



실증 아카데미 리액트 특강


Node.js, npm, yarn의 기본



일시 2021년 01월 12일

주최 동아대학교





CONTENTS—

01 Node.js란?

JavaScript 런타임 환경
Node.js의 특징, 장단점
Node.js 설치

02 Visual Studio Code

텍스트 편집기
Visual Studio Code 설치
유용한 확장

03 브라우저와의 차이

04 Node.js 객체, 응용


Node.js 객체
process 객체 응용
fs, path 객체 응용

05 npm이란?

Node.js 패키지 매니저
npm 명령어
외부 모듈 install

06 yarn이란?

비공식 패키지 매니저
yarn 설치
yarn 명령어



01

Node.js란?

JavaScript 런타임 환경

Node.js의 특징, 장단점

Node.js 설치

JavaScript 런타임 환경

Node.js는 JavaScript 엔진인 크롬 V8에 libuv를 결합하여 빌드한 런타임 환경(플랫폼)

- JavaScript를 브라우저가 아닌 환경에서 실행할 수 있게 해주는 환경
- 프레임워크라고 알려져 있으나, 실제로는 C언어를 실행시켜주는 Visual Studio(gcc), Java를 실행시켜주는 Eclipse(JVM, JRE)와 같은 역할
- 보통 서버 언어로 알려져 있는 경우가 있으나, 이 또한 Node.js의 기능 중 하나이며, Express.js, Next.js라는 프레임워크가 서버 개발로 유명함
- 일반 브라우저와 같이 ECMAScript 사양을 따르나, 브라우저와는 환경이 다르므로 차이가 있을 수 있음



Node.js의 특징

- 고성능의 구글 V8 엔진 활용
- V8 엔진은 C++로 개발되어 확장성이 뛰어남
- 모듈 방식으로 소스 단위를 구성하며, 매우 방대한 오픈 소스 모듈을 제공
- 개발이 빠르고 쉬움

● 비동기 이벤트 기반

고성능의 비동기 방식으로, 한 가지 작업이 끝날 때까지 기다리지 않고 다음 작업을 처리하며, 작업이 끝나면 이벤트를 발생시켜 콜백 함수를 실행하는 방식

● 싱글 쓰레드

Node.js의 개발자는 한 가지 쓰레드만 관리하면 되므로 자원을 관리하는데 굉장히 용이하나, 실제로 내부 구현은 멀티 쓰레드로 되어 성능 또한 우수함

● 오류 처리

Node.js는 기본적으로 오류 핸들링(에러 처리)를 하지 않으면 예외 발생으로 애플리케이션이 종료됨
따라서 확실한 오류 핸들링이 필수



빠른 생산에 유리

어울리는 서비스

- 간단한 로직
- 빠른 응답시간, 빠른 개발 속도
- 비동기 방식에 어울리는 서비스(스트리밍, 채팅)

어울리지 않는 서비스

- 한 가지 처리가 오래 걸리는 경우
- 체크하는 로직이 많은 경우
- 업무 복잡도/난이도가 높은 경우



큰 규모에 적합
하지 않음

Node.js의 장점

01 JavaScript 기반

JavaScript 기반으로 Front-End 개발자가 Back-End 개발까지 쉽게 할 수 있어 높은 생산성을 보여줌

03 방대한 모듈

npm, yarn, github와 같은 패키지 매니저와 오픈 소스 저장소에 매우 방대한 모듈이 있기 때문에 필요한 모듈을 다운 받아 생산성 향상에 매우 큰 역할

02 고성능의 V8 엔진

구글에서 개발하는 V8 엔진은 매우 고성능으로, 느리다는 인식의 JavaScript를 매우 빨리 구동할 수 있음

또한, C++로 개발되어 확장성이 매우 좋음

04 비동기 방식

이벤트 기반 비동기 방식이므로 서버의 무리가 매우 적고, 내부적으로는 멀티 쓰레드이지만, 개발자는 싱글 쓰레드로 관리하므로 편리함



Node.js의 단점

01 작은 규모에 적합

CPU 부하가 큰 작업이나 대용량 파일을 관리하는 등 많은 I/O 등이 발생하면 성능 저하가 눈에 띄게 늘어나며, 오류 처리에 있어 매우 불안정

03 다른 설계 방식

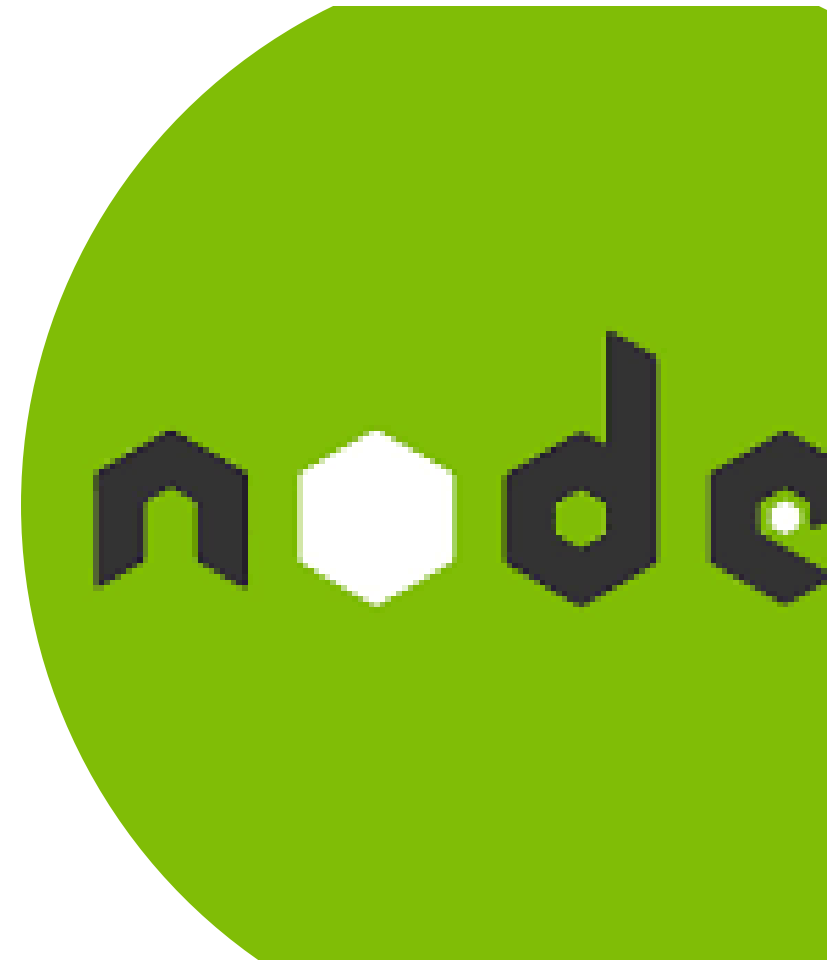
코드를 순차적으로 실행하는 것이 아니라 이벤트를 보내고 응답 이벤트가 오면 처리하는 방식이기 때문에 일반적인 프로그래밍과 같이 설계하면 어려움이 있을 수 있음

02 콜백함수

이벤트 기반 비동기 방식은 양날의 검으로, 로직이 복잡한 경우 콜백 함수의 늪에 빠져 소스코드가 매우 지저분해질 수 있음

04 에러 확인

JavaScript이기 때문에 코드가 실제로 실행되어야 에러가 있는지 알 수 있으며, 에러가 나면 애플리케이션이 종료되므로 테스트가 매우 중요




Node.js 설치

Node.js 홈페이지: <https://nodejs.org/ko/download/>

- 최신 버전은 기능이 불안정하거나 일부 모듈이 작동하지 않을 수 있으므로 LTS 버전 권장

- Node.js가 여러 버전이 있어 NVM이라는 버전 매니저를 통해 쉽게 다른 버전의 Node.js를 설치하고 전환할 수 있음

- 설치가 완료되고 터미널/CMD에서 `node -v`, `npm -v` 를 입력하면 버전 정보가 나옴 (나오지 않으면 재설치)




홈 | ABOUT | 다운로드 | 문서 | 참여하기 | 보안 | 뉴스 | CERTIFICATION


다운로드


최신 LTS 버전: **16.13.1** (includes npm 8.1.2)

플랫폼에 맞게 미리 빌드된 Node.js 인스톨러나 소스코드를 다운받아서 바로 개발을 시작하세요.

LTS
대다수 사용자에게 추천


Windows Installer
node-v16.13.1-x64.msi


macOS Installer
node-v16.13.1.pkg


Source Code
node-v16.13.1.tar.gz

Windows Installer (.msi)
Windows Binary (.zip)
macOS Installer (.pkg)
macOS Binary (.tar.gz)
Linux Binaries (x64)
Linux Binaries (ARM)
Source Code

32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	
ARMv7	ARMv8
node-v16.13.1.tar.gz	

그 밖의 플랫폼

macOS에서 Homebrew를 이용한 설치

1. 터미널에서 Homebrew 설치 명령어 입력

```
> /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. 설치 후 Homebrew를 통해 Node.js 설치

```
> brew install node
```



ubuntu(linux)에서 설치

1. 터미널에서 Node.js 설치 파일 다운로드

```
> curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
```

2. 관리자 권한으로 apt-get 패키지 관리자 실행하여 Node.js 설치

```
> sudo apt-get install -y nodejs
```





02

Visual Studio Code

텍스트 편집기

Visual Studio Code 설치

유용한 확장

텍스트 에디터

무거운 IDE와는 다르게 텍스트 파일을 편집하는 용도로 사용하는 가벼운 애플리케이션

- 주로 소프트웨어 개발에 사용
- 단순 텍스트 편집에서 환경 설정을 통해 IDE까지 넘보는 종류가 많음
- IDE는 개발의 모든 작업(코드 작성, 컴파일, 디버그, 소프트웨어 패키징, 배포) 등의 행동이 모두 포함되어 있어 무거우나, 텍스트 에디터는 코드 작성만을 위한 도구이기 때문에 가벼움
- JavaScript, Node.js는 각각 실행환경(컴파일, 디버그)이 브라우저, 명령어(터미널, CMD)이기 때문에 IDE까지는 필요 없음
- 메모장, Notepad++, Visual Studio Code, Atom, Sublime Text, vim, emacs 등이 있음



```
text-editor-element.js

getComponent () {
  if (!this.component) {
    this.component = new TextEditorComponent(
      element: this,
      mini: this.hasAttribute('mini'),
      updatedSynchronously: this.updatedSynchronously
    )
    this.updateModelFromAttributes()
  }
  return this.component
}

module.exports =
document.registerElement('atom-text-editor', {
  prototype: TextEditorElement.prototype
})
```

Visual Studio Code

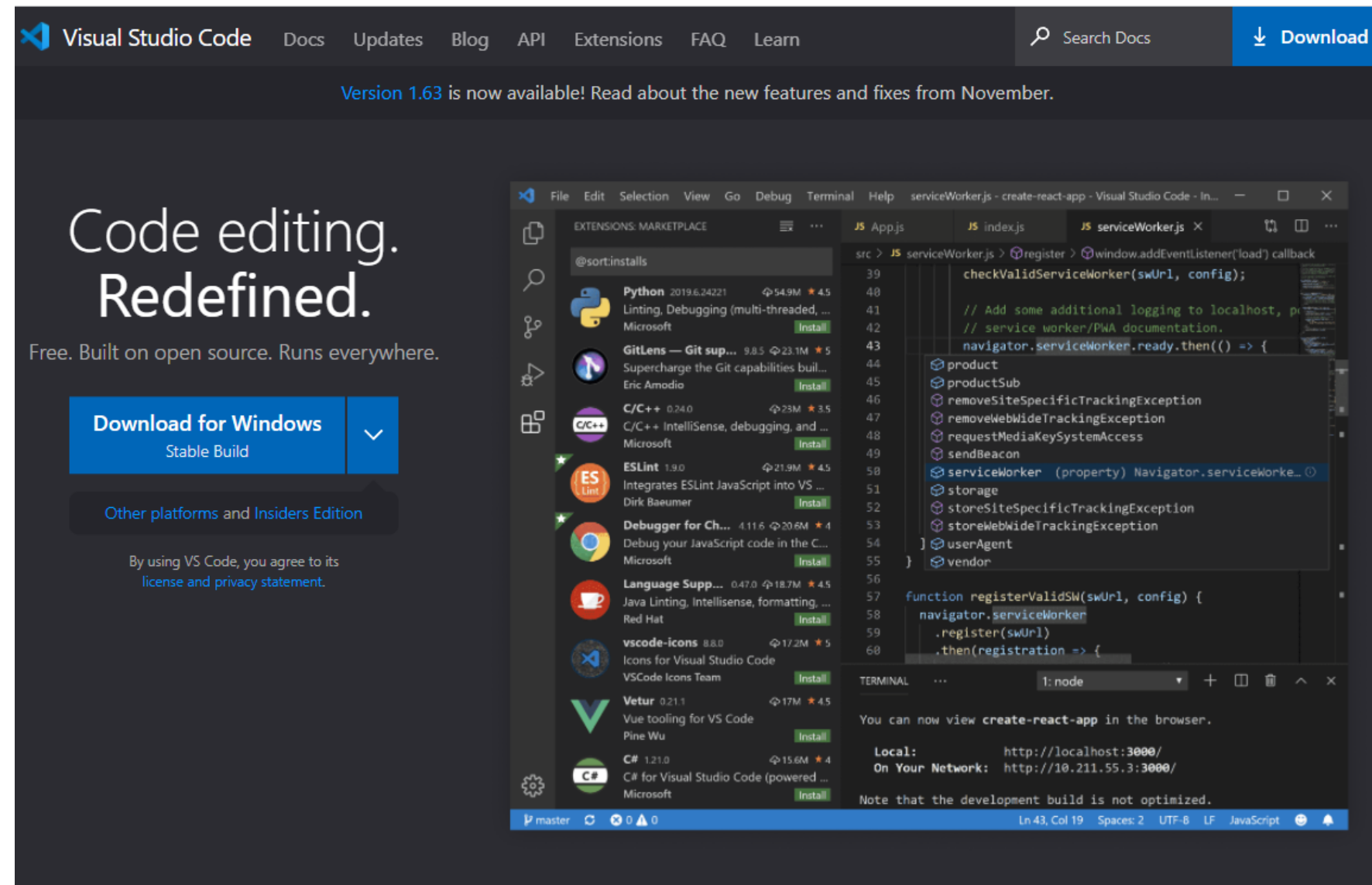
Microsoft에서 개발한 크로스 플랫폼 텍스트 에디터

- JavaScript, Node.js의 프레임워크 중 하나인 Electron.js로 개발
- 텍스트 에디터 내에 터미널/CMD가 내장되어 편리하게 명령어 사용
- IDE가 아니지만 컴파일 환경을 구축하면 C, C++ 등을 컴파일, 디버그 할 수 있음
- 즉, 노력 여하에 따라 IDE(Visual Studio, Eclipse)와 같은 동작 가능
- 강력한 플러그인, 확장 시스템으로 현재 가장 많이 사용되는 개발 도구
- JavaScript와 그 슈퍼셋 버전인 TypeScript와 궁합이 잘 맞음
- 다른 IDE와는 다르게 Windows, macOS 모두 사용 가능함




Visual Studio Code 설치

Visual Studio Code 홈페이지: <https://code.visualstudio.com/>




IntelliSense

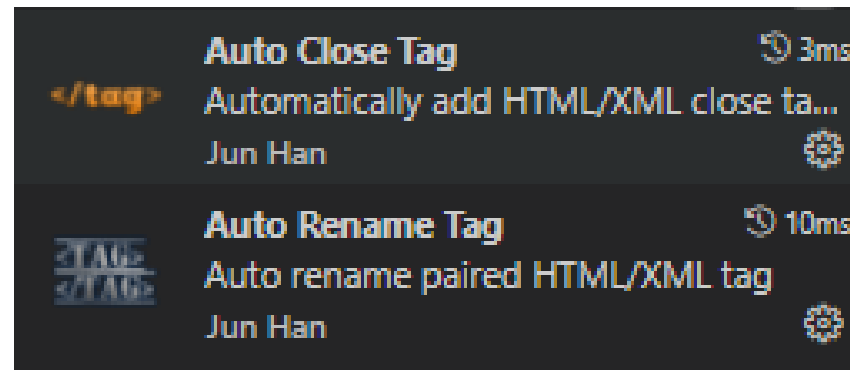

Run and Debug


Built-in Git

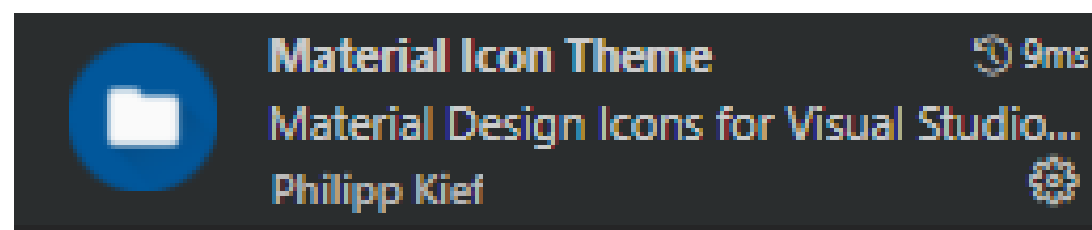

Extensions



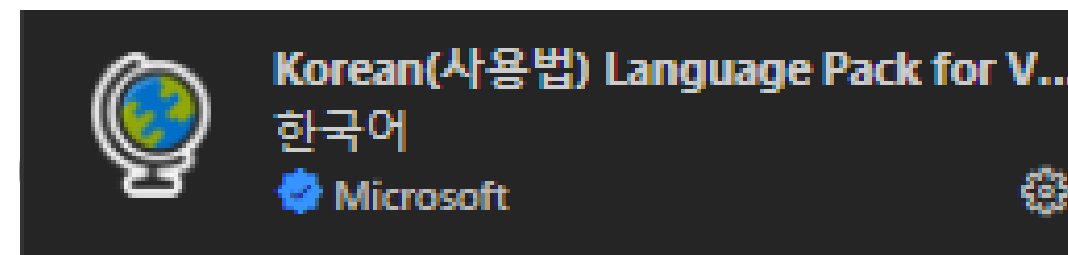
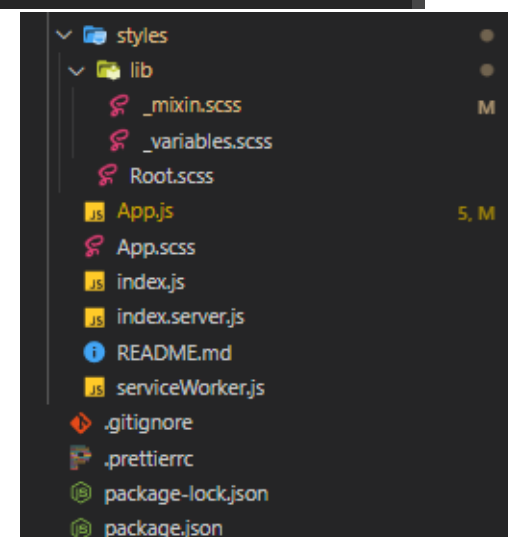
유용한 확장



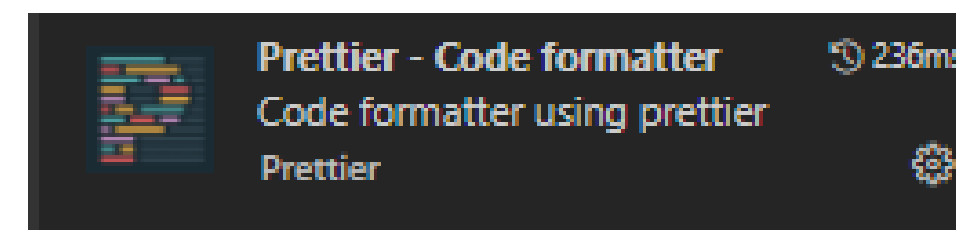
- Auto Close Tag, Auto Rename Tag
- HTML의 태그를 자동 완성해주고, 이름 변경 시 양 옆의 태그를 동시에 변경해주는 기능



- Material Icon theme
- 미관적 요소, 아이콘과 폴더를 쉽게 구분할 수 있게 해줌



- Korean Language Pack
- 설치 후 vscode를 재실행하면 한글 패치 완료



- Prettier - Code formatter
- 코드를 특정한 양식에 자동으로 맞춰주는 기능
소스코드에 양식을 두어 가독성을 높임





03

브라우저와의 차이

Node.js와 브라우저



- 개발 환경을 제공하는 것이 목적
- Node.js는 V8 엔진을 사용하고 있음
- Node.js는 버전을 선택할 수 있음
- Node.js는 CommonJS의 모듈 시스템을 사용함.
require를 통해 모듈 불러옴

Node.js API: <https://nodejs.org/docs/latest/api/>



- 웹 페이지를 화면에 띄워주는 것이 목적
- 브라우저마다 각기 다른 엔진을 사용하고 있음
(Chrome의 경우는 V8 엔진)
- 브라우저는 환경을 선택할 수 없음
JavaScript의 발전 속도를 브라우저가 따라가지 못 하기도 함
- 브라우저는 ES 모듈 표준을 사용하고 있음
import로 모듈 불러옴

브라우저 API: <https://developer.mozilla.org/ko/docs/Web/API>

04

Node.js 객체, 응용

Node.js 객체
process 객체 응용
fs, path 객체 응용

Node.js의 객체

- Node.js는 프로세스 단계에서 실행되기 때문에, 다른 언어와 마찬가지로 파일 입출력, 경로, 현재 실행 중인 컴퓨터의 정보 등을 가져올 수 있음
- 이러한 기능을 수행하는 고성능의 빌트인(내장) 객체 제공
- 내장 객체의 정보를 알고, 잘 사용하는 것이 Node.js 개발의 시작

※ 브라우저에서 사용하는 DOM 객체(document, window 등)은 사용할 수 없음

- **process 객체**

프로세스를 관리하는 객체

시스템 정보와 현재 프로세스의 상태 등을 제어하는 역할

- **global, globalThis 객체**

Node.js에서 전역 객체로, 브라우저의 window 객체와 비슷한 역할을 수행

어디서든 공통 객체 접근 가능

- **fs, os, path 등의 기본 제공 모듈**

fs모듈은 파일 입출력을 담당

os모듈은 process와 비슷하지만 운영체제와 시스템 정보, 컴퓨터 정보를 가져올 수 있음

path 모듈은 파일/폴더의 경로를 혼란 없이 사용할 수 있게 해줌

process 객체

프로세스를 관리하는 객체

- 기본 사용법 예제

```
console.log(process.env);    // 컴퓨터 환경과 관련된 정보를 가진 객체
console.log(process.version); // node.js의 버전
console.log(process.versions); // node.js와 연관된 프로그램들의 버전을 가진 객체
console.log(process.arch);    // 프로세서의 아키텍처(arm/ia32/x64)
console.log(process.platform); // 플랫폼(win32/linux/sunos/freebsd/darwin)
console.log(process.memoryUsage()); // 메모리 사용 정보를 가진 객체
console.log(process.uptime()); // 현재 프로그램이 실행된 시간
```

- 애플리케이션 종료

```
process.exit();
process.exit(0); // 정상 종료

process.exit(-1); // 비정상 종료
```

- 운영체제 구분

```
if (process.platform === "darwin") {
    // macOS일 때 실행할 코드
} else {
    // Windows일 때 실행할 코드
}
```



fs, path 모듈

경로를 혼란없이 사용하여 파일 입/출력 처리

- 기본 사용법(비동기) 예제

```
const fs = require("fs");
const path = require("path");
const file_path = path.join(__dirname, "test.txt");
fs.writeFile(file_path, new Date().toLocaleString(), (err) => {
  console.log("write Error", err);
  fs.readFile(
    file_path,
    {
      encoding: "utf8",
    },
    function (err, data) {
      console.log("read Error", err, data);
    }
  );
});
```

- 동기 사용 예제

```
const fs = require("fs");
const path = require("path");

const sync_file_path = path.join(__dirname, "sync_test.txt");
fs.writeFileSync(sync_file_path, new Date().toLocaleString());
const sync_read_file = fs.readFileSync(sync_file_path, {
  encoding: "utf8",
});
console.log("sync read file:", sync_read_file);
```



fs, path 모듈

- fs 모듈은 파일 입출력을 담당하며, Sync가 아닌 비동기 방식은 콜백 함수를 넘겨줘야 함
- 위 코드처럼 JavaScript는 화살표 함수라는 것이 존재하며, 이를 콜백 함수로 넘긴 예시
- ※ 화살표 함수는 추후 자세히 다룸
- JavaScript는 이런 비동기에 콜백 함수가 많이 들어가기 때문에 순서 보장을 위해 콜백 지옥이 발생
- 때문에 콜백을 최소화 하는 문법이 많이 생김

```
console.log("Pyramid of dooooooooooom!💣💣")

step1(function (value1) {
  step2(function (value2) {
    step3(function (value3) {
      step4(function (value4) {
        step5(function (value5) {
          step6(function (value6) {
            step7(function (value7) {
              //Do something with value 4
              console.log(value7)
            });
          });
        });
      });
    });
  });
});
```

- __dirname은 현재 폴더의 경로를 가져옴
- path 모듈의 join은 여러 문자열을 한 가지 경로로 합쳐주는 역할을 함
- 합쳐주면서 각 OS별 다른 경로 형식을 적용
- '../', '../..', './' 등의 폴더 이동 경로 형식도 적용됨
- new Date()는 현재 시간을 나타내는 Date 객체



05

npm이란?

Node.js 패키지 매니저

npm 명령어

외부 모듈 install

npm이란? Node.js 공식 패키지 매니저

npm 홈페이지: <https://www.npmjs.com/>



JavaScript 모듈
방대하게 흩어짐



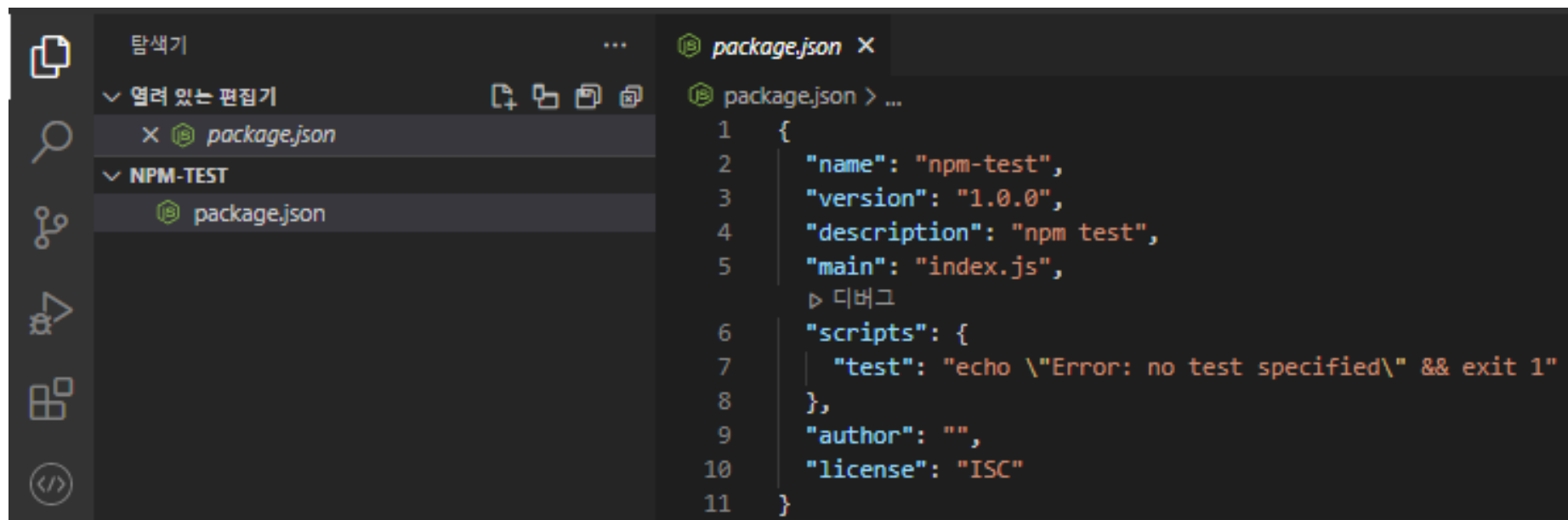
- 기존의 모듈은 방대하게 흩어져 있음
- github에서 직접 clone하는 불편한 방식
- 찾기 매우 어려운 상태

- Node Package Manager(Module)
- Node.js 패키지를 관리할 수 있는 도구, 저장소
- 140만개 이상의 패키지 등록

npm 설정 명령어

npm init

1. 터미널/CMD를 켜고 후 cd Desktop 명령어를 통해 바탕화면 진입
2. mkdir npm-test 명령어를 통해 npm-test라는 폴더 생성
3. cd npm-test 명령어를 입력하면 npm-test 폴더로 진입
4. npm init 명령어를 입력한 후 여러 가지 문항이 나오는데, 모두 Enter를 입력하면 해당 폴더에 package.json이 생성
5. code . 명령어를 입력하여 vscode를 실행



package.json과 /node_modules

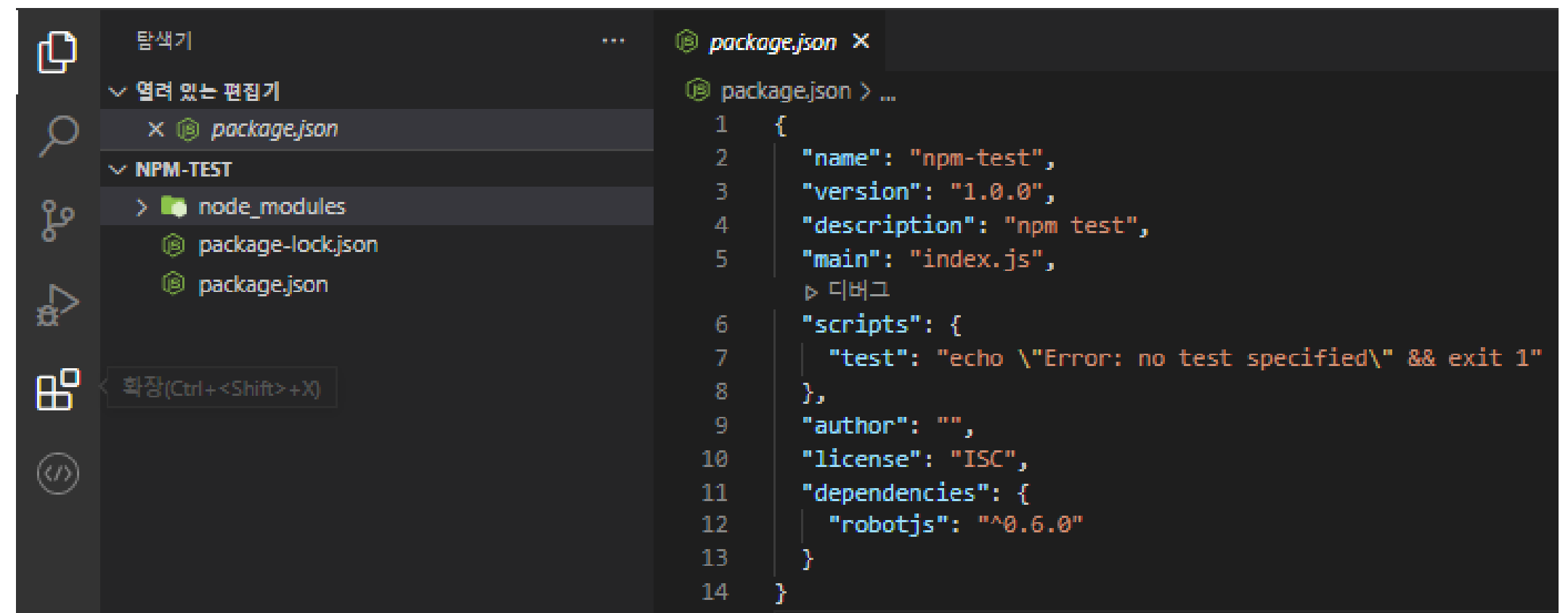
npm 프로젝트에 대해 설정하는 명세서와 의존 모듈 저장 폴더

package.json의 내용

- "name": 프로젝트의 이름
- "version": 프로젝트의 버전
- "description": 프로젝트의 설명
- "scripts": 커스텀 npm 명령어를 정의
- "dependencies": 의존 중인 모듈을 명시
- "devDependencies": 개발할 때만 사용하는 의존 모듈 명시

/node_modules

- 의존 중인 모듈을 실제로 저장하는 로컬 저장소
- npm install 명령어 실행 시 package.json에 명시된 의존 모듈을 모두 설치함
- 모듈이 많이 설치될 수록 용량이 커지며, 폴더가 손상되거나 삭제되더라도 package.json만 있다면 복구 가능
- 삭제된 상태에서 의존 모듈을 사용하려 할 경우 not found 에러 발생



npm install, uninstall

npm 프로젝트에 모듈을 추가/제거하는 명령어

- npm install 명령어를 단일로 입력할 경우, package.json에 명시된 모듈 모두 설치
- npm install <모듈 명> 명령어를 통해 현재 프로젝트에 dependencies로 해당 모듈 추가
- npm install <모듈 명> --save-dev 명령어는 devDependencies로 추가
- npm uninstall <모듈 명> 명령어를 통해 현재 프로젝트에서 모듈 제거
- npm 홈페이지에서 모듈 검색 가능
- 일반적으로 모듈은 설명하는 문서가 해당 모듈의 홈페이지에 기술되어 있음

> npm install robotjs 명령어를 통해 robotjs 모듈을 추가하여 보세요.

※ robotjs 모듈은 python 2.7 버전이 설치 되어야 하며,

Windows의 경우 Visual Studio 2013 이상, macOS의 경우 XCode가 추가로 설치되어야 합니다.



npm run, npm start

npm 커스텀 명령어를 실행하는 명령어

- 프로젝트에 index.js 파일 생성

```
// Move the mouse across the screen as a sine wave.
```

```
const robot = require("robotjs");
```

```
// Speed up the mouse.
```

```
robot.setMouseDelay(1);
```

```
const twoPI = Math.PI * 2.0;
```

```
const screenSize = robot.getScreenSize();
```

```
const height = screenSize.height / 2 - 10;
```

```
const width = screenSize.width;
```

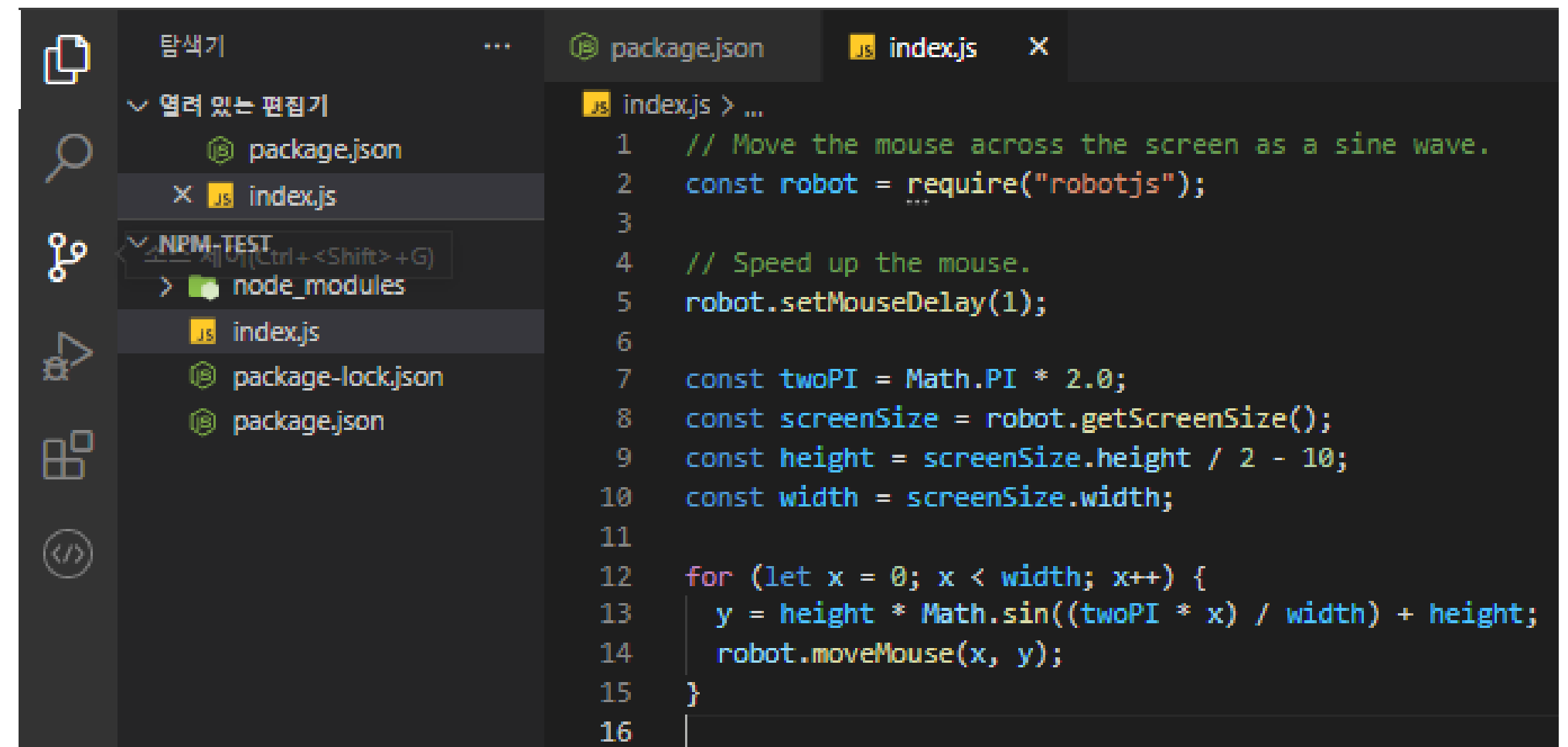
```
for (let x = 0; x < width; x++) {
```

```
  y = height * Math.sin((twoPI * x) / width) + height;
```

```
  robot.moveMouse(x, y);
```

```
}
```

- index.js 파일에 해당 코드를 작성



The screenshot shows a code editor with a dark theme. On the left, the file explorer shows a project named 'NPM-TEST' with a 'node_modules' folder and several files including 'index.js'. The main editor area shows the content of 'index.js' with the following code:

```
1 // Move the mouse across the screen as a sine wave.
2 const robot = require("robotjs");
3
4 // Speed up the mouse.
5 robot.setMouseDelay(1);
6
7 const twoPI = Math.PI * 2.0;
8 const screenSize = robot.getScreenSize();
9 const height = screenSize.height / 2 - 10;
10 const width = screenSize.width;
11
12 for (let x = 0; x < width; x++) {
13   y = height * Math.sin((twoPI * x) / width) + height;
14   robot.moveMouse(x, y);
15 }
16
```

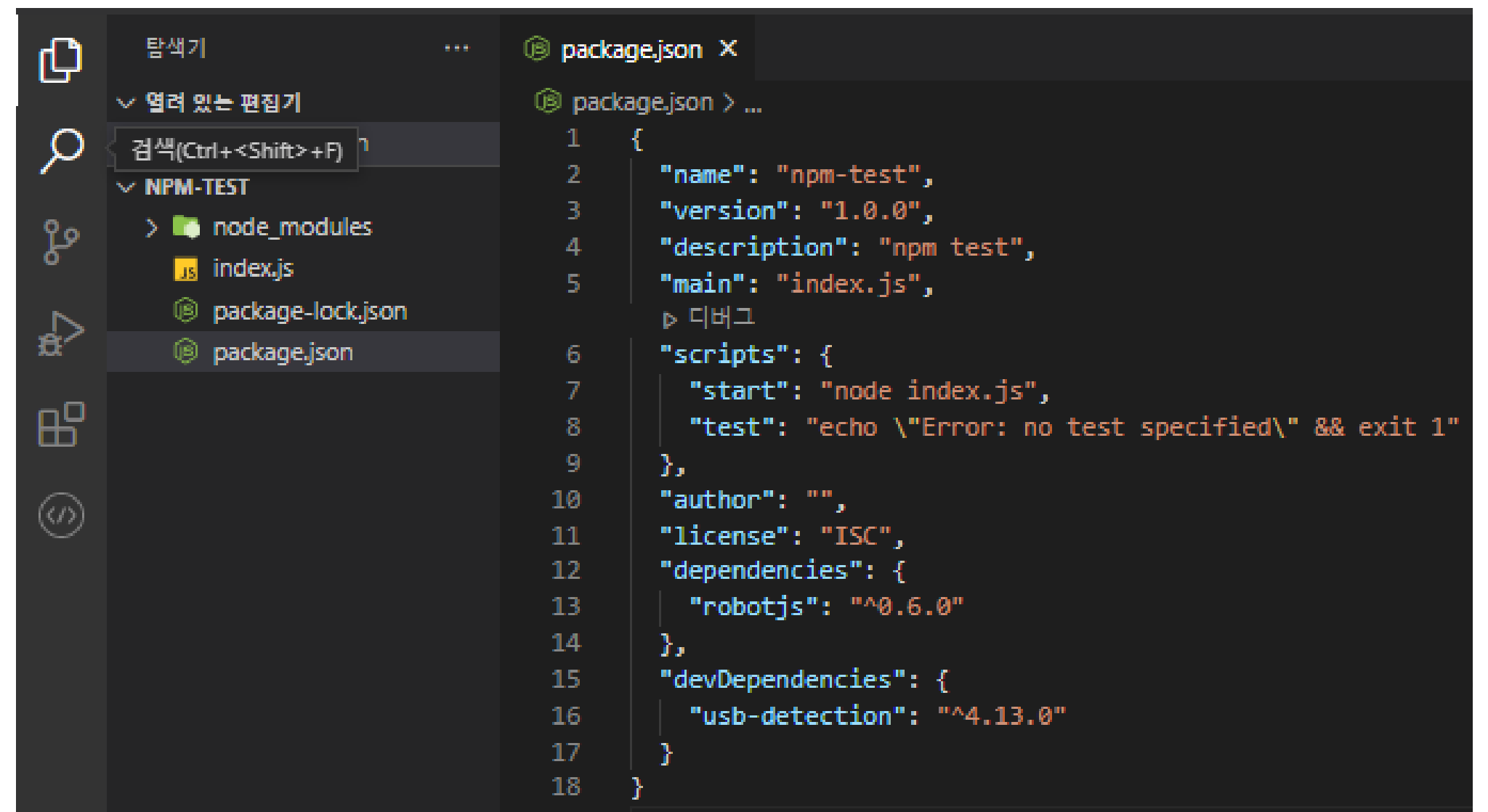
npm run, npm start

npm 커스텀 명령어를 실행하는 명령어

- package.json의 scripts에 start 작성

```
"scripts": {  
  "start": "node index.js",  
  "test": "echo W\"Error: no test specifiedW\" && exit 1"  
},
```

- node 명령어는 Node.js 파일을 실행하는 명령어
- npm run 명령어는 scripts의 커스텀 명령어를 실행
- npm run start를 실행하면 node index.js 실행
- start 명령의 경우 특별히 npm start로도 사용 가능



The screenshot shows a code editor with two panels. The left panel displays a file explorer for a project named 'NPM-TEST'. It shows a 'node_modules' directory, an 'index.js' file, and two JSON files: 'package-lock.json' and 'package.json'. The right panel shows the content of 'package.json', which is a JSON object with the following fields: 'name' (npm-test), 'version' (1.0.0), 'description' (npm test), 'main' (index.js), 'scripts' (start: node index.js, test: echo \"Error: no test specified\" && exit 1), 'author' (empty), 'license' (ISC), 'dependencies' (robotjs: ^0.6.0), and 'devDependencies' (usb-detection: ^4.13.0).

```
package.json X  
package.json > ...  
1  {  
2    "name": "npm-test",  
3    "version": "1.0.0",  
4    "description": "npm test",  
5    "main": "index.js",  
6    "scripts": {  
7      "start": "node index.js",  
8      "test": "echo \"Error: no test specified\" && exit 1"  
9    },  
10   "author": "",  
11   "license": "ISC",  
12   "dependencies": {  
13     "robotjs": "^0.6.0"  
14   },  
15   "devDependencies": {  
16     "usb-detection": "^4.13.0"  
17   }  
18 }
```

06

yarn이란?

비공식 패키지 매니저

yarn 설치

yarn 명령어

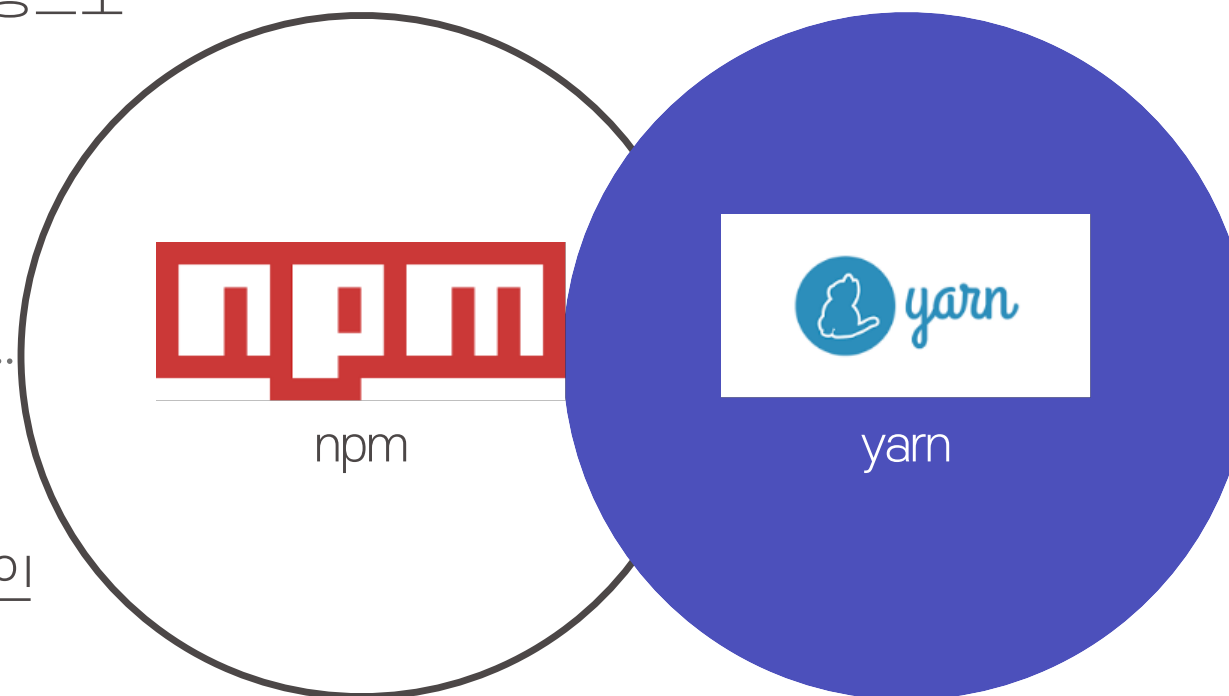
yarn이란?

페이스북에서 발표한 비공식 패키지 매니저

yarn 홈페이지: <https://yarnpkg.com/>

초창기에는 npm을 압도하는 성능으로 인기를 끌었으나 현재는 npm이 개선을 거듭하여 큰 차이가 없음

npm과 동일한 저장소를 사용하므로 yarn import를 통해 손쉽게 마이그레이션



npm은 package-lock.json, yarn은 yarn.lock 파일로 구체적인 의존성 관리, yarn 패키지 알고리즘 상 npm과는 달리 모든 환경에서 동일한 의존성 보장

현재 node_modules의 근본적인 문제점을 해결하기 위해 .pnp.js라는 새로운 방식의 통합 의존성 파일 모색

npm과 yarn 비교: <https://dongmin-jang.medium.com/npm-vs-yarn-7b699c5d6034>

yarn 설치

1. 터미널/CMD에서 npm을 통해 yarn 설치 명령어 입력

```
> npm install -g yarn
```

혹은

```
> npm i -g yarn
```

2. 설치 후 터미널/CMD에서 버전 확인

```
> yarn --version
```

※ npm install에서 -g 옵션은 npm 모듈을 전역적으로 설치하겠다는 뜻

즉, PC 어디에서도 해당 모듈에 접근 가능하게 되며, yarn은 npm 모듈로써 명령어를 제공하고 있음

전역 설치를 하고 싶지 않다면 npm install yarn을 사용한 후에 커스텀 명령어로 해당 프로젝트에서만 사용 가능



yarn 명령어

- yarn init: npm init과 동일, package.json이 있으면 실행하지 않아도 됨
- yarn: npm install과 동일. package.json에 명시된 모듈들을 설치
- yarn add <모듈 명>: npm install <모듈 명>과 동일
- yarn remove <모듈 명>: npm uninstall <모듈 명>과 동일
- yarn add -D <모듈 명>: npm install --save-dev <모듈 명>과 동일
- yarn <커스텀 명령어>: npm run <커스텀 명령어>와 동일



실증 아카데미 리액트 특강

THANK
YOU

(주) 인바이즈 박철현
(주) 인바이즈 개발팀