

# ECMA Script

김 순곤

[soongon@hucloud.co.kr](mailto:soongon@hucloud.co.kr)

# 자바스크립트 과정 내용

---

- 자바스크립트 기본
- Node.js
- ECMAScript 6 (ES 2015)
- 자바스크립트 함수형 프로그래밍 기법
- 브라우저 자바스크립트
  - 웹표준 - HTML5 / CSS3 / JavaScript5.1
  - jQuery
- 프론트엔드 개발 생태계
  - npm
  - webpack
  - babel
- 최신 자바스크립트 개발 기법
  - 프론트엔드 컴포넌트 기반 프레임워크 - React.js
  - Angular vs. React.js vs. Vue.js

Javascript

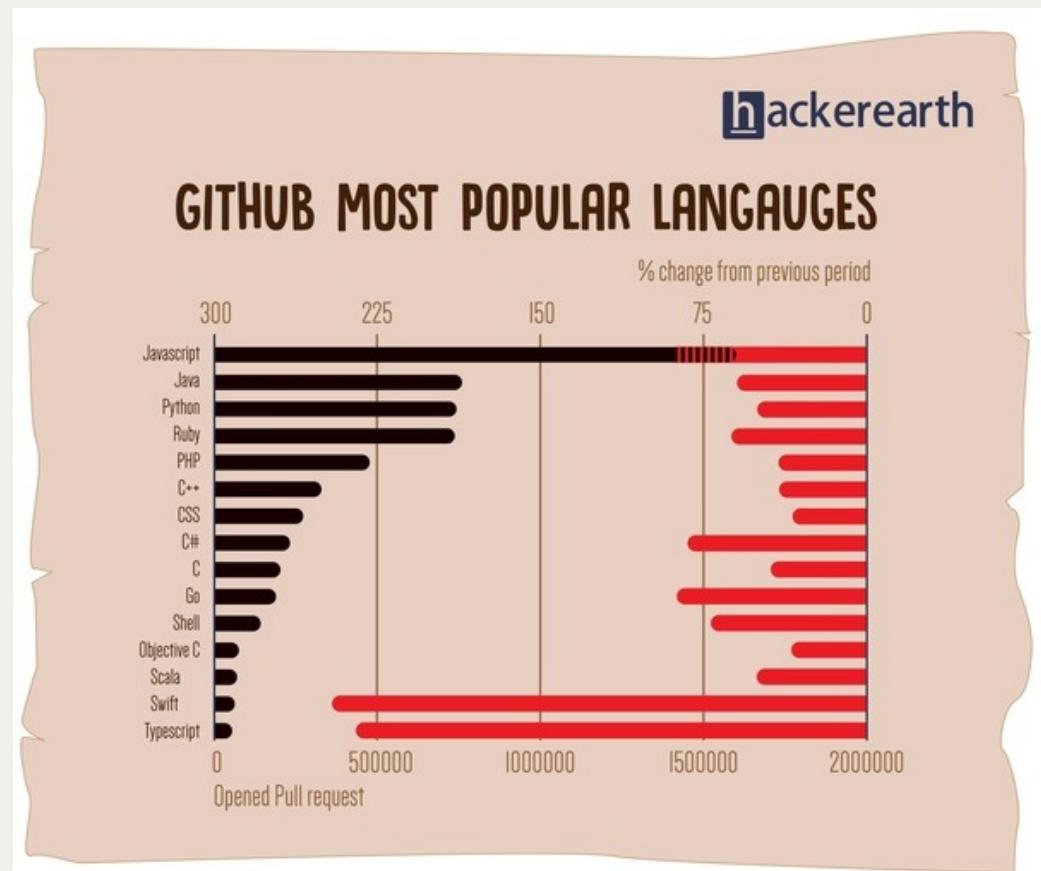
Node.js

# 자바스크립트 소개

# 자바스크립트 소개

- ☑ 웹 브라우저에서 동작하는 언어로 시작
  - 브라우저 DOM 핸들링이 주요 역할
  - 차후 prototype.js → jQuery로 라이브러리 등장

최근에는 가장 핫한 언어로 도약



# **현재 자바스크립트라고 하면 ECMA SCRIPT 5.1 버전을 의미**

조만간 ECMAScript 2015 혹은 ECMAScript 6 라고 불리는 버전의 시대가 올것임

**자바스크립트는 프론트엔드 개발의  
표준이자 유일한 개발 언어!**

# 자바스크립트 활용 범위

---

## 웹 개발

- 브라우저의 자바스크립트 성능향상
- 프론트엔드 개발은 가장 활발한 개발 영역 중 하나

## 서버 개발

- 2009년 Node.js 등장
- 브라우저의 자바스크립트를 OS 레벨로 포팅
- 백엔드 개발이 가능해지고 성능도 뛰어남

## 애플리케이션 개발

- 크롬 OS
- 데스크탑 애플리케이션 : Electron
- 모바일 애플리케이션 : 하이브리드 앱 (Cordova, phonegap, Ionic..)

# 자바스크립트 러닝 로드맵

1. 자바스크립트 언어 기본 (Node.js)
2. 브라우저와 자바스크립트
3. ES6 - ECMA Script 2015+
4. React 또는 Angular

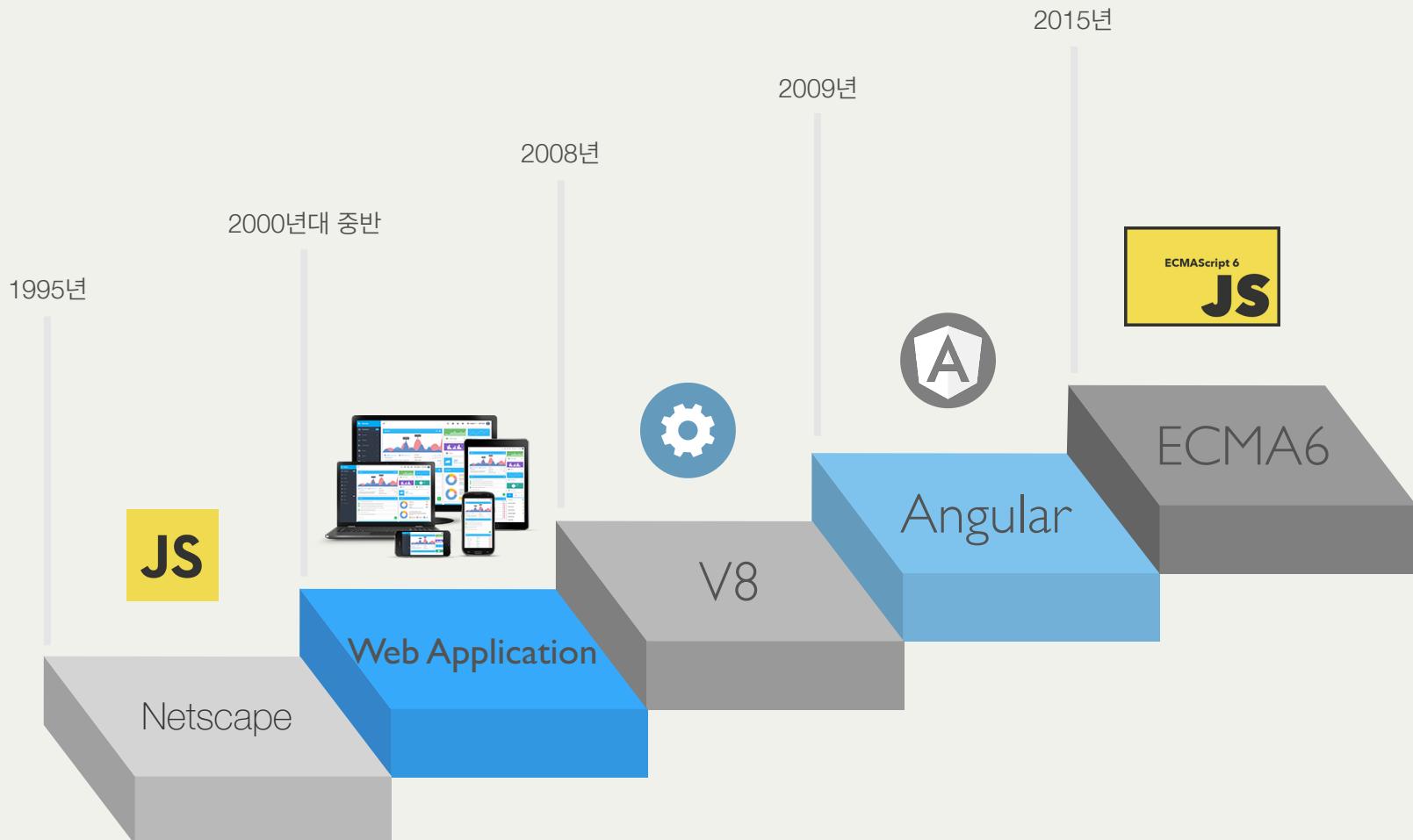
먼저, 1번 자바스크립트 언어에 대해 다룹니다.

# 목 차

---

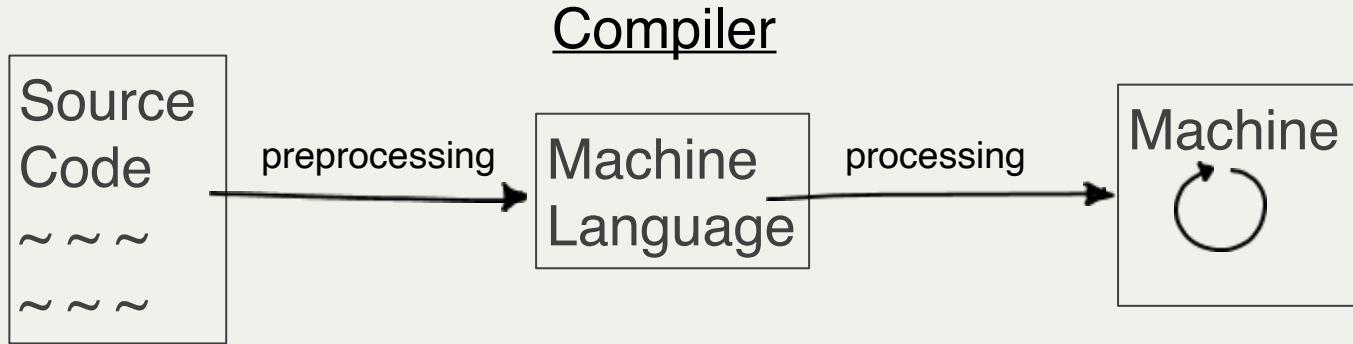
1. 자바스크립트 소개와 개발환경 설정
2. 변수 기본 컨셉
3. 타입 6가지
4. 기본 타입
5. 함수
6. 객체
7. 배열

# 자바스크립트의 발전

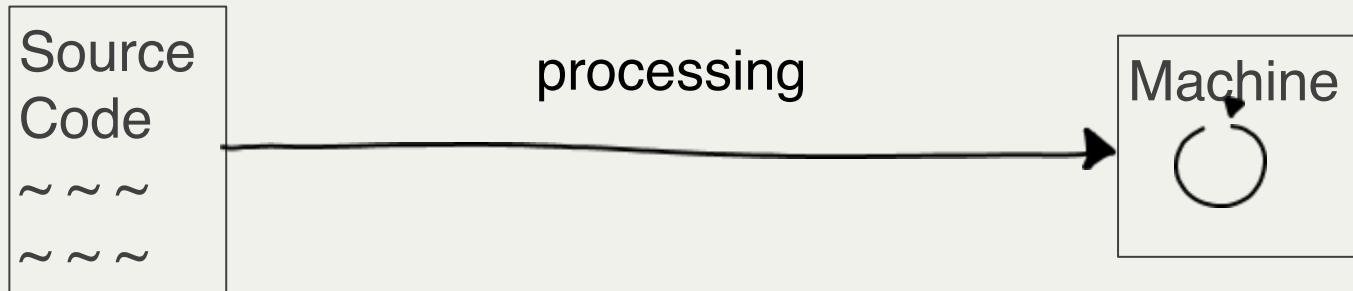


# JavaScript

- ✓ Javascript는 Interpreter 언어.
  - 크롬 개발자 툴(~~혹은 파이어폭스~~)의 JavaScript Console 을 통해 사용



Interpreter/Runtime



# JavaScript

## ✓ Javascript 를 사용

- Web Browser 의 Developer Tool 사용



Firefox



chrome



Opera



Safari



Internet Explore



Google

Google 검색 I'm Feeling Lucky



MS Edge

The screenshot shows the developer tools of a web browser (likely Chrome) with the DOM tree expanded. The tree starts with the `<html>` tag, which contains the `<head>` and `<body>` sections. The `<body>` section includes the Google logo, the search bar, and the "Google 검색" button. Numerous CSS rules are listed on the right side, applying styles to various elements like `div`, `span`, and `script` tags. Some specific rules include `background-color: #fff`, `color: #000`, and `font-family: sans-serif`.

# 개발환경 구성

---

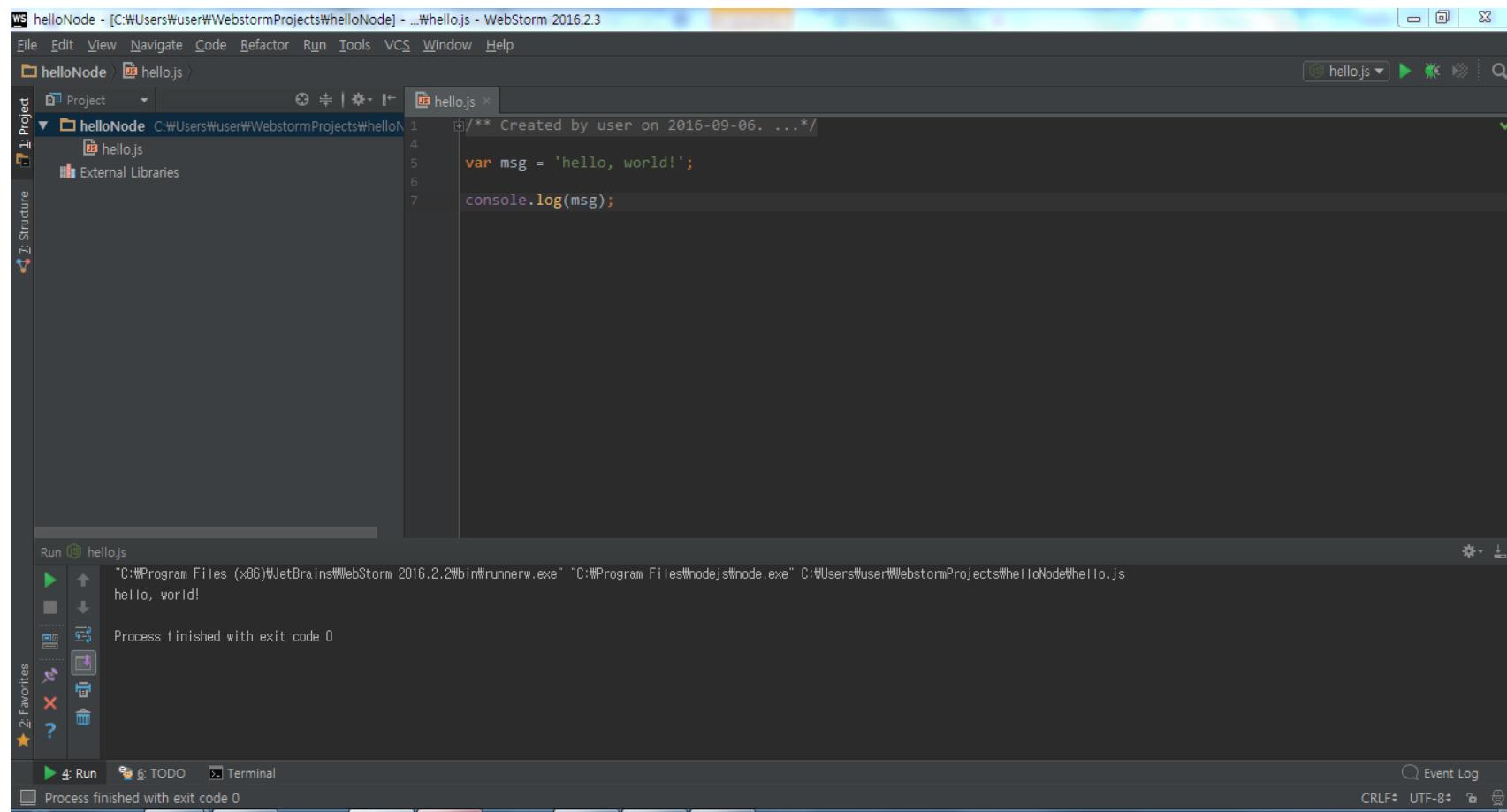
## 로컬에 구성

- JetBrains 의 WebStorm
- VS Code (Microsoft)

## 웹 에디터 (일명 클라우드 에디터)

- JSBin
- CodePen
- Plunker

# 웹스톰 설치 - <http://jetbrains.com>



# VS Code - <https://code.visualstudio.com/>

The screenshot shows the official Visual Studio Code website. At the top, there's a dark header with the "Visual Studio Code" logo, navigation links for "Docs", "Updates", "Blog", "Extensions", and "FAQ", a search bar labeled "Search Docs", and a "Download" button. A banner at the top of the main content area says "Version 1.5 is now available! Read about the new features and fixes in August." Below this, the main heading "Code editing. Redefined." is displayed in large white text on a dark blue background. To its right, a subtitle "Free. Open source. Runs everywhere." is shown. A prominent green "Download for Windows" button is located below the heading. Further down, a section for "Available on other platforms and insiders build" is visible, along with a note about accepting the license and privacy statement. On the right side of the page, a screenshot of the VS Code interface is shown, displaying a file named "www.ts" with some TypeScript code. The interface includes a sidebar with extensions like C#, Python, and Debugger for Chrome, and a status bar at the bottom.



IntelliSense



Debugging



Built-in Git



Extensions

JavaScirpt

- 0. 변수
- 1. Function
- 2. Callback
- 3. 객체

# 자바스크립트 기본

# JavaScript

---

- Browser에서 동작하는 “클라이언트 사이드(Client Side)” 언어
  - HTML Object(DOM)를 이벤트 기반으로 동작시키기 위한 언어
  - 버튼을 클릭했을 때.. 텍스트를 클릭 했을 때.. 화면이 보여졌을 때.. 등등
  - 4~5년 전만 해도 “느린 언어”라고 평가되기도 했던 언어.
  - 최근, Google에서 만든 V8 Javascript 엔진을 통해 속도가 많이 개선됨
  - (V8 : Google Chrome 에 탑재된 JavaScript 실행 엔진)
- Node.js를 사용하기 위해 필요한 필수 언어

Node.js

V8 JavaScript Runtime

# ES6

---

- ECMA Script 2015 또는 ECMA Script 6
  - 줄여서 ES2015 또는 ES6 라고 함
- ECMA International의 ECMA-262에 근거한 표준 스크립트 언어
- ECMA Script 2015 혹은 ES6 라고 불린다.
- 최초 ECMA Script는 브라우저 언어인 Javascript와 Jscript간 차이를 줄이기 위한 공통 스펙 제안으로 출발 (1992, ECMA-262)

# 0. 변수

- 값(Data)을 저장하기 위한 용도

- 자바스크립트는 loose type language
- 즉, 변수의 타입을 규정하지 않음
- var 키워드로 변수를 선언

```
<html>
<head>
<script type="text/javascript">
    var number = 1; // number 변수를 정의하고 숫자 1을 정의함.
    console.log(number); // number를 콘솔에 출력함.
    number = 3.14; // number 변수에 실수 3.14를 재정의함.
    console.log(number); // number를 콘솔에 출력함.
    number = true; // number 변수에 불린 true를 재정의함.
    console.log(number); // number를 콘솔에 출력함.
    number = "글자입니다. "; // number 변수에 문자를 재정의함.
    console.log(number); // number를 콘솔에 출력함.
</script>
</head>
<body></body>
```

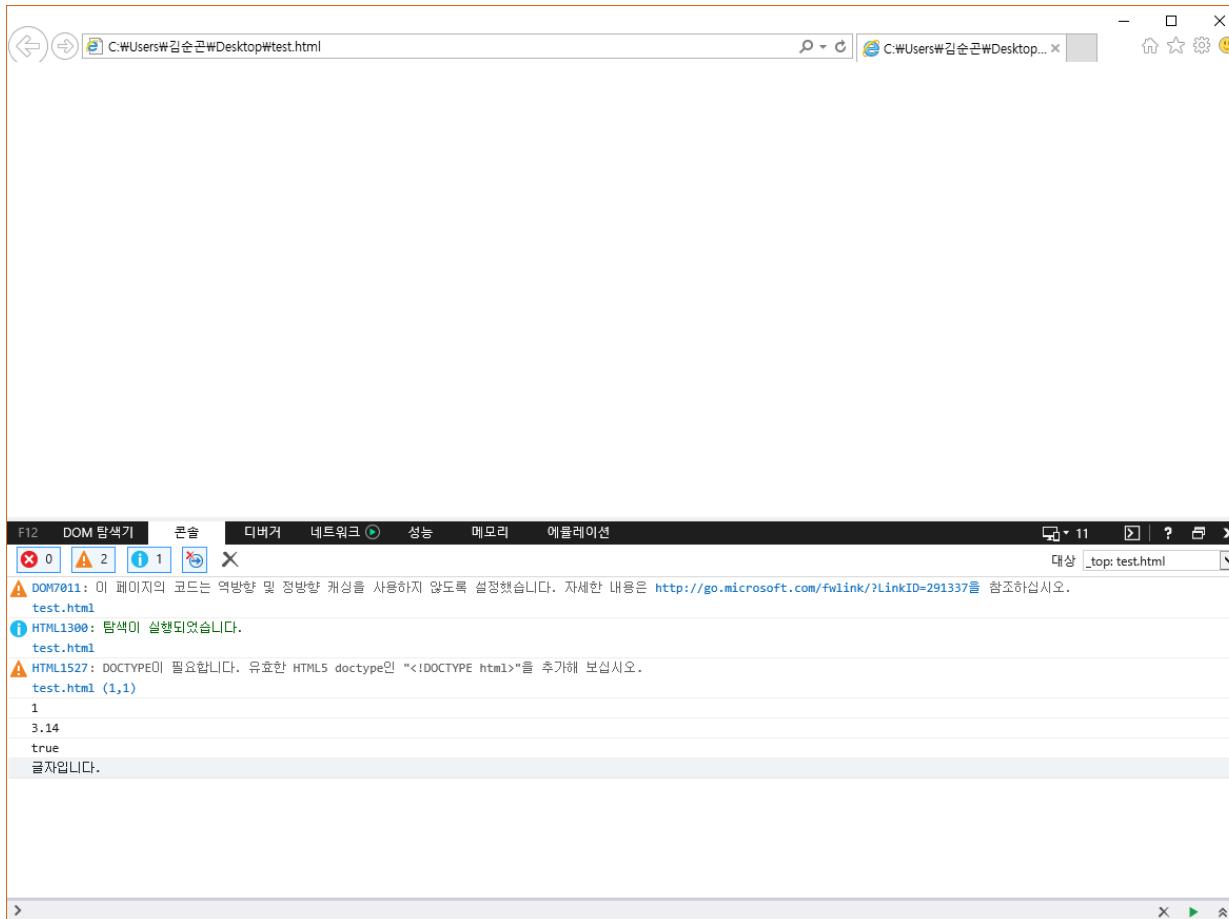
# 0. 변수의 종류

---

- Boolean
  - var isRight = true;
- Number
  - var a = 3; var b = 3.5;
- String
  - var str = 'hello world'; var str2 = "hello, world";
- Array
  - var myArr = []; var myArr2 = [1, 2, 3, 4];
- Function
  - var myFunc = function () { ⋯ };
- Object
  - var myObj = {};

# 0. 변수

- 저장된 파일 실행해 확인 해보기
  - Ctrl + Shift + I 눌러 요소 검사 창을 실행함(크롬)



# ES6 문법

- 변수 선언 키워드 추가
  - let :
    - var : 함수스코프
    - let : 블럭 {} 스코프, 변수의 재정의를 허용하지 않음
  - const
    - 상수를 표현 - Immutable 데이터
    - 변수의 컨텐츠는 변경이 가능하므로 객체를 선언할 때 사용
- 새로운 스트링 리터럴 문법
  - 백틱 (`) : 새로운 포매팅 문법을 사용 가능

```
let someVar = 15;
const str = `My age is ${someVar} years old.`;
```

# 1. Function

- 함수 - First Class Citizen
  - 일련의 작업을 하나의 묶음으로 정의함
  - `function function_name() { … }` 으로 정의함
  - 함수를 변수에 할당, 파라미터를 전달하고, 리턴 값을 반환할 수 있음

```
<html>
<head>
  <script type="text/javascript">
    function hello() {

      var message = "반갑습니다.";
      console.log(message);
    }
    hello();

  </script>
</head>
<body></body>
```

## 2. Callback

- 처리 후 작업
  - 함수에서 일련의 작업이 완료되었을 때, 추가로 실행해야 하는 함수
  - 함수가 종료되는 시점이 불분명할 때, Callback을 사용함

```
<html>
<head>
    <script type="text/javascript">
        function hello( message, callback ) {
            message = message + ". 자바 스크립트 입니다.";
            callback();
            return message;
        }

        var callbackFunction = function() {
            console.log( "콜백 함수가 실행되었습니다." );
        };

        var helloMessage = hello( "반갑습니다", callbackFunction );
        console.log( helloMessage );
    </script>
</head>
<body></body>
</html>
```

## 2. Callback - 계속

- 콜백 함수에 인자 전달 후 반환하기

```
<html>
<head>
    <script type="text/javascript">
        function hello( message, callback ) {
            message = message + ". 자바 스크립트 입니다.";
            message = callback( message );
            return message;
        }

        var callbackFunction = function( message ) {
            message = message + " 콜백에서 수정함.";
            return message;
        };

        var helloMessage = hello( "반갑습니다", callbackFunction );
        console.log( helloMessage );
    </script>
</head>
<body></body>
</html>
```

# ES6 문법

- **파라미터 기본 값 지원**

- 파라미터를 모두다 전달하지 않아도 함수는 잘 실행됩니다.
- 파라미터를 받을 때, 기본값을 정의할 수 있게 되었습니다.
- Validation 검증 로직이 사라지게 됩니다!

```
function foo ( x = 0, y = 0, z = 0 ) {  
    ...  
}  
  
foo(10, 20);
```

# ES6 문법

- 화살표 함수 ( => )
  - Arrow Function : 익명함수를 만드는 새로운 방법

- 기존 :

```
var foo = function ( x, y ) {  
    ...  
}  
  
foo(10, 20);
```

- 변경 :

```
var foo = ( x, y ) => {  
    ...  
}  
  
foo(10, 20);
```

### 3. 객체

---

- Object is King!
  - 자바스크립트에서 객체를 이해하면 모두 이해하는 것
  - 모든 것이 오브젝트
- 객체는 속성과 기능으로 구성됨
  - 속성, Properties : 변수로 구현
  - 기능, Methods : 함수로 구현

### 3. 객체 - 객체 생성하는 방법

---

- 객체 리터럴을 이용한 방법
  - var myObj = {};
- 함수를 이용한 방법
  - function myObj3 () { this.name; }
- JSON : (JavaScript Object Notation)
  - 인터넷 상의 클라이언트/서버 데이터 교환 포맷 사실상 표준
  - Key : Value pair

# ES6 문법

- class 키워드 추가

- 클래스 선언

```
class Student {  
    constructor (name) {  
        this.name = name;  
    }  
}
```

- 클래스 사용

```
let s1 = new Student ('kim');  
console.log(s1.name);
```



# 개발환경 설정

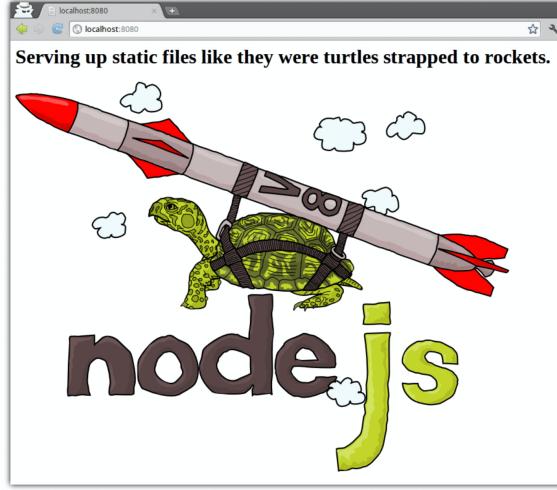
**http-server**  
0.12.0 • Public • Published a month ago

Readme Explore [BETA] 10 Dependencies 592 Dependents 37 Versions

build passing dependencies up to date npm v0.12.0 license MIT

**http-server: a command-line http server**

http-server is a simple, zero-configuration command-line http server. It is powerful enough for production usage, but it's simple and hackable enough to be used for testing, local development, and learning.



Install > npm i http-server

Weekly Downloads 151,756

Version 0.12.0 License MIT

Unpacked Size 19.9 kB Total Files 5

Issues 78 Pull Requests 39

Homepage [github.com/http-party/http-server#readme](https://github.com/http-party/http-server#readme)

Repository [github.com/http-party/http-server](https://github.com/http-party/http-server)

Last publish a month ago

Collaborators 

> Try on RunKit

## npm init -y

---

### npm

- Node package manager
- 프론트엔드 개발 시 프로젝트 관리자 역할 수행
- 필수 개발 툴
- 프로젝트에서 사용하는 여러 패키지를 관리
- 기타 다양한 기능들 제공

### 주요 명령어

- npm init : 프로젝트 초기화, package.json 파일 생성
- npm install [패키지명] : 패키지 설치
- npm update [패키지명] : 패키지 버전 업데이트

# http-server

---

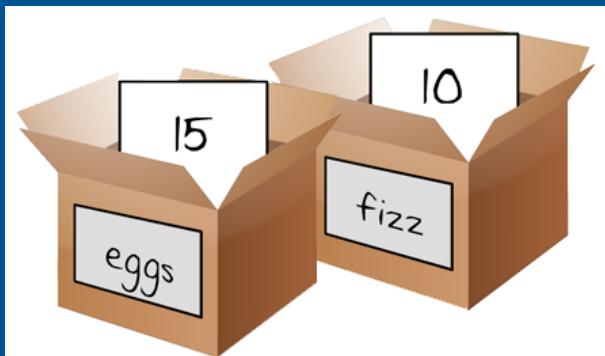
## http-server

- 개발용 웹서버
- npm 명령어를 통해 설치
  - npm install http-server

## http-server 실행

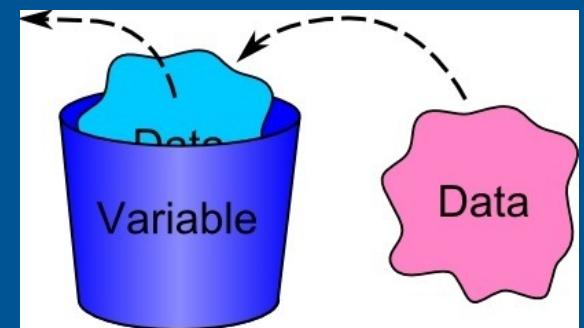
- npm http-server 또는 npm http-server [index.html 파일 위치]

<https://www.npmjs.com/package/http-server>



Basic Concepts of Variables

변수 기본 컨셉



# 변수 - 정의

---

- ✓ Value 를 보관하는 ***named container***
  - ✓ 그래서 변수는 반드시 이름이 있어야하고,
  - ✓ 그 이름 짓는 방법(naming conventions) 또한 매우 중요하다.
  - ✓ 이름을 다른 표현으로 ***identifier*** 라고도 한다.

```
var x;  
var y = 'Hello JS!';  
var z;
```

# 변수

---

```
var x;  
var y = "Hello JS!";  
var z;
```

빨간색 상자는 변수를 각 변수는 이름을 가지고 있다.



자바스크립트의 값은 모두 상자속에 들어있다.

# 변수

```
var x;  
var y = "Hello JS!";  
var z;  
z = false;  
z = 101;
```



x

undefined

y

"Hello JS!"

z

101



변수 생성 후 값을 재할당 하는 것이 가능하다.

## 변수 - 생성 후 재정의 해서 사용 가능

---

- ✓ 타입에 상관없이 재정의 가능

```
var x;  
var y = 'Hello JS!';  
var z;  
  
z = false;  
z = 101;
```

# 예약어, Reserved Words

---

- ☒ 예약어는 변수명으로 사용할 수 없다.

```
null true false break do instanceof typeof  
case else new var catch finally return void  
continue for switch while debugger function  
    this with default if throw delete in try  
class enum extends super const export import  
  
implements let private public yield  
interface package protected static
```

# 주석

---

- 코드들에 대한 설명
- 복잡한 코드에 대한 풀이를 작성하거나
- 코드에 작성된 참고자료들의 출처를 작성할 때 사용된다.

## Single line Command

```
// Double Slash로 주석을 작성할 수 있다.  
var count = 5; // 코드의 바로 옆에 작성할 수도 있다.
```

## Multi line Command

```
/*  
 * 여러 줄의 주석을 작성할 때는 멀티라인 주석을 사용한다.  
 * 주석...  
 */  
var count = 5;
```

Values & Types

# 변수의 타입과 값

# value & type

---

정의

**value** 자바스크립트에서 다루게  
되는 기본적인 값 또는 데이터

value

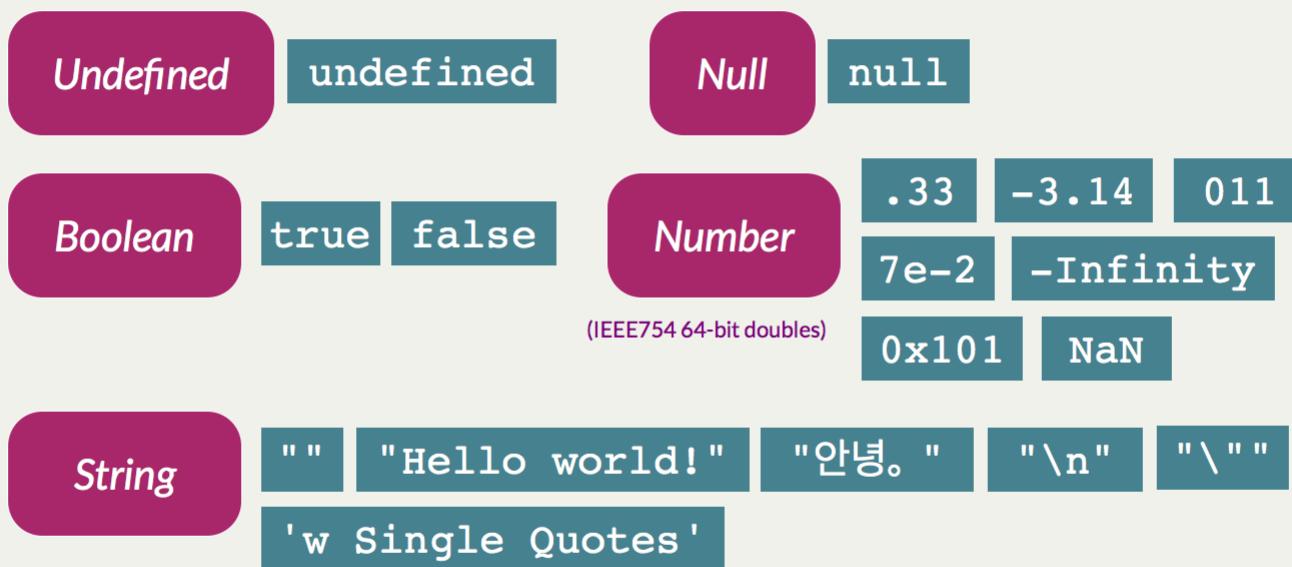
**type**은 데이터 값의 집합  
모두 6가지 데이터 타입이 있음

*Type*

:: { v1 , v2 , v3 , ... }

# Primitive Type, 기본 타입

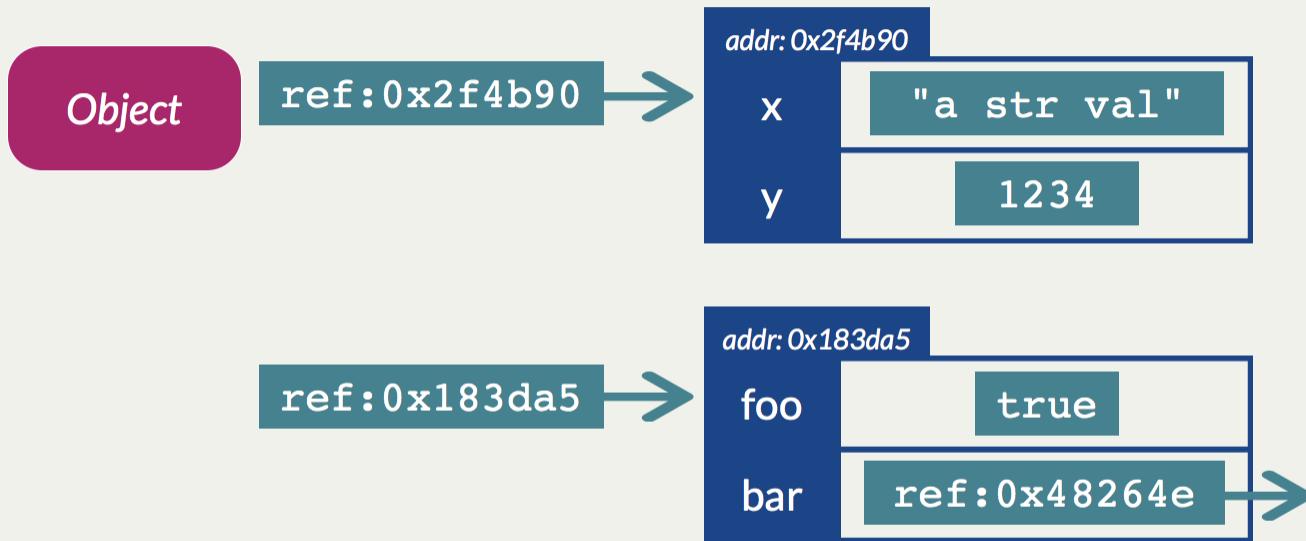
5 가지 기본 (non-Object) 타입이 있다.



여기의 모든 값을 primitive 값이라고 함

# Object type, 객체 타입

그리고 "Object" 타입이 있다.



오브젝트 타입의 값은 실제 오브젝트(객체)를 가리키는 주소값, **reference** 이다  
주로 주소값을 객체라고 한다.

# Property, 속성

---

정의

**object: property** 의 컬렉션

**property: value** 를 가지고 있는 이름

# Property, 속성

---

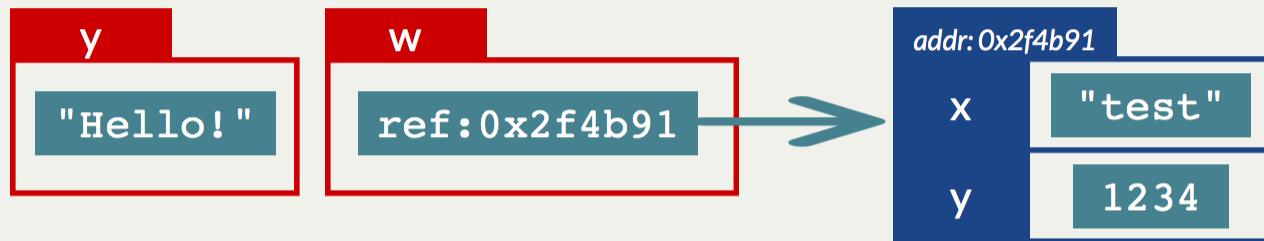
정의

*property* 의 이름 **key**;  
따라서, *object* 는 **key-value pair** 의 컬렉션

다른 프로그래밍 언어의 비슷한 컨셉으로는,  
e.g., *Map*, *Dictionary*, *Associative Array*, *Symbol Table*, *Hash Table*, ...

# 변수 vs. 속성

## object에서 “variable” vs. “property”



```
// 일반 변수와 자바스크립트 객체
var y = "Hello!";
var w = {
    x: "test",
    y: 1234
};
```

```
// 값 가져오기
y;           // "Hello!"
w;           // (the object ref)
w.x;         // "test"
w['x'];      // "test"
w.y;         // 1234
w['y'];      // 1234
```

# 객체 리터럴

중괄호는 자바스크립트 객체를 의미한다.

```
var w = {  
    x: "test",  
    y: 1234,  
    z: {},  
    w: {},  
    "" : "hi"  
};
```

```
var w = new Object();  
w.x = "test";  
w.y = 1234;  
w.z = new Object();  
w.w = new Object();  
w[ "" ] = "hi";
```

왼쪽 코드와 오른쪽 코드의 결과는 동일하다.

# 객체에서 속성 다루기

## Add/Get/Set/Remove a Property

객체 생성 후 동적으로 추가/삭제 등이 가능

```
var obj = {  
    1 : "Hello",  
    "3": "Good",  
    x : "JavaScript",  
    foo: 101,  
    bar: true,  
    "" : null  
};  
  
obj[ "2" ] = "World";      // *1 Add & Set  
obj[ "1" ];                // *2 Get          -> "Hello"  
obj[ 2 ];                  // *3 Get          -> "World"  
obj[ 3 ];                  // *4 Get          -> "Good"  
obj.foo = 202;              // *5 Set  
delete obj.bar;            // *6 Remove  
delete obj[ "" ];          // *7 Remove
```

# Object 타입

모든 Object 타입의 실제 값은 참조값(주소)이다.

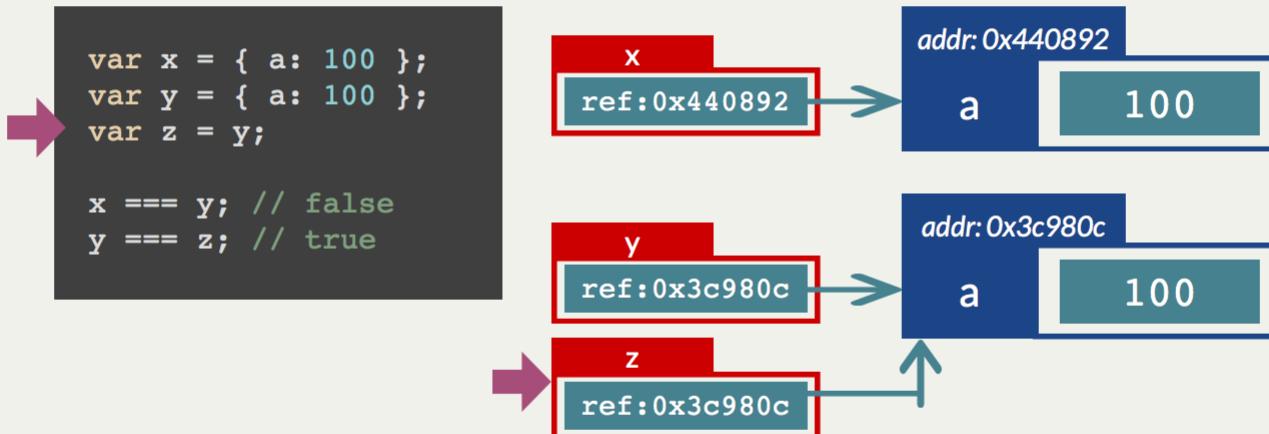
```
var x = { a: 100 };
var y = { a: 100 };
```



C or C++에서의 포인터나 주소값과 비슷하다.

# Object 타입

모든 Object 타입의 실제 값은 참조값(주소)이다.



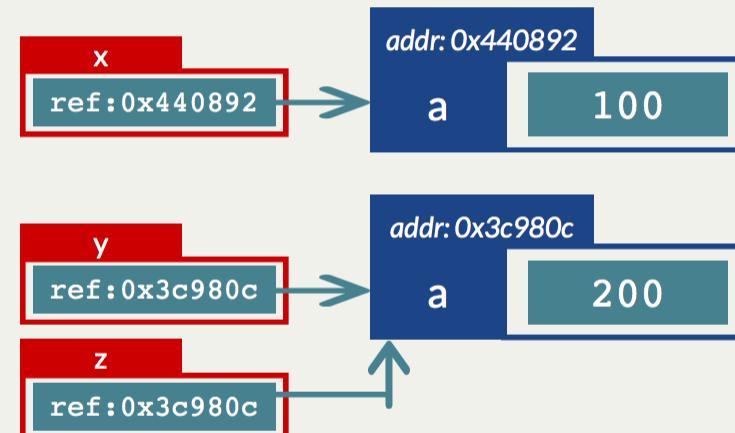
# Object 타입

모든 Object 타입의 실제 값은 참조값(주소)이다.

```
var x = { a: 100 };
var y = { a: 100 };
var z = y;

x === y; // false
y === z; // true

z.a = 200;
```



# Object 타입

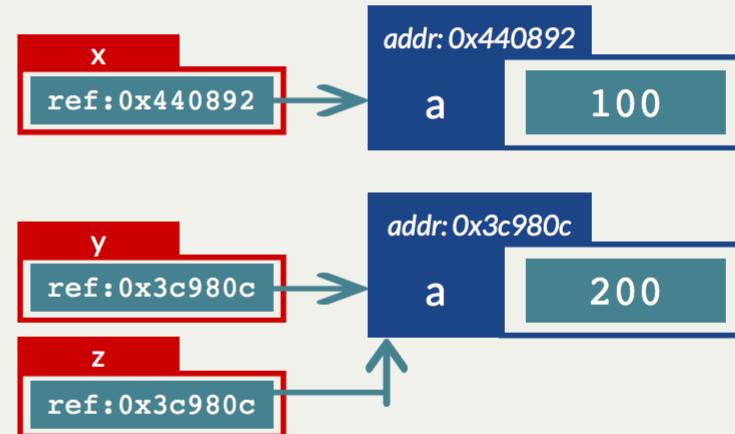
모든 Object 타입의 실제 값은 참조값(주소)이다.

```
var x = { a: 100 };
var y = { a: 100 };
var z = y;

x === y; // false
y === z; // true

z.a = 200;

x.a; // 100
y.a; // 200
z.a; // 200
```



Object 의 종류 중 하나인 “Function”

Function 은 가끔씩 “Method” 라고도 함

# **FUNCTIONS**

# 함수, Function

---

정의

function 은  
호출 가능한 *object*

# 함수 표현식, Function Expressions

“**function**” 키워드를 사용하고 이어서  
**argument list** 와 **code block**  
을 사용해 자바스크립트 함수 객체를 생성할 수 있다.

```
var a = 7;
var sayhi = function (name) {
    var a = "Hello " + name;
    console.log(a);
    return a;
};
```

자바스크립트 함수는 **first-class** 라고 한다.  
변수에 할당 가능하고 함수의 파라미터로 전달될 수 있고 반환값이 될 수도 있다.

# 함수 표현식, Function Expressions

---

## 함수 호출

함수의 이름을 통해 파라미터를 0또는 하나 이상 전달하면서 호출;

함수는 호출 될 때마다

함수에 포함된 변수와 파라미터 변수는 같은 스코프를 가진다.

```
var a = 7;
var sayhi = function (name) {
    var a = "Hello " + name;
    console.log(a);
    return a;
};

sayhi("J"); // "Hello J"
a; // 7
```

# 함수 표현식, Function Expressions

## 인자의 갯수가 일치하지 않아도 된다.

자바스크립트에서는 인자 (Argument)의 갯수가 달라도 호출이 가능  
실행 시에 값이 없이 인자로 전달되면 “**undefined**” 값으로 채워진다.

마치 변수에 값을 할당하지 않으면 **undefined**로 채워지듯이...

```
var f = function (a, b) {
    console.log(typeof b);
    return a + b;
};

f(3);          // NaN ("undefined" printed)
f(3, 4);      // 7   ("number" printed)
f(3, 4, 5);  // 7   ("number" printed)
```

# 함수 표현식, Function Expressions

---

## 함수는 값을 반환하지 않을 수 있다.

리턴값을 명시적으로 반환하는 것은 옵션이다.

명시적으로 값을 반환하지 않으면 “*undefined*” 가 반환된다.

```
var f = function () {
    return;
};

var g = function () {
};

var foo = f();
foo; // undefined
g(); // undefined
```

# 함수 표현식, Function Expressions

함수 표현식은 이름이 생략 가능하다.

```
var fact = function (n) {
    if (n === 0) return 1;
    if (n > 0) return n * fact(n - 1); // *1
};
fact(3); // 6, which is the factorial of 3

var fact2 = fact;
fact = null; // But if the value changes...
fact2(3); // TypeError: "null" is not function
```



```
var f = function fib(n) { "fib" 이름은 함수 내부에서만 사용된다.
    if (n === 0) return 0;
    if (n === 1) return 1;
    if (n > 1) return fib(n - 1) + fib(n - 2); // *2
};
f(10); // 55

var g = f;
f = null; // If the value changes...
g(10); // 55 (still working)
```

# 함수 선언식, Function Declarations

## 함수 선언식

함수 선언식은 호이스팅 영향을 받지만, 함수 표현식은 받지 않는다.

```
// We can invoke a function
// before its "declaration"
//
// Please see the next slide for the reason.....
//
var x = 100;

plusOne(x); // 101
plusOne(y); // NaN

var y = 999;

function plusOne(n) {
    return n + 1;
}
```



**“function declaration”**

함수 선언식, 호이스팅 됨

# 메서드, Method

---

정의

method 는 객체의  
프러퍼티로 사용되는 함수

# 메서드, Method

---

## 객체의 메서드

```
// cat 객체는 3개 프로퍼티가 있다.  
// cat.age, cat.meow, 그리고 cat.sleep  
  
var cat = {  
    age: 3,  
    meow: function () {}  
};  
cat.sleep = function () {};  
  
// cat.meow() 로 호출 가능하고  
// cat.sleep 은 cat의 메서드로 추가됨
```

# 메서드, Method

---

## 메서드에서 자기 객체 참조

객체 내에서 메서드가 호출될 때 **this**를 통해서 런타임 시에 자기 객체에 바인딩 된다.

```
var cat = {  
    age: 3,  
    meow: function () {  
        console.log(this.sound);  
        return this.age;  
    },  
    sound: 'meow~~'  
};  
  
cat.meow(); // 3 ("meow~~")  
  
var m = cat.meow;  
m(); // TypeError or undefined
```

# Static / Lexical Scope Model

---

함수 내부의 변수는 함수가 호출 될 때마다 참조될 수 있다.  
이것은 함수가 실행될 때가 아니라 함수가 만들어 질때 변수를  
기억하고 있기 때문이다.

이를 static 혹은 Lexical Scope 모델이라고 한다.

“**this**”만 제외하고..

또 다른 객체 배열 (Arrays)

# 배열 (ARRAY)

# 배열, Array

## 배열 리터럴

자바스크립트 **Array** 객체를 만들때는 대괄호 [ ] 기호를 사용

```
var w = [
    "test",
    1234,
    {},
    [],
    "hi"
];
w[4]; // "hi"
```

```
var w = new Array(5);
w[0] = "test";
w[1] = 1234;
w[2] = new Object();
w[3] = new Array();
w[4] = "hi";
w[4]; // "hi"
```

양쪽 코드는 모두 동일한 코드이다.

# 배열, Array

## 배열에 엘레먼트 추가

“push” 메서드를 통해 엘레먼트 추가  
인덱스를 통해 데이터 수정/교체 가능

```
var arr = [ "test", 1234, {}, [], "hi" ];

arr.push("sixth"); // 6
arr.length;        // 6
arr[5];           // "sixth"

arr[7] = 012;      // 10
arr.length;        // 8

arr[6];           // undefined
arr[7];           // 10

arr[8];           // undefined
arr.length;        // 8
```

# 배열의 순회, Enumeration of Array

## 배열의 엘레먼트 순회하기 (1/3)

모든 배열 객체에는 “length”라는 특별한 속성을 가진다.

```
var arr = [ "test", 1234, {}, [], "hi" ];

for (var i = 0; i < arr.length; i += 1) {
  console.log(arr[i]);
}
```

NOTE: “For-loop”는 배열을 순회할 때 추천하지 않음

# 배열의 순회, Enumeration of Array

## 배열의 엘레먼트 순회하기 (2/3)

모든 배열 객체에 있는 `forEach` 메소드

```
var arr = [ "test", 1234, {}, [], "hi" ];

arr.forEach(function (val /*, i, arr*/) {
  console.log(val);
});
// undefined
```

“`forEach`” 메서드를 사용하는 것이 좋다.

# 배열의 순회, Enumeration of Array

## 배열의 엘레먼트 순회하기 (3/3)

모든 배열 객체에 포함된 "map" 메서드

```
var arr = [ "test", 1234, {}, [], "hi" ];

arr.map(function (val /*, i, arr*/) {
    return typeof val;
});
// [ "string",
//   "number",
//   "object",
//   "object",
//   "string" ]
```

함수형 기법인 “**map**”, “**every**”, “**some**”, 등 사용

# 객체의 순회, Enumeration of Object

## 객체를 순회하는 방법 (1/2)

Object.keys( ... ) 를 통해 객체의 모든 “key” 를 반환 받을 수 있음

```
var obj = { a: "test", b: 1234, c: "hi" };

Object.keys(obj); // [ "a", "b", "c" ]
```

# 객체의 순회, Enumeration of Object

## 객체를 순회하는 방법 (2/2)

Object.keys( ... ) 를 map 과 함께 사용하여 모든 값을 가져올 수 있음

```
var obj = { a: "test", b: 1234, c: "hi" };

Object.keys(obj).map(function (key) {
    return obj[key];
});
// [ "test",
//   1234,
//   "hi" ]
```

## 기타 참고자료

---

대부분의 자바스크립트 문법은 자바와 C, C++와 유사  
[MDN Reference](#) 를 참조하여 문제를 해결

### 책과 구글을 참조로.

- MDN “A re-introduction to JavaScript”
- “JavaScript Garden” 튜토리얼
- “Eloquent JavaScript” 온라인 북
- MDN JavaScript Reference Page
- Airbnb 자바스크립트 스타일 가이드

Function & Prototype chaining

# 함수와 프로토타입 체이닝

## 함수와 관련된 개념 학습

---

- 함수 생성
- 함수 객체
- 다양한 함수 형태
- 함수 호출과 this
- 프로토타입과 프로토타입 체이닝

# 함수 정의

---

- ☑ 함수를 생성하는 방법
  - 함수 선언문, function statement
  - 함수 표현식, function expression
    - 익명 함수(anonymous function)를 이용해 함수 생성
  - Function() 생성자 함수
  
- ☑ 함수 호이스팅
  - 함수 표현식으로 함수 생성 권장 (호이스팅 발생 되지 않음)

## 함수 객체

---

- ☑ 함수도 객체다
  - 함수에서도 프로퍼티 추가 가능
  
- ☑ 함수는 값으로 취급된다
  - 리터럴에 의한 생성
  - 변수나 배열의 요소, 객체의 프로퍼티에 할당 가능
  - 함수의 인자로 전달 가능
  - 함수의 리턴값으로 리턴 가능
  - 동적으로 프로퍼티를 생성 및 할당 가능

함수를 일급 객체 라고 함

## 함수 객체의 기본 프로퍼티

---

- ☑ 모든 함수는 length 와 prototype 프로퍼티를 가져야 함
  - ECMAScript 명세에 기술되어 있음
  
- ☑ name : 함수의 이름 (익명일 때는 빈 문자열)
- ☑ caller : 자신을 호출 한 함수
- ☑ arguments : 함수를 호출할 때 전달된 인자값
- ☑ length : 함수 생성 시 인자 갯수
- ☑ prototype : 생성자를 이용한 함수 생성 시 사용됨

# 함수의 다양한 형태

---

## 콜백 함수

- 어떤 이벤트가 발생하거나 특정 시점에 도달했을 때 호출되어 짐
- 특정 함수를 인자로 넘겨 함수 내부에서 호출되는 함수

## 즉시 실행 함수

- 함수를 정의 함과 동시에 바로 실행하는 함수
- 최초 단 한번만 실행하면 되는 초기화 함수 같은데서 사용
- 사례로 jQuery 같은 경우는 모든 코드가 즉시 실행함수로 되어 있음
- 변수의 스코프 때문에 사용하기도 함

## 내부 함수

- 함수 코드 내부에서 다시 함수 정의가 가능
- 내부 함수에서는 자신을 둘러싼 부모 함수의 변수에 접근 가능 (클로저)

## 함수를 리턴하는 함수

- 함수 자체를 리턴 가능

# 함수 호출과 this

---

## ☒ arguments 객체

- 자바스크립트는 함수의 인자 갯수와 관계없이 호출 가능
- 인자가 없을 때는 undefined 값이 할당
- 초과된 인자는 무시
- 함수 호출 시 암묵적으로 arguments 객체가 함수 내부로 전달됨
- 유사 배열 객체 (array-like object) 임

## 호출 패턴과 this 바인딩

---

- ☑ 함수 호출 시 arguments 객체와 this 가 함수 내부로 전달
  - 함수가 호출 되는 방식에 따라 this 가 바인딩하는 객체가 달라짐
  
- ☑ 호출 패턴에 의한 this 바인딩
  - 객체의 메서드 호출 시 this 바인딩
    - 해당 메서드를 호출 한 객체로 바인딩 됨
  - 함수를 호출 할 때 this 바인딩
    - 전역 객체에 바인딩 (브라우저 자바스크립트의 경우 window 객체)

노드

Node.js

**NODE.JS**

# Node.js 소개

- Google V8 JavaScript Engine
- 2008년 9월 8일 Google 이 크롬 브라우저 출시

구글은 지난 2년간 구글 크롬 프로젝트를 진행해온 것으로 알려졌다. 2006년 10월 있었던 MS의 익스플로러7 출시가 웹브라우저 개발에 큰 영향을 미쳤다고 한다.



구글에 따르면 '구글 크롬'은 애플 사파리에도 탑재된 오픈소스 엔진 웨비트에 기반하고 있다. 자바 스크립트 버추얼 머신 V8도 탑재, 빠르게 자바 스크립트 환경을 이용할 수 있다.

'V8'은 멀티 프로세서 환경에도 적합하다고 구글은 강조했다

[http://www.zdnet.co.kr/news/news\\_view.asp?artice\\_id=00000039172677&type=det&re=](http://www.zdnet.co.kr/news/news_view.asp?artice_id=00000039172677&type=det&re=)

# Node JS

- Google V8 Javascript Engine

## V8 (자바스크립트 엔진)

위키백과, 우리 모두의 백과사전.

V8 자바스크립트 엔진(V8 JavaScript Engine)은 구글에서 개발된 오픈 소스 JIT 가상 머신형식의 자바스크립트 엔진이며 구글 크롬 브라우저와 안드로이드 브라우저에 탑재되어 있다.<sup>[1]</sup> V8로 줄여 불리기도 하며, 현재 라스 백이 책임 프로그래머이다.<sup>[2]</sup> ECMAScript(ECMA - 262) 3rd Edition 규격의 C++로 작성되었으며, 독립적으로 실행이 가능하다. 또한 C++로 작성된 응용 프로그램의 일부로 작동할 수 있다.

V8은 자바스크립트를 바이트코드(bytecode)로 컴파일하거나 인터프리트(interpret)하는 대신 실행하기 전 직접적인 기계어(x86, ARM, 또는 MIPS)로 컴파일(compile)하여 성능을 향상시켰다. 추가적인 속도향상을 위해 인라인 캐싱(inline caching)과 같은 최적화 기법을 적용하였다.

- Google Chrome에 탑재된 Google V8 Javascript Engine(V8)이 자바스크립트의 속도를 대폭 향상시켜 OS Application을 개발하기에 적합한 언어로 탈바꿈 함.

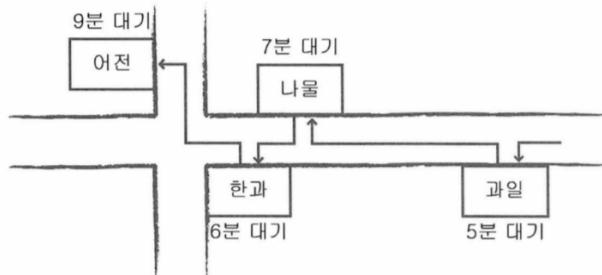
# Node JS

---

- 서버사이드 자바스크립트
  - V8 엔진으로 컴파일 - 구글 크롬의 자바스크립트 엔진
  - 자바스크립트로 코딩 - ECMA5, ECMA6 사용
- 비 동기, 이벤트 기반, **Non-Blocking I/O**
  - 하나의 쓰레드(프로세스)가 이벤트 루프를 구동
  - 고성능 서버의 쉬운 개발이 가능
- 목적은 확장 가능한 네트워크 프로그램을 간단히 만드는 것
  - 구현이 C,C++ 이어서 빠름
- 2009년 라이언 달이 JSConf에 발표하면서 화제
  - 오픈소스, MIT 라이센스

# Node JS

- Node JS 는 싱글 쓰레드 - 이벤트 기반 동작
  - Single Thread 기반의 Non-Event 방식으로 진행할 경우

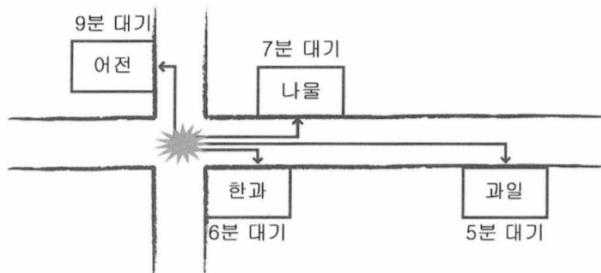


과일 – 나물 – 한과 – 어전 순으로  
이동했을 때 걸리는 시간

$$5 + 7 + 6 + 9 = 27$$

총 27분 소요

- Multi Thread 기반의 Non-Event 방식으로 진행할 경우

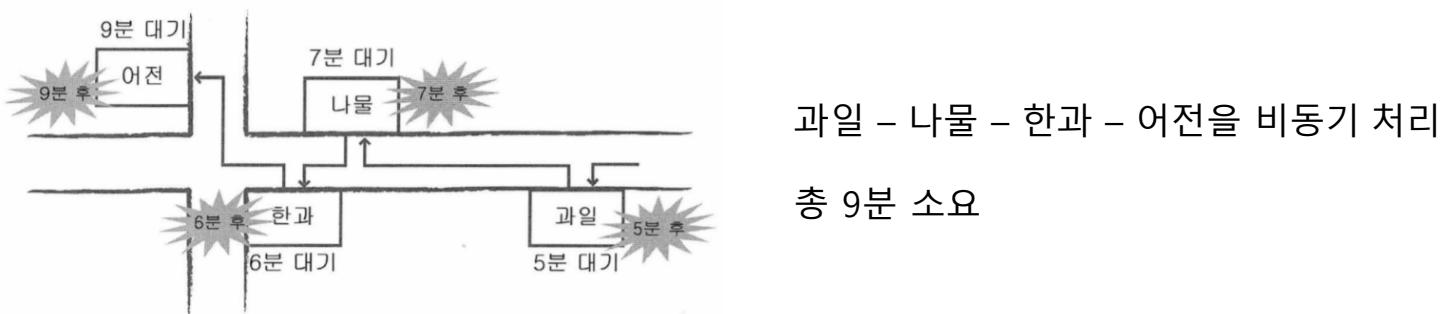


과일 – 나물 – 한과 – 어전을 동시 처리

총 9분 소요

# Node JS

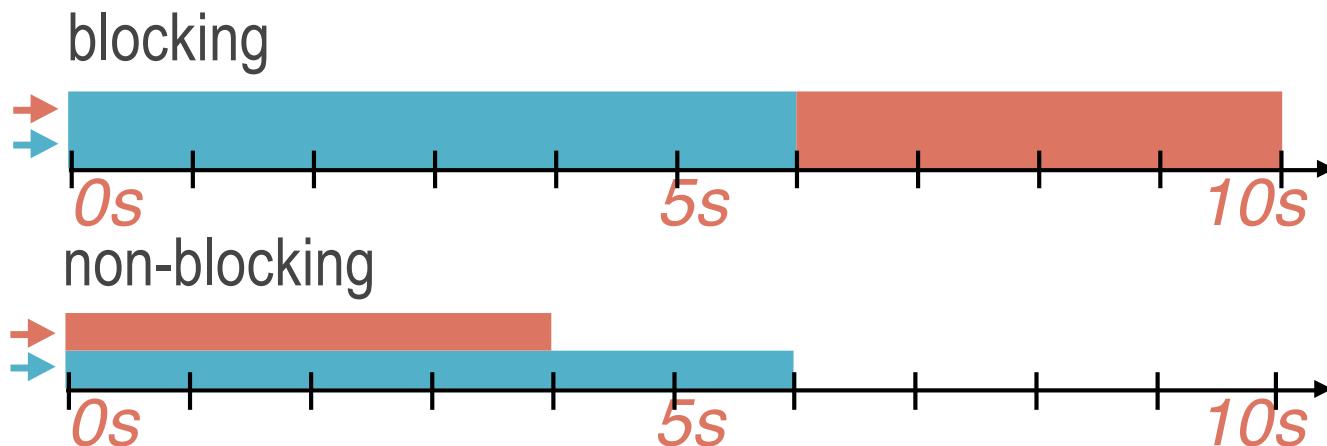
- Multi Thread 기반의 Non-Event 방식의 단점
  - 각 Thread가 데이터 참조를 할 때 잘못 변경될 여지가 농후함.
  - 에러가 발생했을 때 Debugging이 쉽지 않음.
- Single Thread 기반의 Event 비동기 방식으로 진행할 경우



대기표 발급 후 시간이 되었을 때 즉시 처리해주는 방법

# Node JS

- Single Thread 기반 Event 방식의 처리 속도
  - 최근의 Web Application 들은 서버내에서 수학적 연산을 처리하는 경우는 거의 존재하지 않음.
  - 보통 요청을 전달받아 Database 에 전달하는 방식으로 동작됨.
  - 초 고성능의 서버를 필요로 하지 않으면서 최적의 성능을 유지시킬 수 있음.



# Node JS

- Single-threaded / Non-blocking IO / Event-loop 디자인

```
var fs = require('fs');

fs.readFile('treasure-chamber-report.txt', function(report) {
  console.log("oh, look at all my money: "+report);
});

fs.writeFile('letter-to-princess.txt', '...', function() {
  console.log("can't wait to hear back from her!");
});
```

- 노드에게 파일 읽기와 쓰기 명령을 수행하고.. 잠들어 버림
  - 노드가 어떤 일을 끝마치면 콜백이 수행 됨
  - 그 콜백의 수행이 종료 될 때까지 다른 콜백은 그 라인에서 기다림
  - 어느 콜백이 먼저 수행 될지 알 수 없음
  - 즉, 같은 데이터구조에 동시에 접근하는 것을 걱정 할 필요가 없다!

## 참고 : I/O 문제

- 웹 서비스 요청
- 데이터베이스 읽기/쓰기
- 파일 시스템 읽기/쓰기

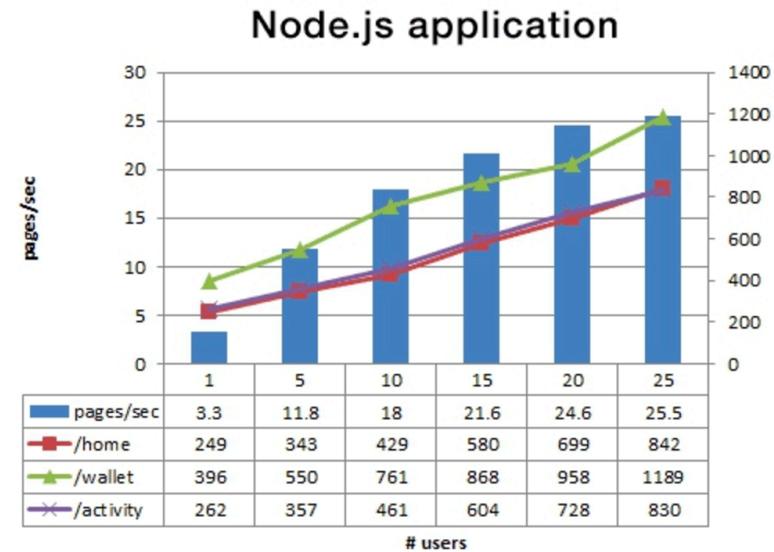
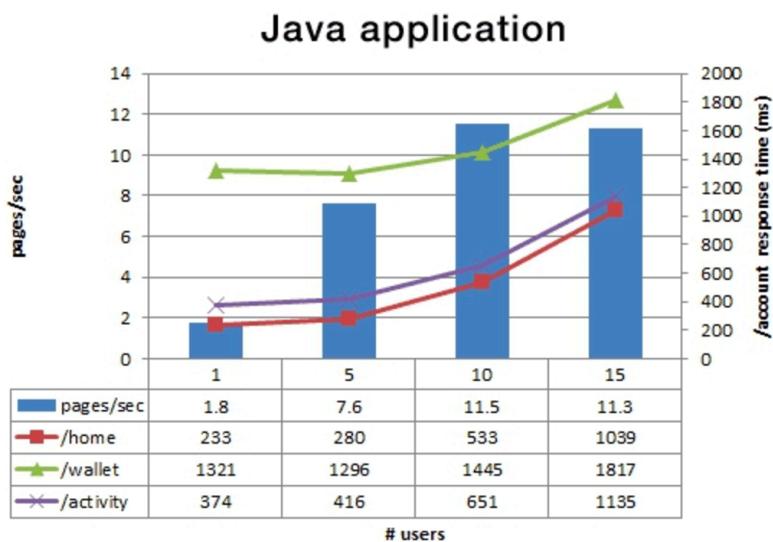
### The cost of I/O

L1-cache	3 cycles
L2-cache	14 cycles
RAM	250 cycles
Disk	41 000 000 cycles
Network	240 000 000 cycles

# Node JS

- 페이팔

- Java Application 을 Node.js 로 개발한 결과 기존 보다 45% 빠르게 처리함.
- <http://blog.builtinnode.com/post/from-java-to-node---the-paypal-story>



# Node JS

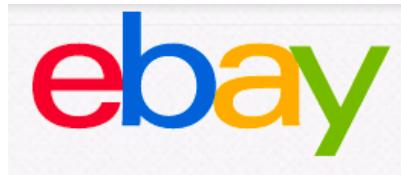
---

- 월마트
  - 2013년 말 블랙프라이데이 당시의 온라인 트랜잭션의 53%를 Node.js로 처리함.
  - 오후 6시 ~ 오후 10시 총 4시간 동안 천만건의 온라인 트랜잭션 발생
  - “Walmart Processes 10 Million Transactions in 4 Hours” - <http://globaleconomicanalysis.blogspot.kr/2013/11/black-friday-roundup-walmart-10m.html>
  - 당시, 서버가 다운되는 사례는 전혀 없었으며, 모든 서버의 CPU 이용률이 1% 미만으로 유지되었음.
- 야후
  - 분당 168만 ~ 200만건의 요청을 관리할 목적으로 Node.js를 사용함. (한달 725억 7천 6백만건)
- 그루폰
  - 분당 5만건의 미국내의 온라인 트랜잭션을 Node.js로 처리함. (한달 21억 6천만건)
- 그루폰·월마트·야후는 왜 노드JS를 품었나
  - [http://www.zdnet.co.kr/news/news\\_view.asp?artice\\_id=20140307103242](http://www.zdnet.co.kr/news/news_view.asp?artice_id=20140307103242)

# Node JS

---

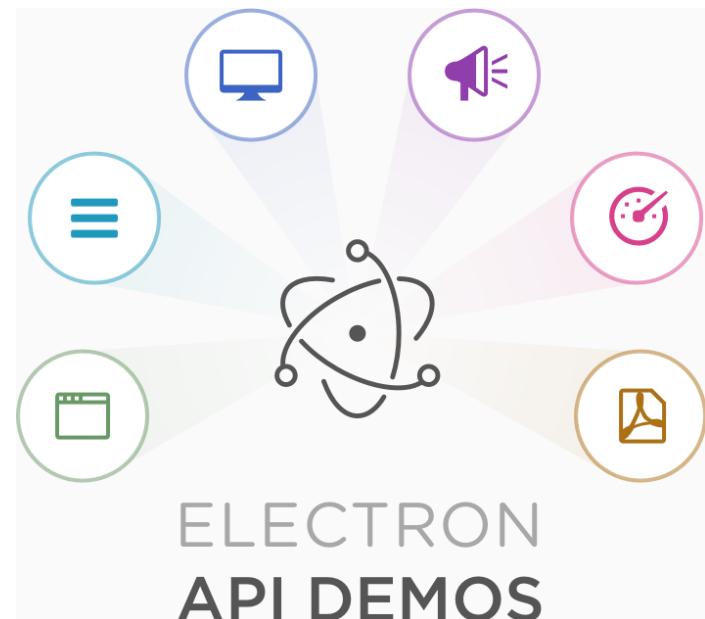
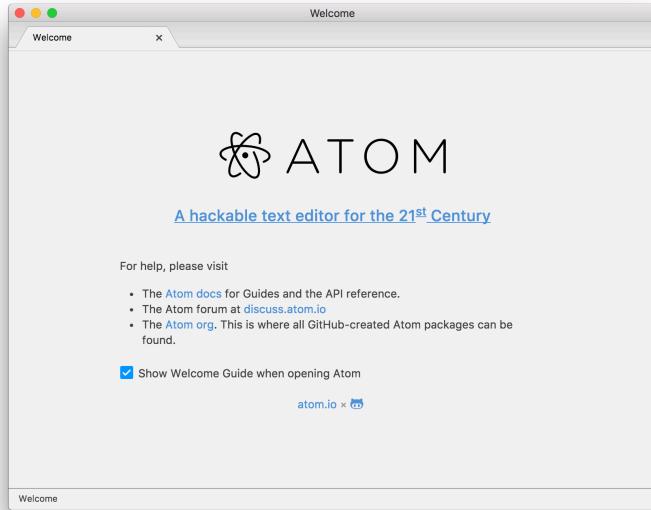
- 그 외 Node.js 를 사용하는 기업들



- 이베이, 마이크로소프트, 다우존스, 우버, 뉴욕타임즈

# Node JS

- Node.JS 는 OS Application 을 제작할 수도 있습니다.
  - 엘렉트론 <https://electron.atom.io>



# Node JS - 마지막

---

- 노드를 써야 되는 이유?
  - 자바 스크립트를 사용 - 세상에서 가장 빠른 인터프리터 엔진 V8
  - IO 가 너무 효율적 이어서 : 낮은 응답시간과 높은 동시성이 중하다면 고려
- 윈도우에도 잘 지원 되나?
  - MS에서 적극적으로 지원, 심지어 비주얼 스튜디오에서도 개발 가능
- 이벤트 주도 프로그래밍의 문화에 적응하자!
  - 매우 쉽게 복잡한 코드를 작성 가능

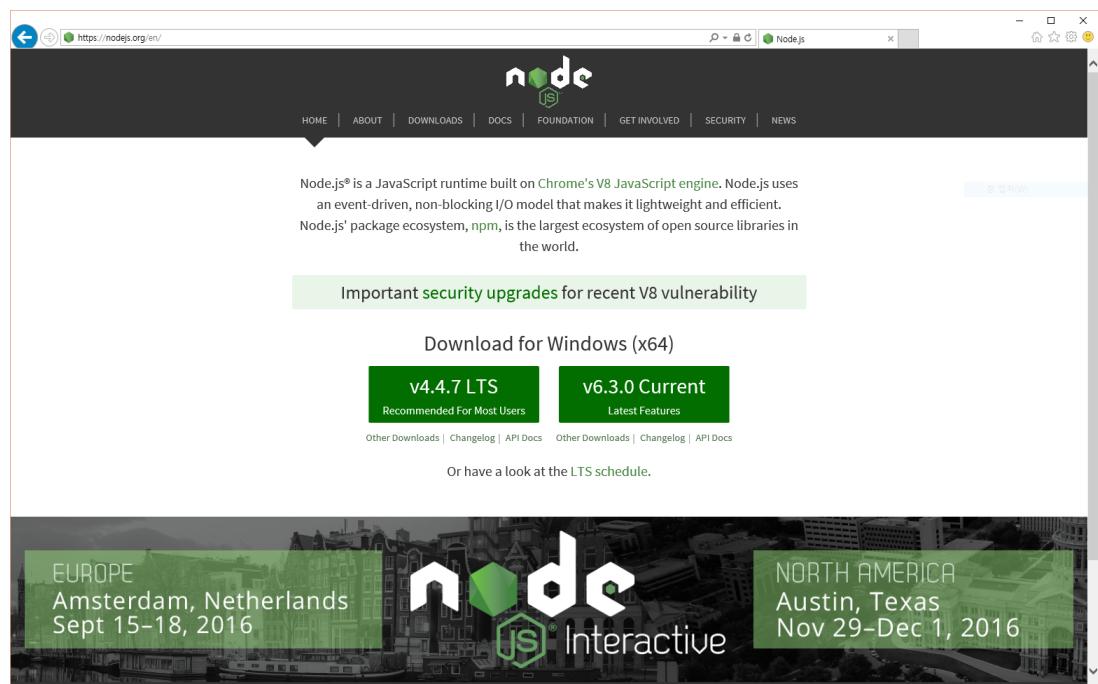
Install Node

# **NODE 설치**

# 노드 설치

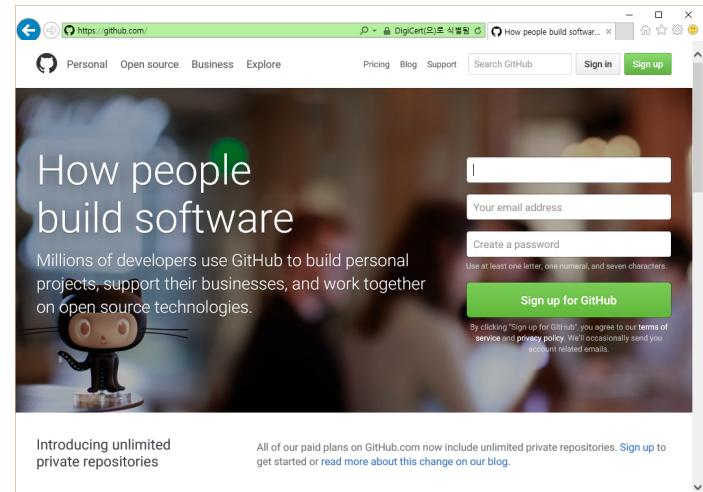
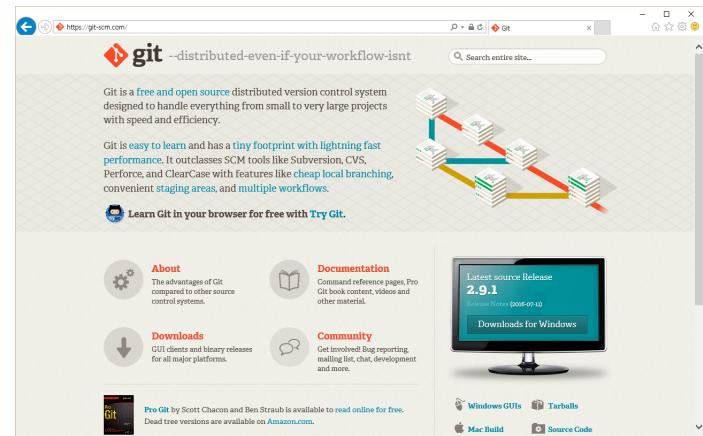
- Node.js 와 npm 설치 : <http://nodejs.org>
  - Node 설치 시 NPM 기본으로 같이 설치 됨
- Git 설치 : <https://git-scm.com/>
- 설치 확인
  - node --version
  - npm --version
  - git --version

위 세 가지  
터미널에서 설치 확인

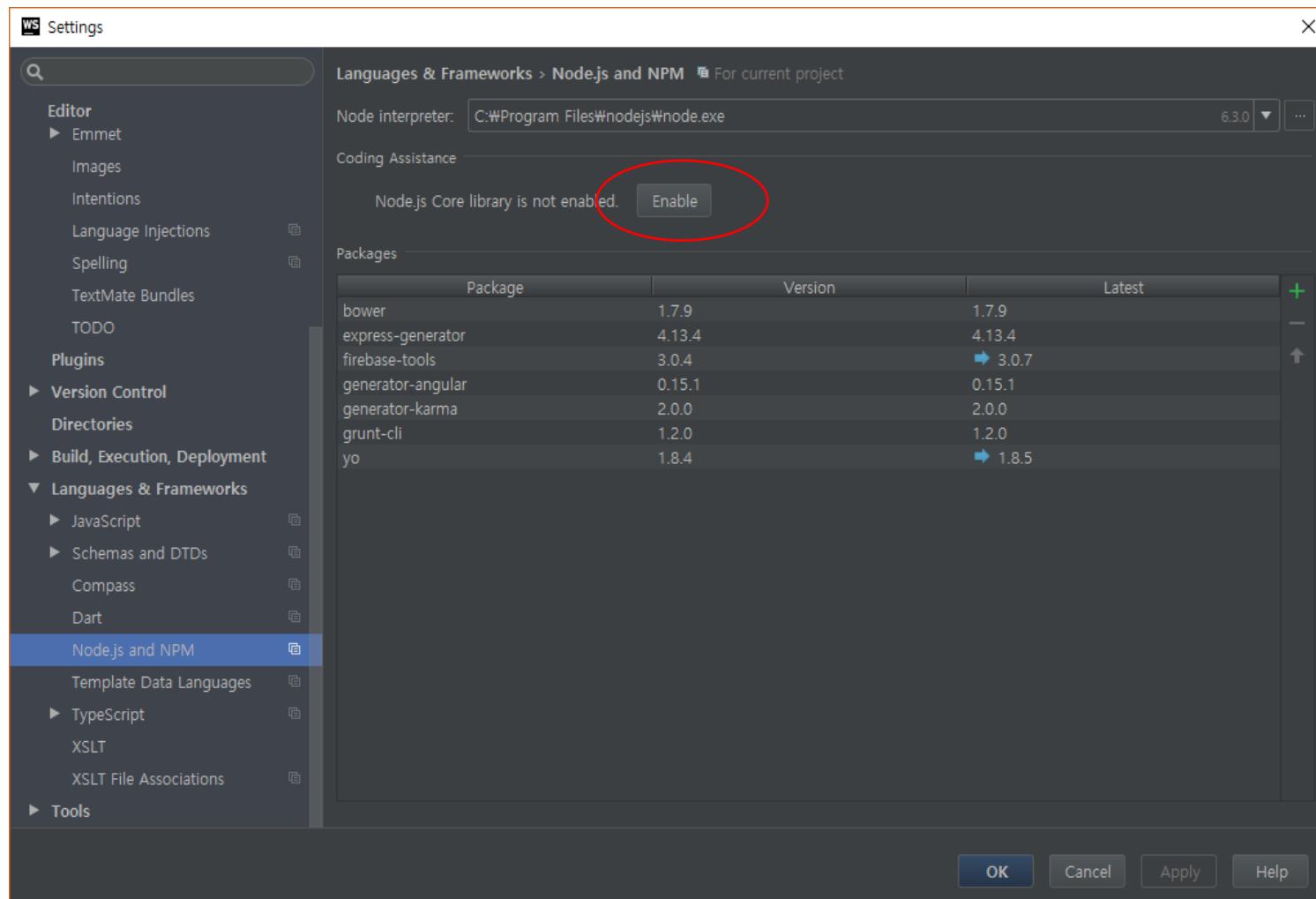


# 노드 설치 : 와 관련된 기타 도구 들

- Git 설치 : <http://git-scm.com>
  - 설치 후 터미널에서 git 명령어 확인
  - 명령어 실행 되지 않으면 path 설정
- Github 계정 등록 (이미 있으시면 ok!)
  - <http://github.com>
- Node 콘솔
  - REPL (노드 설치 시 자동 설치)
- Node 개발 에디터(IDE)
  - 서브라임 텍스트
  - Atom
  - WebStorm (추천)
  - VS Code (마이크로소프트)



# 웹스톰 - 노드 설정



```
var fs = require('fs');
```

Chaining IO

Event loop

## NODE 주요 코어 모듈 - FS

# node 주요 모듈 - fs 모듈 사용 해보기

```
var fs = require('fs');

fs.readFile(__filename, 'utf8', function(err, fileContent) {
  if (err) {
    console.error(err);
  }
  else {
    console.log('got file content:', fileContent);
  }
});
```

- 위의 코드는 콜백 패턴을 사용한다.
  - 단일 쓰레드 비동기 방식 : **이벤트 루프**를 사용
- 실습 : 위 코드를 일부 수정하여 비 동기 호출을 확인한다.

# Blocking Code vs. Non-Blocking Code

## • Blocking Code

```
var contents = fs.readFileSync('/etc/hosts');
console.log(contents);
console.log('Doing something else');
```



## • Non-Blocking Code

```
fs.readFile('/etc/hosts', function(err, contents) {
  console.log(contents);
}) ;
console.log('Doing something else');
```



# 노드 vs. 전통적 웹 서버(아파치) : 기본 실행 원리

Node

전통적 웹 서버

- 비동기 이벤트 위주 I/O 사용
  - 요청에 단일 쓰레드(프로세스)로 작동
  - 적은 양의 리소스 사용
  - 순차적으로 수행
  - 간단한 작업, 접근빈도가 높은 웹 어플리케이션에 적합
- 쓰레드나 별도의 프로세스 사용
  - 요청마다 매번 쓰레드나 프로세스를 생성
  - 많은 양의 리소스를 사용
  - 병렬로 수행

# fs 모듈을 통한 Chaining IO 예제

```
var fs = require('fs');

fs.stat(__filename, function(err, stats) {
  if (err) {
    console.error(err);
  } else {
    if (stats.size < 1024) {
      fs.readFile(__filename, 'utf8', function(err, fileContent) {
        if (err) {
          console.error(err);
        } else {
          console.log('Got file content:', fileContent);
        }
      });
    } else {
      console.log('Didn\'t read the file, it was too long.');
    }
  }
})
```

- 콜백의 depth가 깊어지면 콜백 지옥(callback hell)이 될 수 있음
- 2단계 이상의 콜백이 필요할 때는 별도의 함수로 분리
  - 자바스크립트 Node 코딩 가이드라인 참조
  - <http://nodeguide.com/style.html>

# 웹 서버 : 이벤트 루프 예제

```
var http = require('http');

http.createServer(function (req, res) {
    res.writeHead(200);
    res.write('Dog is running..');

    setTimeout(function () {
        res.write('Dog is done..');
        res.end();
    }, 5000);

    res.write('Dog is still running..');
}).listen(8090);

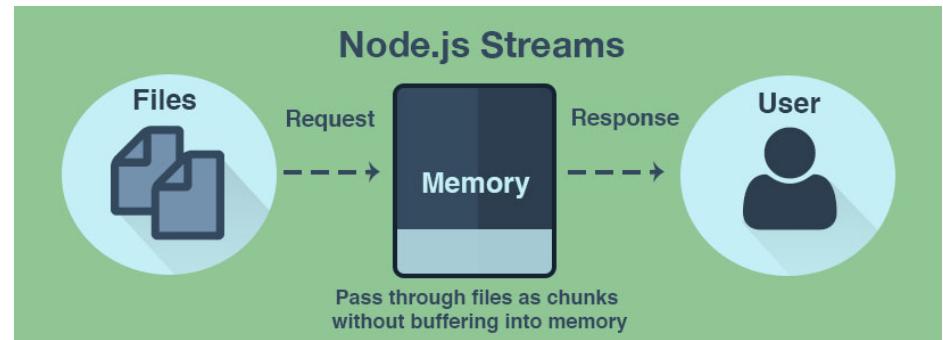
console.log('server ready on port 8090');
```

- setTimeout 는 자바스크립트 함수로써 특정 시간 동안 프로그램을 중지 시킴
  - 5000(ms) → 5초

# 타이머

---

- 클라이언트 자바스크립트 타이머 함수
  - 전역 window 객체에 속해 있음
- Node에서도 타이머 관련 함수가 포함되어 있음
  - 브라우저에서의 타이머 함수와 동일하게 동작 (V8 엔진에서 수행)
  - 종류
    - setTimeout
    - clearTimeout
    - setInterval
    - clearInterval



stream all the things!

**스트림**

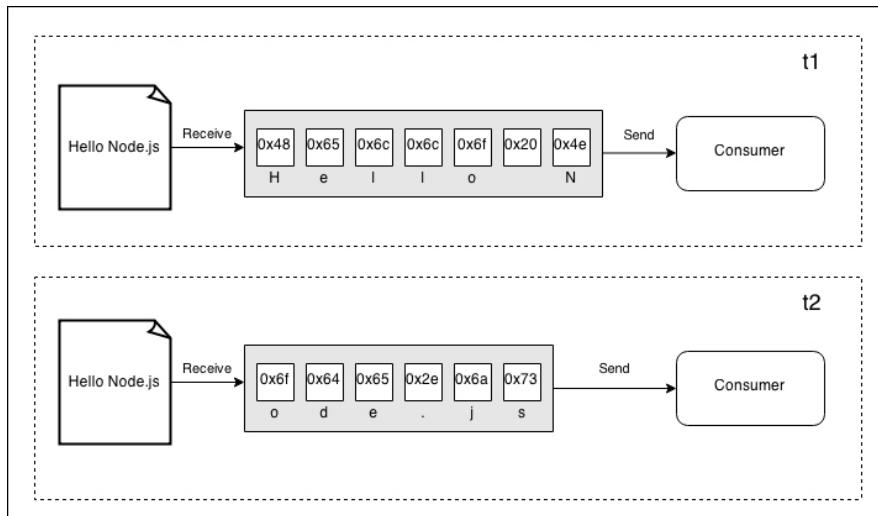
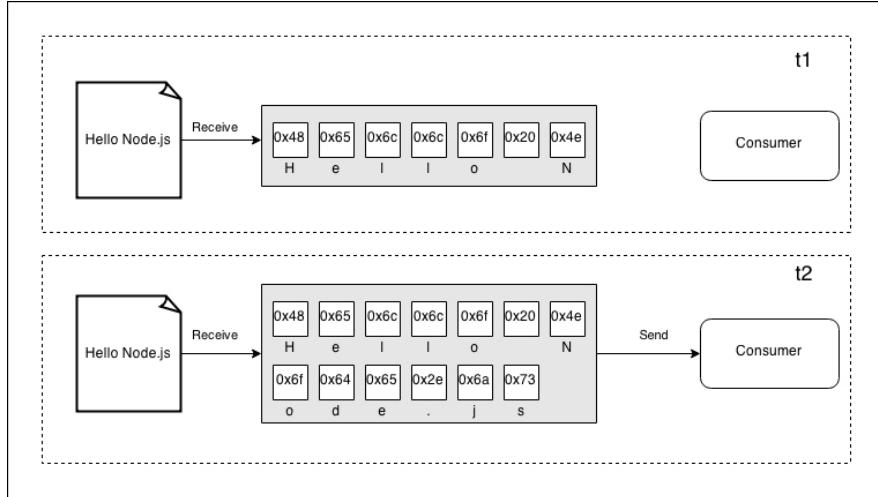


# 스트림

---

- 노드는 내장 스트림을 포함 - 유닉스 스타일
- fs.read() vs. stream.read()
  - 한 번에 메모리에 로딩 읽는 방식 vs. 단위(chunk)로 끊어서 읽는 방식
- Readable Stream : 읽기 스트림
  - 파일이나 소켓으로 부터 데이터를 읽어 들임
- Writable Stream : 쓰기 스트림
  - 메모리로 부터 데이터를 파일이나 소켓으로 쓰기를 수행
- Read/Write 스트림 모두 이벤트를 기반으로 수행 됨

# 버퍼링 vs. 스트리밍



- **버퍼링**

모든 데이터를 메모리에 올린 뒤에 처리

- **스트리밍**

특정 데이터 단위(chunk)로 잘라서 처리

- **공간효율과 시간효율이 생김**

# Readable Stream

```
var fs = require('fs');

var stream = fs.createReadStream('/path/to/large/file');

stream.on('readable', function() {
  var chunk;
  while(chunk = stream.read()) {
    console.log('got NPM data chunk of %d bytes', chunk.length);
  }
});

stream.once('end', function() {
  console.log('stream ended');
});
```

- read 스트림은 ‘readable’과 ‘end’ 이벤트를 발생(emit) 시킨다.
  - readable : 한 번에 읽어들일 단위(chunk)까지 데이터가 모이면 발생
  - end : 데이터를 다 읽어 들였을 때 발생, 한 번만 발생됨

# Writable Stream

```
var fs = require('fs');

var stream = fs.createWriteStream(__dirname + '/out.txt');

var interval = setInterval(function() {
  stream.write('tick ' + Date.now() + '\n');
}, 100);

setTimeout(function() {
  clearInterval(interval);
  stream.end();
}, 4950);
```

- 두 개의 타이머를 이용해 이벤트를 발생 시킨다.
  - 첫 번째 타이머 : 100밀리초마다 현재 날짜정보를 포함한 스트링을 출력
  - 두 번째 타이머 : 5초 후 첫 번째 타이머를 종료하고 쓰기 스트림 종료

Core Module - etc

# 기타 코어 모듈

# Module - OS

- OS Module - 운영체제에 대한 정보를 가지고 있음

```
var os = require("os"); // os module을 가져와 os 변수에 저장

var hostname = os.hostname(); // 운영체제의 호스트 이름을 리턴함
var type = os.type(); // 운영체제의 이름을 리턴함
var platform = os.platform(); // 운영체제의 플랫폼을 리턴함
var arch = os.arch(); // 운영체제의 아키텍처를 리턴함
var release = os.release(); // 운영체제의 버전을 리턴함
var uptime = os.uptime(); // 운영체제가 실행된 시간을 리턴함
var loadavg = os.loadavg(); // 로드 에버리지 정보를 담은 배열을 리턴함
var totalmem = os.totalmem(); // 시스템의 총 메모리를 리턴함
var freemem = os.freemem(); // 시스템의 사용 가능한 메모리를 리턴함
var cpus = os.cpus(); // CPU의 정보를 담을 객체를 리턴함.
var networkInterfaces = os.networkInterfaces(); // 네트워크 인터페이스의 정보를 담은 배열을 리턴함.

console.log("hostname : " + hostname);
console.log("type : " + type);
console.log("platform : " + platform);
console.log("arch : " + arch);
console.log("release : " + release);
console.log("uptime : " + uptime);
console.log("loadavg : " + loadavg);
console.log("totalmem : " + totalmem);
console.log("freemem : " + freemem);
console.log("cpus : " + cpus);
console.log("networkInterfaces : " + networkInterfaces);
```

# Moduel - OS

- OS Module
  - 운영체제에 대한 정보를 가지고 있음

```
>node node.os
hostname : DESKTOP-T870EL7
type : Windows_NT
platform : win32
arch : x64
release : 10.0.14393
uptime : 22304.360046
loadavg : 0,0,0
totalmem : 8512487424
freemem : 5279588352
cpus : [object Object],[object Object],[object Object],[object Object]
networkInterfaces : [object Object]

>
```

ES6

**ECMA SCRIPT 2015+**

# ECMA Script 2015는 무엇인가?

---

- ECMA International의 ECMA-262에 근거한 표준 스크립트 언어
- ECMA Script 2015 혹은 ES6 라고 불린다.
- 최초 ECMA Script는 브라우저 언어인 Javascript와 Jscript간 차이를 줄이기 위한 공통 스펙 제안으로 출발 (1992, ECMA-262)

## ECMA?

- 1961년 설립된 국제 표준화 기구.
- European Computer Manufacturers Association.
- 유럽에서 컴퓨터 시스템을 표준화하기 위해 설립됨.
  - 주요 규격
    - ECMA-119 – CD-ROM 볼륨 및 파일 구조 표준화
    - ECMA-262 – ECMAScript 언어 규격 표준화
    - ECMA-334 – C# 언어 규격 표준화
    - ECMA-335 – CLI(공통 언어 기반) 표준화
    - ECMA-404 – JSON 표준화

# ECMA Script 2015 지원 현황

ES COMPAT ECMAScript 5 6 2016+ next intl non-standard compatibility table Flattr by kangax & webbedspace & zloirock Fork 549

Sort by Engine types Show obsolete platforms Show unstable platforms

V8 SpiderMonkey JavaScriptCore Chakra Carakan KJS Other  
Minor difference (1 point) Small feature (2 points) Medium feature (4 points)  
Large feature (8 points)

Feature name	Current browser	Compilers/polyfills												Desktop												
		Traceur	Babel + core-js <sup>[2]</sup>	Closure	Type-Script + core-js	es6-shim	Kong 4.14 <sup>[3]</sup>	IE 11	Edge 15	Edge 16	Edge 17 Preview	FF 52 ESR	FF 56	FF 57 Beta	FF 58 Ni											
<strong>Optimisation</strong>																										
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
<strong>Syntax</strong>																										
default function parameters	7/7	4/7	4/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	6/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	
rest parameters	5/5	4/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	
spread (...) operator	15/15	15/15	13/15	12/15	4/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	
object literal extensions	6/6	6/6	6/6	4/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	
for...of loops	9/9	9/9	9/9	6/9	3/9	0/9	0/9	0/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	
template literals	5/5	4/5	4/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	
RegExp "y" and "u" flags	5/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	
destructuring, declarations	22/22	20/22	21/22	20/22	15/22	0/22	0/22	0/22	22/22	22/22	22/22	22/22	21/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	
destructuring, assignment	24/24	23/24	24/24	21/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	
destructuring, parameters	24/24	19/24	21/24	18/24	16/24	0/24	0/24	0/24	23/24	23/24	23/24	23/24	21/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	
<strong>Bindings</strong>																										
const	16/16	14/16	14/16	14/16	14/16	0/16	2/16	12/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	
let	12/12	10/12	10/12	c	Yes	Yes	Yes																			

<https://kangax.github.io/compat-table/es6>

# Getting Started with ECMAScript

---

1. let & const 키워드
2. Spread 연산자와 rest 파라미터
3. Iterable 과 객체 구조분해(destructuring)
4. 화살표 함수 (Arrow functions)
5. 클로저와 사용
6. 오브젝트와 클래스

ECMAScript 2015에서 새롭게 추가된 Block–Level Scope 에 대해 학습한다.

let 과 const

# 블록 바인딩

# 기존 Javascript Variable Scope

## ☑ var - function scoped

```
function getValue(condition) {  
  if ( condition ) {  
    var value = "blue";  
  
    // Some code..  
    return value;  
  }  
  else {  
  
    // value는 여기서 undefined로 존재한다.  
    return null;  
  }  
  
  // value는 여기서 undefined로 존재한다.  
}  
}
```

```
function getValue(condition) {  
  var value;  
  if ( condition ) {  
    value = "blue";  
  
    // Some code..  
    return value;  
  }  
  else {  
  
    // value는 여기서 undefined로 존재한다.  
    return null;  
  }  
  
  // value는 여기서 undefined로 존재한다.  
}
```

- ☑ Function 내부에서 선언된 모든 변수들은 Function 선언 아래쪽으로 이동되어 선언
- ☑ 이 현상을 “호이스팅”이라 한다.

# undefined?

- ✓ Java에서 Null 과 같은 자료형
- ✓ 변수가 선언은 되어있지만 데이터가 초기화 되지 않은 상태

```
var variableA = "Value";
var variableB = null;
var variableC;

console.log(variableA);
console.log(variableB);
console.log(variableC);
```

Value  
Null  
Undefined

- ✓ Null은 개발자가 직접 할당해야 한다.

# ECMAScript Variable Scope

---

- ✓ ECMAScript 2015에서  
Lexical Scope(Block-Level Scope)로 변경됨.
- ✓ Lexical Scope(Block-Level Scope)?
  - 함수 내부
  - 블록 내부( { 와 }를 사용하여 지정 )
- ✓ Hosting(호이스팅)이 더 이상 발생하지 않는다.
- ✓ 변수(**let**)와 상수(**const**)로 구분해 지원한다.

# let

---

- ✓ var와 같은 문법으로 변수를 정의함.
- ✓ Block-scoped variables

```
function getValue(condition) {  
  if ( condition ) {  
    let value = "blue";  
  
    // Some code..  
    return value;  
  }  
  else {  
  
    // value는 여기에 존재하지 않는다.  
    return null;  
  }  
  
  // value는 여기에 존재하지 않는다.  
}
```

- ✓ Value는 블록에 따라 제한적으로 존재한다.

# let – 새 정의 금지

- ✓ var는 중복 정의시 Error를 발생시키지 않고 값을 덮어 쓴다.

```
var variableA = "Kim Yoona";
console.log(variableA);
```

Kim Yoona

```
var variableA = "Kim Yuna";
console.log(variableA);
```

Kim Yuna

- ✓ var와 let이 같은 Level에서 정의되면 Error를 발생시킨다.

```
var variableA = "Kim Yoona";
console.log(variableA);

let variableA = "Kim Yuna";
console.log(variableA);
```

SyntaxError: Identifier 'variableA' has already been declared

- ✓ 단, 아래처럼 작성하면 Error가 발생하지 않는다.

```
var variableA = "Kim Yoona";
console.log(variableA);
```

Kim Yoona

```
if ( condition ) {
  let variableA = "Kim Yuna";
  console.log(variableA);
}
```

Kim Yuna

# const(상수)

- ☑ 기존 Javascript에서 지원되지 않던 상수를 const 키워드로 지원함.
- ☑ 초기화되지 않으면 Error를 발생시킴.

```
// 유효한 상수
const maxItems = 30;

// 문법에러 : 초기화 되지 않음
const name;

console.log(maxItems);
console.log(name);
```

SyntaxError: Missing initializer in const declaration

- ☑ let과 같은 Block-Level Scope 를 가진다.

# const(상수)로 객체 선언하기

- ✓ const는 상수 뿐만 아니라 객체를 생성/관리하기에 적합하다.
- ✓ const는 바인딩(초기화/할당)을 변경하도록 막는 것  
→ 바인딩 된 값의 변경을 막지 않는다.

```
// Const에 객체 리터럴 할당
const person = {
  name : "Seo Tae Ji"
};

// 객체의 값 변경
person.name = " And Boys";

// const에 새로운 객체를 재할당 하려할 경우 에러 발생!
person = {
  name : " And Boys"
};
```

TypeError: Assignment to constant variable.

## 반복문 내의 let 사용

- ✓ let을 사용할 경우 복잡한 수식 없이 아래처럼 사용 가능하다.

```
var numbers = [];

for ( let i = 0; i < 10; i++ ) {
  numbers.push(function() {
    console.log(i);
  });
}

numbers.forEach(function(f) {
  f();
});
```

0  
1  
...

- ✓ let은 값이 참조될 때마다 복사본을 전달한다.

# 반복문 내의 let 사용

- ✓ 반복문에서도 let을 사용할 수 있다.

```
const funcs = [];
const object = {
  a: true,
  b: true,
  c: true
};

for ( let key in object ) {
  funcs.push(function() {
    console.log(key, object[key]);
  });
}

funcs.forEach(function(f) {
  f();
});
```

```
a true
b true
c true
```

ECMAScript2015에 새롭게 추가된 문자열 함수들을 살펴보고 학습한다.

# 문자열

# 부분 문자열 식별하기

- ✓ 타 언어에 비해 부족했던 자바스크립트에 몇 가지 유용한 함수가 추가  
되

```
let message = "Hello world!";

console.log(message.startsWith("Hello")); // true
console.log(message.startsWith("hello")); // false
console.log(message.endsWith("!")); // true
console.log(message.includes("o")); // true
console.log(message.includes("0")); // false

console.log(message.startsWith("o", 4)); // true
console.log(message.endsWith("o", 8)); // true
console.log(message.includes("o", 8)); // false
```

- **String.startsWith("") :**
  - 문자열의 시작점에서 주어진 문자를 찾으면 true, 그렇지 않으면 false를 반환. (시작점을 지정할 수도 있다)
- **String.endsWith("") :**
  - 문자열의 끝에서 주어진 문자를 찾으면 true, 그렇지 않으면 false를 반환. (찾을 지점을 지정할 수도 있다)
- **String.includes("") :**
  - 문자열의 어느 곳이든 주어진 문자를 찾으면 true, 그렇지 않으면 false를 반환. (찾을 지점을 지정할 수도 있다)

# 문자 위치 알아내기

## ☒ 기타 유용한 문자열 함수

```
let message = "Hello world!";
console.log(message.indexOf("o"));           // 4
console.log(message.lastIndexOf("o"));        // 7
```

- ☒ `String.indexOf("")` : 문자열에서 주어진 문자의 위치를 반환. 없으면 -1을 반환
- ☒ `String.lastIndexOf("")` : 문자열에서 주어진 문자의 가장 마지막 위치를 반환. 없으면 -1을 반환

```
let message = " Hello world! ";
console.log(message);

message = message.trim();
console.log(message);
```

- ☒ `String.trim()` : 문자열에서 좌우 끝 공백을 모두 제거함.

# 문자 반복 시키기

## ✓ 문자열 반복하기

```
console.log("x".repeat(3));      // xxx
console.log("hello".repeat(2));   // hellohello
console.log("abc".repeat(4))    // abcabcaabcabc
```

- ✓ 원본 문자열을 주어진 횟수만큼 반복해 추가해주는 함수.
- ✓ 주로 편의를 위한 함수.
- ✓ 특히 텍스트/아이디(DB-PK)를 조작할 때 유용하게 사용될 수 있음.

```
let idx = "10"; // 0000010 으로 변경.
let idxFormat = "0000000";

idx = "0".repeat(idxFormat.length - idx.length) + idx;
// idx = "0".repeat( 7 - 2 ) + id;
console.log(idx);
```

# 템플릿 리터럴 사용

- ✓ Javascript에서 문자열을 이어 붙이는 방법은 복잡하다.

```
const year = 2018;
const month = 1;
const date = 16;

// 오늘은 2018년 1월 16일입니다.
let message = "오늘은 " + year + "년" + month + "월" + date + "일입니다."
console.log(message);
```

- ✓ 또한, 여러줄의 텍스트를 만든다면 더욱 그렇다.

```
const year = 2018;
const month = 1;
const date = 16;

// 오늘은 2018년 1월 16일입니다.
// 내일은 몇일 인가요?
let message = "오늘은 " + year + "년" + month + "월" + date + "일입니다.\n";
message += "내일은 몇일 인가요?";
console.log(message);
```

- ✓ 몇 줄 반복되다보면 매우 어지러워진다.

# 템플릿 리터럴 사용

- ✓ 템플릿 리터럴은 간단하게 문자열을 만들어 낼 수 있다.

```
const year = 2018;
const month = 1;
const date = 16;

// 오늘은 2018년 1월 16일입니다.
let message = `오늘은 ${year}년 ${month}월 ${date}일입니다.`;
console.log(message);
```

- ✓ 템플릿 리터럴은 백틱(`)을 사용해 표현한다.
- ✓ 백틱내에 변수를 조합하기 위해서 \${변수명} 을 사용한다.

```
const year = 2018;
const month = 1;
const date = 16;

// 오늘은 2018년 1월 16일입니다.
let message = `오늘은 ${year}년 ${month}_월 ${date}일입니다.`;
console.log(message);
```

- ✓ 위 처럼 잘못된 변수명 \${month\_} 을 사용하면 Reference Error를 발생시킨다.

# 템플릿 리터럴 사용

- ☑ 여러 줄의 텍스트를 만드려면 백틱 내에서 새로운 줄을 만들면 된다.

```
const year = 2018;
const month = 1;
const date = 16;

// 오늘은 2018년 1월 16일입니다.
// 내일은 몇일 인가요?
let message = `오늘은 ${year}년 ${month}월 ${date}일입니다.
내일은 몇일 인가요?`;
console.log(message);
```

- ☑ \${} 내에서 계산식도 사용할 수 있다.

```
let count = 10;
let price = 50;

let message = `${count} items cost ${count * price} USD`;
console.log(message);
```

10 items cost 500 USD

ECMAScript2015에서 변경된 함수들을 살펴보고 학습한다.

# 함수

# 함수

- ✓ ECMAScript의 가장 중요한 객체 중 하나.
- ✓ 함수를 파라미터로 혹은 객체로 저장하며, 다양하게 활용할 수 있음.
- ✓ ECMAScript 2015 이전 함수의 문제점.

- 파라미터를 제대로 전달하지 않더라도 정상적으로 실행이 된다!

```
function foo(bar) {  
    console.log(bar);  
}  
  
foo();  
foo("Bar");  
foo("Bar", "Foo");
```

undefined  
Bar  
Bar

- ECMAScript는 파라미터를 arguments라는 객체를 통해 전달한다.,.

```
function foo(bar) {  
    console.log(arguments);  
}  
  
foo();  
foo("Bar");  
foo("Bar", "Foo");
```

Arguments(0) []  
Arguments(1) ["Bar"]  
Arguments(2) ["Bar", "Foo"]

# 함수

- ✓ 이런 특징 때문에 발생했던 비정상적인 코드

```
function foo(bar) {  
    if ( bar == undefined ) {  
        bar = "Init Bar";  
    }  
    console.log(bar);  
}  
  
foo();  
foo("Bar");  
foo("Bar", "Foo");
```

Init Bar  
Bar  
Bar

- ✓ **Undefined** 체크 로직 때문에 복잡할 필요가 없는 코드가 복잡해지고 길어짐.
- ✓ ECMAScript 2015에서는 이런 불편함을 해소하기 위해 Default Parameter를 제공함.

# 함수 – Default Parameter

- 함수 선언할 때 파라미터의 기본값을 정의할 수 있다.

```
function foo(bar = "Init Bar") {  
    console.log(bar);  
}
```

```
foo();  
foo("Bar");  
foo("Bar", "Foo");
```

```
Init Bar  
Bar  
Bar
```

- 파라미터가 전달되지 않아 Undefined로 정의된다면, 자동으로 Default Parameter 의 값인 "Init Bar" 로 할당된다.
- 아래 코드도 가능하다.

```
function makeRequest(url, timeout = 3000, callback = function() { return "Basic CallBack"; }) {  
    console.log("Request URL ", url);  
    console.log("Timeout ", timeout);  
    console.log("Callback ", callback());  
    console.log("");  
}  
  
// URL만 전달  
makeRequest("url");  
  
// URL, Timeout 전달  
makeRequest("url", 1000);  
  
// URL, Timeout, Callback 전달  
makeRequest("url", 2000, function() {  
    return "hello";  
});
```

# 함수 – Default Parameter

- ✓ Default Parameter의 값으로 함수를 사용할 수도 있다.

```
function basicCallback() {  
    return function() {  
        return "Basic Callback";  
    }  
}  
  
function basicCallback2() {  
    return "Basic Callback2";  
}  
  
function makeRequest(url, timeout = 3000, callback = basicCallback2) {  
    console.log("Request URL ", url);  
    console.log("Timeout ", timeout);  
    console.log("Callback ", callback());  
    console.log("");  
}
```

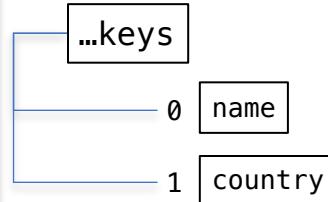
- ✓ 필요에 따라 아래처럼 사용할 수도 있다.

```
function makeRequest(url, timeout = 3000, callback = basicCallback()) {  
    console.log("Request URL ", url);  
    console.log("Timeout ", timeout);  
    console.log("Callback ", callback());  
    console.log("");  
}
```

# 함수 – 가변길이 파라미터

- ✓ 하나의 파라미터 변수에 여러 가지 값을 전달해, 배열처럼 사용할 수 있음.
  - arguments 객체를 대체하기 위해 설계됨.

```
function pick(object, ...keys) {  
  const result = {};  
  for ( let i in keys ) {  
    result[keys[i]] = object[keys[i]];  
  }  
  return result;  
}  
  
const object = {  
  name: 'Hong kildong',  
  city: 'Seoul',  
  country: 'South Korea'  
};  
const result = pick(object, "name", "country");  
console.log(result);
```



Object {name: "Hong kildong", country: "South Korea"}

- ✓ 가변길이 파라미터 뒤에는 다른 파라미터를 사용할 수 없다.

```
function pick(object, ...keys, next) {  
  ...  
}
```

- 가변길이 파라미터는 항상 마지막에 위치해야 한다.

# 함수 – Function

- ✓ 새로운 함수를 동적으로 생성하게 해준다.

```
const add = new Function("first", "second", "return first + second");
console.log(add(10, 50));
```

60

- ✓ Default Parameter / 가변길이 파라미터를 모두 사용할 수 있다.

- Default Parameter

```
const add = new Function("first", "second = first", "return first + second");
console.log(add(10, 50));
console.log(add(10));
```

60

20

- 가변길이 파라미터

```
const pickFirst = new Function("...numbers", "return numbers[0]");
console.log(pickFirst(50, 1, 10));
```

50

- ✓ 제약사항

- 외부의 function을 참조할 수 없다.

```
function add2(first, second) {
  return first + second;
}
const add = new Function("first", "second = first", "return add2(first + second)");
```

ReferenceError: add2 is not defined

# 함수 – 펼침(Spread) 연산자

- ☑ 배열을 단일 값으로 풀어해치는 연산자.

```
console.log("배열 출력 : ", [10, 20, 30]);
console.log("펼침 연산자로 배열 출력 : ", ...[10, 20, 30]);
```

```
배열 출력 : Array(3) [10, 20, 30]
펼침 연산자로 배열 출력 : 10 20 30
```

- ☑ 배열에 다른 배열을 추가할 때에도 유용하게 사용할 수 있다.

- 가변길이 파라미터를 다른 배열에 추가하려 할 때.

```
function push(array, ...args) {
  array.push(args);
}
```

```
const array = [10, 20];
push(array, 30, 50, 60);
console.log(array);
```

```
Array(3) [10, 20, Array(3)]
```

- 가변길이 파라미터를 펼침연산자를 이용해 배열에 추가하려 할 때

```
function push(array, ...args) {
  array.push(...args);
}
```

```
const array = [10, 20];
push(array, 30, 50, 60);
console.log(array);
```

```
Array(5) [10, 20, 30, 50, 60]
```

# 함수 – 화살표(Fat Arrow) 함수

- ✓ 함수를 화살표(`=>`)로 정의할 수 있는 새로운 방법
- ✓ 타 언어의 Lambda와 유사하다.

- `function` 키워드를 사용하지 않는다.
- `=>` 이후 중괄호(`{ 와 }`)의 여부에 따라 Return 유무가 결정된다.

- ✓ 일반적인 형태의 익명함수 (함수표현식)

```
const sum = function(first, second) {  
    return first + second;  
}  
console.log(sum(10, 20));
```

- ✓ 위 함수는 아래처럼 변경이 가능하다.

```
const sum = (first, second) => first + second;  
console.log(sum(10, 20));
```

# 함수 – 화살표(Fat Arrow) 함수

## ✓ 화살표 함수의 다양한 표현법

```
const fn = () => {  
  console.log("반환값이 없는 함수");  
};
```



```
const fn = function() {  
  console.log("반환값이 없는 함수");  
}
```

```
const fn = () => 10;
```



```
const fn = function() {  
  return 10;  
}
```

```
const fn = (value) => value * 2;
```



```
const fn = function(value) {  
  return value * 2;  
}
```

```
const fn = (...value) =>  
  Math.max(...value);
```



```
const fn = function(...value) {  
  return Math.max(...value);  
}
```

```
const fn = (first, second=10) =>  
  first + second;
```



```
const fn = function (first, second=10) {  
  return first + second;  
}
```

# 함수 – 화살표(Fat Arrow) 함수

- ✓ 화살표 함수로 즉시실행 함수를 만들수 있다.

- 일반적인 형태의 즉시 실행 함수

```
const person = (function(name) {  
    return {  
        getName: function() {  
            return name;  
        }  
    };  
})(“Seo Tae Ji”);  
  
console.log(person.getName());
```

- 화살표 함수 형태의 즉시 실행 함수

```
const person = ((name) => {  
    return {  
        getName: () => name  
    };  
})(“Seo Tae Ji”);  
  
console.log(person.getName());
```

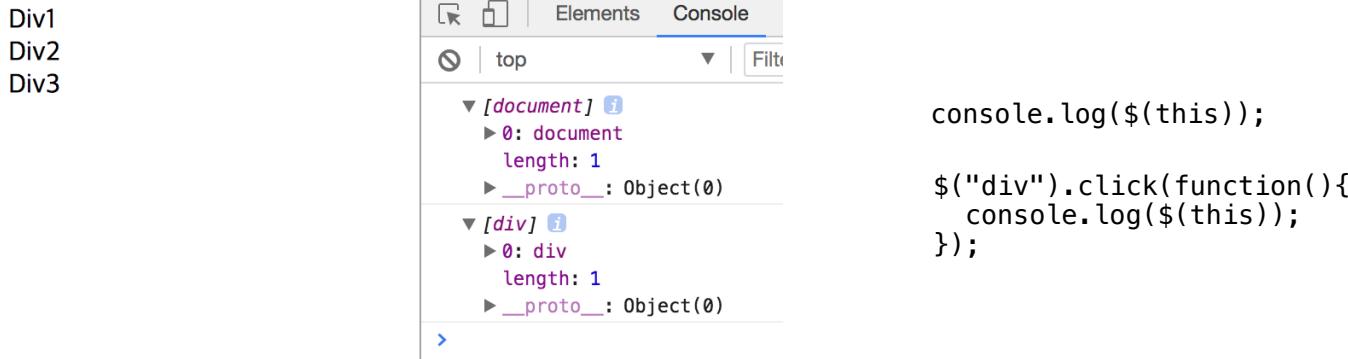
- 자질구레한 코드들이 사라진다.

# 함수 – 화살표(Fat Arrow) 함수

- jQuery에서 화살표 함수를 이용할 때 주의할 점.

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
    <script type="text/javascript">
      $(().ready(function() {
        console.log($(this));
        $("div").click(function(){
          console.log($(this));
        });
      });
    </script>
  </head>
  <body>
    <div>Div1</div>
    <div>Div2</div>
    <div>Div3</div>
  </body>
</html>
```

- jQuery에서는 이벤트가 일어난 DOM이 this 객체로 전달된다.

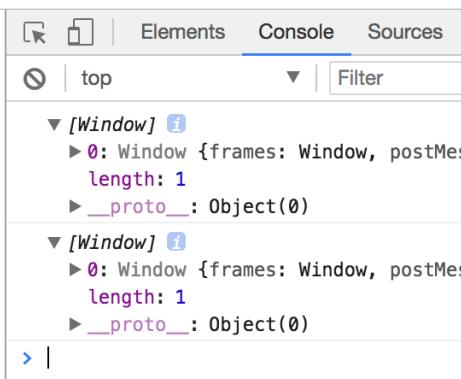


# 함수 – 화살표(Fat Arrow) 함수

- jQuery에서 화살표 함수를 이용할 때 주의할 점.

```
$(().ready(() => {
  console.log($(this));
  $("div").click( () => {
    console.log($(this));
  });
});
```

- 화살표 함수로 바꾸었을 때, `$(this)`는 window가 된다.



The screenshot shows the browser's developer tools with the 'Console' tab selected. The output area displays the following code and its execution results:

```
console.log($(this));  
$("div").click( () => {  
  console.log($(this));  
});
```

The output in the console shows two entries, each representing a Window object. The first entry is the global window object, and the second is a child window object created by the click event handler. Both objects have properties like `frames`, `postMessage`, and `length` set to 1, and a `__proto__` property.

# 함수 – 화살표(Fat Arrow) 함수

## ✓ 왜?

- o this는 인접한 function을 기준으로 만들어지기 때문.

## ✓ 화살표 함수는 this 객체를 가지지 못한다.

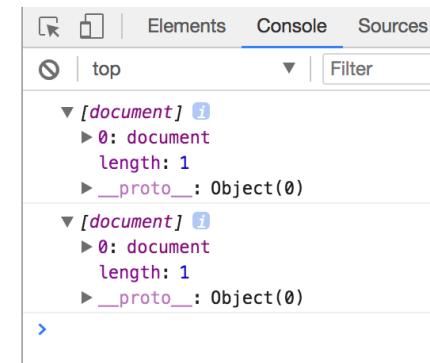
- o 인접한 function이 있을 경우, 그 function의 this를 사용하게 됨.

## ✓ 따라서 아래와 같은 결과가 나타날 수도 있다.

```
$(document).ready(function() {
  console.log(this);
  $("div").click( () => {
    console.log(this);
  });
});
```



Div1  
Div2  
Div3



```
top
[document]
0: document
length: 1
__proto__: Object(0)

[document]
0: document
length: 1
__proto__: Object(0)
```

```
console.log(this);
```

```
$("div").click( () => {
  console.log(this);
});
```

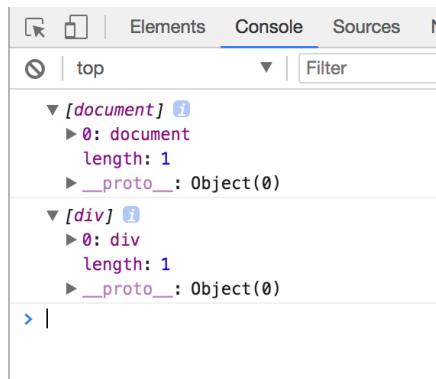
# 함수 – 화살표(Fat Arrow) 함수

- jQuery에서 화살표 함수를 사용하려면...

- Event 객체의 currentTarget 을 가져와야 한다.

```
$(document).ready(function() {
  console.log(this);
  $("div").on("click", (e) => {
    const $this = $(e.currentTarget);
    console.log($this);
  });
});
```

Div1  
Div2  
Div3



```
console.log(this);

$("div").on("click", (e) => {
  const $this = $(e.currentTarget);
  console.log($this);
});
```

ECMAScript2015에서 확장된 함수, 프로토타입, 객체 리터럴을 살펴보고 학습한다.

# 확장된 객체

# 프로퍼티 생략 가능

## ✓ 객체 리터럴을 만들 때 사용되던 일반적인 방법

```
function createPerson(name, age) {  
    return {  
        name: name,  
        age: age  
    };  
}  
  
const person = createPerson("James Dean", 30);
```

- 거의 모든 코드에서 프로퍼티의 이름과 값이 담겨있는 변수의 이름이 동일함.
- 객체의 프로퍼티에 접근하려면 **객체.프로퍼티명 혹은 객체[“프로퍼티명”]**

## ✓ ECMAScript2015에서 프로퍼티 명과 변수명이 같을 경우 프로퍼티를 생략할 수 있도록 개선됨.

```
function createPerson(name, age) {  
    return {  
        name,  
        age  
    };  
}  
  
const person = createPerson("James Dean", 30);
```

# 간결해진 메소드 정의 방법

- ☑ 객체 리터럴내 function을 정의하는 방법도 변경됨.

```
function createPerson(name, age) {  
  return {  
    name,  
    age,  
    sayName: function() {  
      console.log(this.name);  
    }  
  };  
  
  const person = createPerson("James Dean", 30);  
  person.sayName();
```

- ☑ function() 키워드가 생략되면서 간결하게 메소드를 정의할 수 있다.

```
function createPerson(name, age) {  
  return {  
    name,  
    age,  
    sayName() {  
      console.log(this.name);  
    }  
  };  
  
  const person = createPerson("James Dean", 30);  
  person.sayName();
```

# 객체 복사(Mixin)

## ☒ ECMAScript5 이전에 객체를 복사하는 방법

```
const supplier = {  
    name: "Michael",  
    city: "Seoul",  
    sayName() {  
        console.log(this.name);  
    }  
};  
  
const receiver = {  
    address: "Seocho",  
    city: "Seoul"  
};  
  
function mixin(receiver, supplier) {  
    Object.keys(supplier).forEach(function(key) {  
        receiver[key] = supplier[key];  
    });  
}  
  
mixin(receiver, supplier);  
console.log(supplier);  
console.log(receiver);  
receiver.sayName();
```

- Mixin 패턴을 이용해 얇은 복사(값만 복사) 방법이 많이 사용됨.

# 객체 복사(Object.assign)

- ☒ Mixin 패턴이 ECMAScript6에서 지원.

- 아주 빈번하게 사용되는 패턴 → Script에서 자체 지원되도록 추가됨.

```
const supplier = {  
  name: "Michael",  
  city: "Seoul",  
  sayName() {  
    console.log(this.name);  
  }  
};  
  
const receiver = {  
  address: "Seocho",  
  city: "Seoul"  
};  
  
Object.assign(receiver, supplier);  
console.log(supplier);  
console.log(receiver);  
receiver.sayName();
```

- Mixin, Object.assign을 사용하면, 중첩되는 Key/Value는 Supplier의 것으로 덮어쓰게 됨.

## 기준의 프로토타입

# 프로토타입?

- 한 객체가 만들어지기 위해 필요한 객체의 모태.
  - 클래스가 존재하지 않는 자바 스크립트에서 객체 지향 프로그래밍을 가능케 해주는 객체의 원형
  - 확장 및 객체의 재사용을 가능하게 해준다.

Prototype = Prototype Object + Prototype Link

- Prototype Object : Function의 객체가 가지는 속성
  - Prototype Link : Function 객체를 만들 때 사용된 객체의 원형
    - 일반 Function 객체일 경우 Object 가 Prototype Link가 된다. (`__proto__`)

## 아래와 같은 코드의 결과로 Prototype 이해하기

```
function Person() {}  
Person.prototype.x = "10";  
  
const citizen = new Person();  
console.log(citizen);  
console.log(Person.prototype);  
console.log(citizen.x);
```

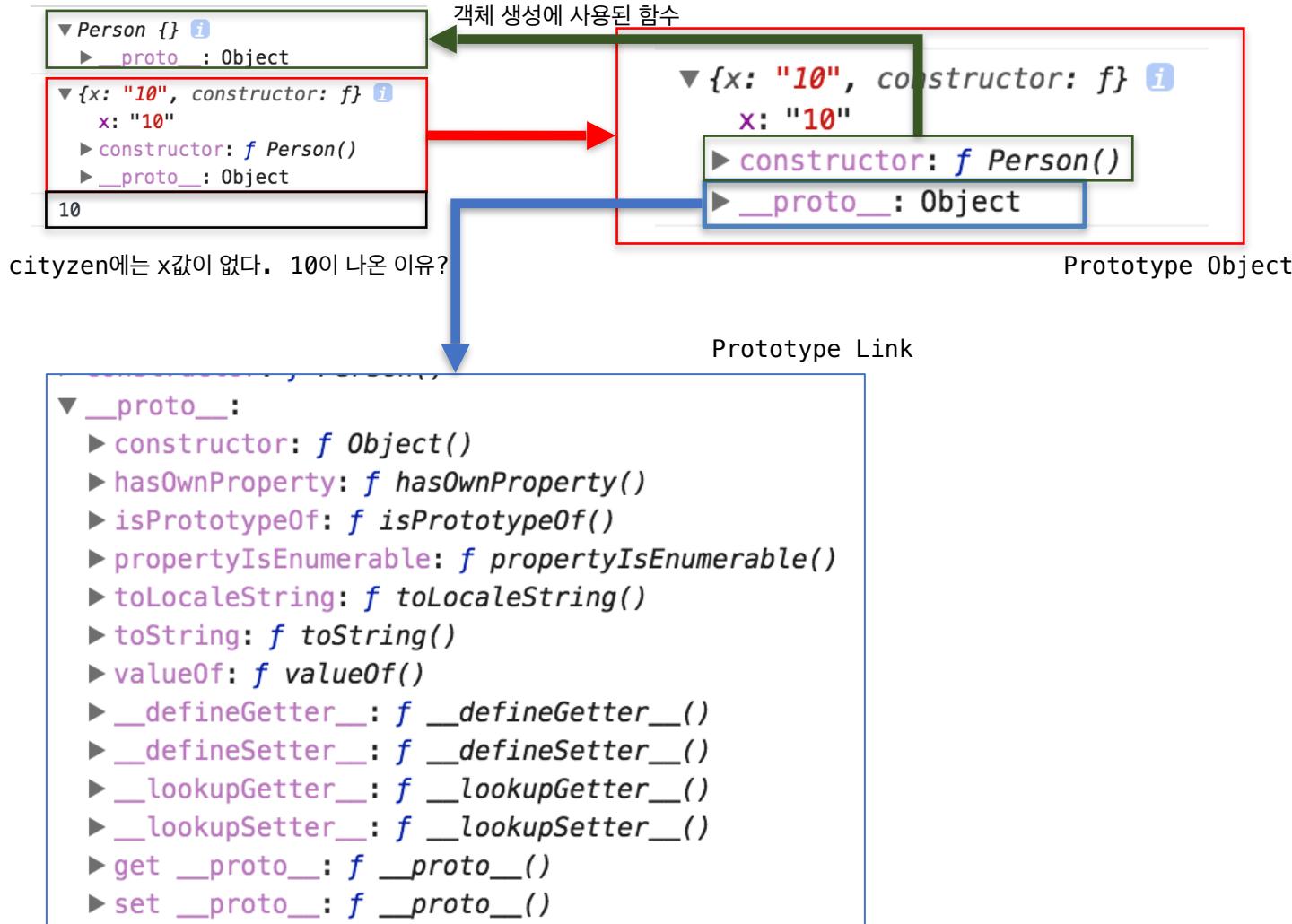
```
▼ Person {} ⓘ           console.log(citizen);
  ► __proto__: Object

▼ {x: "10", constructor: f} ⓘ   console.log(Person.prototype);
  x: "10"
  ► constructor: f Person()
  ► __proto__: Object

10          console.log(citizen.x);
```

# 기존의 프로토타입

## ✓ Prototype Object / Prototype Link



# 기존의 프로토타입

## ✓ 조금 더 상세하게 사용해보기

```
function Person(firstName) {
  this.firstName = firstName;
}

Person.prototype.walk = function(){
  console.log("I am walking!");
};

Person.prototype.sayHello = function(){
  console.log("Hello, I'm " + this.firstName);
};

const citizen = new Person("Michael Kim");
console.log(citizen);
console.log(Person.prototype);
citizen.walk();
citizen.sayHello();
```

```
▼ Person {firstName: "Min Chang Jang"} ⓘ
  firstName: "Min Chang Jang"
  ► __proto__: Object
▼ {walk: f, sayHello: f, constructor: f} ⓘ
  ► sayHello: f ()
  ► walk: f ()
  ► constructor: f Person(firstName)
  ► __proto__: Object
I am walking!
Hello, I'm Min Chang Jang
```

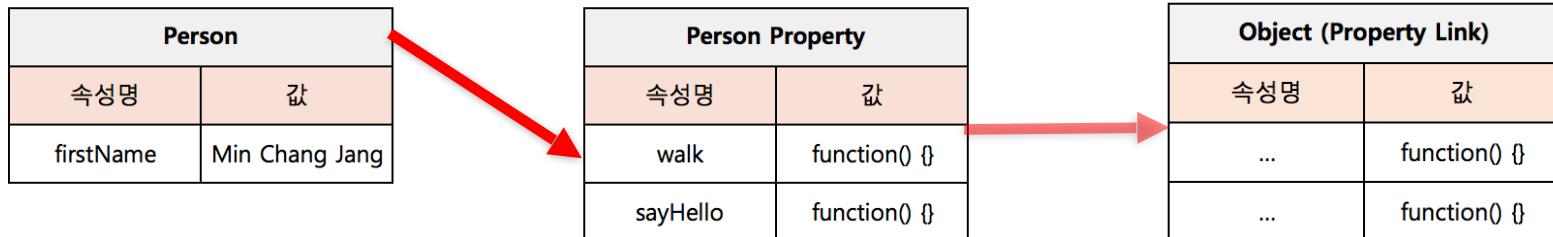
- ✓ citizen에는 walk(), sayHello() 메소드가 없어도 실행이 되는 이유는?
- Property Chaining 때문.

# 기존의 프로토타입

## ✓ Property Chaining

```
function Person(firstName) {  
    this.firstName = firstName;  
}  
  
Person.prototype.walk = function(){  
    console.log("I am walking!");  
};  
Person.prototype.sayHello = function(){  
    console.log("Hello, I'm " + this.firstName);  
};  
  
const citizen = new Person("Michael Kim");  
citizen.walk();  
citizen.sayHello();
```

- ✓ Function 또는 Property 를 먼저 객체(Function)에서 찾고,  
없으면 Prototype Object에서 찾고, 없다면 Prototype Link에서 찾는다.



# 기준의 프로토타입

## ✓ 프로토타입을 이용한 객체 상속( Object.create(Super Prototype) )

```
function Person(firstName) {
  this.firstName = firstName;
}
Person.prototype.walk = function(){
  console.log("I am walking!");
};
Person.prototype.sayHello = function(){
  console.log("Hello, I'm " + this.firstName);
};

const citizen = new Person("Michael Kim");
console.log(citizen);
console.log(Person.prototype);
citizen.walk();
citizen.sayHello();

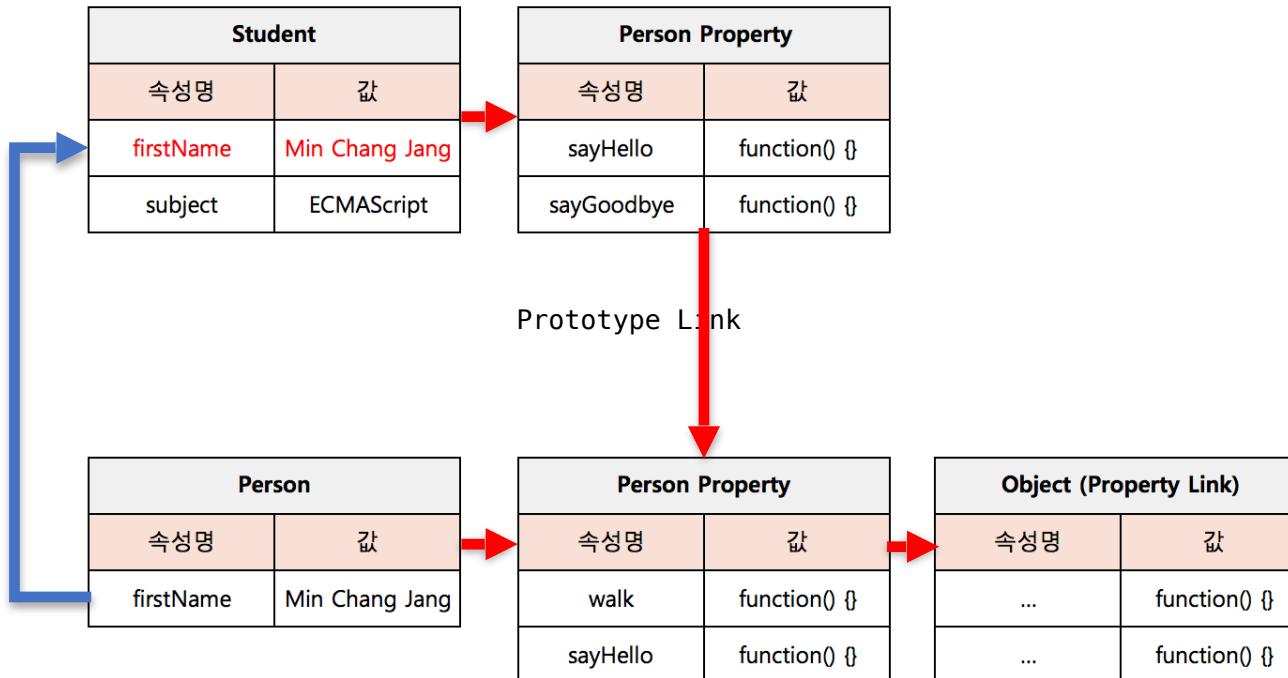
function Student(firstName, subject) {
  Person.call(this, firstName);
  this.subject = subject
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;
Student.prototype.sayHello = function(){
  console.log("Hello, I'm " + this.firstName + ". I'm studying " + this.subject + ".");
};
Student.prototype.sayGoodBye = function(){
  console.log("Goodbye!");
};

const student = new Student("Michael Kim", "ECMAScript");
console.log(student);
console.log(Student.prototype);
student.walk();
student.sayHello();
student.sayGoodBye();
```

# 기존의 프로토타입

✓ 프로토타입을 이용한 객체 상속( Object.create(Super Prototype) )



# ECMAScript 2015 프로토타입

---

## ✓ Prototype의 한계점.

- Prototype은 function 객체에만 존재함.
- 따라서 Non-Function 객체에서는 Prototype을 활용한 상속/확장이 불가능함.

## ✓ ECMAScript 2015 이후부터 객체 리터럴에도 프로토타입을 변경할 수 있도록 지원함.

- 단, 상속의 개념이 아닌 단순 변경.

## ✓ Object.setPrototypeOf();

- 객체 리터럴의 프로토타입을 변경할 수 있도록 하는 명령.

# ECMAScript 2015 프로토타입

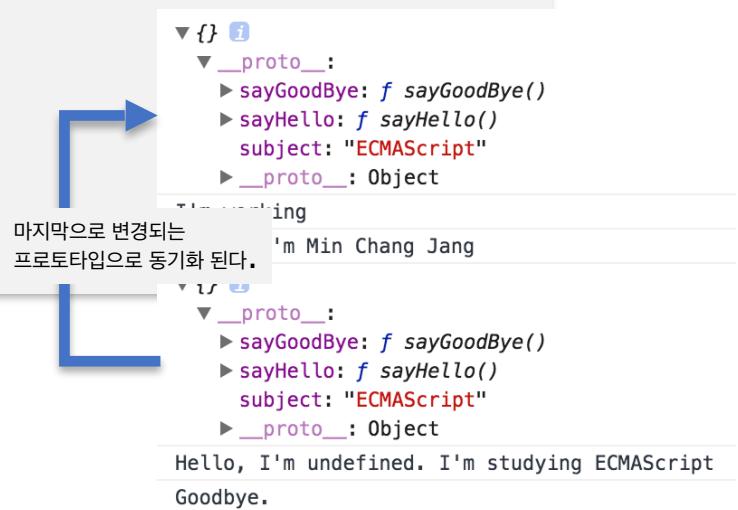
## ✓ 프로토타입 변경 예제

```
const person = {
  firstName: "Michael Kim",
  work() {
    console.log("I'm working");
  },
  sayHello() {
    console.log("Hello, I'm " + this.firstName);
  }
}

const student = {
  subject: "ECMAScript",
  sayHello() {
    console.log("Hello, I'm " + this.firstName + ". I'm studying " + this.subject);
  },
  sayGoodBye() {
    console.log("Goodbye.");
  }
}

const friend = Object.create(person);
console.log(friend);
friend.work();
friend.sayHello();

Object.setPrototypeOf(friend, student);
console.log(friend);
friend.sayHello();
friend.sayGoodBye();
```



# ECMAScript 2015 프로토타입

## ✓ Super Prototype 참조를 통한 쉬운 Prototype 접근

- Object.create() 와 달리, 객체를 상속하여 사용이 가능하다.

```
const person = {  
    firstName: "Michael Kim",  
    work() {  
        console.log("I'm working");  
    },  
    sayHello() {  
        console.log("Hello, I'm " + this.firstName);  
    }  
}  
const student = {  
    subject: "ECMAScript",  
    sayHello() {  
        super.sayHello();  
        console.log("Hello, I'm " + this.firstName + ". I'm studying " + this.subject);  
    },  
    sayGoodBye() {  
        console.log("Goodbye.");  
    }  
}  
  
Object.setPrototypeOf(student, person);  
  
console.log(person);  
person.work();  
person.sayHello();  
  
console.log(student);  
student.work();  
student.sayHello();  
student.sayGoodBye();  
  
const friend = Object.create(student);  
console.log(friend);  
friend.work();  
friend.sayHello();  
friend.sayGoodBye();
```

# ECMAScript 2015 프로토타입

## ✓ Super Prototype 참조를 통한 쉬운 Prototype 접근

- Object.create() 와 달리, 객체를 상속하여 사용이 가능하다.

```
▼ {firstName: "Min Chang Jang", work: f, sayHello: f} ⓘ  
  firstName: "Min Chang Jang"  
  ▶ sayHello: f sayHello()  
  ▶ work: f work()  
  ▶ __proto__: Object
```

I'm working

Hello, I'm Min Chang Jang

```
▼ {subject: "ECMAScript", sayHello: f, sayGoodBye: f} ⓘ
```

```
  ▶ sayGoodBye: f sayGoodBye()
```

```
  ▶ sayHello: f sayHello()
```

```
  subject: "ECMAScript"
```

```
  ▶ __proto__:
```

```
    firstName: "Min Chang Jang"
```

```
    ▶ sayHello: f sayHello()
```

```
    ▶ work: f work()
```

```
    ▶ __proto__: Object
```

I'm working

Hello, I'm Min Chang Jang

Hello, I'm Min Chang Jang. I'm studying ECMAScript

Goodbye.

```
▼ {} ⓘ
```

```
  ▶ __proto__:
```

```
    ▶ sayGoodBye: f sayGoodBye()
```

```
    ▶ sayHello: f sayHello()
```

```
    subject: "ECMAScript"
```

```
    ▶ __proto__:
```

```
      firstName: "Min Chang Jang"
```

```
      ▶ sayHello: f sayHello()
```

```
      ▶ work: f work()
```

```
      ▶ __proto__: Object
```

I'm working

Hello, I'm Min Chang Jang

Hello, I'm Min Chang Jang. I'm studying ECMAScript

Goodbye.

ECMAScript2015에서 추가된 구조 분해(해체) 방법을 살펴보고 학습한다.

# 구조 분해 (해체)

# 구조 분해(해체)

- 객체 리터럴이나 배열을 해체해 최소단위의 변수로 저장하는 방법
- 기존에 객체리터럴이나 배열에서 데이터를 가져오는 방법

객체 리터럴에서 기존의 방법으로 추출

```
let options = {  
    repeat: true,  
    save: false  
};  
  
// 추출  
let repeat = options.repeat;  
let save = options.save;
```

배열에서 기존의 방법으로 추출

```
let colors = ["red", "green", "blue"];  
  
let red = colors[0];  
let green = colors[1];  
let blue = colors[2];  
  
console.log(red, green, blue);
```

객체 리터럴에서 구조 분해해 추출

```
let options = {  
    repeat: true,  
    save: false  
};  
  
// 추출  
let {repeat, save} = options;  
  
console.log(repeat);  
console.log(save);
```

배열에서 구조 분해해 추출

```
let colors = ["red", "green", "blue"];  
let [red, green, blue] = colors;  
  
console.log(red, green, blue);
```

## 구조 분해(해체)

- ✓ 선언된 변수에 구조 분해후 할당

```
let node = {  
  type: "Identifier",  
  name: "foo"  
};  
  
let type = "Literal"  
let name = 5;  
  
console.log(type, name);           Literal 5  
  
({type, name} = node);  
  
console.log(type, name);           Identifier foo
```

- ✓ 미리 선언된 변수에 구조 분해 후 재 할당문에는 반드시 괄호가 필요하다.

# 구조 분해(해체)

## ✓ 기본 값 할당

```
let node = {  
  type: "Identifier",  
  name: "foo"  
};  
  
let {type, name, value} = node;  
  
console.log(type, name, value);
```

Identifier foo undefined

- ✓ 구조 분해시, 지정된 값이 없을 경우 `undefined`로 지정됨.
- ✓ 이를 막기 위해 기본값을 할당할 수 있다.

```
let node = {  
  type: "Identifier",  
  name: "foo"  
};  
  
let {type, name, value = true} = node;  
  
console.log(type, name, value);
```

Identifier foo true

# 구조 분해(해체)

## ✓ 이름이 다른 변수에 할당하기

```
let node = {  
    type: "Identifier",  
    name: "foo"  
};  
  
let {type: localType, name: localName} = node;  
  
console.log(localType, localName);           Identifier foo
```

- 구조 분해는 객체 리터럴 내의 이름이 같은 변수에 할당된다.
- 다른 이름의 변수에 할당하고자 한다면, 아래와 같은 패턴으로 작성한다.

```
let {프로퍼티 명: 변수명, ...} = 객체 리터럴;
```

# 구조 분해(해체)

## ✓ 중첩 구조의 객체 분해

```
let node = {  
  type: "Identifier",  
  name: "foo",  
  loc: {  
    start: {  
      line: 1,  
      column: 1  
    },  
    end: {  
      line: 1,  
      column: 4  
    }  
  }  
};  
  
let { loc: {start} } = node;  
  
console.log(start.line, start.column);
```

1 1

- 복잡한 중첩 구조의 객체를 분해할 때는 리터럴 문법을 사용할 수 있다.

```
let {프로퍼티 명: { 프로퍼티 명 }, ...} = 객체 리터럴;
```

- 최종 중괄호의 프로퍼티명이 지역변수의 이름이 된다.

# 구조 분해(해체)

## ✓ 배열의 구조 분해 할당

```
let colors = ["red", "green", "blue"];

let firstColor = "black";
let secondColor = "purple";

console.log(firstColor, secondColor);           black purple

[firstColor, secondColor] = colors;

console.log(firstColor, secondColor);           red green
```

- 배열의 구조 분해시 객체의 분해처럼 괄호를 필요로하지 않는다.
- 구조 분해시 모든 배열의 개수를 맞출 필요가 없다. (단, 순서는 맞추어야 한다)

## ✓ 중첩된 배열을 분해할 때 대괄호를 한번 더 사용한다.

```
let colors = ["red", ["green", "lightGreen"], "blue"];

let firstColor = "black";
let secondColor = "purple";

console.log(firstColor, secondColor);

[firstColor, [secondColor1, secondColor2], blue] = colors;

console.log(firstColor, secondColor1, secondColor2, blue);
```

# 구조 분해(해체)

## ✓ 구조분해와 나머지 연산자를 이용해 배열 복사하기

```
let colors = ["red", "green", "blue"];

let [...colonedColors] = colors;
colonedColors.push("cyan");

console.log(colors);
console.log(colonedColors);
```

```
red  green  blue
red  green  blue  cyan
```

## 혼합된 구조 분해(해체)

- ✓ 객체 구조 분해와 배열 구조 분해를 함께 사용해 복잡한 표현식 만들기

```
let node = {  
    type: "Identifier",  
    name: "foo",  
    loc: {  
        start: {  
            line: 1,  
            column: 1  
        },  
        end: {  
            line: 1,  
            column: 4  
        }  
    },  
    range: [0, 3]  
};  
  
let {  
    loc: {start},  
    range: [ startIndex, endIndex ]  
} = node;  
  
console.log(start.line, start.column);  
console.log(startIndex, endIndex);
```

## 파라미터 구조 분해(해체)

- ☑ 객체를 파라미터로 보낼 때 function의 파라미터로 구조 분해 할 수 있다.

```
function setAttribute(name, {url, method}) {
  console.log("name", name);
  console.log("url", url);
  console.log("method", method);
}

setAttribute("searchForm", {
  url: "http://localhost",
  method: "post"
});
```

- ☑ 혹은 기본값을 이용해 아래처럼 사용할 수도 있다.

```
function setAttribute(name, {
  url = "http://localhost",
  method = "post"
} = {}) {
  console.log("name", name);
  console.log("url", url);
  console.log("method", method);
}

setAttribute("searchForm");
```

# **클래스**

ECMAScript 2015에 새롭게 등장한 클래스를 학습한다.

## 유사 클래스와 클래스

---

- ✓ ECMAScript 2015 이전에는 Class를 지원하지 않음.  
→ 개발자들이 유사 클래스 형태로 만들어 사용함.
- ✓ 다른 Javascript 라이브러리들이 유사클래스 형태로 사용,  
→ ECMAScript 2015에서 정식으로 클래스를 지원.

# ECMAScript 2015이전의 유사 클래스

- Function을 이용해 유사 클래스를 생성.  
→ new 키워드를 사용해 객체를 생성함.  
→ prototype을 이용해 동적으로 메소드를 할당함.

```
function PersonType(name) {  
    this.name = name;  
}  
  
PersonType.prototype.sayName = function() {  
    console.log("My name is " + this.name);  
}  
  
const person = new PersonType("Michael");  
console.log(person.name);  
person.sayName();
```

Michael  
My name is Michael

- 클래스를 흉내내는 많은 라이브러리들이 이 패턴을 바탕으로 제작함.

# ECMAScript 2015의 클래스

## ☒ Class 키워드를 이용해 클래스를 선언함.

- 클래스를 만드는 기본 바탕은 앞선 function.
- 생성자, 메소드를 모두 지원한다.

```
class PersonClass {  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayName() {  
        console.log("My name is " + this.name );  
    }  
}  
  
const person = new PersonClass("Michael");  
console.log(person.name);  
person.sayName();
```

Michael  
My name is Michael

## ☒ 객체 리터럴과 문법이 유사함.

- 클래스의 요소들 사이에 콤마(,)가 필요 없음

# 클래스의 특징

---

## ✓ ECMAScript 2015 클래스의 특징

- 1. 함수 선언과 달리 클래스 선언은 호이스팅 되지 않는다.
- 2. 클래스 선언 내의 모든 코드는 엄격 모드인 strict 모드에서 실행된다.
- 3. 모든 메소드는 외부에서 열거(출력)할 수 없다.
- 4. new 없이 클래스 생성자를 호출할 수 없다.
  - new 없이 객체를 생성할 수 없다.

# 정적 멤버(Static Member) 생성하기

- ✓ ECMAScript 2015 이전의 유사 클래스에서 정적 멤버를 추가하기

```
function PersonType(name) {  
    this.name = name;  
}  
  
PersonType.create = function(name) {  
    return new PersonType(name);  
};  
  
PersonType.prototype.sayName = function() {  
    console.log("My name is " + this.name);  
};  
  
var person = new PersonType.create("Michael");  
console.log(person.name);  
person.sayName();
```

Michael  
My name is Michael

- Function 을 이용한 유사 클래스에서 정적 멤버를 추가하기 위해서는 PersonType에 멤버를 추가 함으로써 정의할 수 있다.
- 반면, Class를 이용한 클래스에서 정적 멤버를 추가하기 위해서는 static 키워드만 사용하면 된다.

# 정적 멤버(Static Member) 생성하기

## ☒ ECMAScript 2015의 클래스에서 정적 멤버를 추가하기

```
class PersonClass {  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayName() {  
        console.log("My name is " + this.name);  
    }  
  
    static create(name) {  
        return new PersonClass(name);  
    }  
}  
  
const person = PersonClass.create("Michael");  
console.log(person.name);  
person.sayName();
```

Michael  
My name is Michael

- 생성자(constructor)에는 static을 정의할 수 없다.
- 정적 멤버는 반드시 클래스에서만 접근할 수 있다.
  - 객체에서 정적 멤버에 접근하면 에러가 발생한다.

# Get / Set

✓ Class의 멤버에 접근할 수 있도록 get/set 키워드를 제공한다.

- 멤버에 직접 접근하는 방법보다 get/set을 통해 접근하는 방법이 안정적.
- 멤버의 이름과 get/set의 이름이 같을 경우 “무한반복”이 발생할 수 있다.

```
class PersonClass {  
    constructor(name) {  
        this._name = name;  
    }  
  
    get name() {  
        console.log("Getter!");  
        return this._name;  
    }  
    set name(name) {  
        console.log("Setter!");  
        this._name = name;  
    }  
  
    sayName() {  
        console.log("My name is " + this._name);  
    }  
    static create(name) {  
        return new PersonClass(name);  
    }  
}  
  
const person = PersonClass.create("Michael");  
person.name = "Tae Ji";  
console.log(person.name);  
person.sayName();
```

Tae Ji  
My name is Tae Ji

# 파생 클래스와 상속

- ✓ Extends 키워드를 이용해 클래스 상속이 가능하다.

```
class PersonType {  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayName() {  
        console.log("My name is " + this.name);  
    }  
}  
  
class Student extends PersonType {  
    constructor(name, schoolName) {  
        super(name);  
        this.schoolName = schoolName;  
    }  
  
    saySchoolName() {  
        console.log("I'm going to " + this.schoolName + " school");  
    }  
}  
  
const student = new Student("Michael", "None");  
console.log(student.name, student.schoolName);  
student.sayName();  
student.saySchoolName();
```

- ✓ 상속을 받은 클래스에서 상속한 클래스의 멤버를 사용할 수 있다.

# 파생 클래스와 상속

- ✓ Super 키워드로 통해 상속한 클래스의 멤버에 접근할 수 있다.

```
class Student extends PersonType {  
    constructor(name, schoolName) {  
        super(name);  
        this.schoolName = schoolName;  
    }  
  
    saySchoolName() {  
        console.log("I'm going to " + this.schoolName + " school");  
    }  
  
    sayName() {  
        super.sayName();  
        this.saySchoolName();  
    }  
}  
  
const student = new Student("Michael", "None");  
console.log(student.name, student.schoolName);  
student.saySchoolName();
```

- ✓ 파생 클래스에서 상속한 클래스의 멤버를 재 정의할 경우, 파생 클래스의 멤버를 사용한다. (오버라이딩)

# **프로미스와 비동기 프로그래밍**

# Javascript의 비동기 프로그래밍

## ✓ Javascript의 개발 목적

- 사용자 웹 페이지에서 사용될 목적
- 주된 기능은 사용자의 반응하기.
  - 마우스 이벤트(클릭, 이동 등)
  - 키보드 입력(keyup, keydown)

## ✓ Javascript는 애초부터 비동기에 반응하기 위한 목적으로 개발됨.

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script type="text/javascript">
      window.onload = function() {
        var btn = document.querySelector("#btn");
        btn.onclick = function(event) {
          alert("클릭!");
        }
      }
    </script>
  </head>
  <body>
    <input type="button" id="btn" value="클릭" />
  </body>
</html>
```

주로 함수표현식이나 콜백으로  
비동기를 처리한다.

또한, 처리 결과에 따라 다른  
방법들을 제공하게 됨에 따라  
코드가 복잡해지기 시작한다.

# Javascript의 비동기 프로그래밍

## ✓ 자바스크립트에서의 확정되지 않은 상태에 대한 처리 방법

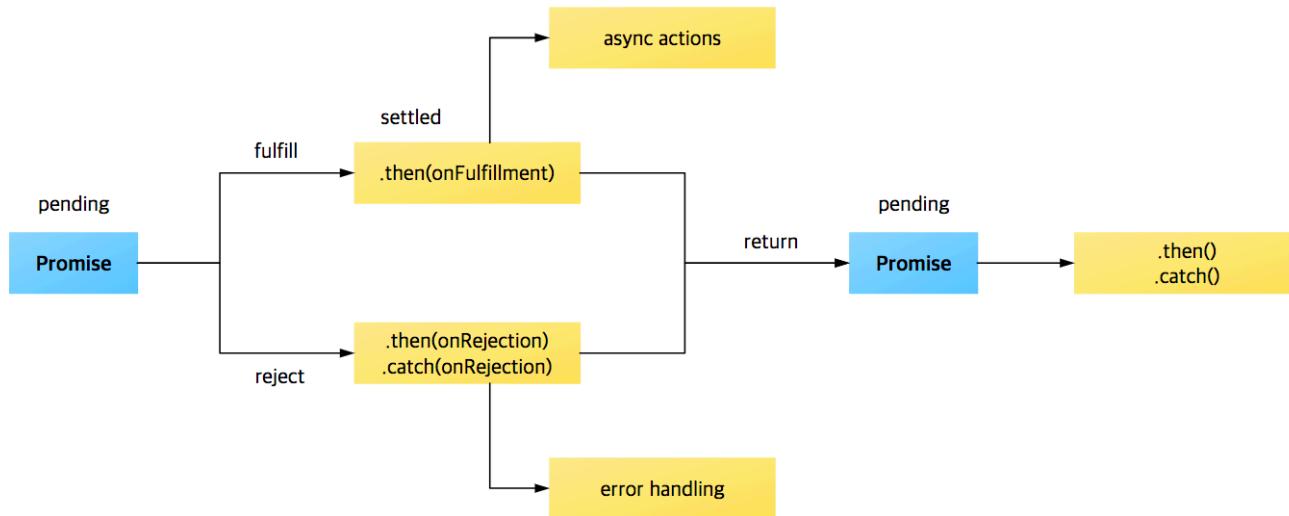
```
var btn = document.querySelector("#btn");
btn.onclick = function(event) {
    validateName(function() {
        alert("이름을 입력하세요!");
    }, function () {
        alert("성공!");
    });
}

function validateName(error, success) {
    var name = document.querySelector("#name");
    if ( name.value == "" ) {
        error();
    }
    else {
        success();
    }
}
```

- ✓ 콜백이 늘어나 자칫 콜백지옥이 빠지기 쉽다.
- ✓ 프로미스는 이런 처리에 대한 쉬운 해결책을 제공한다.

# 프로미스

- ☑ 프로미스는 비동기 처리 결과가 확정되지 않은 이벤트에 대한 처리 방법을 제공한다.
- ☑ 이것을 위해 프로미스는 “보류(pending)”, “성공(fulfilled)”, “실패(rejected)” 상태를 제공한다.
- ☑ Promise의 상태 변화



# 프로미스

- ✓ 프로미스가 제공하는 상태에 따라 개발자는 .then() 혹은 .catch() 메소드를 제공한다.

```
<script type="text/javascript">
window.onload = function() {
    let btn = document.querySelector("#btn");
    btn.onclick = function(event) {
        let promise = validate("#name");

        promise.then(function(element) { // 성공
            alert(element.id + " 입력되었습니다.");
        }).catch(function(value) { // 실패
            alert(value);
        });
    }
}

function validate(selector) {
    return new Promise(function(resolve, reject) {
        var element = document.querySelector(selector);
        if (element.value == "") {
            reject(element.dataset.error); // 실패
        } else {
            resolve(element); // 성공
        }
    });
}
</script>
...
<input type="text" id="name" data-error="이름을 입력하세요." placeholder="이름을 입력하세요." />
<input type="button" id="btn" value="클릭" />
```

```
btn.onclick = function(event) {
    let promise = validate("#name");

    promise.then(function(element) { // 성공
        alert(element.id + " 입력되었습니다.");
    }, function(value) { // 실패
        alert(value);
    });
}
```

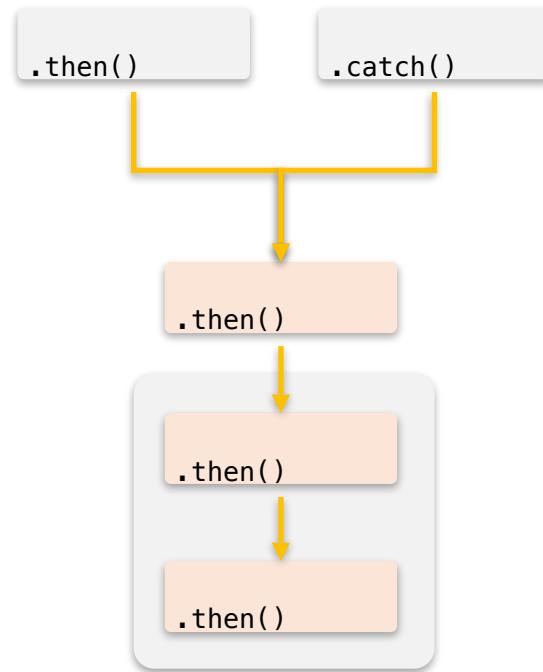
.then()은 두개의 파라미터를 가진다.  
.then(성공, 실패);

# 프로미스 연결하기

- ✓ 프로미스는 상태에 따른 처리가 주된 목적이지만, 프로미스 내에서 다른 프로미스를 만들어 비동기 처리를 연속으로 처리할 수 있도록 한다.
- ✓ `.then()`, `.catch()`는 수행될 때마다 다른 프로미스를 만들어 반환한다.

```
let btn = document.querySelector("#btn");
btn.onclick = function(event) {
  let promise = validate("#name");
  promise.then(function(element) {
    alert(element.id + " 입력되었습니다.");
  }).catch(function(value) {
    alert(value);
  }).then(function() {
    alert("다음 프로미스!");
  });
}

function validate(selector) {
  return new Promise(function(resolve, reject) {
    var element = document.querySelector(selector);
    if (element.value == "") {
      reject(element.dataset.error);
    } else {
      resolve(element);
    }
  });
}
```

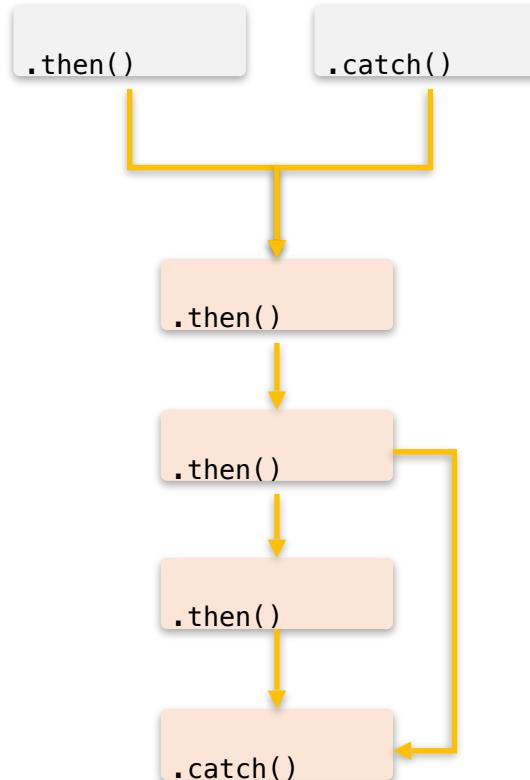


Then이 계속 이어지게 할 수도 있다.

# 연결된 프로미스에 에러 처리하기

- ✓ 프로미스 중간에 Error가 발생하면, catch()로 처리할 수 있다.

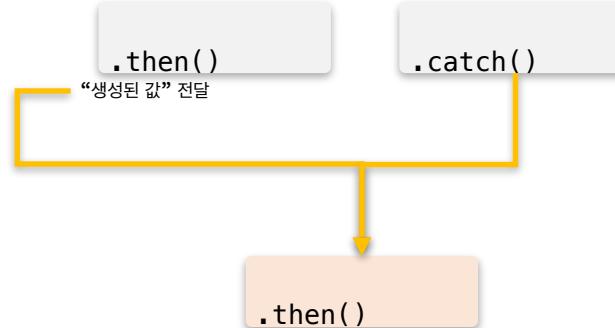
```
promise.then(function(element) {
  alert(element.id + " 입력되었습니다.");
}).catch(function(value) {
  alert(value);
}).then(function() {
  alert("다음 프로미스!1");
}).then(function() {
  alert("다음 프로미스!2");
  throw new Error("에러 발생!");
}).then(function() {
  alert("다음 프로미스!3");
}).catch(function(error) {
  alert(error);
});
```



# 프로미스 연결에서 값 반환하기

- ☒ .catch()나 .then()에서 다음 .then()으로 넘어갈 때, 값을 전달할 수 있다.

```
promise.then(function(element) {  
    alert(element.id + " 입력되었습니다.");  
    return "생성된 값";  
}).catch(function(value) {  
    alert(value);  
}).then(function(value) {  
    if (value) {  
        alert(value);  
    }  
});
```



12장

# 모듈로 캡슐화하기

# 모든 것을 공유하는 Javascript

- ☑ 기존의 Javascript는 패키지 등 코드를 분할 할 수 있는 방법이 없음.
- ☑ 따라서 아래와 같은 코드가 있을 경우, 함수나 변수의 이름이 충돌할 가능성이 높았고, 이는 곧 에러를 발생시키게 됨.

```
<script type="text/javascript" src="common.js"></script>
<script type="text/javascript" src="ui.js"></script>
<script type="text/javascript" src="wyswyg.js"></script>
```

  - 자바스크립트에서 최상위에 선언된 함수나 변수는 Global Scope에 등록된다.
  - 다른 스크립트를 로드하더라도 Global Scope에 등록된다.
- ☑ ECMAScript 2015에서는 이런 문제를 해결하기 위해 “모듈” 시스템을 도입.
  - 함수나 변수의 이름을 충돌시킬 수 있는 가능성을 낮춰줌.
- ☑ 모듈은 스크립트 파일 내에서 원하는 함수나 변수만 골라 사용할 수 있다.

## 모듈이란

---

- ✓ 모든 것을 공유하는 구조와 다르게, 모듈의 최상위 수준에서 만들어진 변수는 Global Scope에 등록되지 않는다.
  
- ✓ 모듈에 등록된 변수나 함수는 모듈내의 Global Scope에 등록된다.
- ✓ 모듈에 등록된 변수나 함수는 외부에서 자유롭게 사용할 수 없다.
  - 외부에서 사용하기 위해서는 반드시 `export` 키워드를 사용해 주어야 한다.
  - `export`된 함수나 변수를 외부에서 사용하려면 `import` 키워드를 사용해야 한다.

# Export 기본

- ✓ 다른 모듈에 코드 일부를 노출시키기 위해 export 키워드 사용.

```
// 데이터 익스포트
export var color = "red";
export let name = "Michael";
export const magicNumber = 7;

// 함수 익스포트
export function sum(num1, num2) {
    return num1 + num2;
}

// 비공개 함수
function subtract(num1, num2) {
    return num1 - num2;
}

// 비공개 함수
function multiply(num1, num2) {
    return num1 * num2;
}

// 위에서 정의한 함수 익스포트
export { multiply }
```

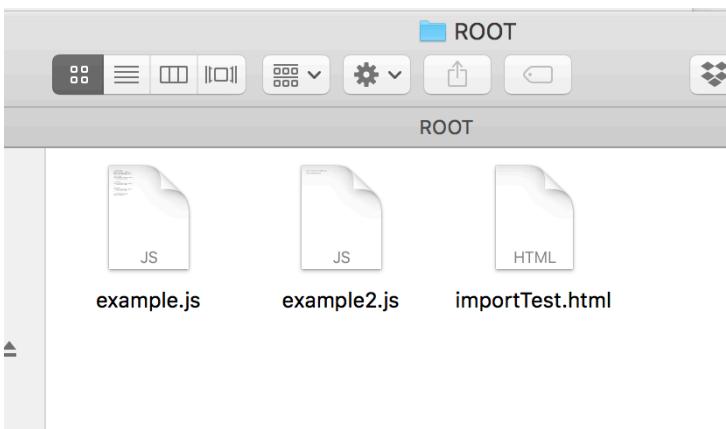
Export 키워드로 선언되지 않은 함수나 변수는 외부에서 사용할 수 없다.

# Import 기본

- ✓ Export 한 모듈이 있을 때, import 키워드를 이용해 접근할 수 있다.

```
// example2.js  
  
import { sum } from "./example.js";  
  
console.log(sum(10, 20));
```

- ✓ CommonsJS를 사용하는 Node.js에서는 import를 사용할 수 없다.
  - require(); 사용
- ✓ 이 코드는 Local에서 실행될 수 없다. 반드시 서버(tomcat 등)에 등록해 테스트 해야 한다.



```
// importTest.html  
  
<!DOCTYPE html>  
<html lang="ko">  
  <head>  
    <meta charset="UTF-8">  
    <title>Document</title>  
    <script type="module">  
      import { sum } from "./example.js";  
      console.log(sum(10, 20));  
    </script>  
  </head>  
  <body>  
  </body>  
</html>
```

# Import 기본

- ✓ Export된 변수나 함수는 필요에 따라 하나 또는 여러개 또는 모두 import할 수 있다.

```
// 하나 임포트하기  
  
import { sum } from "./example.js";  
  
console.log(sum(10, 20));
```

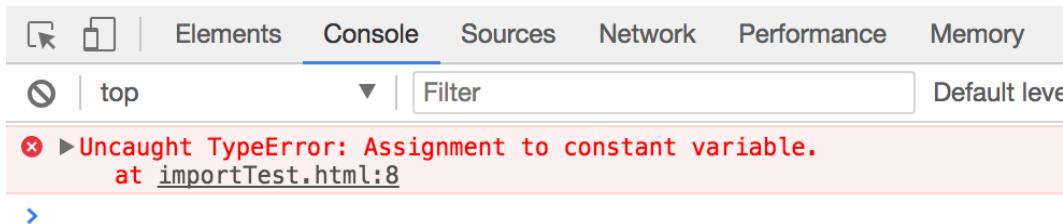
```
// 여러개 임포트하기  
  
import { sum, multiply } from "./example.js";  
  
console.log(sum(10, 20));  
console.log(multiply(10, 20));
```

```
// 모두 임포트하기  
  
import * as example from "./example.js";  
  
console.log(example.sum(10, 20));  
console.log(example.multiply(10, 20));  
console.log(example.magicNumber);
```

# Import의 특징

- ✓ Export된 변수나 함수를 Import 했을 경우, 함수 및 변수 등은 모두 읽기 전용으로 바인딩 된다.

```
import { sum } from "./example.js";
sum = function() {
    return "Hello";
}
console.log(sum());
```

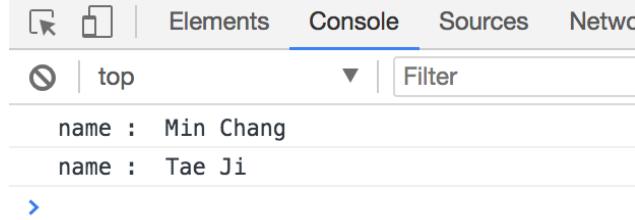


- ✓ 단, 함수를 통한 모듈내의 값은 변경이 가능하다.

```
// 데이터 익스포트
export let name = "Michael";

export function setName(newName) {
    name = newName;
}
```

```
import { name, setName } from "./example.js";
console.log("name : ", name);
setName("Tae Ji");
console.log("name : ", name);
```



## 별칭으로 Import 하기

---

- ✓ 일반적으로 Export한 이름 그대로 Import 됨.
- ✓ 때때로, 중복된 이름 혹은 너무 긴 이름의 변수, 함수, 클래스가 있을 경우 새로운 이름으로 Import해 올수 있다.

```
import { sum as add } from "./example.js";
// Uncaught ReferenceError: sum is not defined
// console.log(sum(10, 20));
console.log(add(10, 20));
```

# 모듈의 대표(변수/함수/클래스) 만들기

- ✓ 모듈에서 여러개의 변수/함수/클래스를 export할 수도 있지만 대표 변수/함수/클래스 하나만 export할 수도 있다.
- ✓ default 키워드를 사용해 대표 변수/함수/클래스를 만들수 있다.
- ✓ 하나의 모듈에는 하나의 default만 사용할 수 있다.

```
// Default 함수 만들기  
export default function(num1, num2) {  
    return num1 + num2;  
}
```

```
import sum from "./example.js";  
console.log(sum(10, 20));
```

```
function sum(num1, num2) {  
    return num1 + num2;  
}  
  
export default sum;
```

모듈에 Default가 등록되어 있으면, 그 자체가 모듈이 되므로 이름을 작성하지 않아도 된다.

Default 모듈을 임포트 하려면, 임의의 이름을 부여한다.

```
function sum(num1, num2) {  
    return num1 + num2;  
}  
  
export { sum as default };
```

함수를 정의한 후 export default 를 사용해도 된다.

혹은 as default 식별자를 사용할 수도 있다.

# Default와 Export의 동시 Import

- ✓ Default로 정의된 변수/함수/클래스와 동시에 일반적인 export를 함께 Import 할 수도 있다.
- ✓ 아래와 같이 정의된 모듈이 있을 때

```
export let color = "red";  
  
export default function (num1, num2) {  
    return num1 + num2;  
}
```

- ✓ 아래와 같은 방법들로 Import할 수 있다.

```
import sum, {color} from "./example.js";  
console.log(sum(10, 20));  
console.log(color);
```

```
import {default as sum, color} from "./example.js";  
console.log(sum(10, 20));  
console.log(color);
```

자바스크립트로 웹 브라우저의 HTML DOM 을 다루는 방법

# **HTML DOM WITH JAVASCRIPT**

# DOM 다루기 목차

---

## DOM 다루기

- CSS 셀렉터를 이용
- DOM 의 노드 찾기와 만들기 그리고 붙이기

## 이벤트 처리

- addEventListener('event', 콜백함수)

## CSS 스타일

- .classList.add('클래스 명')
- add() 대신에 remove(), toggle(), contains() 메소드 제공

# DOM 선택하기

---

## HTML 코드

```
<div class="container" id="root"></div>
```

## 아이디로 노드 찾기

```
document.getElementById('root');  
document.querySelector('#id');
```

# DOM 선택하기

---

## HTML 코드

```
<div class="container" id="root"></div>
```

## 클래스 이름으로 노드 찾기

```
document.getElementsByClassName('container');  
document.querySelector('.container');  
document.querySelectorAll('.container');
```

# DOM 선택하기

---

## HTML 코드

```
<div class="container" id="root">Hello</div>
```

## 태그 이름으로 노드 찾기

```
document.getElementsByTagName('div');  
document.querySelector('div');  
document.querySelectorAll('div');
```

모두 *IE8*부터 지원

# DOM 에서 데이터 가져오기

---

## HTML 코드

```
<div class="container" id="root">Hello</div>
```

## “textContent” 속성 사용

```
var root = document.querySelector('#root');
console.log(root.textContent);

// 노드를 찾아서 내용을 새롭게 교체한다.
document.querySelector('#root').textContent = '안녕하세요';
```

# 새로운 노드 만들기

## HTML 코드

```
<div class="container" id="root">Hello</div>
```

새로운 노드를 만들고 기존 노드에 붙인다.

```
// 루트 노드
var root = document.querySelector('#root');

// 새로운 노드를 만든다.
var newNode = document.createElement('h2');
newNode.textContent = '이것은 H2 엘레먼트입니다.';

// 루트노드에 새로운 노드를 붙인다.
root.appendChild(newNode);
```

# 이벤트 처리

---

## HTML 코드

```
<button type="button">Contact us</button>
```

## 버튼에 클릭 이벤트 부여

```
const theButton = document.querySelector('...');

theButton.addEventListener('click', function() {
    alert('hello');
});
```

# 노드에 스타일 동적으로 부여하기

## HTML 코드

```
<div class="container" id="root">Hello</div>
```

HTMLElement 는 classList 라는 DOMTokenList 를 반환

```
document.querySelector('#root').classList.add('active');
```

```
// add() 이외에 remove(), toggle(), contains() 사용 가능
```

IE에서는 메서드 별로 지원 버전이 다르다.  
폴리필을 추가하거나 다른 방법을 사용해야 한다.

```
document.querySelector('#root').className += ' active';
```

5장.

webpack

npm

babel

# 프론트엔드 개발환경 관련 도구

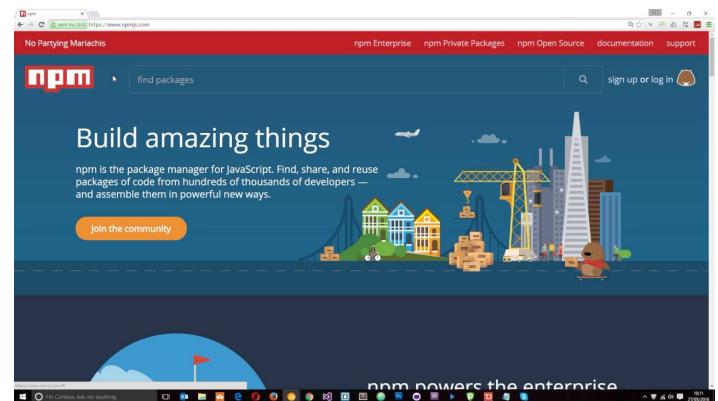
npm - Node Package Manager

**NPM**

# Node Package Manager

- ✓ NPM - 자바스크립트(노드) 프로젝트 매니저
  - 대부분 자바스크립트(노드) 프로젝트에서 사용
  - 간단한 프로젝트 구조를 생성해 줌
  - 노드의 모듈을 관리 (참고. 자바의 maven과 유사)
  - package.json 파일이 생성됨

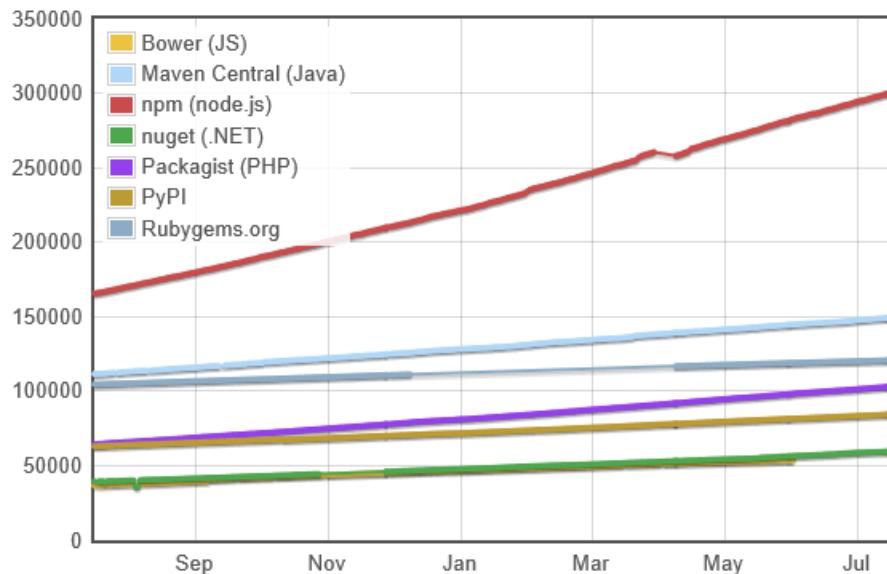
- ✓ 주요 기능
  - 프로젝트 생성
  - 모듈 설치 및 업데이트
  - 프로젝트 관련 스크립트 생성 및 실행
    - 빌드, 개발환경 수행, 테스트 등



# 노드 생태계 - NPM

- 노드용 외부 모듈 :
  - <http://modulecounts.com>
- NPM
  - <https://www.npmjs.com/>
  - 노드 용 모듈관리 시스템

## Module Counts



# npm init :: 프로젝트 생성

```
1. soongon@Soongonui-MacBook-Pro: ~/frontend-projects/football-highlights (zsh)
→ frontend-projects mkdir football-highlights && cd football-highlights
→ football-highlights npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

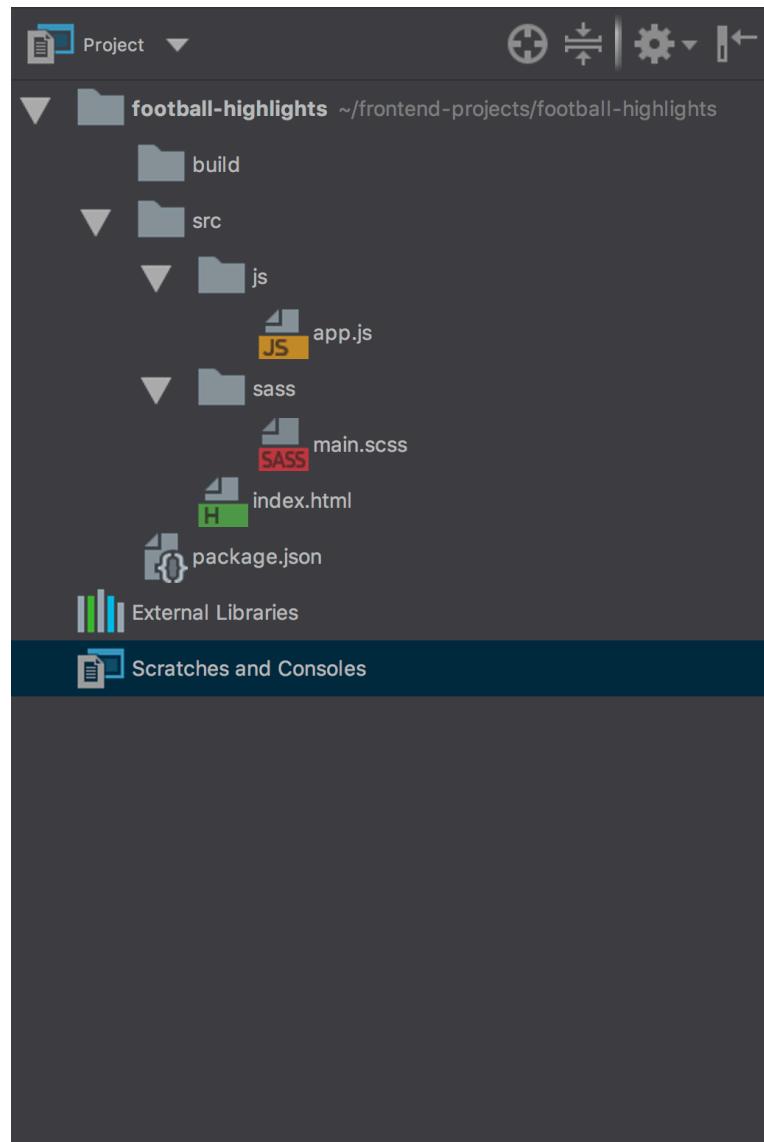
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (football-highlights)
version: (1.0.0)
description: football highlights
entry point: (index.js)
test command:
git repository:
keywords:
author: soongon
license: (ISC)
About to write to /Users/soongon/frontend-projects/football-highlights/package.json:

{
  "name": "football-highlights",
  "version": "1.0.0",
  "description": "football highlights",
  "main": "index.js",
```

# 디렉토리 구조 생성

- 프로젝트 루트 디렉토리
  - package.json
- Build 와 src 를 구분
- src/js 디렉토리
- src/sass 디렉토리



# package.json 으로 작업하기

## ☒ 필수 항목

```
{  
  "name": "my-awesome-package",  
  "version": "1.0.0"  
}
```

## ☒ package.json 작성

```
> npm init
```

```
> npm init --yes    > npm init -y
```

# npm : 의존성 관리

- package.json 을 통해 의존성 관리

## my\_app/package.json

```
{  
  "name": "my_app",  
  "version": "1.0.0",  
  "dependencies": {  
    "connect": "1.8.7"  
  }  
}
```

*version number*



**“npm init”** 으로 package.json 생성 가능

```
$ npm install
```

node\_modules 디렉토리에 설치

my\_app

node\_modules

connect

# npm 주요 명령어

## ☒ 패키지 설치

```
> npm install <package_name>
```

```
> npm i <package_name>
```

- -S : save dependency
- -D : dev save dependency

## ☒ 패키지 업데이트

```
> npm outdated
```

```
> npm update
```

## ☒ 패키지 삭제

```
> npm uninstall <package_name>
```

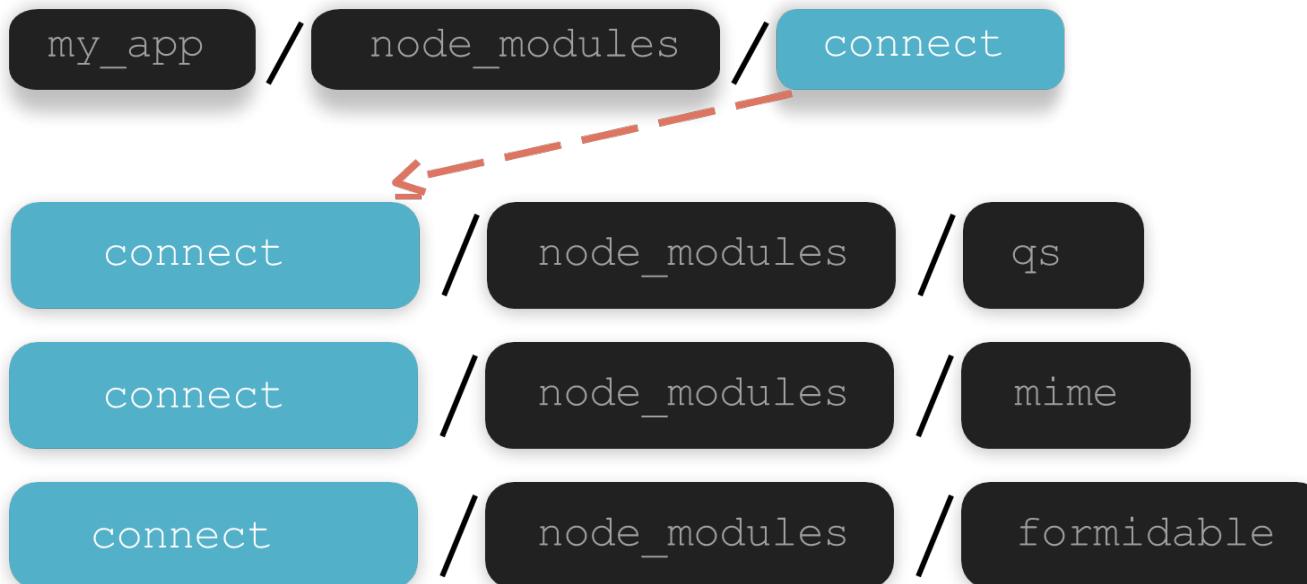
```
> npm rm <package_name>
```

# 하위 의존성 관리

## my\_app/package.json

```
"dependencies": {  
    "connect": "1.8.7"  
}
```

### 하위 종속성 자동 설치



# 모듈 검색

```
$ npm search request
```

The screenshot shows a web browser window titled "results for coffee". The URL in the address bar is <https://www.npmjs.com/search?q=coffee>. The page header includes links for "Nodeschool Public Materials", "npm Enterprise", "features", "pricing", "documentation", and "support". On the right side, there is a "sign up or log in" button with a user icon.

The search bar contains the query "coffee-sc". Below it, a list of packages is displayed:

- coffee-script**  
Unfancy JavaScript
- coffeescript-mixins**  
easy to use mixins with CoffeeScript
- ember-cli-coffeescript**  
Adds precompilation of CoffeeScript files and all the basic generation types to the 'ember generate' command.
- coffeescript-compiler**  
Compile CoffeeScript code from JavaScript.
- coffeescript-concat**  
A utility for combining coffeescript files and resolving their dependencies.
- coffee-script**  
coffee script

On the left side, there are two package cards:

- coffee** (popo)  
Test command line on node  
★ 1 v3.2.4  
⌚ test, shell, spawn,
- coffee-bean**  
Coffee measurements from coffee  
★ 0 v0.1.0  
⌚ coffee

At the bottom, there is a footer note: "front and lots of other javascript developers." and a "powered by Constructor.io" link.

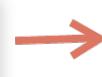
# 시멘틱 버저닝

```
"connect": "1.8.7"
```



## Ranges

```
"connect": "~1"
```



```
>=1.0.0 <2.0.0
```

*Dangerous*

```
"connect": "~1.8"
```



```
>=1.8.0 <1.9.0
```

*API could change*

```
"connect": "~1.8.7"
```



```
>=1.8.7 <1.9.0
```

*Considered safe*

<http://semver.org/>

# 새로운 패키지 매니저 :: yarn

 yarn Getting Started Docs Packages Blog A ം English ▾    

Search packages (i.e. babel, webpack, react...)

# FAST, RELIABLE, AND SECURE DEPENDENCY MANAGEMENT.

[GET STARTED](#) [INSTALL YARN](#)  Star 32,964

Stable: [v1.9.4](#)  
Node: ^4.8.0 || ^5.7.0 || ^6.2.2 || >=8.0.0

Ultra Fast.

Yarn caches every package it downloads so it never needs to download it again. It also parallelizes operations to maximize resource utilization so install times are faster than ever.

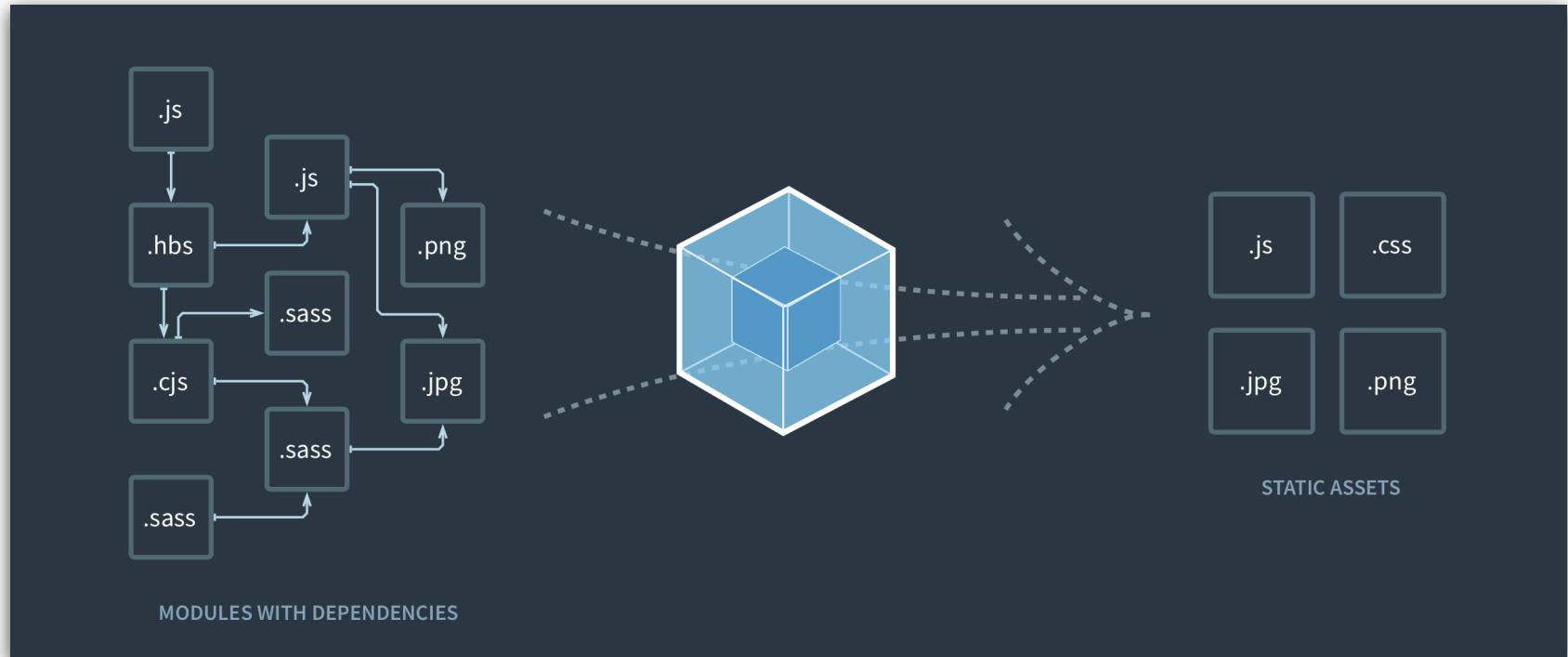


**웹팩**

# 웹팩, webpack

## ✓ Code bundler

- html, script, css 파일을 하나로 묶어 줌
- 빌드 과정 중 여러 기능을 포함 할 수 있음 (변환, 최적화 등)
  - Loader, plugin 사용



# 웹팩, webpack

## Write your code

src/index.js

```
import bar from './bar';
bar();
```

src/bar.js

```
export default function bar() {
  //
}
```

## Bundle with webpack

Without config or provide custom webpack.config.js

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  }
};
```

page.html

```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
    <script src="dist/bundle.js"></script>
  </body>
</html>
```

Then run `webpack` on the command-line to create `bundle.js`.

# webpack 설치

## ✓ 설치

- npm install --save-dev webpack
- npm install --save-dev webpack-cli

```
1. soongon@Soongonui-MacBook-Pro: ~/frontend-projects/football-highlights (zsh)
→ football-highlights npm install --save-dev webpack && npm install --save-dev webpack-cli

> fsevents@1.2.4 install /Users/soongon/foreground-projects/football-highlights/node_modules/f
sevents
> node install

[fsevents] Success: "/Users/soongon/foreground-projects/football-highlights/node_modules/fseve
nts/lib/binding/Release/node-v57-darwin-x64/fse.node" already installed
Pass --update-binary to reinstall or --build-from-source to recompile
npm WARN football-highlights@1.0.0 No repository field.

+ webpack@4.16.5
added 357 packages from 288 contributors and audited 3547 packages in 5.891s
found 0 vulnerabilities

npm WARN football-highlights@1.0.0 No repository field.

+ webpack-cli@3.1.0
updated 1 package and audited 3547 packages in 3.578s
found 0 vulnerabilities
```

# webpack.config.js

## ✓ 설정 파일

### ○ webpack.config.js

```
var path = require('path');

module.exports = {
  entry: {
    app: './src/js/app.js'
  },
  output: {
    path: path.resolve(__dirname, 'build'),
    filename: '[name].bundle.js'
  }
};
```

```
1. soongon@Soongnui-MacBook-Pro: ~/frontend-projects/football-highlights (zsh)
→ football-highlights webpack --config webpack.dev.config.js
zsh: command not found: webpack
→ football-highlights npx webpack --config webpack.dev.config.js
Hash: 37e4d406b20344f9022b
Version: webpack 4.16.5
Time: 269ms
Built at: 2018-08-12 12:08:23
          Asset      Size  Chunks             Chunk Names
app.bundle.js  930 bytes     0  [emitted]  app
Entrypoint app = app.bundle.js
[0] ./src/js/app.js 0 bytes {0} [built]
```

npx : 로컬로 설치된 모듈을 경로를  
별도로 지정하고 실행하는 방법

# webpack dev server

## ✓ 설치

- npm install webpack-dev-server --save-dev

## ✓ 실행

- webpack-dev-server --config webpack.dev.config.js

```
1. npx webpack-dev-server --config webpack.dev.config.js (open)
→ football-highlights npm i webpack-dev-server --save-dev
npm WARN football-highlights@1.0.0 No repository field.

+ webpack-dev-server@3.1.5
updated 1 package and audited 6210 packages in 4.319s
found 0 vulnerabilities

→ football-highlights npx webpack-dev-server --config webpack.dev.config.js
i [wds]: Project is running at http://localhost:8080
i [wds]: webpack output is served from /
i [wds]: Content not from webpack is served from ./src
i [wdm]: Hash: 515983daf6a0f882016f
Version: webpack 4.16.5
Time: 476ms
Built at: 2018-08-12 12:47:39
      Asset      Size  Chunks             Chunk Names
app.bundle.js  338 KiB     app  [emitted]    app
Entrypoint app = app.bundle.js
[./node_modules/ansi-html/index.js] 4.16 KiB {app} [built]
```

```
"scripts": {
  "test": "echo \\\"Error: no test specified\\\" && exit 1",
  "serve": "npx webpack-dev-server --config webpack.dev.config.js"
},
```

npm script 활용

# Dev server 설정

---

```
var path = require('path');

module.exports = {
  entry: {
    app: './src/js/app.js'
  },
  output: {
    path: path.resolve(__dirname, 'build'),
    filename: '[name].bundle.js'
  },
  devServer: {
    contentBase: path.resolve(__dirname, 'build'),
    compress: true,
    open: true
  }
};
```

# Css 파일 설정

---

- ✓ 부트스트랩 css 파일을 프로젝트 엔트리 포인트에 포함

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

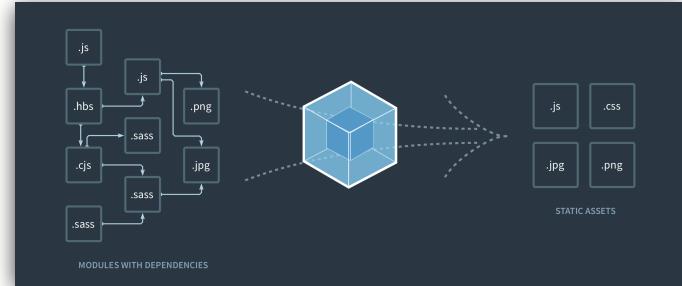
- ✓ css 파일에 대한 로더를 설정

```
...
module: {
  rules: [
    {
      test: /\.css$/,
      use: ['style-loader', 'css-loader']
    }
  ]
}
...
```

# Webpack plugin

- ✓ webpack 플러그인 사용
  - html-webpack-plugin

```
npm i -D html-webpack-plugin
```



# HTML 웹팩 플러그인 :: html-webpack-plugin

- ✓ 번들링 된 파일을 자동으로 HTML 페이지에 삽입

```
const HtmlWebpackPlugin = require('html-webpack-plugin')

module.exports = {
  entry: 'index.js',
  output: {
    path: __dirname + '/dist',
    filename: 'index_bundle.js'
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './src/index.html'
    })
  ]
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Webpack App</title>
  </head>
  <body>
    <script src="index_bundle.js"></script>
  </body>
</html>
```

ES6 지원

**바벨, BABEL**

# Babel, 바벨

- ✓ ES6를 포함 자바스크립트 호환 언어를 서로 간에 변환해 줌
  - 트랜스파일 : ES6 → ES5

The screenshot shows the Babeljs.io homepage. At the top, there's a navigation bar with links like 'Learning', 'Generators', 'Setup', 'Try it out', 'Blog', 'Search', 'Donate', 'Team', and 'GitHub'. Below the navigation, a large banner features the text 'Babel is a JavaScript compiler.' and 'Use next generation JavaScript, today.' A central feature is a code editor split into two panes: 'Put in next-gen JavaScript' and 'Get browser-compatible JavaScript out'. The left pane contains the code: 

```
const x = [1, 2, 3];
foo([...x]);
```

. The right pane shows the transformed code: 

```
var x = [1, 2, 3];
foo([] .concat(x));
```

. Below the code editor is a button that says 'Check out our REPL to experiment more with Babel!'. At the bottom of the page, there are sections for 'Welcome!', 'Friends of Open Source', and a footer with links for 'GET STARTED' and 'VIDEOS'.

**Welcome!**

We're currently just a small group of volunteers that spend their free time maintaining this project. If Babel has benefited you in your work, becoming a contributor might be a great way to give back.

Learn more about Babel by reading the get started guide or watching talks about the concepts behind it.

[GET STARTED](#) [VIDEOS](#)

**Friends of Open Source**

These companies are awesome and pay these engineers to work on Babel

 AMP

The AMP Project is an open-source initiative aiming to make the web better for all.

# 바벨 로더

- ☒ ES6를 ES5(브라우저 호환 언어)로 변환하기 위해 필요
- 개발은 ES6, 배포는 ES5

Nauseating Packaged Meat

npm  log in or sign up

Share your code. npm Orgs help your team discover, share, and reuse code. [Create a free org »](#)

**babel-loader**

7.1.5 • Public • Published a month ago

Readme 3 Dependencies 7,055 Dependents 54 Versions

npm v7.1.5 build passing build passing codecov 69%

**Babel Loader**

This package allows transpiling JavaScript files using [Babel](#) and [webpack](#).

**Notes:** Issues with the output should be reported on the [babel issue tracker](#).

**Install**

```
webpack 1.x | babel-loader <= 6.x
webpack 2.x | babel-loader >= 7.x (recommended) (^6.2.10 will also work, but with deprecation warnings)
webpack 3.x | babel-loader >= 7.1
```

install

```
> npm i babel-loader
```

weekly downloads  
2,149,623

version  
7.1.5

open issues  
95

homepage  
[github.com](#)

last publish  
a month ago

collaborators

Test With RunKit

# 바벨 로더 설정

## 설치

```
npm install -D @babel/core @babel/preset-env babel-loader
```

## 설정

```
module: {
  rules: [
    {
      test: /\.js$/,
      exclude: /node_modules/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['@babel/preset-env']
        }
      }
    }
  ]
}
```

6장.

# **REACT 프로그래밍**

# 목차

---

- SPA 개요
- 리액트 소개
- React, Webpack, Babel로 React + ES6 개발환경 설정
- 리액트 컴포넌트
- 리액트 스타일링
- props 다루기
- state 다루기
- 이벤트 처리
- 컴포넌트 생명주기
- React-Router
- Todo 앱 만들기
- 서버로 요청하기

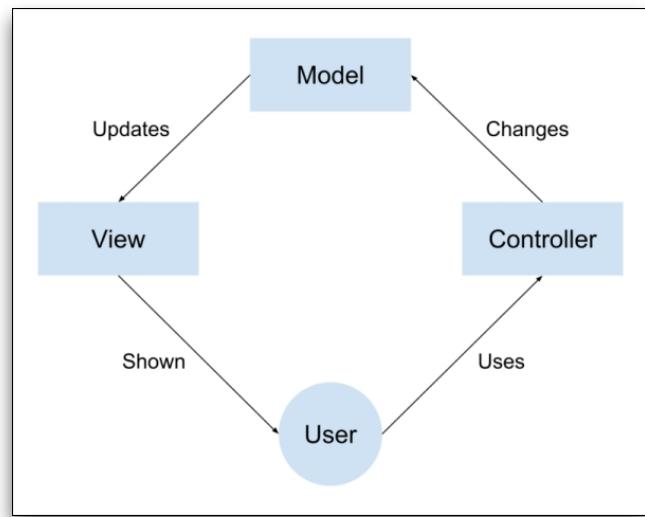
# 3-tier 웹 개발

대부분 웹 애플리케이션은 3-tier 아키텍처로 구성됨

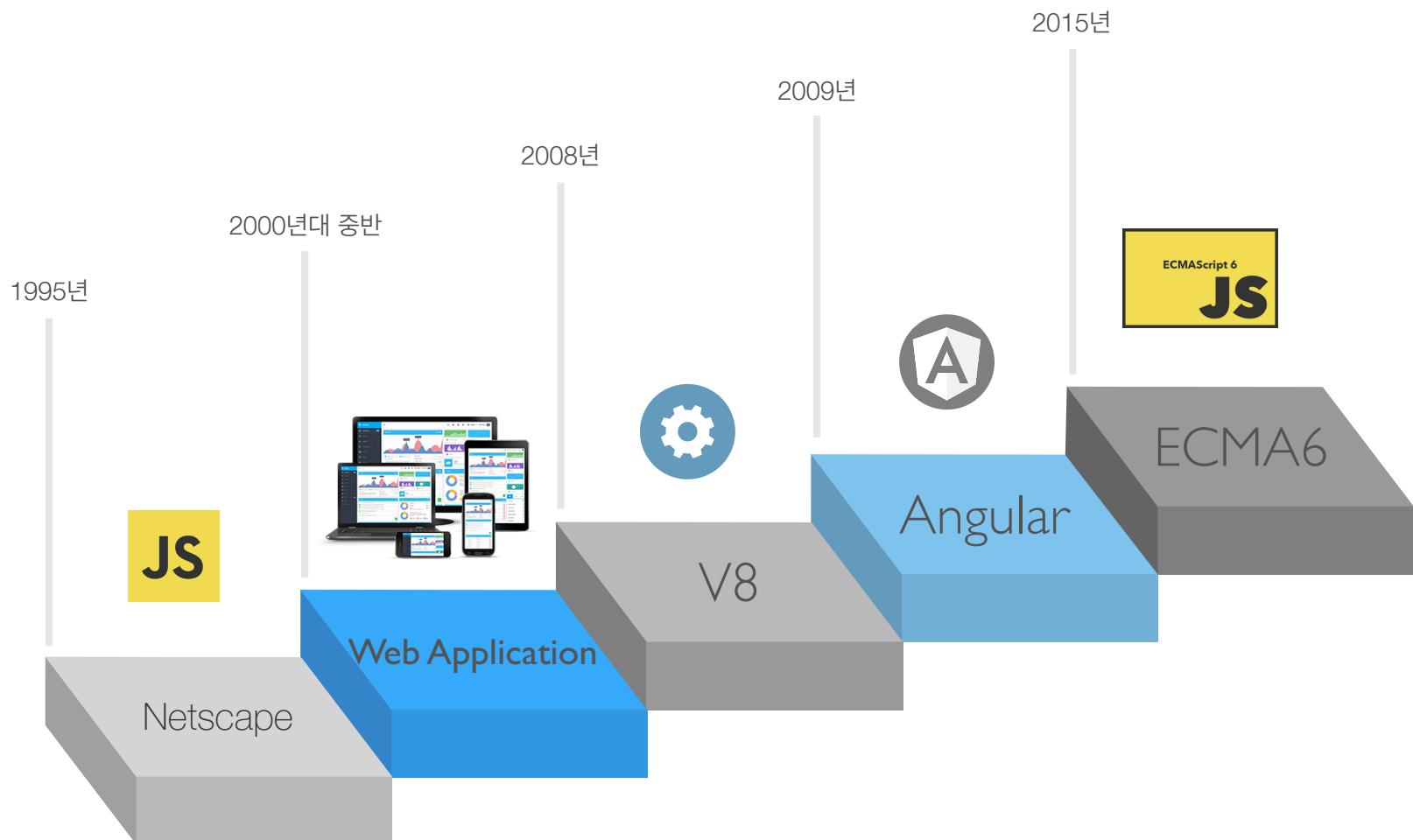
- 데이터 - 로직 - 화면
- 데이터베이스 - 서버 - 클라이언트
- 데이터베이스 - 서버 로직 - 클라이언트 로직, 클라이언트 UI

기존 LAMP 스택이나 .NET 스택

- 각 tier 별로 기술 지식이 필요 : 팀 수행, 리스크 증가



# 자바스크립트의 발전



# ECMAScript 2015 소개

---

- Modules - 언어레벨에서 지원
- Classes - 프로토타입에서 클래스로
- Arrow Functions ( => ) - 람다식 문법 지원
- Let and Const - 새로운 변수 선언 키워드
- Default, Rest, Spread - 함수 파라미터 세 가지 새로운 기능

Introduction ReactJS

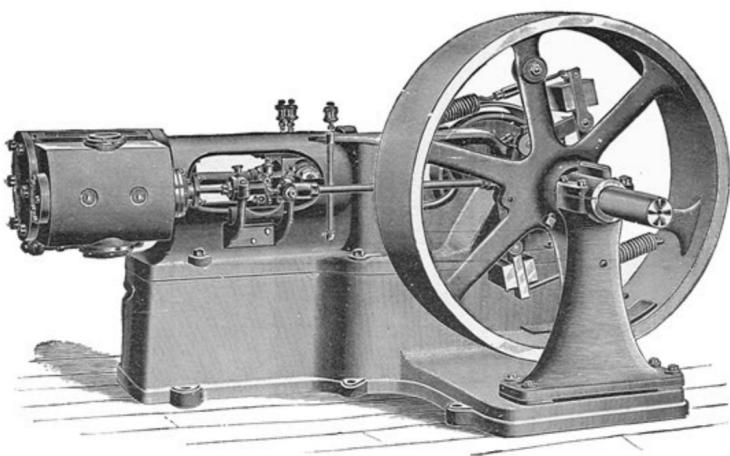
# 리액트 소개

# 전형적인 웹앱 - 멀티 페이지



Just your typical web app!

# 옛날 방식의 멀티 페이지 디자인



So...does this charge via USB?



# SPA - 싱글 페이지 앱



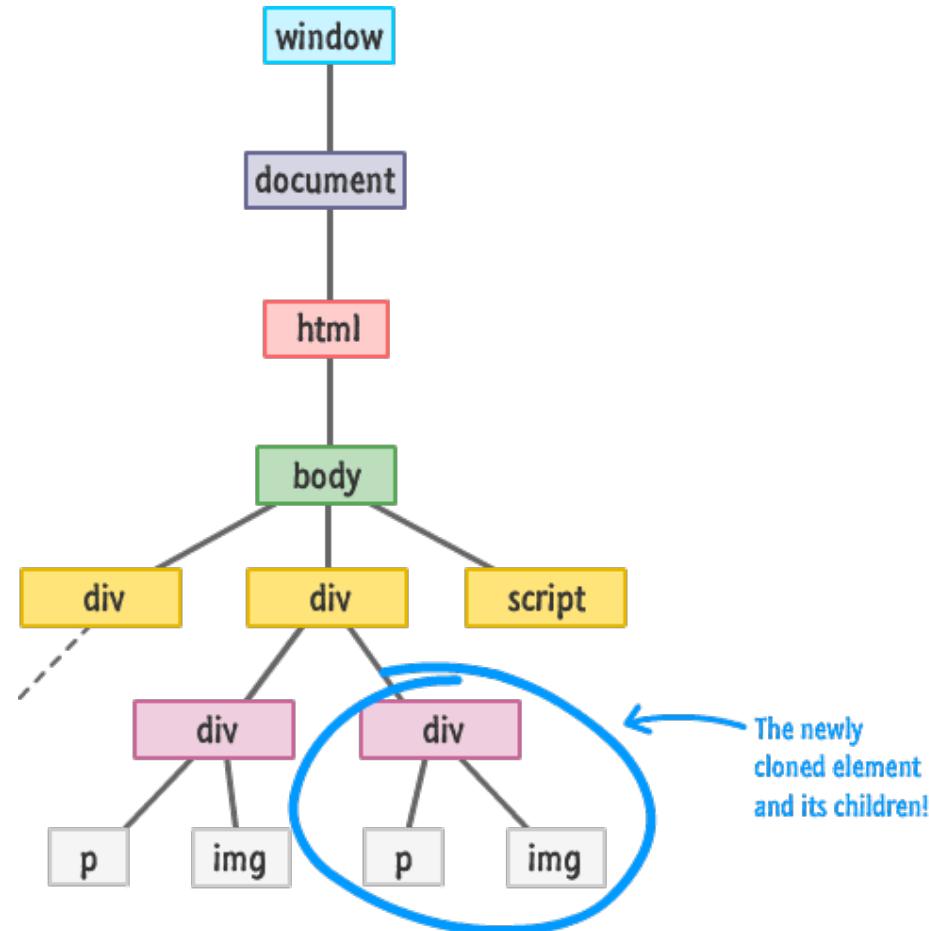
# SPA 어플리케이션 개발 시 문제점

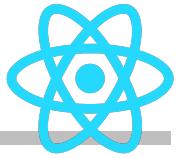
- 데이터와 UI 동기화

- DOM 조작 - DOM 은 느리다

- HTML 에 데이터 바인딩하기

- Interpolation





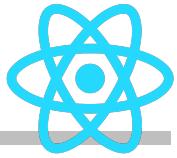
# 리액트의 특징 및 장점

## UI 상태 자동관리

- 리액트에서는 UI의 마지막 상태만 신경 쓰면 된다.



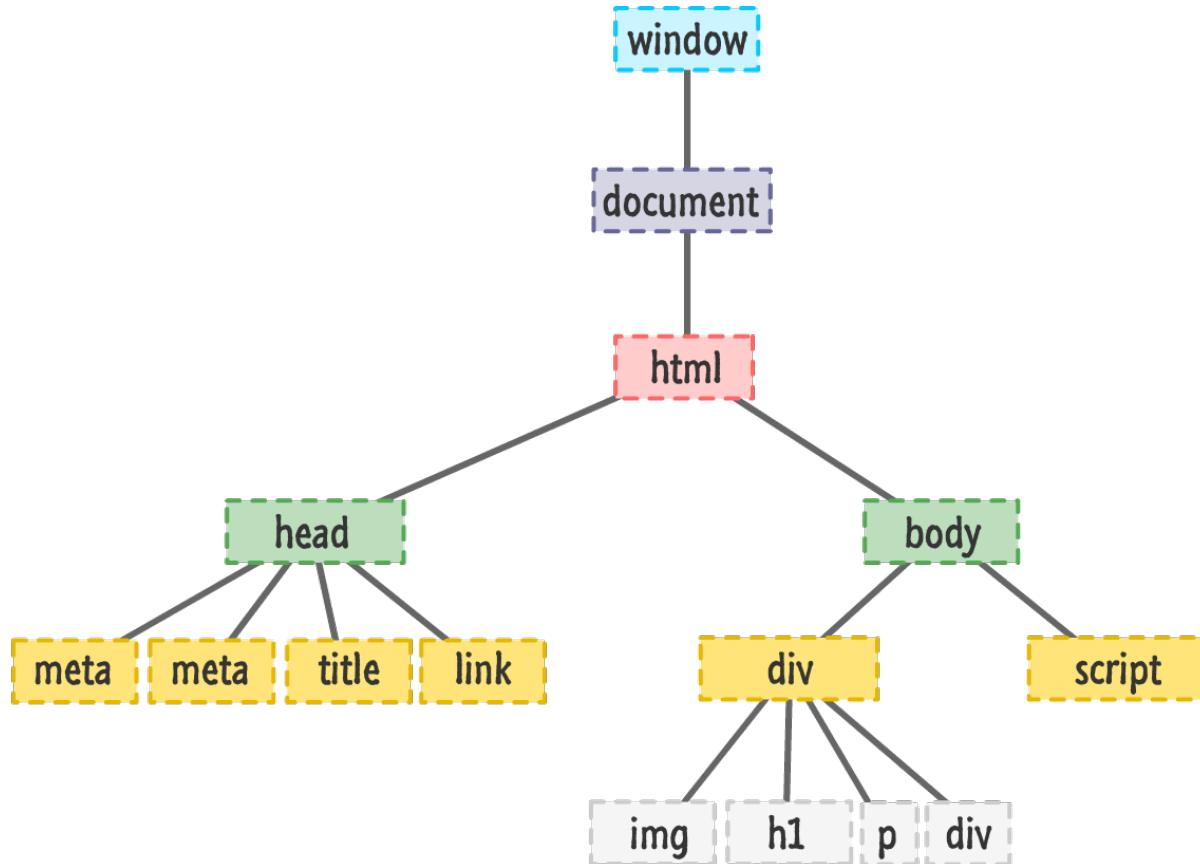
*The end state is what  
React cares about!*



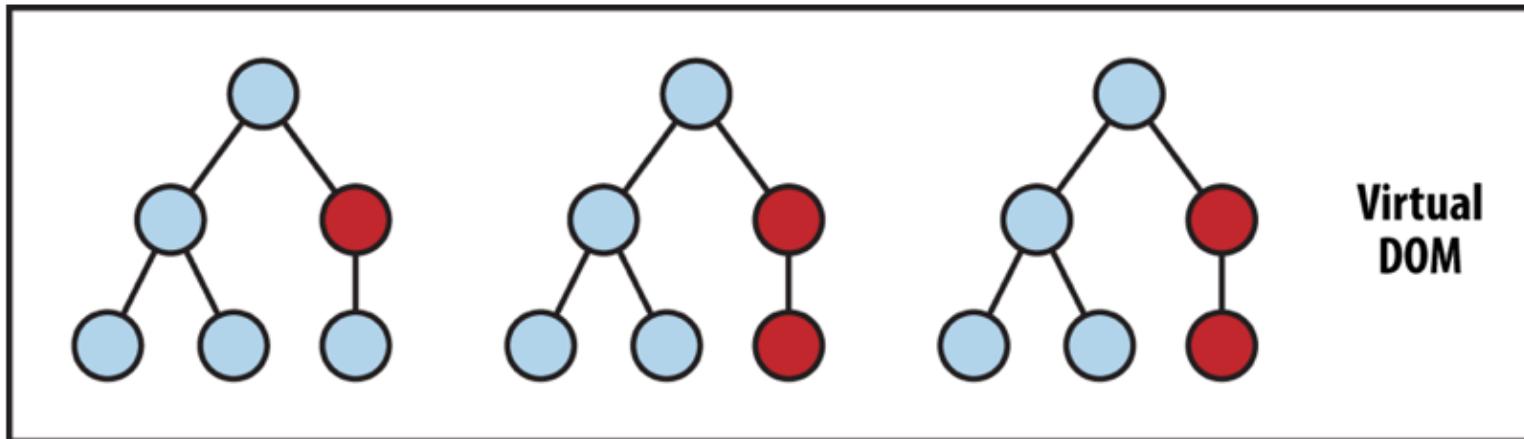
# 리액트의 특징 및 장점

## ✓ 번개처럼 빠른 DOM 조작

- DOM 을 직접 조작하지 않고 가상 DOM을 사용한다.



# Virtual DOM - diff 로 변경



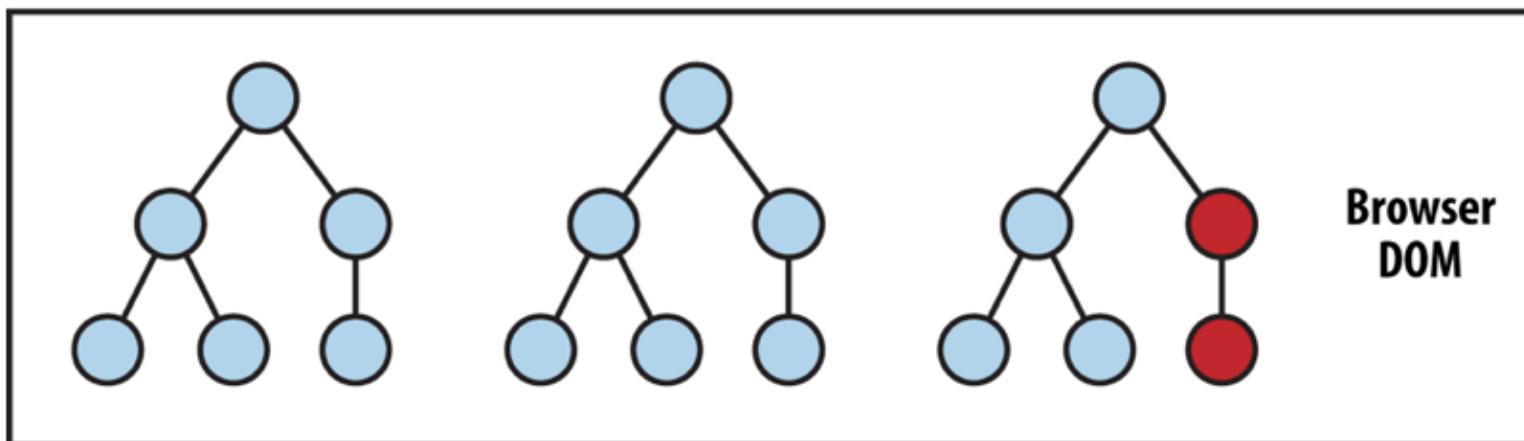
State Change

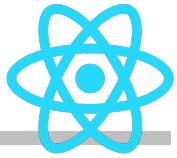


Compute Diff



Re-render

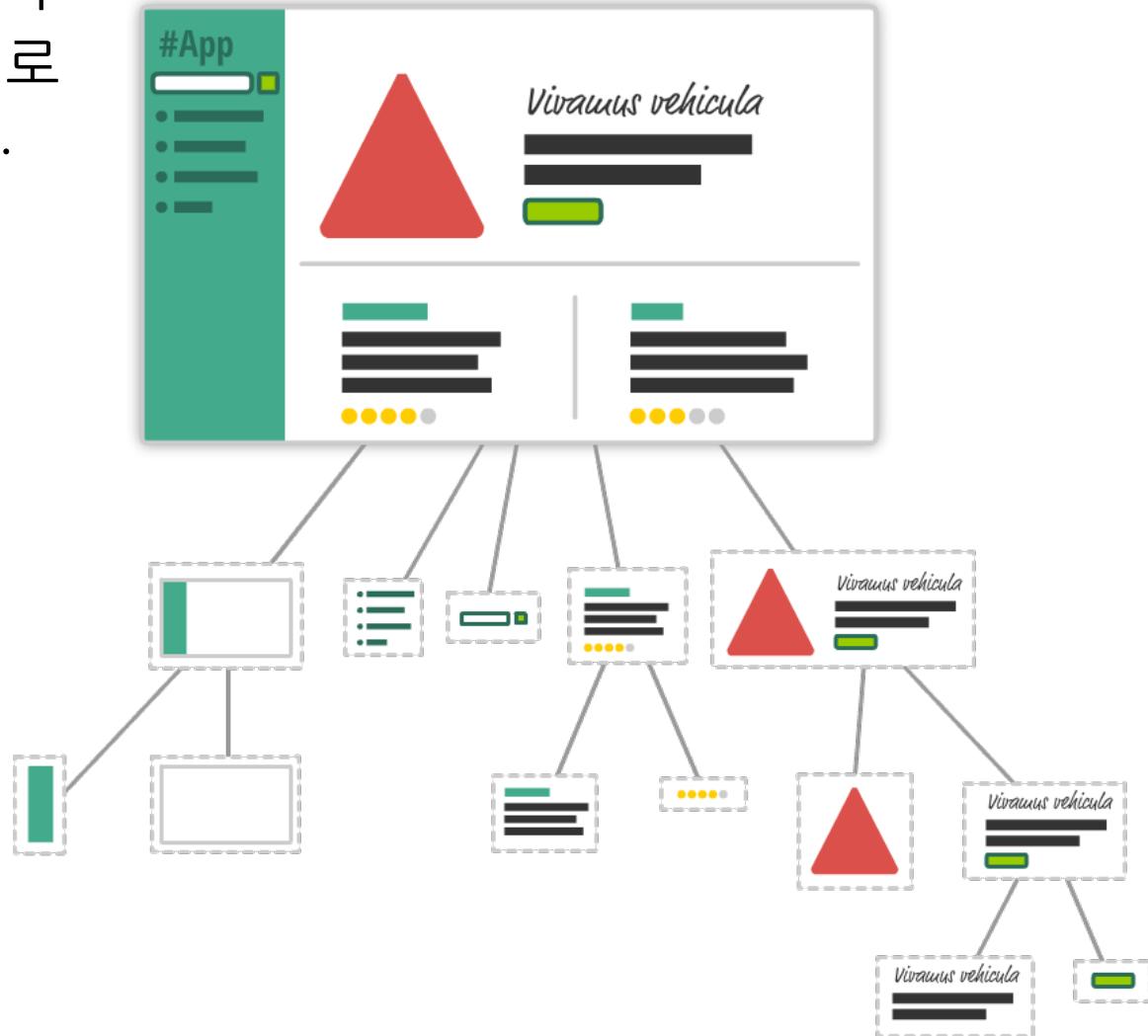


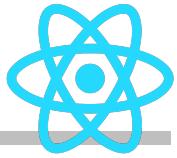


# 리액트의 특징 및 장점

## 컴포넌트 기반 UI 관리

- 전체 UI 를 단위 별로 (컴포넌트) 쪼갠다.





# 리액트의 특징 및 장점

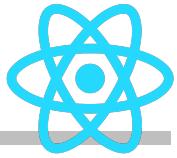
## ✓ JSX 문법 사용

### ○ HTML 과 유사한(거의 같은) 템플릿을 사용

```
ReactDOM.render(  
  <div>  
    <h1>Batman</h1>  
    <h1>Iron Man</h1>  
    <h1>Nicolas Cage</h1>  
    <h1>Mega Man</h1>  
  </div>,  
  destination  
) ;
```

왼쪽 오른쪽 모두 같은 코드이다.

```
ReactDOM.render(React.createElement(  
  "div",  
  null,  
  React.createElement(  
    "h1",  
    null,  
    "Batman"  
) ,  
  React.createElement(  
    "h1",  
    null,  
    "Iron Man"  
) ,  
  React.createElement(  
    "h1",  
    null,  
    "Nicolas Cage"  
) ,  
  React.createElement(  
    "h1",  
    null,  
    "Mega Man"  
)  
, destination);
```

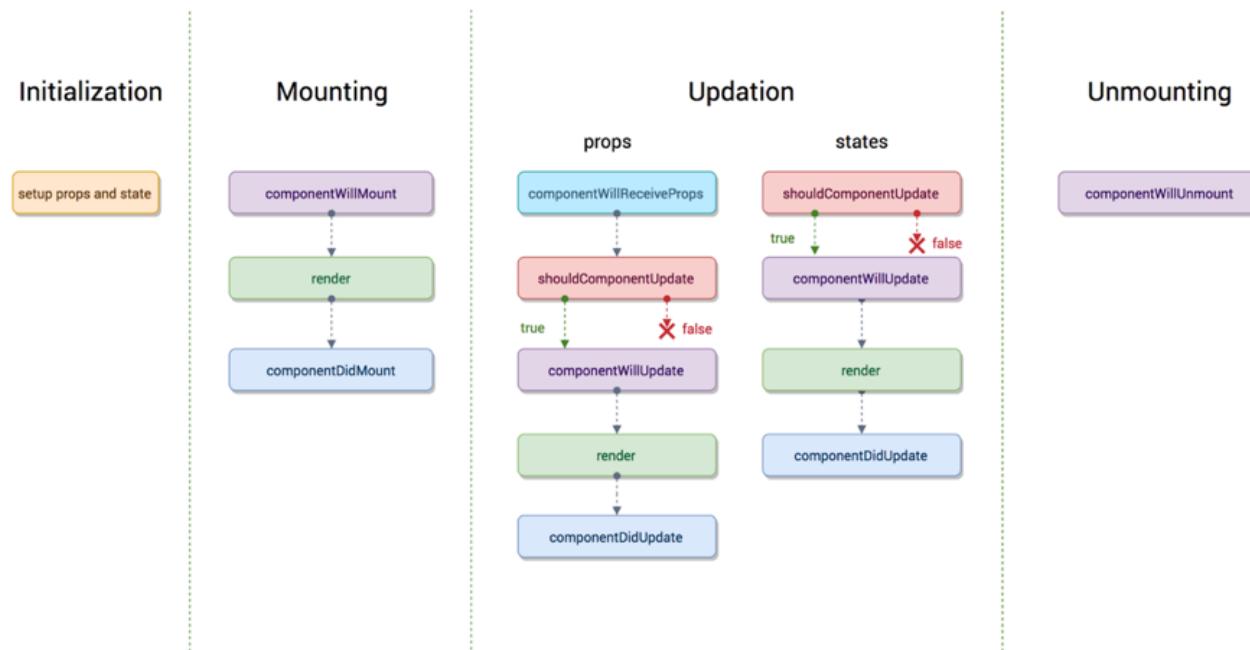


# 리액트의 특징 및 장점

- MVC 아키텍처에서 V 즉, 뷰(Render) 만 담당한다.
  - Model 과 Controller 는 다른 라이브러리가 필요하다는 뜻.
- Component Based Development - 작업의 단위
- Virtual DOM - DOM을 직접 다루지 않는다.
- JSX
  - NOT Template
  - transpile to JS
- CSR & SSR

# 선언적, Declarative 개발방식

## Declarative



Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

# React Developer Tools

The screenshot shows the Chrome Web Store page for the "React Developer Tools" extension. The page includes a sidebar with filters for category, rating, and price. The main content area displays the extension's details: title, developer, rating, reviews, and a screenshot of the browser interface showing the React DevTools extension in action. A large green button allows users to add the extension to their Chrome browser. Below the main details, there is a description of the extension's purpose, user reviews, and a related products section.

**React Developer Tools**

Facebook 님이 작성

★★★★★ (849) | 개발자 도구 | 사용자 820,962명

개요 리뷰 지원 관련 프로그램

React Developer Tools

Adds React debugging tools to the Chrome Developer Tools.

You will get a new tab called React in your Chrome DevTools. This shows you the root React components that were rendered on the page, as well as the subcomponents that they ended up rendering.

관련 프로그램

- react-detector
- Redux DevTools
- React-Sight
- JSONView

# CSR & SSR

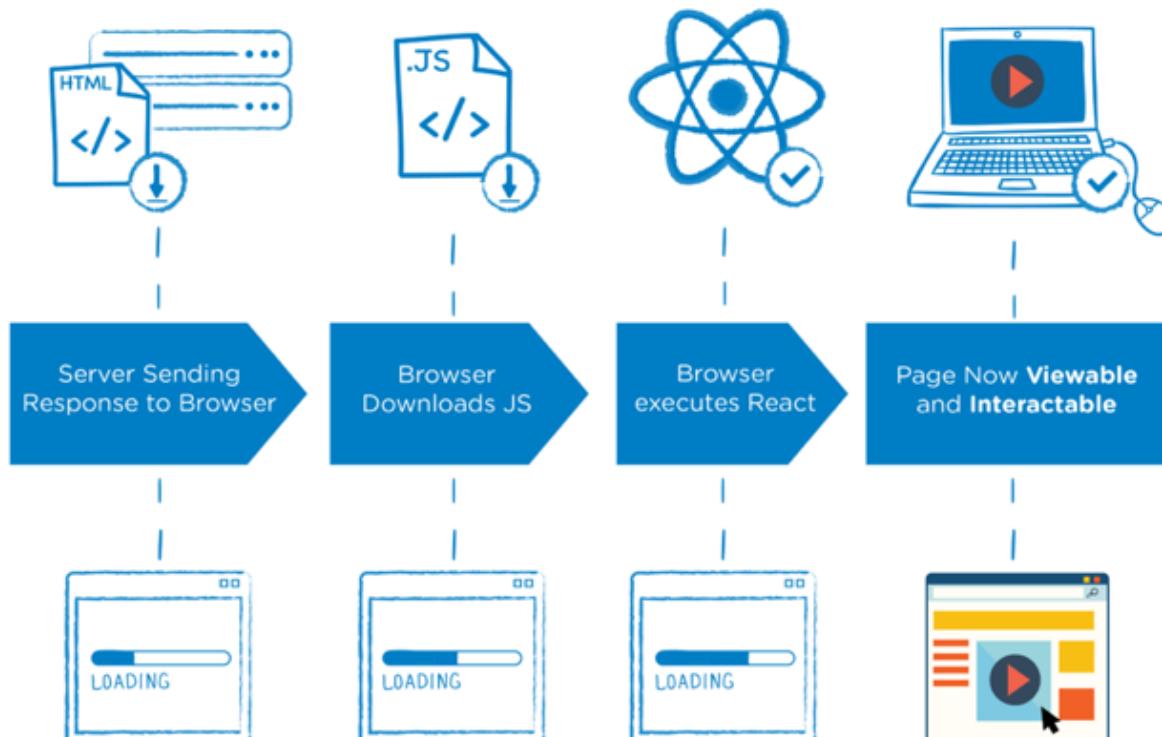
- React 는 CSR 이 기본
  - ReactDOM.render(element, container[, callback]);
- React 에서 SSR 지원을 위한 API 존재
  - ReactDOMServer.renderToString(element);
  - 리액트 컴포넌트를 서버에서 랜더해서 문자열로 만들어줌

```
app.get('*', (req, res) => {
  const html = path.join(__dirname, '../build/index.html');
  const htmlData = fs.readFileSync(html).toString();

  const ReactApp = ReactDOMServer.renderToString(React.createElement(App));
  const renderedHtml = htmlData.replace('{{SSR}}', ReactApp);
  res.status(200).send(renderedHtml);
});
```

# React Client Side Rendering

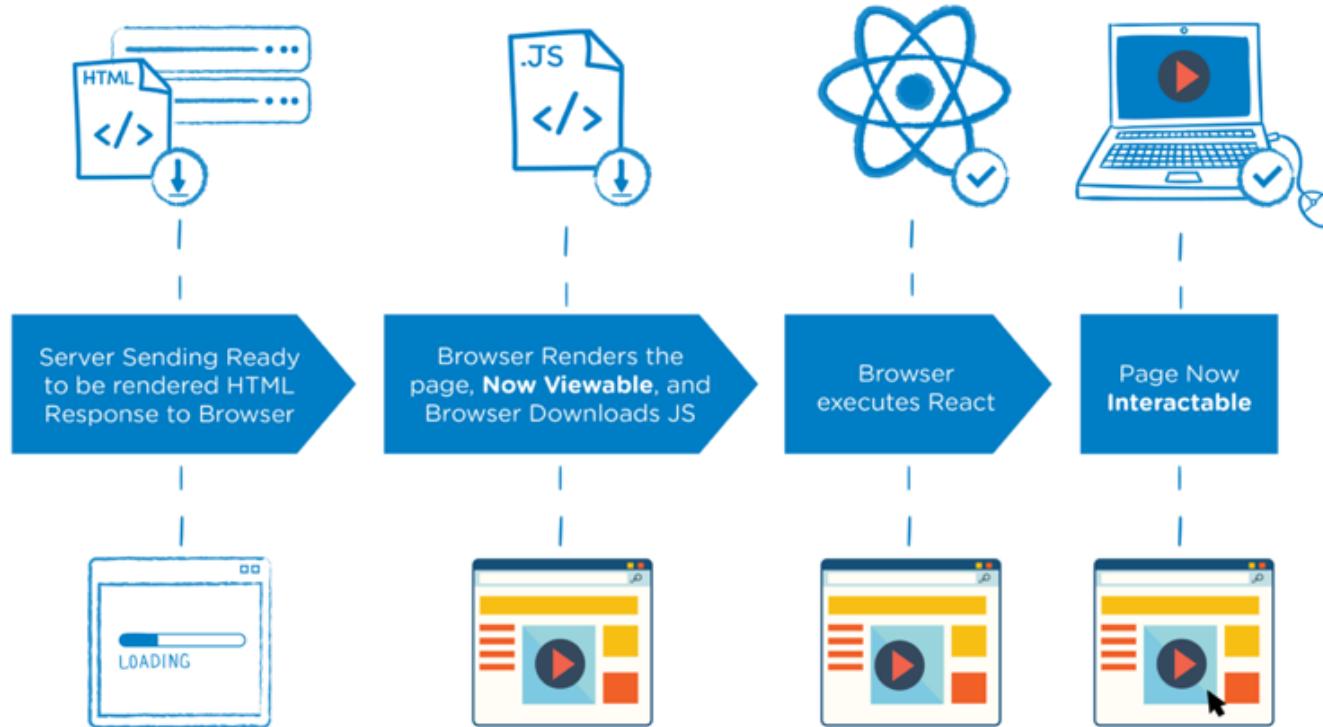
CSR



# React Server Side Rendering

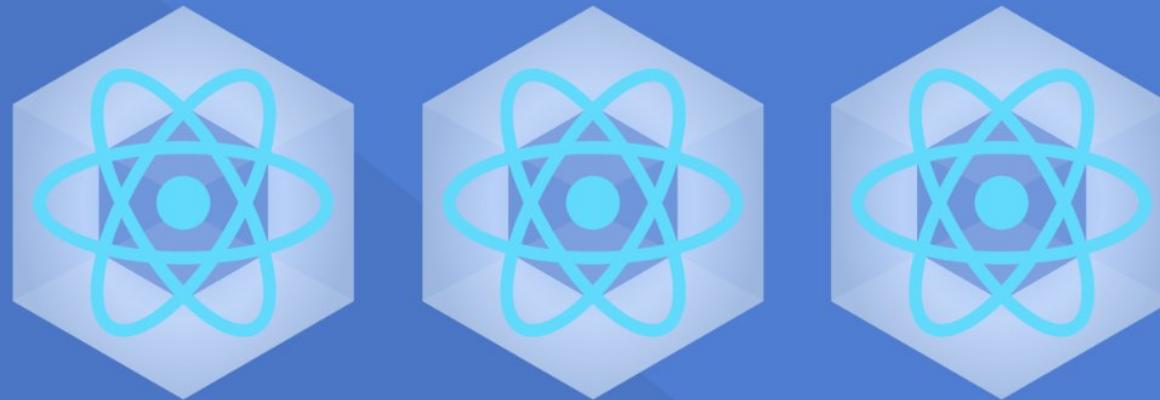
---

# SSR



# 첫 번째 리액트 앱

**같이 코딩해 봅시다!!**



SETTING UP  
**REACT, WEBPACK, BABEL**

Setup a React Environment Using webpack and Babel  
**React, Webpack, Babel**로  
React + ES6 개발환경 설정

# React with web pack, babel-loader [jsx]

---

## 모듈 번들러

- web pack, webpack-dev-server
- html-webpack-plugin

## loader

- babel-loader
- style-loader, css-loader

## babel

- @babel/core
- @babel/preset-react

## react

- react
- react-dom

# 각 모듈 설치하기

## React with webpack, babel-loader [ jsx ]

```
~/Project/webpack-react-js is 📦 v1.0.0 via ⚡ v8.9.4
→ npm i webpack webpack-dev-server html-webpack-plugin -D

~/Project/webpack-react-js is 📦 v1.0.0 via ⚡ v8.9.4
→ npm i babel-loader babel-core babel-preset-react-app -D

~/Project/webpack-react-js is 📦 v1.0.0 via ⚡ v8.9.4
→ npm i react react-dom -D

~/Project/webpack-react-js is 📦 v1.0.0 via ⚡ v8.9.4
→ npm i style-loader css-loader -D
```

# package.json

```
{  
  "name": "react-test",  
  "version": "1.0.0",  
  "description": "",  
  "main": "app.js",  
  "scripts": {  
    "build": "npx webpack",  
    "serve": "npx webpack-dev-server"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "@babel/cli": "^7.0.0",  
    "@babel/core": "^7.0.0",  
    "@babel/preset-env": "^7.0.0",  
    "@babel/preset-react": "^7.0.0",  
    "babel-loader": "^8.0.2",  
    "css-loader": "^1.0.0",  
    "html-webpack-plugin": "^3.2.0",  
    "react": "^16.4.2",  
    "react-dom": "^16.4.2",  
    "style-loader": "^0.23.0",  
    "url-loader": "^1.1.1",  
    "webpack": "^4.17.2",  
    "webpack-cli": "^3.1.0",  
    "webpack-dev-server": "^3.1.7"  
  }  
}
```

# babel-preset-react-app

- ☑ 바벨을 사용해 React 를 쉽게 설정 할 수 있도록 도와줌

- 바벨 설치

```
npm i -D @babel/core @babel/preset-react
```

- .babelrc 설정 - 프로젝트 루트 디렉토리에 위치

```
{  
  "presets": ["@babel/react-app"]  
}
```

## webpack.config.js

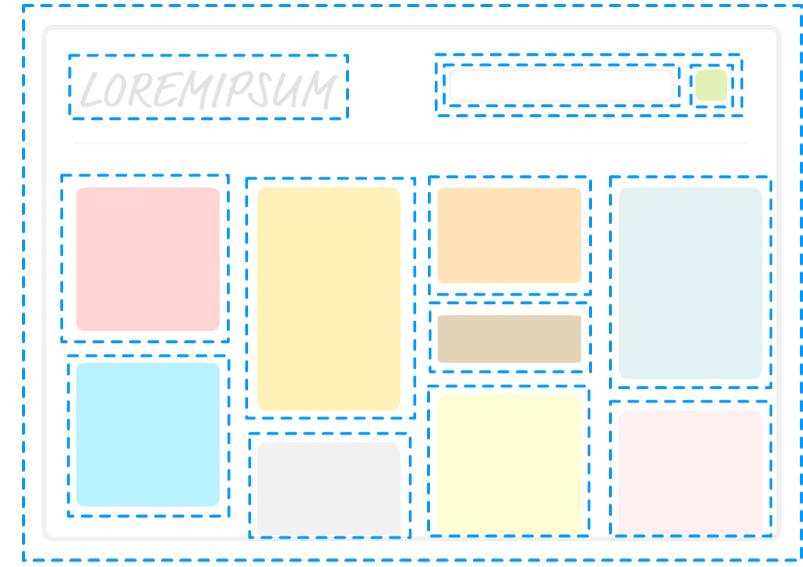
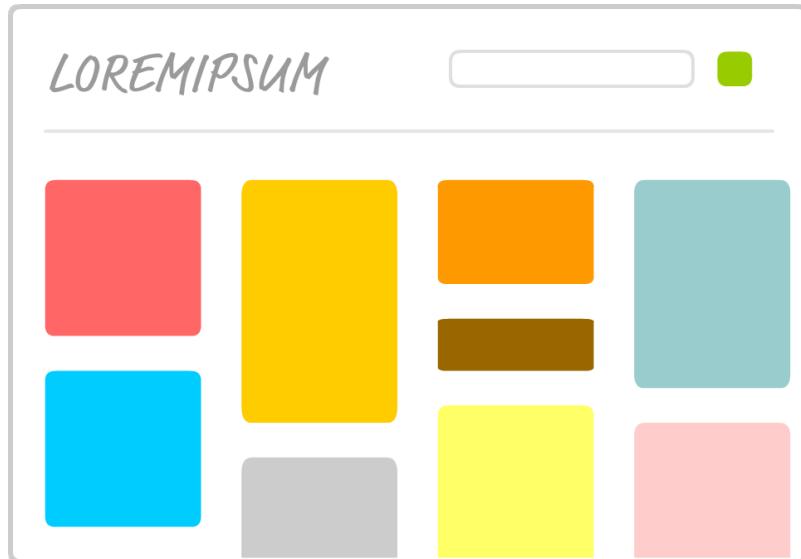
```
var path = require('path');
var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: {
    app: './src/js/app.js'
  },
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  devServer: {
    contentBase: './dist',
    open: true
  },
  module: {
    rules: [
      { test: /\.css$/, use: ['style-loader', 'css-loader'] },
      { test: /\.svg|ttf|woff|woff2|eot$/i, loader: 'url?limit=5000' },
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: [
              '@babel/preset-env',
              '@babel/preset-react'
            ]
          }
        }
      },
      {
        test: /\.html$/,
        use: [
          {
            loader: 'html-loader',
            options: {
              minimize: true
            }
          }
        ]
      }
    ],
    plugins: [
      new HtmlWebpackPlugin({
        template: './src/index.html'
      })
    ]
  };
};
```

Components in React

# 리액트 컴포넌트

# UI 단위 - 컴포넌트



*That's a lot of COMPONENTS!*

# 컴포넌트 만들기

---

## ✓ Hello World! 컴포넌트 만들기

- 각 컴포넌트는 별도의 파일로 만든다.
- 컴포넌트의 이름은 반드시 대문자로 시작되어야 한다.
- 컴포넌트 render() 에는 하나의 태그만 위치

## ES6 스타일 컴포넌트 (not ES5)

```
export default class App extends React.Component {  
  render() {  
    return <h1>Hello, World</h1>;  
  }  
}
```

# React = f(props, states)

---

- props - 컴포넌트에 인자를 넘겨주자!!
  
- 컴포넌트 내부에서는
  - this.props.\* 로 접근
  - this.props.children 이라는 속성을 통해 자식 엘레먼트에 접근

# JSX

---

- Templates 스타일이 아님.
- 옵션이지만, 다들 사용
- JavaScript XML - JSX 자체는 문법
- 리액트에서는 JSX.Element 로 그려질 컴포넌트를 표현합니다.
- React.createElement 함수를 통해서도 JSX.Element 를 만들 수 있습니다.

props 실습

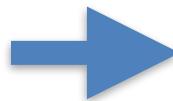
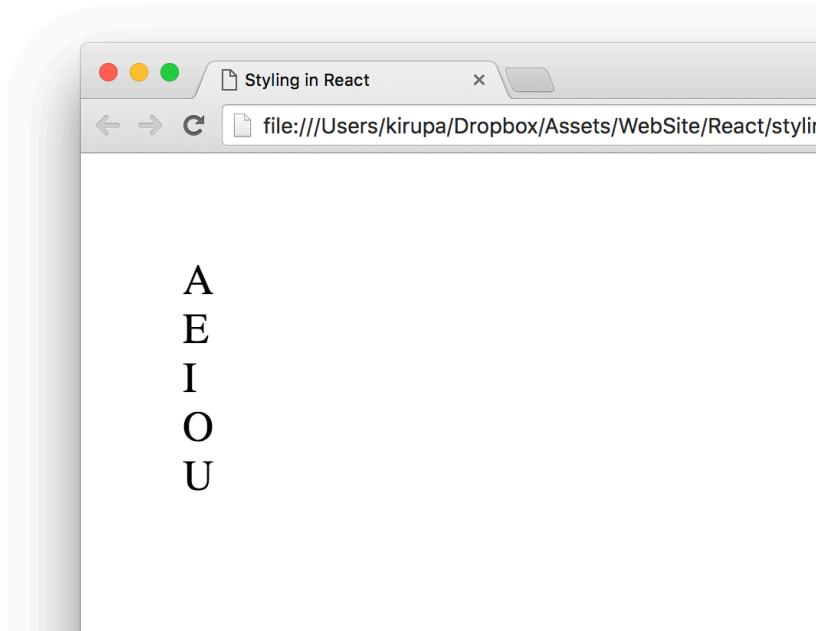
Styling in React

# 리액트 스타일링

# 리액트 컴포넌트에서 CSS 사용

## ☒ 목표

- CSS 스타일을 사용해야 함



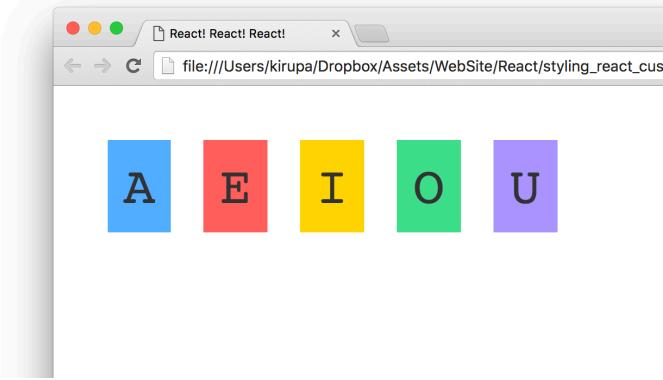
```
div div div {  
  padding: 10px;  
  margin: 10px;  
  background-color: #ffdde00;  
  color: #333;  
  display: inline-block;  
  font-family: monospace;  
  font-size: 32px;  
  text-align: center;  
}
```

# 스타일을 객체로 설정

- style 속성으로 설정
  - 일종의 inline-style
- JSX에서 class 속성은
  - className 으로
- id는 id 로 그대로 사용

```
class Letter extends React.Component {  
  render() {  
    var letterStyle = {  
      padding: 10,  
      margin: 10,  
      backgroundColor: "#ffde00",  
      color: "#333",  
      display: "inline-block",  
      fontFamily: "monospace",  
      fontSize: 32,  
      textAlign: "center"  
    };  
  
    return (  
      <div style={letterStyle}>  
        {this.props.children}  
      </div>  
    );  
  }  
}
```

# CSS 속성 동적으로 설정



```
ReactDOM.render(  
  <div>  
    <Letter bgcolor="#58B3FF">A</Letter>  
    <Letter bgcolor="#FF605F">E</Letter>  
    <Letter bgcolor="#FFD52E">I</Letter>  
    <Letter bgcolor="#49DD8E">O</Letter>  
    <Letter bgcolor="#AE99FF">U</Letter>  
  </div>,  
  destination  
) ;
```

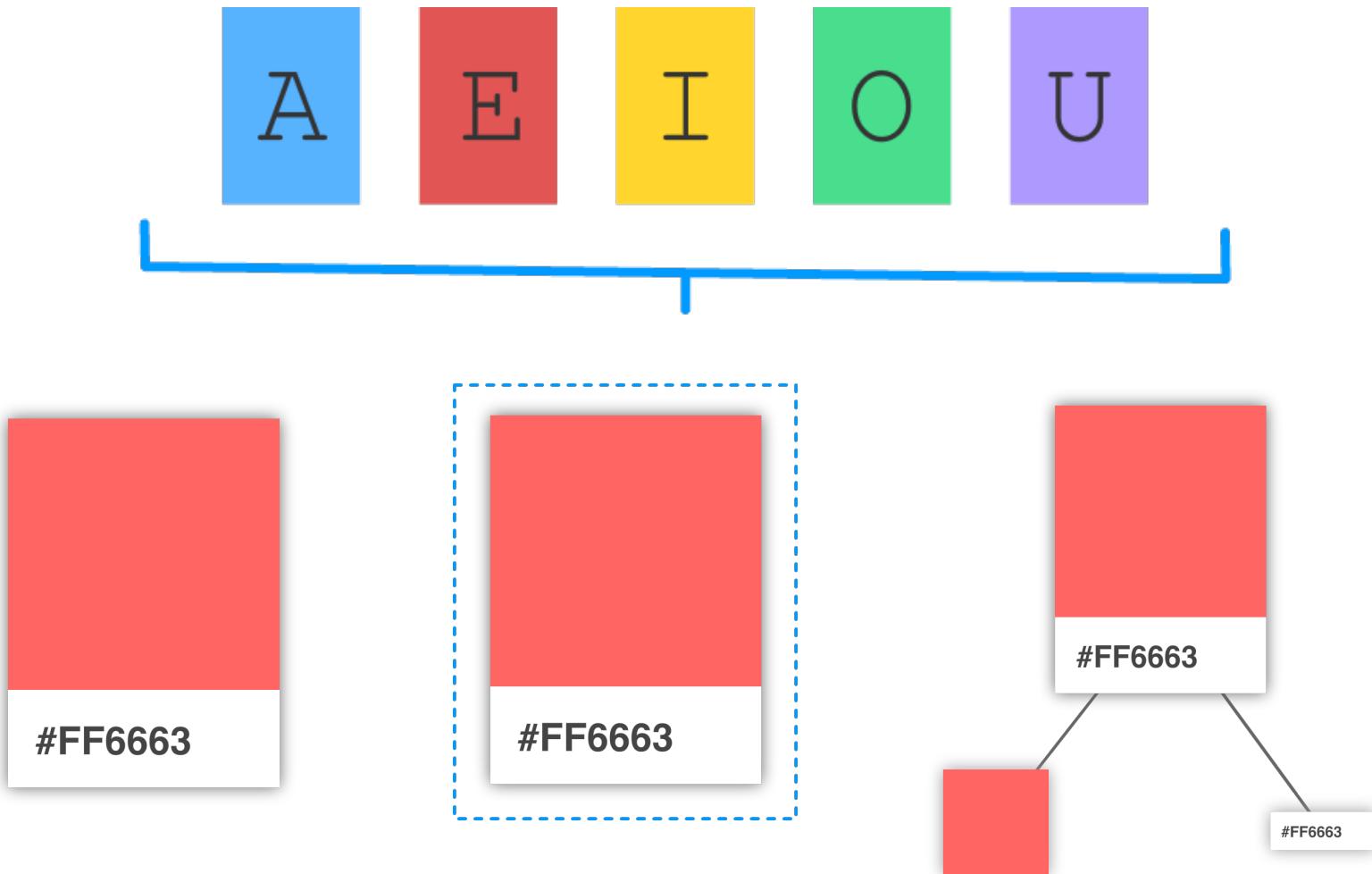
## 스타일을 변수로 설정

```
var letterStyle = {  
  padding: 10,  
  margin: 10,  
  backgroundColor: this.props.bgcolor,  
  color: "#333",  
  display: "inline-block",  
  fontFamily: "monospace",  
  fontSize: 32,  
  textAlign: "center"  
} ;
```

Creating Complex Components

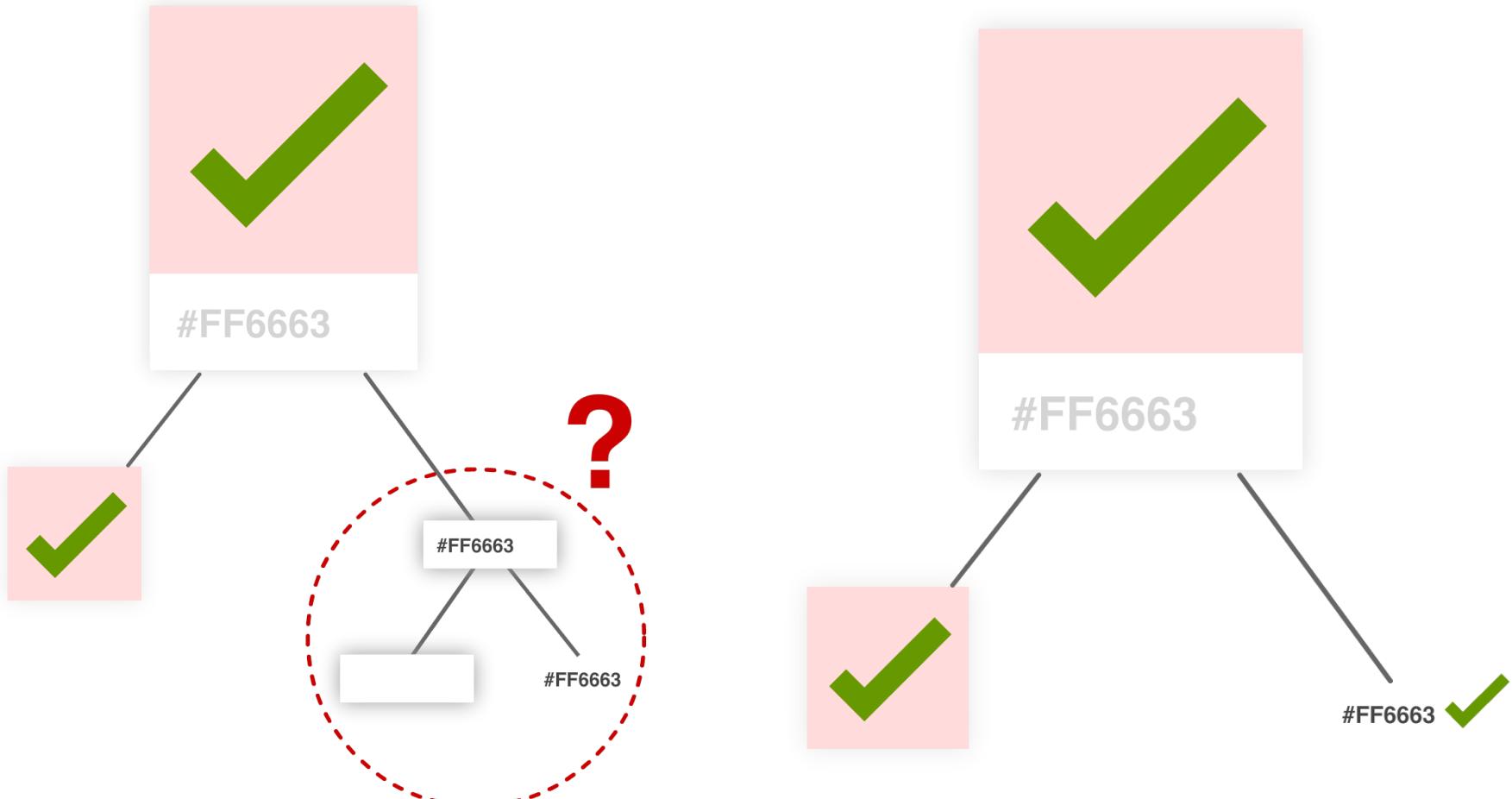
# **복합 컴포넌트**

## 비주얼 요소에서 컴포넌트로



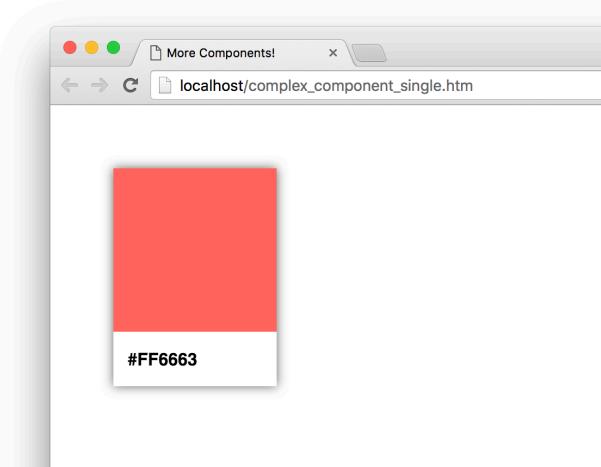
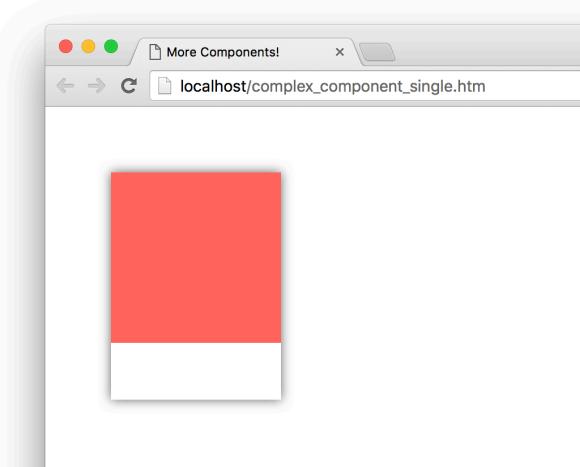
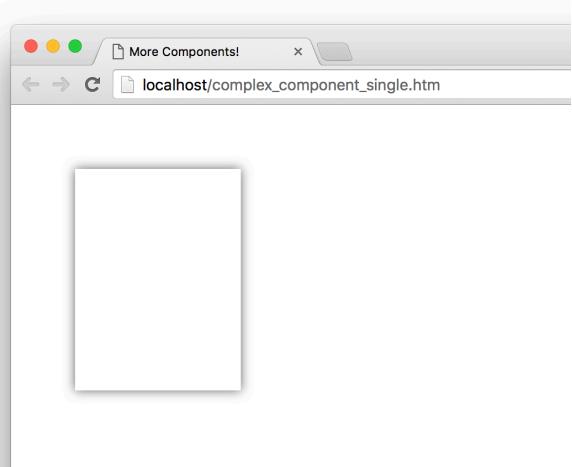
주된 비주얼 요소를 식별하여 컴포넌트를 식별/분리

# 컴포넌트 설계



컴포넌트는 여러 컴포넌트가 모여서 하나의 컴포넌트가 된다.  
컴포넌트는 계층적으로 구성된다.

# 컴포넌트 작성

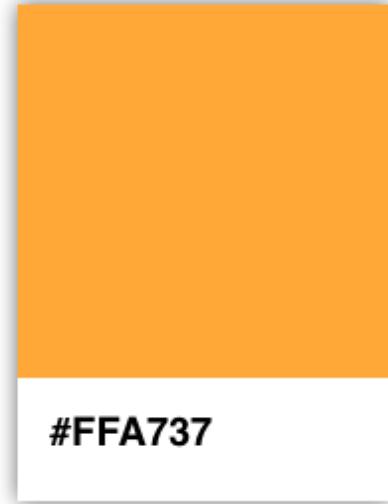


```
class Square extends React.Component {  
  render() {  
    var squareStyle = {  
      height: 150,  
      backgroundColor: this.props.color  
    };  
  
    return (  
      <div style={squareStyle}>  
        </div>  
    );  
  }  
}
```

```
class Label extends React.Component {  
  render() {  
    var labelStyle = {  
      fontFamily: "sans-serif",  
      fontWeight: "bold",  
      padding: 13,  
      margin: 0  
    };  
  
    return (  
      <p style={labelStyle}>{this.props.color}</p>  
    );  
  }  
}
```

# 컴포넌트 작성

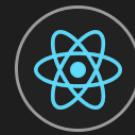
## ☒ <Card /> 컴포넌트



#FFA737

```
class Card extends React.Component {
  render() {
    var cardStyle = {
      height: 200,
      width: 150,
      padding: 0,
      backgroundColor: "#FFF",
      WebkitFilter: "drop-shadow(0px 0px 5px #666)",
      filter: "drop-shadow(0px 0px 5px #666)"
    };

    return (
      <div style={cardStyle}>
        <Square color={this.props.color} />
        <Label color={this.props.color} />
      </div>
    );
  }
}
```



## Props vs State

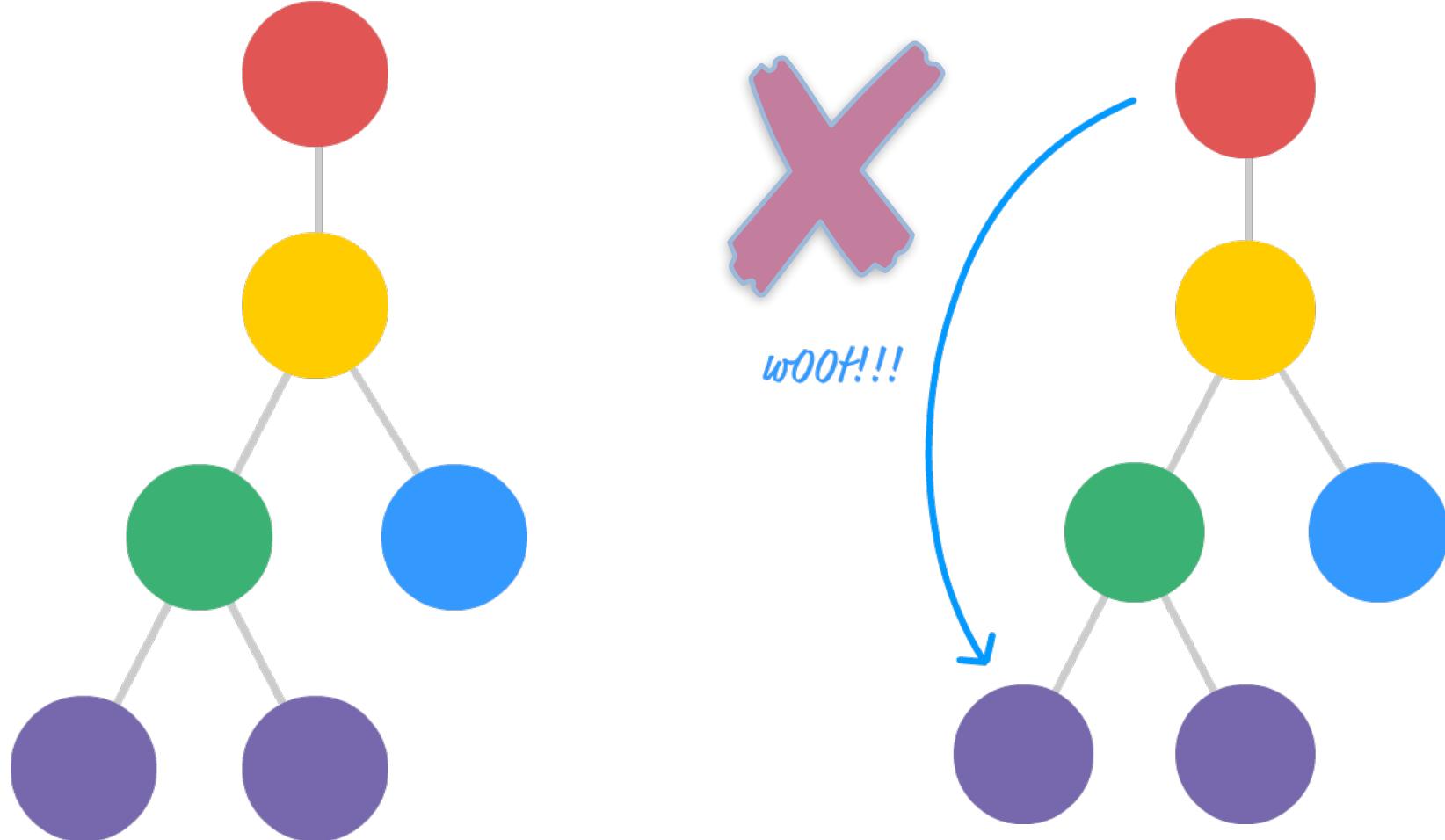
- ✓ props are read-only
- ✓ props can not be modified
- ✓ state changes can be asynchronous
- ✓ state can be modified using `this.setState`

Transferring *Props*.

속성 전달

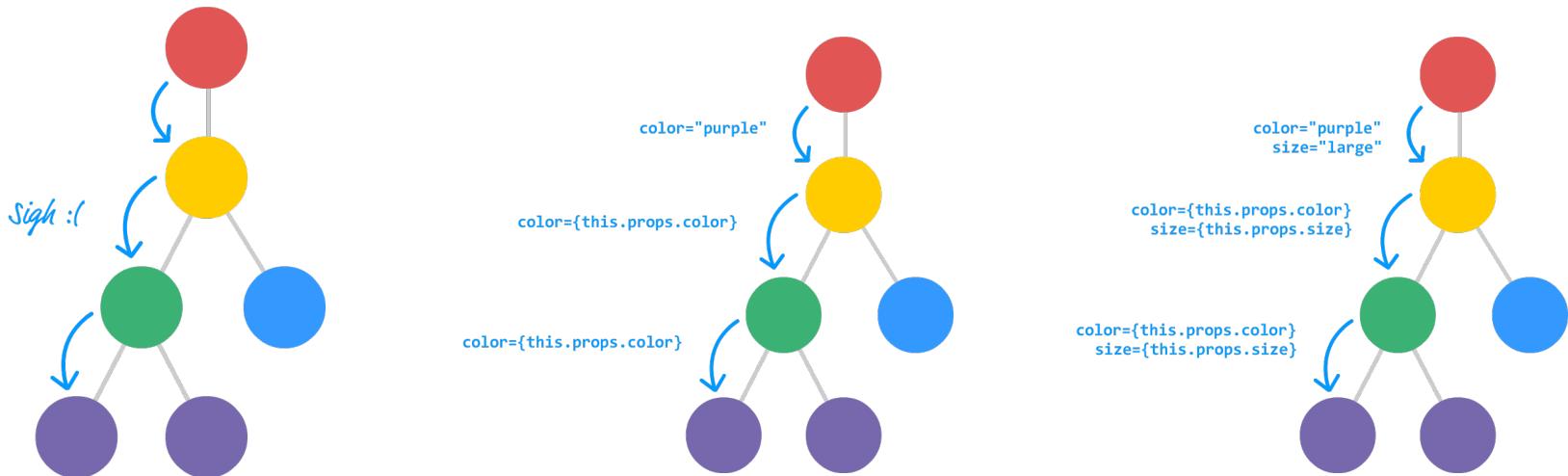
# 컴포넌트에게 속성(Props)을 전달

- ☑ 컴포넌트 간 속성 전달은 반드시 부모에서 자식 컴포넌트로 만 가능



# 컴포넌트에게 속성(Props)을 전달

- ☑ 컴포넌트 계층구조에서 다른 컴포넌트에게 속성을 전달하려면 어떻게 해야 할까?
  - ES6의 스프레드 연산자를 사용한다.



# 컴포넌트에게 속성을(Props)을 전달

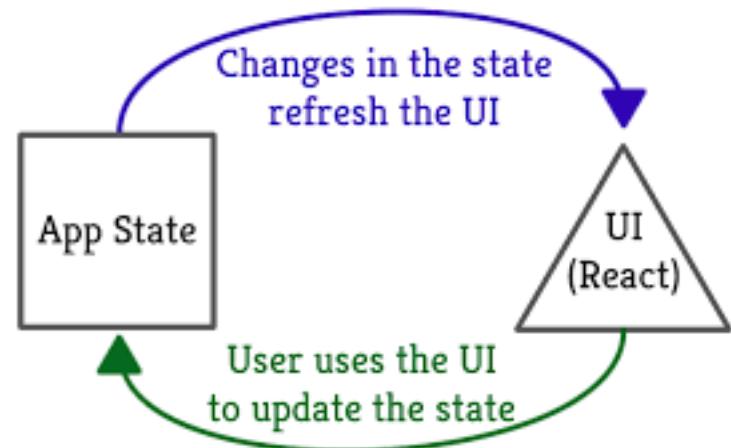
- Props 는 내부적으로 객체로 전달됨
- 스프레드 연산자를 통해서 전달 가능

```
var props = {  
  color: "steelblue",  
  num: "3.14",  
  size: "medium"  
};
```

```
<Display color={this.props.color}  
         num={this.props.num}  
         size={this.props.size}/>
```



```
<Display {...this.props}/>
```



Dealing with *State*

상태 다루기

## State, 상태

---

- ☑ Props 는 부모로 부터 전달되고, Immutable 하다.
  
- ☑ 컴포넌트는 데이터를 사용한다.
  - 서버로 부터 데이터를 가져와서 화면을 구성한다.
  - 화면이 변경되는 것은 화면의 데이터가 변할 때이다.
  - 이 데이터를 컴포넌트(화면)의 상태(State)라고 한다.

지금 까지의 컴포넌트는 상태가 없는 (Stateless) 컴포넌트  
이번 장에서는 상태 있는 (Stateful) 컴포넌트를 다룬다.

## 리액트에서의 상태(State) 관리

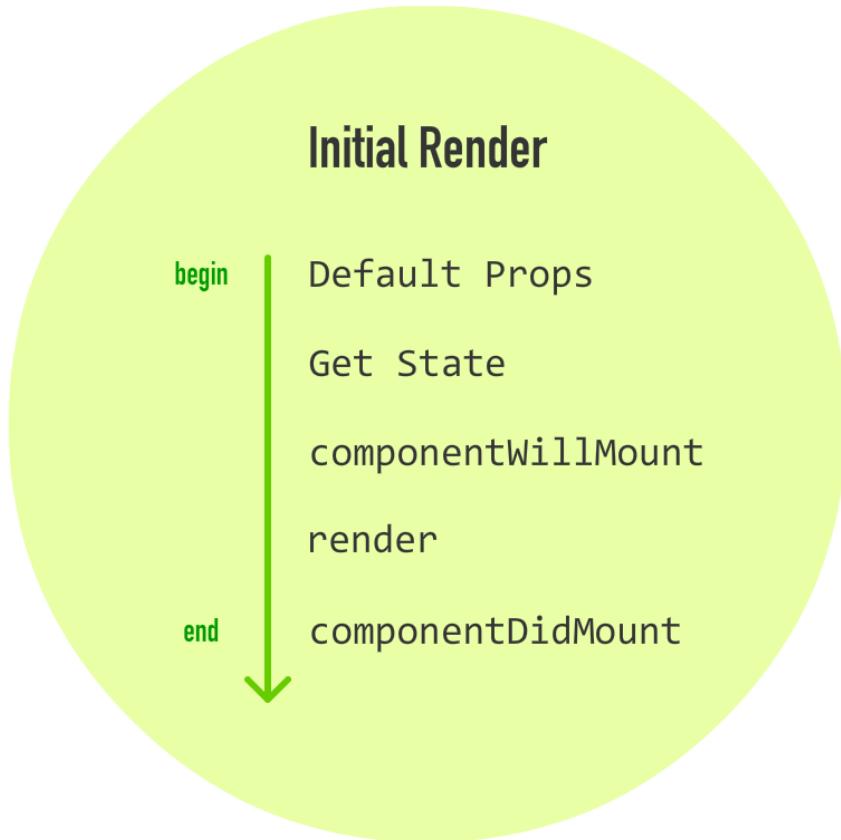
---

- ☒ state 는 생성자에서 초기화 한다.
- ☒ state 를 변경하는 메서드
  - useState
- ☒ state 를 변경하면 뷰가 자동으로 갱신된다.

Component Lifecycle

# 컴포넌트 라이프 사이클

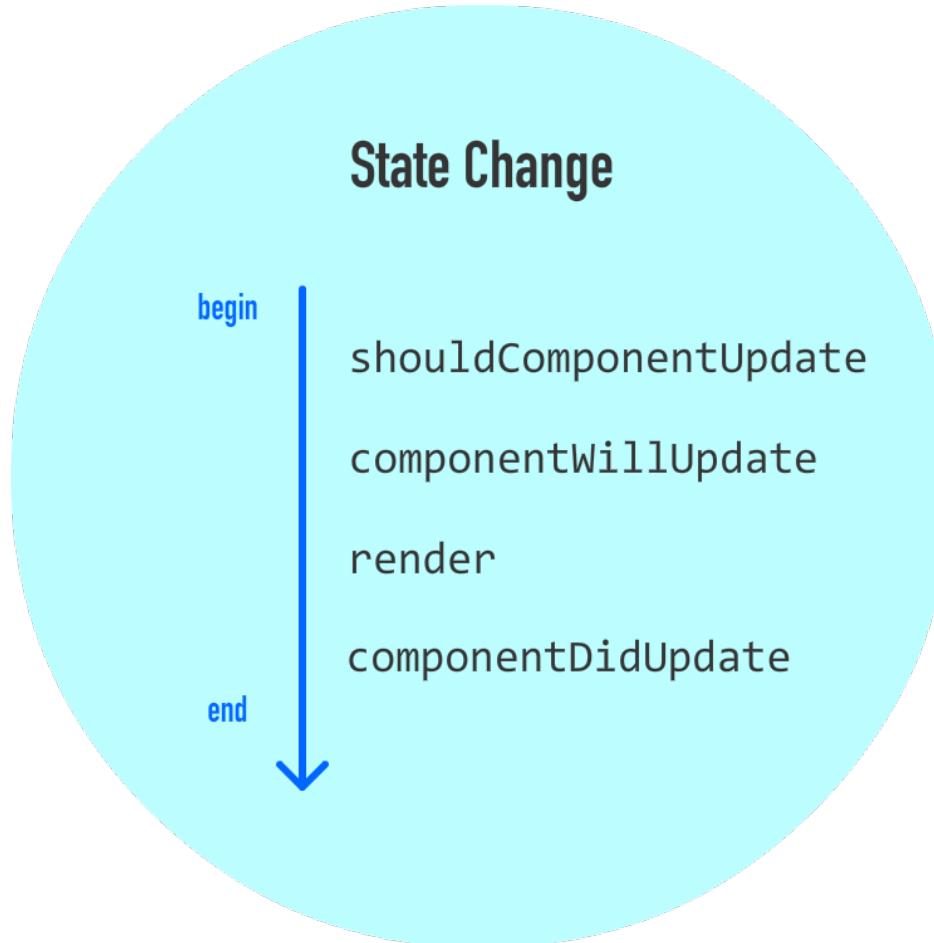
# Component 생성 및 마운트



A screenshot of the Chrome DevTools Console tab is shown, illustrating the output of the component lifecycle events. The console interface includes tabs for Elements, Console, Sources, Network, and Performance. The Console tab is active, showing a list of log messages. The messages are: "defaultProps: Default prop time!", "constructor: Default state time!", "componentWillMount: Component is about to mount!", and "componentDidMount: Component just mounted!". Each message is preceded by a small blue arrow icon.

```
defaultProps: Default prop time!
constructor: Default state time!
componentWillMount: Component is about to mount!
componentDidMount: Component just mounted!
```

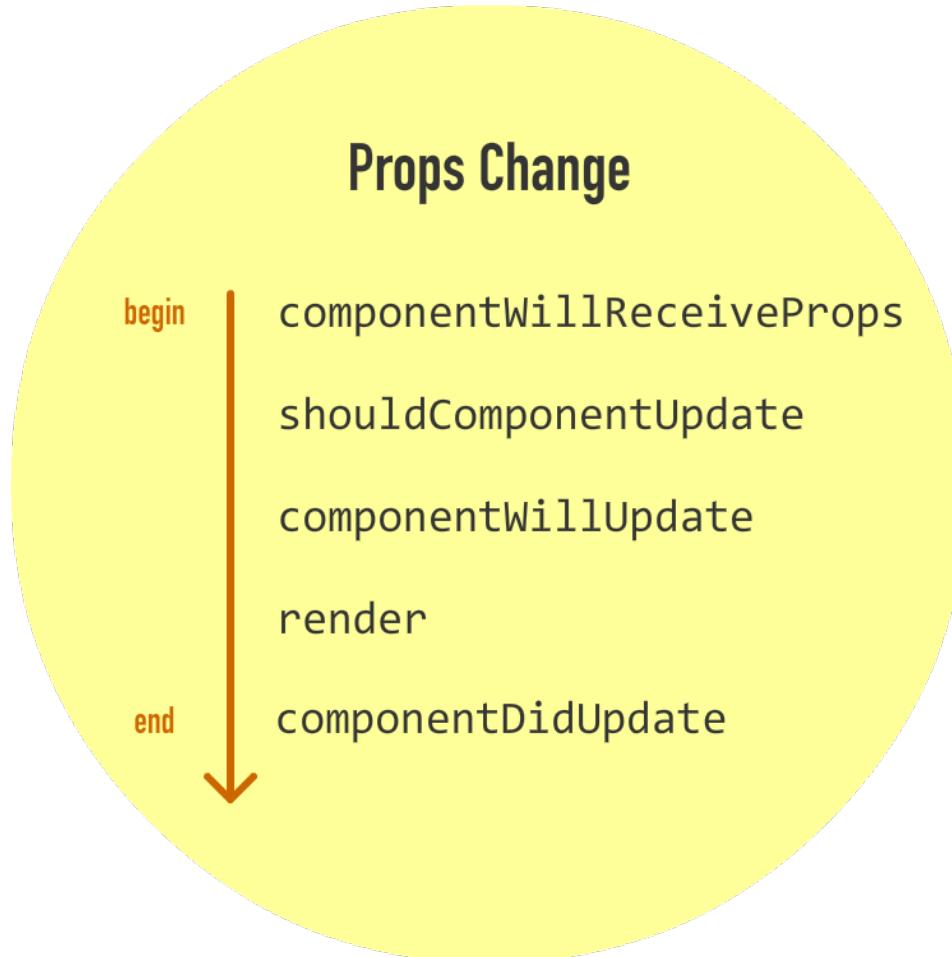
# Component state 변경



A screenshot of a browser's developer tools showing the 'Console' tab selected. The console output shows the following log messages:

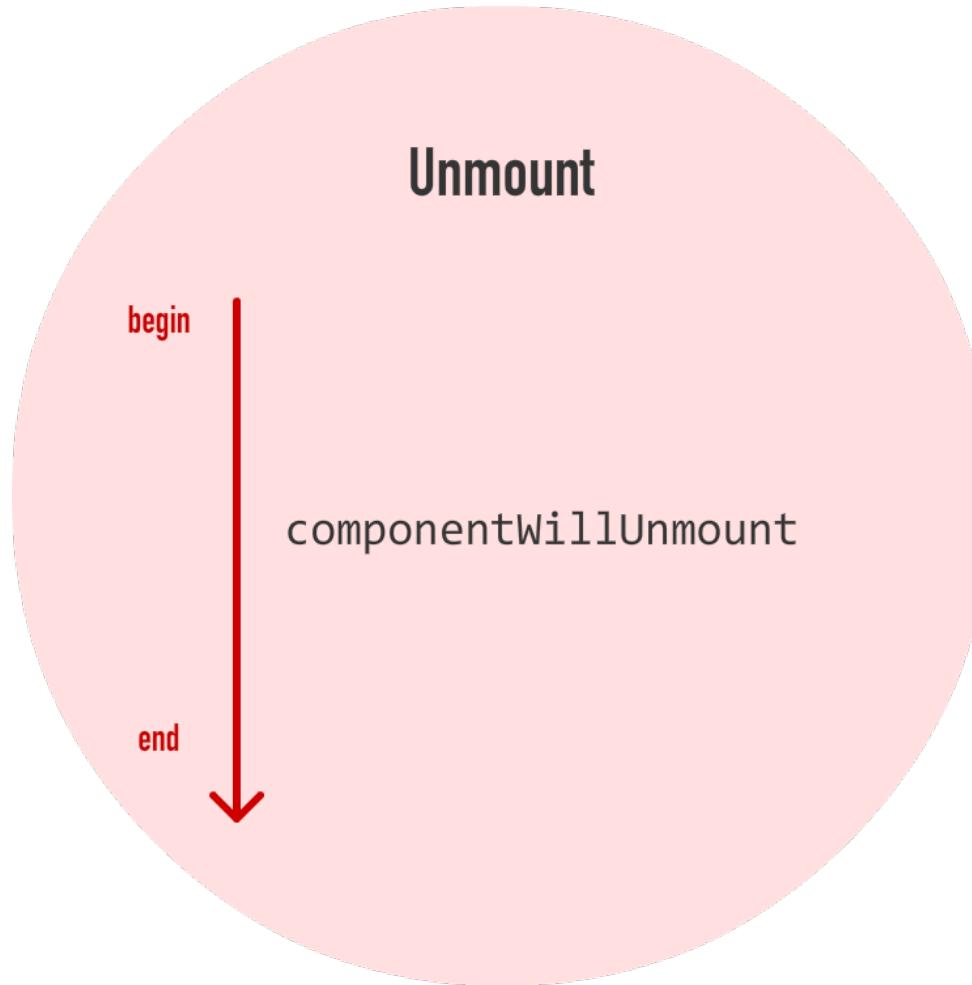
```
componentWillMount: Component is about to mount!
componentDidMount: Component just mounted!
shouldComponentUpdate: Should component update?
shouldComponentUpdate: Component should update!
componentWillUpdate: Component is about to update!
componentDidUpdate: Component just updated!
```

# Component props 변경



# Component unmount

---



Event in React

이벤트

# DOM onclick => JSX onClick

## Synthetic Event

- 리액트는 DOM 이벤트를 래핑하여 React 만의 방식으로 처리
- 다른 브라우저에서도 동일한 속성을 사용하기 위한 방식

## ○ Performance!!

```
return (
  <div style={backgroundStyle}>
    <Counter display={this.state.count} />
    <button onClick={this.increase}
      style={buttonStyle}>+</button>
  </div>
);
```

### Supported Events

React normalizes events so that they have consistent properties across different browsers.

The event handlers below are triggered by an event in the bubbling phase. To register an event handler for the capture phase, append `Capture` to the event name; for example, instead of using `onClick`, you would use `onClickCapture` to handle the click event in the capture phase.

- [Clipboard Events](#)
- [Composition Events](#)
- [Keyboard Events](#)
- [Focus Events](#)
- [Form Events](#)
- [Mouse Events](#)
- [Selection Events](#)
- [Touch Events](#)
- [UI Events](#)
- [Wheel Events](#)
- [Media Events](#)
- [Image Events](#)
- [Animation Events](#)
- [Transition Events](#)
- [Other Events](#)

# 이벤트 처리 방식

## ✓ 이벤트 핸들러를 함수로 지정

- increase 함수에서 파라미터로 이벤트 객체를 받을 수 있음
- 이벤트 핸들러 함수의 this 를 바인딩 해줘야 함

```
class CounterParent extends React.Component {  
  constructor(props) {  
    super(props);  
  
    this.state = {  
      count: 0  
    };  
    this.increase = this.increase.bind(this);  
  }  
  increase(e) {  
    this.setState({  
      count: this.state.count + 1  
    });  
  }  
  render() {  
    ...  
  }  
}
```

Building an Awesome Todo List App in React

# **TO-DO APP 만들기**

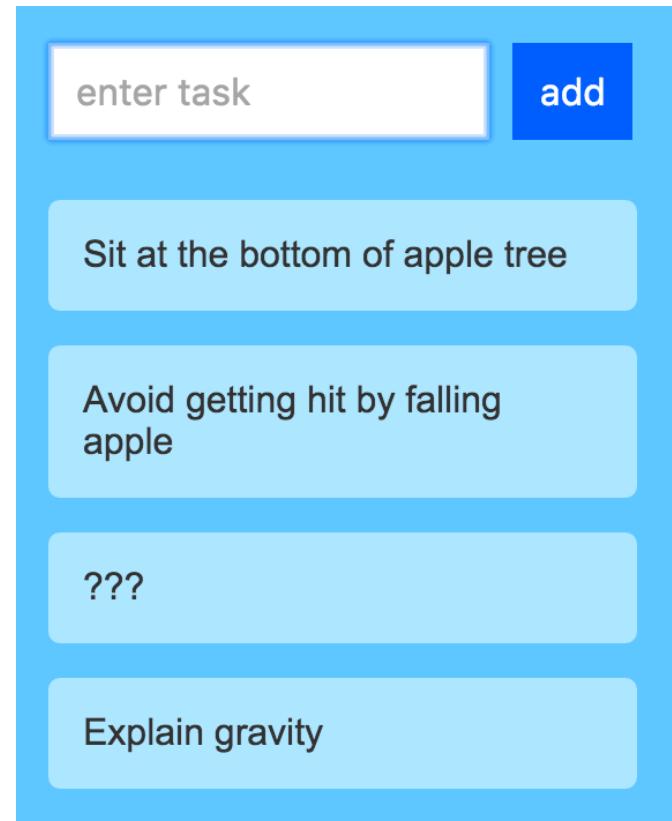
# To-Do App :: 지금까지 배운 내용 총정리

## 기능

- todo 입력
  - 텍스트박스에 글 입력 후 엔터키
- todo 삭제
  - 글 박스 클릭

## 기타

- 스타일링
- 글 등록/삭제 시 애니메이션

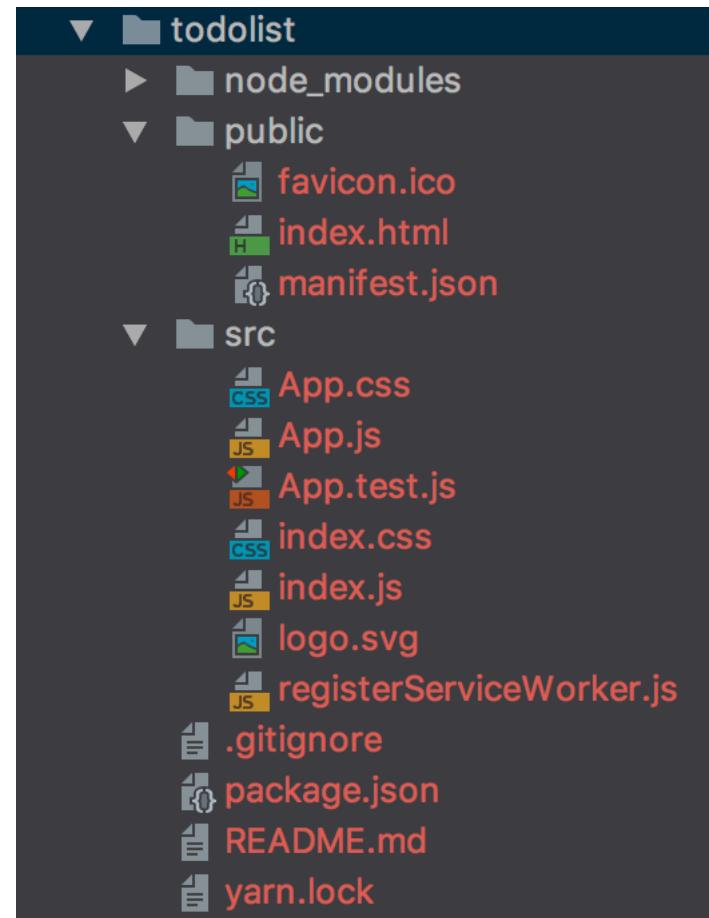


# create-react-app

## ☑ 프로젝트 스캐폴딩 - create-react-app

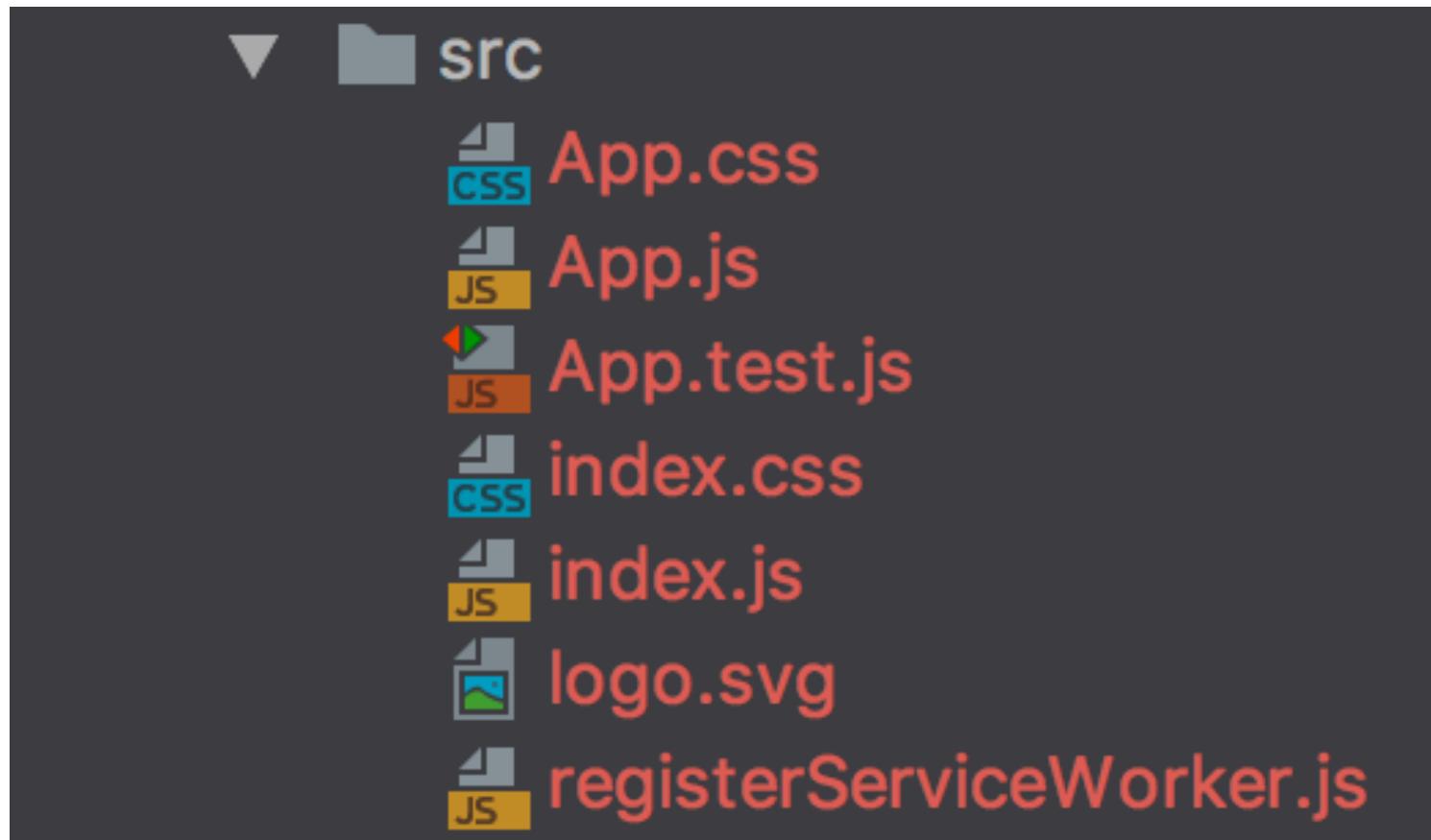
- 지금까지 best practice 보일러 플레이트 사용 개발
- 리액트에서 공식적으로 프로젝트 스캐폴딩 툴 제공
- CLI 방식

```
$ npm i -g create-react-app  
$ create-react-app todolist  
$ cd todolist
```



## src 디렉토리에서 코딩을 시작

---



## src 디렉토리

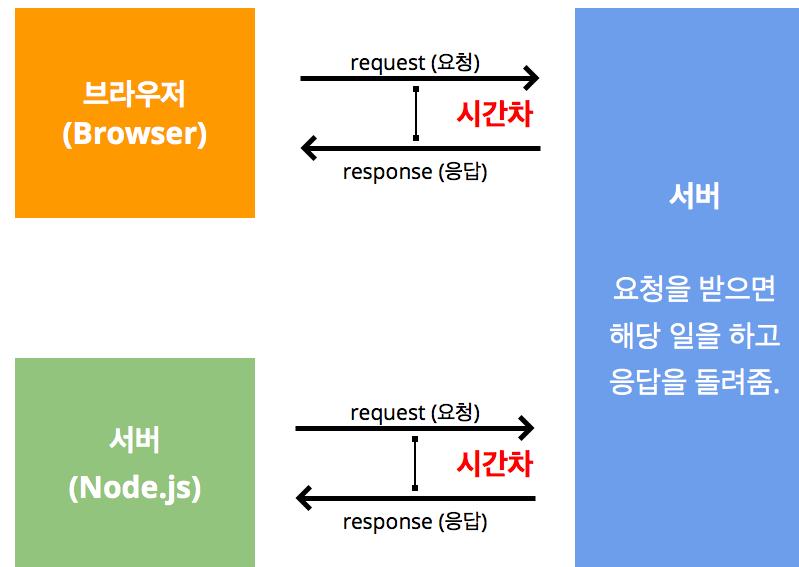
---

- index.js
  - 메인 엔트리 파일
  - 꼭대기에서 ReactDOM.render 를 수행
  - pwa 를 위한 서비스 워커 등록 작업
- index.css
  - 글로벌 스타일 작성 => 프로그래밍 적으로 제한되지 않는다.
- App.js
  - App 컴포넌트 (샘플 컴포넌트)
  - 클래스 이름과 파일 이름을 맞추는 것이 관례
- App.css
  - App 컴포넌트에서 쓰이는 스타일 => 일종의 암묵적 합의
  - App.test.js
- App 컴포넌트에 대한 테스트 작성 파일
- registerServiceWorker.js
  - pwa 서비스 워커 사용 등록 => pwa...

Http Client

# **HTTP CLIENT**

# 요청과 응답



# node 에서의 http client

## node.js

The screenshot shows a browser window displaying the Node.js API documentation for the 'HTTP' module. The URL is [https://nodejs.org/dist/latest-v8.x/docs/api/http.html#http\\_http](https://nodejs.org/dist/latest-v8.x/docs/api/http.html#http_http). The page title is 'HTTP'. A green header bar indicates 'Stability: 2 - Stable'. The main content area starts with a note: 'To use the HTTP server and client one must `require('http')`'. It explains that the HTTP interfaces are designed to support many features of the protocol which have been traditionally difficult to use, such as streaming data. Below this, it shows an example of an HTTP message header object:

```
{ 'content-length': '123',
  'content-type': 'text/plain',
  'connection': 'keep-alive',
  'host': 'mysite.com',
  'accept': '*/*' }
```

It notes that keys are lowercased and values are not modified. The page continues with information about the low-level nature of the API, handling streams, and parsing headers. It also mentions the `rawHeaders` property and provides an example of raw headers:

```
[ 'Content-Length', '123456',
  'content-LENGTH', '123',
  'content-type', 'text/plain',
  'CONNECTION', 'keep-alive',
  'Host', 'mysite.com',
  'accept', '*/*' ]
```

At the bottom, there is a section for the `http.Agent` class, noting its addition in v0.3.4 and its role in maintaining connection persistence.

# browser javascript에서의 http client

## browser

The screenshot shows a browser window displaying the MDN web docs page for XMLHttpRequest. The URL in the address bar is <https://developer.mozilla.org/ko/docs/XMLHttpRequest>. The page title is "XMLHttpRequest". The content discusses the XMLHttpRequest object, mentioning it's available in Microsoft's JavaScript engine and Mozilla's Gecko engine. It includes a code example for creating an XMLHttpRequest instance and sending a GET request to mozilla.org. A note at the bottom states that the code is synchronous and should not be used in UI code. Another section, "비동기 사용", describes using XMLHttpRequest in an asynchronous manner.

XMLHttpRequest는 Microsoft가 만든 JavaScript 객체(object)입니다. 후에 Mozilla도 이것을 받아 들었습니다. XMLHttpRequest는 HTTP를 통해서 쉽게 데이터를 받을 수 있게 해줍니다. 이름과는 좀 동떨어지기도 XML 문서 이상의 용도로 쓰일 수 있습니다. Gecko에서 이 객체는 `nsIJSXMLHttpRequest`와 `nsIXMLHttpRequest` 인터페이스를 구현한 객체입니다. 최신 버전 Gecko에서 이 객체에 변경 사항이 좀 있었었습니다. XMLHttpRequest changes for Gecko1.8을 보십시오.

### 기본 사용

XMLHttpRequest의 사용법은 아주 간단합니다. 이 객체의 인스턴스를 만들고, URL을 알고, 요청을 보내면 됩니다. 그 후에는 인스턴스의 결과 문서와 HTTP 상태 코드를 사용할 수 있게됩니다.

### 예

```
1 | var req = new XMLHttpRequest();
2 | req.open('GET', 'http://www.mozilla.org/', false);
3 | req.send(null);
4 | if(req.status == 200)
5 |   dump(req.responseText);
```

참고: 이 예제는 동기적으로 동작하므로 이 함수를 JavaScript에서 호출하면 UI가 멈춥니다. 실제 제품 코드에서는 이 코드를 사용하지 마십시오.

### 비동기 사용

XMLHttpRequest를 확장 기능에서 사용하려면 반드시 비동기적으로 동작하도록 해야합니다. 비동기적으로 사용할 때, 데이터가 오면 콜백을 받게 됩니다. 이로써 브라우저가 우리 요청을 처리하는 동안에도 평상시처럼 계속 동작하게 됩니다.

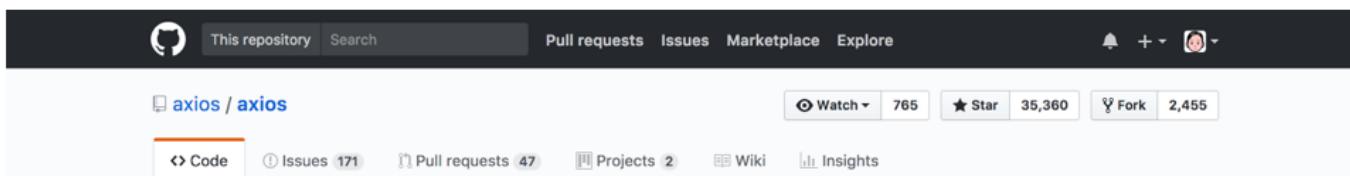
# Universal or Isomorphic

---

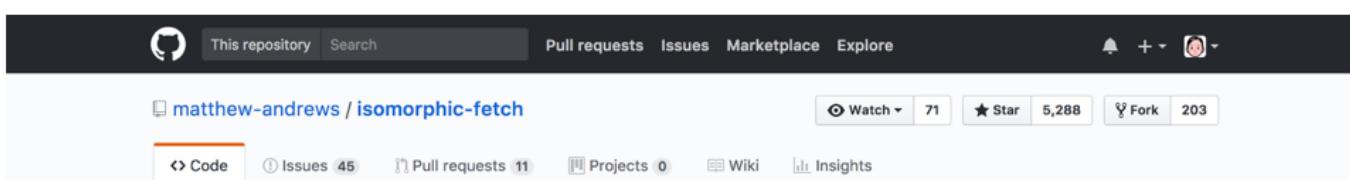
- 복잡하고 사용하기 어려운 XMLHttpRequest 를 편하게
- 서버와 브라우저에서 동시에 사용하도록 처리
  - Universal JavaScript
  - Isomorphic JavaScript
    - [https://en.wikipedia.org/wiki/Isomorphic\\_JavaScript](https://en.wikipedia.org/wiki/Isomorphic_JavaScript)
  - Universal vs Isomorphic
    - <https://github.com/facebook/react/pull/4041>

# axios vs. isomorphic-fetch vs. superagent

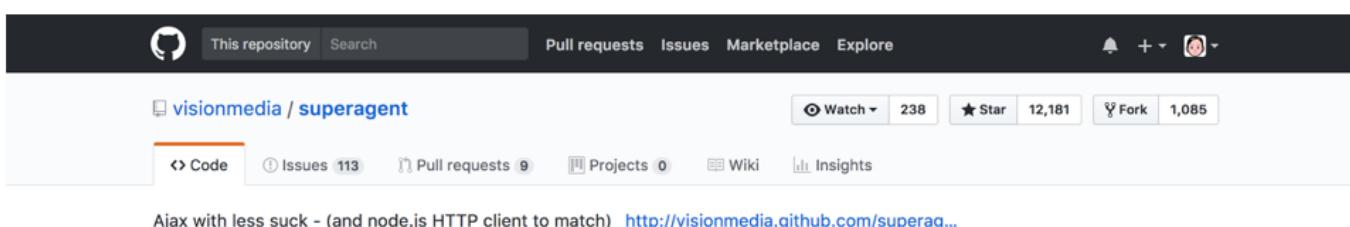
**axios ★ 35360**



This screenshot shows the GitHub repository page for 'axios / axios'. The repository has 35,360 stars and 2,455 forks. It features tabs for Code, Issues (171), Pull requests (47), Projects (2), Wiki, and Insights. The description is 'Promise based HTTP client for the browser and node.js'.



This screenshot shows the GitHub repository page for 'matthew-andrews / isomorphic-fetch'. The repository has 5,288 stars and 203 forks. It features tabs for Code, Issues (45), Pull requests (11), Projects (0), Wiki, and Insights. The description is 'Isomorphic WHATWG Fetch API, for Node & Browserify'.



This screenshot shows the GitHub repository page for 'visionmedia / superagent'. The repository has 12,181 stars and 1,085 forks. It features tabs for Code, Issues (113), Pull requests (9), Projects (0), Wiki, and Insights. The description is 'Ajax with less suck - (and node.js HTTP client to match) <http://visionmedia.github.com/superag...>'.

**2018.02**

# axis

---

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

# promise code sample

```
export default class AxiosTest extends React.Component {
  componentDidMount() {
    axios
      .get('https://api.github.com/users')
      .then(response => {
        console.log('===== ===== =====');
        console.log(response);
        console.log('===== ===== =====');
      })
      .catch(error => {
        console.log('===== ===== =====');
        console.log(error);
        console.log('===== ===== =====');
      });
  }
  render() {
    return (
      <div>
        <h2>Axios Test</h2>
      </div>
    );
  }
}
```

# async-await code sample

```
export default class AxiosTest extends React.Component {
  async componentDidMount() {
    let response = null;
    try {
      response = await axios.get('https://api.github.com/users');
    } catch (error) {
      console.log('===== ===== =====');
      console.log(error);
      console.log('===== ===== =====');
    }
    if (response !== null) {
      console.log('===== ===== =====');
      console.log(response);
      console.log('===== ===== =====');
    }
  }
  render() {
    return (
      <div>
        <h2>Axios Test</h2>
      </div>
    );
  }
}
```

SPA Deploy

# 클라우드 디플로이

# create-react-app SPA Build

---

- Single Page Application
- npm run build
  - production 모드로 빌드되어, 'build' 파일에 생성
    - service worker 가 디폴트
  - 이렇게 만들어진 파일들을 웹서버를 통해 사용자가 접근할 수 있도록 처리

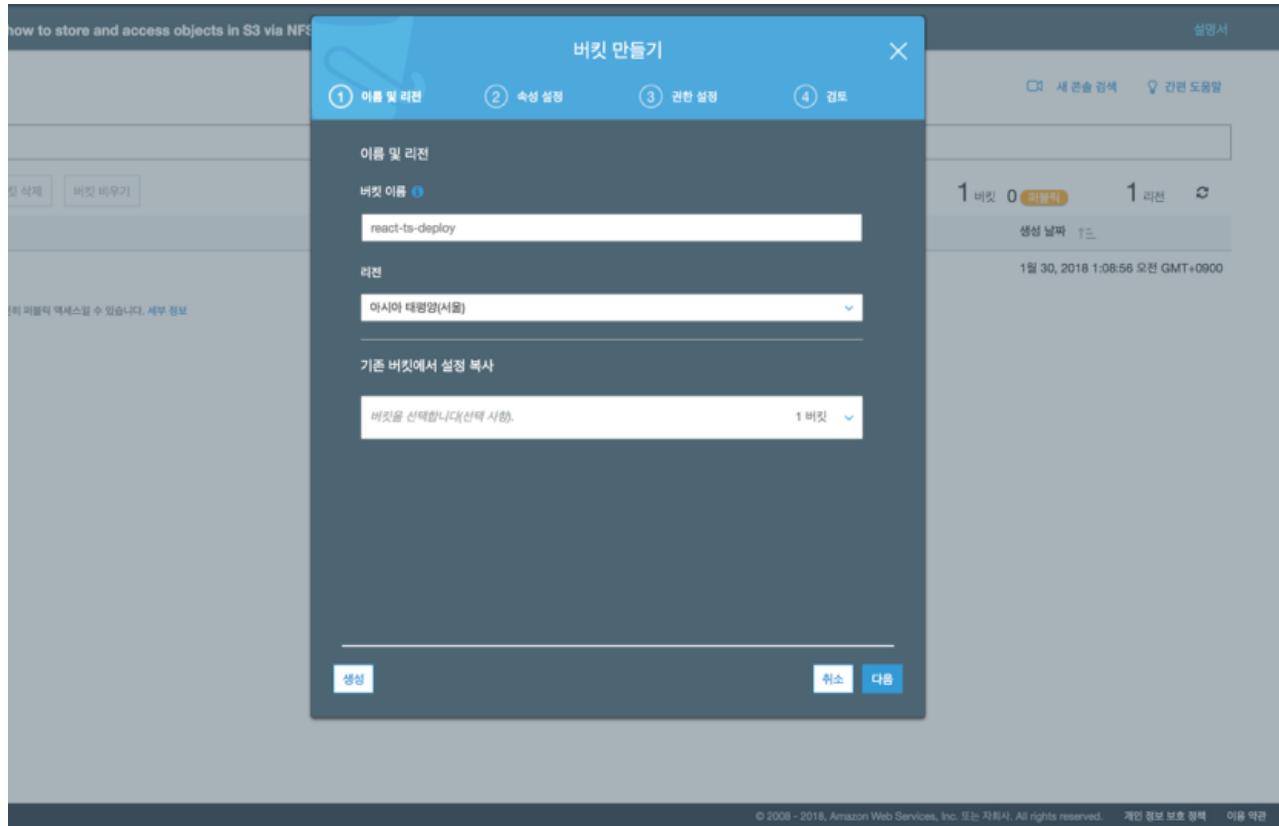
## 배포 방법

---

- serve -s dist
  - <https://github.com/zeit/serve>
- AWS S3 에 배포
- node express

# Amazon S3 정적 웹 사이트 호스팅

## ✓ 버킷 생성



# 버킷 정책 설정

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PublicReadGetObject",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [ "s3:GetObject" ],  
            "Resource": [ "arn:aws:s3:::react-ts-deploy/*" ]  
        }  
    ]  
}
```

Amazon S3 > react-ts-deploy

설정

관리

액세스 제어 목록버킷 정책CORS 구성

**⚠️ 이 버킷에 퍼블릭 액세스 권한이 있을 때**  
이 버킷에 퍼블릭 액세스를 제공했습니다. S3 버킷에 대한 퍼블릭 액세스 권한을 부여하지 않는 것이 좋습니다.

버킷 정책 편집기 ARN: arn:aws:s3:::react-ts-deploy

이전 버킷에 새 정책을 추가하거나 다른 정책을 편집하여 업데이트합니다.

```
1 {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Sid": "PublicReadGetObject",  
6             "Effect": "Allow",  
7             "Principal": "*",  
8             "Action": [  
9                 "s3:GetObject"  
10            ],  
11            "Resource": [  
12                "arn:aws:s3:::react-ts-deploy/*"  
13            ]  
14        }  
15    ]  
16 }
```

삭제취소저장

# 정적 웹사이트 호스팅 설정

The screenshot shows the AWS S3 console interface for a bucket named "react-ts-deploy". The top navigation bar includes tabs for "개요" (Overview), "속성" (Properties), "권한" (Permissions) (which is selected and highlighted in orange), and "관리" (Management). The main content area displays several configuration panels:

- 버전 관리**: Describes versioning within a bucket.
- 서버 액세스 로깅**: Describes logging for server access requests.
- 정적 웹 사이트 호스팅** (Modal):
  - Endpoint: `http://react-ts-deploy.s3-website.ap-northeast-2.amazonaws.com`
  - Selected option:  이 버킷을 사용하여 웹 사이트를 호스팅합니다. [세부 정보](#)
  - Index Document: `index.html`
  - Optional Document: `index.html` (highlighted with a blue border)
  - Prefix: `www/` (disabled)
  - Other options:
    - 요청 리디렉션 [세부 정보](#)
    - 웹 사이트 호스팅 사용 안 함
  - Buttons: [취소](#) and [저장](#)
- 객체 수준 로깅**: Describes CloudTrail event logging for object-level API activities.
- 기본 암호화**: Describes automatic encryption of objects stored in S3.

# build => s3

☒ 빌드 된 모듈들을 s3에 업로드 한다.

The screenshot shows the Amazon S3 console interface. The top navigation bar includes 'Amazon S3' and the bucket name 'react-ts-deploy'. Below the navigation is a toolbar with tabs: '개요' (Overview), '속성' (Properties) (selected), '검색' (Search) (highlighted in orange), and '관리' (Management). A search bar at the top of the main content area contains the placeholder text 'Q... 검색하려면 접두사를 입력하고 Enter 키를 누릅니다. 지우려면 Esc 키를 누릅니다.' (Q... To search, enter a prefix and press Enter. To clear, press Esc.). Below the search bar are three buttons: '업로드' (Upload), '폴더 만들기' (Create Folder), and '더 보기' (More Options). On the right side of the search bar, there is a location indicator '아시아 태평양(서울)' and a refresh icon. The main content area displays a table of objects in the bucket:

선택	이름	마지막 수정	크기	스토리지 클래스
<input type="checkbox"/>	_static	2월 2, 2018 6:00:22 오후 GMT+0900	257.0 B	стандарт
<input type="checkbox"/>	.asset-manifest.json	2월 2, 2018 6:00:21 오후 GMT+0900	3.8 KB	стандарт
<input type="checkbox"/>	.favicon.ico	2월 2, 2018 6:00:22 오후 GMT+0900	548.0 B	стандарт
<input type="checkbox"/>	.index.html	2월 2, 2018 6:00:22 오후 GMT+0900	317.0 B	стандарт
<input type="checkbox"/>	.manifest.json	2월 2, 2018 6:00:22 오후 GMT+0900	3.2 KB	стандарт
<input type="checkbox"/>	.service-worker.js	2월 2, 2018 6:00:22 오후 GMT+0900		

At the bottom of the page, there is a footer bar with the text '작동' (Working), '0 진행 중' (0 in progress), '6 성공' (6 successful), and '0 오류 발생' (0 errors occurred).

## 참고

---

- [https://docs.aws.amazon.com/ko\\_kr/AmazonS3/latest/dev/WebsiteHosting.html](https://docs.aws.amazon.com/ko_kr/AmazonS3/latest/dev/WebsiteHosting.html)
- [https://docs.aws.amazon.com/ko\\_kr/AmazonS3/latest/dev/WebsiteAccessPermissionsReqd.html](https://docs.aws.amazon.com/ko_kr/AmazonS3/latest/dev/WebsiteAccessPermissionsReqd.html)
- <https://medium.com/@omgwtfmarc/deploying-create-react-app-to-s3-or-cloudfront-48dae4ce0af>

# node.js express로 배포

## server.js

```
const express = require('express');
const path = require('path');
const app = express();

app.use(express.static(path.join(__dirname, 'build')));

// app.get('/', function(req, res) {
app.get('*', function(req, res) {
    res.sendFile(path.join(__dirname, 'build', 'index.html'));
});

app.listen(9000);
```

수고하셨습니다.

김순곤

[soongon@hucloud.co.kr](mailto:soongon@hucloud.co.kr)