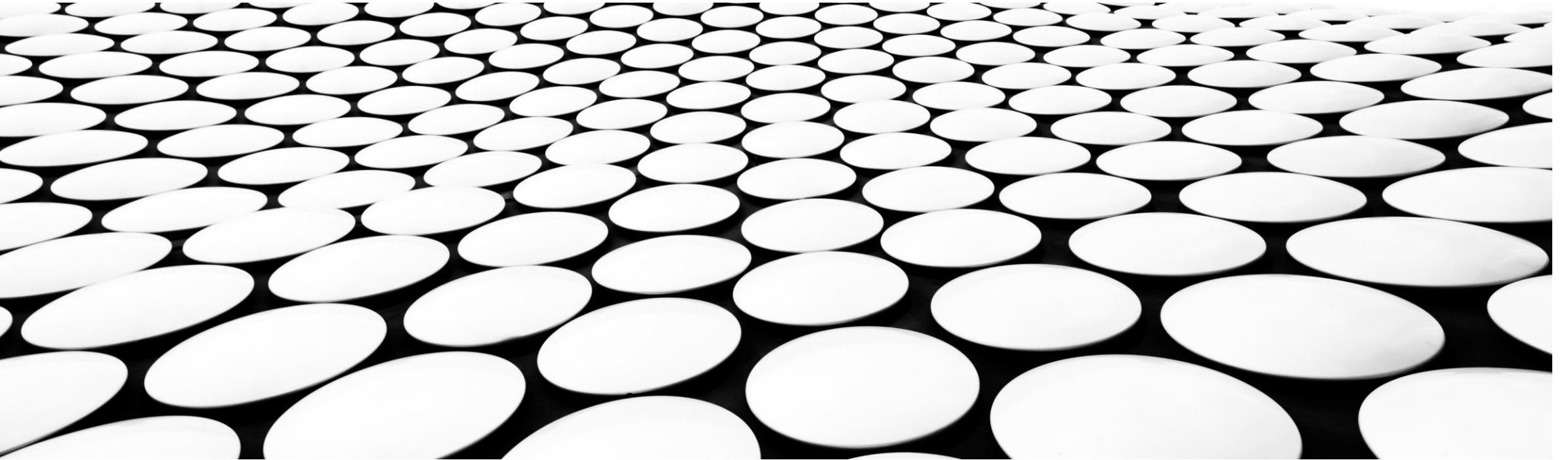


---

# JAVASCRIPT



# JAVASCRIPT

- 자바스크립트(JavaScript)는 객체(object) 기반의 스크립트 언어다.
- HTML로는 웹의 내용을 작성하고, CSS로는 웹을 디자인하며, 자바스크립트로는 웹의 동작을 구현할 수 있다.
- 자바스크립트는 주로 웹 브라우저에서 사용되나, Node.js와 같은 프레임워크를 사용하면 서버 측 프로그래밍에서도 사용할 수 있다.
- 현재 컴퓨터나 스마트폰 등에 포함된 대부분의 웹 브라우저에는 자바스크립트 인터프리터가 내장되어 있다.

## 예제

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Intro</title>
  <script>
    document.write("<h2>여러분을 환영합니다!!</h2>")
  </script>
</head>

<body>
  <noscript>
    <p>여러분의 웹 브라우저가 자바스크립트를 지원하지 않습니다!</p>
  </noscript>
</body>

</html>
```

# JAVASCRIPT

- 자바스크립트(JavaScript)는 객체(object) 기반의 스크립트 언어다.
- HTML로는 웹의 내용을 작성하고, CSS로는 웹을 디자인하며, 자바스크립트로는 웹의 동작을 구현할 수 있다.
- 자바스크립트는 주로 웹 브라우저에서 사용되나, Node.js와 같은 프레임워크를 사용하면 서버 측 프로그래밍에서도 사용할 수 있다.
- 현재 컴퓨터나 스마트폰 등에 포함된 대부분의 웹 브라우저에는 자바스크립트 인터프리터가 내장되어 있다.

## 예제

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Intro</title>
  <script>
    document.write("<h2>여러분을 환영합니다!!</h2>")
  </script>
</head>

<body>
  <noscript>
    <p>여러분의 웹 브라우저가 자바스크립트를 지원하지 않습니다!</p>
  </noscript>
</body>

</html>
```

# JAVASCRIPT

## 자바스크립트란 ?

- 자바스크립트는 정적인 HTML 콘텐츠를 프로그램 구현을 통해 동적으로 변경하거나 사용자와의 상호작용을 담당하게 된다.
- HTML이나 CSS와 달리 자바스크립트는 C언어, 자바와 같은 일반 프로그래밍언어와 비슷한 구조를 가지고 있다. 따라서 단순히 콘텐츠 제작 만을 생각하는 프론트엔드(Front-End) 초보 개발자에게는 가장 어려운 부분이라 할 수 있다.
- 자바스크립트는 객체(Object) 기반의 스크립트 언어로 기본적으로는 웹 브라우저에서 해석되는 인터프리터 언어이며 Node.js와 같은 프레임워크를 사용하면 서버 프로그래밍에도 사용할 수 있다.
- 현재 컴퓨터나 스마트폰 등에 포함된 웹 브라우저에는 자바스크립트 인터프리터가 내장되어 있습니다.

# JAVASCRIPT

## 자바스크립트 특징

- 동적이며 타입을 명시할 필요가 없는 인터프리터 언어 이다.
- 객체지향 프로그래밍과 함수형 프로그래밍을 모두 표현할 수 있다.
- HTML의 내용, 속성, 스타일을 변경 할 수 있다.
- 이벤트를 처리하고 사용자와의 상호작용을 가능하게 함.
- AJAX 기술을 이용해 서버와 실시간 통신 기능을 제공.



C언어와 같은 언어는 소스 파일을 작성한 후, 해당 파일을 컴파일(compile)하여 사용자가 실행할 수 있는 실행 파일(.exe)로 만들어 사용합니다. 하지만 인터프리터 언어는 이러한 컴파일 작업을 거치지 않고, 소스 코드를 바로 실행할 수 있는 언어를 의미합니다. 자바스크립트는 웹 브라우저에 포함된 자바스크립트 인터프리터가 소스 코드를 직접 해석하여 바로 실행해 줍니다.

# JAVASCRIPT

## 자바와 자바스크립트

- 자바와 자바스크립트는 그 이름만 놓고 보면 서로 관련이 있는 언어로 생각되기 쉽다.
- 하지만 두 언어는 서로 직접적인 관련은 없으며, 비슷한 점보다는 다른 점이 훨씬 많다.
- 문법상 비슷한 부분은 두 언어의 문법이 모두 C 언어를 기반으로 만들어졌기 때문이다.

자바	자바스크립트
컴파일 언어	인터프리터 언어
타입 검사를 엄격하게 함.	타입을 명시하지 않음.
클래스(class) 기반의 객체 지향 언어	프로토타입(prototype) 기반의 객체 지향 언어

# JAVASCRIPT

## 자바와 자바스크립트

- 자바와 자바스크립트는 그 이름만 놓고 보면 서로 관련이 있는 언어로 생각되기 쉽다.
- 하지만 두 언어는 서로 직접적인 관련은 없으며, 비슷한 점보다는 다른 점이 훨씬 많다.
- 문법상 비슷한 부분은 두 언어의 문법이 모두 C 언어를 기반으로 만들어졌기 때문이다.

자바	자바스크립트
컴파일 언어	인터프리터 언어
타입 검사를 엄격하게 함.	타입을 명시하지 않음.
클래스(class) 기반의 객체 지향 언어	프로토타입(prototype) 기반의 객체 지향 언어

# JAVASCRIPT

## 자바스크립트 표준

- 1996년에 넷스케이프(Netscape)는 자바스크립트를 국제 표준안으로 만들기 위해 ECMA(European Computer Manufacturers Association)에 제출한다.
- 그 결과 ECMA는 ECMAScript라는 새로운 표준을 제정하였고, 그 첫 번째 버전인 ECMA-262를 1997년에 공표한다.
- ECMAScript는 자바스크립트뿐만 아니라 마이크로소프트의 Jscript나 어도비의 액션스크립트도 따르는 국제 표준이 된다.
- 현재 자바스크립트의 최신 표준은 2015년에 발표된 ECMAScript 6이다.



# JAVASCRIPT

## ES5(2009) 주요 특징

- 10년 만에 release된 새로운 javascript 버전.
- 일반적으로 javascript의 가장 기본이 되는 버전.
- 배열에 forEach, map, filter, reduce, some, every 등의 메서드 지원.
- Object 에 대한 getter/setter 지원.
- JSON 지원.

# JAVASCRIPT

## ES6(2015)

- 2019년 현재 가장 보편적으로 적용되는 JS 버전 임.
- 대부분의 브라우저에서 지원되는 사실상의 표준 규격.
- let, const 키워드 추가.
- arrow 문법 지원.
- iterator / generator 추가.
- module import / export 추가.
- Promise : javascript 비동기 콜백 문제 해결.

# JAVASCRIPT

## ES8(2017) 주요 특징

- 향후 점진적으로 고려해야 할 JS 버전.
- async / await 비동기 관련 처리 추가.
- Object, String 등에 기능 추가.

# JAVASCRIPT

Vue, Angular, React

- JavaScript가 현재 가장 널리 사용되는 분야는 프론트엔드 프레임워크 영역 이다. JavaScript 본래의 영역 이기도 하고 하이브리드 앱, 모바일 웹, 최근에는 데스크톱 영역까지 확대되었다.
- Vue, Angular, React 는 가장 대표적인 프론트엔드 프레임워크로 향후 기본적인 JavaScript 학습 이후 도전해야 할 것들이다.

# JAVASCRIPT

## AngularJS

- 구글에서 만들었으며 MIT 라이선스로 누구나 무료로 사용 할 수 있다.
- 본격적인 자바스크립트 프론트엔드 프레임워크의 시초라 할 수 있으며 2010년 처음 발표 되었다.

- 주요 특징
  - 기존보다 훨씬 적은 코드로 원하는 기능을 구현할 수 있음.
  - 양방향 데이터 바인딩.
  - MVC, MVVM 을 지원해 구성요소의 명확한 분리가 가능.

# JAVASCRIPT

## AngularJS

- 구글에서 만들었으며 MIT 라이선스로 누구나 무료로 사용할 수 있다.
- 본격적인 자바스크립트 프론트엔드 프레임워크의 시초라 할 수 있으며 2010년 처음 발표 되었다.
- AngularJS2(이후 Angular) 가 2016년에 발표되면서 타입스크립트(Type Script)가 기본으로 사용되고, 구성방식, 아키텍처, 도구 등의 변경으로 기존 개발자들이 적응하기 어렵게 되었으며 점점 거대화 되는 구조와 높은 학습 곡선(Learning Curve) 그리고 React 와 Vue 같은 새로운 프레임워크의 등장으로 현재는 처음보다 사용자는 줄어든 상태라 볼 수 있다.

### ■ 주요 특징

- 기존보다 훨씬 적은 코드로 원하는 기능을 구현할 수 있음.
- 양방향 데이터 바인딩.
- MVC, MVVM 을 지원해 구성요소의 명확한 분리가 가능.

# JAVASCRIPT

## React

- 페이스북에서 만들었으며 UI 라이브러리에 특화되어 있으며 재사용 가능한 UI 컴포넌트 생성을 지원 한다. 또한 가상 돔(Virtual DOM)이라는 개념을 사용해 보다 빠르게 웹 콘텐츠의 동적 처리를 가능하게 한다.
- React 자체는 자바스크립트에 대한 기본적인 경험이 있다면 배우기 쉽다고 알려져 있다. React 가 인기를 더해가는 이유 중 하나에는 React Native 가 있다. React Native 는 React 로 만든 웹 어플리케이션을 안드로이드나 iOS 의 네이티브 앱으로 만들 수 있도록 도와주는 라이브러리로 이해할 수 있다.
- 기존 하이브리드 방식 보다 훨씬 다양한 기능과 속도 향상이 있어 최근 많은 개발에 활용되는 추세 이다.

## ■ 주요 특징

- Virtual DOM을 사용 한다.
- JSX(JavaScript XML)를 사용한다.(Vue.js등 여러 곳에서 사용)
- Airbnb, Netflix, Dropbox, Twitter, Evernote, Uber 등에서 사용.
- 프레임워크 보다는 View 라이브러리에 가까움.

# JAVASCRIPT

## Vue.js

- Vue는 사용자 인터페이스를 만들기 위한 진보적인 프레임워크로 다른 단일형 프레임워크와 달리 점진적으로 채택할 수 있도록 설계 되어 있다. 2014년 구글을 위해 Angular를 작업하던 Evan You에 의해 처음 발표 되었으며 React, Angular의 장점을 통합해 빠른 속도로 성장하고 있는 가장 최신의 프레임워크 이다.
- Vue.js는 전세계적으로 최근 가장 주목받는 프론트엔드 프레임워크 이다.

### ■ 주요 특징

- Virtual DOM을 사용 한다.
- 전반적인 문서화가 잘되어 있고 특히 한글 문서가 잘 정리되어 있음.
- 서버사이드 렌더링이 React 보다 훌륭함.
- 부분적 도입에서부터 전체로의 적용까지 유연하게 적용할 수 있음.
- 2.0 버전에서 부터 개발 편의성의 대폭 향상.



# JAVASCRIPT

## Node.js

- Node.js(노드)는 자바스크립트를 백 엔드 프로그램 개발에 사용할 수 있도록 만들어진 서버사이드 자바스크립트 런타임 이다. 정확하게는 Chrome V8 JavaScript Engine에 기반하고 있다.
- 기본적으로 스레드를 사용하지 않도록 설계 되었지만 필요하다면 다수의 cpu 코어에 로드밸런싱이 가능한 구조로 경량의 빠른 웹 서버 개발에 적합 하다.
- 별도의 웹 서버 소프트웨어 없이 자체적으로 웹 서버 구동이 가능하고 자바스크립트 문법을 통해 서버 프로그램 구현이 가능한 구조 이다.
- 기존의 대형 시스템들을 여러 개의 독립된 소형 서비스로 분리 하는 마이크로 서비스 아키텍처(MSA: Micro Service Architecture)의 확산으로 node.js 의 활용이 급증하게 되었다.

- 주요 특징
  - 이벤트 기반, 논 블로킹 I/O모델 사용.
  - npm 이라고 하는 패키지 매니저 사용.(65만개 이상의 오픈소스 패키지)
  - 빠른 처리 속도와 뛰어난 확장성
- Node.js 주요 적용 분야
  - 입출력이 잦은 어플리케이션
  - 데이터 스트리밍 어플리케이션
  - 데이터를 실시간으로 다루는 어플리케이션
  - 경량의 Restful API 기반 어플리케이션
  - 싱글 페이지 어플리케이션

# JAVASCRIPT 문법

프로그램(program)이란?

- 프로그램은 컴퓨터가 실행할 수 있는 명령(instruction)으로 이루어진다.
- 컴퓨터 프로그래밍에서 컴퓨터가 실행할 수 있는 명령들을 실행 문(statement)이라고 한다.
- 즉, 프로그램이란 특정 결과를 얻기 위해서 컴퓨터에 의해 실행되는 실행 문의 집합이라고 할 수 있다.

# JAVASCRIPT 문법

## 소스코드위치

- 자바스크립트는 기본적으로 HTML 문서의 <head></head> 사이에 위치 한다. 그러나 그 외 위치에 둘 수도 있고 외부파일이나 다른 서버를 통해 참조하는 방식으로도 사용할 수 있다.
- 여기서는 자바스크립트의 소스가 위치하는 몇몇 유형에 대해 살펴 본다.
  - 내부 JavaScript
  - 외부 JavaScript

# JAVASCRIPT 문법

## 내부 JavaScript

- HTML 문서 내부에 자바스크립트 소스코드를 두는 유형입니다. <head></head> 혹은 <body></body> 에 둘 수 있으며 양쪽에 모두 있어도 상관 없다.
  - 현재 HTML 파일의 문서 구조(DOM)에 쉽게 접근이 가능.
  - 현재 화면에 동적인 요소를 부여하는 자바스크립트가 같은 소스 파일에 위치하기 때문에 코드 관리와 이해가 쉬움.
  - 자바스크립트 소스가 복잡해지는 경우 관리가 어려움.
  - 공통된 기능을 만들기 어렵고 코드의 재활용이 어려움.

```
<script>  
  alert('hello world');  
</script>
```

# JAVASCRIPT 문법

## 외부 JavaScript

- HTML 문서 외부에 별도의 파일을 생성하고 HTML에서 불러와 사용하는 방식 이다. 이때 자바스크립트 파일의 위치는 HTML과 동일한 서버 혹은 외부 서버 일수도 있다.
  - 웹의 HTML코드로부터 웹의 동작을 구현하는 자바스크립트 코드를 분리할 수 있음.
  - HTML과 자바스크립트 코드의 읽기가 수월해지고 유지 보수 간편.
  - 공통 기능의 모듈화와 코드 재활용이 가능해 짐.
  - 소스가 분리되어 있고 HTML, 자바스크립트 모두 복잡한 경우 소스 이해가 어려울 수 있음.

```
//external.js 파일  
function printDate() {  
    alert('hello world');  
}
```

```
<head>  
  <script src="../external.js"></script>  
  <!-- 외부 서버의 js파일 참고인 경우 다음과 같이 사용 -->  
  <script src="https://www.cdn.com/myjs/external.js"></script>  
</head>
```

# JAVASCRIPT 문법

## 소스파일 위치 결정

### ■ <head> </head>

- 브라우저 렌더링에 방해가 될 수 있으며 무거운 스크립트가 실행되는 경우 오랫동안 화면이 보여지지 않을 수 있음.
- 문서를 초기화하거나 설정하는 가벼운 스크립트들을 주로 사용.
- 문서의 DOM(Document Object Model) 구조가 필요한 경우 HTML이 모두 로드 된 이후 실행되어야 하므로 window.onload와 같은 로드 이벤트가 추가되어야 함.

### ■ <body> </body>

- 태그내 모든 위치에 둘 수 있음.
- 웹 페이지 로딩이 완료된 다음 실행하기 위해 일반적으로는 </body> 바로 앞에 위치.
- 이경우 문서의 DOM 구조가 완료된 시점에 실행되기에 별다른 추가 설정이 필요 없음.

# JAVASCRIPT 문법

## 소스파일 위치 결정

- 내부 자바스크립트 VS 외부 자바스크립트
  - 비교적 간단한 코드로 구성되며 현재 파일에만 적용되는 경우 내부 자바스크립트를 사용.
  - 공통 기능 구현이나 소스가 길어지면 외부 자바스크립트로 관리함.
  - 공통 라이브러리로 개발된 경우 CDN을 통해 외부 서버로 부터 참조해서 사용함.

# JAVASCRIPT 문법

## 변수와 자료 형

- 자바스크립트는 다른 언어들과 달리 자료 형이 고정되어 있지 않은 동적 타입 언어 이다.
- 따라서 변수를 선언할 때 별도의 자료 형을 명시하지 않아도 된다.



# JAVASCRIPT 문법

## 자료 형

- 내부적으로는 Primitive(기본형)과 Object(객체형)이 있다.

# JAVASCRIPT 문법

## Primitive

- Boolean: true, false
- null: 빈 값을 표현
- undefined: 값을 할당하지 않은 변수가 가지는 값
- Number: 숫자 형으로 정수와 부동 소수점, 무한대 및 NaN(숫자가 아님)등.
- String: 문자열

## Object

- Reference 타입
- 클래스 뿐만 아니라, 배열과 함수, 사용자 정의 클래스도 모두 Object.
- JSON(Java Script Object Notation)의 기본 구조

# JAVASCRIPT 문법

## 변수 선언

- 변수 이름은 대소문자를 구별.
- 여러 변수를 한번에 선언할 수 있음.
- 지역 변수와 전역 변수가 있음.
- 기본적으로 소문자로 시작되는 Camel Case 를 많이 사용

## \* 식별 자 작성 방식

- 자바스크립트에서는 식별 자를 작성할 때 다음과 같은 작성 방식을 사용할 수 있다.
  - 1. Camel Case 방식
  - 2. Underscore Case 방식
- Camel Case 방식이란 식별자가 여러 단어로 이루어질 경우에 첫 번째 단어는 모두 소문자로 작성하고, 그 다음 단어부터는 첫 문자만 대문자로 작성하는 방식이다.
- Underscore Case 방식은 식별 자를 이루는 단어들을 소문자로만 작성하고, 그 단어들은 언더스코어(\_)로 연결하는 방식이다.
- 자바스크립트에서는 식별 자를 작성할 때 관행적으로 Camel Case 방식을 많이 사용한다.

# JAVASCRIPT 문법

## 변수 선언 - var, let, const

- ES6 이전에는 var 만 존재했으며 function-scoped 로 인해 다른 언어들과 다른 문제가 있었음.
  - var는 지역변수 개념으로 함수 범위에서 유효함.
  - var를 선언하지 않으면 자동으로 전역 변수가 됨.
  - let과 const 는 ES6 에서 등장한 block-scoped 변수 선언.
  - let은 값의 재할당이 가능하고 const 는 불가능(immutable).
  - const로 선언된 배열이나 객체의 경우 새로운 객체로 재할당하는 것은 안되고, 배열 값의 변경/추가, 객체의 필드 변경 등은 가능.

# JAVASCRIPT 문법

변수 선언 - var, let, const

```
var foo = 'foo1';  
console.log(foo); // foo1  
  
if (true) {  
  var foo = 'foo2';  
  console.log(foo); // foo2  
}  
  
console.log(foo); // foo2
```

**var를 이용한 예제**

```
let foo = 'foo1';  
const bar = 'bar1';  
console.log(foo); // foo1  
  
if (true) {  
  let foo = 'foo2';  
  console.log(foo); // foo2  
  console.log(bar); // bar1  
}  
  
console.log(foo); // foo1  
bar = 'bar2'; // error
```

**let, const를 이용한 예제**

# JAVASCRIPT 문법

## 자바스크립트 출력

- 자바스크립트는 여러 방법을 통해 결과물을 HTML 페이지에 출력할 수 있다.
- 자바스크립트에서 사용할 수 있는 출력 방법은 다음과 같다.
  - 1. window.alert() 메소드
  - 2. HTML DOM 요소를 이용한 innerHTML 프로퍼티
  - 3. document.write() 메소드
  - 4. console.log() 메소드

# JAVASCRIPT 문법

## window.alert() method

- 자바스크립트에서 가장 간단하게 데이터를 출력할 수 있는 방법은 window.alert() 메소드를 이용하는 것이다.
- Window.alert() 메소드는 브라우저와는 별도의 대화 상자를 띄워 사용자에게 데이터를 전달해 준다.

```
<script>
    function alertDialogBox() {
        alert("확인을 누를 때까지 다른 작업을 할 수 없어요!");
    }
</script>
```

# JAVASCRIPT 문법

## HTML DOM 요소를 이용한 innerHTML property

- 실제 javascript 코드에서 출력을 위해 가장 많이 사용되는 방법은 HTML DOM 요소를 이용한 innerHTML property를 이용하는 방법입니다.
- 우선 document 객체의 getElementById()나 getElementsByTagName() 등의 method를 사용하여 HTML 요소를 선택한다.
- 그리고서 innerHTML property를 이용하면 선택된 HTML 요소의 내용(content)이나 속성(attribute)값 등을 손쉽게 변경할 수 있다.

예제

```
<script>  
  var str = document.getElementById("text");  
  str.innerHTML = "이 문장으로 바뀌었습니다!";  
</script>
```



# JAVASCRIPT 문법

## document.write() method

- document.write() 메소드는 웹 페이지가 로딩될 때 실행되면, 웹 페이지에 가장 먼저 데이터를 출력한다.
- 따라서 document.write() 메소드는 대부분 테스트나 디버깅을 위해 사용된다.

예제

```
<script>  
    document.write(4 * 5);  
</script>
```

# JAVASCRIPT 문법

## document.write() method

- document.write() 메소드는 웹 페이지가 로딩될 때 실행되면, 웹 페이지에 가장 먼저 데이터를 출력한다.
- 따라서 document.write() 메소드는 대부분 테스트나 디버깅을 위해 사용된다.
- 하지만 웹 페이지의 모든 내용이 로딩된 후에 document.write() 메소드가 실행되면, 웹 페이지 내에 먼저 로딩된 모든 데이터를 지우고 자신의 데이터를 출력하게 된다.
- 따라서 document.write() 메소드를 테스트 이외의 용도로 사용할 때에는 충분히 주의해서 사용해야 한다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Output</title>
</head>

<body>

  <h1>Document 객체의 write() 메소드</h1>

  <button onclick="document.write(4 * 5)">버튼을 눌러보세요!</button>

</body>
</html>
```

# JAVASCRIPT 문법

## console.log() method

- console.log() method는 웹 브라우저의 콘솔을 통해 데이터를 출력해 준다.
- 대부분의 주요 웹 브라우저에서는 F12를 누른 후, 메뉴에서 콘솔을 클릭하면 콘솔 화면을 사용할 수 있다.
- 이러한 콘솔 화면을 통한 데이터의 출력은 좀 더 자세한 사항까지 출력되므로, 디버깅하는데 많은 도움을 줍니다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Output</title>
</head>

<body>

  <h1>console.log() 메소드</h1>

  <p>F12를 눌러서 콘솔 화면을 열면 결과를 확인할 수 있습니다.</p>
  <script>
    console.log(4 * 5);
  </script>

</body>

</html>
```

# JAVASCRIPT 문법 – 타입(TYPE)

## 기본 타입

- 타입(data type)이란 프로그램에서 다룰 수 있는 값의 종류를 의미한다.
- 자바스크립트에서는 여러 가지 형태의 타입을 미리 정의하여 제공하고 있으며, 이것을 기본 타입이라고 한다.
- 자바스크립트의 기본 타입은 크게 원시 타입과 객체 타입으로 구분할 수 있다.

# JAVASCRIPT 문법 – 타입(TYPE)

원시 타입(primitive type)

- 1. 숫자(number)
- 2. 문자열(string)
- 3. 불리언(boolean)
- 4. 심볼(symbol) : ECMAScript 6부터 제공됨
- 5. undefined

객체 타입(object type)

- 객체(object)

# JAVASCRIPT 문법 – 타입(TYPE)

## 숫자(number)

- 자바스크립트는 다른 언어와는 달리 정수와 실수를 따로 구분하지 않고, 모든 수를 실수 하나로만 표현한다.
- 또한, 매우 큰 수나 매우 작은 수를 표현할 경우에는 e(지수) 표기법을 사용할 수 있다.

### 예제

```
var firstNum = 10;    // 소수점을 사용하지 않은 표현
var secondNum = 10.00; // 소수점을 사용한 표현
var thirdNum = 10e6;   // 10000000
var fourthNum = 10e-6; // 0.00001
```

# JAVASCRIPT 문법 – 타입(TYPE)

## 문자열(string)

- 자바스크립트에서 문자열은 큰따옴표("")나 작은따옴표('')로 둘러싸인 문자의 집합을 의미한다.
- 큰따옴표는 작은따옴표로 둘러싸인 문자열에만 포함될 수 있으며, 작은따옴표는 큰따옴표로 둘러싸인 문자열에만 포함될 수 있다.
- 자바스크립트에서는 숫자와 문자열을 더할 수도 있다.
- 이럴 경우에 자바스크립트는 숫자를 문자열로 자동 변환하여, 두 문자열을 연결하는 연산을 수행한다.

### 예제

```
var firstStr = "이것도 문자열입니다."; // 큰따옴표를 사용한 문자열
var secondStr = '이것도 문자열입니다.'; // 작은따옴표를 사용한 문자열
var thirdStr = "나의 이름은 '홍길동'이야." // 작은따옴표는 큰따옴표로 둘러싸인 문자열에만 포함될 수 있음.
var fourthStr = '나의 이름은 "홍길동"이야.' // 큰따옴표는 작은따옴표로 둘러싸인 문자열에만 포함될 수 있음.
```

### 예제

```
var num = 10;
var str = "JavaScript";
document.getElementById("result").innerHTML = (num + str); // 10JavaScript
```

# JAVASCRIPT 문법 – 타입(TYPE)

## 불리언(boolean)

- 불리언 값은 참(true)과 거짓(false)을 표현한다.
- 자바스크립트에서 불리언 값은 예약어인 true와 false를 사용하여 나타낼 수 있다.

예제

```
var firstNum = 10;  
var secondNum = 11;  
document.getElementById("result").innerHTML = (firstNum == secondNum); // false
```



# JAVASCRIPT 문법 – 타입(TYPE)

## 심볼(symbol)

- 심볼 타입은 ECMAScript 6부터 새롭게 추가된 타입이다.
- 심볼은 유일하고 변경할 수 없는 타입으로, 객체의 프로퍼티를 위한 식별자로 사용할 수 있다.

### 예제

```
var sym = Symbol("javascript"); // symbol 타입  
var symObj = Object(sym);      // object 타입
```

# JAVASCRIPT 문법 – 타입(TYPE)

## typeof 연산자

- typeof 연산자는 피연산자의 타입을 반환하는 피연산자가 단 하나뿐인 연산자이다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript DataType</title>
</head>

<body>

  <h1>typeof 연산자</h1>
  <p id="result"></p>

  <script>
    document.getElementById("result").innerHTML = (typeof 10) + "<br>";
    document.getElementById("result").innerHTML += (typeof "문자열") + "<br>";
    document.getElementById("result").innerHTML += (typeof true) + "<br>";
    document.getElementById("result").innerHTML += (typeof undefined) + "<br>";
    document.getElementById("result").innerHTML += (typeof null);
  </script>

</body>

</html>
```

# JAVASCRIPT 문법 – 타입(TYPE)

## null과 undefined

- 자바스크립트에서 null이란 object 타입이며, 아직 '값'이 정해지지 않은 것을 의미한다.
- 또한, undefined란 null과는 달리 '타입'이 정해지지 않은 것을 의미한다.
- 따라서 자바스크립트에서 undefined는 초기화되지 않은 변수나 존재하지 않는 값에 접근할 때 반환된다.
- null과 undefined는 동등 연산자(==)와 일치 연산자(===)로 비교할 때 그 결과 값이 다르므로 주의해야 한다.
- Null과 undefined는 타입을 제외하면 같은 의미지만, 타입이 다르므로 일치하지는 않는다.

# JAVASCRIPT 문법 – 타입(TYPE)

## null과 undefined

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript DataType</title>
</head>

<body>

  <h1>null과 undefined</h1>
  <p id="result"></p>

  <script>
    var num; // 초기화하지 않았으므로 undefined 값을 반환함.
    var str = null; // object 타입의 null 값

    // 정의되지 않은 변수에 접근하면 undefined 값을 반환함.
    document.getElementById("result").innerHTML =
      (typeof num) + "<br>" + (typeof str) + "<br>" + (typeof secondNum);
  </script>

</body>

</html>
```

# JAVASCRIPT 문법 – 타입(TYPE)

## 객체(object)

- 자바스크립트의 기본 타입은 객체(object)이다.
- 객체(object)란 실생활에서 우리가 인식할 수 있는 사물로 이해할 수 있다.
- 객체는 여러 프로퍼티(property)나 메소드(method)를 같은 이름으로 묶어놓은 일종의 집합체이다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript DataType</title>
</head>

<body>

  <h1>객체 타입</h1>
  <p id="result"></p>

  <script>
    var dog = { name: "해피", age: 3 };    // 객체의 생성

    // 객체의 프로퍼티 참조
    document.getElementById("result").innerHTML =
      "강아지의 이름은 " + dog.name + "이고, 나이는 " + dog.age + "살 입니다.";
  </script>

</body>
</html>
```

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 타입 변환(type conversion)

- 자바스크립트는 타입 검사가 매우 유연한 언어이다.
- 자바스크립트의 변수는 타입이 정해져 있지 않으며, 같은 변수에 다른 타입의 값을 다시 대입할 수도 있다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Type Conversion</title>
</head>

<body>

  <h1>같은 변수에 다른 타입의 값 대입</h1>
  <p id="result"></p>

  <script>
    var num = 20; // 변수의 선언과 함께 초기화
    document.getElementById("result").innerHTML = num + "<br>";
    num = "이십"; // 문자열 대입
    document.getElementById("result").innerHTML += num + "<br>";
    var num; // 한 변수를 여러 번 초기화할 수는 있으나, 재선언은 할 수 없습니다.
    document.getElementById("result").innerHTML += num;
  </script>

</body>

</html>
```

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 묵시적 타입 변환(implicit type conversion)

- 자바스크립트는 특정 타입의 값을 기대하는 곳에 다른 타입의 값이 오면, 자동으로 타입을 변환하여 사용한다.
- 즉, 문자열 값이 오길 기대하는 곳에 숫자가 오더라도 자바스크립트는 알아서 숫자를 문자열로 변환하여 사용한다.
- 옆의 세 번째 예제에서 뺄셈 연산을 위해 문자열이 숫자로 변환되어야 하나, 해당 문자열은 두 번째 예제의 문자열과는 달리 숫자로 변환될 수 없는 문자열이다.
- 따라서 의미에 맞게 자동으로 타입을 변환할 수 없으므로, 자바스크립트는 NaN 값을 반환한다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Type Conversion</title>
</head>

<body>

  <h1>묵시적 타입 변환</h1>
  <p id="result"></p>

  <script>
    var result = 10 + "문자열"; // 문자열 연결을 위해 숫자 10이 문자열로 변환됨.
    document.getElementById("result").innerHTML = result + "<br>";
    result = "3" * "5"; // 곱셈 연산을 위해 두 문자열이 모두 숫자로 변환됨.
    document.getElementById("result").innerHTML += result + "<br>";
    result = 1 - "문자열"; // NaN
    document.getElementById("result").innerHTML += result;
  </script>

</body>

</html>
```



NaN은 Not a Number의 축약형으로, 정의되지 않은 값이나 표현할 수 없는 값이라는 의미를 가집니다.

이러한 NaN은 Number 타입의 값으로 0을 0으로 나누거나, 숫자로 변환할 수 없는 피연산자로 산술 연산을 시도하는 경우에 반환되는 읽기 전용 값입니다.

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 명시적 타입 변환(explicit type conversion)

- 자바스크립트에서는 묵시적 타입 변환을 많이 사용하지만, 명시적으로 타입을 변환할 방법도 제공합니다.
- 명시적 타입 변환을 위해 자바스크립트가 제공하는 전역 함수는 다음과 같습니다.
  1. Number()
  2. String()
  3. Boolean()
  4. Object()
  5. parseInt()
  6. parseFloat()

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Type Conversion</title>
</head>

<body>

  <h1>명시적 타입 변환 함수</h1>
  <p id="result"></p>

  <script>
    document.getElementById("result").innerHTML = Number("10") + "<br>"; // 10
    document.getElementById("result").innerHTML += String(true) + "<br>"; // "true"
    document.getElementById("result").innerHTML += Boolean(0) + "<br>"; // false
    document.getElementById("result").innerHTML += Object(3); // new Number(3)
  </script>

</body>

</html>
```



# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 숫자를 문자열로 변환

- 숫자를 문자열로 변환하는 가장 간단한 방법은 String() 함수를 사용하는 것이다.
- 또한, null과 undefined를 제외한 모든 타입의 값이 가지고 있는 toString() 메소드를 사용할 수도 있다.
- 숫자(Number) 객체는 숫자를 문자열로 변환하는 다음과 같은 메소드를 별도로 제공한다.
  1. toExponential()
  2. toFixed()
  3. toPrecision()
- 메소드(method)란 객체의 프로퍼티 값으로 함수를 갖는 프로퍼티를 의미한다.

메소드	설명
toExponential()	정수 부분은 1자리, 소수 부분은 입력받은 수만큼 e 표기법을 사용하여 숫자를 문자열로 변환함.
toFixed()	소수 부분을 입력받은 수만큼 사용하여 숫자를 문자열로 변환함.
toPrecision()	입력받은 수만큼 유효 자릿수를 사용하여 숫자를 문자열로 변환함.

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 불리언 값을 문자열로 변환

- 불리언 값을 문자열로 변환하는 방법에는 String() 함수와 toString() 메소드를 사용하는 방법이 있습니다.
- 예제
- String(true); // 문자열 "true"
- false.toString(); // 문자열 "false"

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 날짜를 문자열이나 숫자로 변환

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Type Conversion</title>
</head>

<body>

  <h1>날짜를 문자열이나 숫자로 변환</h1>
  <p id="result"></p>

  <script>
    var result = String(Date());
    document.getElementById("result").innerHTML = result + "<br>";
    result = Date().toString();
    document.getElementById("result").innerHTML += result + "<br>";
    var date = new Date();          // Date 객체 생성
    result = date.getFullYear();
    document.getElementById("result").innerHTML += result + "<br>";
    result = date.getTime();        // 1970년 1월 1일부터 현재까지의 시간을 밀리초 단위의 숫자로 반환함.
    document.getElementById("result").innerHTML += result;
  </script>

</body>
</html>
```

## 날짜를 문자열이나 숫자로 변환

Sun Oct 24 2021 21:13:41 GMT+0900 (한국 표준시)  
Sun Oct 24 2021 21:13:41 GMT+0900 (한국 표준시)  
2021  
1635077621537

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 문자열을 숫자로 변환

- 문자열을 숫자로 변환하는 가장 간단한 방법은 Number() 함수를 사용하는 것입니다.
- 자바스크립트는 문자열을 숫자로 변환해 주는 두 개의 전역 함수를 별도로 제공합니다.
  - 1. parseInt()
  - 2. parseFloat()

함수	설명
parseInt()	문자열을 파싱하여 특정 진법의 정수를 반환함.
parseFloat()	문자열을 파싱하여 부동 소수점 수를 반환함.

# JAVASCRIPT 문법 – 타입(TYPE) 변환

## 불리언 값을 숫자로 변환

- 불리언 값을 숫자로 변환하는 방법에는 Number() 함수를 사용하는 방법이 있다.
- 예제
- `Number(true);` // 숫자 1
- `Number(false);` // 숫자 0

# JAVASCRIPT 문법 - 변수

## 변수의 선언과 초기화

- 변수(variable)란 데이터(data)를 저장할 수 있는 메모리 공간을 의미하며, 그 값이 변경될 수 있다.
- 자바스크립트에서는 var 키워드를 사용하여 변수를 선언한다.
- 자바스크립트에서는 선언되지 않은 변수를 사용하려고 하거나 접근하려고 하면 오류가 발생한다.
- 단, 선언되지 않은 변수를 초기화할 경우에는 자동으로 선언을 먼저 한 후 초기화를 진행한다.
  - `var month;` // month라는 이름의 변수 선언
  - `date = 25;` // date라는 이름의 변수를 묵시적으로 선언
- 선언된 변수는 나중에 초기화할 수도 있고, 선언과 동시에 초기화할 수도 있다.
  - `var month;` // 변수의 선언
  - `var date = 25;` // 변수의 선언과 동시에 초기화
  - `month = 12;` // 변수의 초기화

- **쉼표(,) 연산자를 이용하여 여러 변수를 동시에 선언하거나 초기화할 수도 있습니다.**
  - `var month, date;` // 여러 변수를 한 번에 선언
  - `var hours = 7, minutes = 15;` // 여러 변수를 선언과 동시에 초기화
  - `month = 10, date = 5;` // 여러 변수를 한 번에 초기화

# JAVASCRIPT 문법 - 변수

## 변수의 타입과 초기 값

- 자바스크립트의 변수는 타입이 정해져 있지 않으며, 같은 변수에 다른 타입의 값을 다시 대입할 수도 있다.
- 이렇게 한 변수에 다른 타입의 값을 여러 번 대입할 수는 있지만, 한 번 선언된 변수를 재 선언할 수는 없다.
- 자바스크립트에서 선언만 되고 초기화하지 않은 변수는 undefined 값을 갖습니다.
  - var num; // undefined
  - num = 10; // 10

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Variables</title>
</head>

<body>

  <h1>같은 변수에 다른 타입의 값 할당</h1>
  <p id="result"></p>

  <script>
    var num = 10;          // 변수의 선언과 함께 초기화
    document.getElementById("result").innerHTML = num + "<br>";
    num = [10, 20, 30];    // 배열 할당
    document.getElementById("result").innerHTML += num + "<br>";
    var num;               // 이 재선언문은 무시됨.
    document.getElementById("result").innerHTML += num;
  </script>

</body>

</html>
```

# JAVASCRIPT 문법 - 변수

## 변수의 이름

- 자바스크립트에서 변수는 이름을 가지고 식별하므로, 변수의 이름은 식별자(identifier)이다.
- 변수의 이름은 영문자(대소문자), 숫자, 언더스코어(\_) 또는 달러(\$)로만 구성된다.
- 또한, 숫자와의 구분을 빠르게 하려고 숫자로 시작할 수 없다.
- 이러한 변수의 이름은 대소문자를 구분하며, 자바스크립트 언어에서 예약된 키워드는 이름으로 사용할 수 없다.



# JAVASCRIPT 문법 – 연산자

연산자(operator)

- 자바스크립트는 여러 종류의 연산을 위한 다양한 연산자(operator)를 제공하고 있다.
  - 산술연산자, 대입연산자, 증감연산자, 비교연산자, 논리연산자, 비트연산자, etc

# JAVASCRIPT 문법 – 연산자

## 산술 연산자

- 산술 연산자는 사칙연산을 다루는 가장 기본적인 면서도 많이 사용하는 연산자이다.
- 산술 연산자는 모두 두 개의 피 연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽이다.
  - 항이란 해당 연산의 실행이 가능하기 위해 필요한 값이나 변수를 의미한다.
  - 따라서 이항 연산 자란 해당 연산의 실행을 위해서 두 개의 값이나 변수가 필요한 연산자를 의미한다.

산술 연산자	설명
+	왼쪽 피연산자의 값에 오른쪽 피연산자의 값을 더함.
-	왼쪽 피연산자의 값에서 오른쪽 피연산자의 값을 뺌.
*	왼쪽 피연산자의 값에 오른쪽 피연산자의 값을 곱함.
/	왼쪽 피연산자의 값을 오른쪽 피연산자의 값으로 나눔.
%	왼쪽 피연산자의 값을 오른쪽 피연산자의 값으로 나눈 후, 그 나머지를 반환함.

# JAVASCRIPT 문법 – 연산자

## 산술 연산자

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Operators</title>
</head>

<body>

  <h1>산술 연산자</h1>

  <script>
    var x = 10, y = 4;
    document.write(x + y + "<br>");
    document.write(x - y + "<br>");
    document.write(x * y + "<br>");
    document.write(x / y + "<br>");
    document.write(x % y);
  </script>

</body>
</html>
```

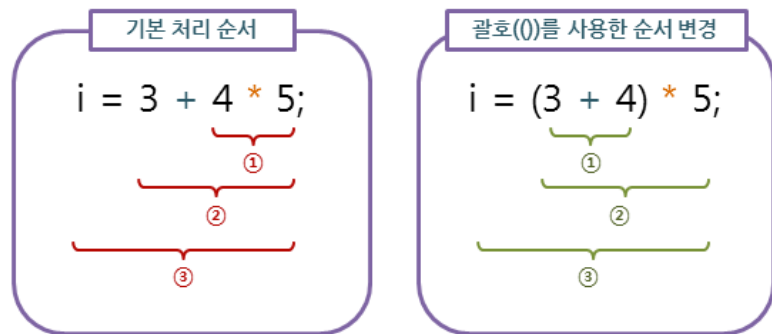
## 산술 연산자

14  
6  
40  
2.5  
2

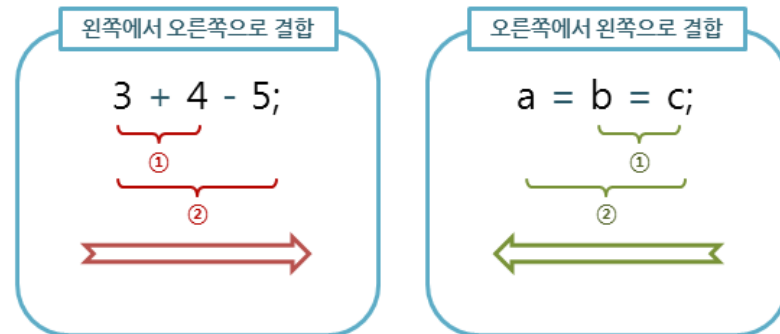
# JAVASCRIPT 문법 – 연산자

## 연산자의 우선순위(operator precedence)와 결합 방향(associativity)

다음 그림은 가장 높은 우선순위를 가지고 있는 괄호(()) 연산자를 사용하여 연산자의 처리 순서를 변경하는 것을 보여줍니다.



연산자의 결합 방향은 수식 내에 우선순위가 같은 연산자가 둘 이상 있을 때, 먼저 어느 연산을 수행할 것인가를 결정합니다.



연산자의 우선순위는 수식 내에 여러 연산자가 함께 등장할 때, 어느 연산자가 먼저 처리될 것인가를 결정한다.

# JAVASCRIPT 문법 – 연산자

## 자바스크립트 연산자의 우선순위표

우선순위	연산자	설명	결합 방향
1	()	묶임(괄호)	-
2	.	멤버 접근	왼쪽에서 오른쪽으로
	new	인수 있는 객체 생성	-
3	()	함수 호출	왼쪽에서 오른쪽으로
	new	인수 없는 객체 생성	오른쪽에서 왼쪽으로
4	++	후위 증가 연산자	-
	--	후위 감소 연산자	-
5	!	논리 NOT 연산자	오른쪽에서 왼쪽으로
	~	비트 NOT 연산자	오른쪽에서 왼쪽으로
	+	양의 부호 (단항 연산자)	오른쪽에서 왼쪽으로

우선순위	연산자	설명	결합 방향
	-	음의 부호 (단항 연산자)	오른쪽에서 왼쪽으로
	++	전위 증가 연산자	오른쪽에서 왼쪽으로
	--	전위 감소 연산자	오른쪽에서 왼쪽으로
	typeof	타입 반환	오른쪽에서 왼쪽으로
	void	undefined 반환	오른쪽에서 왼쪽으로
	delete	프로퍼티의 제거	오른쪽에서 왼쪽으로
6	**	거듭제곱 연산자	오른쪽에서 왼쪽으로
	*	곱셈 연산자	왼쪽에서 오른쪽으로
	/	나눗셈 연산자	왼쪽에서 오른쪽으로
	%	나머지 연산자	왼쪽에서 오른쪽으로

# JAVASCRIPT 문법 – 연산자

## 자바스크립트 연산자의 우선순위표

우선순위	연산자	설명	결합 방향	
7	+	덧셈 연산자 (이항 연산자)	왼쪽에서 오른쪽으로	21
	-	빼셈 연산자 (이항 연산자)	왼쪽에서 오른쪽으로	22
8	<<	비트 왼쪽 시프트 연산자	왼쪽에서 오른쪽으로	23
	>>	부호 비트를 확장하면서 비트 오른쪽 시프트	왼쪽에서 오른쪽으로	24
	>>>	부호 비트를 확장하지 않고 비트 오른쪽 시프트	왼쪽에서 오른쪽으로	25
9	<	관계 연산자 (보다 작은)	왼쪽에서 오른쪽으로	26
	<=	관계 연산자 (보다 작거나 같은)	왼쪽에서 오른쪽으로	27
	>	관계 연산자 (보다 큰)	왼쪽에서 오른쪽으로	28
	>=	관계 연산자 (보다 크거나 같은)	왼쪽에서 오른쪽으로	29
	instanceof	인스턴스 여부 판단	왼쪽에서 오른쪽으로	30

우선순위	연산자	설명	결합 방향	
10	==	동등 연산자	왼쪽에서 오른쪽으로	31
	===	일치 연산자	왼쪽에서 오른쪽으로	32
	!=	부등 연산자	왼쪽에서 오른쪽으로	33
	!==	불일치 연산자	왼쪽에서 오른쪽으로	34
11	&	비트 AND 연산자	왼쪽에서 오른쪽으로	35
12	^	비트 XOR 연산자	왼쪽에서 오른쪽으로	36
13		비트 OR 연산자	왼쪽에서 오른쪽으로	37
14	&&	논리 AND 연산자	왼쪽에서 오른쪽으로	38
15		논리 OR 연산자	왼쪽에서 오른쪽으로	39
16	?:	삼항 연산자	오른쪽에서 왼쪽으로	40
17	=	대입 연산자 (=, +=, -=, *=, /=, %=, <=<, >>=, >>>=, &=, ^=,  =)	오른쪽에서 왼쪽으로	41
18	...	전개	-	42
19	,	쉼표 연산자	왼쪽에서 오른쪽으로	43

# JAVASCRIPT 문법 – 연산자

## 대입 연산자(assignment operator)

- 대입 연산자는 변수에 값을 대입할 때 사용하는 이항 연산자이며, 피연산자들의 결합 방향은 오른쪽에서 왼쪽이다.
- 또한, 앞서 살펴본 산술 연산자와 결합한 다양한 복합 대입 연산자가 존재한다.

대입 연산자	설명
=	왼쪽 피연산자에 오른쪽 피연산자의 값을 대입함.
+=	왼쪽 피연산자의 값에 오른쪽 피연산자의 값을 더한 후, 그 결과 값을 왼쪽 피연산자에 대입함.
-=	왼쪽 피연산자의 값에서 오른쪽 피연산자의 값을 뺀 후, 그 결과 값을 왼쪽 피연산자에 대입함.
*=	왼쪽 피연산자의 값에 오른쪽 피연산자의 값을 곱한 후, 그 결과 값을 왼쪽 피연산자에 대입함.
/=	왼쪽 피연산자의 값을 오른쪽 피연산자의 값으로 나눈 후, 그 결과 값을 왼쪽 피연산자에 대입함.
%=	왼쪽 피연산자의 값을 오른쪽 피연산자의 값으로 나눈 후, 그 나머지를 왼쪽 피연산자에 대입함.

# JAVASCRIPT 문법 – 연산자

## 대입 연산자(assignment operator)

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Operators</title>
</head>

<body>

  <h1>대입 연산자</h1>

  <script>
    var x = 10, y = 10, z = 10;
    x = x - 5;
    document.write(x + "<br>");
    y -= 5; // y = y - 5 와 같은 표현임.
    document.write(y + "<br>");
    z -= 5; // z = -5 와 같은 표현임.
    document.write(z);
  </script>

</body>
</html>
```

### 대입 연산자

5  
5  
-5



# JAVASCRIPT 문법 – 연산자

## 증감 연산자(increment and decrement operator)

- 증감 연산자는 피 연산자를 1씩 증가 혹은 감소시킬 때 사용하는 연산자이다.
- 이 연산자는 피 연산자가 단 하나뿐인 단항연산자이다.
- 증감 연산자는 해당 연산자가 피 연산자의 어느 쪽에 위치하는가에 따라 연산의 순서 및 결과가 달라진다.

증감 연산자	설명
++X	먼저 피연산자의 값을 1 증가시킨 후에 해당 연산을 진행함.
X++	먼저 해당 연산을 수행하고 나서, 피연산자의 값을 1 증가시킴.
--X	먼저 피연산자의 값을 1 감소시킨 후에 해당 연산을 진행함.
X--	먼저 해당 연산을 수행하고 나서, 피연산자의 값을 1 감소시킴.

# JAVASCRIPT 문법 – 연산자

## 증감 연산자(increment and decrement operator)

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Operators</title>
</head>

<body>

  <h1>증감 연산자</h1>

  <script>
    var x = 10, y = 10;
    document.write(++x - 3) + "<br>"); // x의 값을 우선 1 증가시킨 후에 3을 뺌.
    document.write(x + "<br>");         // 11
    document.write(y++ - 3) + "<br>"); // 먼저 y에서 3을 뺀 후에 y의 값을 1 증가시킴.
    document.write(y);               // 11
  </script>

</body>
</html>
```

## 증감 연산자

8  
11  
7  
11

# JAVASCRIPT 문법 – 연산자

## 증감 연산자의 연산 순서

- 증감 연산자는 피 연산자의 어느 쪽에 위치하는가에 따라 연산의 순서가 달라진다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Operators</title>
</head>

<body>

  <h1>증감 연산자의 연산 순서</h1>

  <script>
    var x = 10;
    var y = x-- + 5 + --x; // 8
                          // 23

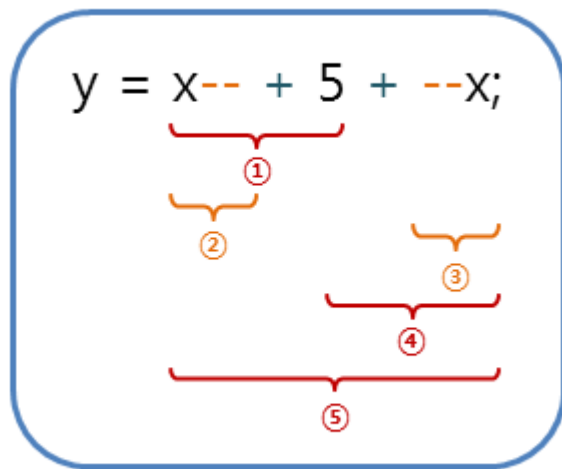
    document.write(x + "<br>");
    document.write(y);
  </script>
</body>
</html>
```

## 증감 연산자의 연산 순서

8  
23

# JAVASCRIPT 문법 – 연산자

## 증감 연산자의 연산 순서



- ① : 첫 번째 감소 연산자(decrement operator)는 피 연산자의 뒤쪽에 위치하므로, 덧셈 연산이 먼저 수행된다.
- ② : 덧셈 연산이 수행된 후에 감소 연산이 수행된다. (x의 값 : 9)
- ③ : 두 번째 감소 연산자는 피 연산자의 앞쪽에 위치하므로, 덧셈 연산보다 먼저 수행된다. (x의 값 : 8)
- ④ : 감소 연산이 수행된 후에 덧셈 연산이 수행된다.
- ⑤ : 마지막으로 변수 `y`에 결과 값의 대입 연산이 수행된다. (y의 값 : 23)

# JAVASCRIPT 문법 – 연산자

## 비교 연산자(comparison operator)

- 비교 연산자는 피 연산자 사이의 상대적인 크기를 판단하여, 참(true)과 거짓(false)을 반환한다.
- 비교 연산자는 모두 두 개의 피 연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽이다.
- 자바스크립트에서 비교 연산자는 피 연산자의 타입에 따라 두 가지 기준으로 비교를 진행한다.
  - 1. 피 연산자가 둘 다 숫자 면, 해당 숫자를 서로 비교한다.
  - 2. 피 연산자가 둘 다 문자열이면, 문자열의 첫 번째 문자부터 알파벳 순서대로 비교한다.

비교 연산자	설명
==	왼쪽 피연산자와 오른쪽 피연산자의 값이 같으면 참을 반환함.
===	왼쪽 피연산자와 오른쪽 피연산자의 값이 같고, 같은 타입이면 참을 반환함.
!=	왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않으면 참을 반환함.
!==	왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않거나, 타입이 다르면 참을 반환함.
>	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크면 참을 반환함.
>=	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크거나 같으면 참을 반환함.
<	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작으면 참을 반환함.
<=	왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작거나 같으면 참을 반환함.

# JAVASCRIPT 문법 – 연산자

## 비교 연산자(comparison operator)

- 위의 세 번째 연산에서 변수 x의 값은 숫자이고, 변수 a의 값은 문자열입니다.
- 비교 연산자 < 는 x의 값이 a의 값보다 작을 때만 참을 반환하고, 나머지 경우에는 전부 거짓을 반환하는 연산자입니다.
- 따라서 타입이 서로 달라 비교할 수 없는 경우에는 참의 조건을 만족하게 하지 못하므로, 언제나 거짓(false)만을 반환하게 됩니다.

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title> JavaScript Operators</title>
</head>

<body>

  <h1>비교 연산자</h1>

  <script>
    var x = 3, y = 5;
    var a = "abc", b = "bcd";
    document.write((x > y) + "<br>"); // y의 값이 x의 값보다 크므로 false
    document.write((a <= b) + "<br>"); // 알파벳 순서상 'a'가 'b'보다 먼저 나오므로 'a'가 'b'보다 작음.
    document.write(x < a);           // x의 값은 숫자이고 a의 값은 문자열이므로 비교할 수 없음.
  </script>

</body>

</html>
```

### 비교 연산자

false  
true  
false

# JAVASCRIPT 문법 – 연산자

## 동등 연산자와 일치 연산자

- 동등 연산자(==, equal)와 일치 연산자(===, strict equal)는 모두 두 개의 피 연산자가 서로 같은지를 비교해 준다.
- 두 연산자 모두 피 연산자의 타입을 가리지는 않지만, 그 값을 정의하는 기준이 조금 다르다.
- 동등 연산자(==)는 두 피 연산자의 값이 서로 같으면 참(true)을 반환한다.
- 이때 두 피 연산자의 타입이 서로 다르면, 비교를 위해 강제로 타입을 같게 변환한다.
- 하지만 일치 연산자(===)는 타입의 변환 없이 두 피 연산자의 값이 같고, 타입도 같아야만 참(true)을 반환한다.

### 예제

```
var x = 3, y = '3', z = 3;  
document.write((x == y) + "<br>"); // x와 y의 타입이 서로 다르므로 타입을 서로 같게 한 후 비교를 하므로 true  
document.write((x === y) + "<br>"); // x와 y의 타입이 서로 다르므로 false  
document.write(x === z);          // x와 z은 값과 타입이 모두 같으므로 true
```

# JAVASCRIPT 문법 – 연산자

## 논리 연산자(logical operator)

- 논리 연산자는 주어진 논리식을 판단하여, 참(true)과 거짓(false)을 반환합니다.
- && 연산자와 || 연산자는 두 개의 피연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽입니다.
- ! 연산자는 피연산자가 단 하나뿐인 단항 연산자이며, 피연산자의 결합 방향은 오른쪽에서 왼쪽입니다.

논리 연산자	설명
&&	논리식이 모두 참이면 참을 반환함. (논리 AND 연산)
	논리식 중에서 하나라도 참이면 참을 반환함. (논리 OR 연산)
!	논리식의 결과가 참이면 거짓을, 거짓이면 참을 반환함. (논리 NOT 연산)



# JAVASCRIPT 문법 – 연산자

논리 연산자(logical operator) – 진리 표(truth table)

A	B	A && B	A    B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

예제

```
var x = true, y = false;  
document.write((x && y) + "<br>"); // false (논리 AND 연산)  
document.write((x || y) + "<br>"); // true (논리 OR 연산)  
document.write(!x); // false (논리 NOT 연산)
```

# JAVASCRIPT 문법 – 연산자

## 비트 연산자(bitwise operator)

- 비트 연산자는 논리 연산자와 비슷하지만, 비트 (bit) 단위로 논리 연산을 수행한다.
- 또한, 비트 단위로 전체 비트를 왼쪽이나 오른쪽으로 이동시킬 때도 사용한다.

비트 연산자	설명
&	대응되는 비트가 모두 1이면 1을 반환함. (비트 AND 연산)
	대응되는 비트 중에서 하나라도 1이면 1을 반환함. (비트 OR 연산)
^	대응되는 비트가 서로 다르면 1을 반환함. (비트 XOR 연산)
~	비트를 1이면 0으로, 0이면 1로 반전시킴. (비트 NOT 연산)
<<	지정한 수만큼 비트를 전부 왼쪽으로 이동시킴. (left shift 연산)
>>	부호를 유지하면서 지정한 수만큼 비트를 전부 오른쪽으로 이동시킴. (right shift 연산)
>>>	지정한 수만큼 비트를 전부 오른쪽으로 이동시키며, 새로운 비트는 전부 0이 됨.

# JAVASCRIPT 문법 – 연산자

## 비트 연산자(bitwise operator)

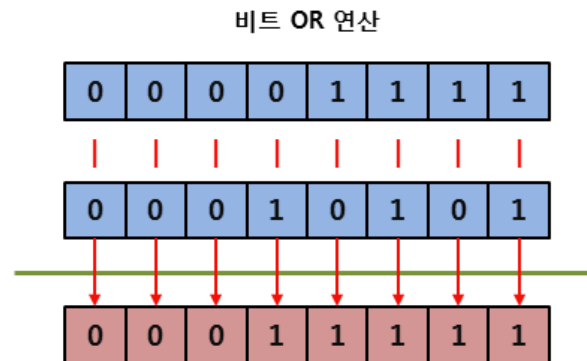
다음 그림은 비트 AND 연산자(&)의 동작을 나타냅니다.

이처럼 비트 AND 연산자는 대응되는 두 비트가 모두 1일 때만 1을 반환하며, 다른 경우는 모두 0을 반환합니다.



다음 그림은 비트 OR 연산자(|)의 동작을 나타냅니다.

이처럼 비트 OR 연산자는 대응되는 두 비트 중 하나라도 1이면 1을 반환하며, 두 비트가 모두 0일 때만 0을 반환합니다.



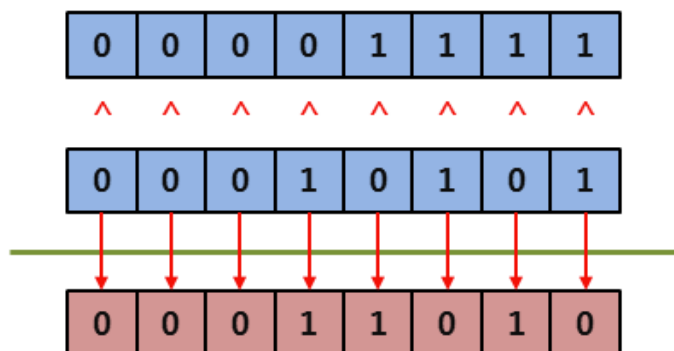
# JAVASCRIPT 문법 – 연산자

## 비트 연산자(bitwise operator)

다음 그림은 비트 XOR 연산자(^)의 동작을 나타냅니다.

이처럼 비트 XOR 연산자는 대응되는 두 비트가 서로 다르면 1을 반환하고, 서로 같으면 0을 반환합니다.

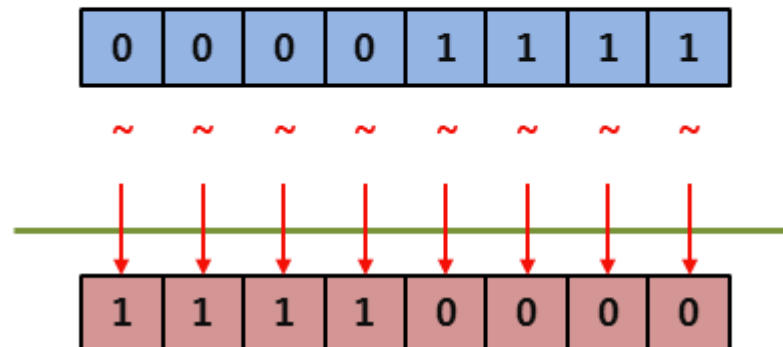
비트 XOR 연산



다음 그림은 비트 NOT 연산자(~)의 동작을 나타냅니다.

이처럼 비트 NOT 연산자는 해당 비트가 1이면 0을 반환하고, 0이면 1을 반환합니다.

비트 NOT 연산



# JAVASCRIPT 문법 – 연산자

## 비트 연산자(bitwise operator)

- 예제에서 첫 번째 연산은 비트를 전부 왼쪽으로 1비트씩 이동시키는 연산입니다.
- 따라서 그 결과 값은 처음 값에 2를 곱한 것과 같게 됩니다.
- 반대로 두 번째 연산은 비트를 전부 오른쪽으로 1비트씩 이동시키는 연산입니다.
- 따라서 그 결과 값은 처음 값에 2를 나눈 것과 같게 됩니다.

## 비트 연산자

30  
-4  
2147483644  
-16

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Operators</title>
</head>

<body>

  <h1>비트 연산자</h1>

  <script>
    var x = 15, y = -8, z = 15;
    document.write((x << 1) + "<br>"); // 곱하기 2
    document.write((y >> 1) + "<br>"); // 나누기 2
    document.write((y >>> 1) + "<br>"); // 부호 비트까지도 전부 오른쪽으로 이동됨.
    document.write(~z); // 1의 보수
  </script>

</body>

</html>
```

# JAVASCRIPT 문법 - 연산자

## 기타 연산자 - 문자열 결합 연산자

- 자바스크립트에서 덧셈(+) 연산자는 피 연산자의 타입에 따라 두 가지 다른 연산을 수행한다.
- 1. 피 연산자가 둘 다 숫자이면, 산술 연산인 덧셈을 수행한다.
- 2. 피 연산자가 하나라도 문자열이면, 문자열 결합을 수행한다.
- 피 연산자가 하나는 문자열이고 다른 하나는 문자열이 아닐 때, 자바스크립트는 문자열이 아닌 피 연산자를 자동으로 문자열로 변환한 후 문자열 결합을 수행한다.

### 예제

```
var x = 3 + 4;           // 피연산자가 둘 다 숫자이면 덧셈 연산을 수행함.  
var y = "좋은 " + "하루 되세요!" // 피연산자가 둘 다 문자열이면 문자열 결합 연산을 수행함.  
var z = 12 + "월"       // 피연산자가 하나라도 문자열이면 문자열 결합 연산을 수행함.
```

# JAVASCRIPT 문법 – 연산자

## 기타 연산자 - 삼항 연산자(ternary operator)

- 삼항 연산자는 유일하게 피 연산자를 세 개나 가지는 조건 연산자이다.
- 삼항 연산자의 문법은 다음과 같다.
  - 표현식 ? 반환값1 : 반환값2
- 물음표(?) 앞의 표현식에 따라 결과 값이 참이면 반환 값1을 반환하고, 결과 값이 거짓이면 반환 값2를 반환한다.
- 삼항 연산자는 짧은 if / else 문 대신 사용할 수 있으며, 코드를 간결하게 만들어 준다.

### 예제

```
var x = 3, y = 5;  
var result = (x > y) ? x : y // x가 더 크면 x를, 그렇지 않으면 y를 반환함.  
document.write("둘 중에 더 큰 수는 " + result + "입니다.");
```

# JAVASCRIPT 문법 - 연산자

## 기타 연산자 - 쉼표 연산자

- 쉼표 연산자를 for 문에서 사용하면, 루프마다 여러 변수를 동시에 갱신할 수 있다.

### 예제

```
// 루프마다 i의 값은 1씩 증가하고, 동시에 j의 값은 1씩 감소함.  
for (var i = 0, j = 9; i <= j; i++, j--) {  
    document.write("i의 값은 " + i + "이고, j의 값은 " + j + "입니다.<br>");  
}
```



# JAVASCRIPT 문법 – 연산자

## 기타 연산자 - delete 연산자

- delete 연산자는 피 연산자인 객체, 객체의 프로퍼티(property) 또는 배열의 요소(element) 등을 삭제해 준다.
- 피 연산자가 성공적으로 삭제되었을 경우에는 참(true)을 반환하고, 삭제하지 못했을 경우에는 거짓(false)을 반환한다.
- 이 연산자는 피 연산자가 단 하나뿐인 단항 연산자이며, 피 연산자의 결합 방향은 오른쪽에서 왼쪽이다.

### 예제

```
var arr = [1, 2, 3];           // 배열 생성
delete arr[2];                 // 배열의 원소 중 인덱스가 2인 요소를 삭제함.
document.write(arr + "<br>"); // [1, 2, ]
// 배열에 빈자리가 생긴 것으로 undefined 값으로 직접 설정된 것은 아님.
document.write(arr[2] + "<br>");
// 배열의 요소를 삭제하는 것이지 배열의 길이까지 줄이는 것은 아님.
document.write(arr.length);
```

# JAVASCRIPT 문법 – 연산자

## 기타 연산자 - typeof 연산자

- typeof 연산자는 피 연산자의 타입을 반환합니다.
- 이 연산자는 피 연산자가 단 하나뿐인 단항 연산자이며, 피 연산자의 결합 방향은 오른쪽에서 왼쪽입니다.

### 예제

```
typeof "문자열" // string
typeof 10       // number
typeof NaN      // number
typeof false    // boolean
typeof undefined // undefined
typeof new Date() // object
typeof null     // object
```

값	typeof 연산자의 결과값
숫자, NaN	"number"
문자열	"string"
true, false	"boolean"
null	"object"
undefined	"undefined"
함수	"function"
함수가 아닌 객체	"object"

# JAVASCRIPT 문법 – 연산자

## 기타 연산자 - typeof 연산자

```
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <title>JavaScript Operators</title>
</head>

<body>

  <h1>typeof 연산자</h1>

  <script>
    document.write((typeof "문자열") + "<br>");
    document.write((typeof 10) + "<br>");
    document.write((typeof NaN) + "<br>");
    document.write((typeof false) + "<br>");
    document.write((typeof undefined) + "<br>");
    document.write((typeof new Date()) + "<br>");
    document.write(typeof null);
  </script>

</body>

</html>
```

## typeof 연산자

string  
number  
number  
boolean  
undefined  
object  
object

# JAVASCRIPT 문법 – 연산자

## 기타 연산자 - instanceof 연산자

- instanceof 연산자는 피 연산자인 객체가 특정 객체의 인스턴스인지 아닌지를 확인해 준다.
- 피 연산자가 특정 객체의 인스턴스이면 참(true)을 반환하고, 특정 객체의 인스턴스가 아니면 거짓(false)을 반환한다.
- 이 연산자는 두 개의 피 연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽이다.

### 예제

```
var str = new String("이것은 문자열입니다.");

str instanceof Object; // true
str instanceof String; // true
str instanceof Array;  // false
str instanceof Number; // false
str instanceof Boolean; // false
```

# JAVASCRIPT 문법 – 연산자

## 기타 연산자 - void 연산자

- void 연산자는 피 연산자로 어떤 타입의 값이 오던지 상관없이 언제나 undefined 값을 반환한다.
- 이 연산자는 피 연산자가 단 하나뿐인 단항 연산자이며, 피 연산자의 결합 방향은 오른쪽에서 왼쪽이다.

### 예제

```
<a href="javascript:void(0)">이 링크는 동작하지 않습니다.</a>
```

```
<a href="javascript:void(document.body.style.backgroundColor='yellow')">  
  이 링크도 동작하지 않지만, HTML 문서의 배경색을 바꿔줍니다.  
</a>
```

# JAVASCRIPT 문법

## 제어문

- 프로그램의 순차적인 흐름을 제어해야 할 때 사용하는 실행 문을 제어 문이라고 한다.
- 이러한 제어 문에는 조건 문, 반복 문 등이 포함된다.

# JAVASCRIPT 문법 - 제어문

## 조건문(conditional statements)

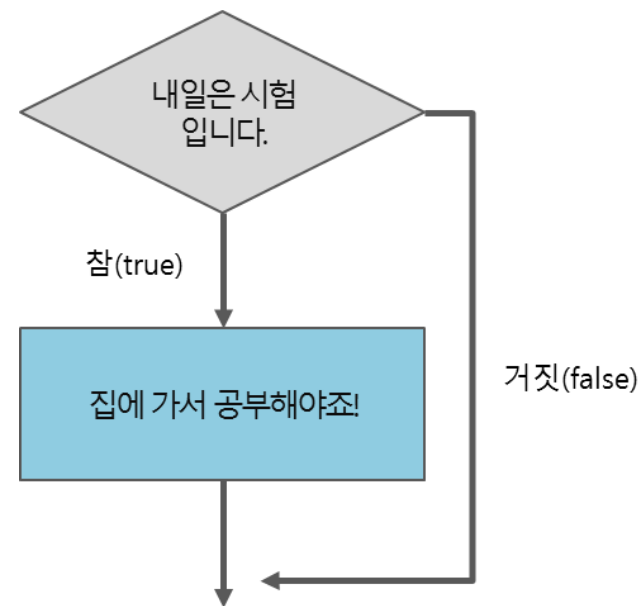
- 조건 문이란 프로그램 내에서 주어진 표현식의 결과에 따라 별도의 명령을 수행하도록 제어하는 실행 문이다.
- 조건 문 중에서 가장 기본이 되는 실행 문은 if 문이다.
- 자바스크립트에서 사용할 수 있는 조건문의 형태는 다음과 같다.
- 1. if 문
- 2. if / else 문
- 3. if / else if / else 문
- 4. switch 문

# JAVASCRIPT 문법 - 제어문

## 조건문(conditional statements)

### ■ if 문

- if 문은 표현식의 결과가 참(true)이면 주어진 실행 문을 실행하며, 거짓(false)이면 아무것도 실행하지 않는다.
- if 문을 순서도로 표현하면 다음 그림과 같이 표현할 수 있다.





# JAVASCRIPT 문법 - 제어문

## 조건문(conditional statements)

- if 문의 문법
  - if (표현식) {
  - 표현식의 결과가 참일 때 실행하고자 하는 실행 문;
  - }

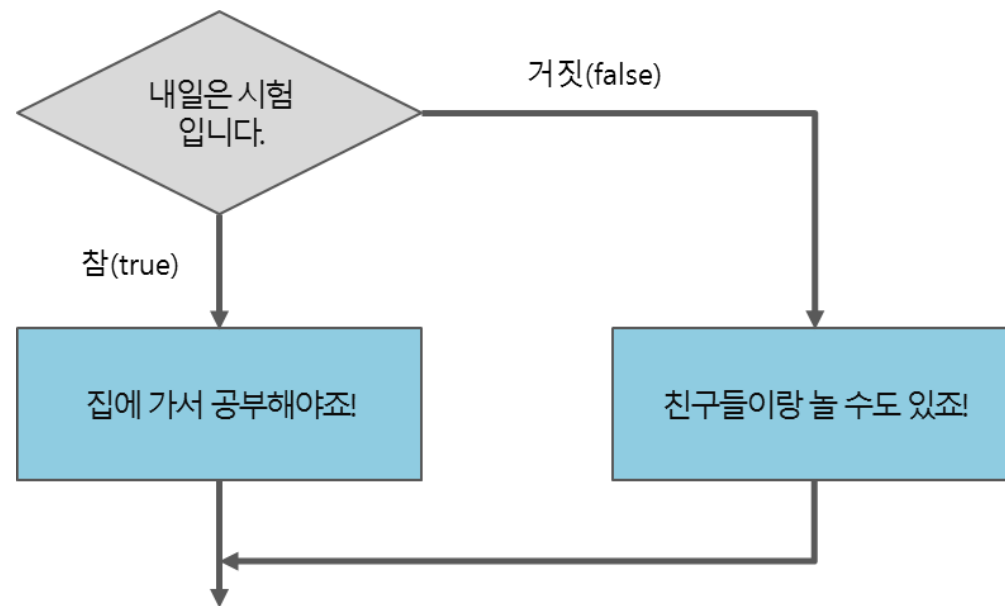
### 예제

```
var x = 10, y = 20;
if (x == y) {
    document.write("x와 y는 같습니다.");
}
if (x < y) {
    document.write("x가 y보다 작습니다.");
}
if (x > y) // 실행될 실행문이 한 줄뿐이라면 중괄호({})를 생략할 수 있음.
    document.write("x가 y보다 큼니다.");
```

# JAVASCRIPT 문법 - 제어문

## 조건문(conditional statements)

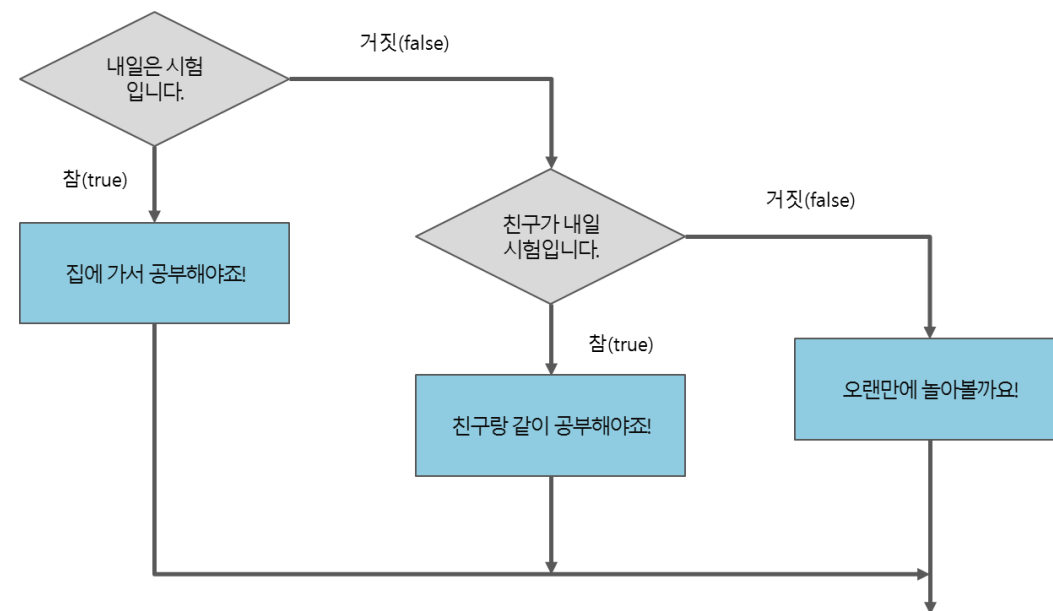
- If { .. } else { .. } 문
  - if (condition) {
  - // block of code to be executed if the condition is true
  - } else {
  - // block of code to be executed if the condition is false
  - }



# JAVASCRIPT 문법 - 제어문

## 조건문(conditional statements)

- If { .. } else { .. } 문
  - if (condition1) {
  - // block of code to be executed if condition1 is true
  - } else if (condition2) {
  - // block of code to be executed if the condition1 is false and condition2 is true
  - } else {
  - // block of code to be executed if the condition1 is false and condition2 is false
  - }



# JAVASCRIPT 문법 - 제어문

## 조건문(conditional statements)

- 삼항 연산자에 의한 조건문
  - 자바스크립트에서는 간단한 if / else 문을 삼항 연산자를 이용하여 간단히 표현할 수 있다.
  - variablename = (condition) ? value1:value2

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Comparison</h2>

<p>Input your age and click the button:</p>

<input id="age" value="18" />

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  let age = document.getElementById("age").value;
  let voteable = (age < 18) ? "Too young":"Old enough";
  document.getElementById("demo").innerHTML = voteable + " to vote.";
}
</script>

</body>
</html>
```

# JAVASCRIPT 문법 - 제어문

## switch 문

- switch 문은 if / else 문과 마찬가지로 주어진 조건 값에 따라 프로그램이 다른 명령을 수행하도록 하는 조건 문입니다.
- 이러한 switch 문은 if / else 문보다 가독 성 측면에서 더 좋다.
  - 스위치 표현식은 한 번 평가된다.
  - 표현식의 값은 각각의 case 값과 비교된다.
  - 일치하는 항목이 있으면 연결된 코드 블록이 실행됩니다.
  - 일치하는 항목이 없으면 default 코드 블록이 실행됩니다.

- switch 문의 문법
  - `switch(expression) {`
  - `case x:`
  - `// code block`
  - `break;`
  - `case y:`
  - `// code block`
  - `break;`
  - `default:`
  - `// code block`
  - `}`

# JAVASCRIPT 문법 - 제어문

## switch 문

- switch 문은 if / else 문과 마찬가지로 주어진 조건 값에 따라 프로그램이 다른 명령을 수행하도록 하는 조건 문입니다.
- 이러한 switch 문은 if / else 문보다 가독 성 측면에서 더 좋다.
  - 스위치 표현식은 한 번 평가된다.
  - 표현식의 값은 각각의 case 값과 비교된다.
  - 일치하는 항목이 있으면 연결된 코드 블록이 실행됩니다.
  - 일치하는 항목이 없으면 default 코드 블록이 실행됩니다.

- switch 문의 문법
  - `switch(expression) {`
  - `case x:`
  - `// code block`
  - `break;`
  - `case y:`
  - `// code block`
  - `break;`
  - `default:`
  - `// code block`
  - `}`

# JAVASCRIPT 문법 - 제어문

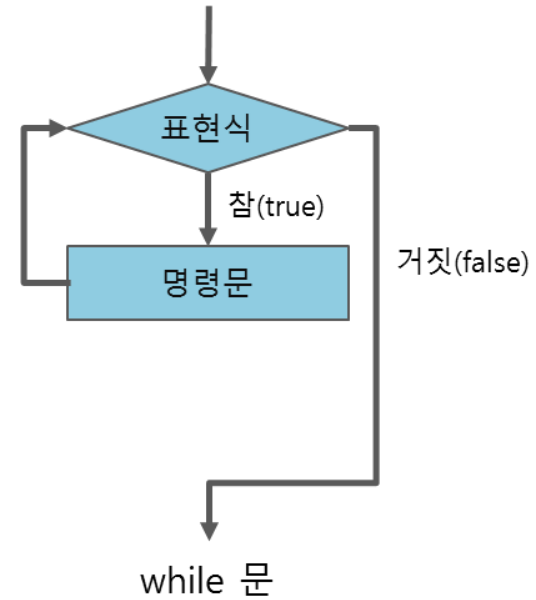
## 반복문(iteration statements)

- 반복 문이란 프로그램 내에서 똑같은 명령을 일정 횟수만큼 반복하여 수행하도록 제어하는 실행 문이다.
- 프로그램이 처리하는 대부분의 코드는 반복적인 형태가 많으므로, 가장 많이 사용되는 실행 문 중 하나이다.
- 자바스크립트에서 사용할 수 있는 반복문의 형태는 다음과 같다.
  - 1. while 문
  - 2. do / while 문
  - 3. for 문
  - 4. for / in 문
  - 5. for / of 문

# JAVASCRIPT 문법 - 제어문

## while 문

- while 문은 특정 조건을 만족할 때까지 계속해서 주어진 실행문을 반복 실행한다.
- While 문을 순서도로 표현하면 다음 그림과 같다.
  - while (condition) {
  - // code block to be executed
  - }





# JAVASCRIPT 문법 - 제어문

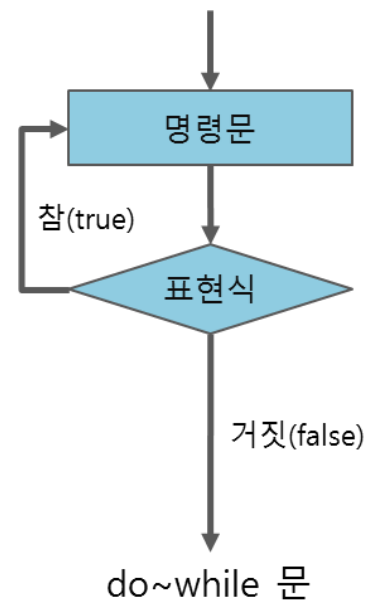
## while 문

- while 문은 우선 표현식이 참(true)인지를 판단하여 참이면 내부의 실행 문을 실행한다.
- 내부의 실행 문을 전부 실행하고 나면, 다시 표현식으로 돌아와 또 한 번 표현식이 참인지를 판단하게 된다.
- 이렇게 표현식의 검사를 통해 반복해서 실행되는 반복 문을 루프(loop)라고 합니다.
- while 문 내부에 표현식의 결과를 변경하는 실행 문이 존재하지 않을 경우 프로그램은 루프를 영원히 반복하게 된다.
- 이것을 무한 루프(infinite loop)에 빠졌다고 하며, 무한 루프에 빠진 프로그램은 영원히 종료되지 않는다.
- 무한 루프는 특별히 의도한 경우가 아니라면 반드시 피해야 하는 상황이다.
- 따라서 while 문을 작성할 때는 표현식의 결과가 어느 순간에는 거짓(false)을 갖도록 표현 식을 변경하는 실행 문을 반드시 포함해야 한다.

# JAVASCRIPT 문법 - 제어문

## do / while 문

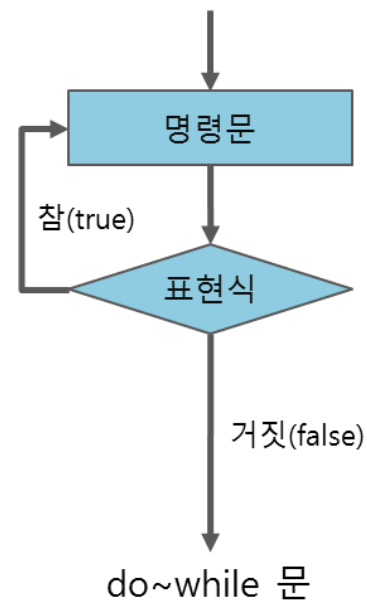
- while 문은 루프에 진입하기 전에 먼저 표현식부터 검사한다.
- 하지만 do / while 문은 먼저 루프를 한 번 실행한 후에 표현식을 검사한다.
- 즉, do / while 문은 표현식의 결과와 상관없이 무조건 한 번은 루프를 실행한다.
  - do {
  - // code block to be executed
  - }
  - while (condition);



# JAVASCRIPT 문법 - 제어문

## do / while 문

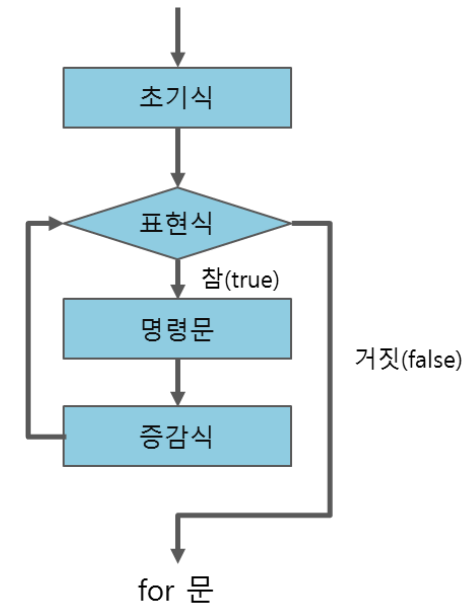
- while 문은 루프에 진입하기 전에 먼저 표현식부터 검사한다.
- 하지만 do / while 문은 먼저 루프를 한 번 실행한 후에 표현식을 검사한다.
- 즉, do / while 문은 표현식의 결과와 상관없이 무조건 한 번은 루프를 실행한다.
  - do {
  - // code block to be executed
  - }
  - while (condition);



# JAVASCRIPT 문법 - 제어문

## for 문

- for 문은 while 문과는 달리 자체적으로 초기 식, 표현식, 증감 식을 모두 포함하고 있는 반복 문이다.
- 따라서 while 문보다는 좀 더 간결하게 반복 문을 표현할 수 있다.



# JAVASCRIPT 문법 - 제어문

## for 문

- for (statement 1; statement 2; statement 3) {
- // code block to be executed
- }
- statement1은 초기 식을 정의 한다.
- statement2는 조건을 정의한다.
- statement3은 증감 식을 정의 한다.

### 예제

```
for (var i = 1; i < 10; i++) {  
    document.write(i + "<br>");  
}
```

# JAVASCRIPT 문법 - 제어문

## For In Loop

- JavaScript for in 문은 객체의 속성을 반복한다.
  - for (key in object) {
  - // code block to be executed
  - }

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
    text += person[x];
}
```

# JAVASCRIPT 문법

## Array(배열)

- 자바스크립트에서 배열(array)은 이름과 인덱스로 참조되는 정렬된 값의 집합으로 정의된다.
- 배열을 구성하는 각각의 값을 배열 요소(element)라고 하며, 배열에서의 위치를 가리키는 숫자를 인덱스(index)라고 한다.
- 자바스크립트에서 배열의 특징은 다음과 같습니다.
  - 1. 배열 요소의 타입이 고정되어 있지 않으므로, 같은 배열에 있는 배열 요소끼리의 타입이 서로 다를 수도 있습니다.
  - 2. 배열 요소의 인덱스가 연속적이지 않아도 되며, 따라서 특정 배열 요소가 비어 있을 수도 있습니다.
  - 3. 자바스크립트에서 배열은 Array 객체로 다뤄집니다.
- 배열의 선언은 [] 나 new Array() 로 생성하고 크기의 제약이 없으며 하나의 배열에 서로 다른 타입의 변수가 들어갈 수 있다.

```
var emptyArray = [];  
var numbers1 = [10, 30, 50, 70, 90];  
var numbers2 = new Array(20, 40, 60, 80, 100);  
var mixedArr = ['a', 1, 'b', 2, new Date(), "today"];  
  
console.log(numbers2[2]);  
console.log(mixedArr[4]);
```

# JAVASCRIPT 문법

## Array(배열)

- 아직 자바스크립트 객체를 배우기 전이지만 배열에 자바스크립트 객체를 넣을 수 있습니다. 이를 이용하면 JSON 데이터의 집합을 만들 수 있으며 서버와의 데이터 교환에 사용할 수 있는 구조가 된다.

```
var objArr = [
  {
    "id" : 20192010,
    "name" : "홍길동",
    "dept" : "컴퓨터공학과"
  },
  {
    "id" : 20192011,
    "name" : "김사랑",
    "dept" : "경영학과"
  },
  {
    "id" : 20192012,
    "name" : "강동수",
    "dept" : "전자공학과"
  }
];

console.log(objArr[2].name + "," + objArr[2].dept);
```



# JAVASCRIPT 문법

## Array(배열)의 주요 메서드

### ■ 데이터 변경

- push(): 배열의 끝에 값을 추가.
- pop(): 배열 마지막 값을 제거.
- shift(): 배열 데이터를 왼쪽으로 하나씩 밀어 맨 앞 값을 제거.
- splice(): 배열값 을 추가하거나 제거해서 반환.
- reverse(): 배열을 역순으로 재배치.
- sort(): 배열 데이터를 정렬.

### ■ 배열의 일부를 반환

- concat(): 두개의 배열을 합침.
- join(): 배열 데이터 사이에 원하는 문자열을 넣어 구분자로 사용.
- slice(): 배열의 일부를 지정해서 가져옴

# JAVASCRIPT 문법

## Array(배열)의 주요 메서드

- 데이터 순회
  - map(): 모든 배열 데이터 마다 반복 처리가 필요한 경우 사용.
  - filter(): 특정 조건을 만족하는 데이터만 처리할 경우 사용.
  - reduce(): 모든 데이터를 순화하면서 누적 연산이 필요한 경우 사용

# JAVASCRIPT 문법

## Map

- Map 은 `new Map()` 으로 선언 하고 `map.set()`을 이용해 키와 값을 추가 한다.
- 자바스크립트의 Map 은 Key:Value 쌍으로 구성된 Object와 비슷한 구조로 볼 수 있지만 다음과 같은 차이가 있다.
  - Object의 키는 문자열이며, Map의 키는 모든 값을 가질 수 있다.
  - Object는 크기를 수동으로 추적해야하지만, Map은 크기를 쉽게 얻을 수 있다.
  - Map은 삽입된 순서대로 반복된다.
  - 객체(Object)에는 prototype이 있어 Map에 기본 키들이 있다.
- 언제 Object 혹은 Map을 사용할까?
- 실행 시점까지 키를 알 수 없고, 모든 키가 동일한 type이며 모든 값들이 동일한 type일 경우에는 object를 대신해서 map을 사용하는 것이 좋다. 만일 각 개별 요소에 대해 적용해야 하는 logic 있다면 object를 사용하길 권장 한다.

# JAVASCRIPT 문법

## Map

- forEach 의 인자는 value, key, map object 이다.
- For-of 형태를 사용하는 경우 배열로 키(0), 값(1) 이 전달 된다.
- map.keys()로 키 값의 집합인 MapIterator 객체를 받아 for-of 를 이용한다.

```
const map = new Map();
map.set("2019101", "홍길동");
map.set("2019102", "김사랑");
map.set("2019103", "강동수");

console.log(map.get("2019101"));
map.delete("2019103");
console.log(map.has("2019103"));

map.forEach((value, key) => console.log(key+" ", "+value));

for(let item of map) {
  console.log(item[0]+" ", "+item[1]);
}

for(let [key, value] of map) {
  console.log(key+" ", "+value);
}

const keys = map.keys();
for(let key of keys) {
  console.log(map.get(key));
}
```

# JAVASCRIPT 문법

## Set

- Set 은 중복된 값을 허용하지 않고 입력된 순서에 따라 데이터를 저장하는 자료구조 이다. Set.add() 를 이용해 데이터를 추가할 수 있고 데이터 관리를 위한 메서드(method)가 제공 된다.

# JAVASCRIPT 문법

## Set

- 배열과 유사하지만 다음과 같은 차이가 있다.
- `indexOf()`를 사용하여 배열 내에 특정 요소가 존재하는지 확인하는 것은 느리다.
- 배열에선 해당 요소를 배열에서 잘라내야 하는 반면 Set은 삭제 기능을 제공한다.
- NaN은 배열에서 `indexOf`메서드로 찾을 수 없다.
- Set객체는 값의 유일성을 보장하기 때문에 직접 요소의 중복 성을 확인할 필요가 없다.

```
const set = new Set();
set.add("홍길동");
set.add("김사람");
set.add("강동수");

set.delete("강동수");
console.log(set.has("강동수"));

set.forEach((value) => console.log(value));

for(let item of set) {
  console.log(item);
}
```

# JAVASCRIPT 문법

## Set

- Set 은 배열로 배열은 Set 으로 변경이 가능하다.
- 배열에서 Set 으로 변환될 경우 중복된 값은 제거 된다.
- `let arr = Array.from(set);`
- `let newSet = new Set([1,2,3,4,3]);`

# JAVASCRIPT 문법

## 함수(Function)

- 함수의 개념은 대부분의 프로그램 언어에서 존재하지만 특히 자바스크립트는 함수를 변수에 할당할 수 있으며 인자로 함수를 전달하거나 콜백 함수(Callback function)를 구현하는 형태 등 다양한 함수 활용을 보여주고 있다.
- 이러한 특징들과 함께 최근의 함수형 프로그래밍(FP: Functional Programming) 개념도 자바스크립트에서 지원되고 있다.



# JAVASCRIPT 문법

## 함수(Function) 선언

- 기본적으로 function 키워드로 선언하며 인자(argument)를 가질 수 있다. JavaScript는 타입을 명시하지 않기 때문에 리턴 값의 유무와 상관없이 함수 선언 후에 리턴 데이터에 대한 선언은 없습니다. 필요시 함수 바디에서 return 문을 사용해 값을 반환하면 된다.
  - 인자의 매개변수 역시 타입없이 갯 수에 따라 변수 명을 나열.
  - 함수 안에 사용되는 변수는 block-scoped 인 let 이나 const 사용을 권장.
  - 함수 안에 또다른 함수를 정의할 수 있습니다.

```
function calc(a, b) {  
  let sum1 = a + b;  
  sum2 = a + b;  
  return sum1;  
}  
  
console.log(calc(20, 30)); // 50  
console.log(sum2); //50  
console.log(sum1); //Error - not defined
```

# JAVASCRIPT 문법

## 익명 함수(Anonymous Function)

- 익명 함수는 이름이 없는 함수로 변수에 함수를 구현하거나 함수를 return하는 형태가 가능하며 이를 사용하면 소위 Callback function의 구현이 가능해 진다.

```
var calc1 = function(a, b) {  
    return a+b;  
}  
  
function calc2(func) {  
    console.log(func(20,30));  
}  
  
console.log(calc1(20,30));  
calc2(calc1);
```

# JAVASCRIPT 문법

## 콜백 함수(Callback Function)

- JavaScript에서 Callback Function은 특히 event 처리에 많이 사용된다. 특정 UI에서 이벤트가 발생 되었을 때 처리될 코드를 익명 함수에 넣고 callback 되어 처리될 수 있도록 하는 형식이다.
- 다음은 제곱을 구하는 함수에서 Callback Function을 호출하는 예이다. Callback Function의 기능은 단순히 값을 콘솔에 출력하고 있다. setTimeout 함수는 일정시간 후에 동작을 정의한다. 여기서는 200ms 이후 callback을 호출 하며 parameter로 제곱 값을 전달하게 되어 있다.

```
function square(x, callback) {  
    setTimeout(callback, 200, x*x);  
}  
  
square(2, function(number) {  
    console.log(number);  
}));
```

# JAVASCRIPT 문법

## 자료구조의 값에 함수 사용

- 함수를 객체의 속성으로 사용할 수 있으며 Map의 Value 혹은 배열의 원소로도 사용할 수 있다.

```
// 배열원소로 함수 선언.  
let funcArr = [function() {console.log("v1")}, "v2"];  
  
// 맵원소로 함수 선언.  
let mapArr = new Map();  
mapArr.set("calcFunc", function(n1, n2) {return n1*n2});  
  
// 객체의 속성으로 함수 선언.  
let student = {  
  'id': 2019101,  
  'name': '홍길동',  
  'score': function() {  
    alert('A+');  
  }  
}  
  
console.log(funcArr[0]());  
console.log(mapArr.get("calcFunc")(20, 10));  
console.log(student.id);  
student.score();
```

# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

## 객체(Object)

- 객체지향에서 객체는 우리 눈에 보이는 구체화된 대상을 의미한다. 객체는 속성과 행위로 정의 할 수 있으며 객체를 정의할 수 있는 틀을 클래스(Class)라고 한다. 클래스는 멤버 변수와 메서드로 정의하게 되고 클래스로부터 객체를 생성(인스턴스)해서 사용하는 개념이다.
- ES6에서 클래스가 나오기 이전에는 Object 타입을 주로 사용 해서 객체를 정의 했다.
- Primitive 타입과 달리 참조 형이며 배열도 객체(object) 이다.
- 객체는 속성(property)의 모음이며 속성은 Key:Value 구조를 가진다.
- 속성 값이 함수인 경우 메서드(Method) 라고 표현 한다.

# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

## 객체 리터럴을 이용한 객체 생성(JSON)

- 최근 프로그램 개발에 많이 쓰이는 JSON은 포맷은 JavaScript Object Notation 으로 자바스크립트에서 사용하는 객체 표기법을 의미한다.
- 자바스크립트 객체를 단순화 해서 표현하면 Key:Value 형태의 일종의 맵 혹은 사전(Dictionary) 형태라고 할 수 있다. 앞에서 언급한 것처럼 객체의 값으로 함수를 사용할 수 있기 때문에 클래스와 유사한 형태가 되는 것이다. 객체 이니셜라이저(Object Initializer)라고도 한다.

```
let student = {  
  'id': 2019101,  
  'name': '홍길동',  
  'scores': [95, 80, 91, 85],  
  'getTotalScore': function() {  
    return this.scores.reduce((prev, curr) => prev + curr);  
  }  
}  
  
console.log(student.getTotalScore());
```

# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

## 생성자를 이용한 객체 생성

- 만일 클래스와 같이 구체적인 값을 가지지 않는 형태를 미리 선언하고 객체를 생성하려면 다음과 같이 생성자 함수를 정의하고 사용할 수 있다.

```
function Product(name, price) {  
  this.name = name;  
  this.price = price;  
  this.getInfo = function() {  
    return this.name + ", " + this.price;  
  }  
}  
  
let p1 = new Product("애플 아이폰", 1000000);  
let p2 = new Product();  
p2.name = "삼성 갤럭시"  
p2.price = 1100000;  
  
console.log(p1.getInfo());  
console.log(p2.getInfo());
```

# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

## 클래스(Class)

- ES6에서 부터 지원되기 시작했으며 좀 더 객체지향 개념에 가까운 객체 사용을 가능하게 한다. 앞에서 생성자 기반의 객체 선언을 살펴 보았는데 객체의 함수를 `this.getInfo` 와 같이 정의 하는 방법은 사실 좋은 방법이 아니다.
- 이유는 객체를 생성할 때마다 동일한 함수도 매번 생성되기 때문으로 이를 효과적으로 처리하려면 함수의 경우 `Product.prototype.getInfo = function () {}` 처럼 정의해 주어야 한다. 이경우 객체 선언 부와 함수 선언 부가 분리 되어 코드 관리가 불편하게 된다.
- 당연히 클래스를 사용하면 이런 문제점들을 해결할 수 있다.



# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

## 클래스(Class) 구조

- class 키워드로 선언 한다.
- 클래스 이름은 보통 대문자로 시작한다.

```
class 클래스명 {  
  constructor(인자..) {  
    this.var = 인자;  
    ...  
  }  
  
  메서드() {  
  }  
  
  get 메서드() {  
    return ..  
  }  
  
  set 메서드() {  
  }  
}
```

# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

## 클래스(Class) 구조

- 함수와 마찬가지로 클래스를 익명으로 선언할 수도 있고 변수에 할당할 수도 있다.
- new 연산을 통해 호출되는 것은 클래스 이름이 아니라 생성자 메서드이다.
- 클래스 바디에는 메서드 선언만 가능하고 다른 언어와 같이 필드(멤버) 선언은 불가능하다.
- 클래스 내부에서 선언한 속성은 클래스인스턴스를 가리키는 this에 바인딩한다.

```
class Product {  
  constructor(name, price) {  
    this.name = name;  
    this.price = price;  
  }  
  
  getInfo() {  
    return this.name + ", " + this.price;  
  }  
}  
  
let p1 = new Product("애플 아이폰", 1000000);  
let p2 = new Product();  
p2.name = "삼성 갤럭시"  
p2.price = 1100000;  
  
console.log(p1.getInfo());  
console.log(p2.getInfo());
```

# JAVASCRIPT 문법 - 객체(OBJECT)와 클래스(CLASS)

getter, setter

- 보통 클래스 필드에 접근할 때 직접 변수에 접근하는 형태가 아닌 getter, setter 메서드를 통한 접근방법을 사용한다. 클래스 입장에서는 외부에 값을 전달하거나 외부로부터 가지고 올 때 추가적인 조작이 가능하기 때문이고 내부 구조를 캡슐화 하는 데에도 유리하기 때문이다.
- 자바스크립트 클래스에서는 get, set 메서드를 정의할 수 있고 해당 메서드는 함수 호출이 아닌 클래스의 필드개념으로 사용하게 된다.

```
class Product {  
  constructor(name, price) {  
    this.name = name;  
    this.price = price;  
  }  
  
  get printName() {  
    return "제품명:" + this.name;  
  }  
  
  set downPrice(price) {  
    this.price = price;  
  }  
}  
  
let p = new Product("애플아이폰", 1000000);  
console.log(p.printName);  
  
p.downProce = 500000;  
console.log(p.price);
```

# JAVASCRIPT 문법

## 이벤트와 DOM

- 이번 강의에서는 JavaScript에서 사용자 이벤트를 처리하는 방법과 동적인 콘텐츠 핸들링을 위해 필요한 문서 객체 모델(DOM)이 무엇인지 배우게 된다.
- 이 강의를 통해 웹페이지에서 발생하는 이벤트를 자바스크립트 코드와 연계할 수 있고 DOM을 통해 콘텐츠 혹은 디자인을 동적으로 핸들링 할 수 있다.

# JAVASCRIPT 문법

## 이벤트

- 일반적으로 프로그래밍 언어에서 이벤트라고 하면 사용자의 동작 혹은 프로그램에서 발생하는 특정한 상황을 의미 한다. 이벤트가 발생하면 보통 사전에 정의된 특정 코드가 실행되고 그에 따라 기능이 동작하거나 화면이 변경되는 등의 변화가 발생 한다.
  - 웹 브라우저가 알려주는 HTML요소에 대한 사건의 발생을 의미.
  - 자바스크립트는 이벤트에 반응하여 특정 동작을 수행할 수 있음.
  - 입력 양식으로부터 사용자의 입력 값을 가져올 수 있음.
  - HTML이벤트속성은 자바스크립트 구문을 직접 실행 하거나 함수를 호출할 수 있다.

# JAVASCRIPT 문법

## HTML 이벤트

- HTML에서 발생하는 이벤트는 주로 마우스 및 입력양식에서 발생하며 문서의 로딩과 관련한 특별한 이벤트를 포함한다. 다음은 대표적인 이벤트 목록이다.

이벤트 명	설명
click	클릭시 발생
change	변동이 있을시 발생
focus	포커스를 얻었을때 발생
keydown	키를 눌렀을때 발생
keyup	키에서 손을 뗄때 발생
load	문서의 로드가 완료 되었을때 발생
unload	문서가 언로드 되었을때 발생
resize	윈도우 크기가 변경될 경우 발생
mouseover	마우스가 특정 객체 위로 올려졌을 시에 발생
mousedown	마우스를 클릭 했을때 발생
mouseout	마우스가 특정 객체 밖으로 나갔을 때 발생
mousemove	마우스가 움직였을 때 발생
mouseup	마우스에서 손을 뗄때 발생
select	option 태그 등에서 선택을 했을때 발생
submit	입력 양식이 제출 요청 될 때 발생

# JAVASCRIPT 문법

## 이벤트 핸들러(Event Handler)

- 이벤트 핸들러는 이벤트발생을 감지하고 처리할 코드를 수행하는 역할을 담당한다.
- HTML 태그의 속성으로 지정하거나 JavaScript에서 DOM엘리먼트의 속성에 콜백 함수를 정의하는 형식으로 사용할 수 있다. 이벤트 핸들러는 앞의 이벤트 이름의 앞에 on을 붙여 사용한다.

# JAVASCRIPT 문법

## HTML 속성으로 정의

- 이벤트를 감지하기 위한 HTML 태그의 속성에 이벤트 핸들러를 기술하는 방식이다.
- 보통 입력 양식(form)에 작성하게 된다.
- `<input type="button" value="Button" onclick="alert('버튼 클릭됨!!')"></input>`
  - onclick 속성에 자바스크립트 코드를 직접 작성하거나 이벤트 처리를 위해 구현한 함수를 호출한다.
  - onXXX 를 이용해 여러 이벤트 핸들러 속성을 부여하는 것도 가능 하다.



# JAVASCRIPT 문법

## 자바스크립트 에서 정의

- 자바스크립트 코드를 사용해 DOM element 즉, 특정 HTML 태그를 선택해 속성으로 event 발생 시 호출될 Callback method를 정의하는 형식 이다.
  - document.getElementById() 메서드는 HTML문서에서 지정한 id 속성을 찾아 바인딩 한다.
  - 이벤트 발생시 할당된 함수 부가 호출 되어 callback function 이라고 한다.

```
<script>
  document.getElementById('b1').onclick = function() {alert("버튼 클릭됨!!")}
</script>

<body>
  <input type="button" id="b1">
</body>
```

# JAVASCRIPT 문법

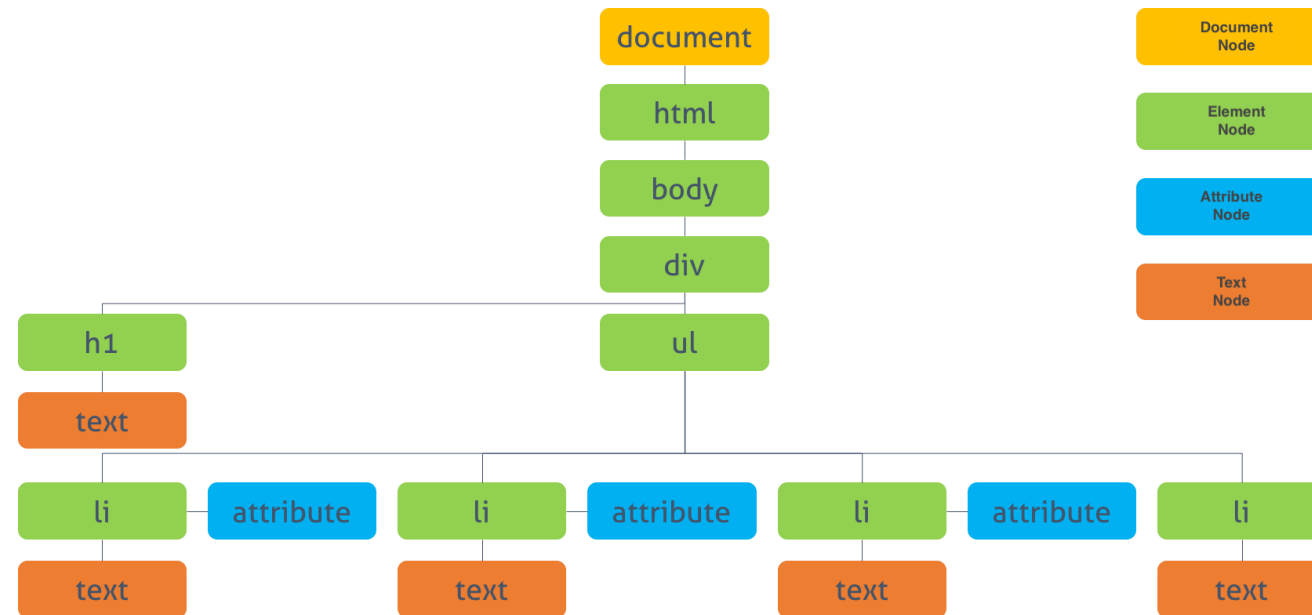
## DOM(Document Object Model)

- DOM 개요

- 문서 객체 모델이라고 하며 HTML 혹은 XML문서의 구조화된 표현을 제공하는 표준이다. HTML에서는 javascript가 DOM 구조에 접근할 수 있는 방법이 제공되며 이를 통해 문서 구조, 스타일, 내용 등을 변경 할 수 있다.
- 자바스크립트는 DOM을 이용하여 HTML의 태그, 속성, 스타일 등을 추가/삭제 하거나 변경할 수 있다.

# JAVASCRIPT 문법

## DOM(Document Object Model)



# JAVASCRIPT 문법

## DOM(Document Object Model)

- Document Node

- document 객체를 말하며 DOM 트리에 접근하기 위한 최상위 노드로 모든 DOM 트리에 접근하기 위한 시작점이 된다.

- Element Node

- HTML 구성 요소 즉 대표적으로 태그를 의미한다. 문서 내 태그들은 모두 Element Node 들 이다. 각각의 Element Node 는 다시 Attribute와 Text Node를 가진다.

- Attribute Node

- 태그들의 속성을 객체 화한 노드를 말한다. 만일 특정 태그의 속성에 접근하려면 Attribute Node를 사용해야 한다.

- Text Node

- 태그의 텍스트를 객체 화한 노드이다. 자식 노드를 가질 수 없으며 DOM 트리구조의 최종단 노드가 된다.

# JAVASCRIPT 문법

## HTML 요소 핸들링

- 자바스크립트에서 HTML DOM 요소에 접근하는 방법은 태그 이름, 아이디, 클래스, 이름 등을 이용해 특정 노드 객체를 선택하는 것이다. CSS의 선택자와 유사하다고 볼 수 있다.
- 예를 들어 버튼에서 이벤트가 발생했을 때 특정 입력 양식의 입력 값을 읽어 값에 따라 화면 구성을 변경해야 하는 경우 id로 특정 입력 양식 노드를 찾아서 입력 값을 가지고 온 다음 다시 화면 변경을 위해 특정 태그의 id값으로 해당 노드의 값이나 속성을 변경하는 등의 작업을 거치게 된다.

# JAVASCRIPT 문법

## HTML 요소 선택

- getElement(s)ByXXX() 를 이용해 특정 요소(태그) 노드를 가지고 온다.
- 모든 getElementsByXXX() 함수의 경우 동일 조건의 모든 노드를 목록으로 가져옴.
- getElementByName()의 경우 태그의 name 속성으로 노드를 찾음.

```
document.getElementsByTagName("tag_name")
document.getElementById("id_name")
document.getElementsByClassName("class_name")
document.getElementsByName("name_attribute")
```

```
let elist = document.getElementsByTagName("h2");

for(let el of elist) {
    console.log("##"+el.tagName);
}
```

# JAVASCRIPT 문법

## 쿼리 셀렉터

- getElementByXXX() 와 달리 단일 메서드로 원하는 요소를 찾을 수 있습니다. 단 유효한 CSS 셀렉터를 사용해야 하며 그렇지 않으면 예외가 발생합니다.
  - 콤마로 구분된 여러 셀렉터를 나열 할 수 있으며 해당 조건에 맞는 첫번째 노드만 가져옴.
  - 해당 조건의 모든 노드를 가지고 오려면 querySelectorAll()을 사용 함.

```
document.querySelector("#main, #title, #footer");  
document.querySelectorAll("p.note, p.tip")
```

# JAVASCRIPT 문법

## HTML 요소 생성, 제거, 추가

- HTML 요소를 선택하는 것 이외에도 특정 요소를 생성, 제거 하거나 자식 노드를 추가하는 것도 가능하다.

```
document.createElement(element)  
document.removeChild(element)  
document.appendChild(element)  
document.replaceChild(element)
```



# JAVASCRIPT 문법

## HTML 요소의 속성 핸들링

- HTML 요소를 선택한 다음에는 해당 요소의 속성과 텍스트 노드에 접근할 수 있다. 물론 해당 요소 내에 다른 요소들을 포함하고 있다면 마찬가지로 getElement(s)XXX() 함수를 사용해 자식노드들을 가지고 올 수 있다.

# JAVASCRIPT 문법

## HTML 요소의 속성값 변경

- 해당 태그의 속성값을 변경할 수 있다. 이 경우 원래 HTML 소스가 바뀌는 것이 아니라 동적으로 내용만 변경되고 브라우저에 반영된다.

```
element.setAttribute(attribute, value)
element.getAttribute(attribute, value)
element.removeAttribute(attribute)
```

다음 예제는 `title-img` 라는 이름의 id 값으로 `img` 태그를 가져와 `src` 속성을 `b.jpg` 로 변경하는 예입니다. 이벤트와 연결해서 실행하면 동적으로 이미지가 바뀌게 됩니다.

```
const el = document.getElementById("title-img");
el.setAttribute('src', 'b.jpg');
```

# JAVASCRIPT 문법

## HTML 이벤트핸들러 추가

- 해당 요소의 이벤트를 처리하는 방법으로 이벤트에서 살펴본 것처럼 이벤트 속성에 콜백 함수를 할당하는 형식으로 이벤트 핸들러를 추가할 수 있다.
  - `document.getElementById("id_name").onclick = function(){ code };`

## addEventListener()를 이용한 이벤트 핸들러 추가

- 이벤트를 요소에 추가하기 위한 가장 구조적인 방법으로 onclick 속성에 콜백 함수를 구현하는 것보다 좋은 방법입니다.
  - `document.getElementById("id_name").addEventListener("click", function(){ code });`

# JAVASCRIPT 문법

## 텍스트 노드 변경

- 해당 태그의 태그 바디에 있는 텍스트 노드의 값을 변경하기 위해서는 innerHTML 혹은 innerText 속성을 사용할 수 있다.
  - innerHTML 은 태그 바디에 다른 HTML태그와 자식노드를 포함한 텍스트를 처리할 때 사용.
  - innerText는 단순한 텍스트만을 처리할 때 사용.
  - innerText는 CSS에 종속적으로 CSS에 의해 hidden 처리가 되어 있다면 텍스트노드를 읽을 수 없다.
  - 따라서 가능하면 innerHTML사용을 권장.

```
element.innerHTML = '<tag>text</tag>';  
element.innerText = 'text';
```

# JAVASCRIPT 문법

## 디자인 요소 변경

- 자바스크립트에서의 이벤트 처리는 대부분 화면을 동적으로 변경하는 것과 관련이 있다.
- DOM 요소를 이용해 특정 영역에 데이터를 표현하거나 혹은 감출 수 있으며 CSS 디자인 속성을 변경해 모양이나 색상 등을 변경하는 것도 가능하다.
- 디자인 변경은 CSS의 기본적인 구조와 동작원리를 이해하고 있다면 어려울 것이 없다.
- 해당 요소의 style 속성 변경, style 객체 변경, 클래스 지정 등의 방법으로 변경이 가능하다.

# JAVASCRIPT 문법

## style 속성 변경

- 모든 HTML 태그는 style 속성을 가질 수 있으므로 다음과 같이 setAttribute로 style 속성을 변경할 수 있다.

```
document.getElementById('box1').setAttribute('style','background-color:yellow');
```

# JAVASCRIPT 문법

## style 객체 변경

- style 속성 대신 요소의 style 객체의 속성값들을 변경하는 형식이다.
  - 이 방법은 좀 더 구조적인 방법이며 프로그램 친화적인 형태임.
  - css 디자인 속성 이름은 - 로 연결되지만 이경우 Camel Case로 연결해야 함.
  - 예를 들어 background-color 속성은 backgroundColor 와 같이 사용.

```
document.getElementById('box1').style.backgroundColor = 'yellow';
```

# JAVASCRIPT 문법

## 클래스 지정

- 별도의 디자인 클래스를 지정해 놓고 해당 요소의 class 속성을 지정하는 형태입니다.
  - 이경우 별도의 디자인 클래스가 정의 되어 있어야 함.

```
document.getElementById('box1').className = 'bgyellow';
```



# JAVASCRIPT 문법

## 화면 숨기기

- 화면 구성요소를 상황에 따라 숨기거나 보여주는 방법은 UI 측면에서 매우 유용하다. CSS 를 이용해 요소를 보여주거나 숨기는 두가지 방법은 다음과 같다.
  - `display:none`
  - `visibility:hidden`

# JAVASCRIPT 문법

display:none

- 요소를 숨기며 요소가 차지하는 공간도 함께 사라진다. 다시 보이게 하려면 속성을 block 으로 변경하면 된다.

```
document.getElementById('box1').style.display = 'none';
```

# JAVASCRIPT 문법

visibility:hidden

- 요소를 숨기며 차지하는 공간도 유지하는 속성이다.
- width, height 가 지정되어 있으면 내용은 보이지 않아도 공간은 존재하게 된다. 다시 보이게 하려면 visible 이라고 하면 된다.

```
document.getElementById('box1').style.visibility = 'hidden';
```