



Seoul
Software
ACademy

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

With. 팀 뽀빠드

Node.js 란?

Node.js



- 서버 측 자바스크립트 런타임 환경
- 브라우저 밖에서 자바스크립트를 사용할 수 있음
- 자바스크립트 실행 엔진으로 구글 크롬에서 사용하는 V8 엔진을 탑재해 실행 속도 빠름
- 이벤트 기반, 비동기 I/O 모델을 사용해 가볍고 효율적
- NPM 패키지 매니저는 세계에서 가장 큰 오픈 소스 라이브러리

런타임이란?

- 프로그래밍 언어가 구동되는 환경

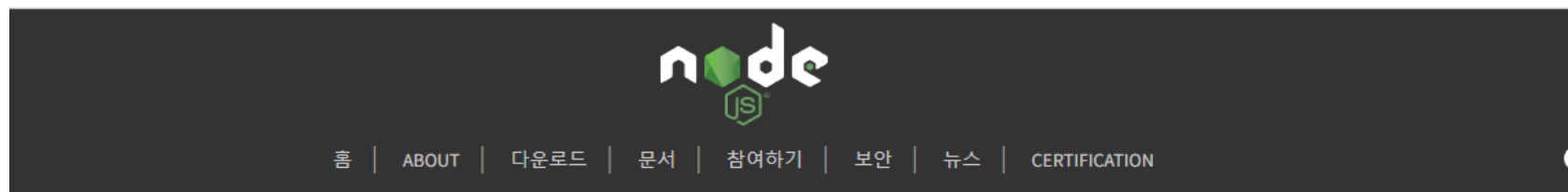


- Node.js = JavaScript 런타임 = JavaScript로 만든 프로그램을 실행할 수 있는 프로그램
- JavaScript의 런타임 환경은 웹 브라우저만 존재했었음.
 - JavaScript를 서버단 언어로 사용하기 위해 나온 것이 Node.js
 - 즉, 자바스크립트 코드를 웹 브라우저 없이 실행 가능 🤖

Node.js 설치

Node.js 설치 - 윈도우

[Node.js \(nodejs.org\)](https://nodejs.org)




다운로드

최신 LTS 버전: 16.16.0 (includes npm 8.11.0)

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


LTS
대다수 사용자에게 추천

현재 버전
최신 기능




Windows Installer

node-v16.16.0-x64.msi



macOS Installer

node-v16.16.0.pkg



Source Code

node-v16.16.0.tar.gz

Windows Installer (.msi)

Windows Binary (.zip)

32-bit	64-bit
32-bit	64-bit

Node.js 설치 – MAC

- 1. HomeBrew 설치
 - 이미 설치되어 있다면 생략
 - 터미널에 *brew -v* 명령어 입력 후 버전이 뜬다면 설치된 것
 - https://brew.sh/index_ko
- 2. Node.js 설치
 - *brew install node*

Node.js 설치 - 버전확인

```
C:\Users\> node -v
v16.17.1

C:\Users\> npm -v
8.7.0

C:\Users\>
```

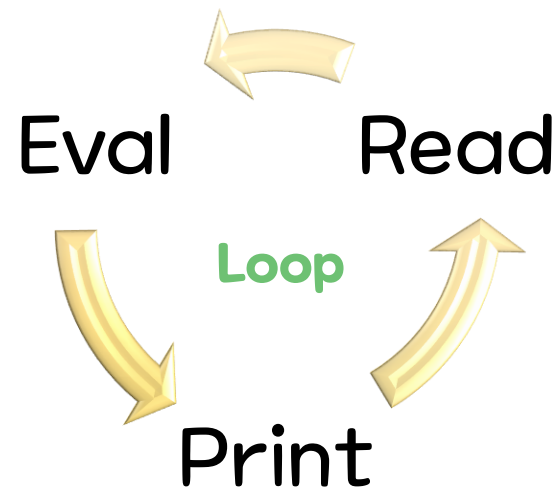
📌 **NPM** 이란?

JavaScript로 개발된 각종 모듈의 설치, 업데이트, 구성, 제거 과정을 자동화하여 관리하는 패키지

Node.js의 대화형 모드, REPL

- Read-Eval-Print-Loop 줄임말로 셸이 동작하는 방식
- 윈도우에서의 cmd, 맥에서의 terminal처럼 노드에는 REPL 콘솔이 있음

```
C:\Users\Linda>node
Welcome to Node.js v12.22.12.
Type ".help" for more information.
>
```



REPL 사용하기

```
C:\Users\linda>node
Welcome to Node.js v12.22.12.
Type ".help" for more information.
> var a = "안녕";
undefined
> var b = "반가워";
undefined
> console.log ( a + " 000. " + b );
안녕 000. 반가워
undefined
> .exit

C:\Users\linda>
```

터미널에서 JS 코드 입력
(간단한 코드 테스트 용도)

Node.js 특징

Node.js 특징

1. 자바스크립트 언어 사용
2. Single Thread
3. Non-blocking I/O
4. 비동기적 Event-Driven

특징1. JavaScript 언어 사용

- JavaScript 언어는 원래 웹 브라우저 환경에서만 동작하였음
- Node.js 의 등장으로 터미널에서도 브라우저 없이 바로 실행 가능!
- JavaScript 언어 한가지로 **프론트엔드와 백엔드(서버)**를 모두 만들 수 있게 되었음 🙌

특징 2. Single Thread

프로세스

- 실행 중인 프로그램
- 운영체제에서 할당하는 작업의 단위

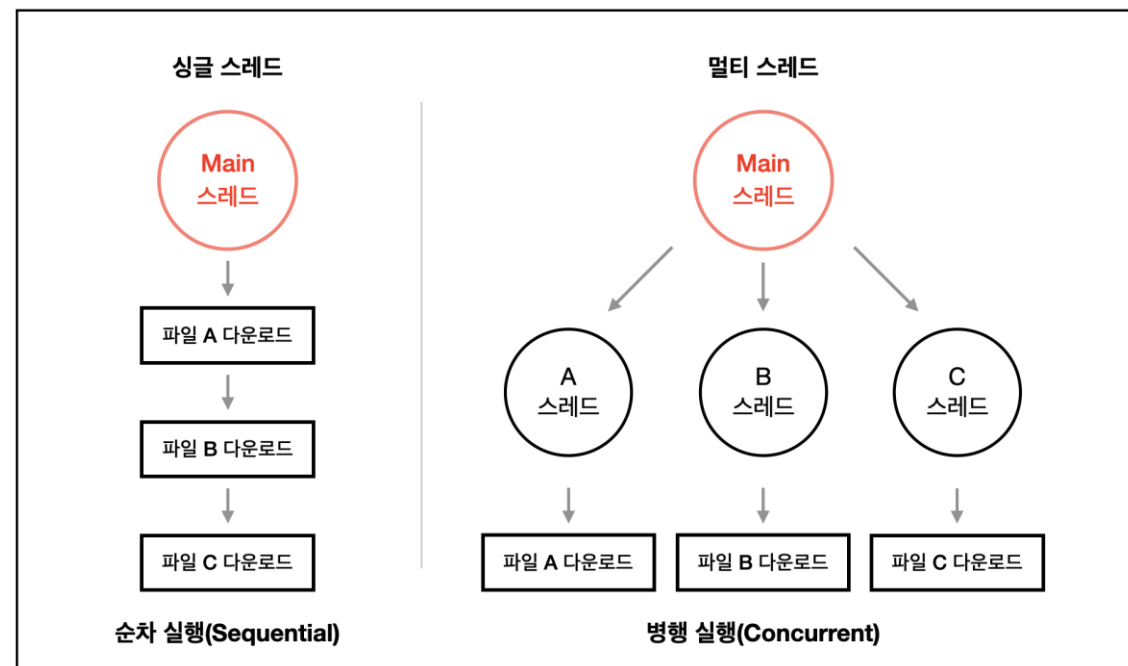
Thread(스레드)

- 프로세스 내에서 실행되는 흐름의 단위
- 하나의 프로세스에는 n 개의 스레드가 존재하며 동시에 작동할 수 있다.

특징 2. Single Thread

Node.js는 사용자가 직접 제어할 수 있는 **스레드는 하나**이다. == **Call Stack 하나**이다.

- 싱글 스레드라 **한 번에 하나의 작업만 가능**
- Non-blocking I/O 기능으로 일부 코드는 백그라운드(다른 프로세스)에서 실행 가능
- 에러를 처리하지 못하는 경우 멈춤
- 프로그래밍 난이도가 쉬움
- cpu, 메모리 자원을 적게 사용



특징 - Single Thread

싱글 스레드?
멀티 스레드 프로세스?
CPU?



에러를 처리하지 못하면 프로그램이 아예 중단됨



예외처리의 중요성 ↑

콜 스택 (Call Stack)

- 현재 어떤 함수가 동작하고 있는지, 그 함수 안에서 어떤 함수가 동작하고 있으며 다음에는 어떤 함수가 호출되어야 하는 지 등을 제어
- 싱글 스레드 = 콜 스택이 하나만 있음 = 한 번에 하나의 작업만 가능

코드로 Call Stack 동작을 이해해보자!

콜 스택 (Call Stack)

```
function first() {
  second();
  console.log(1);
  return;
}

function second() {
  console.log(2);
  return;
}

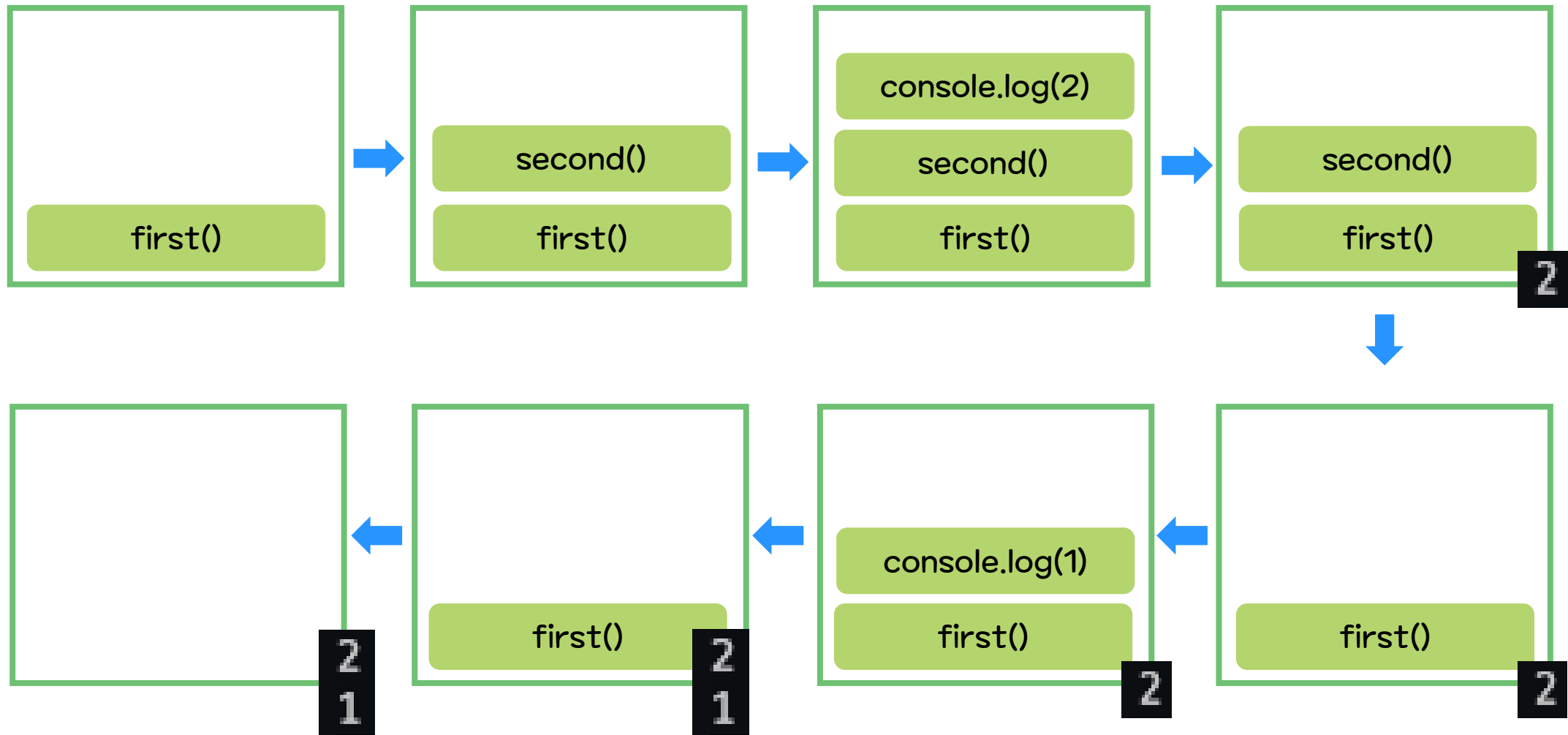
first();
```

- first() 와 second() 2개의 함수 정의
- first() 함수만 실행
- first() 함수에서는 숫자 1 출력 후 second() 함수 실행
- 출력 결과

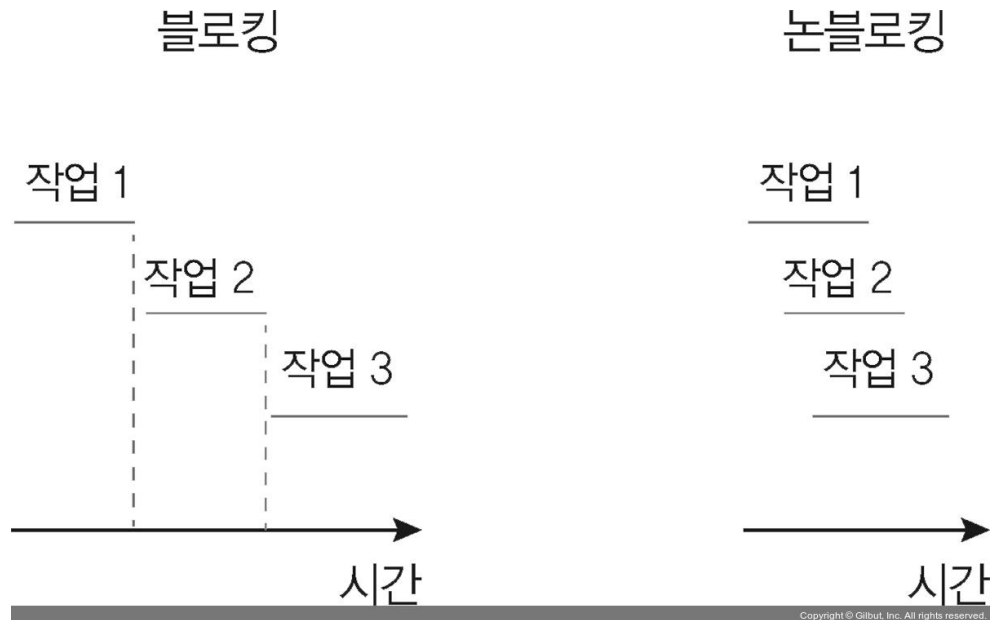
```
▶ node callstack.js
2
1
```

콜 스택 (Call Stack) 동작 예시

Call Stack은 **LIFO 구조**
Last-in First-out (후입선출)



특징 3. Non-Blocking I/O



• 블로킹 (Blocking)

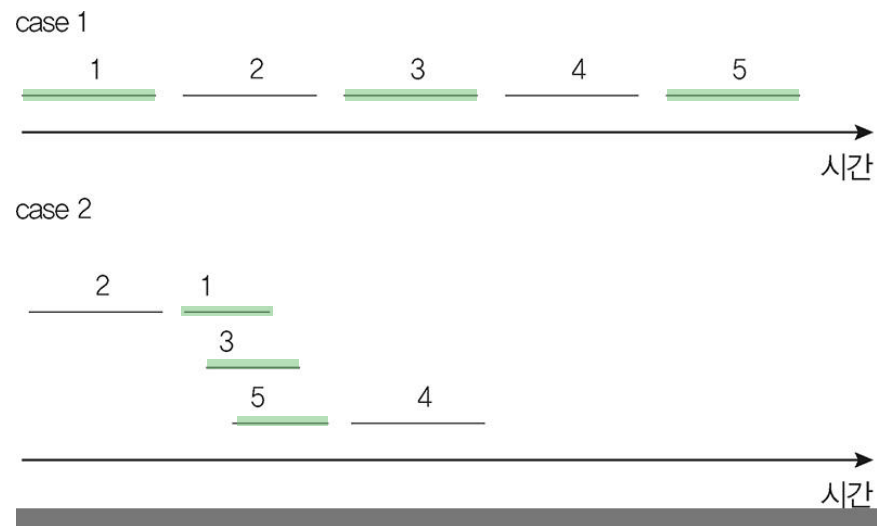
- 해당 작업이 끝나야만 다음 작업을 수행

• 논 블로킹 (Non-Blocking)

- 작업이 완료될 때까지 대기하지 않고 다음 작업 수행. 즉, 빨리 완료된 순서로 처리
- 블로킹 방식보다 같은 작업을 더 짧은 시간에 처리 가능

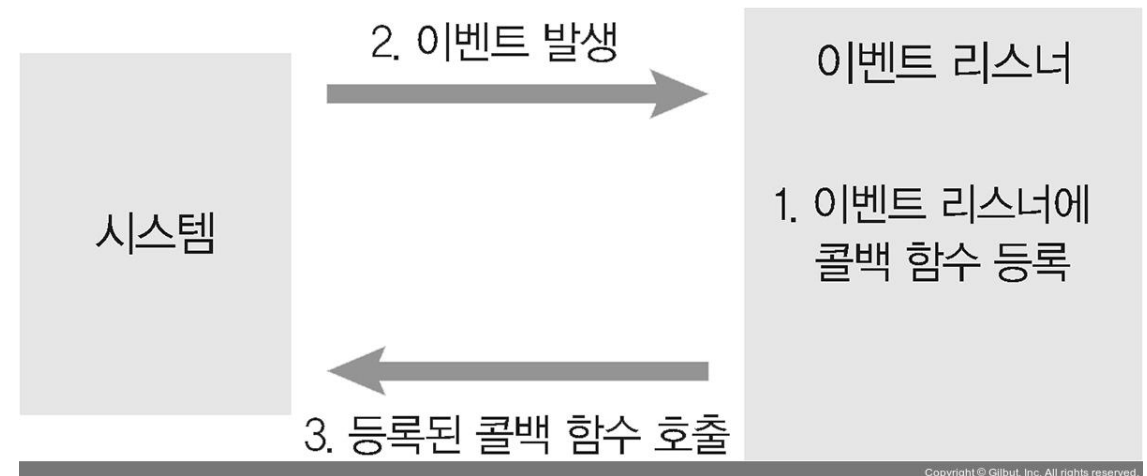
특징 3. Non-Blocking I/O

- I/O
 - 입출력(input/output) 작업
 - ex. 파일 시스템 접근 (읽기, 쓰기, 만들기 등), 네트워크 요청 등
- Node는 I/O 작업을 할 때 논블로킹 방식으로 처리
 - 동시에 처리될 수 있는 작업을 최대한 묶어서 백그라운드로 넘김
 - 시간적 이득을 획득



특징4. 이벤트 기반(Event-Driven) 아키텍처

- 이벤트가 발생할 때 미리 지정해둔 작업을 수행
- 이벤트 ex) 클릭, 네트워크 요청, 타이머 등
- 이벤트 기반 아키텍처에서는 특정 이벤트가 발생할 때 무엇을 할지 미리 등록해야 함
 - 이벤트 리스너 (Event Listener): 이벤트 등록 함수
 - 콜백 함수 (Callback Function): 이벤트가 발생했을 때 실행되는 함수



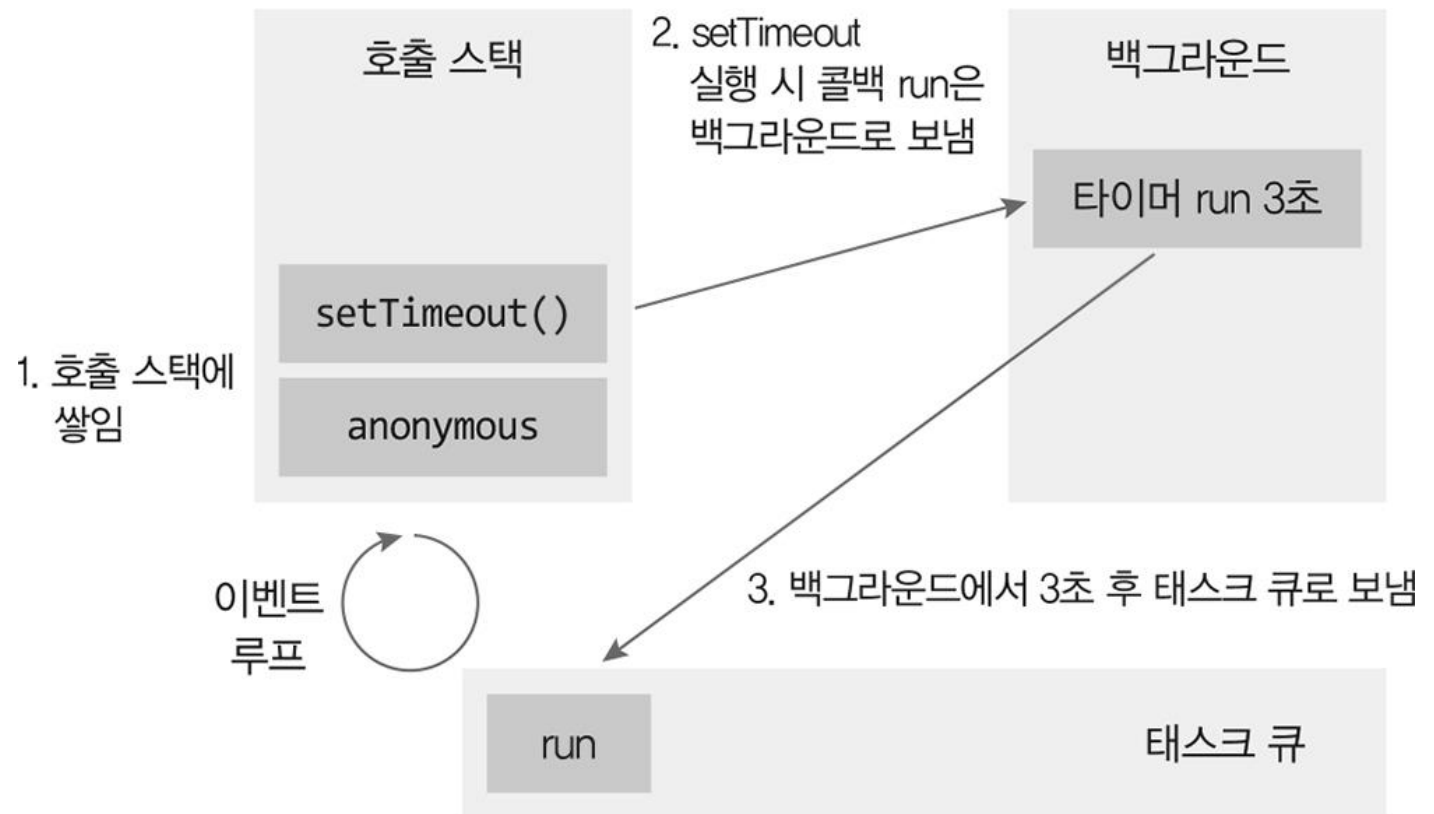
Event Loop란?

- Node.js는 오래 걸리는 작업은 이벤트 루프에 맡긴다.
- 이벤트 발생시 호출할 콜백함수들을 관리하고, 호출된 콜백함수의 실행 순서를 결정하는 역할을 담당
- 노드가 종료될 때까지 이벤트 처리를 위한 작업을 반복하므로 루프(loop)라고 부름

```
function run() {
  console.log("3초 뒤 실행");
}
```

```
console.log("시작");
setTimeout(run, 3000);
console.log("끝");
```

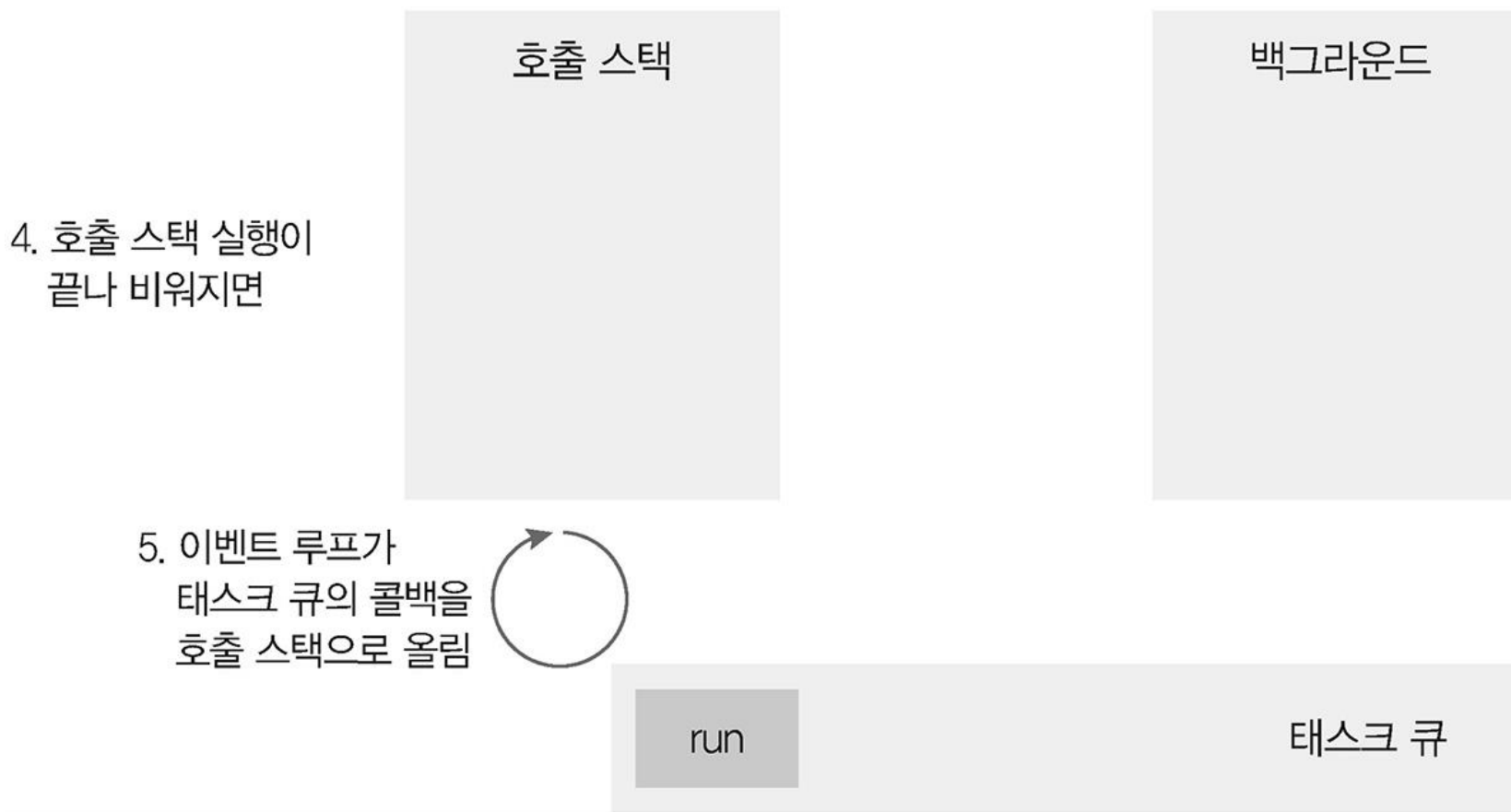
```
/*
시작
끝
3초 후 실행
*/
```



Copyright © Gilbut, Inc. All rights reserved.

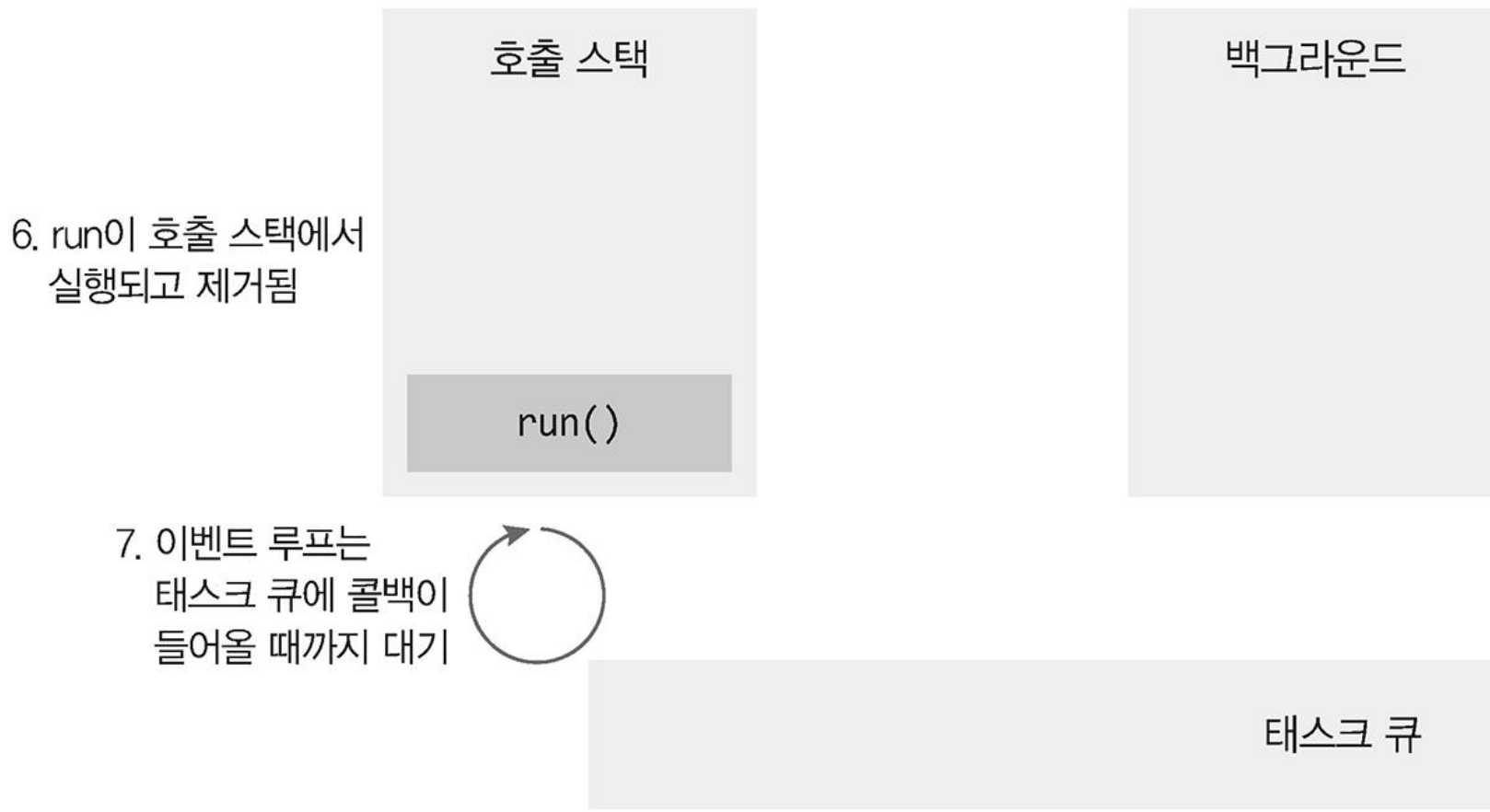
Event Loop란?

- 백그라운드: 타이머(ex. setTimeout)나 이벤트 리스터들이 대기하는 공간으로 여러 작업이 동시에 실행 될 수 있음
- 태스크 큐: 이벤트 발생 후, 백그라운드에서 태스크 큐로 타이머/이벤트 리스너의 콜백함수를 보냄.



Event Loop란?

- 백그라운드: 타이머(ex. setTimeout)나 이벤트 리스터들이 대기하는 공간으로 여러 작업이 동시에 실행 될 수 있음
- 태스크 큐: 이벤트 발생 후, 백그라운드에서 태스크 큐로 타이머/이벤트 리스너의 콜백함수를 보냄.



Copyright © Gilbut, Inc. All rights reserved.


Node.js 의 역할

- 간단한 로직
- 대량의 클라이언트가 접속하는 서비스 (입출력이 많은 서비스)
- 빠른 개발 요구
- 빠른 응답시간 요구
- 비동기 방식에 어울리는 서비스 (스트리밍 서비스, 채팅 서비스 등)


Node.js 를 사용 중인 기업은?!

이커머스


* 총 13개의 기술 스택




트렌비



식스샵




빌리버




11번가

금융/보험


* 총 12개의 기술 스택




핀다



파운트




음악카우




비바리퍼블리카

소셜/컨텐츠


* 총 17개의 기술 스택




당근마켓




퍼블리




리디




왓차




채널코퍼레이션




라인




브이씨앤씨




미스터블루




천명앤컴퍼니




라이너




디스콰이엇




더블유클럽




데브시스스터즈




베이글코드



콘텐츠퍼스트



이제이엔



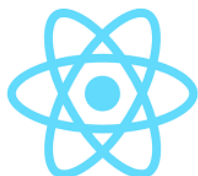
아임웹

<https://www.codenary.co.kr/techstack/detail/nodejs>

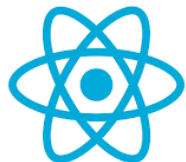
서버 외의 Node

- 노드는 자바스크립트 런타임으로 서버에만 용도가 한정되어 있지 않음!

- 웹



- 모바일



React Native

- 데스크톱 애플리케이션



NPM

NPM

- Node Package Manager (<https://www.npmjs.com/>)
- **노드 패키지**를 관리해주는 툴

Npm에 업로드 된 노드 모듈
패키지들 간 의존 관계가 존재



NPM 사용하기

```
npm init
```

- 프로젝트를 시작할 때 사용하는 명령어
- `package.json`에 기록될 내용을 문답식으로 입력한다.

```
npm init --yes
```

- `package.json`이 생성될 때 **기본 값으로** 생성된다.

```
npm install 패키지 이름
```

- 프로젝트에서 사용할 **패키지를 설치**하는 명령어
- 설치된 패키지의 이름과 정보는 `package.json`의 `dependencies` 에 입력된다.

package.json

- 패키지들이 서로 의존되어 있어, 문제가 발생할 수 있는데 이를 관리하기 위해 필요한 것
- 프로젝트에 대한 정보와 사용 중인 패키지 이름 및 버전 정보가 담겨 있는 파일

```
{
  "name": "220721",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▶ 디버그
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```


package.json

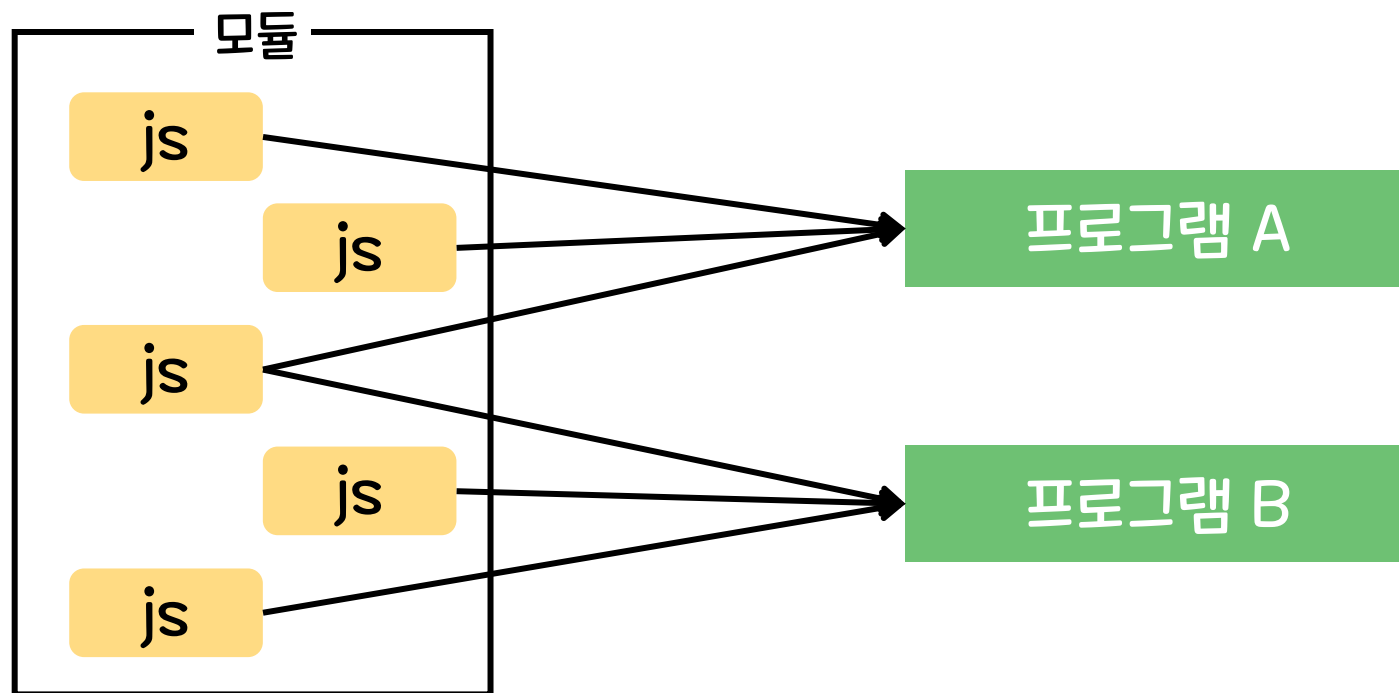
- name: 패키지 이름
- version: 패키지의 버전
- main: 자바스크립트 실행 파일 진입점 (문답식에서의 entry point)
- description: 패키지에 대한 설명
- scripts: npm run 을 이용해 정해놓는 스크립트 명령어
- license: 해당 패키지의 라이선스

```
{
  "name": "220721",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▶ 디버그
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

모듈 (Module)

모듈이란?

- 특정한 기능을 하는 함수나 변수들의 집합
- 재사용 가능한 코드 조각



모듈의 장점

- 코드 추상화
- 코드 캡슐화
- 코드 재사용
- 의존성 관리



노드 모듈 만들기 (1)

- 하나의 모듈에 하나 만들기

```
const add = (a, b) => a + b;
```

```
module.exports = add;
```

math.js

- module.exports 구문으로 내보내야 다른 파일에서 사용 가능

`module.exports` = 내보내려는 항목

노드 모듈 불러오기 (1)

- 하나의 모듈에 하나 만들기

```
const add = (a, b) => a + b;

module.exports = add;                                math.js
```

- math 모듈에서 add 함수 불러오기

```
const add = require('./math');

console.log(add);                                    app.js
```

```
▶ node app.js
[Function: add]
```

불러온 노드 모듈 사용하기 (1)

- 하나의 모듈에 하나 만들기

```
const add = (a, b) => a + b;

module.exports = add;                                     math.js
```

- math 모듈에서 불러온 add 함수 사용하기

```
const add = require('./math');

console.log(add);
console.log(add(4, 5));                                   app.js
```

```
▶ node app.js
[Function: add]
9
```

노드 모듈 만들기 (2)

- 하나의 모듈 파일에 여러 개 만들기

```
const add = (a, b) => a + b;
const E = 2.718;
const PI = 3.141592;
```

```
// case1
module.exports = {
  add,
  E,
  PI
};
```

```
// case2
module.exports.add = add;
module.exports.E = E;
module.exports.PI = PI;
```

```
// case2 생략
exports.add = add;
exports.E = E;
exports.PI = PI;
math2.js
```


노드 모듈 불러오고 사용하기 (2)

- math 모듈 불러오기

```
const math = require('./math2');

console.log(math);
```

app2.js

- math 모듈에서 불러온 함수, 변수 사용하기

```
const math = require('./math2');

console.log(math);
console.log(math.add(math.PI, math.E));
```

app2.js

```
▶ node app2.js
{ add: [Function: add], E: 2.718, PI: 3.141592 }
5.859592
```

노드 모듈 불러올 때 주의할 점

- `const { }` 로 가져올 때는 객체 구조분해해 가져오기에 이름이 동일해야 함

```
const math = require('./math2');
console.log(math);

const { add, E, PI } = require('./math2');
console.log(add(E, PI));
```

- 하나만 내보낸 모듈은 이름이 달라져도 불러올 수 있음

```
const add = require('./math');
console.log(add);

const onlyOne = require('./math');
console.log(onlyOne);
```

ES2015 모듈

- 자바스크립트 자체 모듈 시스템 문법
- 노드 모듈 시스템과 방식이 약간 다름
- package.json 에 “type”: “module” 을 추가해 사용

```
"main": "index.js",
```

```
"type": "module",
```

```
  ▶ Debug
```

```
"scripts": {
```

ES2015 모듈

export : 모듈 내보내기

```
const a = "a 변수";
const b = "b 변수";

module.exports = {
  a,
  b
};
```



```
const a = "a 변수";
const b = "b 변수";

export { a, b };
```

```
function connect() {
  return a + b;
}

module.exports = connect;
```



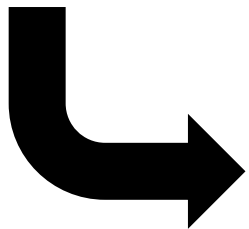
```
function connect() {
  return a + b;
}

export default connect;
```

ES2015 모듈

import ~ from ~ : 모듈 가져오기

```
const { a, b } = require("./var.js");  
const returnString = require("./func.js");
```



```
import { a, b } from './var.js';  
import returnString from './func.js';
```