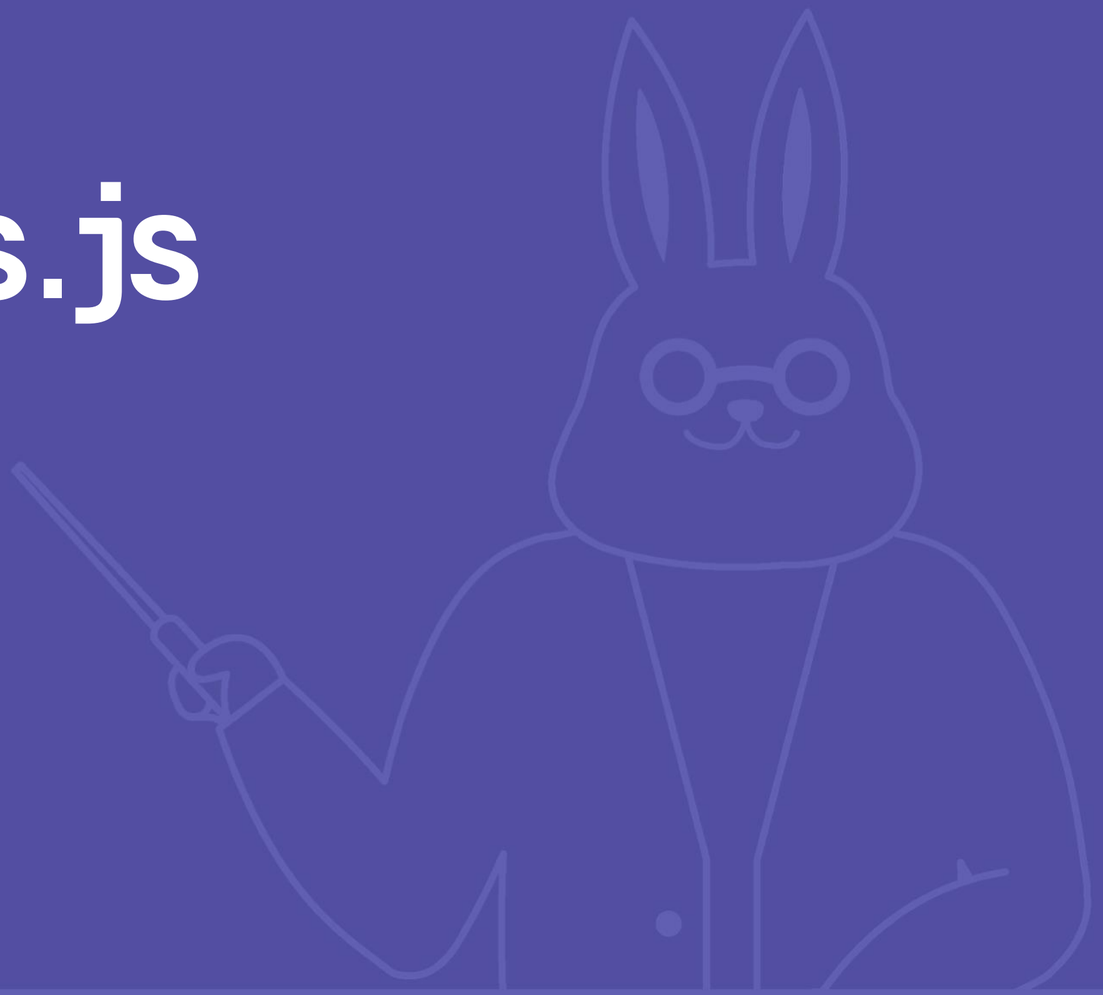
Node.js와 Express.js

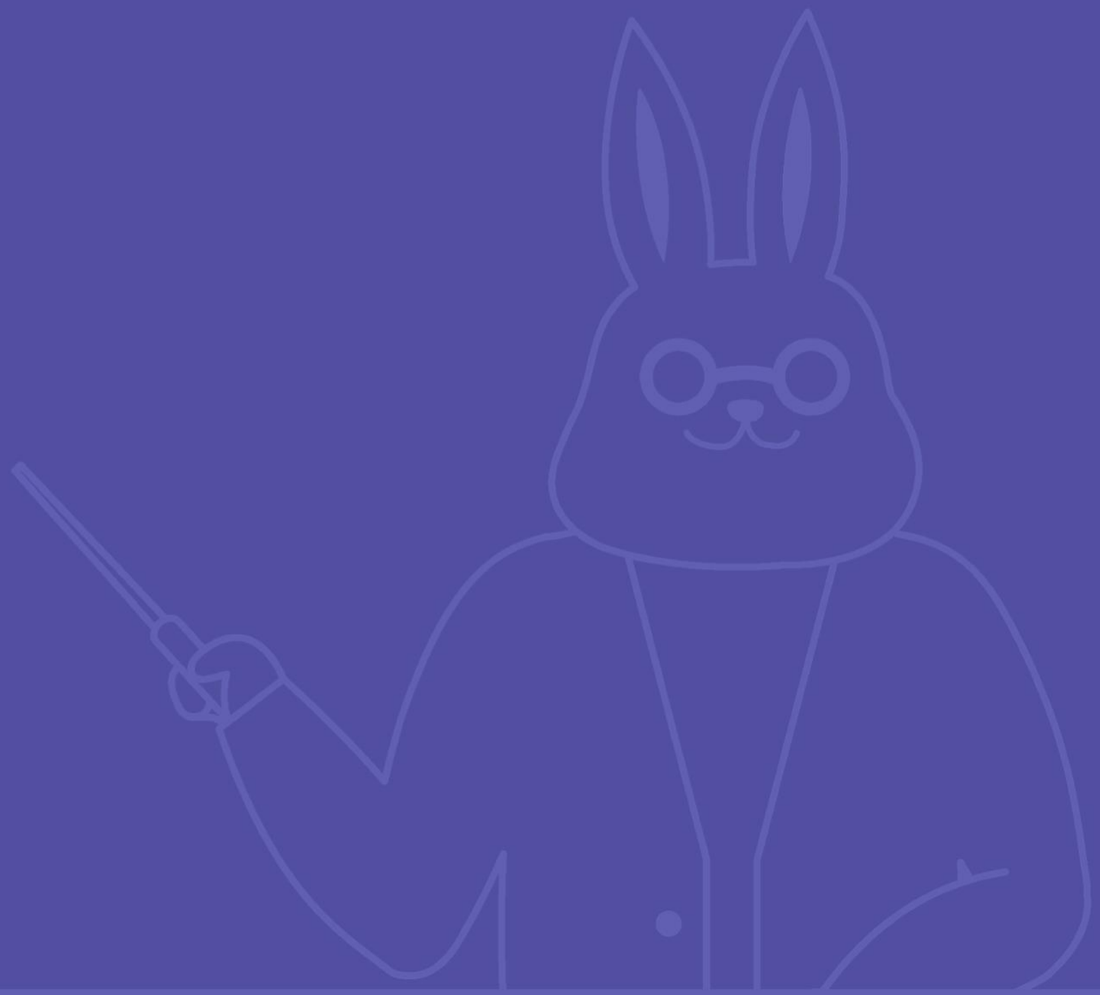
02 Node.js 이해하기



/* elice */

01

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 { name: n1, age: a1 } = obj;

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

자주 사용되는 문법 6 - promise, async - await

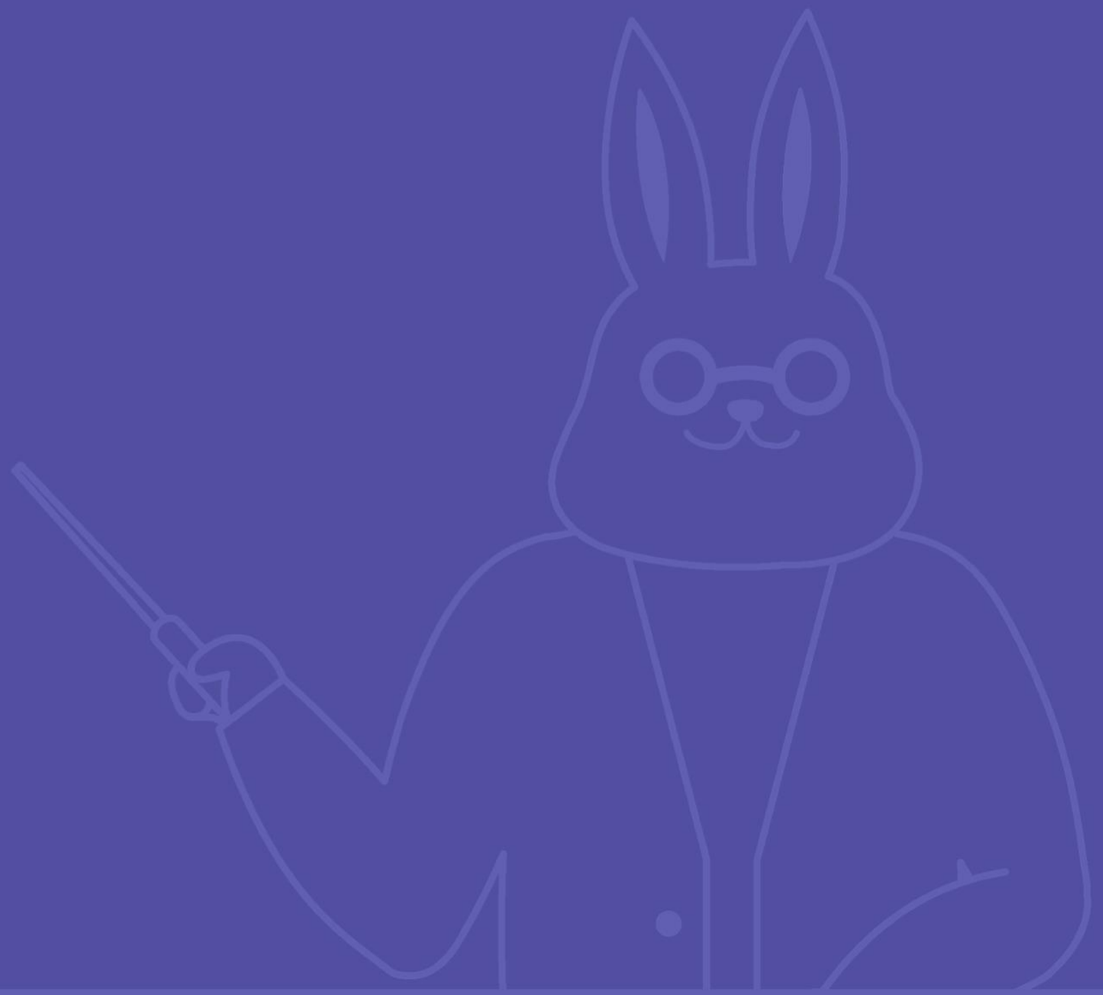
Promise와 Async - Await은 간단한 요약이 어려움
다음 장에서 비동기 코딩과 함께 학습합니다.

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.

02

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**

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io