

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

Node.js란?

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\>
```

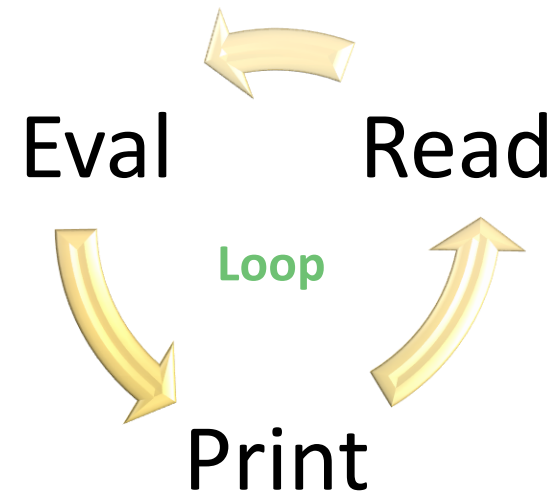
NPM이란?

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

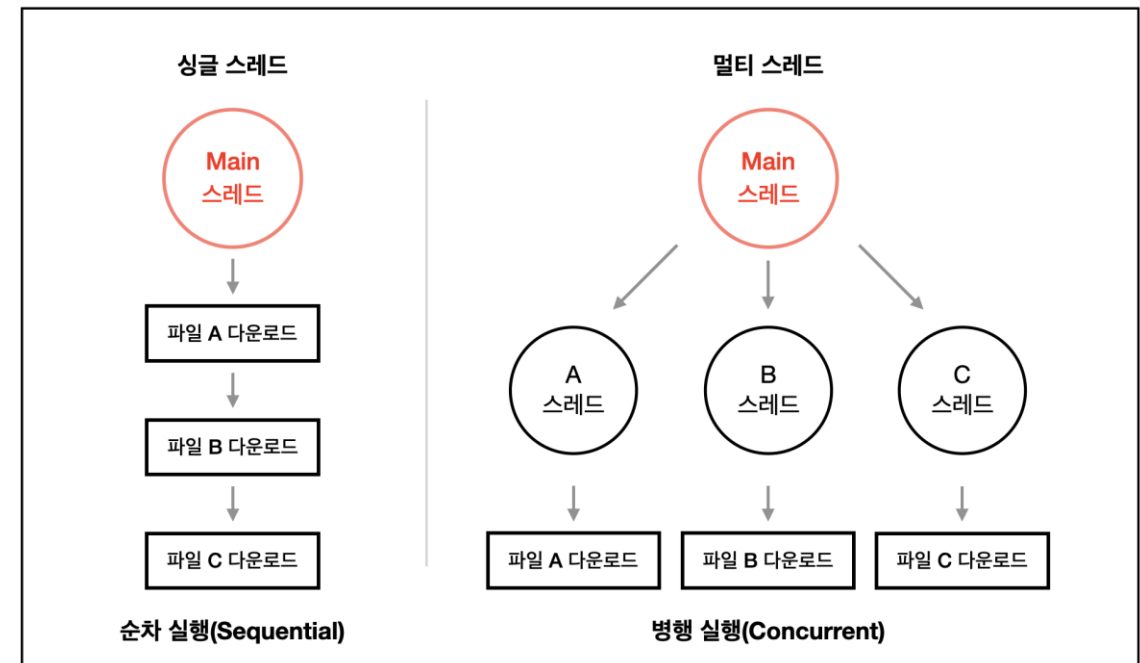
Thread(스레드)

- 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)

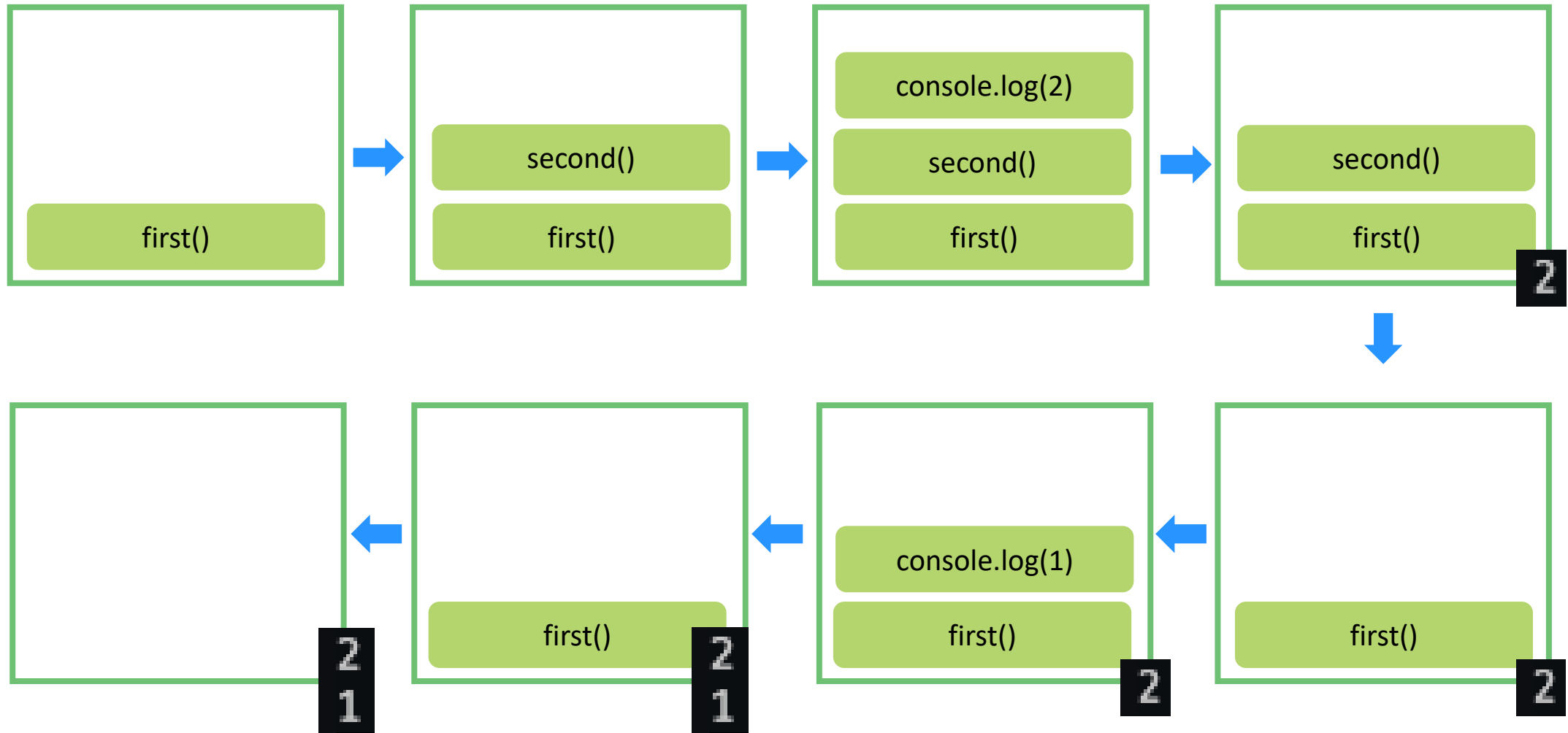
```
function first() {  
  second();  
  console.log(1);  
  return;  
}  
  
function second() {  
  console.log(2);  
  return;  
}  
  
first();
```

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

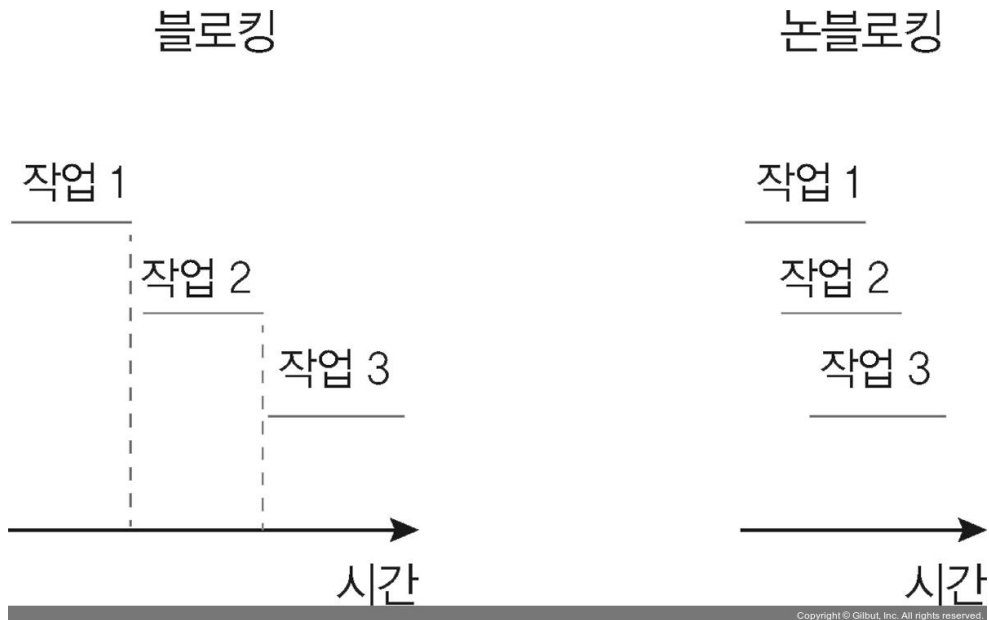
```
▶ node callstack.js  
2  
1
```

콜 스택 (Call Stack) 동작 예시

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



특징 3. Non-Blocking I/O



- 블로킹 (Blocking)

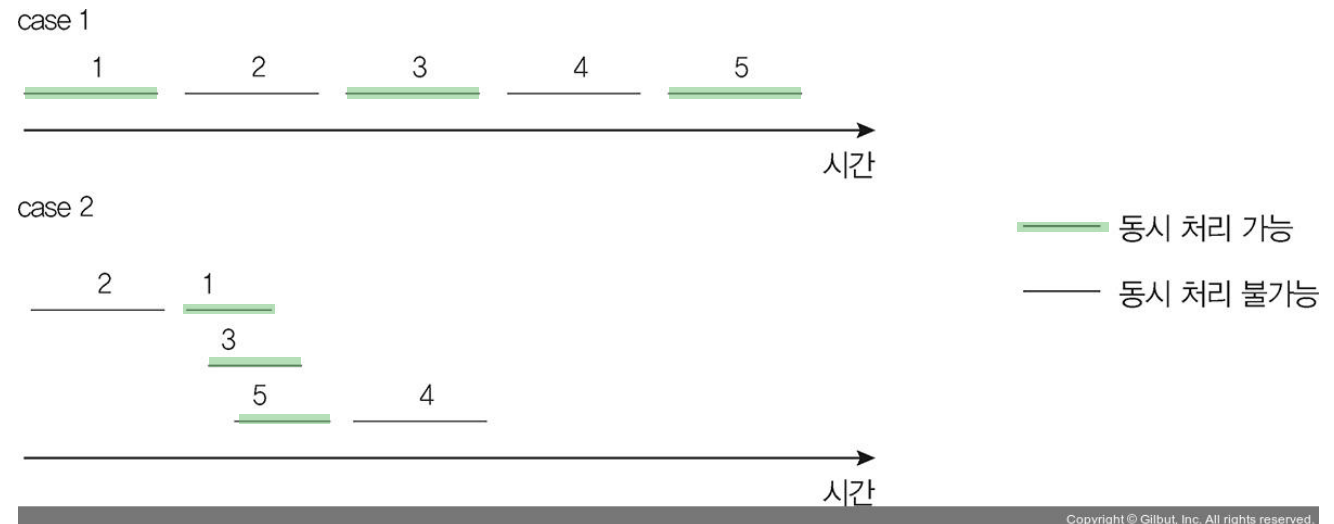
- 해당 작업이 끝나야만 다음 작업을 수행

- 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

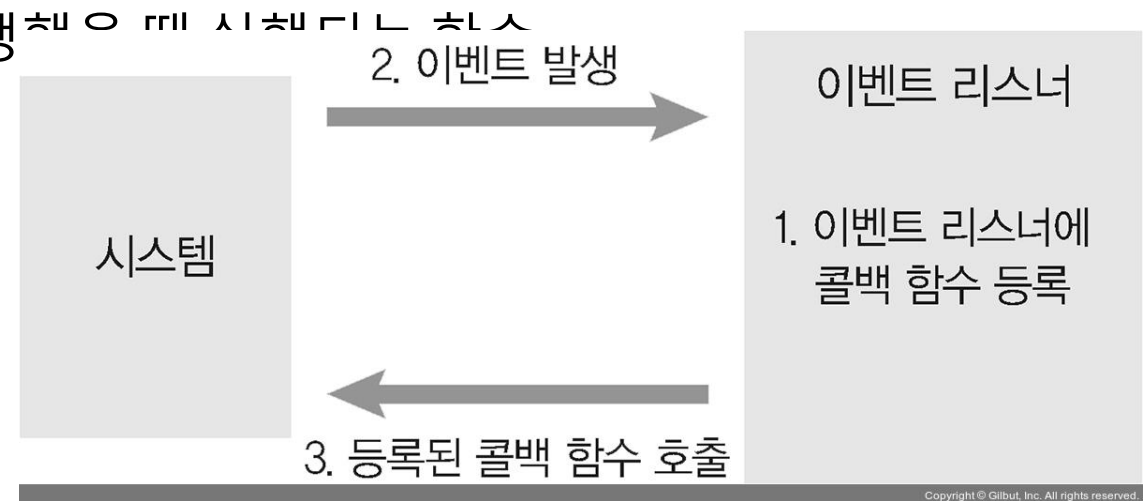
특징 3. Non-Blocking I/O

- 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 시스템이 호출하는 함수



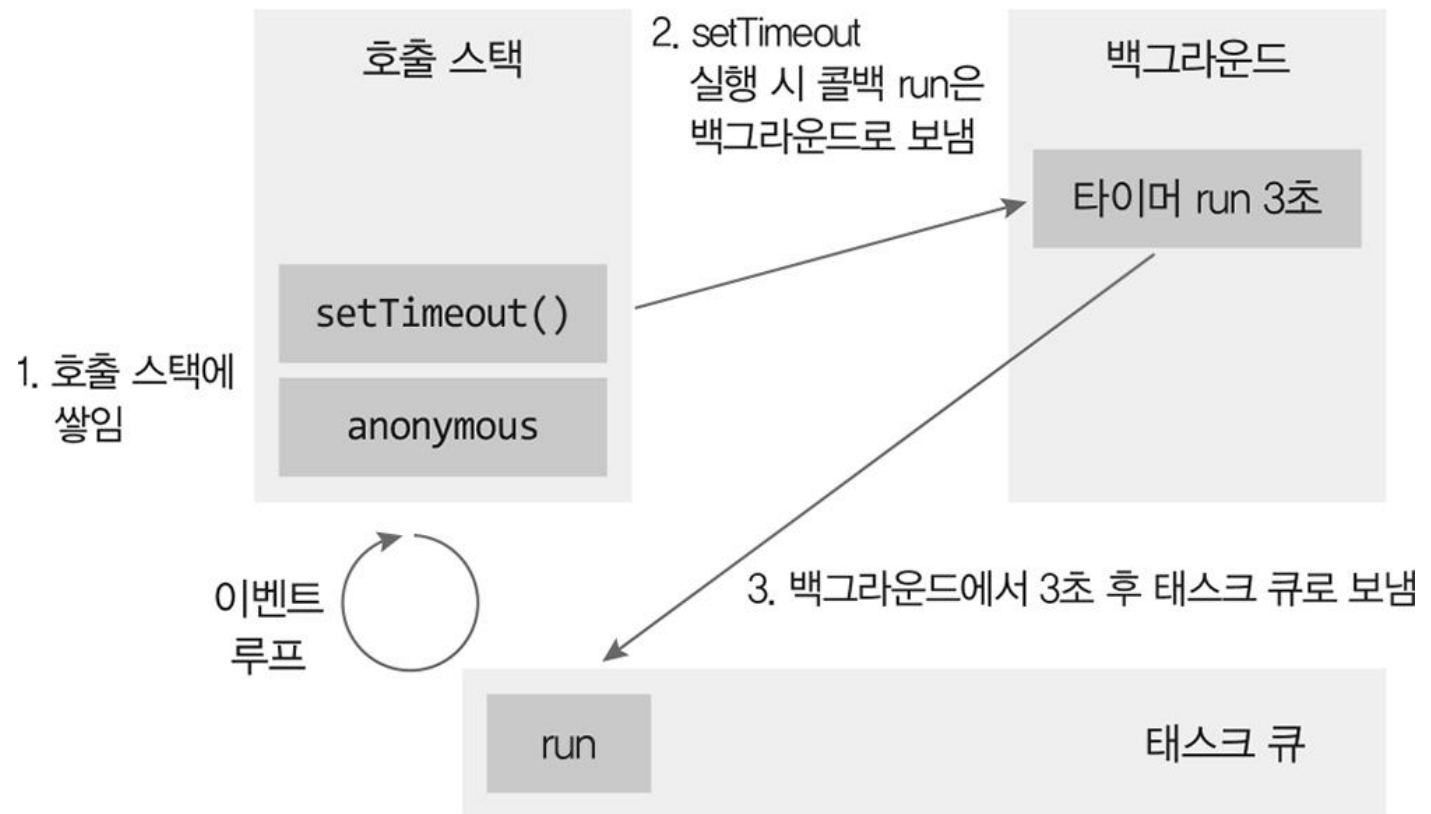
Event Loop란?

- 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");
```

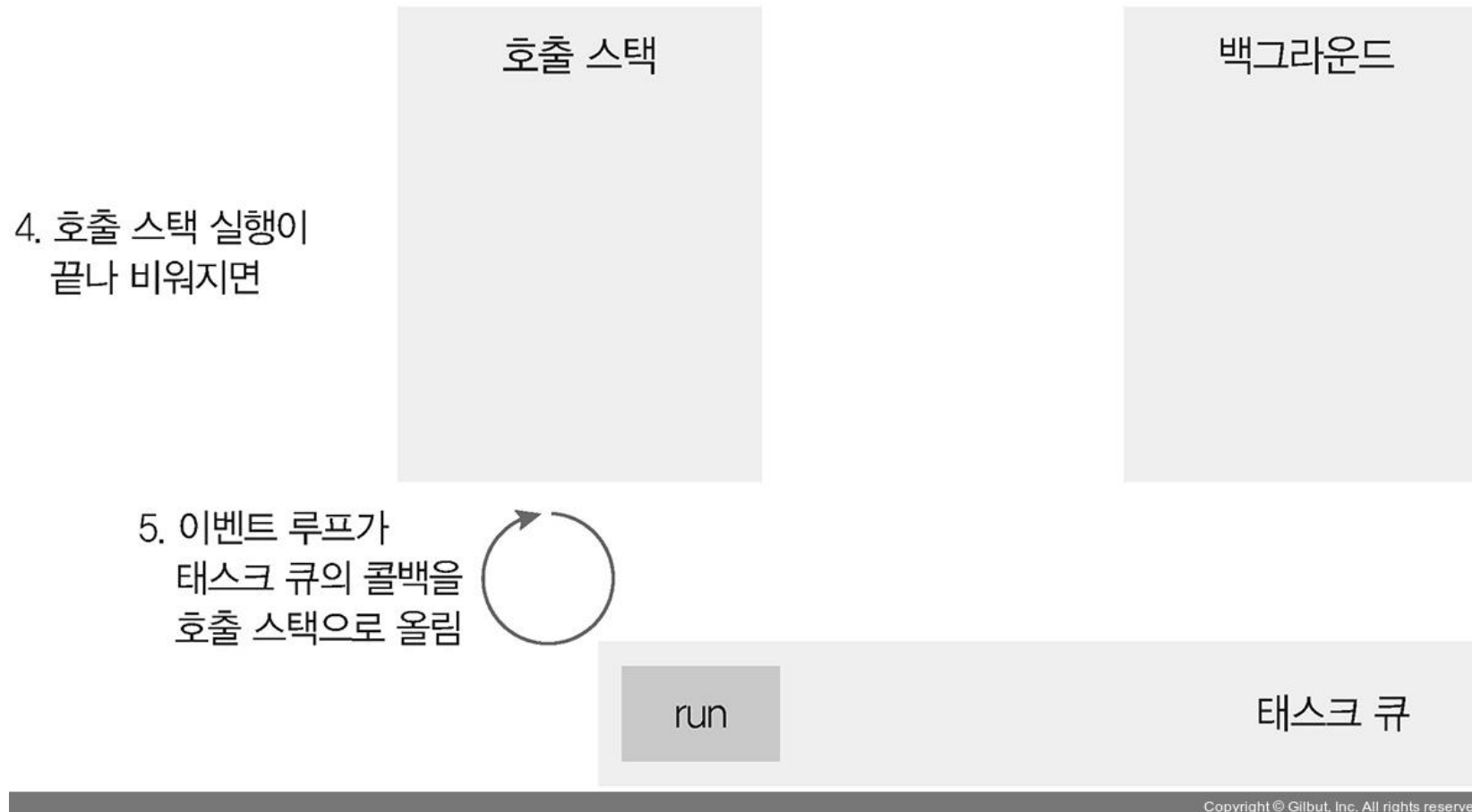
```
/*  
시작  
끝  
3초 후 실행  
*/
```



Copyright © Gilbut, Inc. All rights reserved.

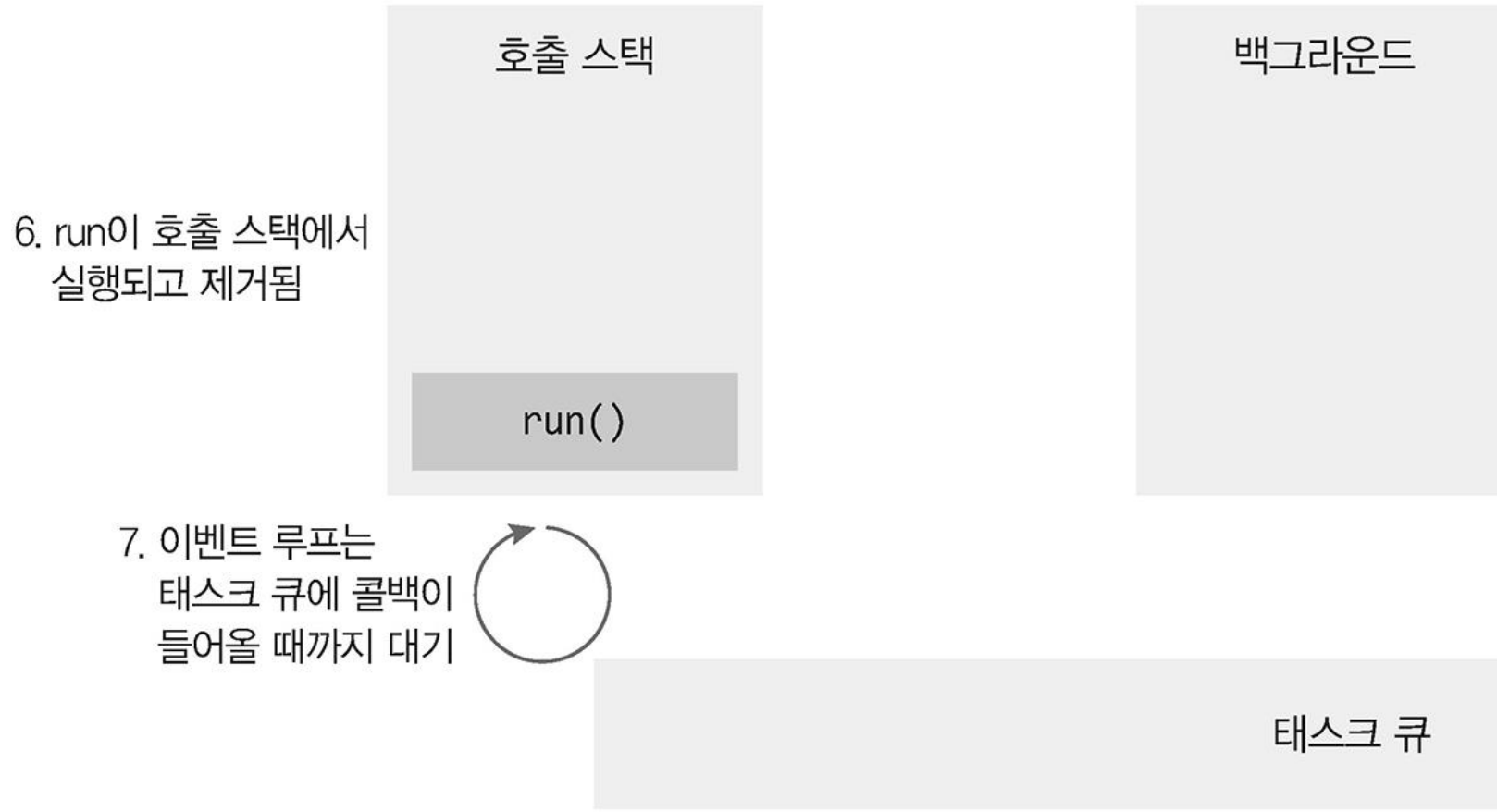
Event Loop란?

- 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.



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
Faster
빌리버
11번가
더기:

금융/보험
* 총 12개의 기술 스택

FINDA
핀다
fount
파운트
musicUw
음악카우
toss
비바리퍼블리카

소셜/컨텐츠
* 총 17개의 기술 스택

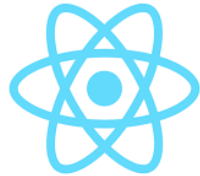
당근마켓
퍼블리
리디
왓차
채널코퍼레이션
라인
브이씨앤씨
미스터블루
천명
라이너
D*
더블유클럽
DEV SISTERS
베이글코드
콘텐츠퍼스트
이제이엔
아임웹

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

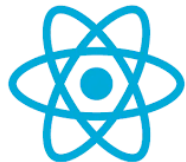
서버 외의 Node

- 노드는 자바스크립트 런타임으로 서버에만 용도가 한정되어 있지 않음!

- 웹



- 모바일



React Native

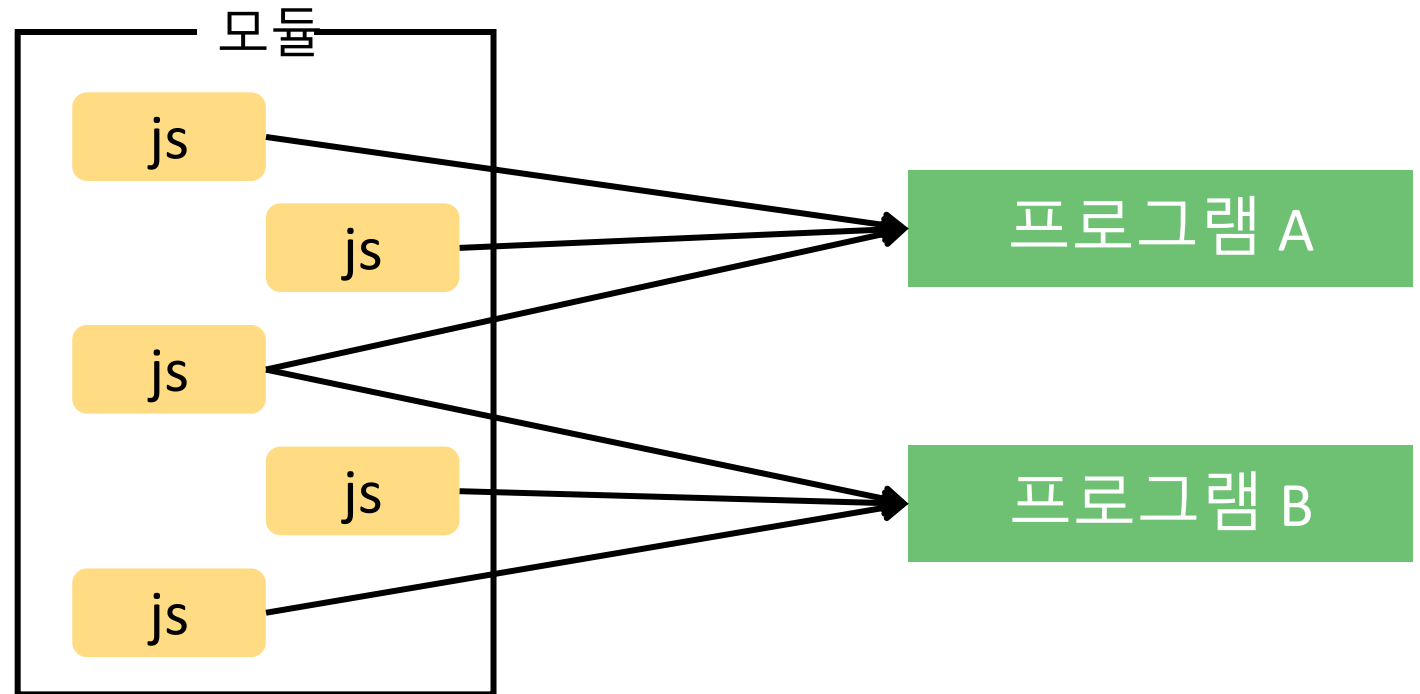
- 데스크톱 애플리케이션



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;
```

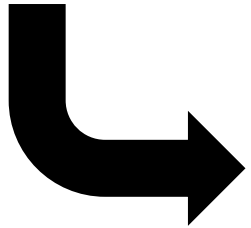


```
function connect(a, b){  
  return a+b;  
}  
  
export default connect;
```

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';
```


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 설치하는 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"
}
```

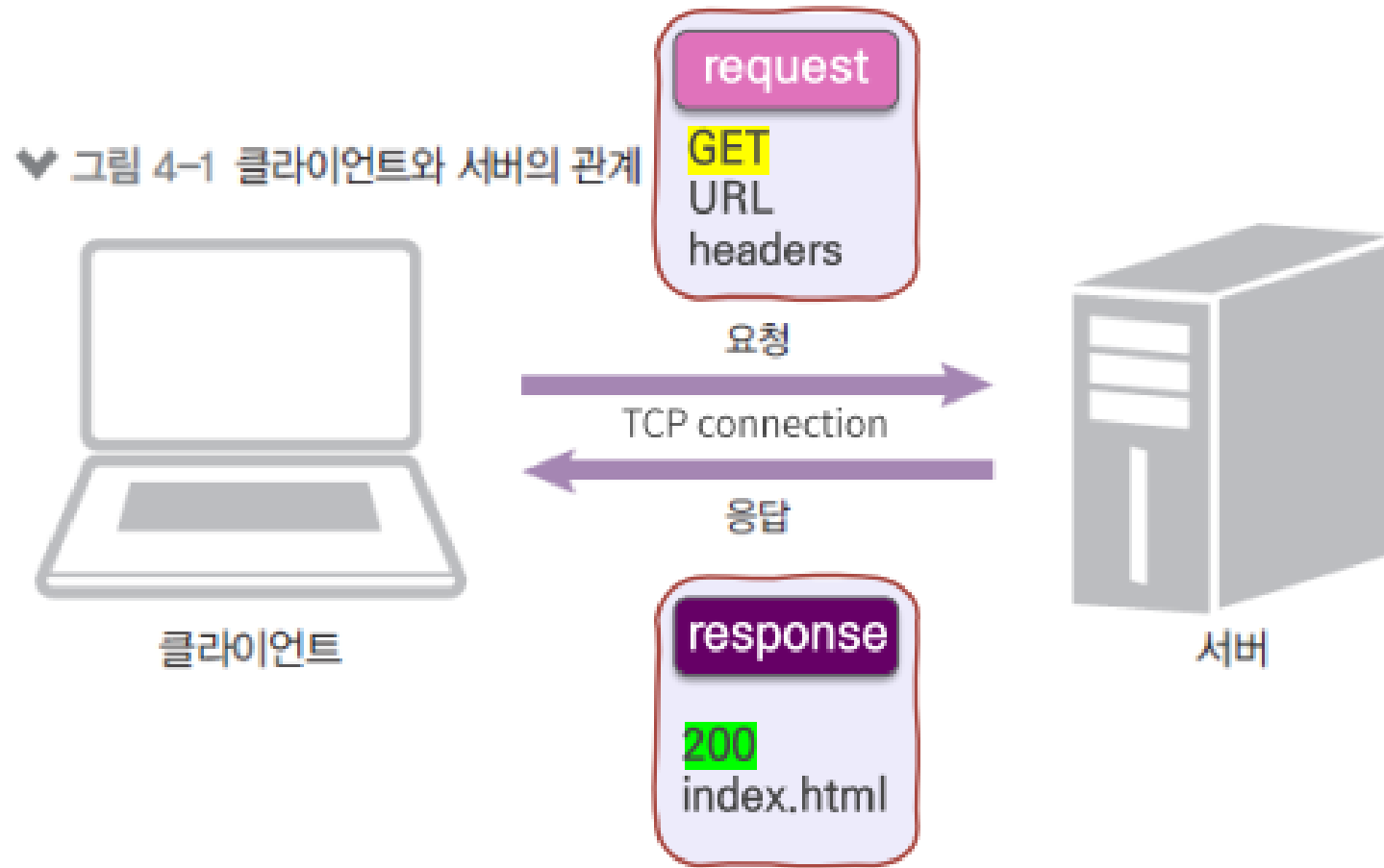
package.json

- 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",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

서버 만들기

http 통신



http 모듈

- Nodejs 를 통해 서버를 구축하는 방법
 - http
 - express
- http 모듈
 - 웹 서버를 구동하기 위한 node.js 내장 웹 모듈
 - server 객체, request 객체, response 객체를 사용한다.
 - server 객체 : 웹 서버를 생성할 때 사용하는 객체
 - response 객체 : 응답 메시지를 작성할 때 두 번째 매개변수로 전달되는 객체
 - request 객체 : 응답 메시지를 작성할 때 첫 번째 매개변수로 전달되는 객체

http 모듈 서버 만들기

```
const http = require('http');  
  
const server = http.createServer();  
  
server.listen(8080, function(){  
  console.log( '8080번 포트로 서버 실행' );  
});
```

listen(port, callback)

: 서버를 첫번째 매개변수의 포트로 실행한다.

http 모듈 서버 만들기

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

const server = http.createServer( function(req, res){
  res.writeHead( 200 );
  res.write( "<h1>Hello!</h1>" );
  res.end("<p>End</p>");
});

server.listen(8080, function(){
  console.log( '8080번 포트로 서버 실행' );
});
```

Response 객체

writeHead : 응답 헤더 작성

write : 응답 본문 작성

end : 응답 본문 작성 후 응답 종료

localhost 와 port

- localhost

- localhost는 컴퓨터 내부 주소 (127.0.0.1)
- 자신의 컴퓨터를 가리키는 호스트이름(hostname)

- port

- 서버 내에서 데이터를 주고받는 프로세스를 구분하기 위한 번호
- 기본적으로 http 서버는 80번 포트 사용 (생략 가능, https는 443)

server 객체

listen()	서버를 실행하고 클라이언트를 기다린다.
close()	서버를 종료한다.
on()	server 객체에 이벤트를 등록한다.

request	클라이언트가 요청할 때 발생하는 이벤트
connection	클라이언트가 접속할 때 발생하는 이벤트
close	서버가 종료될 때 발생하는 이벤트
checkContinue	클라이언트가 지속적인 연결을 하고 있을 때 발생하는 이벤트
upgrade	클라이언트가 http 업그레이드를 요청할 때 발생하는 이벤트
clientError	클라이언트에서 오류가 발생할 때 발생하는 이벤트

server 객체 - 이벤트

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

const server = http.createServer( function(req, res){
  res.writeHead( 200 );
  res.write( "<h1>Hello!</h1>" );
  res.end("<p>End</p>");
});

server.on('request', function(code){
  console.log( "request 이벤트" );
});

server.on('connection', function(code){
  console.log( "connection 이벤트" );
});

server.listen(8080, function(){
  console.log( '8080번 포트로 서버 실행' );
});
```

html 파일 전송

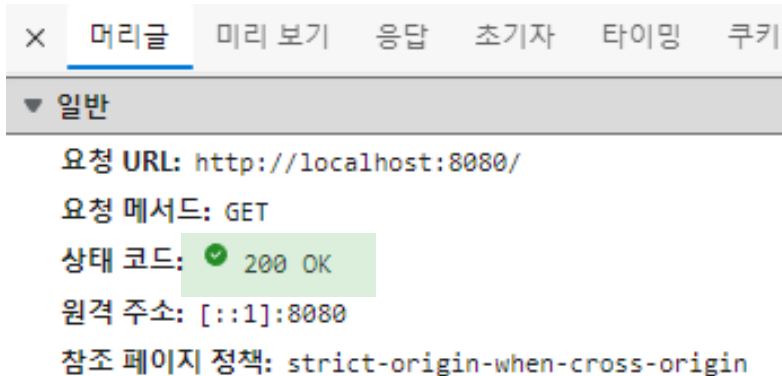
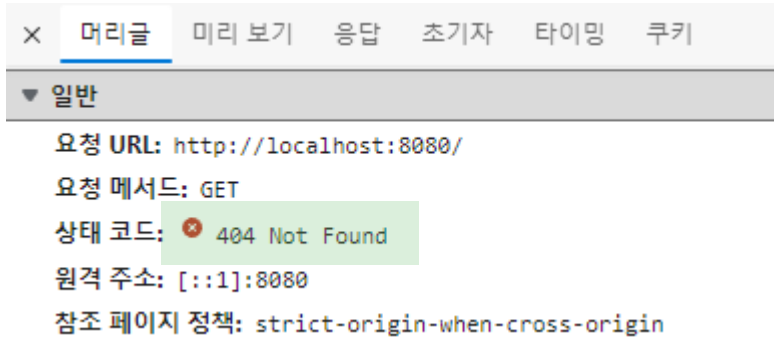
```
<html>
  <head>
    <title>http모듈</title>
  </head>
  <body>
    <h1>Hello http</h1>
    <p>p태그</p>
  </body>
</html>
```

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

const server = http.createServer((req, res) => {
  try {
    const data = fs.readFileSync("index.html");
    res.writeHead(200);
    res.write(data);
    res.end();
  } catch (err) {
    console.error(err);
    res.writeHead(404);
    res.write(err.message);
    res.end();
  }
});

server.listen(8000, () => {
  console.log(`http://localhost:8000`);
});
```

http 응답



- 1XX : 처리중
 - 100: Continue, 102: Processing
- 2XX : 성공
 - 200: OK, 201: Created, 202: Accepted
- 3XX : 리다이렉트(다른 페이지로 이동)
- 4XX : 요청 오류
 - 400: 잘못된 요청, 401: 권한 없음, 403: 금지됨
 - 404: 찾을 수 없음(Page not found)
- 5XX : 서버 오류

Express 모듈

Express

- 웹 서버를 생성하는 것과 관련된 기능을 담당하는 프레임워크
- 웹 애플리케이션을 만들기 위한 각종 메소드와 미들웨어 등이 내장되어 있다.
- http 모듈 이용 시 코드의 가독성↓, 확장성↓
→ 이를 해결하기 위해 만들어진 것이 **Express 프레임워크**

Express 설치

```
> npm install express
```

- `npm_modules` 가 만들어지며 `express`에 관련된 폴더가 생성
- `package.json`의 `dependencies` 에 `express` 기록

```
> node_modules
```

```
"dependencies": {  
  "express": "^4.18.1"  
}
```

.gitignore



```
.gitignore U X
.gitignore
1    /node_modules
2    package-lock.json
```

npm 사용

1. (프로젝트를 진행할 폴더에서) `npm init` → (npm 을 사용할거야!)
 - `package.json` : 프로젝트 폴더에서 `npm init` 명령어를 사용하는 순간 자동으로 생기는 파일, 모듈의 정보(버전 등)와 프로젝트 정보 등을 담고 있음
2. 사용하고 싶은 모듈 설치하는 npm `모듈이름`
 - `node_modules` : 모듈 저장 공간, 모듈을 설치하게 되면 이 곳에 설치됨

node_modules는 용량이 너무 커요

- node_modules는 아주 많은 폴더와 파일로 이루어져 있기 때문에 용량이 아주 큽니다.
- 깃허브에 올리지 않아요! 항상 .gitignore에 추가해주세요!
- 실제 모듈이 없으면 다른 컴퓨터에서는 사용할 수 없는데, node_modules를 github에 올리지 않는다면, 어떻게 할까요?
 - 모듈에 대한 모든 정보를 가지고 있는 package.json이 있기 때문에 문제 없습니다!
 - node_modules가 없어도 package.json이 있다면 node_modules를 설치할 수 있어요 `npm install`
 - 바로 `npm install`이라는 명령어를 통해서요!

.gitignore

.gitignore?

- Git 버전 관리에서 제외할 파일 목록을 지정하는 파일
- Git 관리에서 특정 파일을 제외하기 위해서는 git에 올리기 전에 .gitignore에 파일 목록을 미리 추가해야 한다.

.gitignore

***.txt** → 확장자가 txt로 끝나는 파일 모두 무시

!test.txt → test.txt는 무시되지 않음.

test/ → test 폴더 내부의 모든 파일을 무시 (b.exe와 a.exe 모두 무시)

/test → (현재 폴더) 내에 존재하는 폴더 내부의 모든 파일 무시 (b.exe 무시)

```
(base) [07:25 PM] cwjcsk:~/99_test/99_tmp$ tree -a
.
├── .gitignore
├── test
│   └── b.exe
└── tmp
    └── test
        └── a.exe

3 directories, 3 files
```

Express 사용

```
const express = require('express');
const app = express();
const PORT = 8000;

app.get('/', function (req, res) {
  res.send('hello express');
});

app.listen(PORT, function () {
  console.log(`Listening on port ${PORT}! http://localhost:${PORT}`);
});
```


Express 사용

- **express()**
 - Express 모듈이 export 하는 최상위 함수로, **express application**을 만듦
- **app 객체**
 - Express() 함수를 호출함으로써 만들어진 **express application**

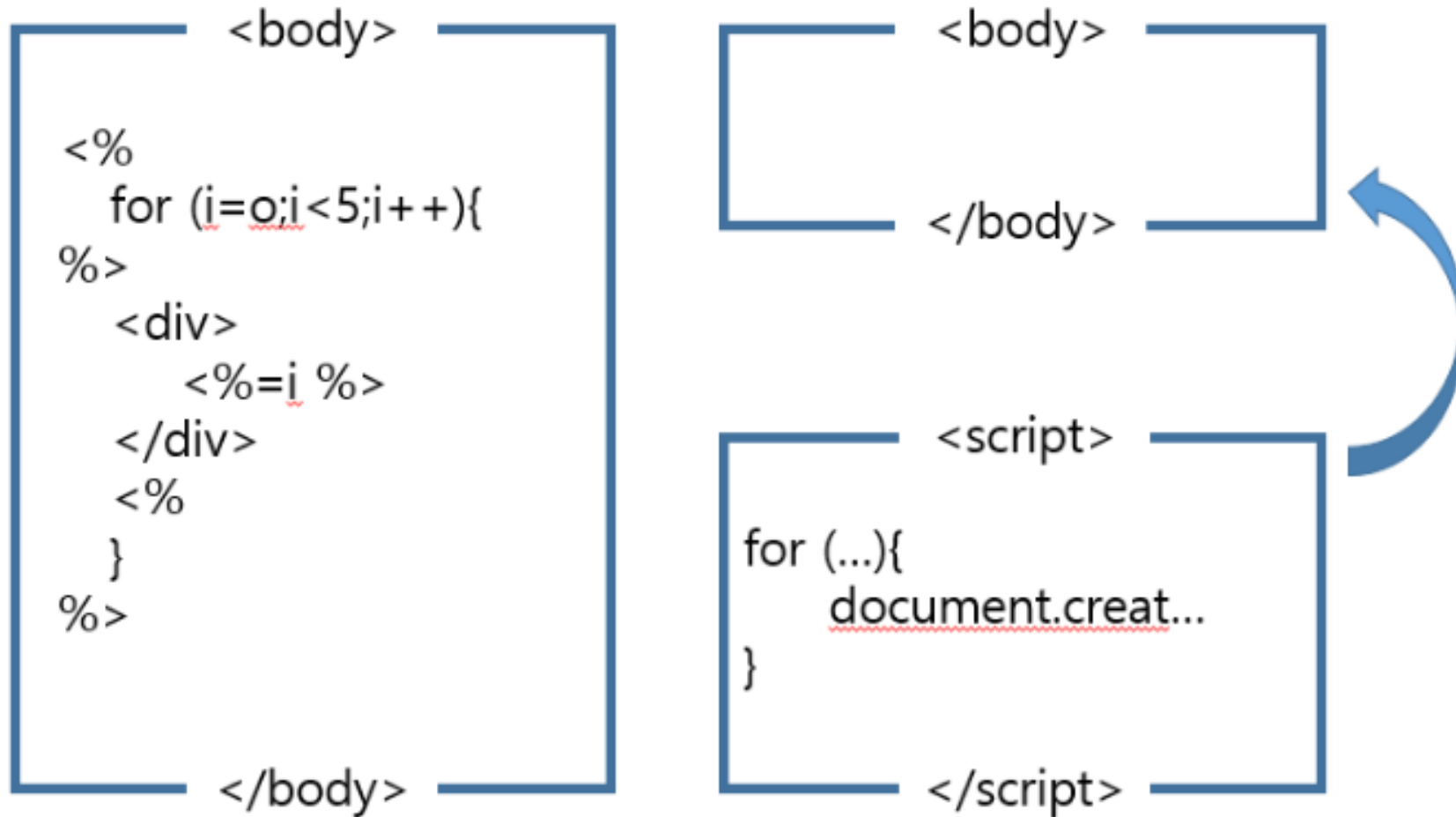
```
1  const express = require('express');  
2  const app = express();
```

템플릿 엔진

EJS 템플릿

- 템플릿 엔진
 - 문법과 설정에 따라 파일을 html 형식으로 변환시키는 모듈
- ejs
 - Embedded JavaScript 의 약자로, 자바스크립트가 내장되어 있는 html 파일
 - 확장자는 .ejs

ejs 템플릿



ejs 템플릿

```
$ npm install ejs
```

```
app.set('view engine', 'ejs');  
app.set('views', './views');
```

ejs 템플릿

```
const express = require("express");
const app = express();
const PORT = 8000;

app.set("view engine", "ejs");
app.set("views", "./views");

app.get("/", (req, res) => {
  res.send("Hello Express");
});

app.get("/test", (req, res) => {
  res.render("test");
});

app.listen(PORT, () => {
  console.log(`http://localhost:${PORT}`);
});
```



ejs 템플릿 설정



ejs 템플릿 렌더링

ejs 템플릿

```
<html>
  <head>
    <title>EJS TEST</title>
  </head>
  <body>
    <% for (var i = 0; i < 5; i++) { %>
      <h1>안녕</h1>
    <% } %>
  </body>
</html>
```

ejs 문법 사용하기

```
<% %>
```

- 무조건 자바스크립트 코드가 들어가야 하고, 줄바꿈을 할 경우에는 새로운 `<% %>` 를

```
<%= %>
```

- 값을 템플릿에 출력할 때 사용

```
<%- include('view의 상대주소') %>
```

- 다른 view 파일을 불러올 때 사용

미들웨어

- 요청이 들어옴에 따라 응답까지의 **중간 과정을 함수로 분리한 것**
- **서버와 클라이언트를 이어주는 중간 작업**
- **use()** 를 이용해 **등록**할 수 있다.

```
app.set('view engine', 'ejs');  
app.use('/views', express.static(__dirname + '/views'));
```

미들웨어 - static

- 이미지, CSS 파일 및 JavaScript 파일(front)과 같은 정적 파일 제공
- Express 에 있는 static 메소드를 이용해 미들웨어로 로드
- 등록 방법

```
app.use('/static', express.static(__dirname + '/static'));
```

ejs 템플릿

```
const express = require("express");
const app = express();
const PORT = 8000;

app.set("view engine", "ejs");
app.set("views", "./views");
app.use("/public", express.static(__dirname + "/public"));

app.get("/", (req, res) => {
  res.send("Hello Express");
});

app.get("/test", (req, res) => {
  res.render("test");
});

app.listen(PORT, () => {
  console.log(`http://localhost:${PORT}`);
});
```



정적 파일 로드 코드
[keyword] 미들웨어, static