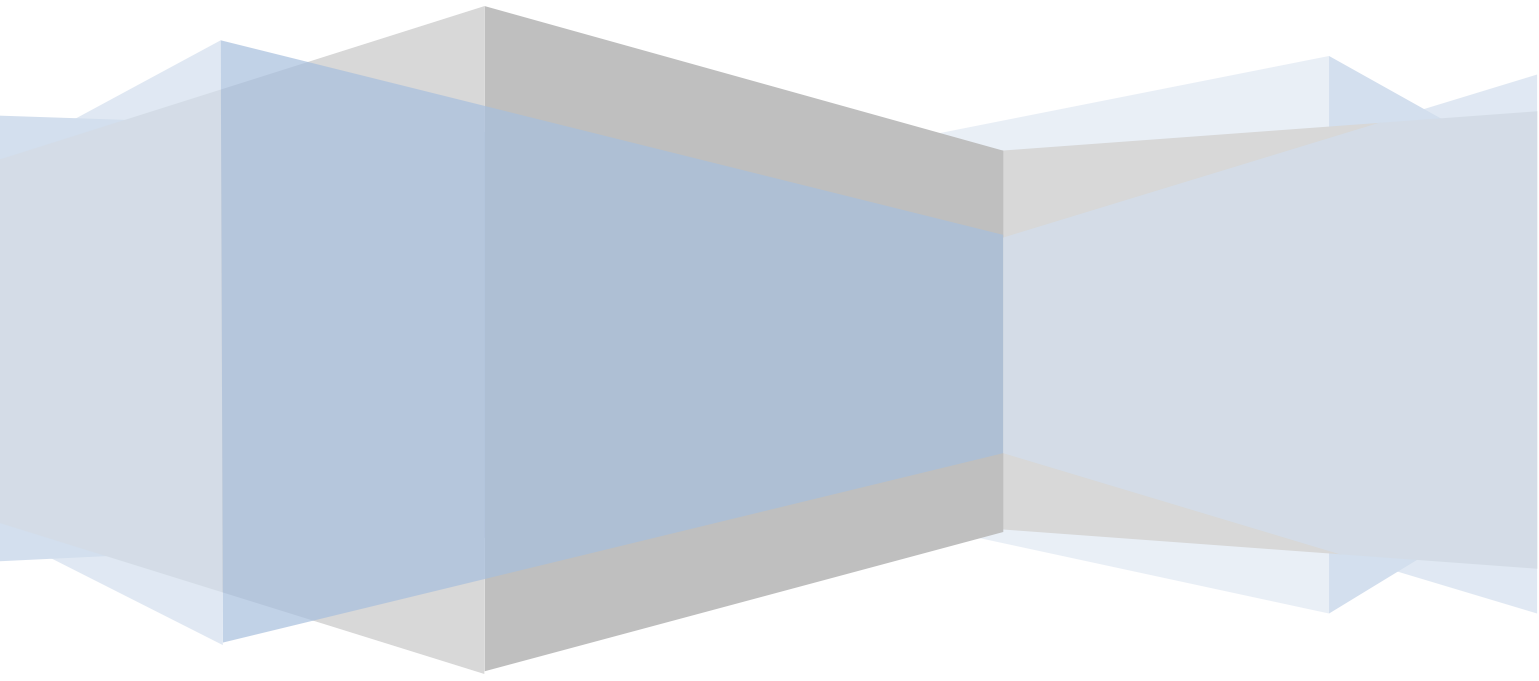
MEGAZONE CLOUD

JavaScript



자바스크립트 소개

- 자바스크립트는 동적인 웹 페이지를 위해서 만들어진 객체 기반 스크립트 언어이다. 웹 브라우저 내에서 주로 사용되었으나 Node.js 가 나오면서 자바스크립트는 더이상 웹 브라우저에 종속된 언어가 아니다. 웹 애플리케이션은 물론이고 데스크톱 애플리케이션, 모바일 앱, 키오스크, 게임 등 자바스크립트를 사용할 수 있는 분야는 점점 다양해지고 있다.

자바스크립트 엔진

1. V8 - Chrome, Opera
2. SpiderMonkey - Firefox
3. 'Trident', 'Chakra' - ChakraCore'는 Microsoft Edge, Trident 는 IE
4. 'SquirrelFish' - Safari

자바스크립트 엔진의 동작원리

1. 자바스크립트 엔진(브라우저라면 내장 엔진)이 스크립트를 읽는다. (파싱)

2. 읽어 들인 스크립트를 기계어로 전환한다. (컴파일).

3. 기계어로 전환된 코드가 실행된다. 기계어로 전환되었기 때문에 실행 속도가 빠르다.

4. 엔진은 프로세스 각 단계마다 최적화를 진행합한다. 심지어 컴파일이 끝나고 실행 중인 코드를 감시하면서, 이 코드로 흘러가는 데이터를 분석하고, 분석 결과를 토대로 기계어로 전환된 코드를 다시 최적화하기도 한다.. 이런 과정을 거치면 스크립트 실행 속도는 더욱 더 빨라진다.

자바스크립트 개발환경

1. VIsual Studio Code 설치

- https://code.visualstudio.com 사이트에서 다운로드한다.

2. Extensions 아이콘을 선택하여 확장 플러그인을 검색하여 설치한다.

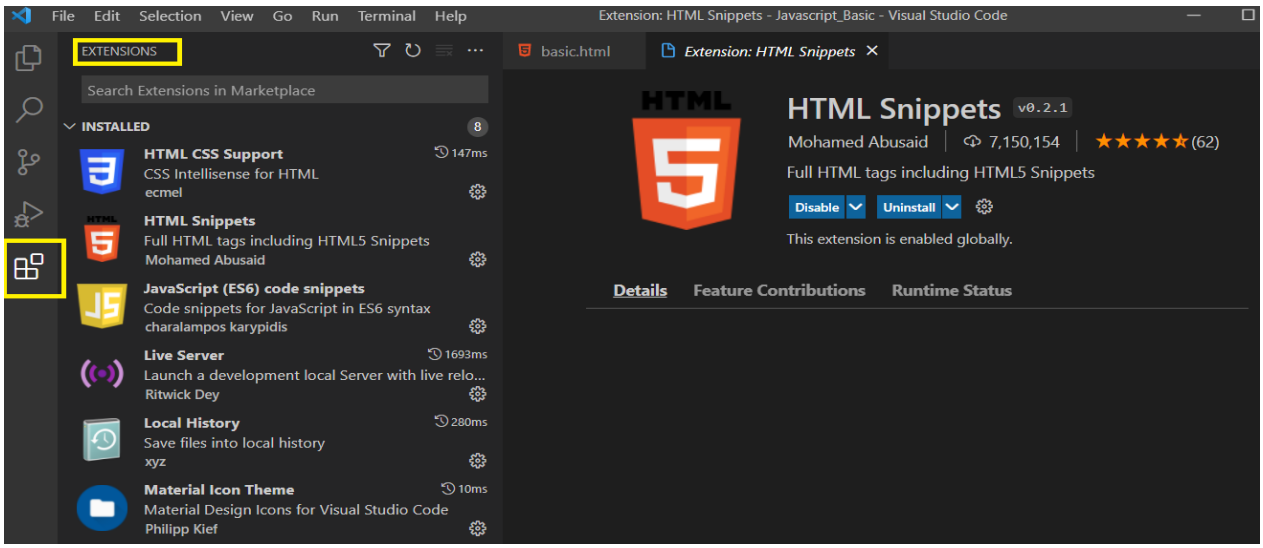
- HTML Snippets

- HTML CSS Support

- JavaScript (ES6) code snippets

- Live Server

- Open In Default Browser



자바스크립트 작성법

1. script 태그 사이에 기술하는 방법

```
<script>

    // 자바스크립트 코드

</script>
```

2. 태그에 직접 기술하는 방법

```
<input type="button" onclick="alert('Hello, Javascript!!');" />
```

3. 외부 파일로 로드하는 방법

```
<script src = "외부 자바스크립트 파일의 경로"></script>
```

콘솔 사용하기

● console.log( )

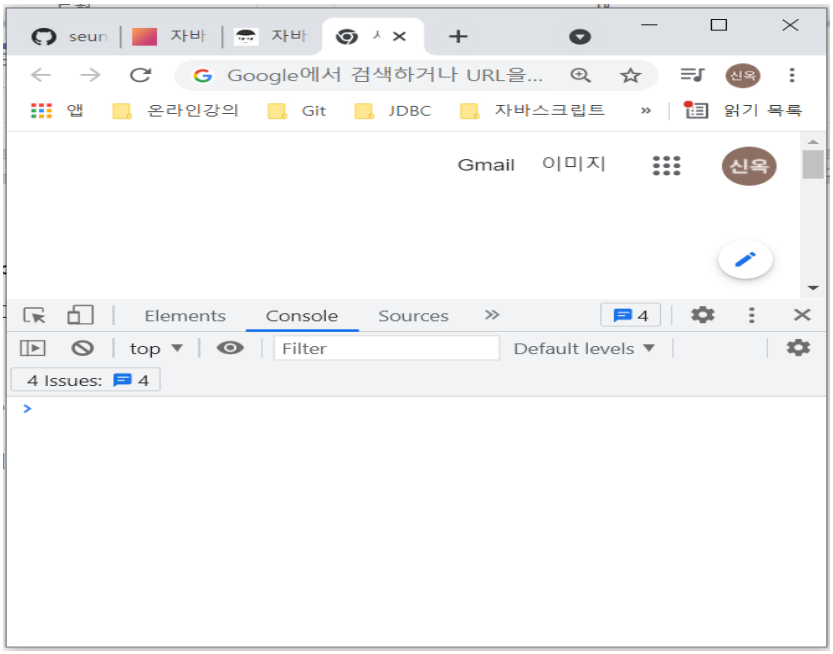
- console.log( ) 함수는 인수로 설정한 값을 디버거 콘솔에 표시하는 함수이다.

- 자바스크립트 프로그래밍에서 디버깅할때 사용된다.

● Google Chrome 에서 console.log( )를 사용하는 방법

- Chrome 브라우저에서 개발자도구(F12 또는 CTRL + SHIFT + I)를 선택한다.

- 자바스크립트 콘솔 화면에서 변수 값이나 객체 내용등을 표시할 수 있다.



변수 (Variable)

- 변수는 데이터를 기억하는 메모리 공간이다.

● 변수 선언 3 가지 방법

1. var

[문법] var name = 'Emma';

- 함수 유효 범위를 갖는다.

- Hoisting 된다.

2. let (added in ECMAScript 6)

- 블록{ } 유효 범위를 갖는다.

- Hoisting 된다.

3. const (added in ECMAScript 6)

- 블록{ } 유효 범위의 상수를 선언한다.

- Hoisting 된다.

※ Hoisting 이란?

- var 변수과 함수 선언와 import 구문을 맨 위로 이동시키는 것을 말한다.

  (식별자 정보를 실행 컨텍스트의 맨 위로 끌어 올리다.)

TDZ(Temporay Dead Zone)은 무엇인가?

※ https://noogoonaa.tistory.com/78 사이트 참조할 것!!

※ https://poiemaweb.com/js-execution-context 참조할 것!!

- 일시적인 사각지대는 스코프의 시작 지점부터 초기화 시작 지점까지의 구간을 TDZ (Temporal Dead Zone) 라고 한다.

- TDZ 은 let , const , class 구문의 유효성을 관리한다.

1. let

2. const

3. class

4. constructor() 내부의 super()

5. Default Function Parameter

데이터 타입

1. 기본 타입

- boolean : true 와 false 의 두 가지 값을 갖는다.

- null : 값이 없는 상태이다.

- undefined : 변수는 선언되어 있지만 값이 할당되지 않는 변수는 undefined 값을 갖는다.

- number : 64 비트 부동소수점형태의 값을 갖는다. 정수만을 표현하기 위한 특별할 자료형은 없다.

- string : 문자열

- Symbol (added in ECMAScript 6)

2. 참조 타입

- 객체

- Function

- 배열

- RegExp

- Set / WeakSet (added in ECMAScript 6)

- Map / Weak (added in ECMAScript 6)

※ 자료형 확인하기

```
var name = “Emma”;


console.log( typeof   name );


// 객체 타입 확인

console.log(toString.call(name));
```

연산자

1. 산술 연산자

   +, -, *, /, %

2. 문자열연산자

   문자열과 더할 때 문자열이 아닌 데이터는 문자열로 바꿔서 연결된다.

3. 증감 연산자

   ++, --

4. 대입 연산자

5. 비교 연산자 === (값 뿐만 아니라 자료형까지 비교한다.)

6. 논리 연산자

7. 삼항 연산자

   조건식? 참 : 거짓

논리 연산자

- 자바스크립트에서 논리연산자는 참과 거짓을 판단해주는 연산자일 뿐 아니라 연산에 사용된 피 연산자 중 하나를 반환해주는 연산자이다.

const a = true;

a || 8

3 || 4

false || 4

0 || 9

- 논리 OR 연산자는 → 방향으로 연산을 진행하고 가장 먼저 참(true) 의 형태를 가진 value 가 나오는 경우 그 피연산자를 바로 반환하고 연산을 끝낸다.

※ 논리 OR 연산자

논리합 연산자 패턴	단축평가 결과
false || 값	값
true || 값	true
값 || false	값
값 || true	값
값 A || 값 B	값 A

※ 논리 AND 연산자

논리곱 연산자 패턴	단축평가 결과
false && 값	false
true && 값	값
값 && false	false
값 && true	true
값 A && 값 B	값 B

함수(function)

- 특별한 목적의 작업을 수행하도록 설계된 독립적인 블록이다.

- 자바스크립트에서 함수는 일급 함수(First-Class-Function) 이다

※ First-Class Citizen 이란?

1) 함수를 변수(variable)에 담을 수 있다.

2) 함수를 함수의 매개변수(parameter)로 전달할 수 있다.

3) 함수를 함수의 반환값(return value)으로 전달할 수 있다.

● 함수 생성 방법

1. 함수 선언문

```
function 함수명 ( [매개변수 1, 매개변수 2,...] ) {
    실행문;
  [ return 값; ]
}
```

2. 함수 표현식(리터럴) : 익명 함수 표현식

```
const 변수명 = function ( [매개변수 1, 매개변수 2,...] ) {
    실행문;
    [ return 값; ]
};
```

3. 중첩함수 (Nested Function)

- 다른 함수 내부에서 정의되는 함수를 의미한다.
- 캡슐화를 구현할 수 있다.

[Example]

```
// 외부 함수
function outer( ) {
      const prefix = 'Hello, ';

      // 내부 함수
      function inner(name) {
          // 내부 함수에서는 외부 함수(상위스코프)의 지역 변수를 사용할 수 있다.
          return prefix + ' ' + name;
       }

        return inner;
 }

// 외부함수가 종료된 이후에도 내부 함수를 통해서
// 외부 함수의 지역 변수를 접근할 수 있다. (클로저)
 const inner = outer( );
 console.log(inner('kim'));
```

※ Closure(클로저) 란?

- 내부함수가 정의될 때 외부 함수의 환경(Lexical Context)을 기억하고 있는 내부 함수를 말한다.

· 내부 함수에서 외부 함수(상위 스코프)의 지역 변수에 접근하여 사용할 수 있다.

- 외부 함수가 종료된 이후에도 내부 함수를 통해 외부 함수의 지역 변수에 접근할 수 있는 것이 클로저 특성이다.

[Example]

```
 // Click 버튼 클릭 시 count를 1씩 증가하여 출력하는 예제

<button id="btn" type="button">Clcik</button>

<script>
      let count = 1;
      const btn = document.getElementById('btn');

       btn.addEventListener('click', function() {
            console.log('count = ', count++);
       });

       // 문제가 발생할 수 있다.
       count = 'Hello';

</script>
```

[Example]

```
const btn = document.getElementById('btn');

 btn.addEventListener('click', function( ) {
      increase( );
 });

 // 즉시 실행 함수 (immediate function)
 const increase = ( function( ) {
      let count = 1;
      return function() {
           console.log('count = ', count++);
      };
 } )( );
```

※ 클로저를 사용함으로써 전역 변수의 남용을 막을 수 있고 변수 값을 은닉하는 용도로도 사용할 수 있다.

4. 콜백함수 (Callback Function)

- 다른 함수(A)의 인자로 콜백 함수(B)를 전달하면 A 가 B 의 제어권을 갖게 된다.

※ 제어권 위임

1. 어떤 시점에 콜백함수를 호출할 지
2. 인자에 어떤 값들을 지정할 지
3. this 에 무엇을 바인딩할 지

```
 // add : 고차 함수
function add(a, b, func) {
      // 콜백 함수
      func(a + b);

 }

 function printConsole(result) {
     console.log(reuslt);
 }

 // 매개 변수로 전달되는 함수를 콜백함수라고 한다.
 // printConsole : 콜백 함수
 add(1, 2, printConsole);
```

실행 컨텍스트(Execution Context)

- ECMAScript 스펙에 따르면 실행 컨텍스트를 실행 가능한 코드를 형상화하고 구분하는 추상적인 개념 이라고 정의한다. 좀 더 쉽게 말하자면 실행 컨텍스트는 실행 가능한 코드가 실행되기 위해 필요한 환경이다. 여기서 말하는 실행 가능한 코드는 아래와 같다.

  ► 전역 코드 : 전역 영역에 존재하는 코드

  ► 함수 코드 : 함수 내에 존재하는 코드

- 자바스크립트 엔진은 코드를 실행하기 위하여 실행에 필요한 여러가지 정보를 알고 있어야 한다.

  ► 변수 : 전역변수, 지역변수, 매개변수, 객체의 프로퍼티

  ► 함수 선언

  ► 변수의 유효범위(Scope)

  ► this

이와 같이 실행에 필요한 정보를 형상화하고 구분하기 위해 자바스크립트 엔진은 실행 컨텍스트를 물리적 객체의 형태로 관리한다

1. Lexical Environment

- 실행 컨텍스트를 구성하는 환경 정보들을 모아 사전처럼 구성한 객체이다.

- 식별자 정보 및 상위 스코프 정보를 저장 관리한다.

- 각 식별자의 데이터를 추적하여 변화가 실시간으로 반영된다.

environmentRecord

- 현재 컨텍스트의 식별자 정보가 저장된다.

- Hoisting

outerLexicalEnvriomnetReference

- 외부렉시컬환경정보를 참조한다.

- Scope Chain

2. ThisBinding

This binding

· 실행 컨텍스트가 활성화될 때 this 가 바인딩된다.

· 함수가 호출될 때 this 가 동적으로 바인딩된다.

전 역 공 간 에 서	window / global(node.js) 전역 객체
함수 호출 시	window / global(node.js) 전역 객체
메소드 호출 시	메소드 호출 주체
콜백 호출 시	기본적으로 함수의 this 와 같다. 제어권을 가진 함수가 콜백의 this 를 지정해 둔 경우가 있다. 개발자가 this 를 바인딩해서 콜백을 넘기면 그에 따른다.
생성자 함수 호출 시	인스턴스 (생성자 함수를 통해서 생성된 객체)

※ 명시적으로 this 바인딩하는 방법

1. call

2. apply

3. bind

함수 호출 시

```
function add(a, b) {
      // this : window 객체
      console.log('this : ', this);
      return a + b;
 }


  // add 함수 호출

  console.log(add(1, 2));
```

메소드 호출 시

```
const person = {
      userName: 'kim',
      age: 20,
      getPerson( ) {
           return 'userName : ' + this.userName + ", age : " + this.age;
      }
 }

 // 메소드 호출 시
 // this 는 메소드를 호출한 객체 person 을 가리킨다.
 console.log(person.getPerson());
```

콜백 함수 호출 시

```
function add(a , b, callback) {
    callback(a + b);
}

 // 콜백 함수
 // 기본적으로 기본 함수의 this 와 같다.
 // this : window 객체
function consoleResult(value) {
    console.log('this : ' , this);
    console.log(value);
}

add(1, 2, consoleResult);
```

명시적으로 this 바인딩

```
const apple = {
    name: '사과'
}

const banana = {
    name: '바나나'
}

function update(color, price) {
    this.color = color;
    this.price = price;
}

function whatIsThis() {
    console.log(this);
}

// whatIsThis 함수 호출 시 apple 객체를 전달하여
// this 를 apple 객체로 바인딩 할 수 있다.
whatIsThis.call(apple);

whatIsThis.appley(banana);

update.call(apple, '빨강', 1000);
console.log(apple);

update.apply(banana, ['빨강', 1000]);
console.log(banana);
```

```
// bind 함수 호출 시 apple 객체를 전달하여 update 함수에서
// this 객체로 바인딩 할 수 있다.
const apple1 = update.bind(apple);
apple1('빨강', 2000);
console.log(apple);
```

내장 객체

Object 객체

- 자바스크립트의 최상위 객체이다.

● Object 객체 생성

var obj = { };

var obj = new Object( );

● Object 객체의 메소드

hasOwnProperty(name) : 객체가 name 속성을 가지고 있는지 확인한다.

isPrototypeof(object) : 객체가 object 의 프로토타입인지 검사한다.

propertyIsEnumerable(name) : 반복문으로 열거 가능 여부 확인한다.

toString() : 객체를 문자열로 변경한다.

valueOf() : 객체의 값을 표시한다.

String 객체

- 문자열을 표현할 때 사용하는 객체이다.

● String 객체의 속성

length : 문자열의 길이

● String 객체의 메소드

charAt(position) : 해당 인덱스 문자 반환한다.

charCodeAt(position) : 해당 인덱스 문자를 유니코드로 반환한다

concat(args) : 매개변수로 입력한 문자열을 결합한다.

indexOf(searchString, position) : 앞에서부터 일치하는 문자열의 인덱스 반환한다

lastIndexOf(searchString, position) : 뒤에서부터 일치하는 문자열의 인덱스 반환한다

match(regExp) : 문자열 안에 regExp 가 있는지 확인한다.

replace(regExp, replacement) : 문자열 안의 regExp 를 replacement 로 바꾼 뒤 리턴

search(regExp) : regExp 와 일치하는 문자열의 위치 반환한다

slice(start, end) : 특정 위치의 문자열을 반환한다

split(separator, limit) : separator 로 문자열을 자른 후 배열로 반환한다

substr(start, count) : start 부터 count 까지 문자열을 잘라서 반환한다

substring(start, end) : start 부터 end 까지 문자열을 잘라서 반환한다

toLowerCase() : 문자열을 소문자로 바꾸어 반환한다.

toUpperCase() : 문자열을 대문자로 바꾸어 반환한다.

padStart(newLength, [padString]) : 현재 문자열의 시작을 다른 문자열로 채워 주어진 길이 값을 갖는 문자열을 반환한다.

padEnd(newLength, [padString]) : 채워넣기는 대상 문자열의 끝(우측)부터 적용된다.

trim() : 문자열 양 끝의 공백(space, ta, 개행문자)을 제거한다.

Number 객체

- 숫자를 표현할 때 사용하는 객체이다. -

● Number 객체 생성

var num = 1; -

-

var num = new Number(1); -

● Number 객체의 메소드

parseInt(string, [n]) : 문자열을 특정 진수(2 진수, 10 진수)의 정수로 반환한다.

parseFloat(string) : 문자열을 부동소수점 실수로 반환한다.

Array 객체

· 배열을 표현할 때 사용하는 객체이다.

● 배열 생성

var array = [값 1, 값 2...]; // 생성과 동시에 초기화

var array = new Array(); // 배열을 생성하기만 함

var array = new Array(배열크기); // 생성과 동시에 공간을 만들어 둠

var array = new Array(값 1, 값 2); // 생성과 동시에 초기화

● Array 객체의 속성

length : 배열의 크기를 반환

● Array 객체의 메소드

pop() : 배열의 마지막 요소를 제거 후 리턴

push() : 배열의 마지막에 새로운 요소 추가

unshift() : 새로운 요소를 배열의 맨 앞쪽에 추가

shift() 메서드는 배열에서 첫 번째 요소를 제거하고 제거된 요소를 반환

sort() : 배열 요소 정렬 // 기본값은 문자열 오름차순

reverse() : 배열의 요소 순서 반전

splice() : 요소의 지정된 부분 삭제 후 삭제한 요소 리턴

slice() : 요소의 지정한 부분 반환

concat() : 매개변수로 입력한 배열의 요소를 모두 합쳐서 배열 생성 후 반환

join() : 배열 안의 모든 요소를 문자열로 변환 후 반환

indexOf() : 배열의 앞쪽부터 특정 요소의 위치 검색

lastIndexOf() : 배열의 뒷쪽부터 특정 요소의 위치 검색

반복 메소드

- forEach() : 배열을 for in 반복문처럼 사용 가능

배열.forEach(function(element, index, array) { // 콜백 함수

// 배열요소, 인덱스, 배열자체

});

```
const persons = [
    { firstName: "Julia", lastName: 'Roberts', age: 20} ,
    { firstName: "Brad", lastName: 'Pit', age: 20 },
    { firstName: "Leonardo", lastName: 'Dicaprio', age: 30 }
]


// Array 객체의 forEach 메소드를 사용하여 배열 요소의 내용 출력하기
persons.forEach(function(p) {
    console.log(`firstName : ${p.firstName}, lastName : ${p.lastName}, age: ${p.age}`);
});


// Arrow Function 를 사용하여 배열 요소의 내용 출력하기
persons.forEach( p => {
    console.log(`firstName : ${p.firstName}, lastName : ${p.lastName}, age: ${p.age}`);
});
```

map() : 기존의 배열에 특정 규칙을 적용해서 새로운 배열 생성 (return 필수)

```
const persons = [
    { firstName: "Julia", lastName: 'Roberts', age: 20} ,
    { firstName: "Brad", lastName: 'Pit', age: 20 },
    { firstName: "Leonardo", lastName: 'Dicaprio', age: 30 }
]


const arr = persons.map(person => {
    return {
        fullName: `${person.firstName} ${person.lastName} `,
        age: `${person.age}`
    }
});


arr.forEach(p => {
    console.log(`fullName : ${p.fullName}, age : ${p.age}`);
});
```

조건 메소드

filter() : 특정 조건을 만족하는 요소를 추출해 새로운 배열 생성

```
const array = array.filter(function(element, index, array) {
    // 반드시 true, false 를 반환한다.
    return element < 5;
});
```

every() : 배열의 요소가 조건을 만족하는지 확인

some() : 배열의 요소가 특정 조건을 적어도 하나는 만족하는지 확인

```
array.every(function (element, index, array) {
    return element < 10;
});
```

- every 는 모든 요소, some 은 단 하나라도 조건에 부합하는지 확인 후 true or false 반환한다.

연산 메소드

reduce() : 배열의 요소가 하나가 될 때까지 요소를 왼쪽부터 두 개씩 묶는 함수

(return 필수)

reduceRight() : 위와 같으나 오른쪽부터 실행

```
const intarr = [ 1, 2, 3, 4, 5,, 6, 7, 8, 9, 10 ];

 const result = intarr.reduce((prev, curr, index) => {
        return prev + curr;
 }, 0);

 console.log(result);
```

prev	curr	index
0	1	0
0 + 1	2	1
0 + 1 + 2	3	2
…	…	…
0 + 1 + 2 … + 9 + 10	10	9

Date 객체

- 날짜와 시간을 표시하는 객체이다.

● Date 객체 생성

var date = new Date(); // 매개변수를 입력하지 않으면 현재 시각으로 초기화

var date = new Date(2018, 12, 11, 2, 24, 23); // 연, 월-1, 일, 시, 분, 초 순서

new Date(1351511); // Unix time, 1970 년 1 월 1 일 12 시 자정 기준으로 경과한 시간 (밀리초)

● Date 객체의 메소드

getTime() : Unix time 반환, 날짜 간격 계산시 사용

getFullYear() : 연도를 반환한다.

getMonth() : 월을 반환한다. (1 월 : 0)

getDate()

getDay() : 요일을 반환한다. (월 : 0)

Math 객체

- 수학과 관련된 속성과 메소드를 가진 객체이다.

● Math 객체의 메소드

ceil() : 크거나 같은 가장 작은 정수 반환한다.

floor() : 작거나 같은 가장 큰 정수 반환한다.

pow(x, y) : x 의 y 제곱 반환한다.

random() : 0 부터 1 까지의 임의의 수 반환한다

round() : 반올림해서 반환한다

trunc() : 주어진 값의 소수부분을 제거하고 숫자의 정수부분을 반환한다.

JSON 객체

- https://www.json.org/json-en.html 사이트 참조할 것.

- Javascript Object Notation

- 서버와 클라이언트 간의 네트워크 통신에 사용하는 경량의 데이터 교환 형식(포맷)이다.

● JSON 객체의 메소드

JSON.stringify() : 자바스크립트 값이나 객체를 JSON 문자열로 변환하여 반환한다.

JSON.parse() : JSON 문자열을 자바스크립트 값이나 객체로 변환하여 반환한다.

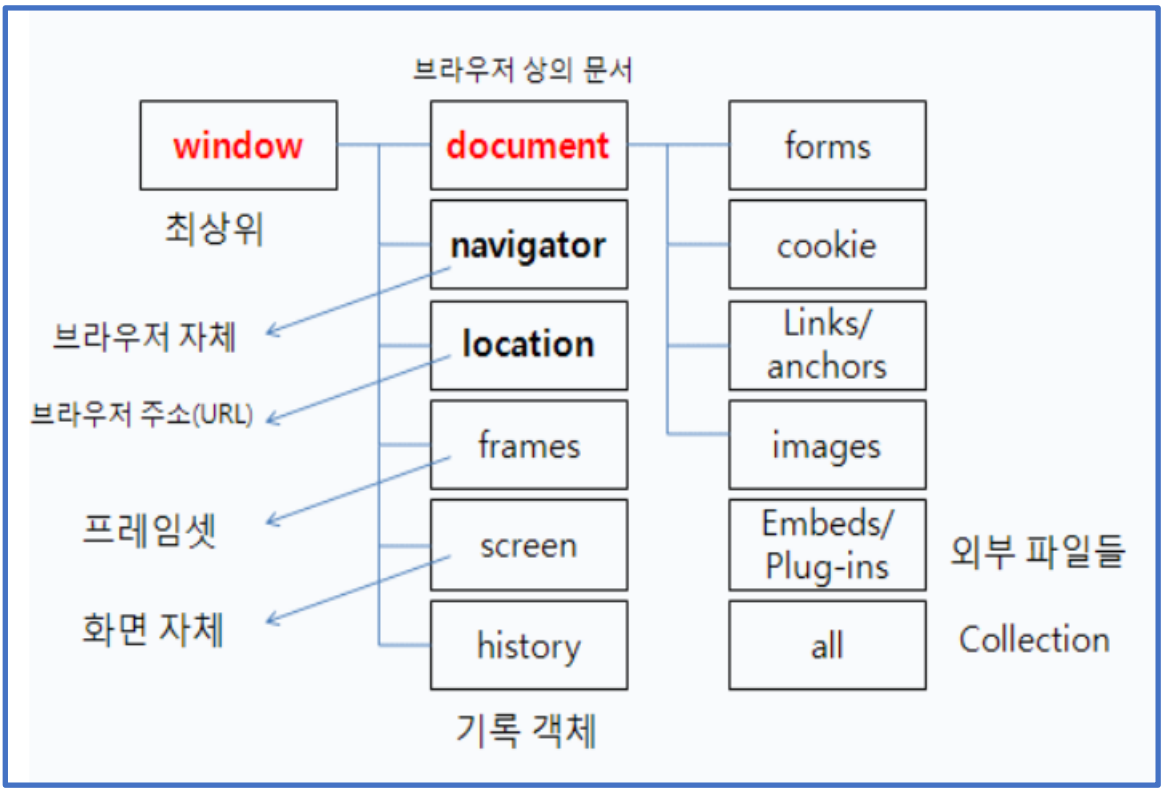
toJSON() : 자바스크립트의 Date 객체의 데이터를 JSON 형식의 문자열로 변환하여 반환한다.

toJSON() : 접미사 Z 로 식별되는 UTC 표준 시간대의 날짜를 ISO 8601 형식의 문자열로 반환한다. (2021-09-01T07:56:03.474Z)

브라우저 객체 모델 (BOM)

- 웹 브라우저와 관련된 객체의 집합이다.

- 정확히는 자바스크립트가 아닌 웹 브라우저가 제공하는 객체이다.



window 객체

- 웹 브라우저 기반 자바스크립트의 최상위 객체이다.

- var 키워드로 변수를 선언하거나 함수를 선언하면 window 객체의 프로퍼티가 된다.

- alert(), prompt() 등 많은 메소드를 가지고 있다.

타이머 함수

함수 이름	실행
setTimeout(함수, 시간)	일정 시간(밀리초) 경과 후 함수 실행
setInterval(함수, 시간)	일정 시간(밀리초) 간격으로 함수 반복 실행
clearTimeout(id)	실행중인 timeout 을 중지한다.
clearInterval(id)	실행중인 interval 을 중지한다.

```
 <button type="button" id="startBtn">Start</button>
<button type="button" id="stopBtn">Stop</button>
<div id="datetime"></div>

<script>
    const startBtn = document.getElementById('startBtn');
    const stopBtn = document.getElementById('stopBtn');
    const div = document.getElementById('datetime');
    let timerid = null;

    // Start 버튼 클릭 시 이벤트 처리
    // 1 초 간격으로 showDateTime 함수를 호출하여 현재 날짜와 시간을 보여준다.
    startBtn.addEventListener('click', function() {
        showDateTime();
        timerid = setInterval(showDateTime, 1000);
    });

     // Stop 버튼 클릭 시 이벤트 처리
     // 실행중인 setInterval 함수를 중지한다.
     stopBtn.addEventListener('click', function() {
            if (timerid != null) {
                clearInterval(timerid);
            }
     });

     function showDateTime() {
         const now = new Date();
         let htmlStr = '<p>';
         htmlStr += `${now.getFullYear()}/${now.getMonth() + 1}/${now.getDate()}   `;
         htmlStr += `${now.getHours()}:${now.getMinutes()}:${now.getSeconds()}`;
         htmlStr += '</p>';
         div.innerHTML = htmlStr;
     }
</script>
```

```
<button type="button" id="startBtn">Start</button>
<button type="button" id="stopBtn">Stop</button>
<div id="datetime"></div>

<script>
    const startBtn = document.getElementById('startBtn');
    const stopBtn = document.getElementById('stopBtn');
    const div = document.getElementById('datetime');
    let timerid = null;

    startBtn.addEventListener('click', function() {
        showDateTime();
    });

    stopBtn.addEventListener('click', function() {
            if (timerid != null) {
                clearTimeout(timerid);
            }
    });

    function showDateTime() {
        const now = new Date();
        let htmlStr = '<p>';
        htmlStr += `${now.getFullYear()}/${now.getMonth() + 1}/${now.getDate()}    `;
        htmlStr += `${now.getHours()}:${now.getMinutes()}:${now.getSeconds()}`;
        htmlStr += '</p>';
        div.innerHTML = htmlStr;
         // 이 부분이 중요
        timerid = setTimeout(showDateTime, 1000);
    }
</script>
```

location 객체

- 웹 브라우저의 주소 표시줄과 관련된 객체이다.

- location 객체는 프로토콜의 종류, 호스트 명, 문서 위치 등의 정보가 있다.

● location 객체의 속성

href : 문서의 URL 주소

host : 호스트 이름과 포트번호 // localhost:30763

hostname : 호스트 이름 // localhost

port : 포트 번호 // 30763

pathname : 디렉토리 경로 // /a/index.html

hash : 앵커 이름(#~) // #beta

search : 요청 매개변수 // ?param=10 (Query String : 질의문자열)

protocol : 프로토콜 종류 // http:

● location 객체의 메소드

assign(link) : 현재 위치를 이동

reload() : 새로고침

replace(link) : 현재 위치를 이동(뒤로가기 사용 불가)

자바스크립트 페이지 이동

location.href	location.replace
새로운 페이지로 이동한다.	기존 페이지를 새로운 페이지로 변경시킨다.
속성	메소드
히스토리에 기록된	히스토리에 기록되지 않는다.

location.href = '이동하고자 하는 페이지 주소';

location.replace ('이동하고자 하는 페이지 주소');

navigator 객체

- 웹 페이지를 실행 중인 브라우저에 대한 정보가 담긴 객체이다.

● navigator 객체의 속성

appCodeName : 브라우저의 코드 이름

appName : 브라우저의 이름

appVersion : 브라우저의 버전

platform : 사용 중인 운영체제의 시스템 환경

userAgent : 웹 브라우저 전체 정보

history 객체

- 웹 브라우저의 history 정보(사용자가 방문한 URL 정보)를 저장하는 객체이다.

히스토리 내 이동하기

- history.back(): 윈도우 열람 이력에서 뒤로 이동

- history.forward(): 윈도우 열람 이력에서 앞으로 이동

- history.go();

문서 객체 모델(DOM)

https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html 사이트 참고할 것.

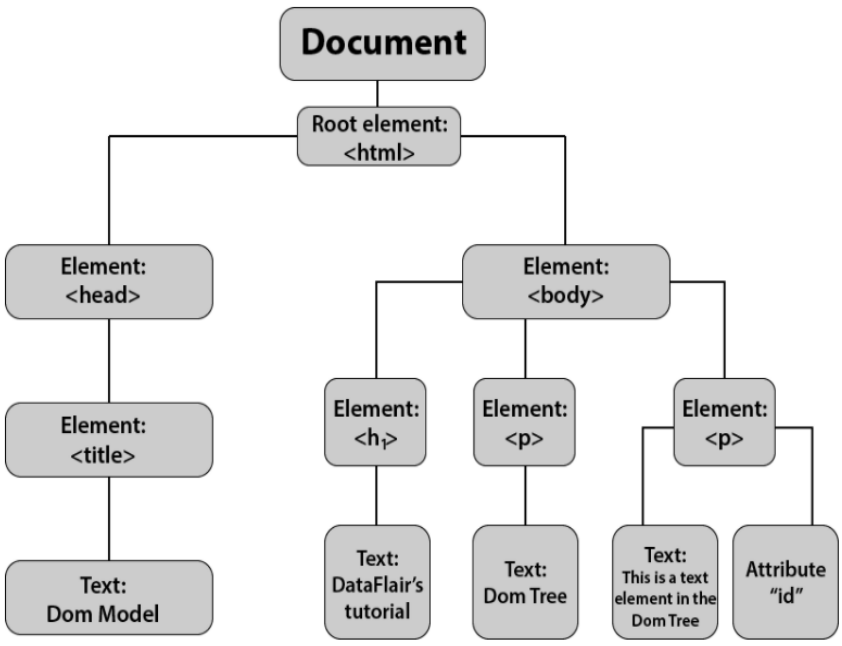
● DOM(Document Object Model) 이란?

· HTML 문서의 구조와 관계를 객체로 표현한 모델이다.

· DOM 은 트리구조이다.

· DOM 은 HTML 문서의 요소를 제어하기 위해 웹 브라우저에서 처음 지원되었다.

· DOM API 는 프로그래밍 언어에서 DOM 구조에 접근할 수 있는 방법을 제공하며, 동적으로 문서 구조, 스타일, 내용 등을 변경할 수 있다. ₩



● DOM 요소 노드 취득

1. id 를 이용한 요소 노드 취득

const elem = document.getElementById(‘id 속성값’);

2. CSS 선택자를 이용한 요소 노드 취득

const elems = document.querySelectorAll("선택자");

- DOM 컬렉션 객체인 NodeList 객체를 반환한다.

- NodeList 객체는 실시간으로 노드 객체의 상태 변경을 반영하지 않는다.

const elem = document.querySelector(‘선택자’);

3. 태그 이름을 이용한 요소 노드 취득

const elems = document.getElementsByTagName(‘태그이름’);

4. class 를 이용한 요소 노드 취득

const elems = document.getElementsByClassName(‘class 속성값’);

- HTMLCollection 객체를 반환한다.

- HTMLCollection 객체는 노드 객체의 상태 변화를 실시간으로 반영하는 DOM 컬렉션 객체이다.(라이브 객체)

[Example]

```
<ul class="fruits">
        <li class="red">Apple</li>
        <li class="red">Banana</li>
        <li class="red">Cherry</li>
        <li class="red">Shine musket</li>
</ul>

<script>
        // HTMLCollection 객체를 반환한다.
        const elems = document.getElementsByClassName('red');

        for (const elem of elems) {
            elem.className = 'blue';
        }
        console.log(elems);

</script>
```

인덱스	요소	요소
0	<li class="red">Apple	<li class="blue">Apple
1	<li class="red">Apple	<li class="red">Apple
2	<li class="red">Apple	<li class="red">Apple
3	<li class="red">Apple	<li class="red">Apple

● 노드 추가

Node Interface

1. Node appendChild(newChild : Node)

새로운 노드를 해당 노드의 자식 노드 리스트에 맨 마지막 노드로 추가한다.

2. Node insertBefore(newChild : Node, refChild : Node)

새로운 노드(newChild)를 특정 노드(refChild) 바로 앞에 추가한다.

● 노드 생성

Document Interface

1. Element createElement(tagName : DOMString)

새로운 요소 노드를 생성하여 반환한다.

2. Attr createAttribute(name : DOMString) :

새로운 속성 노드를 생성하여 반환한다.

3. Text createTextNode(data : DOMString)

새로운 텍스트 노드를 생성하여 반환한다.

● 노드 제거

Node Interface

1. Node removeChild(oldChild : Node)

기존의 노드 리스트에서 특정 노드(oldChild)를 제거한다.

Element Interface

2. removeAttribute(name : DOMString)

속성의 이름을 이용하여 특정 속성 노드를 제거함.

● 노드의 값 변경

- nodeValue 프로퍼티를 사용하면 특정 노드의 값을 변경할 수 있다.

- setAttribute() 메소드는 속성 노드의 속성값을 변경할 수 있게 해준다.

Node Interface

1. readonly attribute DOMString nodeValue ; // 속성

Element Interface

2. setAttribute(name : DOMString, value : DOMString)

● 노드 교체

replaceChild 메소드를 사용하면 특정 노드를 다른 노드로 바꿀 수 있다.

Node Interface

1. replaceChild(newChild : Node, oldChild : Node)

자바스크립트 이벤트 처리

● 이벤트 (Event)

- 웹페이지에서 마우스를 클릭했을 때, 키를 입력했을 때, 특정요소에 포커스가 이동되었을 때 어떤 사건을 발생시키는 것이다.

● 이벤트 유형

1. 마우스 이벤트

이벤트	설명
click	요소에 마우스를 클릭했을 때 이벤트가 발생
Dbclick	요소에 마우스를 더블클릭했을 때 이벤트가 발생
mouseover	요소에 마우스를 오버했을 때 이벤트가 발생
mouseout	요소에 마우스를 아웃했을 때 이벤트가 발생
mousedown	요소에 마우스를 눌렀을 때 이벤트가 발생
mouseup	요소에 마우스를 떼었을 때 이벤트가 발생
mousemove	요소에 마우스를 움직였을 때 이벤트가 발생
contextmenu	마우스 오른쪽 버튼을 눌렀을 때 나오는 메뉴가 나오기 전에 이벤트 발생

2. 키 이벤트

이벤트	설명
keydown	키를 눌렀을 때 이벤트 발생
keyup	키를 떼었을 때 이벤트 발생
keypress	키를 누른 상태에서 이벤트 발생

3. 폼 이벤트

이벤트	설명
focus	요소에 포커스가 이동되었 때 이벤트 발생
blur	요소에 포커스가 벗어났을 때 이벤트 발생
change	요소에 값이 변경되었을 때 이벤트가 발생
submit	Submit 버튼을 눌렀을 때 이벤트가 발생
reset	Reset 버튼 눌렀을 때 이벤트가 발생
select	Input 이나 textarea 요소안의 텍스트를 드래그하여 선택하였을 때 이벤트 발생

4. 로드 및 기타 이벤트

이벤트	설명
load	페이지의 로딩이 완료되었을 때 이벤트 발생
abort	이미지의 로딩이 중단되었을 때 이벤트 발생
unload	페이지가 다른 곳으로 이동할 때 이벤트가 발생
resize	요소 사이즈가 변경되었을 때 이벤트가 발생
scroll	스크롤바를 움직였을 때 이벤트가 발생

● 이벤트 핸들러

- 이벤트가 발생했을 때 이벤트를 처리하기 위해서 실행되는 함수를 이벤트 처리기 또는 이벤트 리스너 라고 한다.

● 이벤트 핸들러를 등록하는 방법

1. HTML요소의 이벤트 핸들러 속성에 설정하는 방법 (인라인 방식)

```
<태그명  on 이벤트 = '자바스크립트 코드'> </태그명>
```

```
<input type="button" onclick="changeColor();">
```

2. DOM요소 객체의 이벤트 핸들러프로퍼티에 설정하는 방법 (고전방식)

```
1)
객체.on이벤트명 = function() {
      //이벤트 처리기
}

2)
function 함수명() {
      //이벤트 처리기
}
객체.on이벤트명 = 함수명
```

```
var btn=document.getElementById("button");
btn.onclick=changeColor();
```

3. addEventListener메서드를 사용하는 방법 (표준)

target.addEventListener(type, listener, usecapture);

target	이벤트 리스너를 등록할 DOM 노드
type	이벤트 유형
listener	이벤트가 발생했을 때 처리를 담당하는 콜백 함수
useCapture	이벤트 단계 (true : 캡처링, false : 버블링)

```
var btn = document.getElementById("button");
btn.addEventListener("click", changeColor, false);
```

● 이벤트 핸들러 삭제하기

target.removeEventListener(type, listener, useCapture);

● 이벤트 객체

- 해당 이벤트의 다양한 정보를 저장한 프로퍼티와 이벤트의 흐름을 제어하는 메서드를 가지고 있다.

```
function changeColor ( e ) {
        e.currentTarget.style.backgroundColor="red";
}
```

- 인수 e가 이벤트 객체이다.

[이벤트 객체 프로퍼티]

프로퍼티	설명
type	이벤트 유형
target	이벤트가 발생한 요소
currentTarget	이벤트 리스너가 등록된 요소
eventPhase	이벤트 전파 단계 (1: 캡처링, 2 : 타깃, 3 : 버블링)
cancleable	preventDefault()로 기본 이벤트를 취소할 수 있는지 여부 확인
defaultPrevented	preventDefault()로 기본 작업이 취소되었는지 여부 확인 (논리값)

이벤트 리스너에 추가적인 정보 전달하는 방법

[Example]

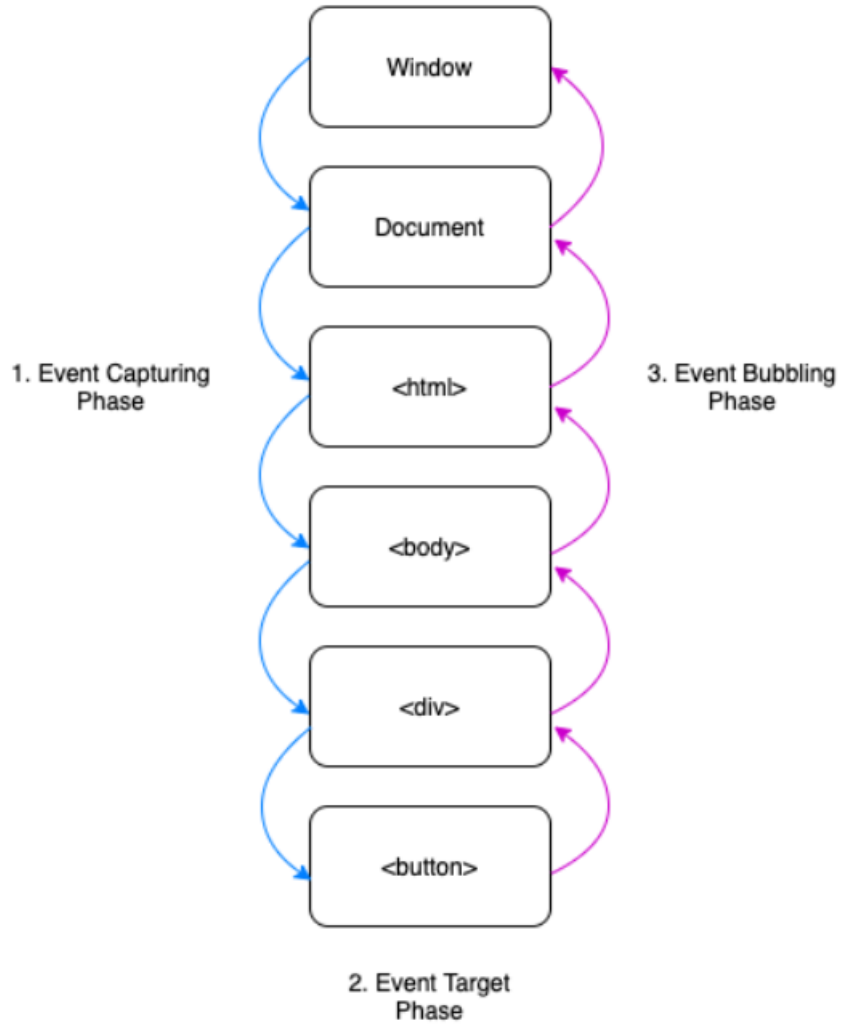
```
<button id="btn"   style='width:100px; height:30px;'>Click me</button>

<script>

        function changeColor(e, color) {
            e.currentTarget.style.backgroundColor = color;
        }

        const box = document.getElementById('btn');
        box.addEventListener('click',   function(e) {
            changeColor(e, 'yellow');
        }, false);

</script>
```

[Example]

```
<button id="btn"   style='width:100px; height:30px;'>Click me</button>

<script>

    function changeColor(color) {
        return function(e) { // e : 이벤트 객체
            e.currentTarget.style.backgroundColor = color;
        }
    }

    const box = document.getElementById('btn');
    box.addEventListener('click', changeColor('yellow'), false);

</script>
```

● 이벤트 전파

- 웹 브라우저에서 HTML 요소에 대한 event가 발생하면 해당 요소에 할당된 이벤트 핸들러가 동작하게 되는데, 이때 handler가 동작하면서 다음과 같이 Bubbling과 Capturing이 발생하게 된다.



1.Bubbling

- 특정 요소에서 event 가 발생했을 때 상위 요소로 event 가 전파되는 것을 말한다.

2. Capturing

- capturing 은 특정 요소에서 event 가 발생했을 때 bubbling 과 반대로 하위 요소로 event 가 전파되는 것을 말한다.

● 이벤트 중단

1. event.stopPropagation()

- event.stopPropagation()은 이벤트 전파를 중단시킨다. 따라서 bubbling 이나 capturing 을 막아야하는 경우에 사용한다.

2. event.preventDefault()

- event.preventDeafult()는 이벤트의 기본 동작을 중단시킨다. html 의 태그들 중에서 <a> 태그 같은 경우엔 지정한 링크로 페이지를 이동하는데 이러한 동작을 중단시킨다.

3. return false

- return false 는 onclick handler 에 사용했을 경우에 대하여 event.preventDefault()를 사용한 것과 같은 동작을 한다. 따라서 이벤트 전파는 발생하지만 기본 동작은 중단되게 된다.

● 이벤트 위임 (Event Delegation)

- 이벤트 위임을 사용하면 상위 요소에 할당한 이벤트 핸들러로 여러 하위 요소를 다룰 수 있다. 이벤트 위임을 사용할 때는 하위 요소마다 이벤트 핸들러를 할당하지 않고, 상위 요소에 하나의 이벤트 핸들러를 할당한다. 이벤트의 버블링 되는 특징을 이용하면 상위 요소에 있는 하나의 이벤트 핸들러만으로 하위 요소에서 발생한 이벤트들을 모두 처리할 수 있다.

이벤트 위임의 장점

- 다수의 이벤트 핸들러를 할당하는 대신에 하나의 이벤트 핸들러만 할당하기 때문에, 코드가 단순해지고 메모리가 절약된다.
- 요소가 추가되거나 제거되는 동작이 많은 경우에도 짧은 코드만으로 이벤트 처리가 가능하다.

이벤트 위임의 단점:

- 이벤트 위임을 사용하기 위해 이벤트가 반드시 버블링 되어야 한다. focus 이벤트처럼 버블링 되지 않는 이벤트나, stopPropagation()을 사용한 경우에는 이벤트 위임을 사용할 수 없다.

ECMAScript 6(2015)

● Default Function Parameter

function 함수이름 (param1 = defaultValue1, ...) { }

· JavaScript 에서 함수의 매개변수는 undefined 가 기본이다. 그러, 일부 상황에서는 다른 기본 값을 설정하는 것이 유용할 수 있다. 이때 바로 기본값 매개변수가 필요할 때 이다.

· undefined 혹은 누락된 파라미터에 대해서만 동작한다.

● Rest Parameter

- ...args 는 맨 마지막에만 올 수 있다.

function 함수이름 (...args) { }

· rest 파라미터는 배열이므로 sort, map, forEach 또는 pop 같은 메서드가 적용될 수 있다.

```
function f(a, b) {
     // arguments
    // 자바스크립트는 함수에서 argument 객체를 사용할 수 있다.
    // arguments 는 유사 배열 객체이므로 배열로 변환하려면 다음과 같다.
     var arr = Array.prototype.slice.call(arguments);
     var arr = Array.from(arguments);      // ES6
}
```

● Arrow Function

· Arrow Function 은 항상 익명이다.

· 자신의 this, arguments, super, new.target 을 바인딩하지 않는다.

· Arrow Function 은 항상 상위 스코프의 this 를 계승받는다.

```
// 매개변수가 여러개인 함수
(param1, param2, ... paramN) => { statement; }
(param1, param2, ... paramN = defaultValueN) => { statement; }
(param1, param2, ... paramN) => { return expression; }
(param1, param2, ... paramN) => expression

// 매개변수가 하나뿐인 함수
(singleParam) => { statement; }
singleParam => { statement; }

// 매개변수가 없는 함수는 괄호가 반드시 필요하다.
() => { statement; }
```

* Arrow funtion 을 쓰면 안되는 경우

1. 메소드

```
const person = {
        userName: 'Kim',
        hello: function() {
            // this : 메소드를 호출한 객체 (person 객체)
            console.log(`Hello, ${this.userName}`);
        }
    }

    person.hello();
```

```
const person = {
        userName: 'Kim',
        hello: ( ) => {
            // Arrow Function 은 항상 상위 스코프의 this 를 계승받는다
            // this :  전역객체 (window)
            console.log(`Hello, ${this.userName}`);
        }
    }

    person.hello();
```

2. prototype

```
// 내부적으로 Object 생성자 함수를 이용하여 객체를 생성한다.
const person = {
        userName: 'Kim'
    }

    // 프로토타입
    Object.prototype.hello = function() {
        // this : person 객체
        console.log(`Hello, ${this.userName}`);
    };

    person.hello();
```

```
const person = {
        userName: 'Kim'
    }

    Object.prototype.hello = ( ) => {
        // Arrow Function 은 항상 상위 스코프의 this 를 계승받는다
        // this : 전역객체 (window)
        // undefined 가 출력됨
        console.log(`Hello, ${this.userName}`);
    }

    person.hello();
```

3. addEventListener 의 콜백 함수

```
const btn = document.getElementById('btn');

btn.addEventListener('click', function() {
        console.log('click');
        // this : 클릭 이벤트를 발생시킨 button 객체
        console.log(this)
});
```

```
const btn = document.getElementById('btn');

btn.addEventListener('click', () => {
      // Arrow Function 은 항상 상위 스코프의 this 를 계승받는다
      // this : 전역객체 (window)
      console.log(this);
});
```

● Template Literals

· 템플릿 리터럴은 내장된 표현식을 허용하는 문자열 리터럴이다.

· 템플릿 리터럴은 이중 따옴표 나 작은 따옴표 대신 백틱(` `) (grave accent) 을 이용한다.

· 템플릿 리터럴은 place holder 를 이용하여 표현식을 넣을 수 있는데 ${expression} 로

표기한다.

```
const name = “Emma”;
console.log(`이름 : ${name}`);
```

● Object Literal Syntax Extension

1. 프로퍼티 초기화 단축 (Property Initializer Shorthand) 기능

```
function createPerson(name, age) {
    return {
        name: name,
        age: age
    }
}
```

```
// 객체의 프로퍼티 이름과 매개변수 이름이 동일한 경우
function createPerson(name, age) {
    return {
        name,
        age
    }
}
```

2. 간결한 메서드 (Concise Method)

```
var person = {
    name: “Emma”,
    sayName: function() {
        console.log(this.name);
    }
};
```

```
var person = {
    name: “Emma”,
    sayName() {
        console.log(this.name);
    }
};
```

3. 프로퍼티의 계산된 이름

```
var lastName = “last name”;

var person = {

    “first name” : “Nicholas”,

    [lastName]: 'Zakas'

}

console.log(person['first name']);

console.log(person[lastName]);
```

● Spread Operator (전개 구문)

1. 함수 호출

myFunction (...이터러블);

```
function myFunc(x, y, z) {   return x + y + z; }
var args = [1, 2, 3];
myFunc(...args);
```

```
function myFunc(v, w, x, y, z) { }
var args = [0, 1];
myFunc(-1, ...args, 2, ...[3]);
```

※ new 에 적용

· new 를 사용해 생성자를 호출 할 때 적용할 수 있다.

```
var dateFields =[2021, 8, 2];
var d = new Date(...dateFields);
```

2. 배열 리터럴

```
var parts = ['shoulders', 'knees'];

var lyrics = ['head', ...parts, 'and', 'toes'];
```

※ 배열 복사

```
var arr = [1, 2, 3];

var arr2 = [...arrr];
```

※ 배열 연결

```
var arr1 = [0, 1, 2];

var arr2 = [3, 4, 5];

arr1 = [...arr1, ...arr2];
```

3. 객체 리터럴

```
var obj1 = { foo:  'bar',  x:  10 };

var obj2 = { foo:  'bar',  y:  20 };


// 객체 복사

var clonedObj = { ... obj1 };


// 객체 병합

var mergedObj = { ...obj1, ...obj2 };
```

● Object || Array Destructuring

· 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 JavaScript 표현식이다.

```
var x = [1, 2, 3, 4, 5];
var [ x, y ] = x;
console.log(x);  // 1
console.log(y);  // 2

var foo = [ ”one”, “two”, “three” ];
var [ red, yellow, green ] = foo;
console.log(red);        // “one”
console.log(yellow);  // “two”
console.log(green);    // “three”

//변수값 교환하기
var a = 1;
var b = 3;
[a, b] = [b, a]
console.log(a);    // 1
```

```
//함수가 반환한 배열 분석
function f() {
    return [1, 2];
}

var a, b;
[a, b] = f();

console.log(a);    // 1
console.log(b);    // 2

//일부 반환값 무시하기
function f() {
    return [1, 2, 3];
}

var a, b;
[a, , b] = f();

console.log(a);    // 1
console.log(b);    // 3
```

객체 구조 분해

```
var obj = { p: 42, q: true };

var {p, q} = obj

console.log(p); // 42

console.log(q); // true
```

for of 반복문과 구조 분해

```
var people = [
    {
        name: 'Mike Smith';
        family: {
            mother: “Jane Smith”,
            father: “Harry Smith”
        }
    },
    {
        name: 'Tome Jones';
        family: {
            mother: “Norah Jones”,
            father: “Richard Jones”
        }
    }
]

for (let { name: n, family: { father: f } } of people) {
    console.log(`Name : ${n}, Father : ${f}`);
}
```

함수 매개변수로 전달된 객체에서 필드 해체하기

```
function userId(id) {
    return id;
}

function whois({dispayName: displayName, fullName: {firstName: name}}) {
    console.log('${displayName} is ${name}');
}

var user = {
    id: 'java',
    displayName: “jode”,
    fullName: {
        firstName: 'John',
        lastName: 'Doe'
    }
};
console.log(`userid : ${userId(user)}`);     // 'java'
console.log(whois);
```

● Promise 객체

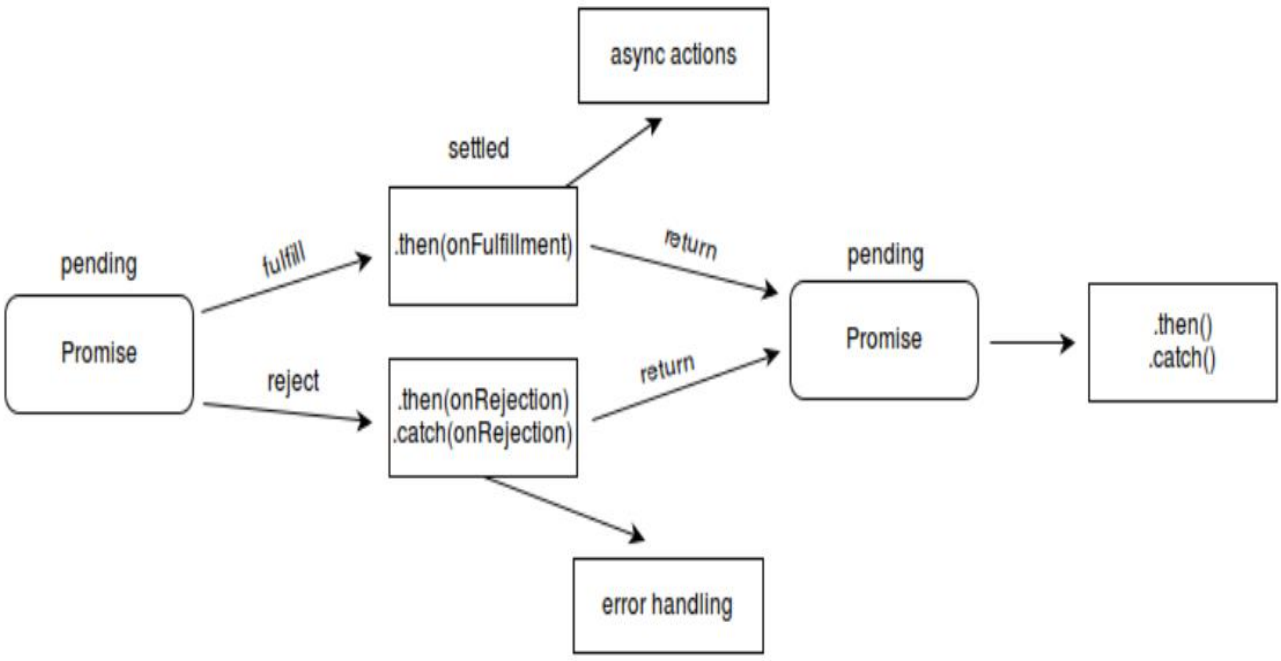
· 비동기 작업의 최종 완료 또는 실패를 나타내는 객체이다.

· Promise 는 다음 중 하나의 상태를 갖는다.

1. 대기(pending): 이행하거나 거부되지 않은 초기 상태.

2. 이행(fulfilled): 비동기 작업이 성공적으로 완료됨.

3. 거부(rejected): 비동기 작업이 실패함.



[Example]

비동기로 음성 파일을 생성해주는 createAudioFileAsync() 라는 함수가 있다고 하자. 해당 함수는 음성 설정에 대한 정보를 받고, 두 가지 콜백 함수를 받습니다. 하나는 음성 파일이 성공적으로 생성되었을때 실행되는 콜백, 그리고 다른 하나는 에러가 발생했을때 실행되는 콜백이다.

```
function successCallback(result) {
    console.log("Audio file ready at URL: " + result);
}

function failureCallback(error) {
    console.log("Error generating audio file: " + error);
}

createAudioFileAsync(audioSettings, successCallback, failureCallback);
```

>>>>

```
// 콜백을 붙여 사용할 수 있게 Promise 객체를 반환한다.
const promise = createAudioFileAsync(audioSettings);
promise.then(successCallback, failureCallback);
```

● Promise Chaining

보통 하나나 두 개 이상의 비동기 작업을 순차적으로 실행해야 하는 상황을 흔히 보게 된다. 순차적으로 각각의 작업이 이전 단계 비동기 작업이 성공하고 나서 그 결과값을 이용하여 다음 비동기 작업을 실행해야 하는 경우를 의미한다. 우리는 이런 상황에서 promise chain 을 이용하여 해결할 수 있다.

```
doSomething(function(result) {
        doSomethingElse(result, function(newResult) {
                doThirdThing(newResult, function(finalResult) {
                        console.log(`finalResult : ${finalResult}`);
                }, failureCallabck);
        }, failureCallback);
}, failureCallback);
```

콜백 함수들을 반환된 promise 에 promise chain 을 형성하도록 추가할 수 있다.

```
doSomething().then(function(result) {
        return doSomethingElse(result);
})
.then(function(newResult)) {
        return doThirdThing(newResult);
})
.then(function(finalResult) {
        console.log(`finalResult : ${finalResult}`);
}
.catch(failureCallabck);
```

● Async / Await

· async function 선언은 AsyncFunction 객체를 반환하는 하나의 비동기 함수를 정의한다.

· 암시적으로 Promise 객체를 사용하여 결과를 반환한다.

```
async function 함수명 ([param, ...] ) {
    // 비동기 처리
}
async function foo() {
    return 1;
}
function foo() {
    return Promise.resolve(1);
}
```

첫번째 await 문을 포함하는 최상위 코드는 동기적으로 실행된다. 따라서 await 문이 없는 async 함수는 동기적으로 실행된다. 하지만 await 문이 있다면 async 함수는 항상 비동기적으로 완료된다.

```
async function foo() {
        statment1;
        statment2;
        var result = await 비동기적 처리 함수(); //Promise.resolve(result)
}
```

[Example]

```
var resolveAfter5Second= function() {
    return new Promise(resolve => {
        setTimeout(function() {
            resolve(20);
        }, 5000);
    });
};

async function() {
   const slow = await resolveAfter5Second(); //Promise 객체를 사용하여 결과(resolve)
반환
    console.log(`slow : ${slow}`);
}
```

● async 함수를 사용한 promise chain 재작성

```
function getProcessData(url) {
    return downloadData(url)      //return a promise
        .catch(e => {
            return downloadFallbackData(url); //return a promise
        })
        .then(v => {
            return processDataInWorker(url); //return a promise
        });
}
```

==========

```
async function getProcessData(url) {
    let v;
    try {
        v = await downloadData(url);
    } catch(e) {
        v = await downloadFallbackData(url);
    }
    return processDataInWorker(v);   //Promise.resolve(processDataInWorker(v));
}
```

● 모듈(ES6 방식)

· 모듈이란 여러 기능들에 관한 코드가 모여있는 하나의 파일이다.

모듈의 특징

1. 유지보수성

· 기능들이 모듈화가 잘 되어있다면, 의존성을 그만큼 줄일 수 있기 때문에 어떤 기능을 개선한다거나 수정할 때 훨씬 편하게 할 수 있다.

2. 네임스페이스화

· 자바스크립트에서 전역변수는 전역공간을 가지기 때문에 코드의 양이 많아질수록 겹치는 네임스페이스가 많아질 수 있다. 그러나 모듈로 분리하면 모듈만의 네임스페이스를 갖기 때문에 그 문제가 해결된다.

3. 재사용성

· 똑같은 코드를 반복하지 않고 모듈로 분리시켜서 필요할 때마다 사용할 수 있다.

모듈 사용법

1. 변수나 함수앞에 export 키워드를 붙이면 외부 모듈에서 해당 변수나 함수에 접근할 수 있다.

```
// 파일명 :  log.js
export const PI = 3.14;
export function log(msg) {console.log(msg);}
```

2. import 키워드를 사용하면 외부 모듈의 기능을 가져올 수 있으며, <script type="module"> 같은 속성을 설정해 해당 스크립트가 모듈이란 걸 브라우저가 알 수 있게 해줘야 한다.

```
<script type='module'>
    import { PI, log } from './modues/log.js'
</script>
```

모듈의 핵심 기능

1. 모듈은 항상 엄격 모드(use strict)로 실행된다.
2. 모듈 레벨 스코프를 갖는다.
3. 동일한 모듈이 여러 곳에서 사용되더라도 모듈은 최초 호출시 단 한번만 실행된다.

● 클래스

· Class 는 객체를 생성하기 위한 템플릿이다.

· 클래스는 데이터와 이를 조작하는 코드를 하나로 추상화한다.

· 자바스크립트에서 클래스는 프로토타입을 이용해서 만들어졌지만 ES5 의 클래스 의미와는 다른 문법과 의미를 갖는다.

1. 클래스 선언

```
class  Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    // 메서드
    getName() {
        return name;
    }
    setName(name) {
        this.name = name;
    }
}
```

2. 객체 생성

```
const p = new Person('홍길동', 10);
```

※ Hosting

- 클래스 선언은 호이스팅이 일어나지 않는다.

Class body 와 메서드 정의

· 클래스의 본문(body)는 strict mode(엄격한 문법이 적용된다.)에서 실행된다.

· constructor(생성자) 메소드는 객체를 생성하고 초기화하기 위한 메소드로 클래스 안에 한개만 존재할 수 있다. (SyntaxError 발생)

· constructor 는 부모 클래스의 constructor 를 호출하기 위해 super 키워드를 사용한다.

정적 메소드와 속성

· static 키워드는 클래스를 위한 정적(static) 메소드를 정의한다.

· 정적 메소드는 클래스의 인스턴스화 없이 호출된다.

· 정적 메소드는 어플리케이션을 위한 유틸리티 함수를 생성하는 데 주로 사용한다.

extends 를 통한 클래스 상속

```
class Dog extends Animal {
    constructor(name) {
        super(name);
    }
}
```