

JavaScript

- 자바스크립트는
 - 웹 브라우저를 위한 스크립트 언어로 처음 만들어짐
 - 웹의 활성화와 함께 다양한 방면으로 적용 범위를 넓혀가고 있음
 - 언어적으로 독특하면서도 실용적 측면을 함께 가지고 있음
 - 고수준의 프로그래밍을 하기에 적합

JavaScript의 탄생

- 1995년, Netscape사의 엔지니어 Brendan Eich가 개발
- 최초의 이름은 LiveScript
- 1995년 Netscape2 최종 베타에서 JavaScript로 이름 변경

- 1996년, 넷스케이프에서 ECMA에 ECMAScript 표준안 제출
- 1999년 ECMAScript 3 Edition 릴리즈
- 2008년 ECMAScript 4 Edition 제정 포기 (회원사간 의견 충돌)
- 2009년 ECMAScript 5 Edition 제정
- 2015년 ECMAScript 6

- 스크립트 언어 표준화
 - JavaScript
 - ActionScript
 - JScript

JavaScript 사용 사례

- 웹 브라우저
 - 사용자 이벤트
 - DOM 제어
- 서버 환경
 - Node.js
 - MongoDB 등 데이터 베이스에서의 동작
- 네트워크
 - js 파일을 전송.
 - 모바일 앱의 동작 로직을 전송

- 사용자 이벤트
- DOM 다루기

JavaScript

- 데이터 다루기
- 제어문
- 함수
- 객체
- 상속

코드 작성 규칙(권장)

- 카멜 케이스(Camel Case)
 - firstName, lastName, masterCard, interCity.
 - 소문자로 시작
- 하이픈(-)은 금지.
- 대소문자 구별(Case sensitive)



정보 출력하기

- 출력 방법
 - console.log
- 정보 출력
 - 여기까지 흐름이 맞는가?
 - 값이 맞는가?
 - 흐름 순서 확인

- 세미콜론은 생략 가능
- 자동으로 문장 끝에 세미콜론이 붙는다.
- 사용하는 것을 권장

- strict 모드
 - `'use strict'`
- 보다 강력한 문법 제약
 - var 없이 변수 선언 불가
 - 변수의 유효 범위 체크 강화
 - 변수 이름에 예약어 검사 강화
 - 등등
- 이전 실습을 strict 모드로 동작시켜 보자.

데이터 다루기

변수/상수

변수와 상수

- 변수
 - Variable
 - 변하는 값을 다룬다.
- 상수
 - Constant
 - 한번 값이 대입되면 변하지 않는다.
 - ES6

변수 선언

- 변수 선언하기, 값 변경

```
var varVal = 1
```

```
varVal = 2
```

- 변수 선언과 대입 별도

```
var valVal2
```

```
valVal2 = 'JavaScript'
```

상수 선언

- 상수 선언하기, 값 변경(에러)

```
const constVal1 = 1
```

```
constVal1 = 2
```

- 상수 선언과 대입 별도

```
const constVal2
```

```
constVal2 = 'hello'
```

이름 규칙

- 첫 글자 : a-z, A-Z, _(언더스코어), \$
- 그 이후 : 알파벳, 숫자, _, \$
- 예약어 사용 불가(var, switch, case, default, if, else, this, ...)
- 하이픈(-) 불가

- 다음 중 변수 이름으로 사용할 수 있는 것은?
 1. 3value
 2. name
 3. kor-score
 4. default

- 01.Data
 - var_const
 - var_naming

변수 사용 에러

- 정의되지 않은 변수를 사용하면?

```
if ( z == 3 ) {  
    console.log('z is 3');  
}
```

ReferenceError: z is not defined

데이터 타입

- 리터럴(Literal) : 변하지 않는 수
 - 숫자(Number) : 10.50, 1001
 - 문자(String) : 'Hello', "World"
 - 평가식(Expression) : $5 + 6$

- 데이터의 형태
 - true/false
 - 1, 3.14
 - 'Hello JavaScript'
 - { name : 'IU' }
 - [1, 2, 3]

- 변수/상수 선언시 데이터 타입은?

- 다른 언어의 변수 선언 코드 예

```
int intVal1 = 1
```

```
var intVal2 : Int = 2
```

- Java Script

```
var intVal = 3
```

변수의 데이터 타입

- 변수의 타입 지정 안함(weak typing)
- 값에 따라서 타입이 결정(dynamic typing)

```
var strVal = '문자열';  
strVal = 12;
```

변수의 데이터 타입

- 변수의 데이터 타입 판단 : typeof

- 사용법 : typeof x, typeof(x)

```
var x = 3
```

```
typeof(x); // number
```

```
val = '10'
```

```
console.log('"10" type : ', typeof(val)) // string
```

- 변수 타입 판단하기

```
if ( (typeof x) == 'number' ) {  
    console.log('x는 Number 타입');  
}  
  
if ( typeof(x) == typeof(y) ) {  
    console.log('x, y 두 타입은 같다. ');  
}
```

데이터 타입

- 데이터 타입
 - boolean
 - number
 - string
 - object
 - Array
 - function

- 01.Data/var_type

undefined, null, NaN

- null, undefined
 - null : 어떠한 객체도 참조되지 않은 상태
 - undefined : 값이 지정되지 않은 상태

undefined

- undefined
 - 값이 지정되지 않은 상태
 - 변수 선언만 했을 때
 - 함수 호출시 파라미터로 값이 바인딩 되지 않았을 때
 - 리턴값이 없는 함수의 반환값

- 코드

```
var x;  
console.log(x); // undefined
```

undefined 구분하기

- if를 이용해서 유효한 값 판단하기

```
if ( x ) {}
```

```
if ( x == undefined ) {}
```

```
if ( x === undefined ) {}
```

- undefined과 유사한 사용

```
x = null;  
if ( x ) {  
    // x는 유효한 값  
}  
  
if ( x == undefined ) {  
    // x는 유효한 값  
}
```

null과 undefined

- undefined 와 비교

```
null == undefined
```

```
null !== undefined
```

- null과 undefined

- typeof undefined // 'undefined'

- typeof null // 'object'

- null과 undefined 비교

```
x = null
```

```
if ( x === undefined ) {}
```

```
else { // undefined 와는 다르다. }
```

- 01.Data
 - undefined
 - null

주요 데이터 타입

- 부울 타입

```
val = ( 1 == 1)
```

```
val2 = true
```

```
typeof val // 'boolean'
```

- 값 비교

```
if ( val ) {
```

```
    console.log( '참' );
```

```
}
```


Numbers

- 숫자형 : Numbers

- 정수형과 실수형 별도 존재 안함

15 // 10진수

0x0F // 16진수

- 타입 변환

- parseInt(), parseFloat()

parseInt('123') // 123

parseInt('Hello') // NaN - isNaN 으로 비교

NaN

- NaN : Not a Number

- 0 / 0

- 'Hello'

- NaN 판단하기 : isNaN()

- `isNaN(1)`

- `isNaN('a')`

- NaN 판단

- ~~`x === NaN`~~

- ~~`x === undefined`~~

- Number.NaN

- 실수와 정수는 같은 타입

```
123 == 123.0 // true
```

- 실수를 정수로 변환

- Math 내장 객체

```
Math.round(3.14)
```

```
Math.ceil(3.14)
```

```
Math.floor(3.14)
```

- Infinity : 무한대의 숫자

```
val = 1 / 0; // Infinity
```

```
val2 = -1 / 0; // -Infinity
```

```
Infinity + 1 = Infinity
```

- NaN과 비교

```
isNaN(val) : false
```

- 01.Data
 - boolean
 - numbers
 - nan
 - infinity

Object 타입

Object

- 객체 타입 : Object
- 객체 리터럴 정의 {}
- 프로퍼티(property)
 - dot으로 접근
 - [] 연산자로 접근
- 메소드

Object

- Object Literal

```
var iu = {  
  name: 'IU',  
  age: 20  
};
```

- 객체 값 접근

```
iu.name = '아이유';  
var age = iu.age;
```

- 비교

```
typeof iu // 'object'
```


Object

- 같은 값을 가진 객체

```
var clone = {  
    name: '아이유',  
    age: 20  
}  
  
iu == clone // false
```

- 같은 객체 참조

```
var same = iu;  
  
iu == same // true
```

Object

- Object 타입을 이용한 객체 생성

```
var ty = new Object();
```

```
ty.name = '태연';
```

```
ty["age"] = 20;
```

- 01.Data
 - object

문자열

- 문자열 Literal 정의

```
var str = 'Hello JavaScript';
```

- 문자열 타입

```
typeof str // 'string'
```

- 문자열 덧붙이기

```
var str2 = 'Hello' + ' JavaScript';
```

```
var str3 = 'Hello'.concat(' JavaScript');
```

- 문자열 비교

```
str == str2 // true
```

```
str === str3 // true
```

문자열 다루기

- 문자열

`charAt()` : 인덱스에 해당하는 문자

- 문자열 내 찾기 : `indexOf`, `lastIndexOf`

```
var str = 'Hello JavaScript';
```

```
str.indexOf('Sc') // 10
```

```
str.indexOf('sc') // -1
```

문자열 다루기

- 부분 문자열 : slice, substr, substring

```
str.slice(1, 4); // ell
```

```
str.substr(1, 4); // ello
```

- 변환

```
toLowerCase(), toUpperCase() : 대소문자 변환
```

```
trim() : 문자열 앞뒤의 공백 문자 삭제
```

- 타입 변환

```
(2).toString()
```

```
2 + '0'
```


- String 객체

```
var strObj = new String('Hello JavaScript');  
typeof strObj // 'object'
```

- 비교

```
str == strObj // true  
str === strObj // false
```

- 01.Data/string

배열

- 기본형

```
var num = 10;
```

```
var str = '자바 스트립트'
```

- 배열, 해쉬

```
var arr = [10, 20, 30, 40];
```

```
var hash = { x:300, y:200 };
```

- 배열

- 다수의 값을 한번에 다루기 위한 타입
- 데이터 다루기 : 인덱스를 이용

- 생성

```
var array = ['Porsche', 'BMW', 'Mercedes Benz'];  
var array2 = new Array('Bus', 'Metro', 'Walk');
```

- 2차원 배열

```
var arr2 = [ [1,2,3], [4,5,6], [7,8,9] ];  
arr2[0][1];  
arr2[1][2] = 10;
```

- 배열 정의

```
var cars = ['Mercedes', 'Volvo', 'BMW'];
```

- 원소 개수

```
cars.length
```

- 배열 원소 접근 : [INDEX]

```
cars[0]
```

```
cars[4] // ???
```

배열 수정

```
var cars = ['Mercedes', 'Volvo', 'BMW'];
```

- 추가

```
cars.push('poli');           // [ 'Mercedes', 'Volvo', 'BMW', 'Poli' ]
cars[4] = 'Hyundai';         // [ 'Mercedes', 'Volvo', 'BMW', 'Poli', 'Hyundai' ]
```

- 원소 삭제

```
cars.splice(3,1);           // [ 'Mercedes', 'Volvo', 'BMW', 'Hyundai' ]
cars.pop();                 // [ 'Mercedes', 'Volvo', 'BMW' ]
```

- 변경

```
cars[1] = 'Porche';         // [ 'Mercedes', 'Porche', 'BMW' ]
```

배열 정렬

```
var cars = ['Mercedes', 'Volvo', 'BMW'];
```

- 정렬

```
cars.sort();  
// [ 'BMW', 'Mercedes', 'Volvo' ]  
cars.sort(function(a, b) { return a.length > b.length });  
// [ 'BMW', 'Volvo', 'Mercedes' ]
```

- 역순

```
cars.reverse();  
// [ 'Mercedes', 'Volvo', 'BMW' ]
```


배열 결합, 분할

```
var cars = ['Volvo', 'Mercedes', 'BMW'];
```

- 결합

```
var cars2 = cars.concat(['Audi', 'Toyota']);  
// ['Volvo', 'Mercedes', 'BMW', 'Audi', 'Toyota']
```

- 분할

```
var german = cars2.slice(1,4);  
// [ 'Mercedes', 'BMW', 'Audi' ]
```

- 01.Data/array

배류 타입, 참조 타입

기본 타입과 참조 타입

- 기본 타입(primitive type)

- 다른 변수에 할당시 복사

```
var a = 10;
```

```
var b = a;
```

```
a = 20;
```

```
b // 10
```

- String, Number, Boolean은 기본 타입

기본 타입과 참조 타입

- 참조 타입(reference type)

- 실제 값의 위치 참조

```
var c = { foo : 10 };
```

```
var d = a;
```

```
c.foo = 20;
```

```
d.foo // 20
```

- String, Number, Boolean을 제외하고는 모두 참조 타입

- 01.Data/referenceType

형 변환

- 명시적 형 변환

- 변환 목적의 코드 사용

`(2).toString()` // '2'

`parseInt("8")` // 8

- 묵시적 형 변환
 - 다른 타입과 혼합 사용
123 + "abc" // ?
true + 1

- 01.Data/typeConvert