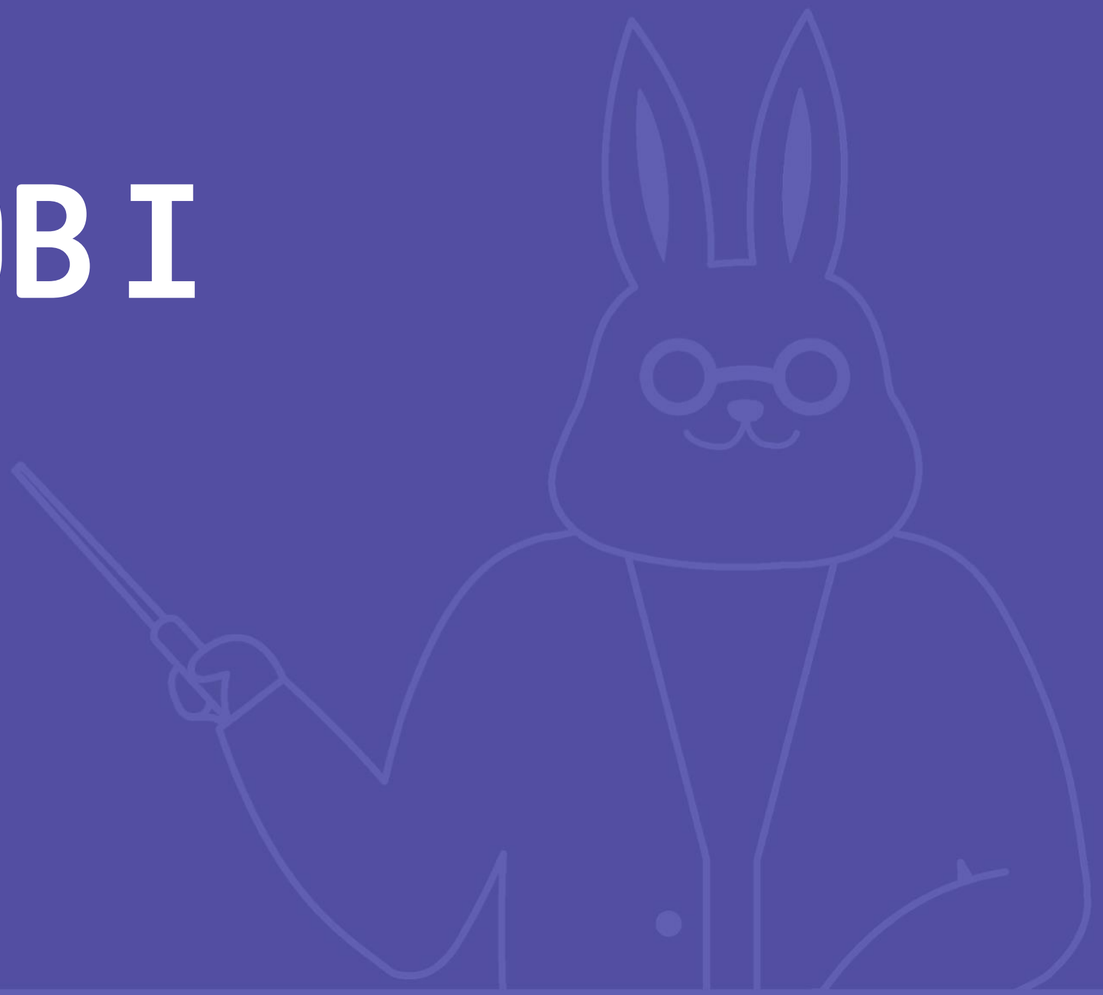
Node.js와 MongoDB I

00 수업 소개



/* elice */

커리큘럼

Node.js 이해하기

Node.js 의 기초부터 심화까지 알아보며
Node.js 에 대해 정확하게 이해합니다.

NPM과 모듈

NPM 을 이용한 프로젝트 관리방법에 대해 학습하고,
Node.js 모듈의 작성과 사용 방법에 대해 학습합니다.

커리큘럼

웹과 Express.js

웹의 기본 개념에 대해 이해하고
Node.js 의 웹 프레임워크인 Express.js 에 대해
학습합니다.

Express.js와 REST API

REST API 의 개념에 대해 이해하고
Express.js 를 이용하여 간단한 REST API를 작성해 봅니다.

💡 추천대상

1. 자바스크립트에 익숙한 프론트엔드 개발자

자바스크립트에 익숙한 사용자가 간단하게 백엔드를 개발할 수 있는 방법에 대해 학습할 수 있습니다.

2. 웹 서비스의 구성을 정확하게 이해하고 싶으신 분

웹 서비스의 구성요소와 동작원리를 하나씩 알아보고, 직접 작성하며 학습할 수 있습니다.

3. 프로젝트를 코드베이스 없이 시작하고 싶으신 분

실습을 통해 프로젝트를 시작하고 관리하는 방법에 대해 알아보며 코드베이스가 없는 상황에서도 프로젝트를 시작할 수 있습니다.

🎯 수강목표

1. Node.js 의 코드 작성방법에 대해 학습합니다.

Node.js 의 동작 원리부터 이해하여 정확한 방법으로 코드를 작성하는 방식에 대해 학습합니다.

2. Express.js 사용 방법에 대해 학습합니다.

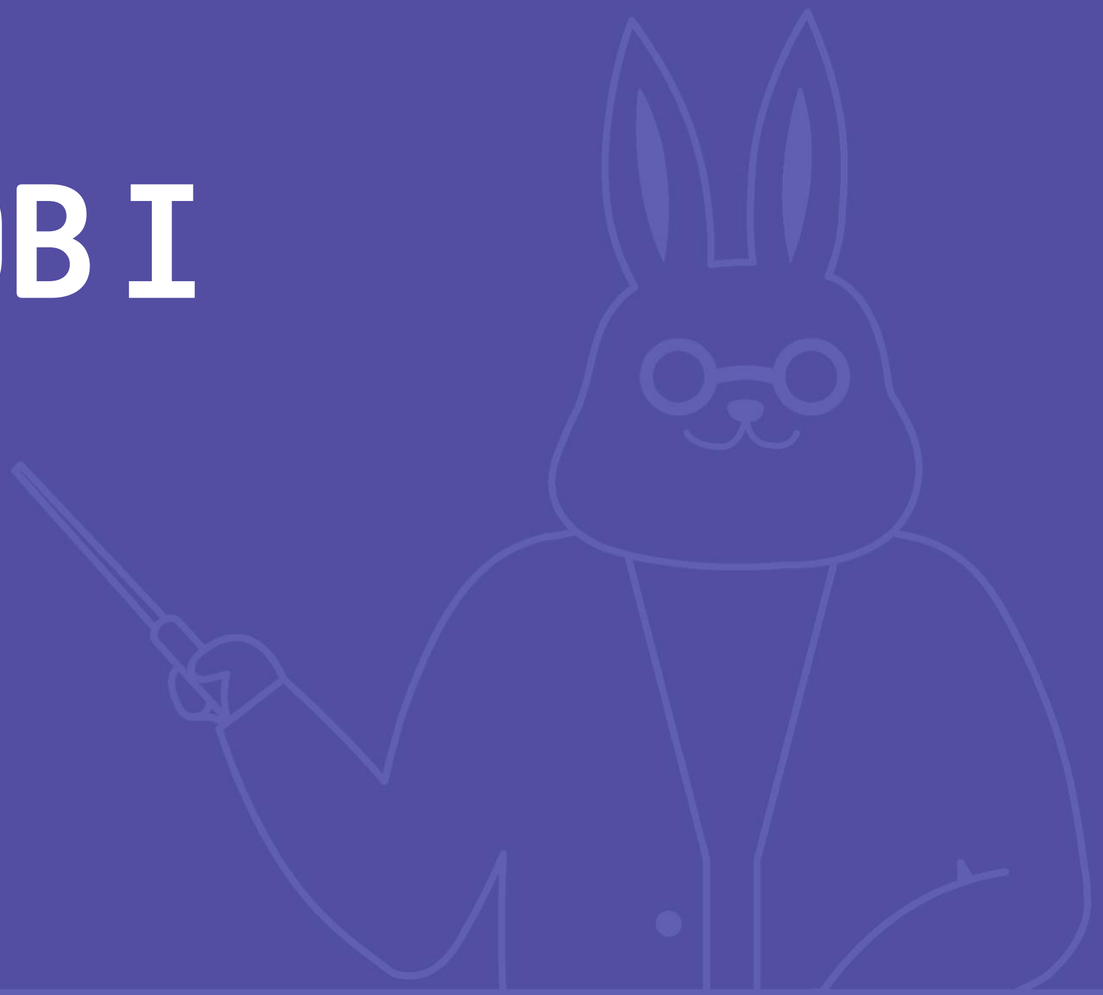
Express.js 의 기본 사용방법과 프로젝트 구성방법에 대해 학습하며, 웹서비스의 동작 원리까지 이해 할 수 있습니다.

3. REST API 를 구성할 수 있습니다.

REST API 에 대해 이해하고 Express.js 를 활용하여 REST API를 구성하는 방법에 대해 학습합니다.

Node.js와 MongoDB I

01 Node.js 이해하기



목차

🎯 수강목표

1. Node.js 이해하기

Node.js의 등장 배경과 최근 가장 핫 한 기술이 된 이유에 대해 알아봅니다.

2. Node.js와 익숙해지기

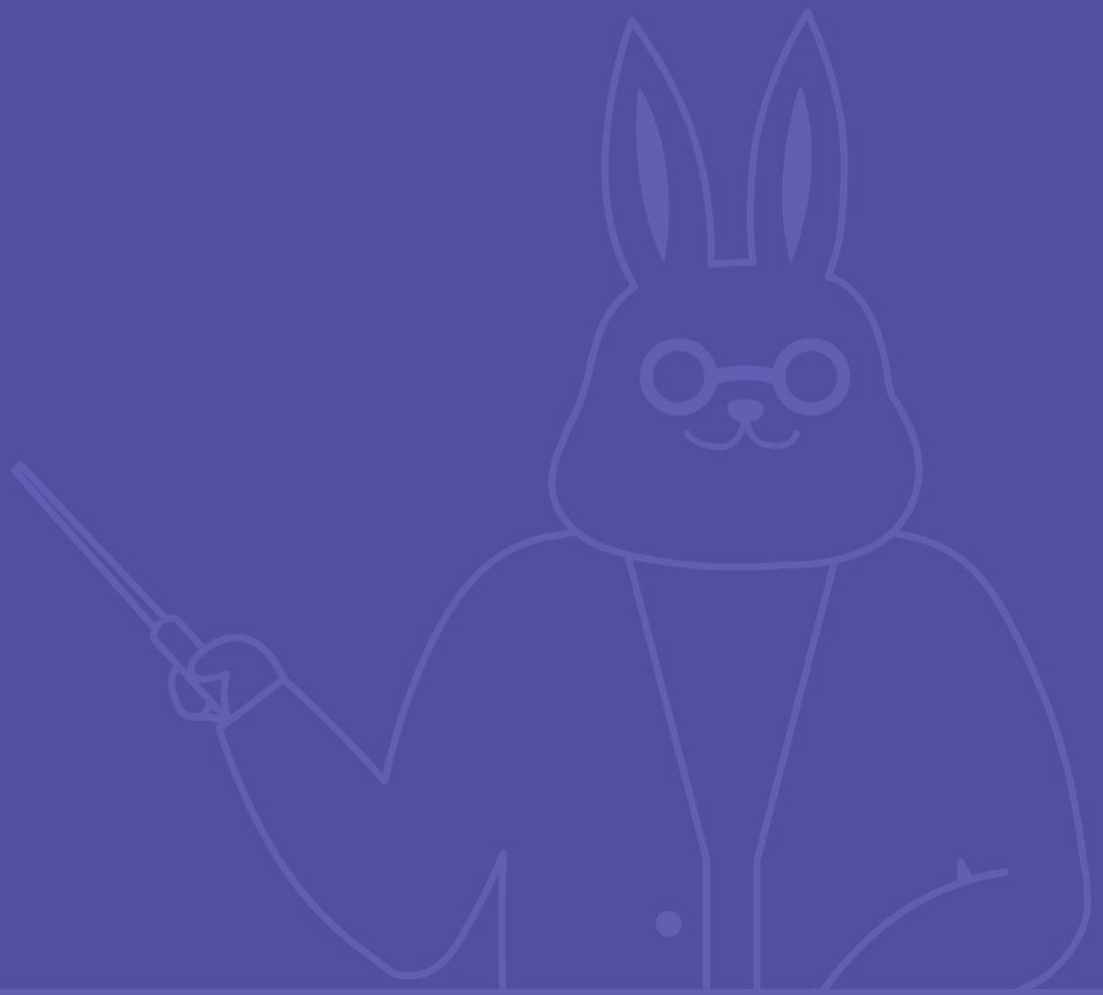
Node.js와 ES6를 사용해보며 Node.js 와 친숙해지는 시간을 가집니다.

3. 심화

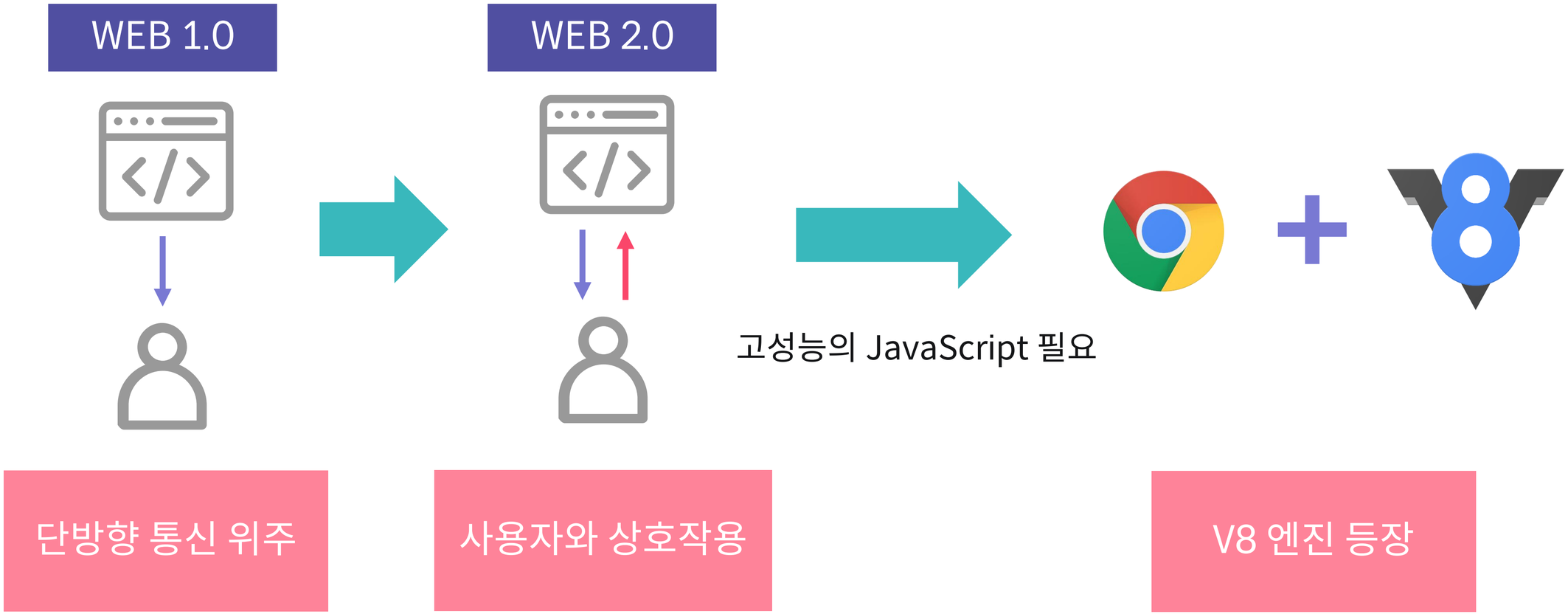
Node.js 의 동작 원리에 대해 더욱 깊게 이해합니다.

01

Node.js 이해



Node.js의 등장 배경



Node.js의 탄생

고성능 JavaScript
실행 가능

JavaScript를 브라우저
외부에서 사용해볼까?

Node.js 탄생

Node.js는 자바스크립트를 어느 환경에서나 실행할 수 있게 해주는 실행기

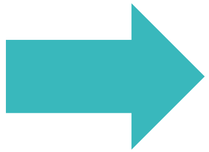
Node.js vs Browser

Browser의 JavaScript

브라우저에서 실행

웹 내부 제한된 동작

웹 프론트 개발자의 언어

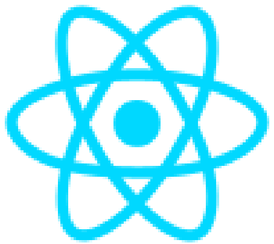
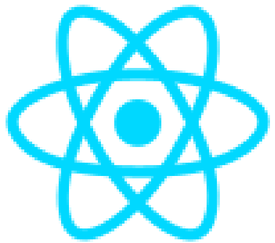


크로스 플랫폼 실행

제한 없는 동작

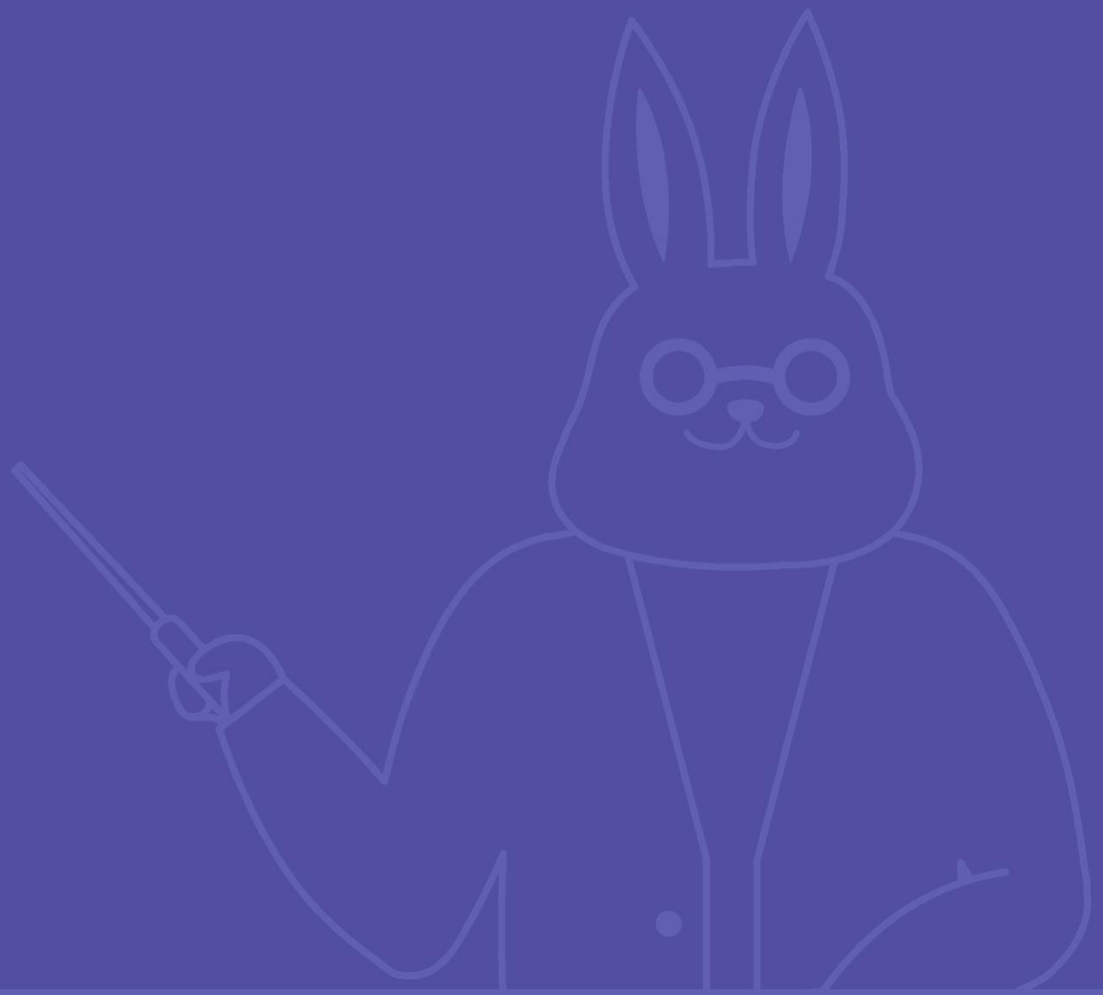
다양한 어플리케이션 개발

Node.js 로 할 수 있는 것들

Front-End	Back-End	Mobile-App	Desktop-App	Machine-Learning
				
React.js	Express.js	React-Native	Electron	Brain.js
최근 가장 인기 있는 웹서비스 구성	한가지 코드로 iOS와 Android 개발	Discord Slack 등 앱 개발	JavaScript로 구현하는 딥러닝	

02

Node.js의 특징



Node.js의 특징 한 줄 요약

싱글 쓰레드 - 비동기 - 이벤트 기반

쓰레드란?

쓰레드란 **명령을 실행하는 단위**
한 개의 쓰레드는 **한 번에 한 가지 동작만** 실행 가능

싱글 쓰레드와 멀티 쓰레드의 차이점

싱글 쓰레드 - 한 번에 한가지 동작만 수행함

멀티 쓰레드 - 동시에 여러 동작 수행 가능

그렇다면 싱글 쓰레드는 안 좋은 것 아닌가요?

장점 - 쓰레드가 늘어나지 않기 때문에 리소스 관리에 효율적
단점 - 쓰레드 기반의 작업들의 효율이 떨어짐 Ex) CPU 연산 작업

그래서 Node.js 는 **비동기 동작으로 쓰레드 기반의 작업을 최소화**합니다.

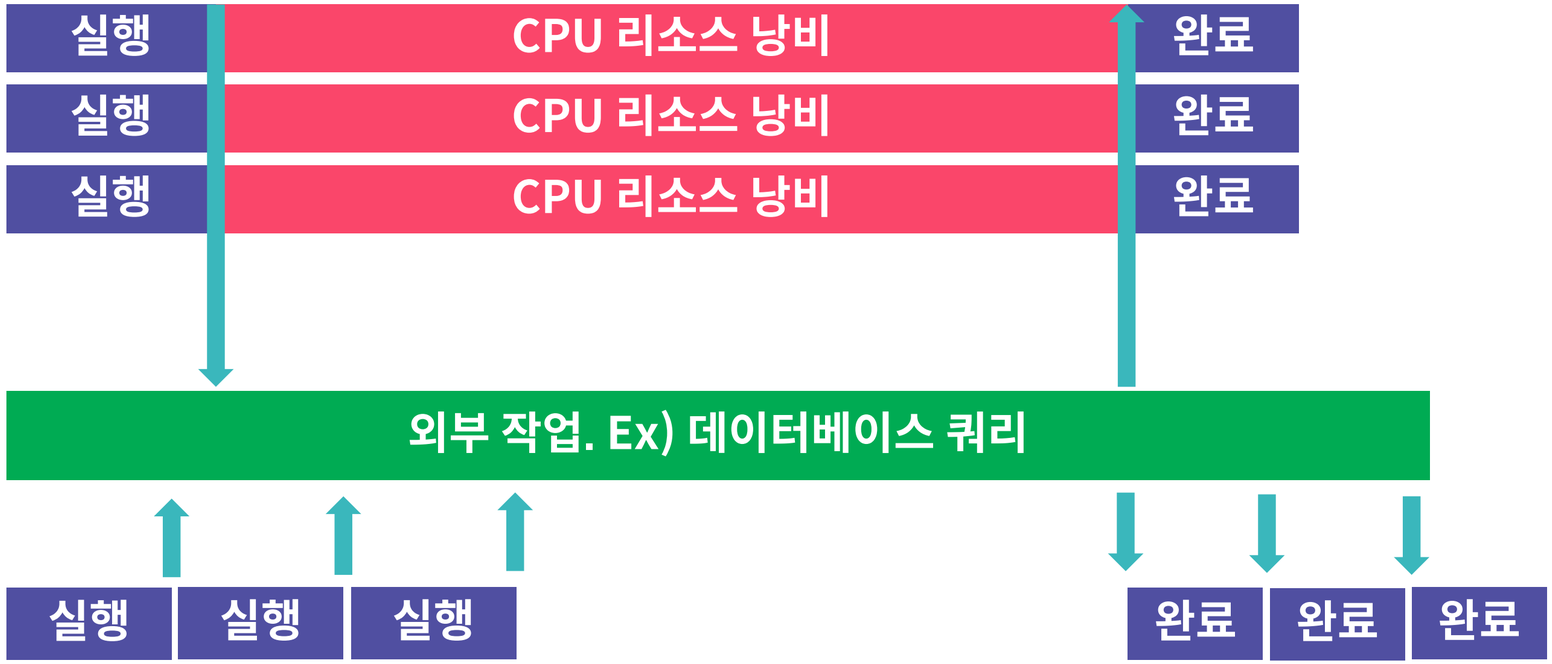
비동기란?

동작을 실행한 후 완료가 되길 **기다리지 않는 방식**
동작의 완료를 기다리지 않기 때문에 **다른 동작을 바로 실행 가능**
Node.js 는 **싱글 쓰레드이기 때문**에 **비동기 방식**을 사용함

동기와 비동기의 차이

멀티 쓰레드
동기 방식

싱글 쓰레드
비동기 방식

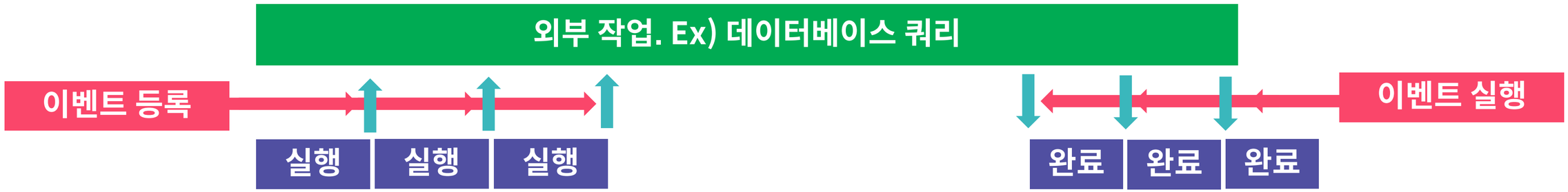


동기와 비동기 방식의 차이

이벤트 기반이란?

비동기 동작의 완료를 처리하는 방법
비동기 방식은 특정 동작을 **실행한 후**, 해당 동작을 **전혀 신경 쓰지 않음.**
대신 해당 동작이 **완료될 경우 실행할 함수**를 **미리 등록**함.
비동기 동작이 **완료가 되면** 미리 등록된 **함수를 실행.**

이벤트 기반



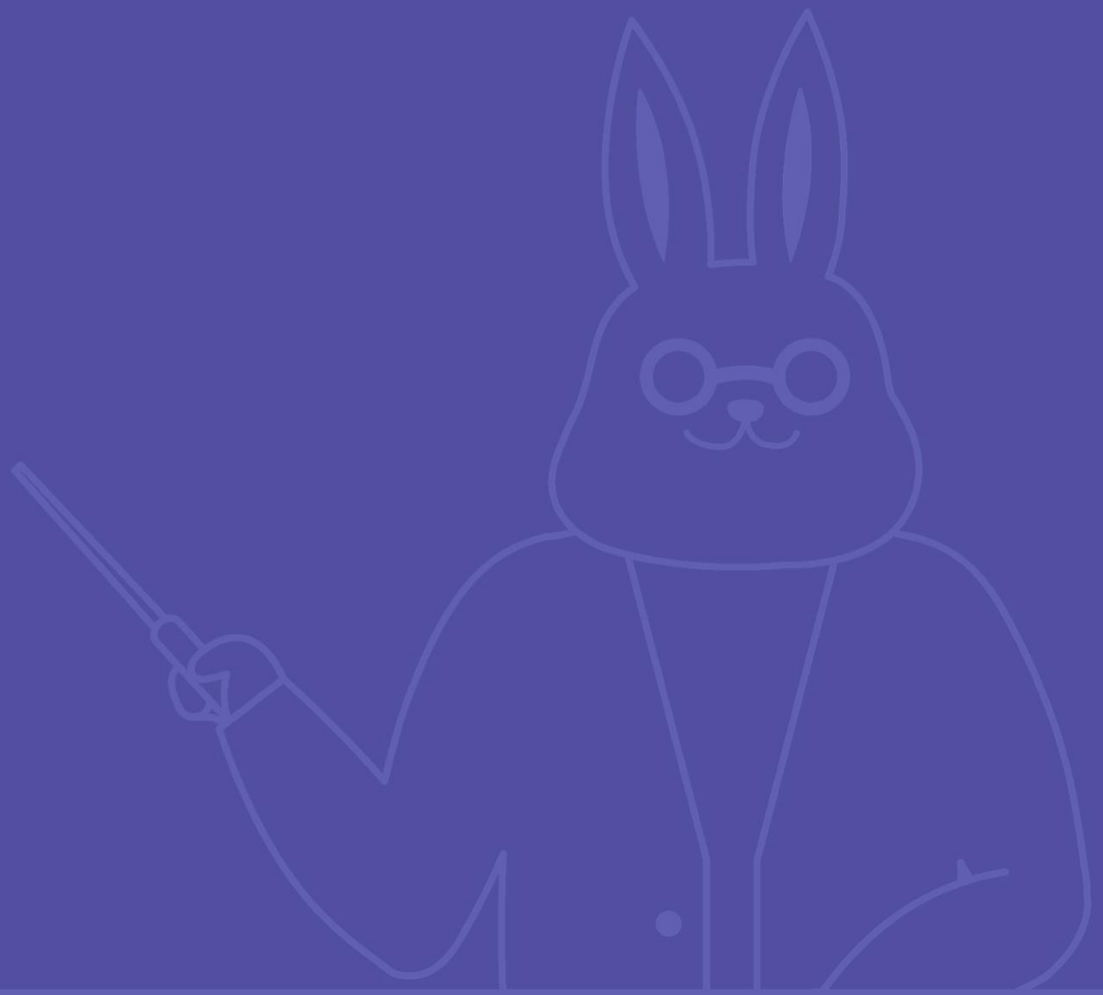
이벤트 기반 동작 방식

Node.js의 특징 요약

싱글 쓰레드**이기 때문에** 비동기 동작 필요
비동기 동작을 **구현하기 위해** 이벤트 기반

03

Node.js 시작하기



어떤 버전으로 시작해야 할까?

Node.js는 빠르게 개발 중

보안 이슈 및 버그 수정, 최신기술 빠르게 적용

급변하는 기술은 가장 안정적인 최신 버전을 선택하는 것이 최선

LTS

Long-Term Support 버전

Node.js의 안정적이고, 오래 지원하는 버전 명

어떤 버전으로 시작해야 할까?

16.13.0 LTS
안정적, 신뢰도 높음

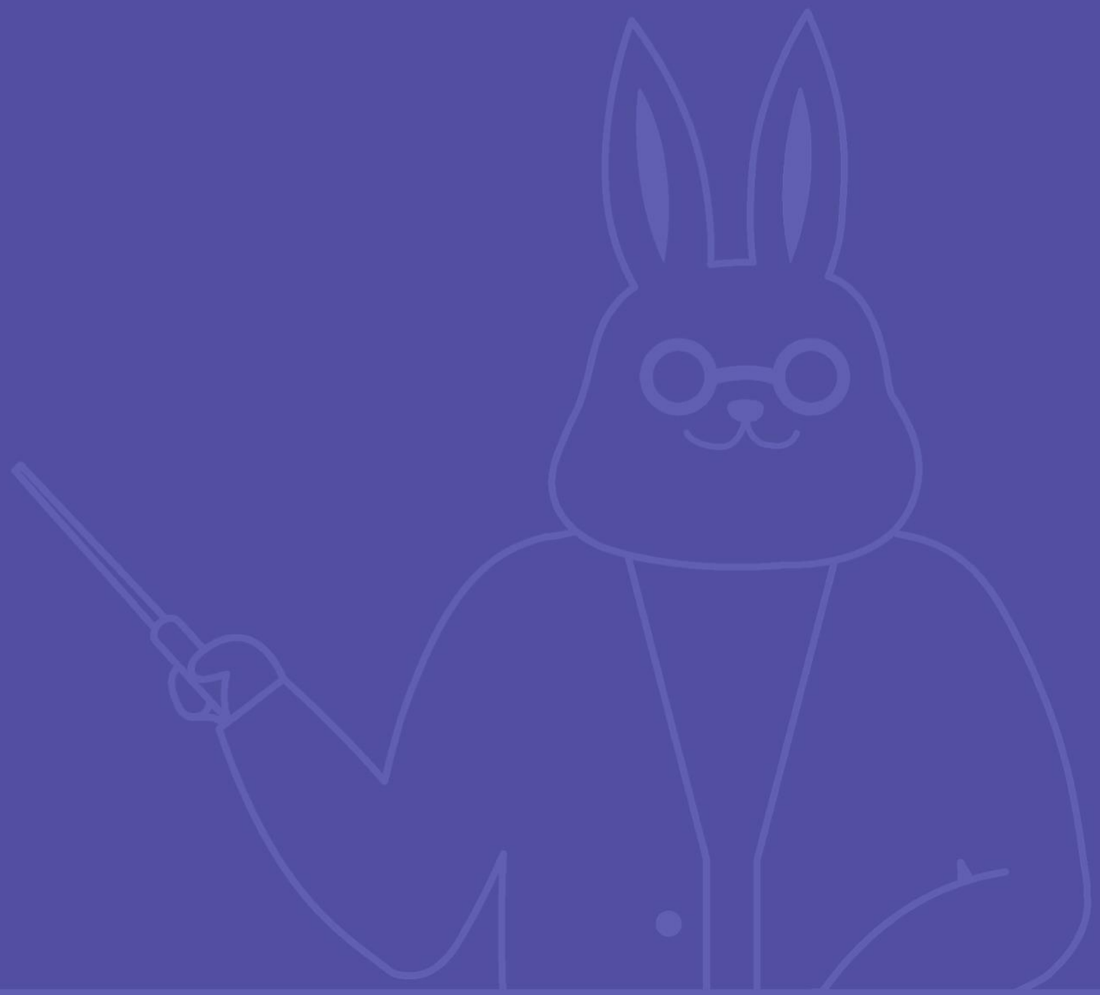
17.0.1 현재 버전
최신 기능

본 강의 기준버전

안정적이고 오래 지원되는 버전 vs 가장 최신기술이 적용된 버전

04

ES6



ES6란?

ES6

ECMAScript 버전 6 이후를 통틀어 일반적으로 ES6라고 부름

ECMAScript

계속해서 발전해가는 JavaScript의 **표준문법**

2015년, **ECMAScript 버전 6 이후**로 많은 **현대적인 문법**이 추가됨

ES6를 사용하는 이유?

현대적인 문법은 **생산성 향상**에 도움을 줌
Node.js 는 빠르게 **최신 ECMAScript를 지원 중**
자주 사용되는 **유용한 문법**을 익히고 필요한 부분에 **적절하게 활용**하는 것이 중요

Node.js 와 ES6

Node.js는 ES6의 **모든 문법을 지원하지는 않음**
Node.js로 **자주 사용되는 유용한 ES6문법**의 코드를 실행해보며
Node.js와 친숙해지는 시간을 가져 봅시다.

자주 사용되는 문법 1 - let, const

기존 문법

```
// 상수와 변수 구분이 없음
var TITLE = 'NODE.JS';
var director = 'elice';
director = 'rabbit';
TITLE = 'ES6' // 오류 없음
```

ES6

```
// 상수와 변수 구분 가능
const TITLE = 'NODE.JS';
let director = 'elice';
director = 'rabbit';
TITLE = 'ES6'; // 오류 발생
```

자주 사용되는 문법 2 - Template String

기존 문법

```
var name = 'elice';
var age = 5;
// + 를 사용해 문자열과 변수 연결
// 줄 바꿈 문자 \n 사용 필요
var hi = 'My name is '
  + name
  + '.\n I\'m '
  + age
  + 'years old.';
console.log(hi);
```

ES6

```
const name = 'elice';
const age = 5;
// 문자열 사이에 간단하게 변수 사용 가능
// 따옴표 간단하게 사용 가능
// 줄 바꿈 지원
const hi =
`My name is ${name}.
I'm ${age} years old`;
console.log(hi);
```

자주 사용되는 문법 3 - arrow function

기존 문법

```
// 기본 함수 표현 방법
function doSomething(param) {
  console.log('do something');
}

// 익명 함수 표현 방법
setTimeout(function(param) {
  console.log('no name function');
}, 0)

// 함수 새로 선언 가능
function doSomething () {
  console.log('do other');
}
```

ES6

```
// 상수형으로 표현 가능
const doSomething = (param) => {
  console.log('do something');
}

// 익명함수 간결하게 표현 가능
setTimeout((param) => {
  console.log('no name function');
}, 0)

// 함수 새로 선언 불가능
doSomething = () => {
  console.log('do other');
}
```

자주 사용되는 문법 4 - class

기존 문법

```
function Model(name, age) {
  this.name = name;
  this.age = age;
}
// prototype으로 class 함수 구현
Model.prototype.getInfo = function() {
  console.log(this.name, this.age);
}
var model = new Model('elice', 5);
model.getInfo();
```

ES6

```
// 일반적인 형태의 class 구현 가능
class Model {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  getInfo() {
    console.log(this.name, this.age);
  }
}
const model = new Model('elice', 5);
model.getInfo();
```

자주 사용되는 문법 5 - destructing

기존 문법

```
var obj = {name: 'elice', age: 5};
var name = obj.name;
var age = obj.age;

var arr = ['some', 'values'];
var first = arr[0];
var second = arr[1];
```

ES6

```
const obj = {name: 'elice', age: 5};
// Object의 key와 같은 이름으로 변수 선언 가능
const { name, age } = obj;
// 다른 이름으로 변수 선언하는 방법
const { n1: name, a1: age } = obj;

const arr = ['some', 'values'];
// arr에서 순차적으로 변수 선언 가능
const [first, second] = arr;
```

자주 사용되는 문법 6 - promise, async - await

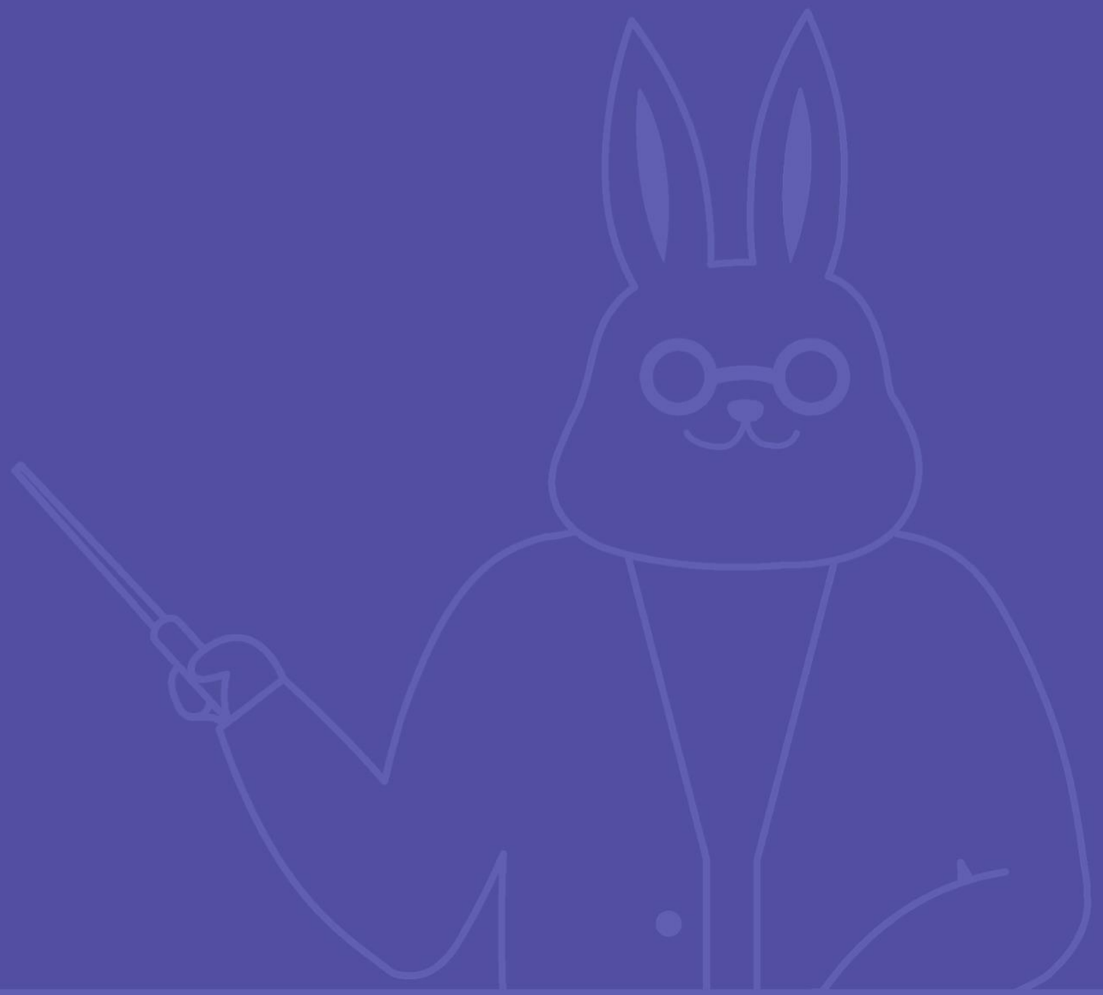
Promise와 Async - Await은 간단한 요약이 어려움
다음 장에서 비동기 코딩과 함께 학습합니다.

ES6 적용 결과

복잡하거나 **직관적이지 않던** 방법을
보기 좋게 만들고 **간결하게** 표현할 수 있게 됨.
현대적인 문법은 **처음 접할 땐 어색**하지만,
익숙해지면 **좋은 코드를 작성**할 수 있게 됨.

05

비동기 코딩



비동기 코딩이란?

비동기 - 이벤트 기반 동작을 코드로 구현하는 방법
Node.js 에서 **비동기 동작을 구현하는 세 가지 방법**을 학습

비동기 코딩의 세가지 방법

Callback
전통적인 JavaScript의 이벤트 기반 코딩 방식

Promise
callback의 단점을 보완한 비동기 코딩 방식

Async -Await
promise의 단점을 보완한 비동기 코딩 방식

Callback

get-users.js

```
db.getUsers((err, users) => {
  console.log(users);
});
```

비동기 동작

db.getUers 함수는 데이터베이스에서 유저 목록을 찾아오는 비동기 동작을 수행

이벤트 등록 / 실행

쿼리가 완료되면 오류가 있는지, 혹은 유저목록의 결과로 미리 등록된 callback 함수를 실행

참고 - callback의 표준

에러와 결과를 같이 전달하는 것이 표준으로 자리 잡혀 있음

콜백 지옥

callback-hell.js

```
db.getUsers((err, users) => {
  if (err) {
    …
    return;
  }
  async1(users, (r1) => {
    async2(r1, (r2) => {
      async3(r2, (r3) => {
        …
      });
    });
  });
});
```

async1, async2, async3 …를 동기적으로 실행해야 할 경우?
Node.js 는 기본적으로 비동기 동작을 callback으로 처리하기 때문에 계속해서 **callback의 callback의 callback의 callback** …

코드가 좋아 보이나요?

Promise의 등장

use-promise.js

```
db.getUsersPromise()
  .then((users) => {
    return promise1(users);
  })
  .then(r1 => promise2(r1))
  .catch( …  );
```

Promise 함수는 동작이 **완료되면 then**에 등록된 callback 실행.

오류가 발생한 경우 catch 에 등록된 callback 실행.

Chaining을 사용해 코드를 간결하게

Short-hand 표현 방법으로 더욱 간결하게

1. Return 생략 가능
2. 인자가 하나인 경우 () 생략 가능

callback 기반 함수를 Promise 함수로 변경하는 방법

promisify.js

```
function getUsersPromise(params) {
  return new Promise((resolve, reject) => {
    getUsers(params, (err, users) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(users);
    });
  });
}
```

Promise 는 **resolve, reject** 두 가지 함수를 가짐.
async1 함수의 **실행 결과에 따라 resolve, reject로 분리**

reject는 catch에 등록된 callback 실행
resolve는 then에 등록된 callback 실행

프로미스 지옥

promise-hell.js

```
promise1()
  .then(r1 => {
    return promise2(r1)
      .then(r2 => promise3(r1, r2))
  });
```

promise3 함수가 promise1와 promise2의 결괏값을 같이 사용하고 싶다면?

직관적으로 생각한다면 위와 같은
콜백 지옥과 유사한 해결책이 생각남.

Async - Await 의 등장

async-await.js

```
async function doSomething() => {
  cosnt r1 = await promise1();
  const r2 = await promise2(r1);
  const r3 = await promise3(r1, r2);
  …
  return r3;
});

doSomething().then(r3 => {
  console.log(r3)
});
```

async - await 은 **promise의 다른 문법**

async 함수 내에서 promise 함수의 결과는
await 으로 받을 수 있음.
await 한 promise 함수가 **완료될 때 까지**
다음 라인으로 넘어가지 않음.

순차적 프로그래밍처럼 작성 가능.

async 함수의 **return 은 Promise**

Async 함수의 오류처리

promise 오류처리

```
function doSomething(msg) {
  return promise1()
    .then(r => {
      console.log(r)
    })
    .catch(e => {
      console.error(e)
    });
}
```

async 오류처리

```
async function doSomething(msg) {
  try {
    const r = await promise1();
    console.log(r);
  } catch(e) {
    console.error(e);
  }
}
```

동일한 동작을 하는 promise 함수와 async 함수

Promise의 병렬 실행

parallel run

```
async function sync() {
  const r1 = await promise1();
  const r2 = await promise2();
  console.log(r1, r2);
}
---
async function parallel() {
  const [r1, r2] = await Promise.all([
    promise1(),
    promise2(),
  ]);
  console.log(r1, r2);
}
```

promise1과 promise2는 각 1초, 2초가 소요되는 비동기 함수
sync 예제에서는 3초의 시간이 소요.
parallel 예제에서는 2초의 시간이 소요.

Promise.all은 promise 함수를 **동시에 실행**시키고 **등록된 모든 함수가 마무리되면** 결과값을 한꺼번에 반환.

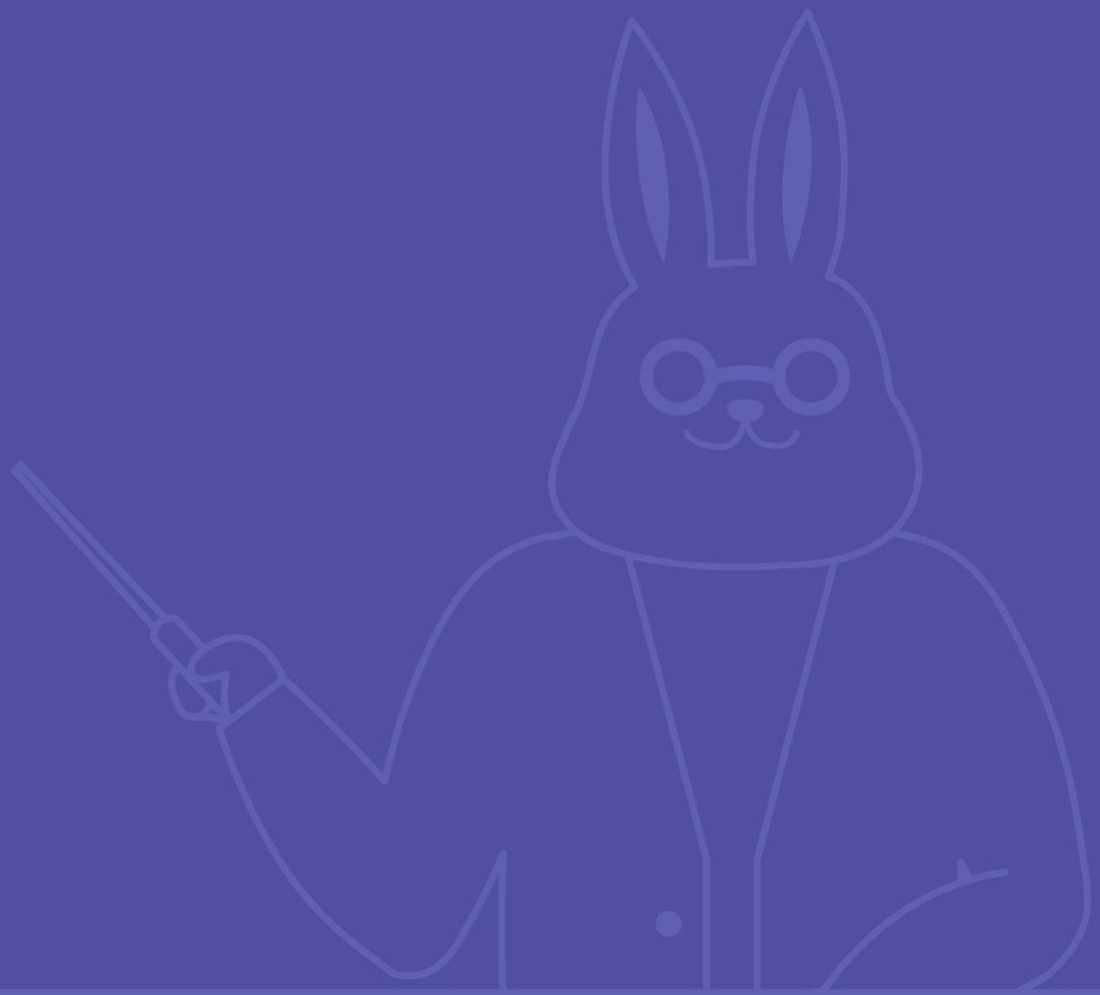
비동기 코딩 정리

callback 지옥 -> **promise chaining으로 해결**
promise 지옥 -> **async - await으로 해결**

현대 JavaScript에서는 대부분 **가독성이 좋은 async - await을 지향**하지만,
특정 상황에 맞는 비동기 코딩 방법들을 **구사할 줄 알아야 함**

06

심화 - 이벤트 루프



이벤트 루프란?

이벤트(event)를 처리하는 **반복되는 동작(loop)**

즉, Node.js가 비동기-이벤트 동작을 처리하는 일련의 반복 동작

비동기 코딩이 **어떤 순서로 수행되는지**에 대해 이해할 수 있음.

브라우저와 Node.js 의 이벤트 루프

이벤트 루프는 Node.js만의 특징은 아님.

JavaScript의 **일반적인 동작 방식**으로, 브라우저에도 있음

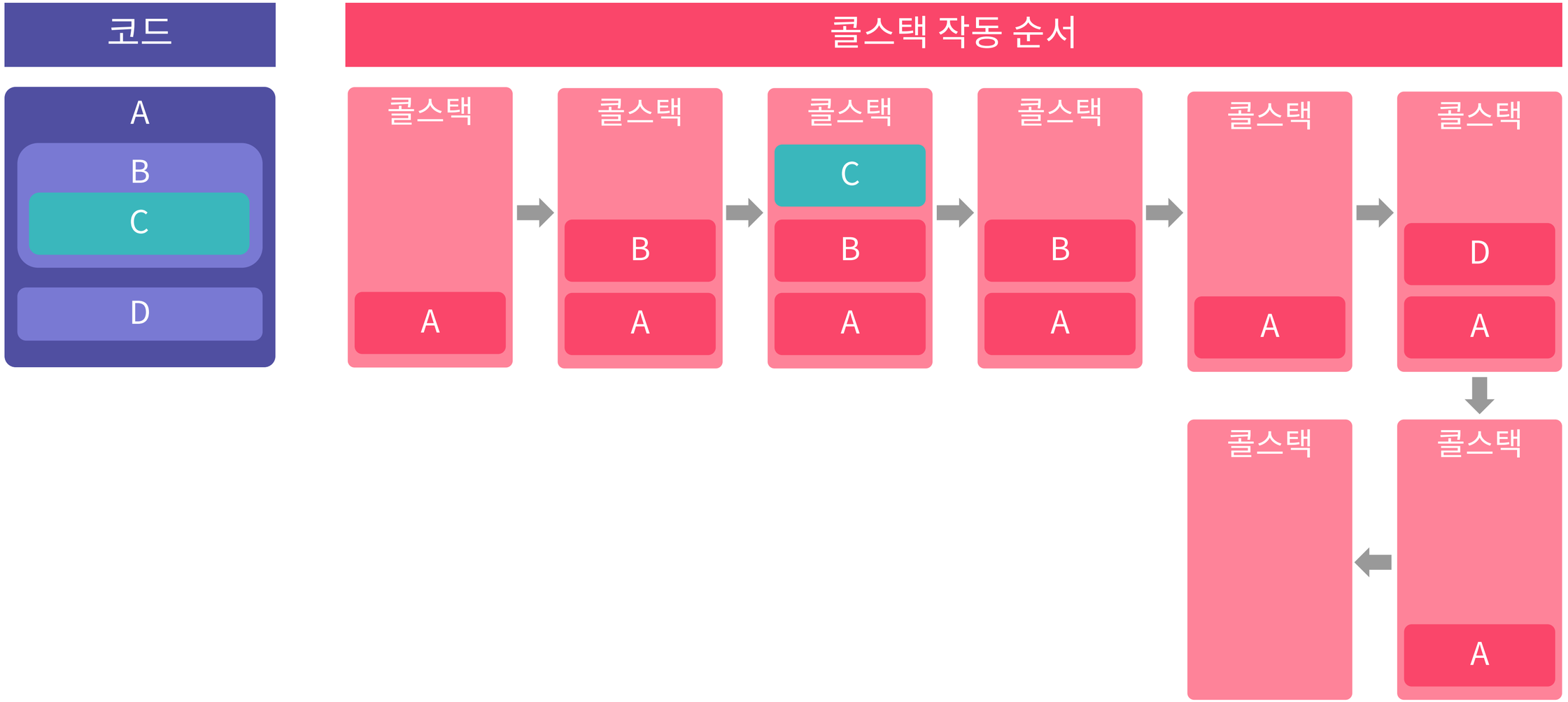
브라우저와 Node.js의 이벤트 루프는 기본적인 **동작방식에 큰 차이가 없음**

이벤트루프의 **기본적인 동작 원리를 이해하는 것**이 중요

이벤트 루프 - 구성요소

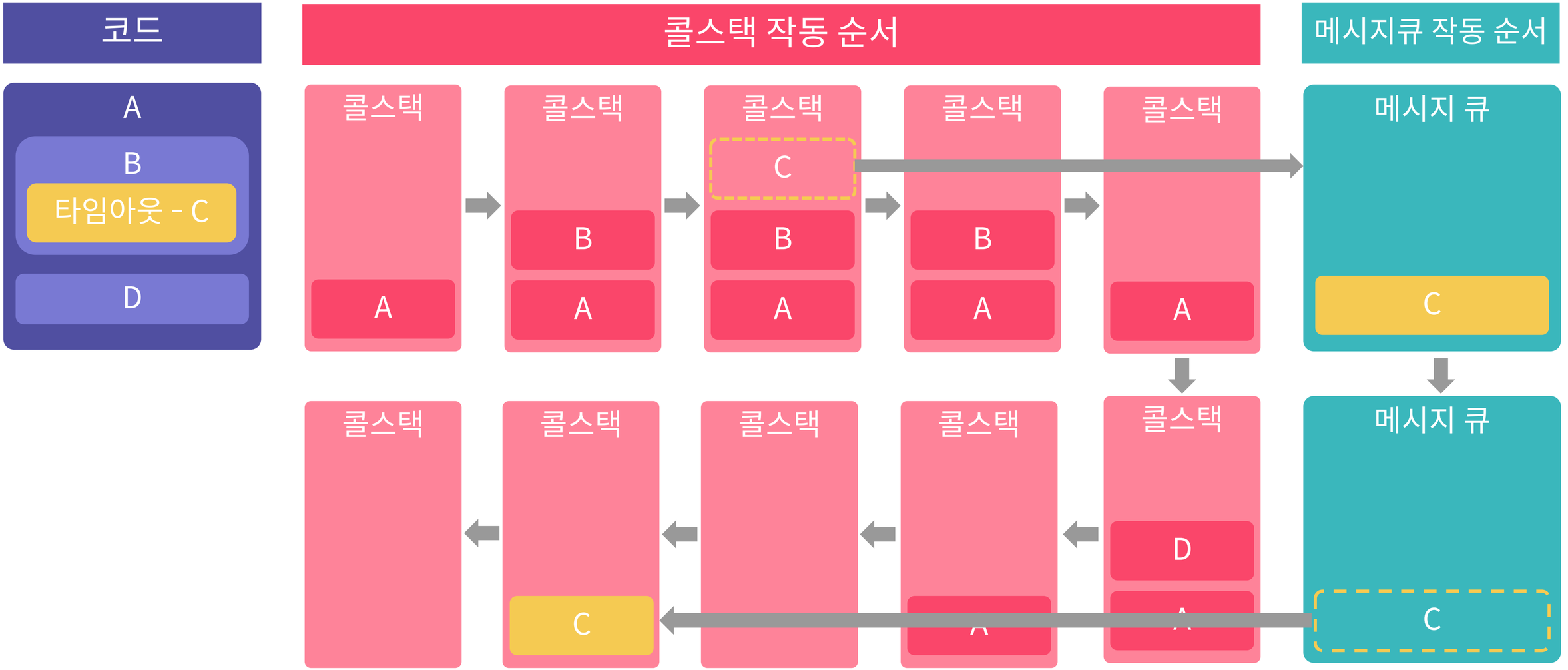
Call Stack	작성된 함수들이 등록되는 LIFO 스택 이벤트 루프는 콜스택이 비어있을 때까지 스택의 함수를 실행
Message Queue	setTimeout 같은 지연실행 함수를 등록하는 FIFO 큐 정해진 timing이 끝나고, 콜스택이 비어있을 경우 등록된 함수를 콜스택에 추가
Job Queue	Promise 에 등록 된 콜백을 등록하는 FIFO 큐 상위 함수가 종료되기 전에 콜스택이 비어있지 않더라도 잡큐에 등록된 콜백을 콜스택에 추가

이벤트 루프 - 콜스택 작동 순서



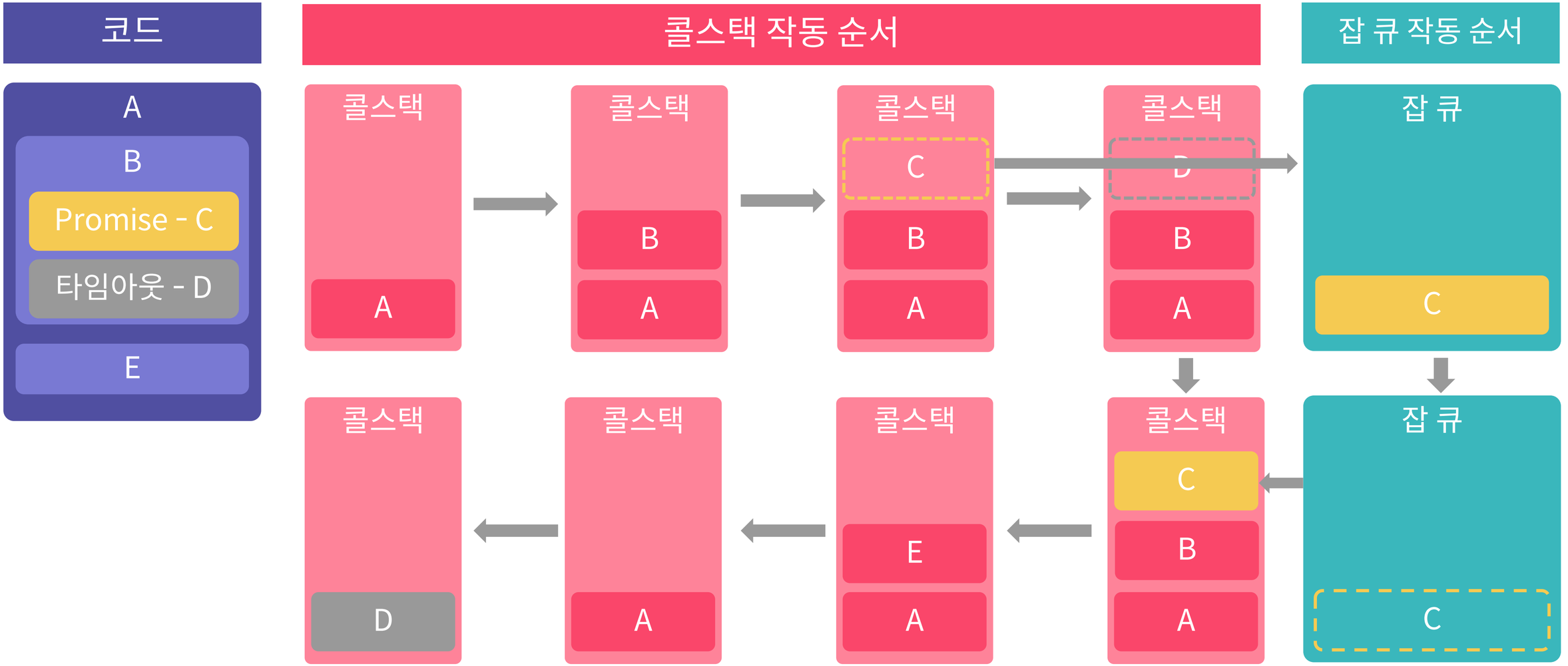
콜스택이 비어있을 때까지 스택의 함수를 실행

이벤트 루프 - 메시지큐의 작동 순서



콜스택이 비어있을 경우 등록된 함수를 콜스택에 추가

이벤트 루프 - 잡큐의 작동 순서



콜스택이 비어있을 경우 등록된 함수를 콜스택에 추가

이벤트 루프 정리

이벤트루프는 **비동기 동작의 실행 타이밍**을 이해하는 것이 중요

setTimeout 은 **콜스택이 비어있을 때** 실행 됨

Promise 는 **상위함수가 종료되기 전**에 실행 됨

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
최규범

강사
최규범

감수자
최규범

디자이너
김루미

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io