

# webProgramming

## ECMA Script(ES6)

in-hee Kim,  
school of Computer Engineering  
[inhee.kim@hansung.ac.kr](mailto:inhee.kim@hansung.ac.kr)



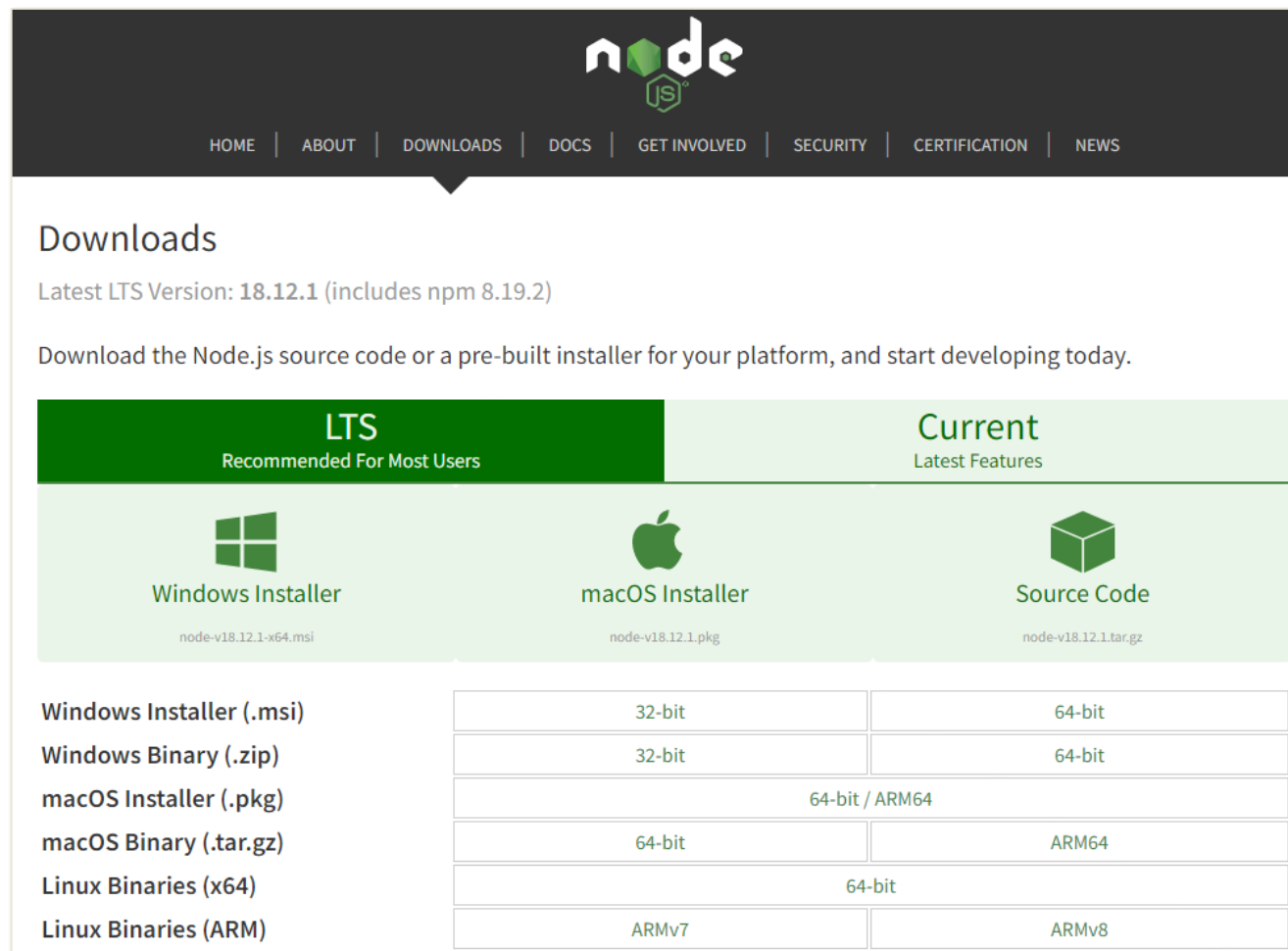
- ES6 이전과 이후의 표현 차이
- nodejs express 서버 구현



- 자바스크립트를 표준화하기 위해 만들어진 언어
- 스크립트 언어의 규칙, 세부사항 및 지침 제공
- 클라이언트 사이드 스크립트, node.js를 사용한 서버 응용 프로그램/서비스
- 1996년 3월. 넷스케이프 네비게이터 2.0 출시하면서 지원 시작
- 1997년 6월. 초판 ECMA-262 제정
- 2015년 6월. ES2015(=ES6)
- 현재 2019년 6월 10판 표준 ES2019 출판됨



## URL : <https://nodejs.org/en/download/>



The screenshot shows the Node.js Downloads page. At the top is the Node.js logo and a navigation bar with links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. Below the navigation bar, the 'Downloads' section is highlighted. It states 'Latest LTS Version: 18.12.1 (includes npm 8.19.2)' and 'Download the Node.js source code or a pre-built installer for your platform, and start developing today.' There are two main tabs: 'LTS Recommended For Most Users' (active) and 'Current Latest Features'. Under the 'LTS' tab, there are three main categories: 'Windows Installer' (node-v18.12.1-x64.msi), 'macOS Installer' (node-v18.12.1.pkg), and 'Source Code' (node-v18.12.1.tar.gz). Below these, there is a table showing the available binaries for each platform.

Platform	Architecture	File Name
Windows Installer (.msi)	32-bit	node-v18.12.1-x64.msi
	64-bit	node-v18.12.1-x64.msi
Windows Binary (.zip)	32-bit	node-v18.12.1-x64.msi
	64-bit	node-v18.12.1-x64.msi
macOS Installer (.pkg)	64-bit / ARM64	node-v18.12.1.pkg
	64-bit	node-v18.12.1.pkg
macOS Binary (.tar.gz)	64-bit	node-v18.12.1.pkg
	ARM64	node-v18.12.1.pkg
Linux Binaries (x64)	64-bit	node-v18.12.1.pkg
	64-bit	node-v18.12.1.pkg
Linux Binaries (ARM)	ARMv7	node-v18.12.1.pkg
	ARMv8	node-v18.12.1.pkg

- **Nodejs**

자바스크립트 엔진으로 빌드된  
이벤트 기반 js 런타임 환경



# Extension 설치 (ES6)

The screenshot shows the Visual Studio Code interface with the 'EXTENSIONS: MARKETPLACE' sidebar on the left and the 'Extension: JavaScript (ES6) code snippets' details on the right.

**Left Panel (EXTENSIONS: MARKETPLACE):**

- Search bar: **es6**
- Extension 1: **JavaScript (ES6) code snippets** (charalampos karypidis) - 10.2M, 5 stars. **Install** button.
- Extension 2: **Babel ES6/ES7** (dzannotti) - 804K, 2.5 stars. **Install** button.
- Extension 3: **React-Native/React/Redux snippets** (EQuimper) - 1M, 5 stars. **Install** button.
- Extension 4: **es6-string-html** - 229K, 5 stars.

**Right Panel (Extension: JavaScript (ES6) code snippets - ECMAScript6 - Visual Studio Code):**

- Icon: JavaScript logo (JS)
- Extension Name: **JavaScript (ES6) code snippets**
- Author: charalampos karypidis
- Downloads: 10,221,337
- Rating: 5 stars
- Description: Code snippets for JavaScript in ES6 syntax
- Install** button (circled in red)
- Buttons: **Details**, **Feature Contributions**
- Section: **Categories**



# Extension 설치 (code runner)

The screenshot shows the Visual Studio Code interface with the Extensions view open. The search bar contains "code runner". The extension list on the left includes "Code Runner" by Jun Han, "Test Runner ..." by Microsoft, and "Runner" by Yasuhiro Matsumoto. The right pane displays the details for the "Code Runner" extension (v0.11.8) by Jun Han, which has 16,225,784 downloads and a 5-star rating. The extension is currently installed and enabled globally. A red circle highlights the play button icon in the top right corner of the extension details pane. The bottom status bar shows "PROBLEMS", "OUTPUT", and "Code" tabs.

EXTENSIONS: M...

code runner

**Code Runner** 1788ms  
Run C, C++, Java, JS, PHP, ...  
Jun Han

**Test Runner ...** 16.6M ★ 4  
Run and debug JUnit or Tes...  
Microsoft Install

**Runner** 67K ★ 3  
Run various scripts.  
Yasuhiro Matsum... Install

**Code Runner** v0.11.8  
Jun Han | 16,225,784 | ★★★★★  
Run C, C++, Java, JS, PHP, Python, Perl, R...  
Disable Uninstall   
This extension is enabled globally.

Details Feature Contributions Changelog Runtime Status

PROBLEMS OUTPUT ... Code



## 화살표 함수 (arrow function) 특징

- function 표현식에 비해 구문이 짧다
- this나 super 객체 바인딩을 하지 않는다
- 기능 정의만 하는 익명함수다
- 반환값만 존재하는 경우 중괄호와 return 생략 가능
- 반환값 이외의 구문이 존재하는 경우 중괄호와 return 필요
- 익명함수 : ( ~, function() { ~ } )
- 람다식 : ( ~, () => { ~ } )



- 함수 선언식

```
function fun(su1, su2) {  
    return su1 + su2;  
}
```

- 함수 표현식

```
const sum = function fun(su1, su2) {  
    return su1 + su2;  
}
```

```
const sum = function (su1, su2) {  
    return su1 + su2;  
}
```





## 화살표 함수 (arrow function)

```
const sum = (su1, su2) => {  
  return su1 + su2;  
}
```



```
const sum = (su1, su2) => su1 + su2;
```

```
const sum = su => su + su;
```

```
const pie = () => 3.14;
```



## 화살표 함수 (arrow function)

```
const student = () => {  
  return {  
    name: "성대",  
    age: 20  
  }  
}
```

```
const student = () => ({  
  name: "성대",  
  age: 20  
}) ;
```

```
const st = student();  
console.log(`name: ${st.name}, age: ${st.age}`); // 템플릿 리터럴
```



```
const v = "global (0)";
```

```
function printVariable() {  
  console.log(this.v);  
}
```

```
const call1 = {  
  v: 'Local (1)',  
  fun: printVariable  
}
```

```
const call2 = {  
  v: 'Local (2)',  
  fun: printVariable  
}
```

```
call1.fun();  
call2.fun();
```

```
const v = "global (0)";
```

```
const printVariable = () => {  
  console.log(v);           // console.log(this.v) -> x  
}
```

```
const call1 = {  
  v: 'Local (1)',  
  fun: printVariable  
}
```

```
const call2 = {  
  v: 'Local (2)',  
  fun: printVariable  
}
```

```
call1.fun();  
call2.fun();
```



# Rest Parameters(나머지 매개변수)

```
function fun(su1, su2, su3, su4) {  
    return su1 + su2 + su3 + su4;  
}
```

```
function fun(...abc) {  
    return abc;  
}
```

```
const fun = (...args) => {  
    return args;  
}  
  
const f = fun(33, 44, 55);  
console.log(f);
```

```
const sum = (...args) => {  
    var result = 0;  
    for(var i = 0; i < args.length; i++) {  
        result += args[i];  
    }  
    return result;  
};  
  
console.log(sum(1, 2, 3, 4, 5));
```



mode.js

```
const hw = function helloworld() {  
    console.log("hello world");  
}  
  
const es = () => {  
    console.log("es world");  
}  
  
const he = () => console.log("hello es6");  
  
hw();  
es();  
he();
```



## mode.js

```
const hw = function helloworld() {  
  console.log("hello world");  
}  
  
const ew = () => {  
  console.log("es world");  
}  
  
const he = () => console.log("hello es6");  
module.exports = ew;
```

## main.js

```
const ew = require('./mode');  
ew()
```



# module(3)

## mode.js

```
const hw = function helloworld() {  
    console.log("hello world");  
}  
  
const ew = () => {  
    console.log("es world");  
}  
  
const he = () => console.log("hello es6");  
  
module.exports = {  
    hw,  
    ew,  
    he  
};
```

## main.js

```
const {  
    hw,  
    ew,  
    he  
} = require('./mode');  
  
hw();  
he();
```



```
class Rectangle {  
    constructor(height, width) {  
        this.height = height;  
        this.width = width;  
    }  
}  
  
const r = new Rectangle(5, 7);  
console.log(r);  
console.log(`height : ${r.height}, width : ${r.width}`);
```





```
class Rectangle {  
    constructor(height, width) {  
        this.height = height;  
        this.width = width;  
    }  
    // Getter  
    get area() {  
        return this.calcArea();  
    }  
    // 메서드  
    calcArea() {  
        return this.height * this.width;  
    }  
}
```

```
const square1 = new Rectangle(5, 5);  
const square2 = new Rectangle(10, 10);  
  
console.log(square1.area);    // 25  
console.log(square2.area);    // 100
```



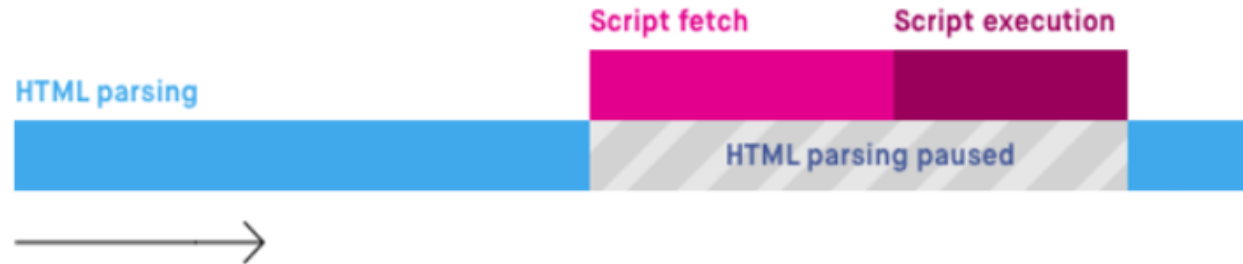
```
const car = {  
  santafe : '산타페',  
  genesis : '제네시스',  
  tesla : '테슬라'  
}  
  
console.log(car.genesis);
```

```
const genesis = '제네시스'  
const tesla = '테슬라'  
  
const car = {  
  genesis,           // genesis: genesis  
  tesla              // tesla: tesla  
}  
  
console.log(car);
```

# 스크립트의 위치?

## 문서 중간

- html 파싱을 멈추고 스크립트 실행

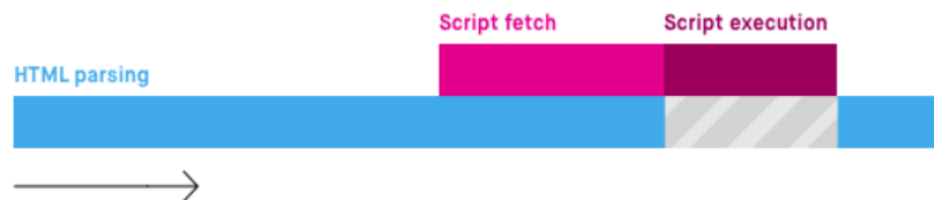


## body 태그 최하단(권장)

- 스크립트로 인한 중단 시점 생성으로 display가 지연되는 것을 방지
- DOM 트리가 생성되기전에 스크립트가 생성되지도 않은 DOM 조작을 시도할 수 있다.

## 스크립트 로딩 순서 제어 : asyc, defer

< asyc >



< defer >



## 브라우저 동작 순서

1. HTML 읽기
2. HTML 파싱
3. DOM 트리 생성
4. Display



잠시 쉬었다가 다음 영상을 이어서 시청하세요

# webProgramming

nodejs express

in-hee Kim,  
school of Computer Engineering  
[inhee.kim@hansung.ac.kr](mailto:inhee.kim@hansung.ac.kr)



# NPM(Node Package Manager) 설치

## > npm init

npm : 라이브러리 설치를 도와주는 도구  
package.json에 설치된 라이브러리 기록

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\webServer\ECMAScript6> npm init
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` 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: (ecmascript6)
```

```
Press ^C at any time to quit.
package name: (todoapp)
version: (1.0.0)
description:
entry point: (index.js) server.js
test command:
git repository:
keywords:
author:
license: (ISC)
```

```
"version": "1.0.0",
"description": "",
"main": "server.js",
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
"author": "",
"license": "ISC"
}
```

```
Is this OK? (yes)
PS C:\Users\harry\Desktop\todoapp>
```



> npm install express

```
PS C:\webServer\ES6> npm install express
[#####.....] / reify:safe-buffer: sill audit bulk request {
npm notice created a lockfile as package-lock.json. You should comm
it this file.
npm WARN todoapp@1.0.0 No description
npm WARN todoapp@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in
2.412s
found 0 vulnerabilities
PS C:\webServer\ES6> 
```



- **localhost** : 컴퓨터 네트워크에서 자신의 컴퓨터 의미
  - ip주소 : 127.0.0.1    혹은    localhost
- **port number** : 어플리케이션 식별 숫자
  - 127.0.0.1 : 5500
- **HTTP 모듈** : 데이터를 주고 받을 수 있는 프로토콜. node에 관련기능 내장됨
- Listen : 클라이언트가 접속할 수 있도록 서버의 대기상태





- NPM

- node 기반 자바스크립트 프로그램을 등록할 수 있는 커뮤니티.

- package.json

- 설치된 모듈 관련 사항 관리하는 파일

- npm init : 노트 프로젝트 폴더 초기화

- npm install 모듈이름 : 모듈 추가



- Express
  - 노드기반 웹 프레임워크
  - http 통신 요청 핸들러 생성
  - 템플릿 데이터 응답을 만들기 위해 view 렌더링 엔진과 결합
  - 접속을 위한 공통 웹 어플리케이션 세팅
  - 추가적인 미들웨어 처리요청

# 서버 동작 코드

وز. -

```
const express = require('express')
const app = express()
const port = 8080

app.get('/', (req, res) => {
  res.send(`<h1 style="color:red;">Hello World!</h1>`)
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
});
```

Handwritten notes: A red box around the port number 8080 with the number 50 written inside. A red box around the console.log message. A red arrow points from the port variable to the port parameter in the listen function. A red arrow points from the require function to the express module name. A red arrow points from the app variable to the app parameter in the get function. A red arrow points from the app variable to the app parameter in the listen function. A red arrow points from the port variable to the port parameter in the listen function. A red arrow points from the console.log function to the message string.

터미널 실행

> node 파일명.js



- PowerShell

- 명령어 > ipconfig

- IPv4 주소 : 192.168.0.xx

}

