Web

자바스크립트

- 명령형(imperative), 함수형(functional), 프로토타입 기반(prototype-based) 객체지향 프로그래밍을 지원하는 멀티 패러다임 프로그래밍 언어
- HTML로는 웹의 내용을 작성하고, CSS로는 웹을 디자인하며, 자바스크립트로는 웹의 동작을 구현 가능
- 인터프리터 언어
- 타입을 명시하지 않음
- ECMAScript

ECMAScript

버전	출시 년도	특징
ES1	1997	초판
ES2	1998	ISO/IEC 16262 국제 표준과 동일한 규격을 적용
ES3	1999	정규 표현식, try...catch 예외 처리
ES5	2009	HTML5와 함께 출현한 표준안. JSON, strict mode, 접근자 프로퍼티(getter, setter), 향상된 배열 조작 기능(forEach, map, filter, reduce, some, every)
ES6 (ECMAScript 2015)	2015	let, const, class, 화살표 함수, 템플릿 리터럴, 디스트럭처링 할당, spread 문법, rest 파라미터, Symbol, Promise, Map/Set, iterator/generator, module import/export
ES7 (ECMAScript 2016)	2016	지수(**) 연산자, Array.prototype.includes, String.prototype.includes
ES8 (ECMAScript 2017)	2017	async/await, Object 정적 메소드(Object.values, Object.entries, Object.getOwnPropertyDescriptors)
ES9 (ECMAScript 2018)	2018	Object Rest/Spread 프로퍼티

관련 기술

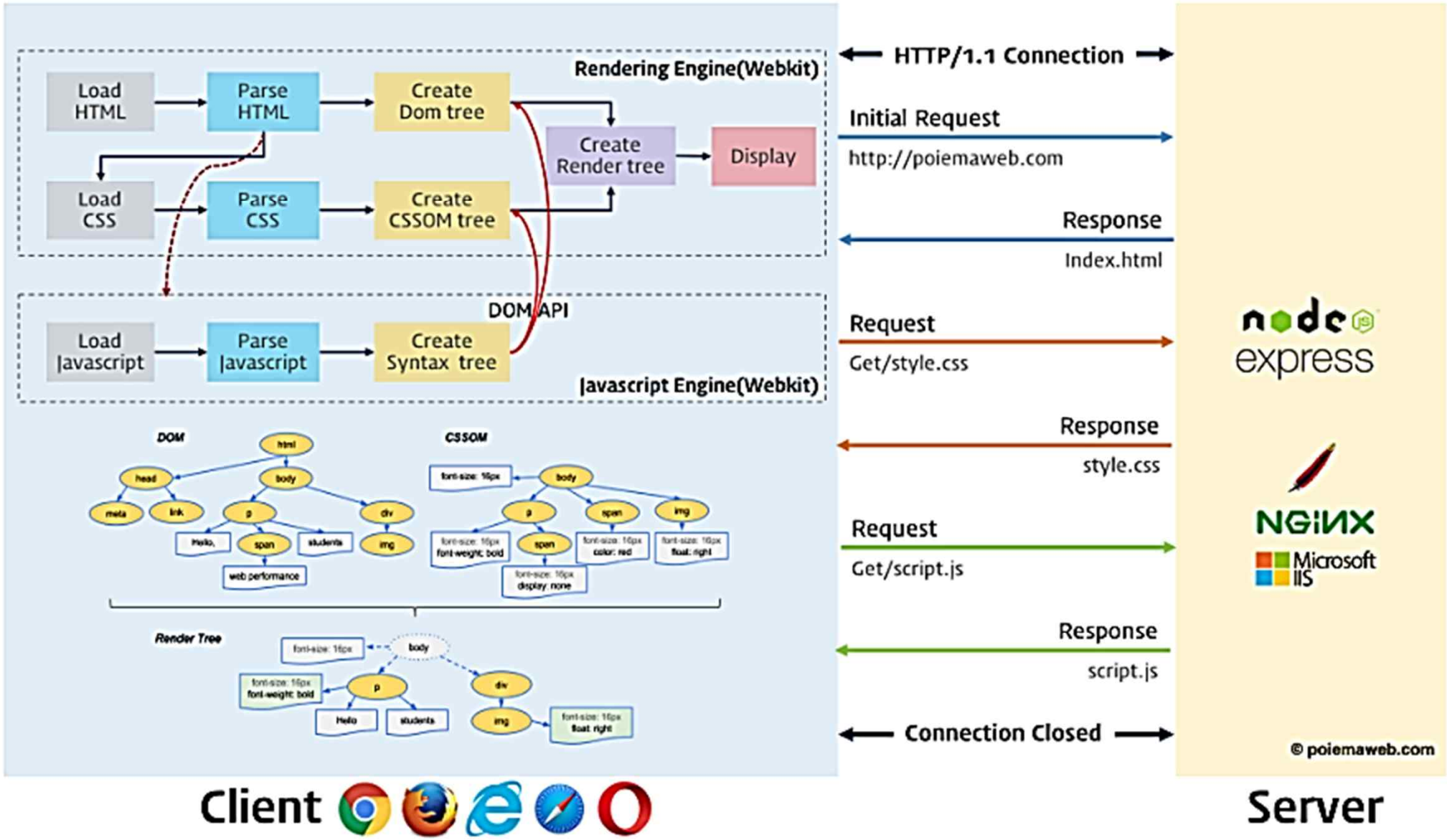
- Ajax(Asynchronous JavaScript and XML)
  - 1999년, 자바스크립트를 이용해서 비동기적(Asynchronous)으로 서버와 브라우저가 데이터를 교환할 수 있는 통신 기능
- jQuery
  - 2006년, 다소 번거롭고 논란이 있던 DOM(Document Object Model)을 보다 쉽게 제어할 수 있게 됨
  - 크로스 브라우징 이슈도 어느 정도 해결
- Node.js
  - 2009년, 브라우저에서만 동작하던 자바스크립트를 브라우저 이외의 환경에서 동작시킬 수 있는 자바스크립트 실행 환경
  - 자바스크립트는 웹 브라우저를 벗어나 서버 사이드 애플리케이션 개발에서도 사용되는 범용 프로그래밍 언어로 발전

웹 브라우저

- 개발자 도구의 활용
  - 크롬브라우저(V8자바스크립트엔진)

패널	설명
Elements	로딩된 웹 페이지의 DOM과 CSS를 편집하여 렌더링된 뷰를 확인해 볼 수 있다. 단, 편집한 내용이 저장되지는 않는다. 웹 페이지가 의도된 대로 렌더링되지 않았다면 이 패널을 확인하여 유용한 힌트를 얻을 수 있다.
Console	로딩된 웹 페이지의 에러를 확인하거나 자바스크립트 소스코드에 포함시킨 console.log 메소드의 결과를 확인해 볼 수 있다.
Sources	로딩된 웹 페이지의 자바스크립트 코드를 디버깅할 수 있다.
Network	로딩된 웹 페이지에 관련한 네트워크 요청(request) 정보와 퍼포먼스를 확인할 수 있다.
Application	웹 스토리지, 세션, 쿠키를 확인하고 관리할 수 있다.

동작원리


Rendering Engine(Webkit)
Load HTML
Parse HTML
Create Dom tree
Create Render tree
Display
Load CSS
Parse CSS
Create CSSOM tree
DOM API
Load Javascript
Parse Javascript
Create Syntax tree
Javascript Engine(Webkit)
DOM
CSSOM
Render Tree
HTTP/1.1 Connection
Initial Request
http://poiemaweb.com
Response
Index.html
Request
Get/style.css
Response
style.css
Request
Get/script.js
Response
script.js
Connection Closed
node
express
NGINX
Microsoft IIS
© poiemaweb.com
Client
Server

선언

- 내부 선언 방식

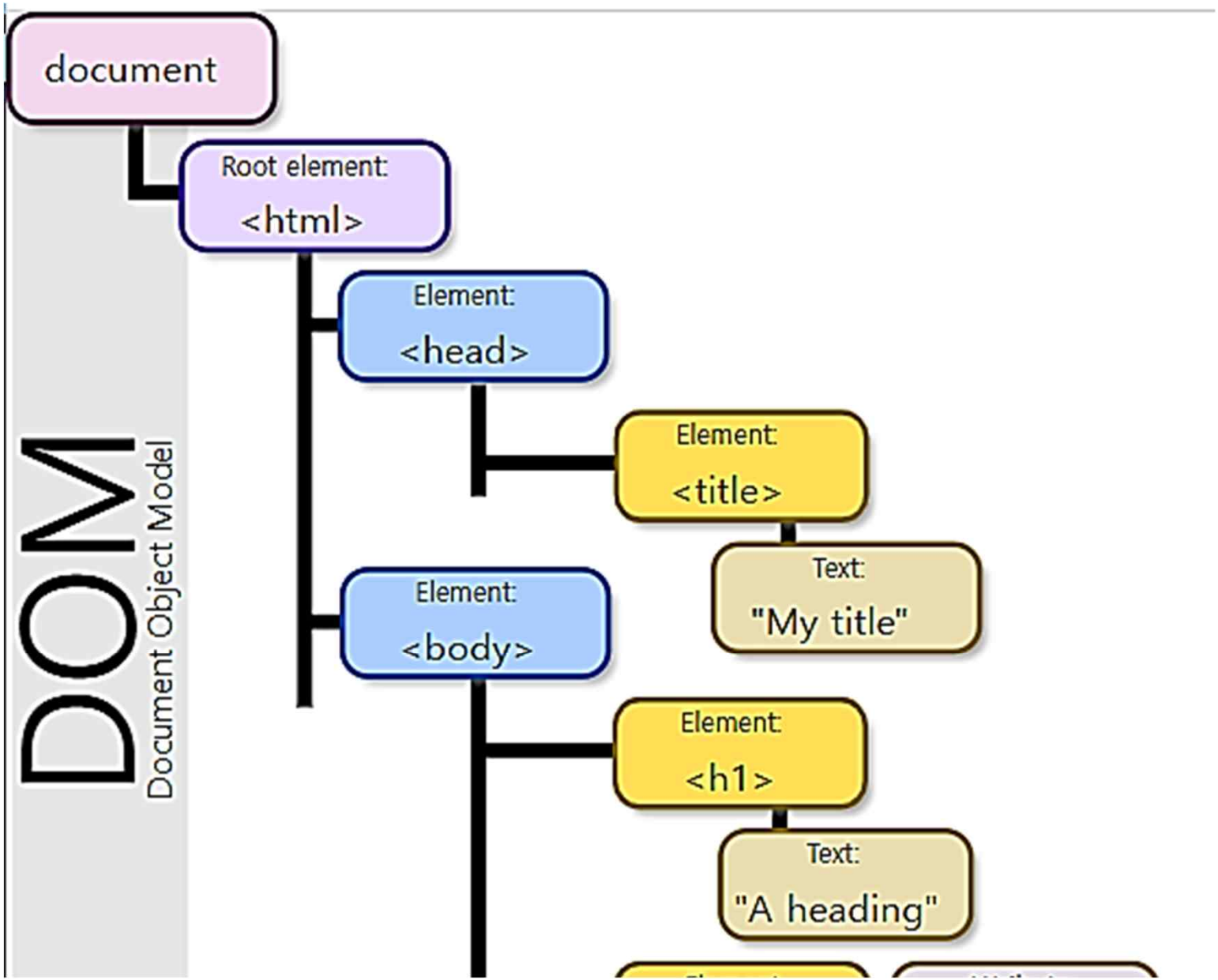
```
<script>
  자바 스크립트 코드
</script>
```

- 외부 선언 방식

```
<script src="자바스크립트 파일 경로"></script>
```

- 위치
  - <head>
  - <body>
  - both

DOM


DOM
Document Object Model
document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"

DOM 요소 찾기

- HTML 태그 이름(tag name)을 이용한 선택
- 아이디(id)를 이용한 선택
- 클래스(class)를 이용한 선택
- name 속성(attribute)을 이용한 선택
- CSS 선택자(selector)를 이용한 선택
- HTML 객체 집합(object collection)을 이용한 선택

출력문

- innerHTML
  - 해당 HTM요소 안에 출력
- **document.write()**
  - **HTML문서에 출력**
- window.alert()
  - alert 박스 안에 출력
- **console.log()**
  - **브라우저 콘솔에 출력**

주석

- 주석(Comment)은 작성된 코드의 의미를 설명하기 위해 사용
- 한줄 주석
  - // 다음에 작성
- 여러 줄 주석
  - /*과 */의 사이에 작성
- 주석은 해석기(parser)가 무시하며 실행되지 않는다.

기본문법

변수(variable)

- 변하는 값(value)을 저장(할당)하고 그 저장된 값을 참조하기 위해 사용하는 메모리 공간
- 메모리 상의 주소(address)를 기억하는 저장소
- 메모리 주소 접근을 위해 사람이 이해할 수 있는 언어로 지정한 식별자(identifier)

```
var x; // 변수의 선언
x = 6; // 정수값의 할당
```

값(value)

```
var str = 'Hello World';
```

- str이라는 이름의 변수를 선언하고 문자열 리터럴 'Hello World'를 값으로 할당

용어	의미
데이터 타입(Data Type)	프로그래밍 언어에서 사용할 수 있는 값의 종류
변수(Variable)	값이 저장된 메모리 공간의 주소를 가리키는 식별자(identifier)
리터럴(literal)	소스코드 안에서 직접 만들어 낸 상수 값 자체를 말하며 값을 구성하는 최소 단위

리터럴 표기법(literal notation)

숫자리터럴	10.50 1001
문자열 리터럴	'Hello' "World"
불리언 리터럴	true false
null 리터럴	null
undefined 리터럴	undefined
객체 리터럴	{name:'Lee',gender:'male'}
배열 리터럴	[1, 2, 3]
정규표현식 리터럴	/AB/ /ab/gi
함수 리터럴	function() {}

변수 호이스팅(Variable Hoisting)

- var 선언문이나 function 선언문 등 모든 선언문이 해당 Scope의 선두로 옮겨진 것처럼 동작하는 특성
- 모든 선언문(var, let, const, function, function*, class)이 선언되기 이전에 참조 가능

1. 선언 단계(Declaration phase)
   - 변수 객체(Variable Object)에 변수를 등록
   - 스코프가 참조하는 대상
2. 초기화 단계(Initialization phase)
   - 등록된 변수를 메모리에 할당
   - 변수는 undefined로 초기화
3. 할당 단계(Assignment phase)
   - undefined로 초기화된 변수에 실제값을 할당

```
console.log(foo);
var foo = 123;
console.log(foo);
{
  var foo = 456;
}
console.log(foo);
function test(){
  console.log(xoo);
  var xoo=1000;
  console.log(xoo);
};
test();
console.log(foo,xoo);
```

스코프(scope)

- 참조 대상 식별자(identifier, 변수, 함수의 이름과 같이 어떤 대상을 다른 대상과 구분하여 식별할 수 있는 유일한 이름)를 찾아내기 위한 규칙
- 전역 스코프 (Global scope)
  - 코드 어디에서든지 참조
- 지역 스코프 (Local scope or Function-level scope)
  - 함수 코드 블록이 만든 스코프로 함수 자신과 하위 함수에서만 참조
- 전역 변수 (Global variable)
  - 전역에서 선언된 변수이며 어디에든 참조
- 지역 변수 (Local variable)
  - 지역(함수) 내에서 선언된 변수이며 그 지역과 그 지역의 하부 지역에서만 참조

```
var global = 'global';
function foo() {
  var local = 'local';
  console.log(global);
  console.log(local);
}
foo();
console.log(global);
console.log(local);
// Uncaught
ReferenceError: local
is not defined
```

ES6 let const

- let
  - 재선언 불가
  - 블록 레벨 스코프
  - 선언 이전에 사용 불가
- const
  - 재선언 불가 / 재할당 불가
  - 블록 레벨 스코프

scope

- 자바스크립트
  - 함수 레벨 스코프(function-level scope)
- 블록 레벨 스코프(block-level scope)
  - ECMAScript 6에서 도입된 let keyword를 사용

함수 레벨 스코프(Function-level scope)
함수 내에서 선언된 변수는 함수 내에서만 유효하며 함수 외부에서는 참조할 수 없다. 즉, 함수 내부에서 선언한 변수는 지역 변수이며 함수 외부에서 선언한 변수는 모두 전역 변수이다.
블록 레벨 스코프(Block-level scope)
모든 코드 블록(함수, if 문, for 문, while 문, try/catch 문 등) 내에서 선언된 변수는 코드 블록 내에서만 유효하며 코드 블록 외부에서는 참조할 수 없다. 즉, 코드 블록 내부에서 선언한 변수는 지역 변수이다.

데이터 타입

- 원시 타입 (primitive data type)
  - number
  - string
  - boolean
  - null
  - undefined
  - symbol (New in ECMAScript 6)
- 객체 타입 (Object data type)
  - object

타입 변환

- 동적 타이핑
  - 자바스크립트는 C나 Java외는 다르게 변수를 선언할 때 데이터 타입을 미리 지정하지 않음.
- 한 변수에 여러 번 대입은 가능하지만 재선언은 무시
- 묵시적 타입 변환(implicit type conversion)
- 명시적 타입 변환(explicit type conversion)
  - Number(), String(), Boolean(), Object(), parseInt(), parseFloat()

연산자(Operator)

- 피연산자(Operand)

Operand	Operator	Operand
100	+	50

- 산술연산자(Arithmetic Operator)

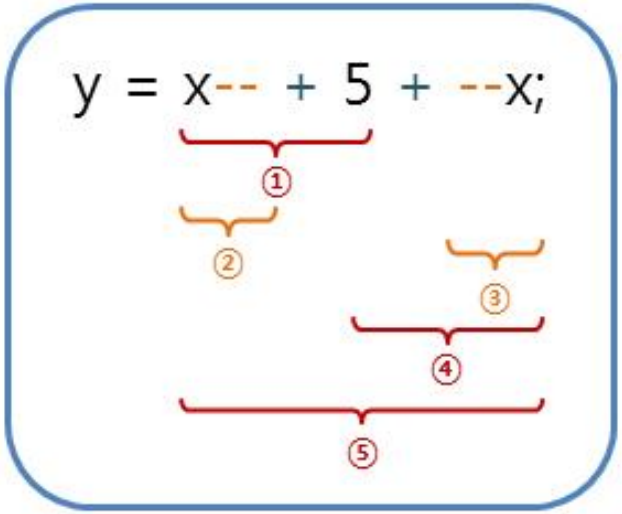
Operator	Description
+	더하기 Addition
-	빼기 Subtraction
*	곱하기 Multiplication
**	거듭제곱 Exponentiation (ES2016)
/	나누기 Division
%	나머지 Modulus (Remainder)

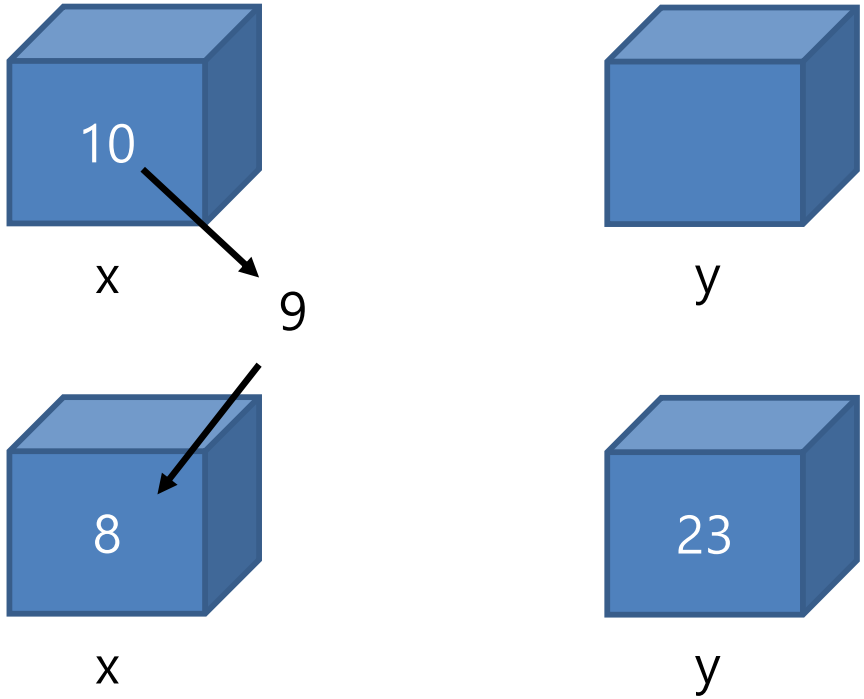
대입 연산자(assignment operator)

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

증감 연산자(increment and decrement operator)

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.





비교연산자(comparison operator)

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.

논리 연산자(logical operator)

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)

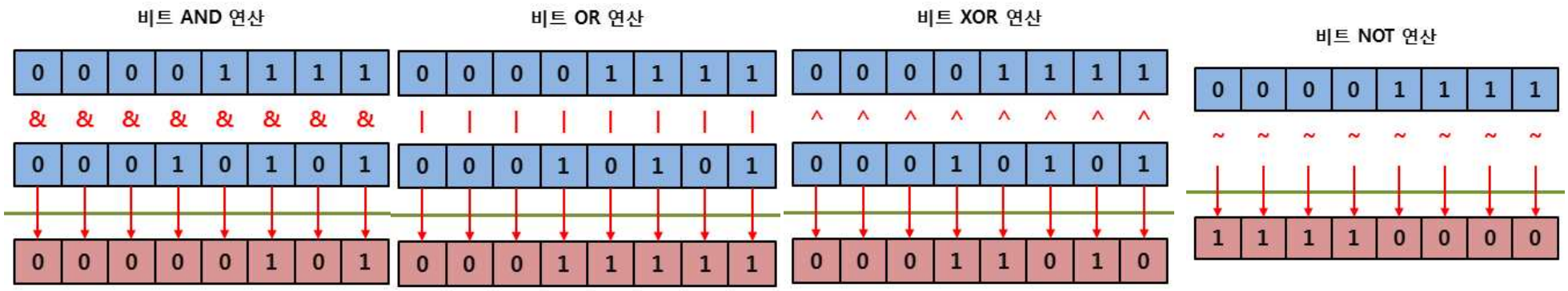
A	B	A && B	A || B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

단축평가

단축 평가 표현식	평가 결과
true || anything	true
false || anything	anything
true && anything	anything
false && anything	false

비트 연산자(bitwise operator)

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.



삼항 연산자(ternary operator)

문법
표현식 ? 반환값1 : 반환값2

물음표(?) 앞의 표현식에 따라
결과값이 참이면 반환값1을 반환
결과값이 거짓이면 반환값2를 반환

타입의 판단

- typeof 연산자
  - 피연산자의 타입을 반환

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

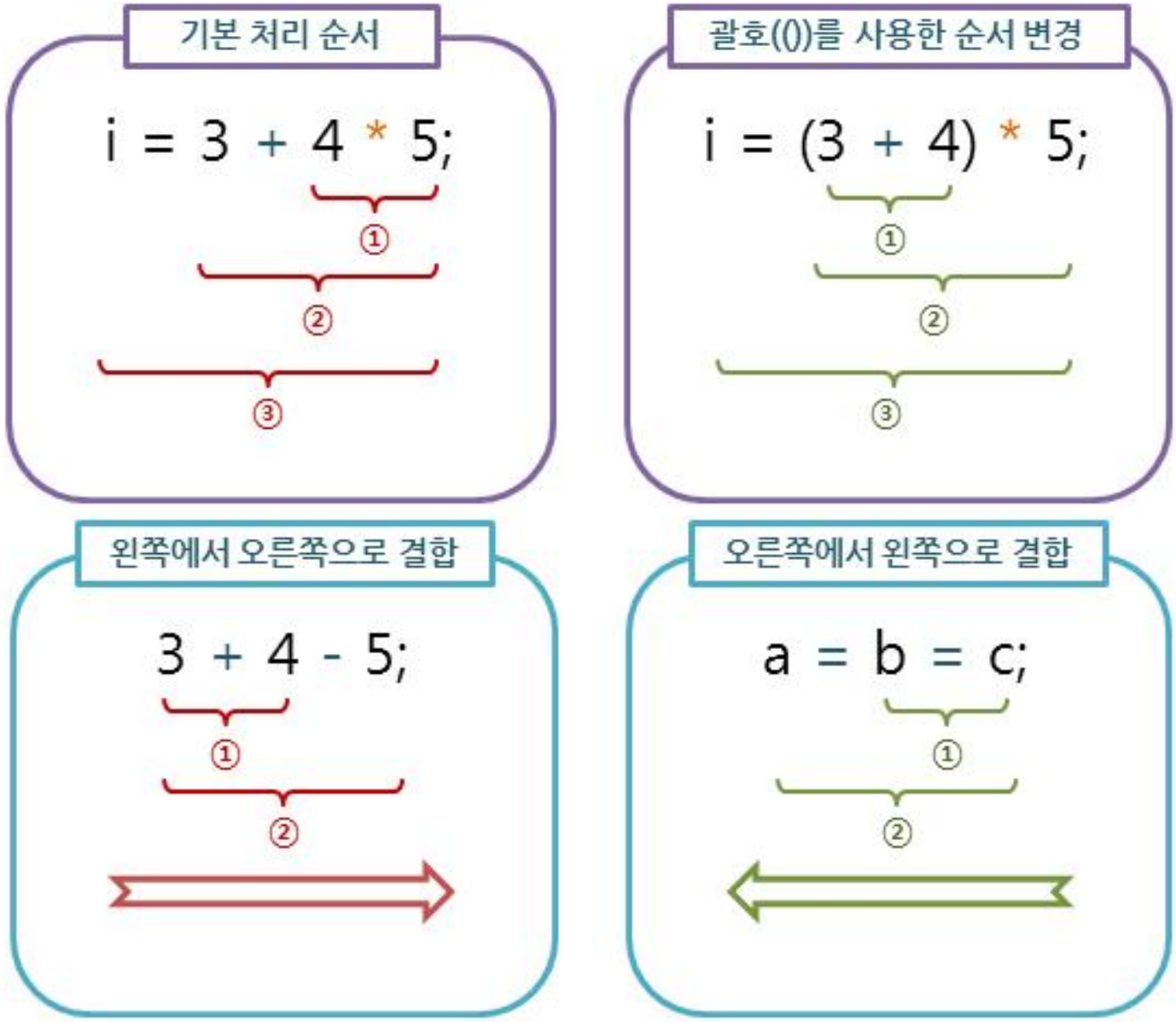
- instanceof 연산자
  - 피연산자인 객체가 특정 객체의 인스턴스인지 아닌지를 확인

void 연산자

- void 연산자는 피연산자로 어떤 타입의 값이 오던지 상관없이 언제나 undefined 값만을 반환

```
<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>
```

연산자의 우선순위(operator precedence)



우선순위표

우선순위	연산자	설명	결합 방향
1	()	묶음(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
	-	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
7	+	덧셈 연산자 (이항 연산자)	왼쪽에서 오른쪽으로
	-	뺄셈 연산자 (이항 연산자)	왼쪽에서 오른쪽으로

우선순위	연산자	설명	결합 방향
8	<<	비트 왼쪽 시프트 연산자	왼쪽에서 오른쪽으로
	>>	부호 비트를 확장하면서 비트 오른쪽 시프트	왼쪽에서 오른쪽으로
	>>>	부호 비트를 확장하지 않고 비트 오른쪽 시프트	왼쪽에서 오른쪽으로
9	<	관계 연산자(보다 작은)	왼쪽에서 오른쪽으로
	<=	관계 연산자(보다 작거나 같은)	왼쪽에서 오른쪽으로
	>	관계 연산자(보다 큰)	왼쪽에서 오른쪽으로
	>=	관계 연산자(보다 크거나 같은)	왼쪽에서 오른쪽으로
	instanceof	인스턴스 여부 판단	왼쪽에서 오른쪽으로
10	==	동등 연산자	왼쪽에서 오른쪽으로
	===	일치 연산자	왼쪽에서 오른쪽으로
	!=	부등 연산자	왼쪽에서 오른쪽으로
	!==	불일치 연산자	왼쪽에서 오른쪽으로
11	&	비트 AND 연산자	왼쪽에서 오른쪽으로
12	^	비트 XOR 연산자	왼쪽에서 오른쪽으로
13	|	비트 OR 연산자	왼쪽에서 오른쪽으로
14	&&	논리 AND 연산자	왼쪽에서 오른쪽으로
15	||	논리 OR 연산자	왼쪽에서 오른쪽으로
16	? :	삼항 연산자	오른쪽에서 왼쪽으로
17	=	대입 연산자 (=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=)	오른쪽에서 왼쪽으로
18	...	전개	-
19	,	쉼표 연산자	왼쪽에서 오른쪽으로

입력

- Prompt
  - window.prompt(message,default);
  - 문자열입력
- confirm
  - window.confirm(message);
  - Boolean 값 입력

제어문(control flow statements)

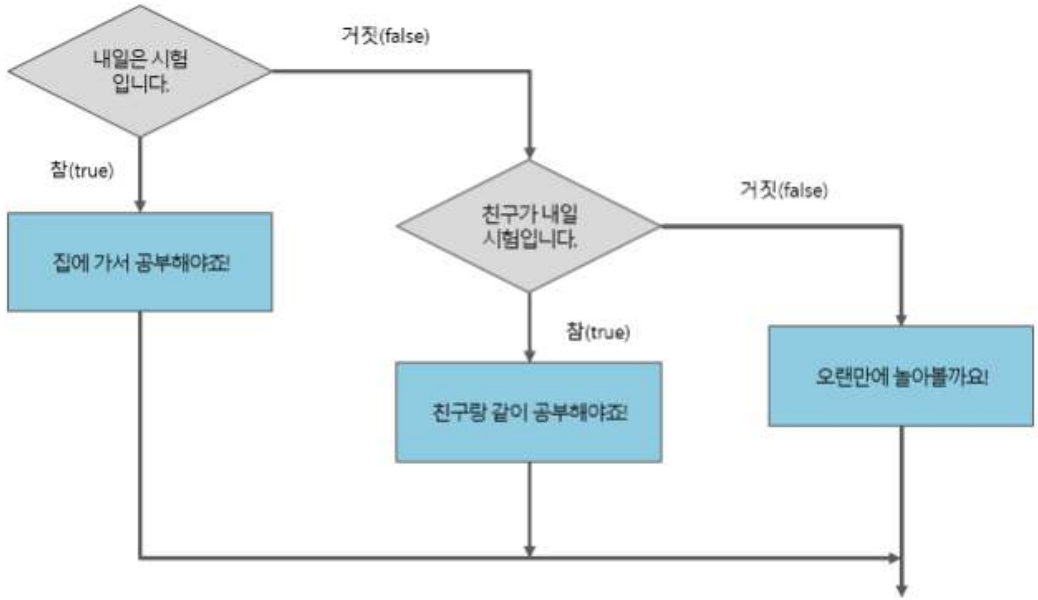
- 프로그램의 순차적인 흐름을 제어해야 할 때 사용하는 실행문
- 조건문(conditional statements)
  - 프로그램 내에서 주어진 표현식의 결과에 따라 별도의 명령을 수행하도록 제어하는 실행문
    - if 문, if / else 문, if / else if / else 문, switch 문
- 반복문(iteration statements)
  - 프로그램 내에서 똑같은 명령을 일정 횟수만큼 반복하여 수행하도록 제어하는 실행문
    - while 문, do / while 문, for 문, for / in 문, for / of 문

조건문

- if / else if / else

문법

```
if (조건1) {
   표현식1의 결과가 참일 때 실행;
}
else if (조건2) {
   조건2의 결과가 참일 때 실행;
}
 else {
   모든 조건이 거짓일 때 실행;
}
```



하나의 조건문 안에서 if 문과 else 문은 단 한 번만 사용될 수 있습니다.

하지만 else if 문은 여러 번 사용되어 다양한 조건을 설정할 수 있습니다.

조건문

- switch

문법

```
switch (조건) {
   case 값1:
      조건의 값이 값1일 때 실행;
      break;
   case 값2:
      조건의 값이 값2일 때 실행;
      break;
   ...
   default:
      조건 값이 어떠한 case 절에도 해당하지 않을 때 실행;
      break;
}
```

반복문(iteration statements)

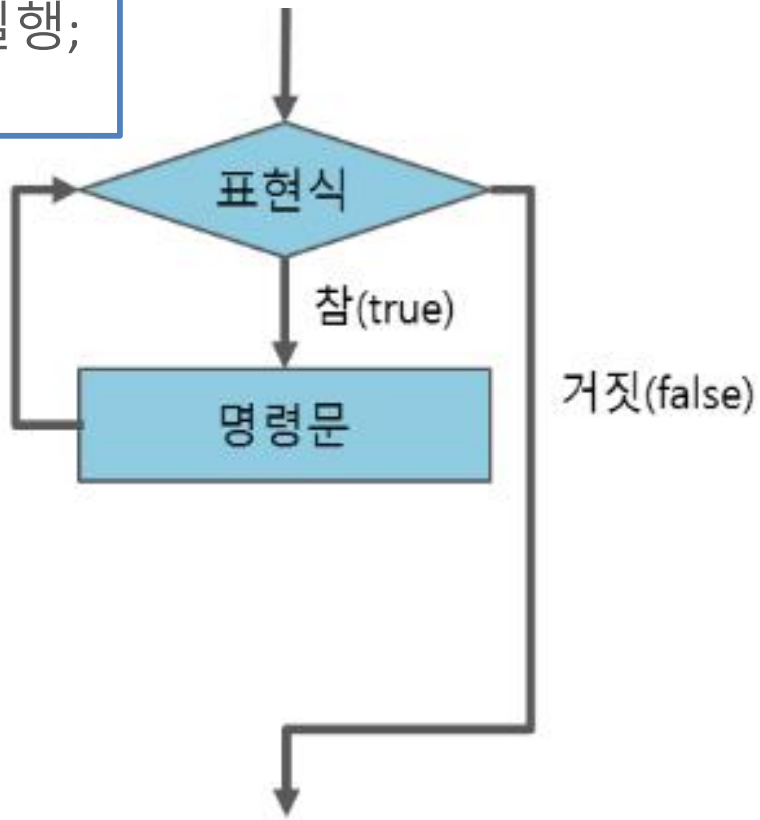
- while

문법

```
while (조건) {
   조건의 결과가 참인 동안 반복적으로 실행;
}
```

예제

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



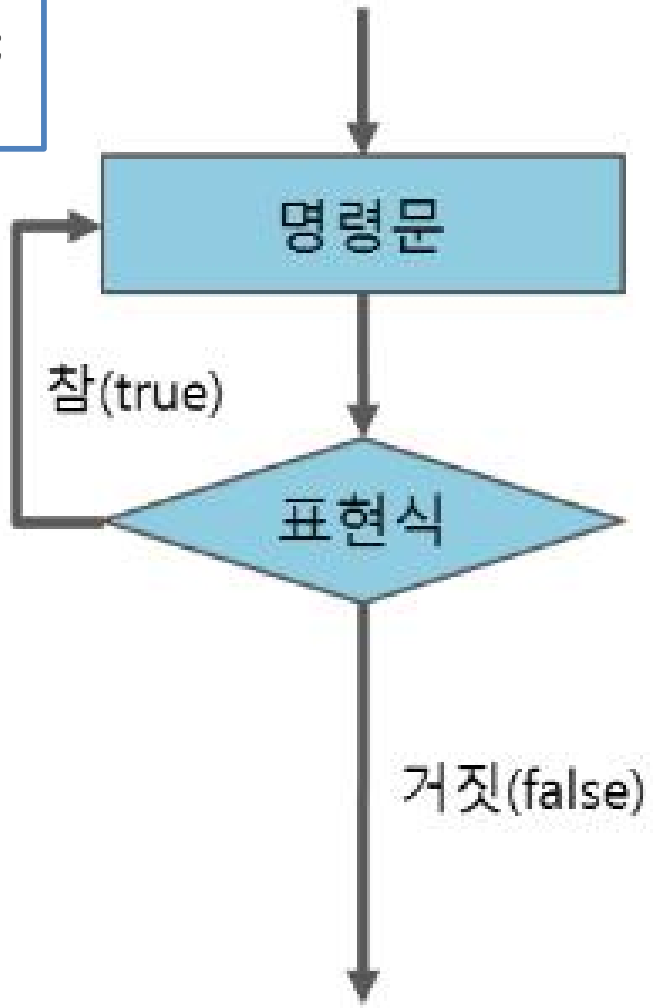
무한 루프(infinite loop) 주의

반복문(iteration statements)

- do / while

문법

```
do {
   조건의 결과가 참인 동안 반복적으로 실행;
} while (조건);
```



예제

```
var  j = 1;
do {
   document.write("j : " + (j++) + "<br>");
} while (j > 3);
```

반복문(iteration statements)

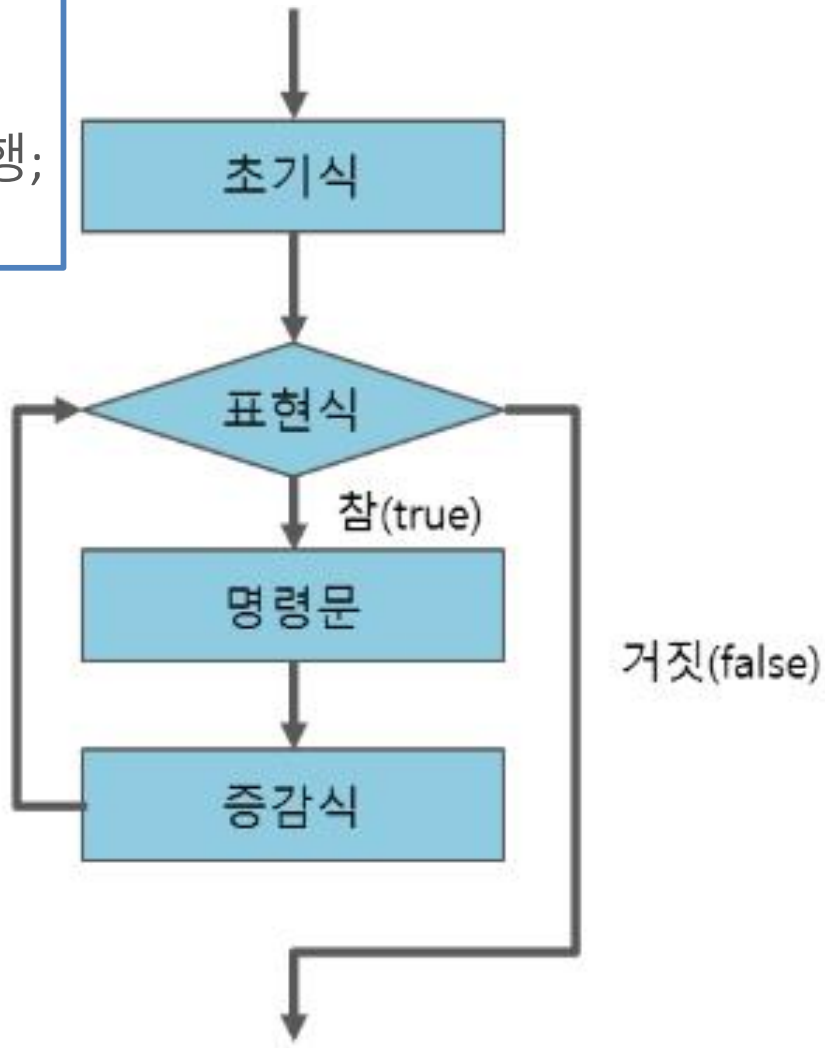
- for

문법

```
for (초기식; 표현식; 증감식) {
   표현식의 결과가 참인 동안 반복적으로 실행;
}
```

예제

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



break & continue

- break
  - 반복문(for, for...in, for...of, while, do...while) 또는 switch 문의 코드 블록을 탈출
- continue
  - 반복문(for, for...in, for...of, while, do...while)의 코드 블록 실행을 현 지점에서 중단하고 반복문의 증감식으로 이동
  - 반복문을 탈출하지 않음
- label
  - label 문은 프로그램 내의 특정 영역을 식별할 수 있도록 해주는 식별자

반복문(iteration statements)

- for / in
  - 해당 객체의 모든 열거할 수 있는 프로퍼티(enumerable properties)를 순회할 수 있도록 해주는 반복문

문법

```
for (변수 in 객체) {
   객체의 모든 열거할 수 있는 프로퍼티의 개수만큼 반복적으로 실행;
}
```

반복문(iteration statements)

- for / of
  - 반복할 수 있는 객체(iterable objects)를 순회할 수 있도록 해주는 반복문
    - Array, Map, Set, arguments 객체 등

문법

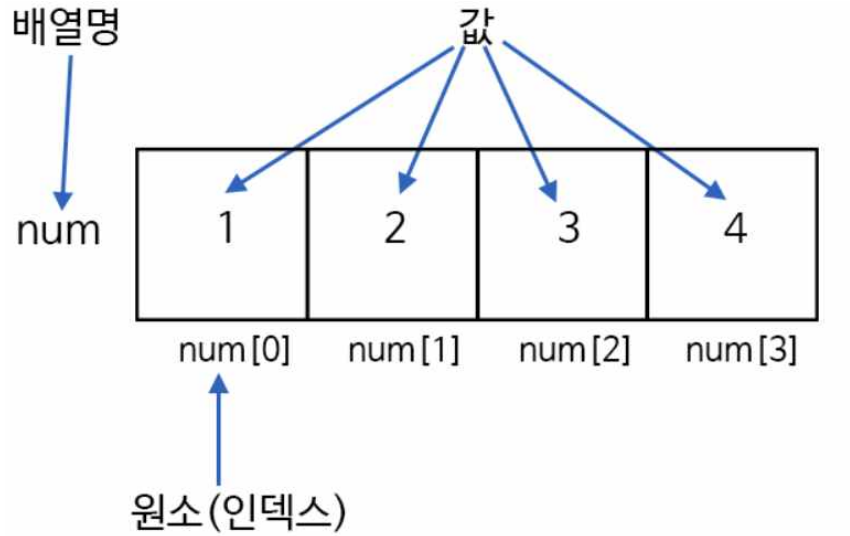
```
for (변수 of 객체) {
   객체의 모든 열거할 수 있는 값의 개수만큼 반복적으로 실행;
}
```

예제

```
var arr = [3, 4, 5];
for (var value of arr) {
   document.write(value + " ");
}
```

배열(array)

- 이름과 인덱스로 참조되는 정렬된 값의 집합
- 배열 요소(element)라고 하며, 배열에서의 위치를 가리키는 숫자를 인덱스(index)
  - 배열 요소의 타입이 고정되어 있지 않음
    - 같은 배열에 있는 배열 요소끼리의 타입이 다른수 있음
  - 배열 요소의 인덱스가 불연속 가능
    - 특정 배열 요소가 빈요소일 수 있음



배열 생성

문법

1. *var* arr = [배열요소1, 배열요소2,...];
2. *var* arr = Array(배열요소1, 배열요소2,...);
3. *var* arr = new Array(배열요소1, 배열요소2,...);

```
var arrLit = [1, true, "JavaScript"];
// 배열 리터럴을 이용하는 방법
var arrObj = Array(1, true, "JavaScript");
// Array 객체의 생성자를 이용하는 방법
var arrNewObj = new Array(1, true, "JavaScript");
// new 연산자를 이용한 Array 객체 생성 방법
```

```
var arr = new Array(2);
console.log(arr);
```

```
arr = new Array(1, 2, 3);
console.log(arr); // [1, 2, 3]
```

배열 생성

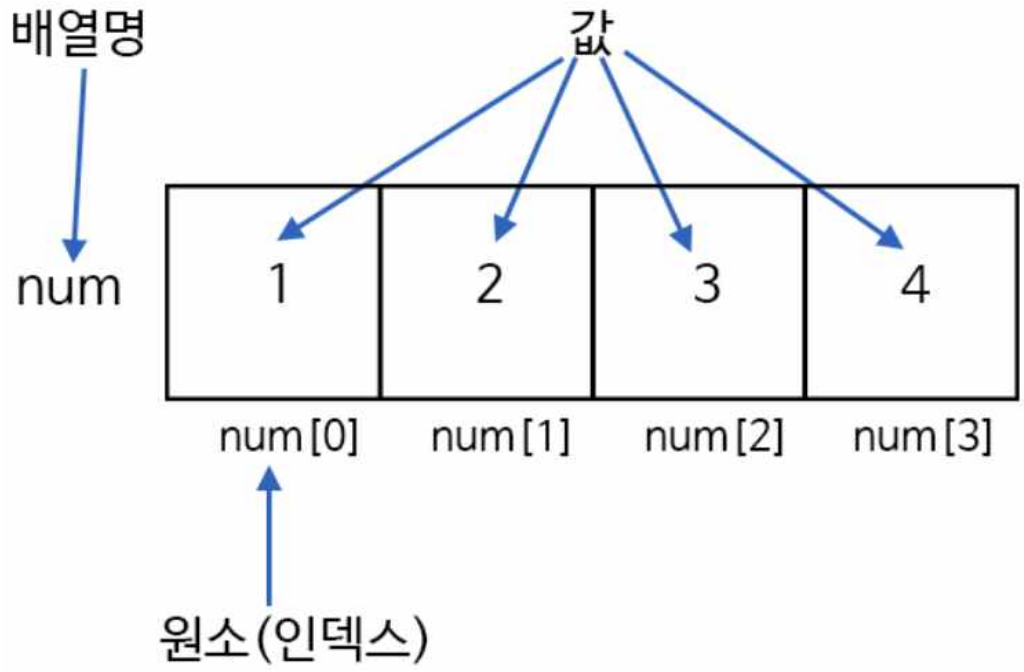
- Array객체의 static함수 (of, from)

```
// of
const arr = Array.of(1, 2);
console.log(arr); // [ 1, 2 ]

// from
const arr = [1, 2, 3, 4, 5, 6];
const arr2 = Array.from(arr);
console.log(arr2); // [ 1, 2, 3, 4, 5, 6 ]
```

값의 참조/ 배열의 길이

문법
배열이름[인덱스]



문법
배열이름.length

배열 요소의 추가

문법

1. arr.push(추가할 요소);
2. arr[arr.length] = 추가할 요소;
3. arr[특정인덱스] = 추가할 요소;

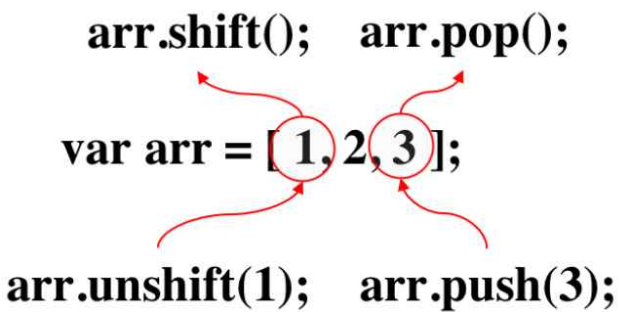
- 배열의 홀(hole)
  - ✓ 인덱스에 대응하는 배열 요소가 없는 부분
  - ✓ undefined 값을 가지는 요소
- 희소 배열(sparse array) <> 밀집 배열(dense array)
  - ✓ 자바스크립트의 배열은 지금까지 살펴본 일반적인 의미의 배열과 다르다.
  - ✓ 배열의 요소를 위한 각각의 메모리 공간은 동일한 크기를 갖지 않아도 되며 연속적으로 이어져 있지 않을 수도 있다.

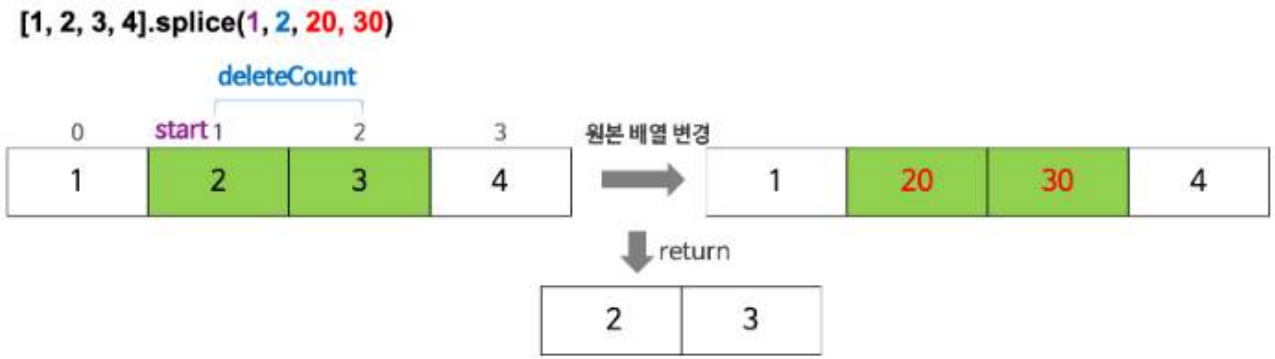
array function

- isArray
  - 특정한 오브젝트가 배열인지 체크
  - ES5 미지원시 instanceof
- indexOf
  - 특정한 아이템의 위치를 찾을 때
- includes
  - 배열안에 특정한 아이템이 있는지 체크

배열 함수

- push
  - 배열 맨 뒤에 요소를 추가 (배열 자체를 수정하고 길이를 반환)
- unshift
  - 배열 맨 앞에 요소를 추가 (배열 자체를 수정하고 길이를 반환)
- pop
  - 배열 맨 뒤에 요소를 제거 (배열 자체를 수정하고 제거된 요소를 반환)
- shift
  - 배열 맨 앞의 요소를 제거(배열 자체를 수정하고 제거된 요소를 반환)
- splice
  - 배열 특정 위치에 요소를 추가, 제거 (배열 자체를 수정하고 제거된 요소를 배열 형태로 반환)





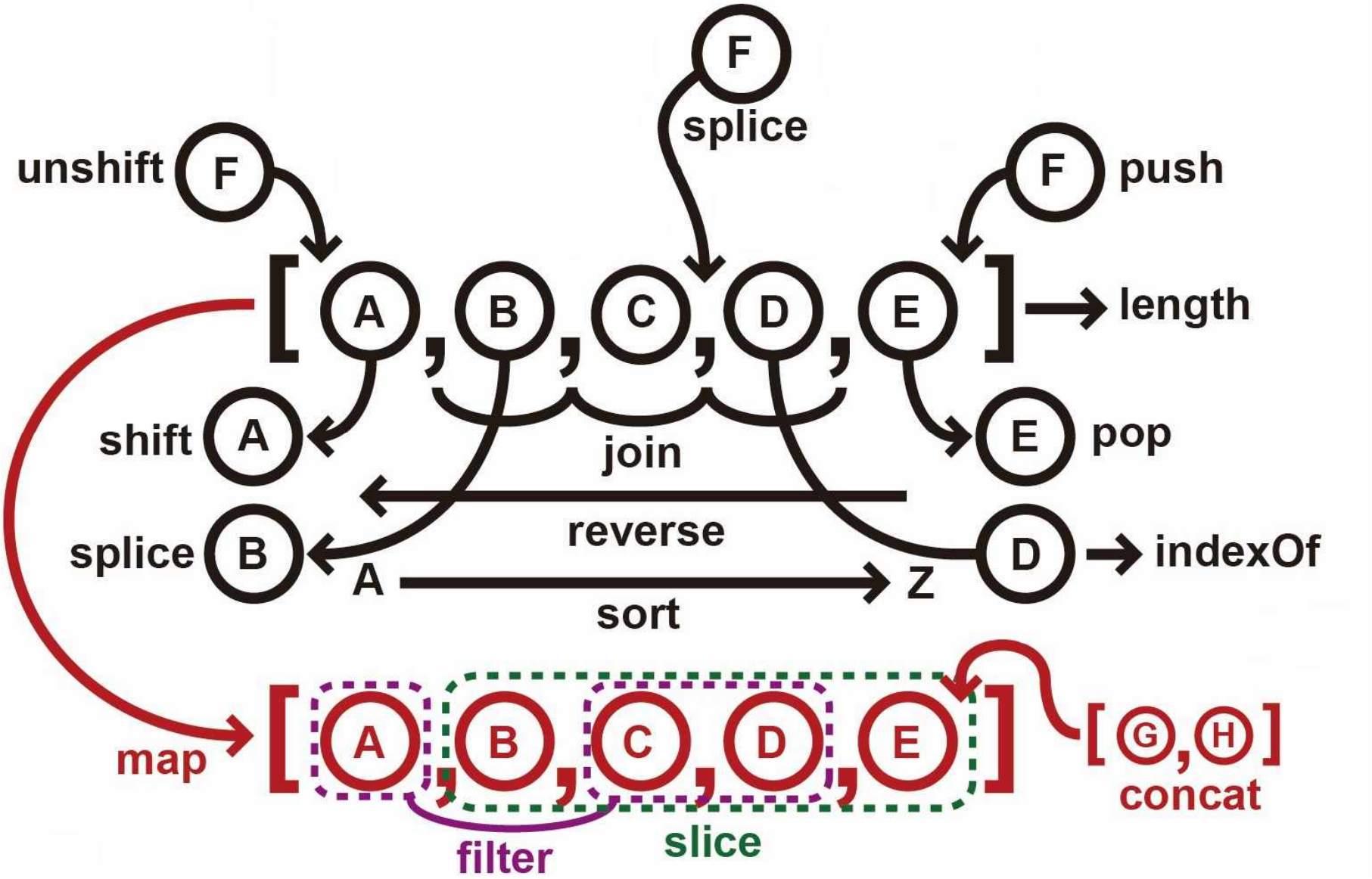
배열의 함수

- slice
  - 잘라진 새로운 배열을 만듬 (배열 자체는 유지하고 새로운 배열을 반환)
- concat
  - 여러개의 배열을 연결 (배열 자체는 유지하고 새로운 배열을 반환)
- sort
  - 배열의 요소를 적절하게 정렬(원본 배열을 직접 변경하며 정렬된 배열을 반환)
- reverse
  - 배열 요소의 순서를 거꾸로 배치 (배열 자체를 수정하고 변경된 요소를 배열 형태로 반환)
- flat
  - 중첩 배열을 하나의 배열로 평탄화 (배열 자체는 유지하고 새로운 배열을 반환)

배열의 함수

- fill
  - 특정한 값으로 배열을 채우기 (배열 자체를 수정하고 변경된 요소를 배열 형태로 반환)
- join
  - 배열을 문자열로 변환 (배열 자체는 유지하고 새로운 배열을 반환)

배열의 함수


F
splice
unshift
F
F
push
[A , B , C , D , E]
length
shift
A
join
E
pop
reverse
splice
B
A
Z
D
indexOf
sort
map
[A , B , C , D , E]
[G , H]
concat
filter
slice

배열의 반복/순회(iteration)

- 배열 내부의 값들은 사용할 때 반복문을 통해 값들을 가져다 사용
- for 문
- for..in 문
- for..of 문
- forEach 함수
  - break문 사용불가

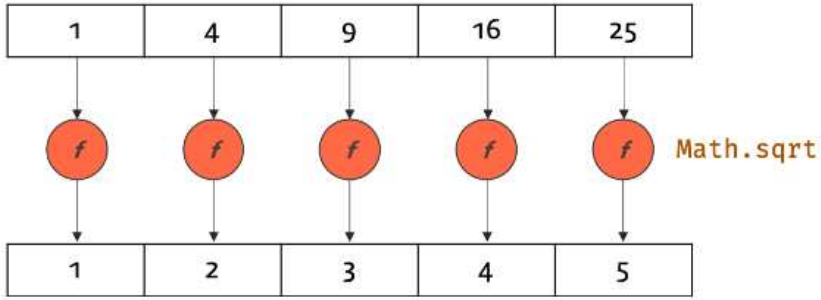
다차원 배열

- 배열 안에 또 다른 배열이 포함되게 하여 차원을 표현

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

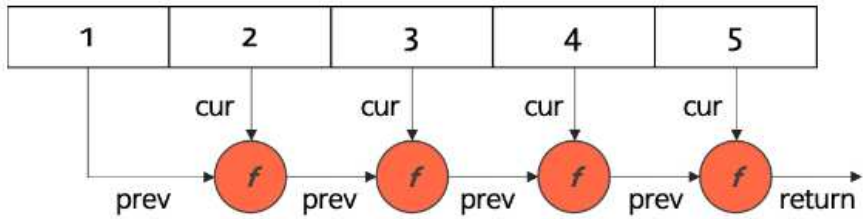
배열 고차 함수(Higher Order function)

- 고차 함수(Higher order function)
  - 함수를 인자로 전달받거나 함수를 결과로 반환하는 함수
- sort, forEach
- map
  - 주어진 콜백 함수의 반환값(결과값)으로 새로운 배열을 생성하여 반환
- filter
  - 배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수의 실행 결과가 true인 배열 요소의 값만을 추출한 새로운 배열을 반환



배열 고차 함수(Higher Order function)

- reduce
  - 배열을 순회하며 각 요소에 대하여 이전의 콜백함수 실행 반환값을 전달하여 콜백함수를 실행하고 그 결과를 반환



- some
  - 배열 내 일부 요소가 콜백 함수의 테스트를 통과하는지 확인하여 그 결과를 boolean으로 반환
- every
  - 배열 내 모든 요소가 콜백함수의 테스트를 통과하는지 확인하여 그 결과를 boolean으로 반환

배열 고차 함수(Higher Order function)

- find
  - 배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수를 실행하여 그 결과가 참인 첫 번째 요소를 반환
  - 참인 요소가 없으면 undefined
- findIndex
  - 배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수를 실행하여 그 결과가 참인 첫 번째 요소의 인덱스를 반환
  - 참인 요소가 존재하지 않는다면 -1을 반환

함수(function)

- 하나의 특별한 목적의 작업을 수행하도록 설계된 독립적인 블록

문법

```
function 함수이름(매개변수1, 매개변수2,...) {
   함수가 호출되었을 때 실행하는 실행문;
}
```

예제

```
function addNum(x, y) {
   return x + y;
}
document.write(addNum(2, 3));
```

- 자바스크립트 함수
  - 자바스크립트에서는 함수도 하나의 타입(datatype)
  - 함수를 변수에 대입하거나, 함수에 프로퍼티를 지정하는 것도 가능
  - 다른 함수 내에 중첩되어 정의 가능

함수

- 반환문(return)
  - 함수는 return 문을 포함할 수 있음
  - 함수내 실행된 결과를 전달
- 함수 호이스팅(hoisting)
  - 함수내 선언된 모든 변수는 함수 전체에 걸쳐 유효

매개변수 / 인수

- 매개변수(parameter)
  - 함수의 정의에서 전달받은 인수를 함수 내부로 전달하기 위해 사용하는 변수
- 인수(argument)
  - 함수가 호출될 때 함수로 값을 전달해주는 값
- ES6----------
- 디폴트 매개변수(default parameter)
  - 함수 호출시 명시된 인수를 미전달 시 사용할 기본 값
  - 자바스크립트에서 매개변수 기본 값은 undefined
- 나머지 매개 변수
  - 나머지 매개변수는 생략 접두사(...)를 사용하여 특정 위치의 인수부터 마지막 인수까지를 한 번에 지정 가능

자바스크립트에서는 함수를 정의할 때는 매개변수의 타입을 따로 명시하지 않습니다.
함수를 호출할 때에도 인수(argument)로 전달된 값에 대해 어떠한 타입 검사도 하지 않습니다.
함수를 호출할 때 함수의 정의보다 적은 수의 인수가 전달되더라도, 다른 언어와는 달리 오류를 발생시키지 않습니다.

arguments 객체

- 함수의 정의보다 더 많은 수의 인수가 전달되면, 매개변수에 대입되지 못한 인수들은 참조불가능
- arguments 객체를 이용하 함수로 전달된 인수의 총 개수를 확인하거나, 각각의 인수에도 접근가능
- arguments 객체는 함수가 호출될 때 전달된 인수를 배열의 형태로 저장

전역 함수

- 자바스크립트는 사용자의 편의를 위해 다양한 기능의 여러 전역 함수를 미리 정의하여 제공
- 전역 함수는 자바스크립트의 어떤 타입의 객체에서도 바로 사용 가능

eval()	문자열로 표현된 자바스크립트 코드를 실행하는 함수
isFinite()	전달된 값이 유한한 수인지를 검사하여 그 결과를 반환
isNaN()	전달된 값이 NaN인지를 검사하여 그 결과를 반환
parseFloat()	문자열을 파싱하여 부동 소수점 수(floating point number)로 반환
parseInt()	문자열을 파싱하여 정수로 반환

전역 함수

decodeURI()	URI에서 주소를 표시하는 특수문자를 제외하고, 모든 문자를 이스케이프 시퀀스(escape sequences) 처리하여 부호화
decodeURIComponent()	URI에서 encodeURI() 함수에서 부호화하지 않은 모든 문자까지 포함하여 이스케이프 시퀀스 처리
encodeURI()	encodeURI() 함수나 다른 방법으로 만들어진 URI(Uniform Resource Identifier)를 해독
encodeURIComponent()	encodeURIComponent() 함수나 다른 방법으로 만들어진 URI 컴포넌트를 해독
escape()	전달받은 문자열에서 특정 문자들을 16진법 이스케이프 시퀀스 문자로 변환
unescape()	escape() 함수나 다른 방법으로 만들어진 16진법 이스케이프 시퀀스 문자를 원래의 문자로 변환
Number()	전달받은 객체의 값을 숫자로 반환
String()	전달받은 객체의 값을 문자열로 반환

함수의 형태

- 즉시 실행 함수(IIFE, Immediately Invoke Function Expression)
  - 함수의 정의와 동시에 실행되는 함수
- 내부함수(Inner function)
  - 함수 내부에 정의된 함수
- 재귀 함수(Recusive function)
  - 자기 자신을 호출하는 함수
  - 재귀 함수는 반복 연산을 간단히 구현할 수 있다는 장점이 있지만 무한 반복에 빠질 수 있고, stackoverflow 에러를 발생시킬 수 있으므로 주의
- 콜백 함수(Callback function)
  - 함수를 명시적으로 호출하는 방식이 아니라 특정 이벤트가 발생했을 때 시스템에 의해 호출되는 함수

객체(object)

- 자바스크립트의 기본 타입(data type)은 객체(object)
  - 자바스크립트에서는 숫자, 문자열, 불리언, undefined 타입을 제외한 모든 것이 객체
    - 숫자, 문자열, 불리언과 같은 원시 타입은 값이 정해진 객체로 취급되어, 객체로서의 특징도 함께 가짐
- 이름(name)과 값(value)으로 구성된 프로퍼티(property)의 정렬되지 않은 집합
- 프로퍼티의 값으로 함수가 올 수도 있는데, 이러한 프로퍼티를 메소드(method)

예제

```
var cat = "나비";
var kitty = { name: "나비", family: "코리안 숏 헤어", age: 1, weight: 0.1 };
cat;
kitty.name;    //참조
kitty["name"];  //참조
```

값 참조

- 객체의 프로퍼티 참조

문법
객체이름.프로퍼티이름
또는
객체이름["프로퍼티이름"]

- 객체의 메소드 참조

문법
객체이름.메소드이름()

객체의 생성

- 자바와 같은 클래스 기반 객체 지향 언어는 클래스를 사전에 정의하고 필요한 시점에 new 연산자를 사용하여 인스턴스를 생성하는 방식으로 객체를 생성
- 자바스크립트는 프로토타입 기반 객체 지향 언어로서 클래스라는 개념이 없고 별도의 객체 생성 방법이 존재

객체의 생성

- 객체 리터럴
  - 중괄호({})를 사용하여 객체를 생성
  - {} 내에 1개 이상의 프로퍼티를 기술하면 해당 프로퍼티가 추가된 객체를 생성
  - {} 내에 아무것도 기술하지 않으면 빈 객체가 생성

문법

```
var 객체이름 = {
   프로퍼티1이름 : 프로퍼티1의값,
   프로퍼티2이름 : 프로퍼티2의값,
   …
};
```

```
var person1 = {
  name: 'Lee',
  gender: 'male',
  sayHello: function () {
    console.log(this.name);
  }
};
```

객체의 생성

- Object 생성자 함수(Constructor function)
  - new 연산자와 Object 생성자 함수를 호출하여 빈 객체를 생성할 수 있다. 빈 객체 생성 이후 프로퍼티 또는 메소드를 추가하여 객체를 완성하는 방법
- 생성자 함수
  - new 키워드와 함께 객체를 생성하고 초기화하는 함수
    - 생성자 함수 이름은 일반적으로 대문자로 시작
  - 프로퍼티 또는 메소드명 앞에 기술한 this는 생성자 함수가 생성할 인스턴스(instance)
    - 인스턴스(instance) : 생성자 함수를 통해 생성된 객체
  - this에 연결(바인딩)되어 있는 프로퍼티와 메소드는 외부에서 참조 가능
  - 생성자 함수 내에서 선언된 일반 변수는 외부에서 참조 불가능

```
var person = new Object();
person.name = 'Lee';
person.gender = 'male';
person.sayHello = function () {
  console.log(this.name);
};
```

```
function Person(name, gender) {
    this.name = name;
    this.gender = gender;
    this.sayHello = function(){
        console.log(this.name);
    };
}
```

참고

- 자바스크립트는 Object 생성자 함수 이외에도 String, Number, Boolean, Array, Date, RegExp 등의 빌트인 생성자 함수를 제공
  - 일반 함수와 생성자 함수를 구분하기 위해 생성자 함수의 이름은 파스칼 케이스(PascalCase)/ 카멜 표기법(Camel case)을 사용하는 것이 일반적

예약어

- abstract arguments boolean break byte
- case catch char class* const
- continue debugger default delete do
- double else enum* eval export*
- extends* false final finally float
- for function goto if implements
- import* in instanceof int interface
- let long native new null
- package private protected public return
- short static super* switch synchronized
- this throw throws transient true
- try typeof var void volatile
- while with yield
- // *는 ES6에서 추가된 예약어

프로토타입(prototype)

- 자바스크립트의 모든 객체는 최소한 하나 이상의 다른 객체로부터 상속을 받으며, 이때 상속되는 정보를 제공하는 객체
- 상속(inheritance)
  - 새로운 클래스에서 기존 클래스의 모든 프로퍼티와 메소드를 사용할 수 있는 것
  - 상속을 통해 새로운 프로그램의 요구에 맞게 기존 클래스를 수정하여 재사용
  - 클래스 간의 종속 관계를 형성함으로써 객체의 관계를 조직화
  - 추상화, 캡슐화와 더불어 객체 지향 프로그래밍을 구성하는 중요한 특징 중 하나
- 자바스크립트는 프로토타입 기반(prototype-based)의 객체 지향 언어
  - C#이나 C++과 같은 클래스 기반(class-based)의 객체 지향 언어
  - 자바스크립트에서는 현재 존재하고 있는 객체를 프로토타입으로 사용하여, 해당 객체를 복제하여 재사용하는 것을 상속이라고 정의
- 프로토타입 체인(prototype chain)

객체 프로퍼티의 참조 및 삭제

- 참조

문법
객체이름.프로퍼티이름
또는
객체이름["프로퍼티이름"]

- 삭제

문법
delete 객체이름.프로퍼티이름;

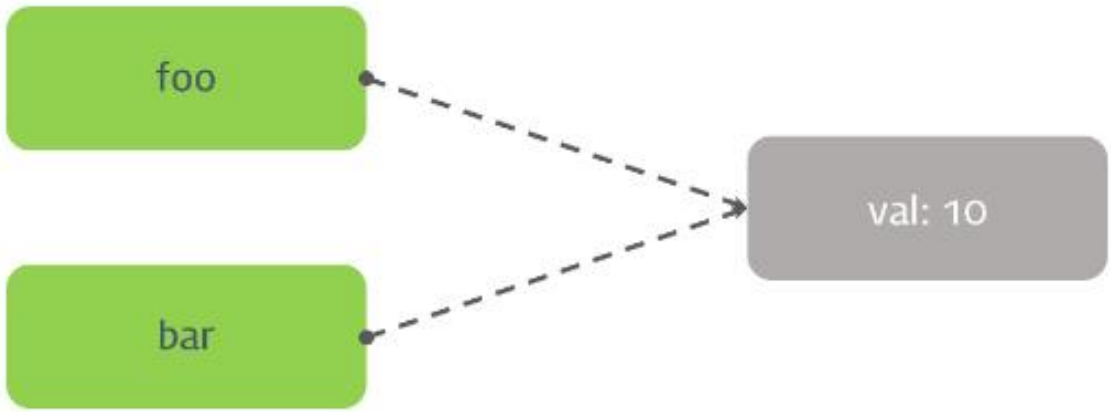
객체 프로퍼티의 순회

- for-in 문
  - 객체의 문자열 키(key)를 순회하기 위한 문법
- for-of 문
  - 배열의 값을 순회하기 위한 문법

객체 레퍼런스(object reference)

- pass by reference

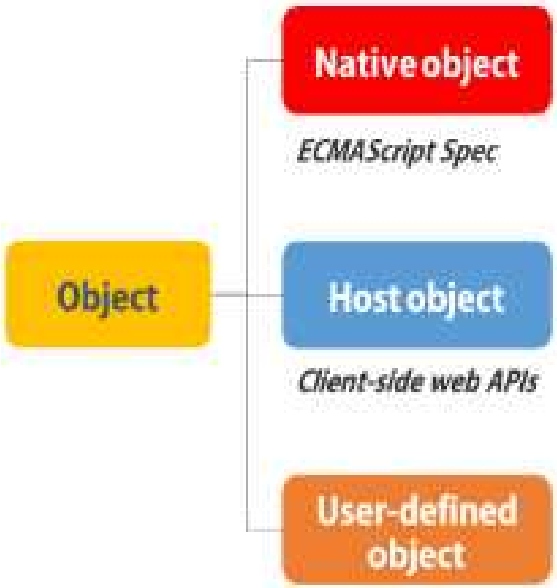
```
// Pass-by-reference
var foo = { val: 10 }
var bar = foo;
console.log(foo.val, bar.val); // 10 10
console.log(foo === bar); // true
bar.val = 20;
console.log(foo.val, bar.val); // 20 20
console.log(foo === bar); // true
```



객체 메소드

- 모든 자바스크립트 객체는 Object 객체와 Object.prototype 객체의 모든 프로퍼티와 메소드를 상속
- hasOwnProperty()
  - 특정 프로퍼티가 해당 객체에 존재하는지를 검사
- propertyIsEnumerable()
  - 특정 프로퍼티가 해당 객체에 존재하고, 열거할 수 있는 프로퍼티인지를 검사
  - Object.defineProperty() 메소드는 객체에 프로퍼티와 추가하고 그 속성까지 설정하는 메소드 (ECMAScript 5)
- isPrototypeOf()
  - 특정 객체의 프로토타입 체인에 현재 객체가 존재하는지를 검사
- isExtensible()
  - 객체에 새로운 프로퍼티를 추가할 수 있는지 여부를 반환
  - preventExtensions() 메소드를 사용하여 해당 객체에 새로운 프로퍼티를 추가할 수 없도록 설정
- toString()
  - 메소드를 호출한 객체의 값을 문자열로 반환
- valueOf()
  - 특정 객체의 원시 타입(primitive type)의 값을 반환

객체의 분류



- Object, String, Number, Boolean, Symbol, Function, Array, RegExp, Date, Math, Promise…

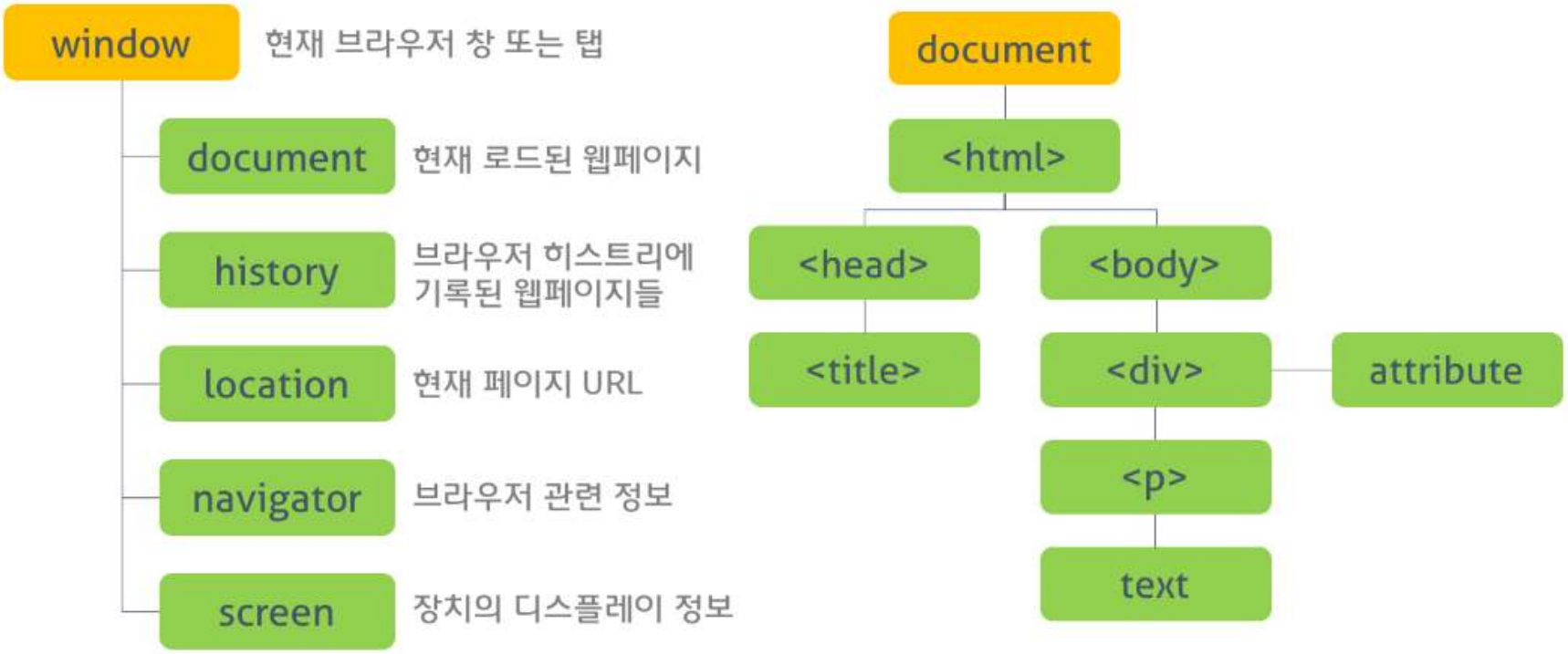
- Global object: window, global(Node.js)
- DOM: Document, Event, HTMLElement, HTMLCollection, NodeList…
- BOM: Location, History, Navigator, Screen
- Ajax: XMLHttpRequest, fetch
- HTML5 APIs: Blob, File, Canvas, Geolocation, Drag&Drop, Web storage, Web worker…

네이티브 객체

- Native objects or Built-in objects
- ECMAScript 명세에 정의된 객체를 말하며 애플리케이션 전역의 공통 기능을 제공
- 네이티브 객체는 애플리케이션의 환경과 관계없이 언제나 사용 가능
- Object, String, Number, Function, Array, RegExp, Date, Math와 같은 객체 생성에 관계가 있는 함수 객체와 메소드로 구성

호스트 객체(Host Object)

- 브라우저 환경에서 제공하는 window, XmlHttpRequest, HTMLElement 등의 DOM 노드 객체와 같이 호스트 환경에 정의된 객체
- BOM(Browser Object Model)
- DOM(Document Object Model)



전역 객체/래퍼 객체

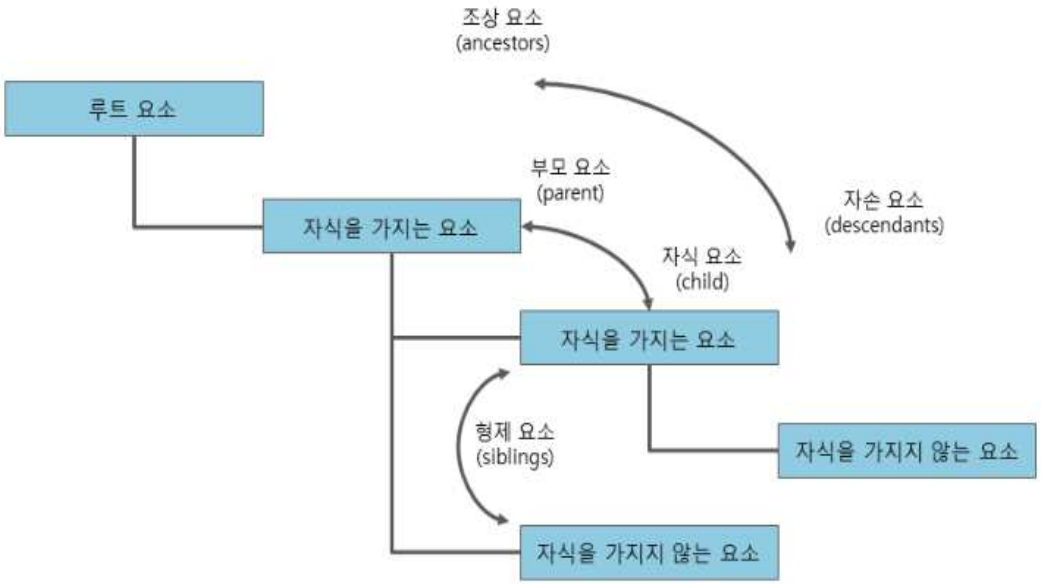
- 전역 객체(global object)
  - 자바스크립트에 미리 정의된 객체로 전역 프로퍼티나 전역 함수를 담는 공간의 역할
  - 전역 객체 그 자체는 전역 범위(global scope)에서 this 연산자를 통해 접근가능
  - 모든 객체는 전역 객체의 프로퍼티
- 래퍼 객체(wrapper object)
  - 원시 타입의 프로퍼티에 접근하려고 할 때 생성되는 임시 객체

DOM

- 문서 객체 모델(DOM, Document Object Model)
- XML이나 HTML 문서에 접근하기 위한 일종의 인터페이스
- 이 객체 모델은 문서 내의 모든 요소를 정의하고, 각각의 요소에 접근하는 방법을 제공
- W3C DOM 표준은 세 가지 모델로 구분됩니다.
  - Core DOM : 모든 문서 타입을 위한 DOM 모델
  - HTML DOM : HTML 문서를 위한 DOM 모델
  - XML DOM : XML 문서를 위한 DOM 모델

XML

- 데이터를 저장하고 전달할 목적으로 만들어졌으며, 저장되는 데이터의 구조를 기술하기 위한 언어
- EXtensible Markup Language




```
<?xml version="1.0" encoding="UTF-8"?>
<shop city="서울" type="마트">
  <food>
    <name>귤</name>
    <sort>과일</sort>
    <cost>3000</cost>
  </food>
  <food>
    <name>상추</name>
    <sort>야채</sort>
    <cost>2000</cost>
  </food>
</shop>
```

Number 객체

- 자바스크립트에서는 정수와 실수를 따로 구분하지 않고, 모든 수를 실수 하나로만 표현
  - IEEE 754 국제 표준에서 정의한 64비트 부동 소수점 수로 저장
  - 64비트 부동 소수점 수의 정밀도는 정수부는 15자리까지, 소수부는 17자리까지만 유효
- 산술 연산 시 모든 수가 10진수로 자동 변환되어 계산
- 숫자에 toString() 메소드를 사용하여 해당 숫자를 여러 진법의 형태로 변환

Number 객체

- Infinity
  - 양의 무한대를 의미하는 Infinity 값
  - 음의 무한대를 의미하는 -Infinity 값
  - Infinity 값은 사용자가 임의로 수정할 수 없는 읽기 전용 값
- NaN(isNaN())
  - NaN(Not A Number)는 숫자가 아니라는 의미
  - 정의되지 않은 값이나 표현할 수 없는 값
  - 0을 0으로 나누거나, 숫자로 변환할 수 없는 피연산자로 산술 연산을 시도하는 경우에 반환되는 읽기 전용 값
- Number 객체
  - 숫자 값을 감싸고 있는 래퍼(wrapper) 객체
  - 숫자는 보통 숫자 리터럴을 사용하여 표현
  - 수를 나타낼 때 new 연산자를 사용하여 명시적으로 Number 객체를 생성 가능

Number 메소드

메소드	설명
Number.parseFloat()	문자열을 파싱하여, 문자열에 포함된 숫자 부분을 실수 형태로 반환함.
Number.parseInt()	문자열을 파싱하여, 문자열에 포함된 숫자 부분을 정수 형태로 반환함.
Number.isNaN()	전달된 값이 NaN인지 아닌지를 검사함.
Number.isFinite()	전달된 값이 유한한 수인지 아닌지를 검사함.
Number.isInteger()	전달된 값이 정수인지 아닌지를 검사함.
Number.isSafeInteger()	전달된 값이 안전한 정수(safe integer)인지 아닌지를 검사함.

- 동명의 전역 함수와 동일 동작
- 숫자로 강제변환 하지 않음
- 사파리와 익스플로러에서 미지원

Number prototype 메소드

메소드	설명
toExponential()	Number 인스턴스를 지수 표기법으로 변환한 후, 그 값을 문자열로 반환함.
toFixed()	Number 인스턴스의 소수 부분 자릿수를 전달받은 값으로 고정한 후, 그 값을 문자열로 반환함.
toPrecision()	Number 인스턴스의 가수와 소수 부분의 합친 자릿수를 전달받은 값으로 고정한 후, 그 값을 문자열로 반환함.
toString()	Number 인스턴스의 값을 문자열로 반환함.
valueOf()	Number 인스턴스가 가지고 있는 값을 반환함.

- Number.isFinite() 메소드
- 전달된 값이 유한한 수인지 아닌지를 검사
- 전역 함수인 isFinite() 함수처럼 전달된 값을 숫자로 강제 변환하지 않음
- Number.isInteger() 메소드
- 전달된 값이 정수인지 아닌지를 검사

Math 객체

- 수학에서 자주 사용하는 상수와 함수들을 미리 구현해 놓은 자바스크립트 표준 내장 객체

메소드	설명
Math.min(x, y, ...)	인수로 전달받은 값 중에서 가장 작은 수를 반환함.
Math.max(x, y, ...)	인수로 전달받은 값 중에서 가장 큰 수를 반환함.
Math.random()	0보다 크거나 같고 1보다 작은 랜덤 숫자(random number)를 반환함.
Math.round(x)	x를 소수점 첫 번째 자리에서 반올림하여 그 결과를 반환함.
Math.floor(x)	x와 같거나 작은 수 중에서 가장 큰 정수를 반환함.
Math.ceil(x)	x와 같거나 큰 수 중에서 가장 작은 정수를 반환함.
Math.abs(x)	x의 절댓값을 반환함.
Math.cbrt(x)	x의 세제곱근을 반환함.
Math.sqrt(x)	x의 제곱근을 반환함.

Math.PI : 원의 원주를 지름으로 나눈 비율(원주율) 값 (3.14159)

Date 객체

- 연월일, 시분초의 정보, 밀리초(millisecond)의 정보 제공
  - 자바스크립트에서 월(month)을 나타낼 때는 1월이 0으로 표현되고, 12월이 11로 표현된다는 사실에 유의
- 객체 초기화
  - new Date()
  - new Date("날짜를 나타내는 문자열")
  - new Date(밀리초)
  - new Date(년, 월, 일, 시, 분, 초, 밀리초)

- new Date(year, month[, day, hour, minute, second, millisecond])

인수	내용
year	1900년 이후의 년
month	월을 나타내는 0 ~ 11까지의 정수 (주의: 0부터 시작, 0 = 1월)
day	일을 나타내는 1 ~ 31까지의 정수
hour	시를 나타내는 0 ~ 23까지의 정수
minute	분을 나타내는 0 ~ 59까지의 정수
second	초를 나타내는 0 ~ 59까지의 정수
millisecond	밀리초를 나타내는 0 ~ 999까지의 정수

자바스크립트 날짜 양식(date format)

- ISO 날짜 양식
  - ISO 8601은 날짜와 시간을 나타내는 국제 표준 양식
  - YYYY-MM-DDTHH:MM:SS
    - T는 UTC(협정세계시)를 나타내는 문자로 시간까지 표현할 때에는 반드시 사용해야 함.
  - YYYY-MM-DD | YYYY-MM | YYYY
- Long 날짜 양식
  - MMM DD YYYY | DD MMM YYYY
  - `new Date("19 Feb 1982");  // DD MMM YYYY`
  - `new Date("February 19 1982");`
- Short 날짜 양식
  - MM/DD/YYYY | YYYY/MM/DD
  - `new Date("02/19/1982"); // MM/DD/YYYY`
  - `new Date("1982/02/19"); // YYYY/MM/DD`
- Full 날짜 양식
  - 자바스크립트에서 사용하는 날짜 양식으로 표현된 문자열도 날짜로 인식
  - Wed May 25 2016 17:00:31 GMT+0900
  - `new Date("Wed May 25 2016 17:00:00 GMT-0500 (New York Time)");`

getter /setter 메소드

- 획득자(getter)
- 설정자(setter)
- 객체의 무결성

Date 메소드

메소드	설명	값의 범위
getDate()	현지 시각으로 현재 일자에 해당하는 숫자를 반환함.	1 ~ 31
getDay()	현지 시각으로 현재 요일에 해당하는 숫자를 반환함.	0 ~ 6
getMonth()	현지 시각으로 현재 월에 해당하는 숫자를 반환함.	0 ~ 11
getFullYear()	현지 시각으로 현재 연도를 4비트의 숫자(YYYY)로 반환함.	YYYY
getHours()	현지 시각으로 현재 시각에 해당하는 숫자를 반환함.	0 ~ 23
getMilliseconds()	현지 시각으로 현재 시각의 밀리초에 해당하는 숫자를 반환함.	0 ~ 999
getMinutes()	현지 시각으로 현재 시각의 분에 해당하는 숫자를 반환함.	0 ~ 59
getSeconds()	현지 시각으로 현재 시각의 초에 해당하는 숫자를 반환함.	0 ~ 59
getTime()	1970년 1월 1일 0시 0분 0초부터 현재까지의 시간을 밀리초 단위로 환산한 값을 숫자로 반환함.	-
getTimezoneOffset()	UTC로부터 현재 시각까지의 시간차를 분 단위로 환산한 값을 숫자로 반환함.	-

Date method

메소드	설명	값의 범위
setDate()	현지 시각으로 특정 일자를 설정함.	1 ~ 31
setMonth()	현지 시각으로 특정 월을 설정함.	0 ~ 11
setFullYear()	현지 시각으로 특정 연도를 설정함. (연도뿐만 아니라 월과 일자도 설정할 수 있음)	YYYY, MM, DD
setHours()	현지 시각으로 특정 시간을 설정함.	0 ~ 23
setMilliseconds()	현지 시각으로 특정 밀리초를 설정함.	0 ~ 999
setMinutes()	현지 시각으로 특정 분을 설정함.	0 ~ 59
setSeconds()	현지 시각으로 특정 초를 설정함.	0 ~ 59
setTime()	1970년 1월 1일 0시 0분 0초부터 밀리초 단위로 표현되는 특정 시간을 설정함.	-

String 객체

- 문자열 표현
- 문자열 리터럴
  - 큰따옴표("")나 작은따옴표('')를 사용

```
let str = "JavaScript";
let strObj = new String("JavaScript");
str;              // "JavaScript"
strObj;           // "JavaScript"
typeof str;       // string
typeof strObj;    // object
(str == strObj); // 문자열 값이 같으므로, true를 반환함.
(str === strObj); // 문자열 값은 같지만 타입이 다르므로, false를 반환함.
```

- 문자열의 길이

```
let str1 = "만우절";
let str2 = "apple";
str1.length; // 3
str2.length; // 5
```

String 객체

- 원시 타입인 문자열을 다룰 때 유용한 프로퍼티와 메소드를 제공하는 레퍼(wrapper) 객체
- String 객체의 별도 생성없이 String 객체의 프로퍼티와 메소드를 사용가능

```
const str = 'Hello world!';
console.log(str.toUpperCase());
// 'HELLO WORLD!'
```

String 메소드

메소드	설명
indexOf()	String 인스턴스에서 특정 문자나 문자열이 처음으로 등장하는 위치의 인덱스를 반환함.
lastIndexOf()	String 인스턴스에서 특정 문자나 문자열이 마지막으로 등장하는 위치의 인덱스를 반환함.
charAt()	String 인스턴스에서 전달받은 인덱스에 위치한 문자를 반환함.
charCodeAt()	String 인스턴스에서 전달받은 인덱스에 위치한 문자의 UTF-16 코드를 반환함. (0 ~ 65535)
charPointAt()	String 인스턴스에서 전달받은 인덱스에 위치한 문자의 유니코드 코드 포인트(unicode code point)를 반환함.
slice()	String 인스턴스에서 전달받은 시작 인덱스부터 종료 인덱스 바로 앞까지의 문자열을 추출한 새 문자열을 반환함.
substring()	String 인스턴스에서 전달받은 시작 인덱스부터 종료 인덱스 바로 앞까지의 문자열을 추출한 새 문자열을 반환함.
substr()	String 인스턴스에서 전달받은 시작 인덱스부터 길이만큼의 문자열을 추출한 새로운 문자열을 반환함.
split()	String 인스턴스에서 구분자(separator)를 기준으로 나눈 후, 나뉜 문자열을 하나의 배열로 반환함.
concat()	String 인스턴스에 전달받은 문자열을 결합한 새로운 문자열을 반환함.
toUpperCase()	String 인스턴스의 모든 문자를 대문자로 변환한 새로운 문자열을 반환함.
toLowerCase()	String 인스턴스의 모든 문자를 소문자로 변환한 새로운 문자열을 반환함.

메소드	설명
trim()	String 인스턴스의 양 끝에 존재하는 공백과 모든 줄 바꿈 문자(LF, CR 등)를 제거한 새로운 문자열을 반환함.
search()	인수로 전달받은 정규 표현식에 맞는 문자나 문자열이 처음으로 등장하는 위치의 인덱스를 반환함.
replace()	인수로 전달받은 패턴에 맞는 문자열을 대체 문자열로 변환한 새 문자열을 반환함.
match()	인수로 전달받은 정규 표현식에 맞는 문자열을 찾아서 하나의 배열로 반환함.
includes()	인수로 전달받은 문자나 문자열이 포함되어 있는지를 검사한 후 그 결과를 불리언 값으로 반환함.
startsWith()	인수로 전달받은 문자나 문자열로 시작되는지를 검사한 후 그 결과를 불리언 값으로 반환함.
endsWith()	인수로 전달받은 문자나 문자열로 끝나는지를 검사한 후 그 결과를 불리언 값으로 반환함.
toLocaleUpperCase()	영문자뿐만 아니라 모든 언어의 문자를 대문자로 변환한 새로운 문자열을 반환함.
toLocaleLowerCase()	영문자뿐만 아니라 모든 언어의 문자를 소문자로 변환한 새로운 문자열을 반환함.
localeCompare()	인수로 전달받은 문자열과 정렬 순서로 비교하여 그 결과를 정수 값으로 반환함.
normalize()	해당 문자열의 유니코드 표준화 양식(Unicode Normalization Form)을 반환함.
repeat()	해당 문자열을 인수로 전달받은 횟수만큼 반복하여 결합한 새로운 문자열을 반환함.
toString()	String 인스턴스의 값을 문자열로 반환함.
valueOf()	String 인스턴스의 값을 문자열로 반환함.

Array 객체

- 배열(array)은 정렬된 값들의 집합으로 정의
- Array.isArray() 메소드
  - 전달받은 값이 Array 객체인지 아닌지를 검사
- ES6 추가 메소드
- Array.from() 메소드
  - 객체들을 배열처럼 변환
    - 배열과 비슷한 객체(array-like objects) : length 프로퍼티와 인덱스 된 요소를 가지고 있는 객체
    - 반복할 수 있는 객체(iterable objects) : Map과 Set 객체 및 문자열과 같이 해당 요소를 개별적으로 선택할 수 있는 객체
- Array.of() 메소드
  - 인수의 수나 타입에 상관없이 인수로 전달받은 값을 가지고 새로운 Array 인스턴스를 생성

array 메소드

메소드	설명
forEach()	해당 배열의 모든 요소에 대하여 반복적으로 명시된 콜백 함수를 실행함.
map()	해당 배열의 모든 요소에 대하여 반복적으로 명시된 콜백 함수를 실행한 후, 그 실행 결과를 새로운 배열로 반환함.
filter()	해당 배열의 모든 요소에 대하여 반복적으로 명시된 콜백 함수를 실행한 후, 그 결괏값이 true인 요소들만을 새로운 배열에 담아 반환함.
every()	해당 배열의 모든 요소에 대하여 반복적으로 명시된 콜백 함수를 실행한 후, 그 결괏값이 모두 true일 때에만 true를 반환함.
some()	해당 배열의 모든 요소에 대하여 반복적으로 명시된 콜백 함수를 실행한 후, 그 결괏값이 하나라도 true이면 true를 반환함.
reduce()	해당 배열의 모든 요소를 하나의 값으로 줄이기 위해, 두 개의 인수를 전달받는 콜백 함수를 실행함. (배열의 첫 번째 요소부터 시작함.)
reduceRight()	해당 배열의 모든 요소를 하나의 값으로 줄이기 위해, 두 개의 인수를 전달받는 콜백 함수를 실행함. (배열의 마지막 요소부터 시작함.)
entries()	배열 요소별로 키와 값의 한 쌍으로 이루어진 새로운 배열 반복자 객체(Array Iterator Object)를 배열 형태로 반환함.
keys()	배열 요소별로 키(key)만 포함하는 새로운 배열 반복자 객체를 배열 형태로 반환함.
values()	배열 요소별로 값(value)만 포함하는 새로운 배열 반복자 객체를 배열 형태로 반환함.
find()	검사를 위해 전달받은 함수를 만족하는 배열 요소의 값을 반환함. 만족하는 값이 없으면 undefined를 반환함.
findIndex()	검사를 위해 전달받은 함수를 만족하는 배열 요소의 인덱스를 반환함. 만족하는 값이 없으면 -1을 반환함.

Document 객체

- 웹 페이지 그 자체를 의미
- 웹 페이지에 존재하는 HTML 요소에 접근하고자 할 때는 반드시 Document 객체부터 시작

Document 객체

메소드	설명
document.getElementsByTagName(태그이름)	해당 태그 이름의 요소를 모두 선택함.
document.getElementById(아이디)	해당 아이디의 요소를 선택함.
document.getElementsByClassName(클래스이름)	해당 클래스에 속한 요소를 모두 선택함.
document.getElementsByName(name속성값)	해당 name 속성값을 가지는 요소를 모두 선택함.
document.querySelector(선택자)	해당 선택자로 선택되는 요소를 선택함.
document.querySelectorAll(선택자)	해당 선택자로 선택되는 요소를 모두 선택함.
document.createElement(HTML요소)	지정된 HTML 요소를 생성함.
document.write(텍스트)	HTML 출력 스트림을 통해 텍스트를 출력함.

Document 객체

객체 집합	설명
document.anchors	name 속성을 가지는 <a>요소를 모두 반환함.
document.applets	applet 요소를 모두 반환함. (HTML5에서 제외됨)
document.body	<body>요소를 반환함.
document.cookie	HTML 문서의 쿠키(cookie)를 반환함.
document.domain	HTML 문서가 위치한 서버의 도메인 네임(domain name)을 반환함.
document.forms	<form>요소를 모두 반환함.
document.images	<img>요소를 모두 반환함.
document.links	href 속성을 가지는 <area>요소와 <a>요소를 모두 반환함.
document.referrer	링크(linking)되어 있는 문서의 URI를 반환함.
document.title	<title>요소를 반환함.
document.URL	HTML 문서의 완전한 URL 주소를 반환함.
document.baseURI	HTML 문서의 절대 URI(absolute base URI)를 반환함.
document.doctype	HTML 문서의 문서 타입(doctype)을 반환함.

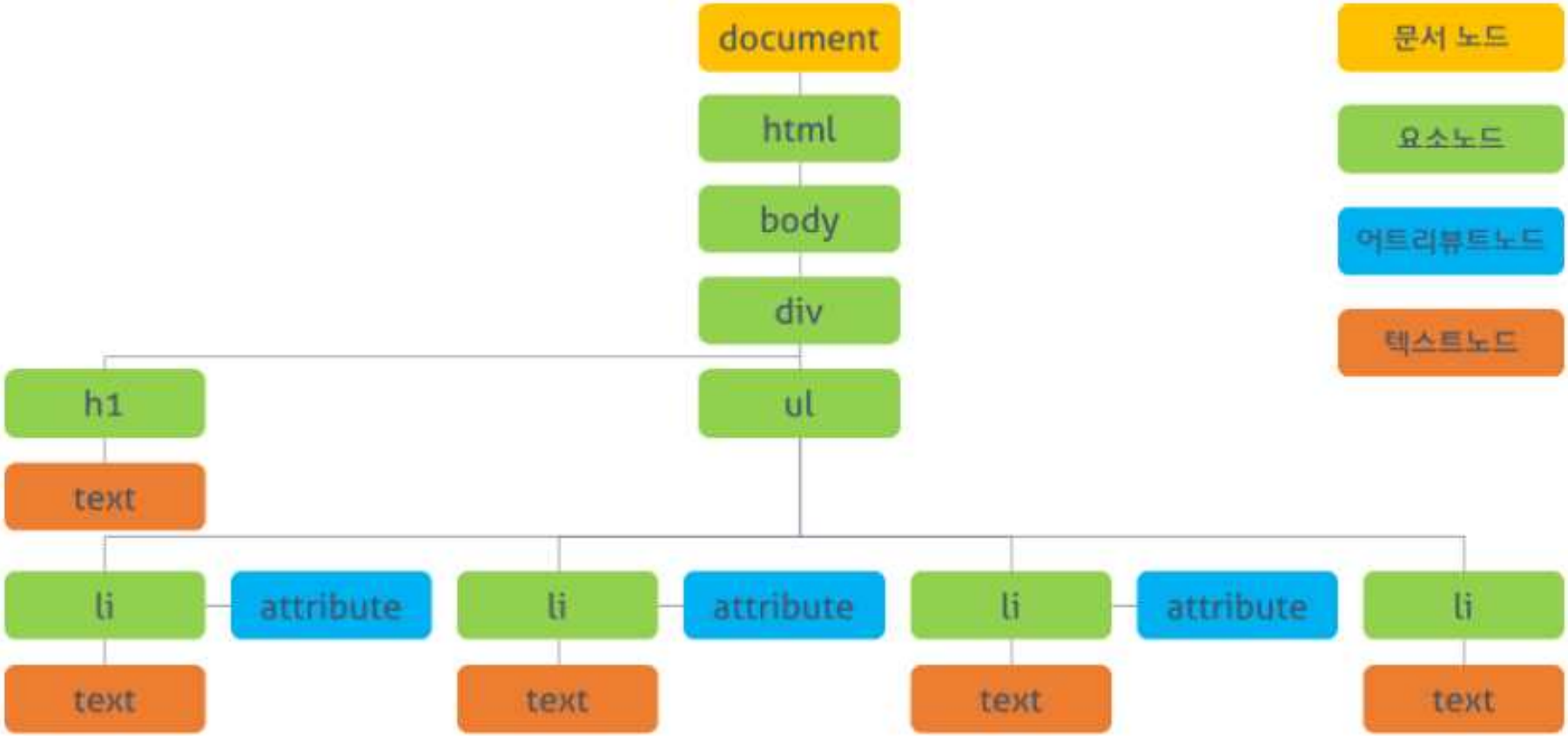
객체 집합	설명
document.documentElement	<html>요소를 반환함.
document.documentMode	웹 브라우저가 사용하고 있는 모드를 반환함.
document.documentURI	HTML 문서의 URI를 반환함.
document.domConfig	HTML DOM 설정을 반환함. (더는 사용하지 않음)
document.embeds	<embed>요소를 모두 반환함.
document.head	<head>요소를 반환함.
document.implementation	HTML DOM 구현(implementation)을 반환함.
document.inputEncoding	HTML 문서의 문자 인코딩(character set) 형식을 반환함.
document.lastModified	HTML 문서의 마지막 갱신 날짜 및 시간을 반환함
document.readyState	HTML 문서의 로딩 상태(loading status)를 반환함.
document.scripts	<script>요소를 모두 반환함.
document.strictErrorChecking	오류의 강제 검사 여부를 반환함.

노드(node)

- HTML DOM은 노드(node)라고 불리는 계층적 단위에 정보를 저장
- 브라우저는 HTML 문서를 로드한 후 HTML 문서에 대한 모델 구성
  - 객체의 트리로 구성 => DOM tree

문서 노드(document node)	HTML 문서 전체를 나타내는 노드임.
요소 노드(element node)	모든 HTML 요소는 요소 노드이며, 속성 노드를 가질 수 있는 유일한 노드임.
속성 노드(attribute node)	모든 HTML 요소의 속성은 속성 노드이며, 요소 노드에 관한 정보를 가지고 있음. 하지만 해당 요소 노드의 자식 노드(child node)에는 포함되지 않음.
텍스트 노드(text node)	HTML 문서의 모든 텍스트는 텍스트 노드임.
주석 노드(comment node)	HTML 문서의 모든 주석은 주석 노드임.

DOM tree

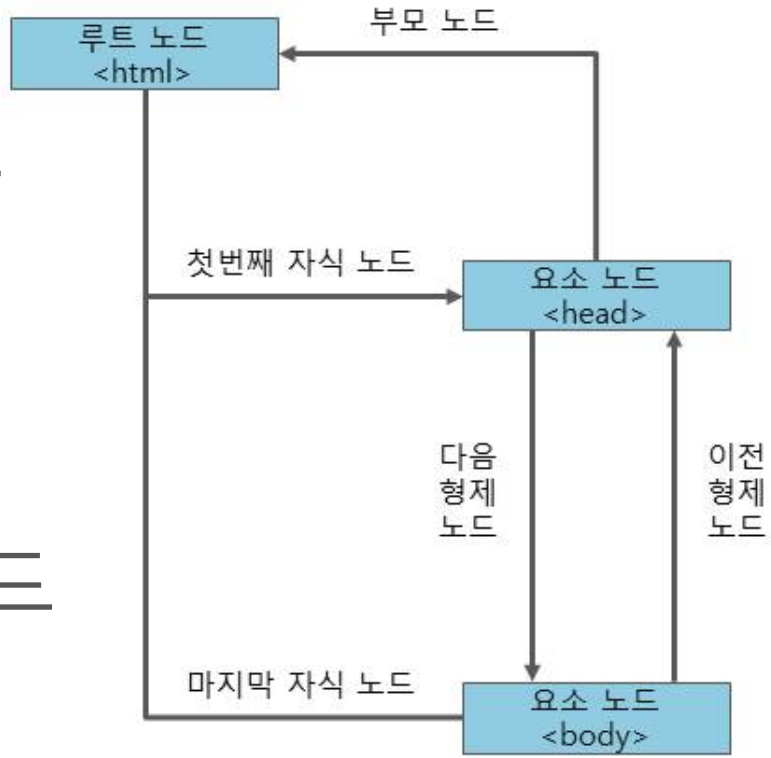

document
html
body
div
h1
ul
text
li
attribute
li
attribute
li
attribute
li
text
text
text
text
문서 노드
요소노드
어트리뷰트노드
텍스트노드

DOM 요소 찾기

- HTML 태그 이름(tag name)을 이용한 선택
- 아이디(id)를 이용한 선택
- 클래스(class)를 이용한 선택
- name 속성(attribute)을 이용한 선택
- CSS 선택자(selector)를 이용한 선택
- HTML 객체 집합(object collection)을 이용한 선택

노드 접근

- getElementsByTagName() 메소드
  - 특정 태그 이름을 가지는 모든 요소를 노드 리스트의 형태로 반환
- 노드 간의 관계를 이용하여 접근
  - parentNode : 부모 노드
  - childNodes : 자식 노드 리스트
  - firstChild : 첫 번째 자식 노드
  - lastChild : 마지막 자식 노드
  - nextSibling : 다음 형제 노드
  - previousSibling : 이전 형제 노드



노드 접근

- parentNode –
  - 부모 노드를 탐색
  - Return: HTMLElement를 상속받은 객체
- firstChild, lastChild –
  - 자식 노드를 탐색
  - Return: HTMLElement를 상속받은 객체
- hasChildNodes()
  - 자식 노드가 있는지 확인하고 Boolean 값을 반환
  - Return: Boolean 값
- childNodes
  - 자식 노드의 컬렉션을 반환
  - Return: NodeList (non-live)
- children
  - 자식 노드의 컬렉션을 반환
  - Return: HTMLCollection (live)
- previousSibling, nextSibling
  - 형제 노드를 탐색
  - Return: HTMLElement를 상속받은 객체
- previousElementSibling, nextElementSibling
  - 형제 노드를 탐색
  - Return: HTMLElement를 상속받은 객체

노드 조작

- nodeName 프로퍼티
  - 노드 고유의 이름을 저장하므로, 수정할 수 없는 읽기 전용 프로퍼티
- nodeValue 프로퍼티
  - 노드의 값을 저장
- nodeType 프로퍼티
  - 노드 고유의 타입을 저장하므로, 수정할 수 없는 읽기 전용 프로퍼티

노드	프로퍼티 값
요소 노드(element node)	1
속성 노드(attribute node)	2
텍스트 노드(text node)	3
주석 노드(comment node)	8
문서 노드(document node)	9

노드의 추가

- appendChild() 메소드
  - 새로운 노드를 해당 노드의 자식 노드 리스트(child node list)의 맨 마지막에 추가
- insertBefore() 메소드
  - 새로운 노드를 특정 자식 노드 앞에 추가

```
부모노드.insertBefore( 새로운자식노드, 기준자식노드);
```

- insertData() 메소드
  - 텍스트 노드의 텍스트 데이터에 새로운 텍스트를 추가식 노드 바로 앞에 추가

```
텍스트노드.insertData(오프셋, 새로운데이터);
```

노드의 생성

- createElement() 메소드를 사용하여 새로운 요소 노드를 생성
- createAttribute() 메소드를 사용하여 새로운 속성 노드를 생성
- createTextNode() 메소드를 사용하여 새로운 텍스트 노드를 생성

노드의 제거

- removeChild() 메소드
  - 자식 노드 리스트에서 특정 자식 노드를 제거
  - 제거된 노드를 반환

- removeAttribute() 메소드
  - 속성의 이름을 이용하여 특정 속성 노드를 제거

노드의 복사

- cloneNode() 메소드는 기존의 존재하는 노드와 똑같은 새로운 노드를 생성하여 반환

노드 변경

- setAttribute() 메소드
  - 값을 변경
  - 속성값을 변경하려는 속성이 존재하지 않으면, 먼저 해당 속성을 생성한 후에 속성값을 설정
- replaceChild() 메소드
  - 기존의 요소 노드를 새로운 요소 노드로 교체
- replaceData() 메소드
  - 텍스트 노드의 텍스트 데이터를 교체

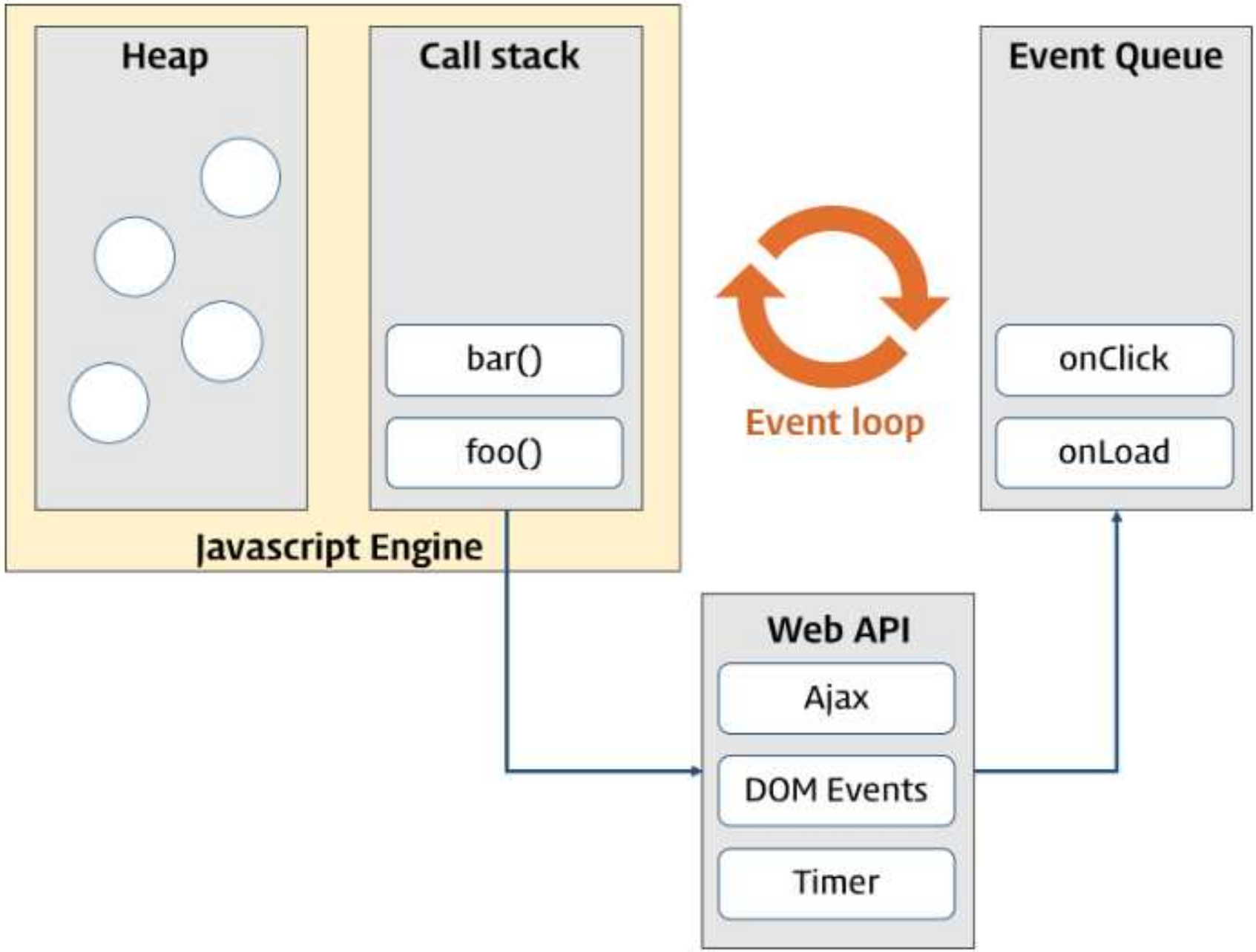
BOM 객체


window
현재 브라우저 창 또는 탭
document
현재 로드된 웹페이지
history
브라우저 히스트리에
기록된 웹페이지들
location
현재 페이지 URL
navigator
브라우저 관련 정보
screen
장치의 디스플레이 정보

이벤트(event)

- 웹 브라우저가 알려주는 HTML 요소에 대한 사건의 발생을 의미
- 브라우저는 이벤트를 감지할 수 있으며 이벤트 발생을 전달해 사용자와 웹페이지는 상호작용(Interaction)
- 이벤트 핸들러
  - 이벤트가 발생하면 일반적으로 함수에 연결
    이벤트가 발생하기 전에는 실행되지 않다가
    이벤트가 발생되면 실행

이벤트 실행 환경


Heap
Call stack
bar()
foo()
Javascript Engine
Event loop
Event Queue
onClick
onLoad
Web API
Ajax
DOM Events
Timer

이벤트 종류

Event	Description
load	웹페이지의 로드가 완료되었을 때
unload	웹페이지가 언로드될 때(주로 새로운 페이지를 요청한 경우)
error	브라우저가 자바스크립트 오류를 만났거나 요청한 자원이 존재하지 않는 경우
resize	브라우저 창의 크기를 조절했을 때
scroll	사용자가 페이지를 위아래로 스크롤할 때
select	텍스트를 선택했을 때
keydown	키를 누르고 있을 때
keyup	누르고 있던 키를 뗄 때
keypress	키를 누르고 뗏을 때
click	마우스 버튼을 클릭했을 때
dbclick	마우스 버튼을 더블 클릭했을 때
mousedown	마우스 버튼을 누르고 있을 때
mouseup	누르고 있던 마우스 버튼을 뗄 때
mousemove	마우스를 움직일 때 (터치스크린에서 동작하지 않는다)

이벤트 종류

Event	Description
mouseover	마우스를 요소 위로 움직였을 때 (터치스크린에서 동작하지 않는다)
mouseout	마우스를 요소 밖으로 움직였을 때 (터치스크린에서 동작하지 않는다)
focus/focusin	요소가 포커스를 얻었을 때
blur/foucusout	요소가 포커스를 잃었을 때
input	input 또는 textarea 요소의 값이 변경되었을 때
change	select box, checkbox, radio button의 상태가 변경되었을 때
submit	form을 submit할 때 (버튼 또는 키)
reset	reset 버튼을 클릭할 때 (최근에는 사용 안함)
cut	콘텐츠를 잘라내기할 때
copy	콘텐츠를 복사할 때
paste	콘텐츠를 붙여넣기할 때

이벤트 종류

이벤트	설명	이벤트	설명
click	요소에 마우스를 클릭했을 때 이벤트가 발생	keydown	키를 눌렀을 때 이벤트가 발생
dbclick	요소에 마우스를 더블클릭했을 때 이벤트가 발생	keyup	키를 떼었을 때 이벤트가 발생
mouseover	요소에 마우스를 오버했을 때 이벤트가 발생	keypress	키를 누른 상태에서 이벤트가 발생
mouseout	요소에 마우스를 아웃했을 때 이벤트가 발생	focus	요소에 포커스가 이동되었을 때 이벤트 발생
mousedown	요소에 마우스를 눌렀을 때 이벤트가 발생	blur	요소에 포커스가 벗어났을 때 이벤트 발생
mouseup	요소에 마우스를 떼었을 때 이벤트가 발생	change	요소에 값이 변경 되었을 때 이벤트 발생
mousemove	요소에 마우스를 움직였을 때 이벤트가 발생	submit	submit 버튼을 눌렀을 때 이벤트 발생
contextmenu	context menu(마우스 오른쪽 버튼을 눌렀을 때 나오는 메뉴)가 나오기 전에 이벤트 발생	reset	reset 버튼을 눌렀을 때 이벤트 발생
		select	input이나 textarea 요소 안의 텍스트를 드래그하여 선택했을 때 이벤트 발생

이벤트 종류

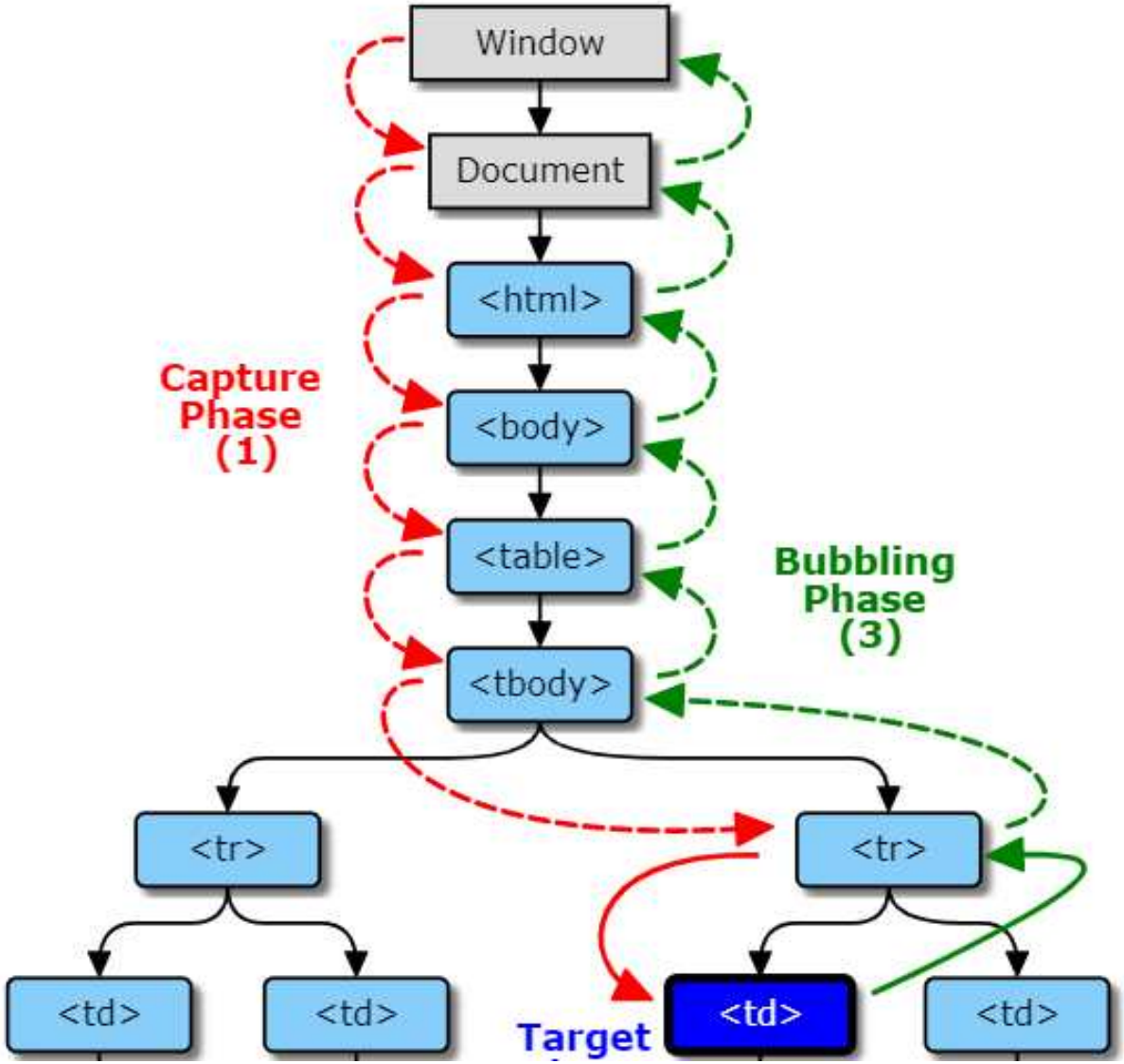
이벤트	설명
load	페이지의 로딩이 완료되었을 때 이벤트 발생
abort	이미지의 로딩이 중단되었을 때 이벤트 발생
unload	페이지가 다른 곳으로 이동될 때 이벤트 발생
resize	요소에 사이즈가 변경되었을 때 이벤트 발생
scroll	스크롤바를 움직였을 때 이벤트 발생

이벤트의 연결

- 이벤트 핸들러(event handler)
  - 사용자가 실제 이벤트를 발생시켰을 때 그 이벤트에 대응하여 처리하는 것
  - 앞에 'on'을 붙여 주고 이벤트에 대한 동작(함수)을 처리
- 인라인 방식
- 프로퍼티 방식
  - 동일 이벤트의 다수 이용 불가
- 표준이벤트 적용

addEventListener 메소드 방식

이벤트의 흐름


Window
Document
<html>
<body>
<table>
<tbody>
<tr>
<tr>
<td>
<td>
<td>
<td>
Capture Phase (1)
Bubbling Phase (3)
Target

이벤트 객체

- 이벤트를 발생시킨 요소와 발생한 이벤트에 대한 정보를 제공
- 이벤트가 발생하면 event 객체는 동적으로 생성
- 이벤트를 처리할 수 있는 이벤트 핸들러에 인자로 전달
- Event.target, Event.currentTarget, Event.type, Event.cancelable, Event.eventPhase,

기본 동작 변경

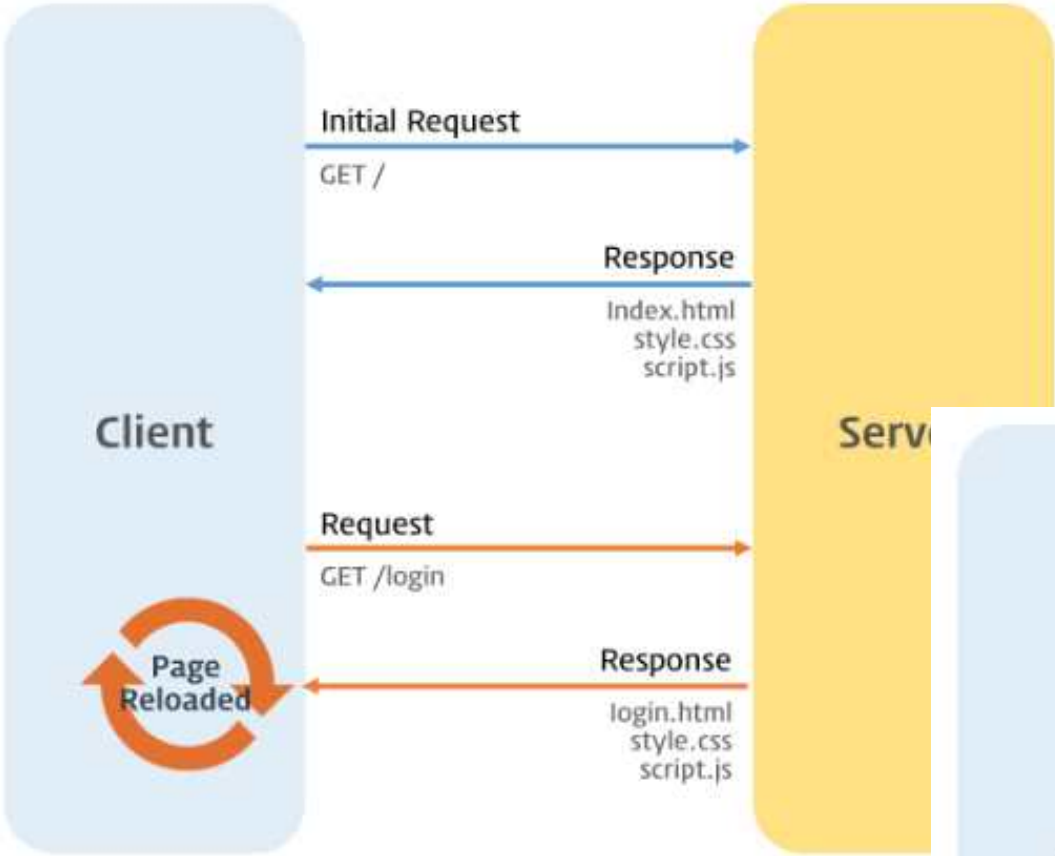
- Event.preventDefault()
  - 해당 이벤트 요소의 기본 동작 정지
- Event.stopPropagation()
  - 이벤트 전파 정지

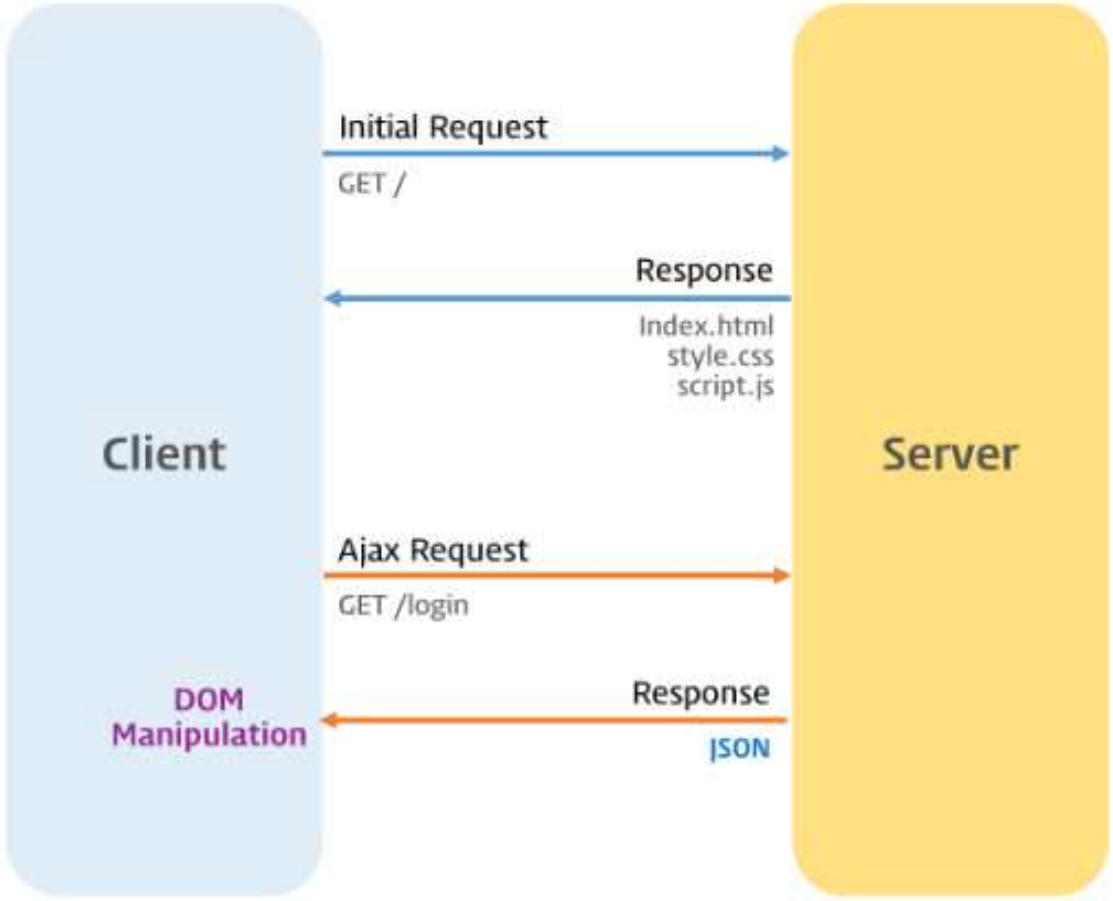
장비의 이벤트

- 장비의 센서에 의해 발생하는 이벤트
  - 휴대폰의 움직임 등

Ajax (Asynchronous JavaScript and XML)

- JSON





예외 처리

- 예외(exception)
  - 프로그램이 실행 중에 발생하는 런타임 오류를 의미
- throws

```
try {
    예외를 처리하길 원하는 실행 코드;
} catch (ex) {
    예외가 발생할 경우에 실행될 코드;
} finally {
    try 블록이 종료되면 무조건 실행될 코드;
}
```

Strict mode

- 자바스크립트 코드에 더욱 엄격한 오류 검사를 적용

```
"use strict"   // 전체 스크립트를 strict 모드로 설정함.
try {
   num = 3.14; // 선언되지 않은 변수를 사용했기 때문에 오류를 발생시킴.
} catch (ex) {
   document.getElementById("text").innerHTML = ex.name + "<br>";
   document.getElementById("text").innerHTML += ex.message;
}
```

strict mode 특징

대상	제한 사항
변수	선언되지 않은 변수나 객체를 사용할 수 없음.
	eval() 함수 내에서 선언된 변수는 외부에서 사용할 수 없음.
프로퍼티	읽기 전용 프로퍼티에는 대입할 수 없음.
	한 프로퍼티를 여러 번 정의할 수 없음.
함수	함수를 구문이나 블록 내에서 선언할 수 없음.
매개변수	매개변수의 이름이 중복되어서는 안됨.
	arguments 객체의 요소 값을 변경할 수 없음.
문자열	문자열 "eval"과 "arguments"는 사용할 수 없음.
8진수	숫자 리터럴에 8진수 값을 대입할 수 없음.
this	this 포인터가 가르키는 값이 null이나 undefined인 경우 전역 객체로 변환되지 않음.
delete	delete 키워드를 사용할 수 없음.
with	with 문을 사용할 수 없음.
예약어	다음 예약어들은 사용할 수 없음. (implements, interface, let, package, private, protected, public, static, yield)

ES6

- let, const와 블록 레벨 스코프
- 이터레이션과 for...of 문
- 매개변수 기본값

- 템플릿 리터럴
- 화살표 함수
- Rest 파라미터, Spread 문법, Rest/Spread 프로퍼티
- 클래스
- 모듈
- 프로미스
- 7번째 타입 심볼(Symbol)
- 제너레이터와 async/await

화살표 함수(arrow function)

```
// 매개변수 지정 방법
() => { ... } // 매개변수가 없을 경우
 x => { ... } // 매개변수가 한 개인 경우, 소괄호를 생략할 수 있다.
(x, y) => { ... }
// 매개변수가 여러 개인 경우, 소괄호를 생략할 수 없다.

// 함수 몸체 지정 방법
x => { return x * x } // single line block
x => x * x
// 함수 몸체가 한 줄의 구문이라면 중괄호를 생략할 수 있으며 암묵적으로 return된다. 위 표현과 동일하다.

() => { return { a: 1 }; }
() => ({ a: 1 })
// 위 표현과 동일하다. 객체 반환 시 소괄호를 사용한다.
() => { // multi line block.
        const x = 10;
        return x * x;
};
```

마무리