Modern Javascript

SPARCS 세미나

rodumani 정창제

급하게 만든 이유로 슬라이드
디자인은 양해 부탁드립니다

이 세미나의 예상 청중

- 1개 이상의 프로그래밍 언어 경험이 있고
  - JavaScript를 써본적 없음
  - JavaScript를 써본적 있음

내용이 많을 수 있는데
모든걸 기억할 필요는 없고
“아 이런게 있구나”만 알면 됩니다

1. Javascript 개요

- 스크립트 언어
- 객체 기반
- C-Style Syntax*

* C++, Obj-C, Java, C# 등이 있다

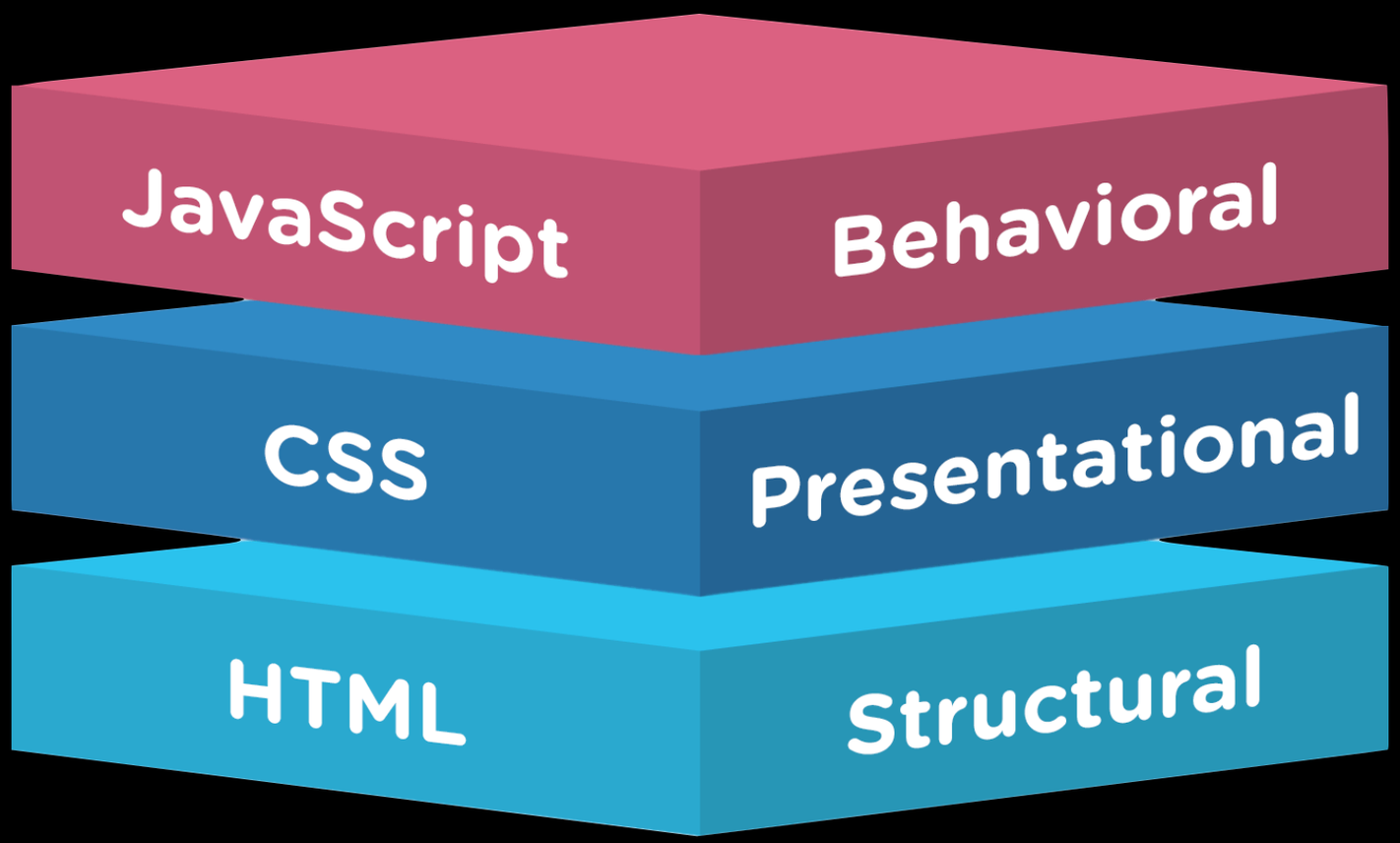
≠

* JavaScript라는 이름은 Java의 유명세에 편승하기 위해서 일부러 헷갈리게 지었다고 한다

JAVA *is to* JAVASCRIPT *as* HAM *is to* HAMSTER

"Java와 JavaScript의 관계는 사자와 바다사자다"

Javascript는 웹페이지를 위한 언어로 탄생!



웹의 동적인 부분에 대한 구현을 담당한다

Server

Browser

Server

Browser

이 세미나에서는

JavaScript 언어 자체만 다룹니다

* 다음주 목요일 같은 시간에 Vue.js 세미나를 별도로 하려고 합니다

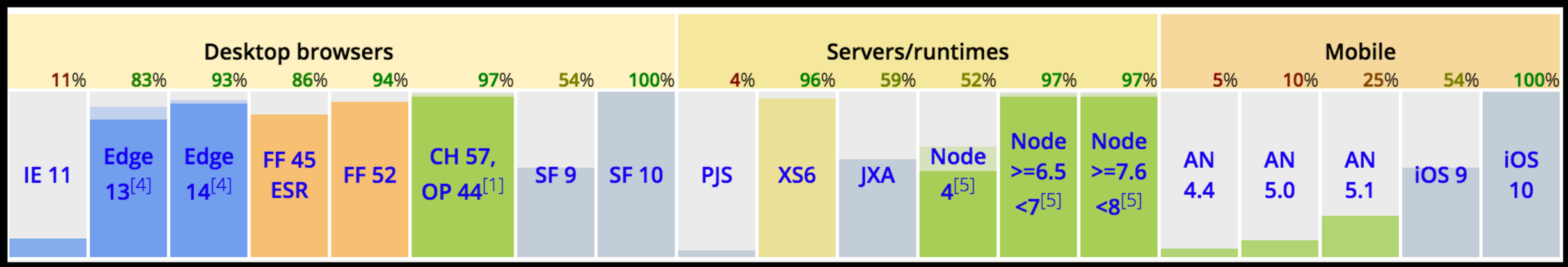
Modern JavaScript는 뭔가요?

Modern JavaScript

- ECMAScript 6 부터 Modern JS라고 부른다
- ECMAScript는 Ecma International에서 정하는 Scripting-Language Specification Standard
- ECMAScript 는 언어 명세, JavaScript는 구현체
- ES6 이후부터 매년 표준 업데이트 예정

ECMAScript 6

- ECMAScript 6번째 에디션
- 2015년 6월에 공표되어서 ECMAScript 2015가 공식명
- 새로운 문법과 언어요소들이 매우 많이 추가되었다
- 대부분의 Runtime에서 지원이 **2017년에** 완성되었다



http://www.ecma-international.org/ecma-262/6.0/

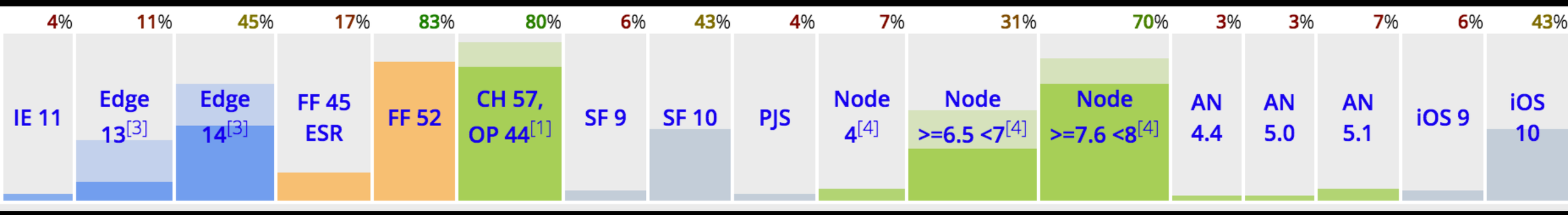
ES6 - 새로운 기능들

- Arrow Function
- Class
- Template String
- Destructuring
- Default, Rest, Spread
- let, const
- for..of
- Promises
- Module System
- Map, Set*
- Generator*
- Symbol*

*본 세미나에서 다루지 않음

ECMAScript 2016

- ECMAScript 7번째 에디션
- 2016년 6월에 공표
- 조금의 기능이 추가되었다
- 대부분의 Runtime에서 **아직 완벽하지 않은 지원**



http://www.ecma-international.org/ecma-262/7.0/

ECMAScript 2017

- ECMAScript 8번째 에디션
- 아직 공식 발표는 안됨
  - 하지만 오늘 일부 다뤄볼 예정
- 조금의 기능이 추가될 예정

https://tc39.github.io/ecma262/

ES6+ 어떻게 써볼 수 있나요

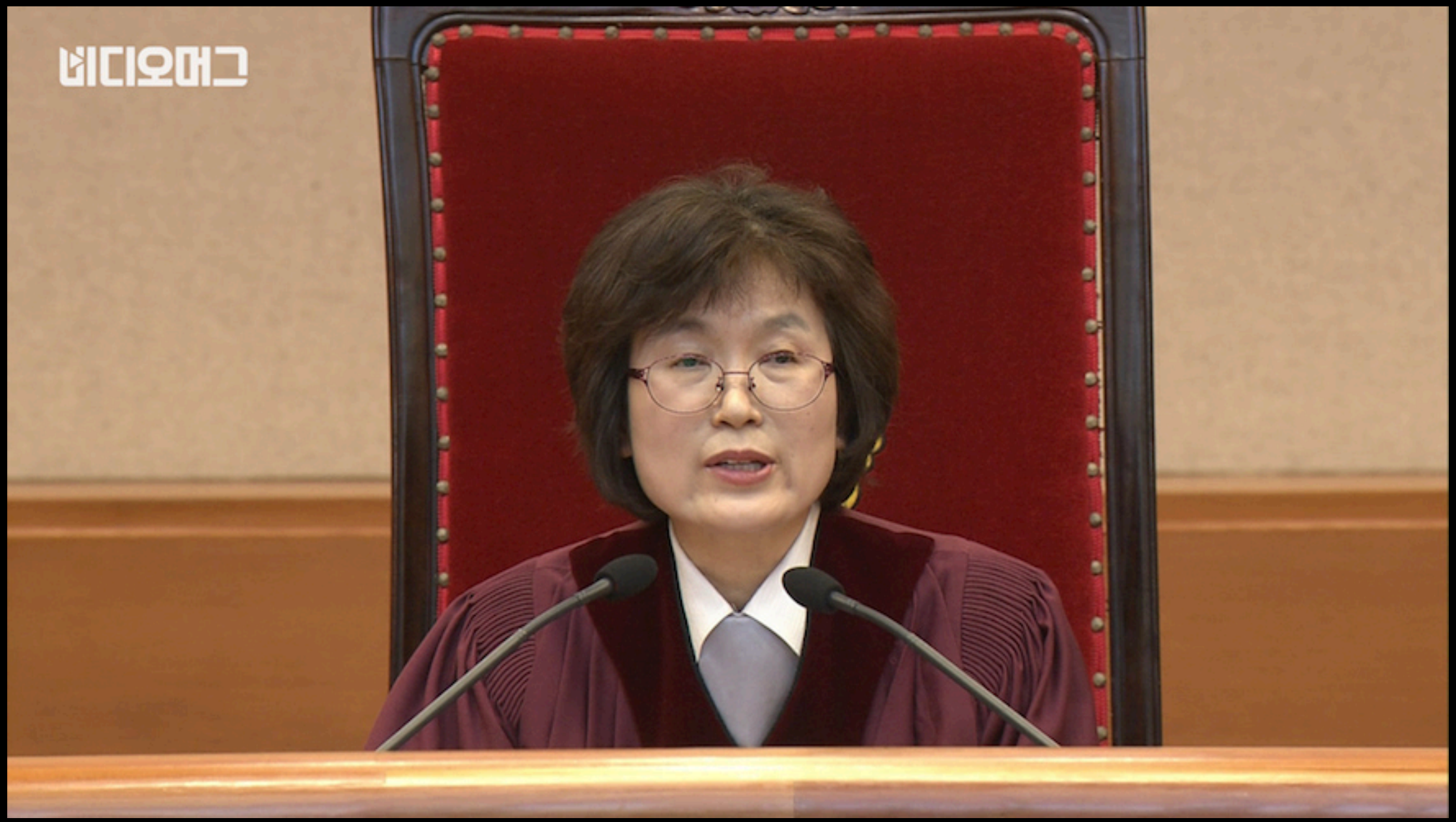
BABEL

https://babeljs.io/

Babel is a JavaScript compiler.

Use next generation JavaScript, today.

*Transpile의 특성상 ES6+의 모든 기능을 지원하지는 않는다



그러나

오늘은 간편하게 크롬을 쓰세요

자바스크립트 콘솔
: ⌘⌥J
: Ctrl + Shift + J

이 세미나는 많은 부분을

MDN - JavaScript 안내서

https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide

에서 가져왔습니다

Hello World

```
function hello() {
  // This is comment
  console.log(“Hello World!”)
}

hello()
```

변수

var i = 0

var nickname = “rodumani”

var name

name = “Changje Jeong”

var 의 Scope?

```
var j = 0
for (var i=0;i<10;i++) {
    var j = i
}
```

다음 중 j 의 값은?

1. 0
2. 9

var 의 Scope?

```
var j = 0
for (var i=0;i<10;i++) {
    var j = i
}
```

다음 중 j 의 값은?

1. ~~0~~
2. 9

Variables declared within a JavaScript function,
become LOCAL to the function.

~~Javscript가 나쁜언어라고 평가받는 이유~~

```
if (true) {
  var x = 5
}
console.log(x)
```

1. Error가 발생한다
2. 5가 출력된다
3. undefined가 출력된다

```
if (true) {
  var x = 5
}
console.log(x)
```

1. Error가 발생한다
2. 5가 출력된다
3. undefined가 출력된다

```
(function() {
  console.log(myvar)
  var myvar = "local value"
})()
```

1. Error가 발생한다
2. local value가 출력된다
3. undefined가 출력된다

```
(function() {
  console.log(myvar)
  var myvar = "local value"
})()
```

1. Error가 발생한다
2. local value가 출력된다
3. undefined가 출력된다

```
(function() {
  console.log(myvar)
  var myvar = "local value"
})()
```

```
(function() {
  console.log(myvar)
  var myvar = "local value"
})()
```

```
(function() {
  var myvar
  console.log(myvar)
  myvar = "local value"
})()
```

```
(function() {
  console.log(myvar)
  var myvar = "local value"
})()
```

Hoisting

```
(function() {
  var myvar
  console.log(myvar)
  myvar = "local value"
})()
```

let, const : Block-Scoped Variable

```
function f() {
  {
    let x
    {
      // okay, block scoped name
      const x = "sneaky"
      // error, const
      x = "foo"
    }
    // error, already declared in block
    let x = "inner"
  }
}
```

ES6 에서 추가됨

```
if (true) {
  let x = 5
}
console.log(x)
```

1. Error가 발생한다
2. 5가 출력된다
3. undefined가 출력된다

```
if (true) {
  let x = 5
}
console.log(x)
```

1. Error가 발생한다
2. 5가 출력된다
3. undefined가 출력된다

ReferenceError: y is not defined

```
let fruit = {
    name: “Apple”,
    count: 5
}
```

JS의 {}와 Python의 {}는 같을까?

- Python에서 {} 리터럴은 Dictionary
- JavaScript에서 {} 리터럴은 Object
- Dictionary와 같은 기능을 하는 것은 Map
- Object의 키는 Strings과 Symbols
  Map의 키는 any value
- Map 객체는 삽입 순서대로 element 들을 순회한다

Map: ES6 에서 추가됨

Object property shorthand

```
const x = 10;
const y = 15;
const p1 = { x: x, y: y }
```

Object property shorthand

```
const x = 10;
const y = 15;
const p1 = { x: x, y: y }
const p2 = { x, y }
```

ES6 에서 추가됨

Trailing comma

```
const obj = {
  x: 10,
  y: 50,
}

const arr = [ 1, 2, 3, ]
```

ES6 에서 추가됨

조건문

```
if (condition) {
  statement_1
} else {
  statement_2
}
```

```
switch (expression) {
  case label_1:
    statements_1
    [break]
    ...
  default:
    statements_def
    [break]
}
```

거짓인 값

- false
- undefined
- null
- 0
- NaN
- the empty string (“")
- 기타 모두 참

예외처리

```
try {
   throw "myException"
}
catch (e) {
    logMyErrors(e)
}
```

반복문

- for
- for...in
- for…of
- do...while
- while
- 레이블
- break
- continue

반복문

```
const arr = [5,4,3,2,1]

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i])
}
```

반복문

```
const arr = [4,3,2,1,0]

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i])
}
```

```
5
4
3
2
1
```

반복문

```
const arr = [4,3,2,1,0]

for (let i in arr) {
  console.log(i)
}
```

```
4
3
2
1
0
```

```
0
1
2
3
4
```

반복문

```
const arr = [4,3,2,1,0];

for (let i in arr) {
  console.log(i)
}
```

4	0
3	1
2	2
1	3
0	4

for…in 은 Index를 순회한다

반복문

```
const arr = [4,3,2,1,0]

for (let o of arr) {
  console.log(o)
}
```

4
3
2
1
0

0
1
2
3
4

for…in 은 Index를 순회하고 for…of 는 element를 순회한다

for…of: ES6 에서 추가됨

반복문

```
let arr = [3, 5, 7]
arr.foo = "hello"

for (let i in arr) {
    console.log(i)
}

for (let i of arr) {
    console.log(i)
}
```

반복문

```
let arr = [3, 5, 7]
arr.foo = "hello"

for (let i in arr) {
   console.log(i)
}
```

"0", "1", "2", "foo"

```
for (let i of arr) {
   console.log(i)
}
```

"3", "5", "7"

함수

```
function square(number) {
  return number * number
}

const n = square(3) // 9
```

함수

```
function add(n1, n2) {
  return n1 * n2
}

const n = add(3, 5) // 8
```

함수

fizzbuzz

- 1에서 부터 n까지의 숫자 중
- 3의 배수는 fizz
- 5의 배수는 buzz
- 15의 배수는 fizzbuzz 출력하기

함수

```
function fizzBuzz(n) {
  if (n % 15 === 0)
    return “fizzbuzz”
  else if (n % 3 === 0)
    return “fizz”
  else if (n % 5 === 0)
    return “buzz”
  return n
}
```

서로 다른 타입도 리턴할 수 있다

JavaScript의 함수는

First-class object이다

함수

```
const square = function (number) {
  return number * number
}
let x = square(4) // x 의 값은 16
```

함수

```
const square = function (number) {
  return number * number
} // 익명함수
```

함수

```
function benchmark(f) {
    const start = new Date()
    f()
    const end = new Date()
    console.log(end.getTime() - start.getTime())
}

benchmark(function() {…})
```

Arrow Function

```
const square = function (number) {
  return number * number
};
```

```
const square = (number) => {
  return number * number
};
```

ES6 에서 추가됨

함수 - default parameter

```
function square(number = 5) {
  return number * number
}
```

```
square() // 25
```

ES6 에서 추가됨

함수 - rest parameter

```
function multiply(multiplier, ...theArgs) {
  return theArgs.map(function(x) {
    return multiplier * x
  })
}
const arr = multiply(2, 1, 2, 3);
console.log(arr); // [2, 4, 6]
```

ES6 에서 추가됨

Destructuring

Array Matching

```
let [ a, , b ] = [1, 2, 3]
// a === 1 b === 3
```

ES6 에서 추가됨

Destructuring

Object Matching

```
let [ a, , b ] = [1, 2, 3]
// a === 1 b === 3

let { name } = { name: “Changje Jeong” }
// name === “Changje Jeong”
```

ES6 에서 추가됨

Destructuring

Fail-soft Matching

```
let [ a, b , c = 3] = [1, 2]
// a === 1 c === 3
```

```
let { a, b = -1 } = { a : 5 }
// a === 5 b === -1
```

ES6 에서 추가됨

Destructuring

Parameter matching

```
const instr = {
  firstName: “Changje”,
  lastName: “Jeong”,
  major: “Computer Science”
};

function fullName({ firstName, lastName }) {
  return firstName + “ “ + lastName
}
```

ES6 에서 추가됨

Spread Operator

```
let params = [ "hello", true, 7 ]
const other = [ 1, 2, ...params ]
```

ES6 에서 추가됨

Spread Operator

```
let params = [ "hello", true, 7 ]
const other = [ 1, 2, ...params ]
// [ 1, 2, "hello", true, 7 ]
```

ES6 에서 추가됨

Spread Operator

```
let params = [ "hello", true, 7 ]
const other = [ 1, 2, ...params ]
// [ 1, 2, "hello", true, 7 ]
```

```
f(1, 2, …params)
```

ES6 에서 추가됨

Spread Operator

```
let params = [ "hello", true, 7 ]
const other = [ 1, 2, ...params ]
// [ 1, 2, "hello", true, 7 ]
```

```
f(1, 2, …params)
f(1, 2, “hello”, true, 7)
```

ES6 에서 추가됨

Template String

```
function fullName(firstName, lastName) {
  return firstName + " " + lastName
}
```

```
function fullNameNew(firstName, lastName) {
  return `${firstName} ${lastName}`
}
```

ES6 에서 추가됨

Array

```
let fruits = [“사과”, “배”, “오렌지”]
fruits.map(fruit => `맛있는 ${fruit}`)
[“맛있는 사과”, “맛있는 배”, “맛있는 오렌지”]
```

```
fruits.filter(fruit => fruit.length > 1)
[“사과”, “오렌지”]

fruits.sort()
[“배”, “사과”, “오렌지”]
-> fruits 스스로를 정렬하고 스스로를 리턴
```

Array

```
// Array에 포함되어 있는지 검사
let fruits = [“사과”, “배”, “오렌지”]
let i = fruits.indexOf(“사과”)
if (i > -1) {
  …
}

if (fruits.includes(“사과”)) {
  …
}
```

includes: ES7 에서 추가됨

class

class

없음

대신 prototype이라는게 있다

class

없음 ^없

ES6 부터 있음

class

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  move(x, y) {
    this.x = x;
    this.y = y;
  }
}
```

class

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  move(x, y) {
    this.x = x;
    this.y = y;
  }
}
```

```
> let p = new Point(3, 5);
<· undefined
> p
<· ▶ Point {x: 3, y: 5}
> p.move(10, 4);
<· undefined
> p
<· ▶ Point {x: 10, y: 4}
```

class - inheritance

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        this.width  = width
        this.height = height
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
}
```

callback

callback

API 요청 - Blocking

```
let resp = getResponse(‘http://naver.com')
console.log(resp)
```

*Pseudo Code 입니다

callback

API 요청 - Blocking

```
let resp = getResponse('http://naver.com')
console.log(resp)
```

1. 서버에 요청 보냄
2. 기다림……
3. 응답이 옴
4. reps에 대입

기다리는 동안 다른코드를 실행하지 않는다

*Pseudo Code 입니다

callback

API 요청 - Non-Blocking

```
getResponse('http://naver.com', function (resp) {
 console.log(resp)
})
```

*Pseudo Code 입니다

callback

API 요청 - Non-Blocking

```
getResponse('http://naver.com', function (resp) {
 console.log(resp)
})
```

callback function

*Pseudo Code 입니다

callback hell

```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

Promise

```
fetch(‘http://www.naver.com')
```

Promise

Promise 객체는 비동기 계산을 위해 사용됩니다.
Promise는 아직은 아니지만 나중에 완료될 것으로 기대되는 연산을 표현합니다.
- MDN

ES6 에서 추가됨

Promise

Promise 객체는 비동기 계산을 위해 사용됩니다.
Promise는 아직은 아니지만 나중에 완료될 것으로 기대되는 연산을 표현합니다.
- MDN

내가 이거 끝나면 너한테 알려줄께 약속

ES6 에서 추가됨

Promise

```
fetch('http://www.naver.com')
.then((resp) => {
    console.log('success', resp.status)
})
.catch((err) => {
    console.log('failed', err)
})
```

1. www.naver.com에 접속한 상태에서 실행해보고

2. www.google.com 에 접속한 상태에서 실행해보기

Promise

1. www.naver.com에 접속한 상태에서 실행

```
> fetch('http://www.naver.com')
  .then(resp => console.log('success', resp.status))
  .catch(err => console.log('failed', err))
<· ▶ Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}
  success 200
```

Promise

2. www.google.com에 접속한 상태에서 실행

```
> fetch('http://www.naver.com')
  .then((resp) => {
      console.log('success', resp)
  })
  .catch((err) => {
      console.log('failed', err)
  })
```

```
▶ Mixed Content: The page at 'https://www.google.co.kr/?gfe_rd=cr&ei=vkbJWOC5AYH-8wedlZGoDQ&gws_rd=ssl' was loaded over HTTPS, but requested an insecure resource 'http://www.naver.com/'. This request has been blocked; the content must be served over HTTPS.   VM99:1

▶ Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}

failed TypeError: Failed to fetch   VM99:6
```

Async/Await

```
function naver() {
  fetch('http://www.naver.com')
  .then((resp) => console.log('success', resp.status))
  .catch((err) => console.log('failed', err))
}
```

```
async function naver() {
  try {
    const resp = await fetch('http://www.naver.com')
    console.log('success', resp.status)
  } catch (err) {
    console.log('failed', err)
  }
}
```

ES7 에서 추가됨

```
//------ lib.js ------
var sqrt = Math.sqrt;
function square(x) {
    return x * x;
}
function diag(x, y) {
    return sqrt(square(x) + square(y));
}
module.exports = {
    sqrt: sqrt,
    square: square,
    diag: diag,
};

//------ main.js ------
var square = require('lib').square;
var diag = require('lib').diag;
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

```
//------ lib.js ------
var sqrt = Math.sqrt;
function square(x) {
    return x * x;
}
function diag(x, y) {
    return sqrt(square(x) + square(y));
}
module.exports = {
    sqrt: sqrt,
    square: square,
    diag: diag,
};

//------ main.js ------
var square = require('lib').square;
var diag = require('lib').diag;
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

Common JS

```
//------ lib.js ------
var sqrt = Math.sqrt;
function square(x) {
    return x * x;
}
function diag(x, y) {
    return sqrt(square(x) + square(y));
}
module.exports = {
    sqrt: sqrt,
    square: square,
    diag: diag,
};

//------ main.js ------
var square = require('lib').square;
var diag = require('lib').diag;
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

Common JS

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export default function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------
import  from 'lib';
import React, { Component, PropTypes } from
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

ES6

마무리

- 변수
- Object
- 조건문
- 반복문
- 함수
- Array
- Callback

- Arrow Function
- Class
- Template String
- Destructuring
- Default, Rest, Spread
- let, const
- Map
- for..of
- Promises
- Async/Await
- Module System