

ch03.JavaScript

디지털마케팅 프로그래밍

by Wooyoung Choi, KMU Digital Marketing MBA, 2025

JavaScript

JavaScript

- 웹페이지의 상호작용을 만들기위한 크로스플랫폼 기반 객체기반 스크립트 언어
- 상호작용: 복잡한 애니메이션, 클릭가능한 버튼, 팝업 메뉴 등
- node.js: Server-side JavaScript
- Client-side JavaScript: 브라우저와 DOM 제어를 위한 객체 제공
- Server-side JavaScript: 서버에서 JavaScript 실행과 관련한 객체 제공

JavaScript == java??

JavaScript	Java
객체 기반	클래스 기반
스크립트 언어	컴파일 언어
동적타이핑 (자료형 선언 필요 x)	정적타이핑 (자료형 선언 필요)
붕어빵	붕어

ECMA Script

- ECMA International에서 JavaScript 표준화
- ECMA-262 표준은 IOS-16262로서 ISO에 의해 승인됨
- ECMAScript 사양을 준수하는 범용 스크립트 언어: JavaScript
- ES14 이 최신 표준
- Google Tag Manager는 ES6 일부 기능을 지원
- ES5를 기준으로 설명 예정

웹 브라우저가 JavaScript를 이해하는 법

- JavaScript Engine
 - V8(Google Chrome)
 - SpiderMonkey(Mozilla Firefox)
 - JavascriptCore(Apple Safari)
 - Chakra(Microsoft Edge)

Hello, world!

```
// Print at console
```

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

```
// Alert message
```

```
alert( 'Hello, world!' );
```

```
// Write to document
```

```
document.write( 'Hello, world!' );
```

Expressions and Statements

표현식과 연산자(Expressions and Operators)

- 표현식: 값으로 평가될 수 있는 코드의 단위

```
// Arithmetic Expressions
```

```
42;
```

```
2+1;
```

```
// String Expressions
```

```
'hello';
```

```
'hello'+'world';
```

```
// Logical Expressions: true or false 로 평가될 수 있는 표현식
```

```
10>9;
```

```
10===9;
```

```
// Primary Expressions
```

```
'hello world'; // A string literal
```

```
23;           // A numeric literal
```

```
true;        // Boolean value true
```

```
sum;         // Value of variable sum
```

```
this;        // A keyword that evaluates to the current object
```

표현식과 연산자(Expressions and Operators)

// Left-hand-side Expressions:

// variables such as i and total

i = 10;

total = 0;

// properties of objects

var obj = {}; // an empty object with no properties

obj.x = 10; // an assignment expression

// elements of arrays

array[0] = 20;

array[1] = 'hello';

// Assignment Expressions

average = 55;

구문(Statement)

- 실행가능한(executable) 최소의 독립적인 코드조각
- 모든 expression은 statement이지만, 모든 statement는 expression이 아님
- $\text{expression} \subset \text{statement}$
- Expressions Statements, Conditional Statements, Loops and Jumps

Statement vs Expression

	Statement	Expression
역할	코드 실행, 프로그램 제어	값 생성
결과	없음	값
사용 위치	if, for, while 등	연산식, 조건식, 함수 인자 등

Statement vs Expression

- 변수 선언 및 할당: `var x = 10; let y = 20; const z = 30;`
- if 문: `if (x > 0) { console.log("x는 양수입니다."); }`
- for 문: `for (let i = 0; i < 10; i++) { console.log(i); }`
- while 문: `while (x < 10) { x++; }`
- 함수 선언: `function myFunction(a, b) { return a + b; }`
- 클래스 선언: `class MyClass { constructor(name) { this.name = name; } }`
- 브라우저 API 호출:
`document.getElementById("myElement").innerHTML = "Hello";`
- 수치 리터럴: `10, 3.14, -20`
- 문자열 리터럴: `"Hello", 'World', "JavaScript"`
- 변수 참조: `x, y, z`
- 연산식: `2 + 3, 4 * 5, 10 / 2`
- 함수 호출: `math.abs(-5), console.log("Hello"), myFunction(1, 2)`
- 조건식: `x > 0, y === "abc", z !== true`
- 논리 연산자: `a && b, c || d, !e`
- 배열 리터럴: `[1, 2, 3], ["a", "b", "c"], [x, y, z]`
- 객체 리터럴: `{name: "John", age: 30}, {x: 10, y: 20}, {f: function() {}}`

연산자(Operators)

- 단항, 이항, 삼항 연산을 수행할 때 행위를 정의하는 기호
- Unary Operators(단항연산자): delete, void, typeof, ~, !
- Arithmetic Operators(산술연산자): +, -, *, /, %, **
- Relational Operators(관계연산자): >, <, >=, <=, in, instanceof
- Equality Operators(같음연산자): ==, !=, ===, !==
- Ternary Operators(삼항 연산자): ? : (ex: condition ? ifTrue : ifFalse)
- Bitwise shift Operators(비트 시프트 연산자): >>, <<, >>>
- Binary Bitwise Operators(이진 비트 연산자): &, |, ^
- Binary Logical Operators(이진 논리 연산자): &&, ||, ??

변수(Variable)

- 변수: 아직 알려지지 않거나 어느정도 까지만 알려진 양이나 정보에 대한 상징적인 이름
- 선언(declare) -> 초기화(initialize) -> 할당(assign)을 거쳐 생성
- 3 types of variable Declarations(After ES6)
 - var: 선언과 초기화가 한번에 이루어짐
 - let: 선언과 초기화가 분리되어 진행, 재할당 가능(mutable)
 - const: 선언과 초기화가 분리되어 진행, 재할당 불가(immutable)
- 기본적으로 const를 사용하되, 재할당이 필요할 경우 let을 사용

연산자 실습

```
var num1 = 3;
```

```
var num2 = 42;
```

```
var numstr = '42';
```

```
num1+num2;
```

```
num1>num2;
```

```
num2==num3;
```

```
num2===num3;
```


변수 실습

```
// declare and assign
```

```
const foo = 1;
```

```
let bar = 2;
```

```
console.log(foo, bar);
```

```
//immutable test
```

```
foo=3;
```

```
bar=4;
```

```
console.log(foo, bar);
```

변수 이름규칙(Variable Naming Convention)

- 변수의 이름은 식별자(Identifiers)라고 불리며 아래의 규칙을 따름
- JavaScript의 식별자는 문자, 밑줄(_) 혹은 달러 기호(\$)로 시작해야 함
- 숫자는 첫 글자에 사용될 수 없음
- 대소문자를 구분함
- Unicode 문자 사용가능(ex) 한글..)
- `javaScriptStyleHungarianNotation`의 형태로 Capitalize

Data Structures and Types

Primitive Data Types

- Boolean: `True` or `False`
- null: null 값을 나타내는 키워드
- undefined: 값이 정의되지 않은 최상위 속성
- Number: 정수 또는 실수(`42` or `3.14`)
- BigInt: 임의정밀도의 정수($2^{53}-1$ 보다 큰 정수, Math 사용 불가)
- String: 문자열(`"Foo"`)
- Symbol(ES6): 인스턴스가 고유하고 불변인 자료형(Class-like)

Object

- 데이터 및 데이터 작업에 대한 지침을 포함하는 데이터 구조

```
var person = {};
```

```
var person = {  
  name: 'John Doe',  
  age: 40,  
  email: 'johndoe@gmail.com',  
  interests: ['music', 'coding'],  
};
```

데이터 형 변환(Type Casting)

- 문자열 -> 정수, 실수: parseInt('숫자', 진수), parseFloat('실수')
- object -> 문자열: obj.toString()

```
var num1 = 10;
```

```
var num2 = '11';
```

```
var num3 = '3.14';
```

```
var str1 = 'hello';
```

```
console.log(num1+num2) // '+' as concat operator
```

```
console.log(num1+parseInt(num2, 10))
```

```
console.log(num1+parseFloat(num3))
```

```
num1.toString()
```

문자열(String)

- 0글자 이상의 문자를 표현하기 위해 사용하는 자료형

문자열 리터럴(String Literal)

```
var str1 = 'foo';
```

```
var str2 = "bar";
```

문자열 메소드(String Methods)

문자열 찾기

`str.indexOf(searchValue[, fromIndex])`

```
var paragraph = 'The quick brown fox jumps over the lazy dog. If the dog  
barked, was it really lazy?';
```

```
var searchTerm = 'dog';
```

```
console.log(paragraph.indexOf(searchTerm));
```

```
console.log(paragraph.indexOf(searchTerm, (paragraph.indexOf(searchTerm)+1)));
```


특정 문자열로 시작, 끝

```
str.startsWith(searchString[, position])
```

```
str.endsWith(searchString[, position])
```

```
var paragraph = 'To be, or not to be, that is the question.';
```

```
//startswith
```

```
console.log(paragraph.startsWith('To be'));
```

```
console.log(paragraph.startsWith('not to be', 10));
```

```
//endswith
```

```
console.log(str.endsWith('question.'));
```

```
console.log(str.endsWith('to be', 19));
```

문자열 합치기

str.concat(string, string[, ...])

```
var hello = 'hello';
```

```
console.log(hello.concat(' ', 'world!'));
```

```
var userName = prompt("What's your name?");
```

```
console.log(hello.concat(' ', ` ${userName} `));
```

문자열 분할하기

str.split()

```
var paragraph = 'The quick brown fox jumps over the lazy  
dog. If the dog barked, was it really lazy?';
```

```
var words = paragraph.split(' ');
```

```
console.log(words);
```

```
var words = paragraph.split(' ', 4);
```

```
console.log(words);
```

문자열 추출하기

```
str.substr(start[, length]);
```

```
str.substring(indexStart[, indexEnd]);
```

```
str.slice(beginIndex[, endIndex]);
```

```
var word = 'Google Analytics';
```

```
console.log(word.substring(3));
```

```
console.log(word.substring(3, 9));
```

```
console.log(word.substr(3));
```

```
console.log(word.substr(3, 9));
```

substring, slice의 차이

- substring: -값은 0으로 인식, 시작>종료 일 경우 값을 치환
- slice: -값 사용가능, 시작위치가 -일 경우 종료값도 -값 지정

```
var word = 'Google Analytics';  
  
//success  
console.log(word.substring(3,0));  
console.log(word.substring(3,-4));  
console.log(word.slice(3,-4));  
console.log(word.slice(-4,-2));  
  
//fail  
console.log(word.slice(-3,4));  
console.log(word.slice(-3,-4));
```

공백 제거하기

`str.trim()`

`str.trimStart()`

`str.trimEnd()`

```
var word = '    \t\nHello \t\t\t\n\n\n\n    ';
```

```
console.log(word.trim());
```

묶음자료형(Group Elements, Collections)

- 0개 이상의 자료를 하나의 변수로 묶어 표현하는 자료형

배열(Array)

- 이름과 인덱스로 참조되는 정렬된 값들의 집합
- [] 로 정의되며 인덱스를 활용하여 값에 접근

배열 생성하기

```
var animals = new Array(element, ..);
```

```
var animals = Array(element, ..);
```

```
var animals = [element, ..];
```

```
// Declare and Assign Empty array
```

```
var animals = new Array(arrayIndex);
```

```
var animals = [];
```


배열에 값 추가하기, 치환하기, 제거하기

```
var animals = [];  
//추가하기  
animals[0] = 'Cat';  
animals[1] = 'Duck';  
animals.push( 'Dog' );  
animals.push( 'Hamster' );  
console.log(animals);  
//치환하기  
animals[1] = 'Chicken';  
console.log(animals);
```

```
// pop: 마지막 요소 제거하고 반환하는 기능  
var lastElement = animals.pop( );  
console.log(lastElement)  
// shift: 첫 요소 제거하고 반환하는 기능  
var firstElement = animals.shift( );  
console.log(firstElement)
```

배열 추출하기

`Array.slice(startIndex, uptoIndex)`

```
var nums = [1,2,3,4,5];  
console.log(nums.slice(2,3));  
console.log(nums);
```

요소 치환하기

`Array.splice(index, count_to_remove, addElement1, addElement2, ...)`

```
var nums = [1,2,3,4,5];  
nums.splice(2,2, 'a', 'b', 'c');  
console.log(nums);  
var removed = nums.splice(2,2, 'd', 'e', 'f');  
console.log(nums);  
console.log(removed);
```

배열 뒤집기, 정렬하기

Array.reverse()

Array.sort()

```
var nums = [3.14, -100, 2.71828, 100, 0, -1];
```

```
nums.reverse( );
```

```
console.log(nums);
```

```
// sort(something's wrong)
```

```
nums.sort( );
```

```
console.log(nums);
```

```
// sort(asc or desc)
```

```
nums.sort(function(a,b){return a-b;}); //asc
```

```
nums.sort(function(a,b){return b-a;}); //desc
```

고차원 배열

```
var a = [];  
for (i = 0; i < 4; i++) {  
    a[i] = [];  
    for (j = 0; j < 4; j++) {  
        a[i][j] = '[' + i + ', ' + j + '];  
    }  
}  
console.log(a);
```