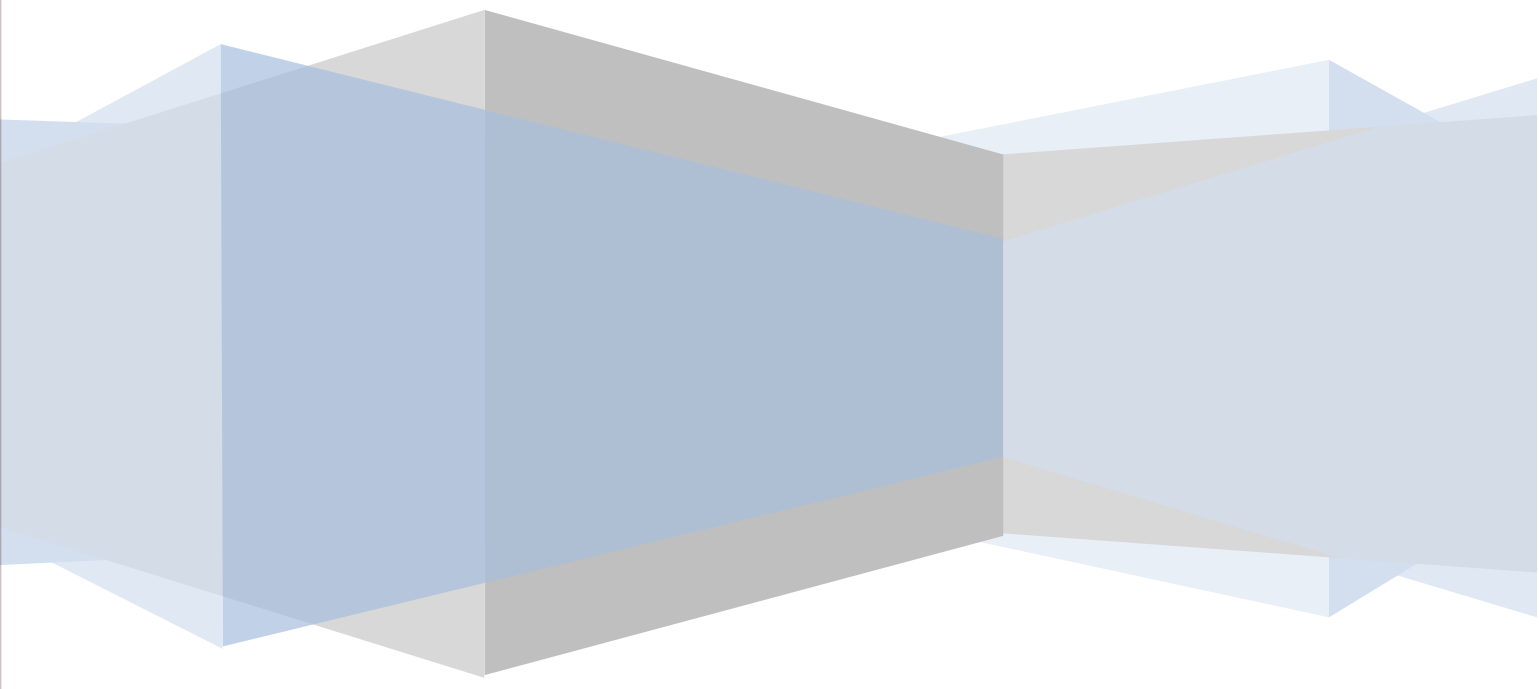
MEGAZONE CLOUD

JAVA 개발자 양성과정

다이나믹 웹 문서 제작

JavaScript & jQuery



자바스크립트 소개

· 객체 기반의 스크립트 프로그래밍 언어 이다. 이 언어는 웹 브라우저 내에서 주로 사용하며, 다른 응용 프로그램의 내장 객체에도 접근할 수 있는 기능을 가지고 있다. 또한 Node.js 와 같은 런타임 환경과 같이 서버 프로그래밍에도 사용되고 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 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( )

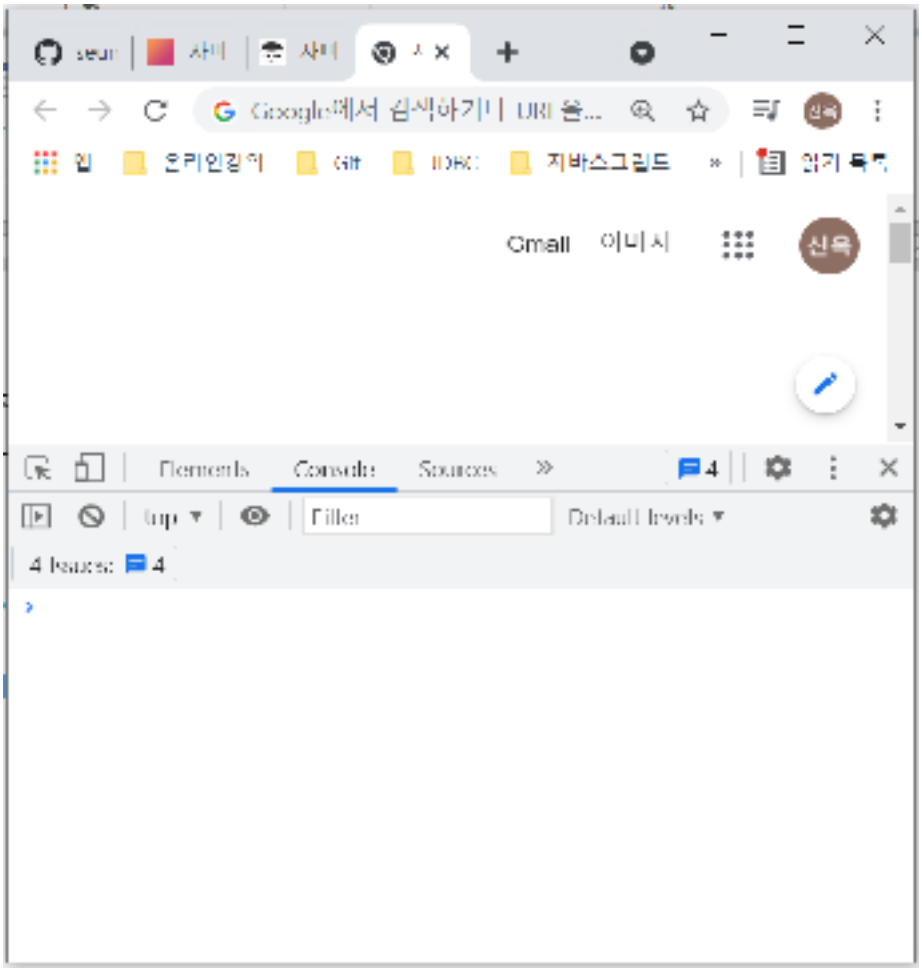
- 자바스크립트 console.log( ) 함수는 인수로 설정한 값을 디버거 콘솔에 표시하는 함수이다.

- 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';

- Hoisting 된다.

- 함수 유효 범위를 갖는다.

2. let (added in ECMAScript 6)

- 블록{ } 유효 범위를 갖는다.

- Hoisting 되지 않는다.

3. const (added in ECMAScript 6)

- 블록 범위의 상수를 선언한다.

※ Hoisting 이란?

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

(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

2. 참조 타입

- 객체

- Function

- 배열

※ 자료형 확인하기

```
var name = “Emma”;

console.log( typeof  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

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; ]    //optioinal
}
```

2. 함수 표현식(리터럴) : 익명 함수 표현식

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

3. 중첩함수 (Nested Function)

· 다른 함수 내부에서 정의되는 함수를 의미한다.

· 캡슐화를 구현할 수 있다.

```
function outter( ) {
   const title = "javascript";
   function inner( ) {
      console.log(`title : ${title}`);
   }
   return inner;
}
-----------------------------------------------------------------
const inner = outter();
inner();
```

※ 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]

```
<button onclick=”print”>click</button>

//즉시 실행함수 (immediate function) : 함수 선언 및 실행
var click = (function() {
   var count = 0;
   return function() {
      retrun ++count;
   }
})();

function print() {
   console.log(click( )); //내부 함수 호출
}
```

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

4. 콜백함수 (Callback Function)

· 콜백 함수는 코드를 통해 명시적으로 호출하는 함수가 아니라, 함수를 등록하면 어떤 이벤트가 발생했거나 특정 시점에 도달했을 때 시스템에서 호출(Called back)되는 함수이다.

※ 비동기적 처리를 하는 경우

1. 사용자 이벤트 처리
2. 네트워크 응답 처리
3. 파일을 읽고 쓰는 등의 파일 시스템 작업
4. 의도적으로 시간 지연을 사용하는 기능

```
// 매개변수로 전달되는 함수를 콜백함수라고 한다.
function add (a, b, callback) {
   callback(a + b);  // 콜백 함수 호출
}

function printConsole ( result ) {
   console.log(result);
}
add(1, 2, printConsole);
```

내장 객체

1. Object 객체

· 자바스크립트의 최상위 객체이다.

● Object 객체 생성

var obj = { };

var obj = new Object( );

● Object 객체의 메소드

hasOwnProperty(name) : 객체가 name 속성을 가지고 있는지 확인

isPrototypeof(object) : 객체가 object 의 프로토타입인지 검사

propertyIsEnumerable(name) : 반복문으로 열거 가능 여부 확인

toString() : 객체를 문자열로 변경

valueOf() : 객체의 값을 표시

2. String 객체

· 문자열을 표현할 때 사용하는 객체이다.

● String 객체의 속성

length : 문자열의 길이

● String 객체의 메소드

charAt(position) : 해당 인덱스 문자 반환

charCodeAt(position) : 해당 인덱스 문자를 유니코드로 반환

concat(args) : 매개변수로 입력한 문자열을 결합

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

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

match(regExp) : 문자열 안에 regExp 가 있는지 확인

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

search(regExp) : regExp 와 일치하는 문자열의 위치 반환

slice(start, end) : 특정 위치의 문자열을 반환

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

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

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

toLowerCase() : 문자열을 소문자로 바꾸어 반환

toUpperCase() : 문자열을 대문자로 바꾸어 반환

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

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

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

3. 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.

4. 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

```
array.forEach(function(element, index, array) {
                                    // 배열요소,   인덱스,   배열자체
});
```

map() : 기존의 배열에 특정 규칙을 적용해서 새로운 배열 생성

· 조건 메소드

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

```
array = array.filter(function(element, index, array) {
   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 반환

· 연산 메소드

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

reduceRight() : 위와 같으나 오른쪽부터 실행

* Array 객체에는 특정 요소를 제거하는 메소드가 없다.

```
Array.prototype.remove = function(i) {
    this.splice(i, 1);
}
intarr = [1, 2, 3, 4, 5];
intarr.remove(2);
console.log(intarr);
```

5. 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)

6. 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.

7. 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.

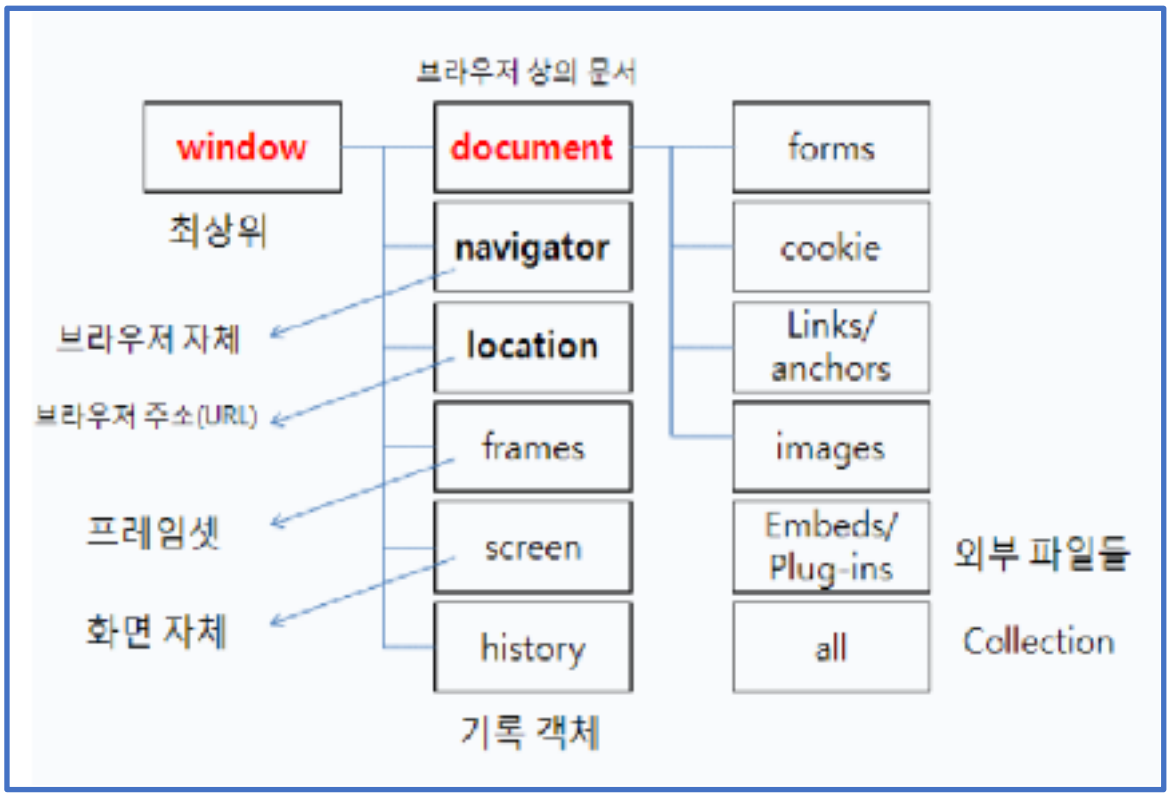
jSON.parse() : JSON 문자열을 자바스크립트 값이나 객체로 변환하여 반환한다.

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 웹브라우저가 제공하는 객체이다.



1. window 객체

· 브라우저 기반 자바스크립트의 최상위 객체이다.

· var 키워드로 변수를 선언하거나 함수를 선언하면, 다 이 window 객체의 프로퍼티가 된다.

· alert(), prompt() 등 많은 메소드를 가지고 있음

● window 객체의 메소드

2. location 객체

· 웹브라우저의 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)

3. navigator 객체

· 웹페이지를 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

문서 객체 모델(DOM)

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

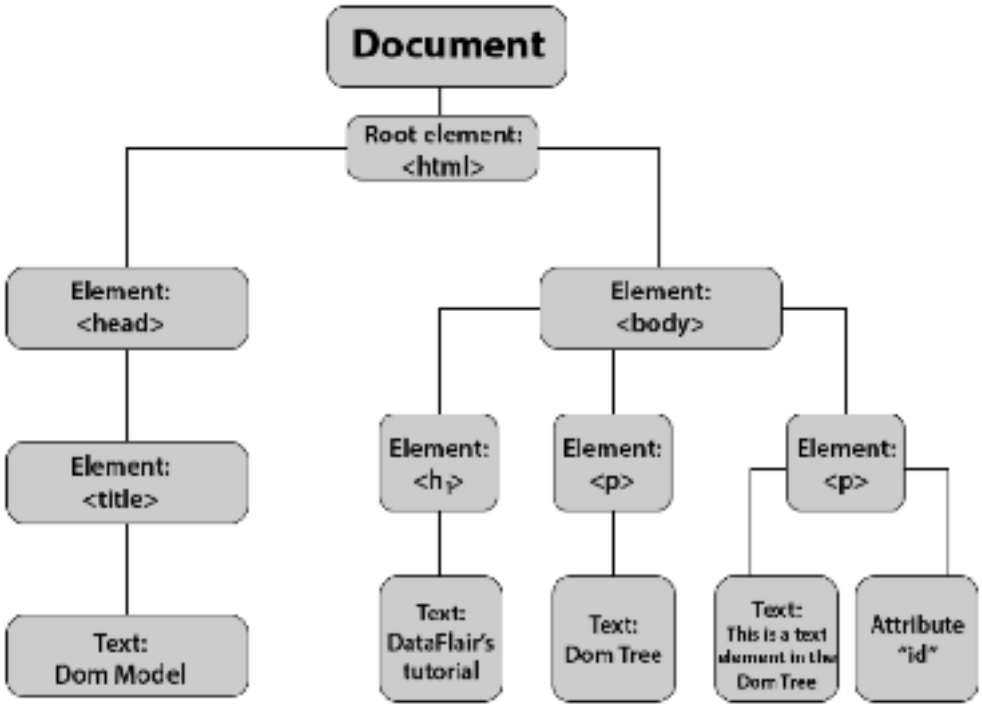
● DOM(Document Object Model) 이란?

· 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.



● 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.

또한, 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)

● DOM 이벤트 처리

1. 이벤트 등록

· 자바스크립트 이벤트 처리는 addEventListener() 함수에서 시작한다.

```
[Element Node].addEventListener('이벤트타입', function(e) { //event 객체
      // Event handler , 콜백함수
});
```

1. 이벤트타입

· 사용자가 웹페이지에서 발생시킨 액션이다. (click, keydown ...)

2. 콜백 함수

· 이벤트가 발생하면 자동으로 실행되는 함수이다. 파라미터로 이벤트 객체가 전달되며, 이벤트 객체에 이벤트를 발생시킨 HTML 요소(타겟)와, 다양한 이벤트 정보가 들어있다.

※ 이벤트타입

이벤트명	타입	대상	발생 시점
DOMContentLoaded	이벤트	DOM	DOM 트리가 생성된 직후(렌더링 전) HTML만 로딩 완료된 상태
load	UI이벤트	도큐먼트, 엘리먼트	리소스와 그 외 리소스 의존 리소스가 모두 로딩 완료되었을 때
reset	이벤트	폼	폼 리셋 버튼 클릭
submit	이벤트	폼	폼 전송 버튼 클릭, 또는 전송 이벤트 발생
resize	UI이벤트	도큐먼트뷰	브라우저 크기 변경
scroll	UI이벤트	도큐먼트뷰, 엘리먼트	브라우저 화면 스크롤, 또는 엘리먼트 요소 내부 스크롤
focus	포커스이벤트	엘리먼트	엘리먼트가 포커스를 얻었을 때(이벤트 버블링 안됨)
blur	포커스이벤트	엘리먼트	엘리먼트가 포커스를 잃었을 때(이벤트 버블링 안됨)
keydown	키보드이벤트	입력요소	키보드 키 누름
keypress	키보드이벤트	입력요소	키보드 키 누를 상태 지속(반복 발생함). Fn, Caps Lock 은 제외
keyup	키보드이벤트	입력요소	키보드 키 누름 해제
mouseenter	마우스이벤트	엘리먼트	엘리먼트 안으로 마우스 커서가 들어옴(버블링 안됨)
mouseover	마우스이벤트	엘리먼트	엘리먼트 위에 마우스 커서가 위치함
mousemove	마우스이벤트	엘리먼트	마우스 커서가 이동
mousedown	마우스이벤트	엘리먼트	마우스 버튼 누름
mouseup	마우스이벤트	엘리먼트	마우스 버튼 누름 해제
click	마우스이벤트	엘리먼트	마우스 버튼 클릭
contextmenu	마우스이벤트	엘리먼트	컨텍스트메뉴 표시(오른쪽 마우스 버튼 클릭)

2. 이벤트 삭제

이벤트를 삭제하려면 등록한 이벤트 타입과 콜백함수 이름을 알아야 한다.

```
document.removeEventListener(이벤트타입, 등록된 콜백함수);
```

ECMAScript 6

● 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.

● Rest Parameter

```
function 함수이름 (...args) { }
```

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

```
function f(a, b) {
   // arguments -->> 배열로 변환
   var arr = Array.prototype.slice.call(arguments);

}
```

Arrow Function

- Arrow Function 은 항상 익명이다.
- 자신의 this , arguments 을 바인딩하지 않는다.

```
// 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; }
```

● 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(...iterableObj);

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

```
//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

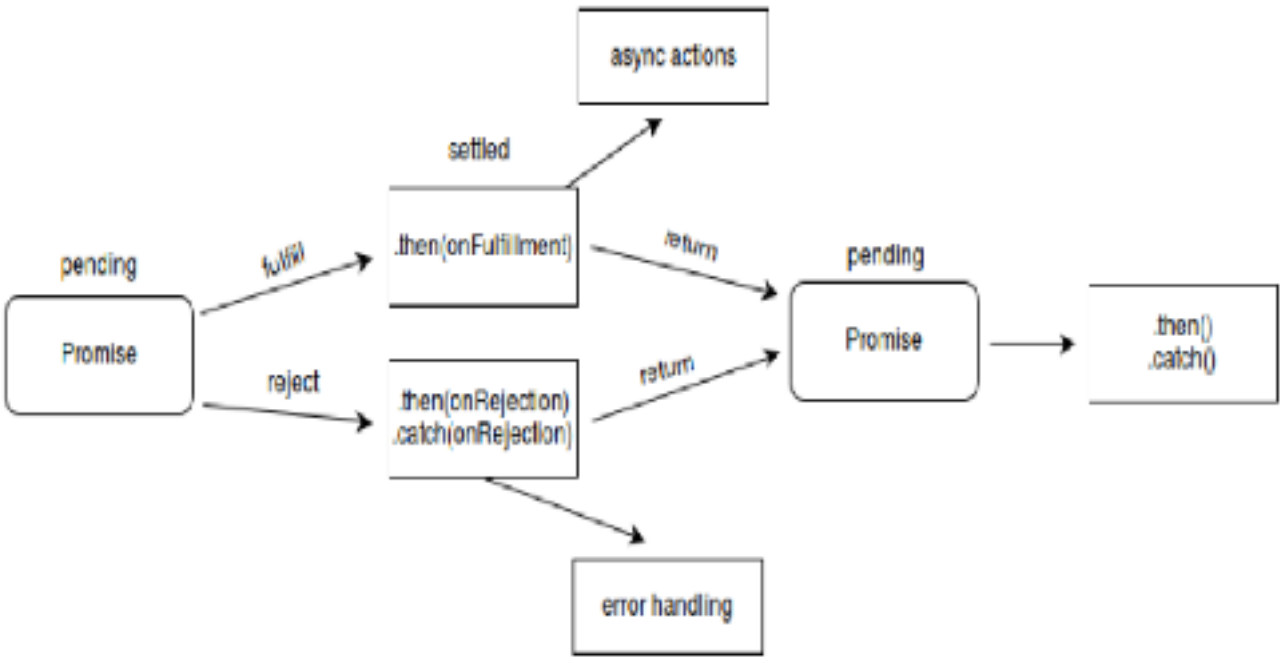
```
function userId(id) {
   return id;
}

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);
   }
}
```