



# Node.js와 Express.js

## 03 NPM과 모듈



01

# NPM 이해하기



## ✓ NPM이란?

Node **P**ackage **M**anager

Node.js 프로젝트를 관리하는 **필수적인 도구**

온라인 저장소

+

커맨드라인 도구

## ✓ NPM 온라인 저장소

수많은 **오픈소스 라이브러리**와 **도구**들이 업로드되는 저장소.  
필요한 라이브러리나 도구를 손쉽게 검색 가능.  
Node.js의 인기로, **거대한 생태계**를 보유.

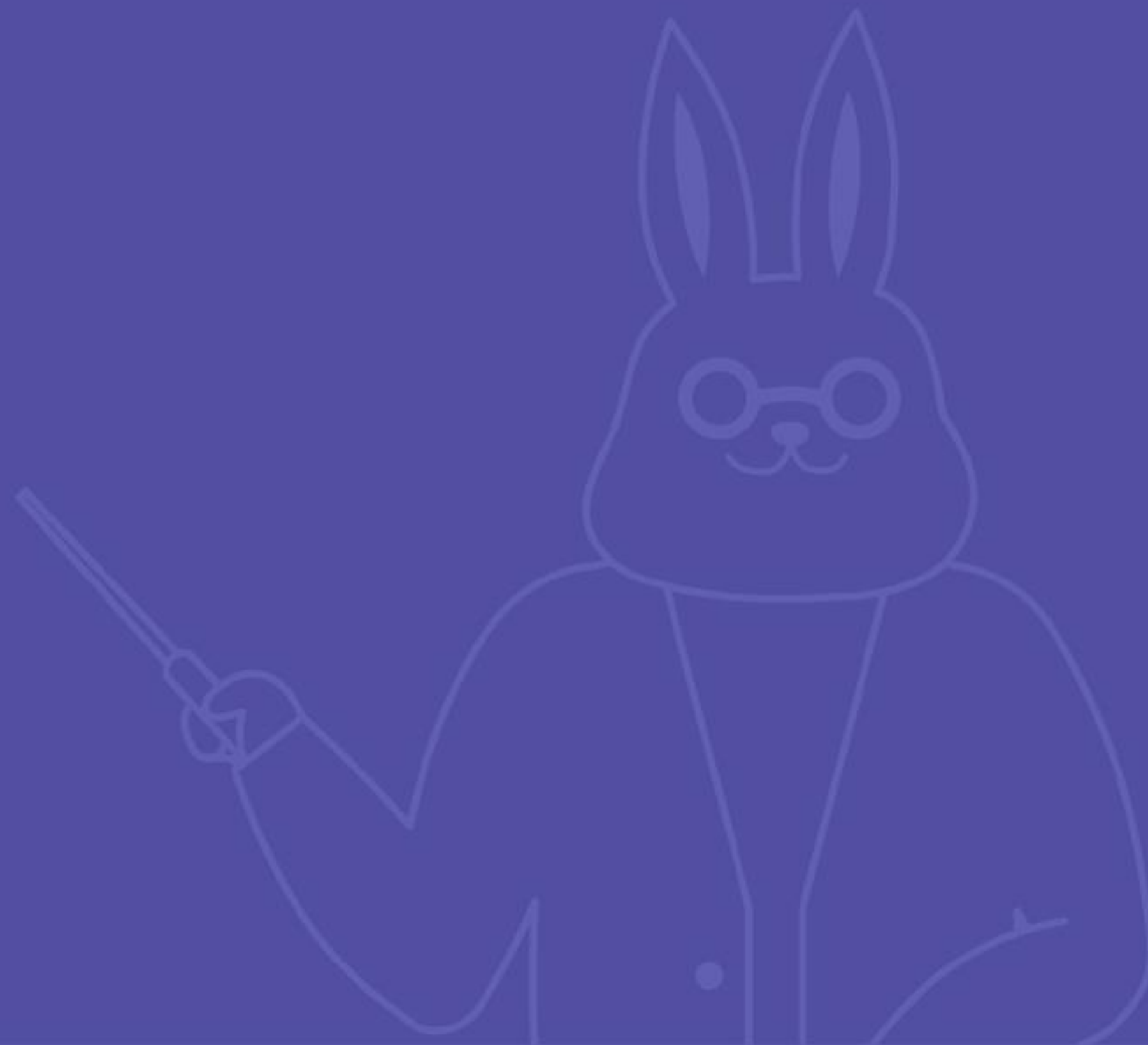
## ✓ 커맨드라인 도구

### 프로젝트 관리를 위한 다양한 명령어를 제공

- ✓ 저장소에서 라이브러리, 도구 설치
- ✓ 프로젝트 설정 / 관리
- ✓ 프로젝트 의존성 관리

02

# NPM 사용해 보기



## ✓ NPM을 사용한다는 것은?

NPM **커맨드라인 도구의 사용법**을 익히는 것  
프로젝트의 생성부터 다양한 기능을 사용하는 법까지 학습

## ✓ 프로젝트 생성하기

```
$npm init
```

프로젝트 디렉터리를 생성하고,  
해당 디렉터리 안에서 **npm init** 명령어를 사용하면  
몇 번의 질문을 통해 **package.json**이라는 파일을 만들어 주고  
이 디렉터리는 **Node.js 프로젝트가 됨**



## ✓ 프로젝트 생성하기

npm init

```
$ npm init

package name: (first-project)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to ~/package.json:
```

```
{
  "name": "first-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {},
  "author": "",
  "license": "ISC"
}
```

Is this OK? (yes)

## ✓ package.json

**프로젝트 관련 정보**들이 저장되는 파일  
이 파일을 **직접 수정**하거나 **npm 명령어**를 사용하여  
프로젝트 정보를 수정할 수 있음

✔ package.json

key	value
version	프로젝트의 버전
name	프로젝트의 이름
description	프로젝트 설명
scripts	npm run [script name]으로 실행할 수 있는 사용자 작성 스크립트
dependencies	의존성 패키지들
devDependencies	개발환경에서만 사용하는 의존성 패키지들
	.
	.
	.

package.json의 구성요소

## ✓ 의존성 관리하기

프로젝트 내에서 사용하는 라이브러리를 관리하는 방법  
프로젝트가 실행되기 위해 라이브러리에 의존하기 때문에  
이러한 라이브러리들을 **dependency(의존성)**라고 이야기함

## ✓ 라이브러리란?

**특정 기능을 수행하는 코드의 묶음**

복잡한 기능을 직접 작성하지 않고, **다른 사람이 구현한 것**을 사용하는 방법

Node.js에서는 **패키지**라고도 부름.

## ✓ npm install 명령어

npm install 명령어를 통해 **프로젝트 의존성을 관리** 할 수 있음  
npm install 명령어는 사용 방법에 따라 **여러 용도로 사용 가능**  
(npm i 를 축약형으로 사용 가능)

- ✓ 의존성 추가
- ✓ 의존성 내려받기
- ✓ 개발용 의존성 추가
- ✓ 전역 패키지 추가

## ✓ 프로젝트에 의존성 추가하기

```
$npm install [package-name]
```

필요한 패키지를 프로젝트에 추가할 수 있음.

추가된 패키지는 **package.json**의 **dependencies** 안에 추가되며,  
**node\_modules** 디렉터리에 저장됨.

## ✓ dependencies와 devDependencies

```
$npm install [package-name] --save-dev
```

npm은 **개발용 의존성을 분리**하여 관리할 수 있음

개발용 의존성이란 **배포 전까지만 사용하는 의존성** (ex. 유닛 테스트 라이브러리)

**--save-dev** 옵션을 이용하면 개발용 의존성을 추가할 수 있음

개발용 의존성은 package.json의 **devDependencies**에 추가됨



## ✓ package-lock.json

프로젝트에 의존성을 추가하면 package-lock.json이라는 파일이 생성됨  
프로젝트에 의존성을 추가하면 **자동으로 ‘^최신버전’으로 추가**가 되는데,  
의존성 버전이 갑자기 변경되지 않도록, 설치된 버전을 고정하는 역할을 함

## ✓ 의존성 버전 표기법

`npm install [package-name]@[version]` 으로 **패키지 버전을 지정**할 수 있음

Ex)

**~1.13.0** - 1.13.x 버전 설치

**^1.13.0** - 1.x.x 버전 설치, 가장 왼쪽의 0이 아닌 버전을 고정.

**0.13.0** - 0.13.0 버전만 설치

## ✓ 프로젝트에 의존성 내려받기

```
$npm install
```

기본적으로 node\_modules 디렉터리는 **코드 관리나 배포 시에 업로드 하지 않음**  
의존성이 많아지면 **용량이 너무 커지는 것과**,  
**운영체제별로 실행되는 코드가 다른 경우**가 존재하기 때문.

npm install 명령어를 아무 옵션 없이 사용하면  
package.json 에 정의된 **dependencies**와 **devDependencies**의 의존성을  
node\_modules 디렉터리에 내려받음.

## ✓ 개발용 의존성을 제외하고 내려받기

```
$npm install --production
```

프로젝트를 **배포할 때**에는 개발용 의존성을 같이 포함할 필요가 없음  
package.json 의 **dependencies**만 node\_modules에 내려받음

## ✓ 전역 패키지 추가

```
$npm install [package-name] --global
```

패키지를 **전역 패키지 디렉터리**에 내려받음

**주로 프로젝트에 종속되지 않는 커맨드라인 도구**들을 전역 패키지로 추가해서 사용

Ex) express-generator, pm2

## ✓ 로컬 패키지와 전역 패키지

### 로컬 패키지

package.json 에 선언되어 있고, node\_modules에 저장된 패키지

### 전역 패키지

npm install -g 를 통해 내려받아, 전역 패키지 저장소에 저장된 패키지

전역 패키지도 프로젝트에서 사용할 수 있으나,  
프로젝트의 의존성이 package.json 내에 **명시적으로 선언되어 있는 것이  
프로젝트 관리의 좋은 방향**

## ✓ 의존성 삭제하기

```
$npm remove [package-name]
```

의존성 패키지를 삭제할 수 있음

package.json의 **dependencies**와 **devDependencies**에서 삭제하고  
**node\_modules**에서도 삭제

## ✓ 스크립트 실행하기

스크립트란 간단한 동작을 수행하는 코드  
package.json의 scripts에 선언된 스크립트를  
npm run [script-name] 명령어로 실행할 수 있음



## ✓ 스크립트 실행하기

package.json

```
{
  ...
  "scripts": {
    "say-hi": "echo \"hi\""
  },
  ...
}
```

실행

```
$ npm run say-hi
"hi"
```

## ✓ npm script를 사용하는 이유

npm script 내에선 **의존성 패키지를 사용할 수 있음**

```
"scripts": {"test": "node_modules/.bin/tap test/*.js"}
```



```
"scripts": {"test": "tap test/*.js"}
```

## ✓ 자주 사용되는 스크립트

npm 스크립트엔 **run**을 제외하고 사용할 수 있는 주요 스크립트들이 있음

**test** - 코드 유닛 테스트 등에 사용

**start** - 프로젝트 실행

**stop** - 프로젝트 종료

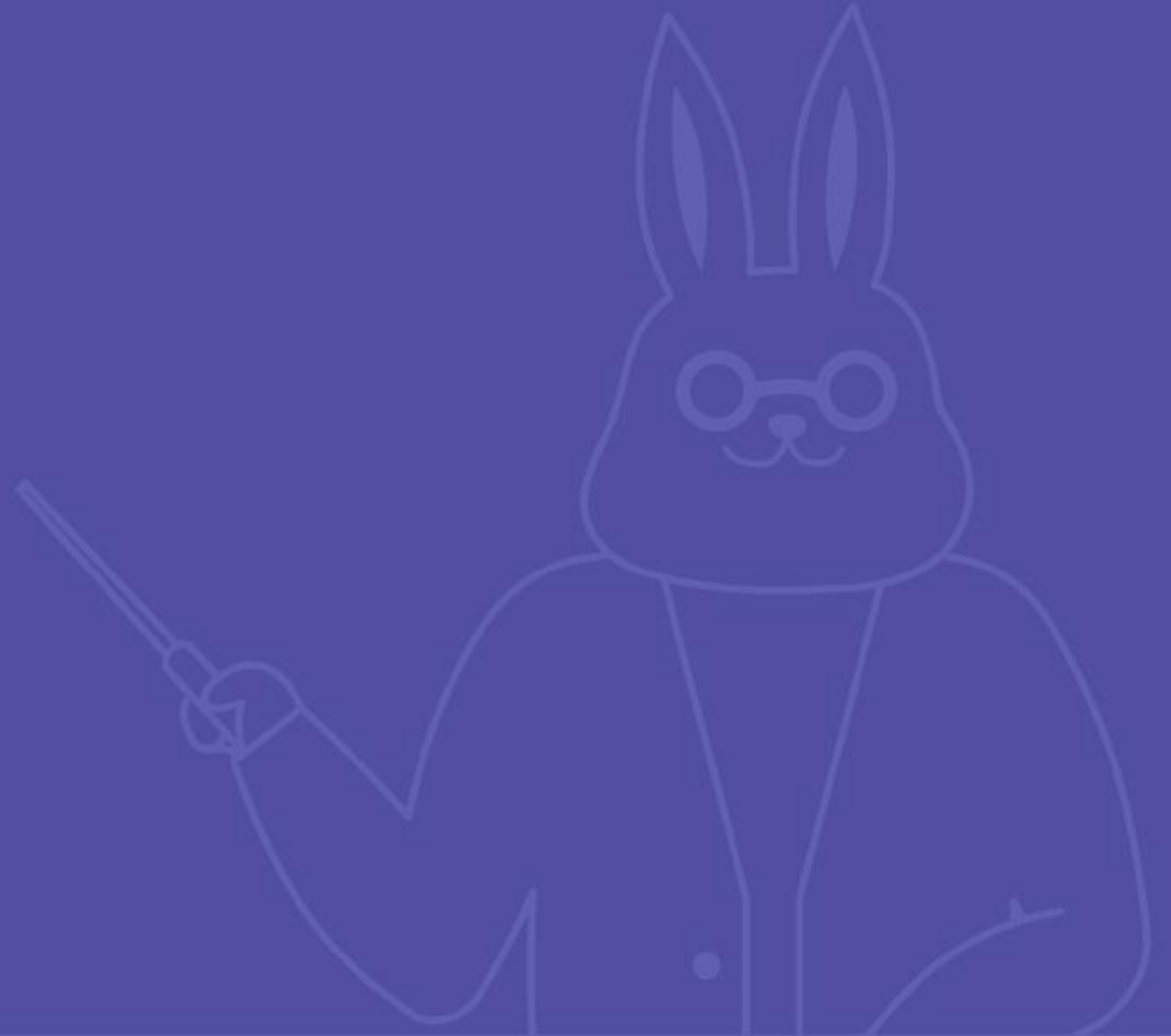
run을 제외하고 사용할 수 있을 뿐, npm 내부적으로 **코드를 제공해 주는 것은 아님**

✓ NPM 요약

명령어	npm init	프로젝트 생성
	npm install	의존성 추가
	npm remove	의존성 삭제
	npm run	스크립트 실행
주요 파일/디렉터리	node_modules	프로젝트 의존성 저장 디렉터리
	package.json	프로젝트 관리 (버전, 의존성, 스크립트, ...)
	package-lock.json	의존성 버전 확인

03

# NPX



## ✓ NPX 란?

npm

```
$ npm i cowsay -g  
$ cowsay hi
```

npx

```
$ npx cowsay hi
```

npm 패키지를 **설치하지 않고 사용**할 수 있게 해주는 도구  
프로젝트에 추가하거나 전역 패키지로 추가하지 않고,  
npx를 이용하여 바로 실행할 수 있음

## ✓ NPX - Node.js 특정 버전으로 실행

Node version

```
$ npx node@12 index.js  
$ npx node@14 index.js
```

npx 를 사용하면 Node.js의 특정 버전을 사용하여 js 파일을 실행할 수 있음  
프로젝트의 Node.js 버전 별 실행환경을 확인할 때 유용

## ✓ NPX - github gist 코드 실행

gist

```
$ npx https://gist.github.com/zkat/4bc19503fe9e9309e2bfaa2c58074d32
```

gist는 github에 등록된 간단한 코드

npx를 이용하면 gist 코드를 **다운받지 않고 바로 실행** 가능

**git이 설치되어 있어야 함**

온라인상의 코드는 어떤 위험이 있을지 모르므로 **코드를 잘 확인하고 실행해야 함**



04

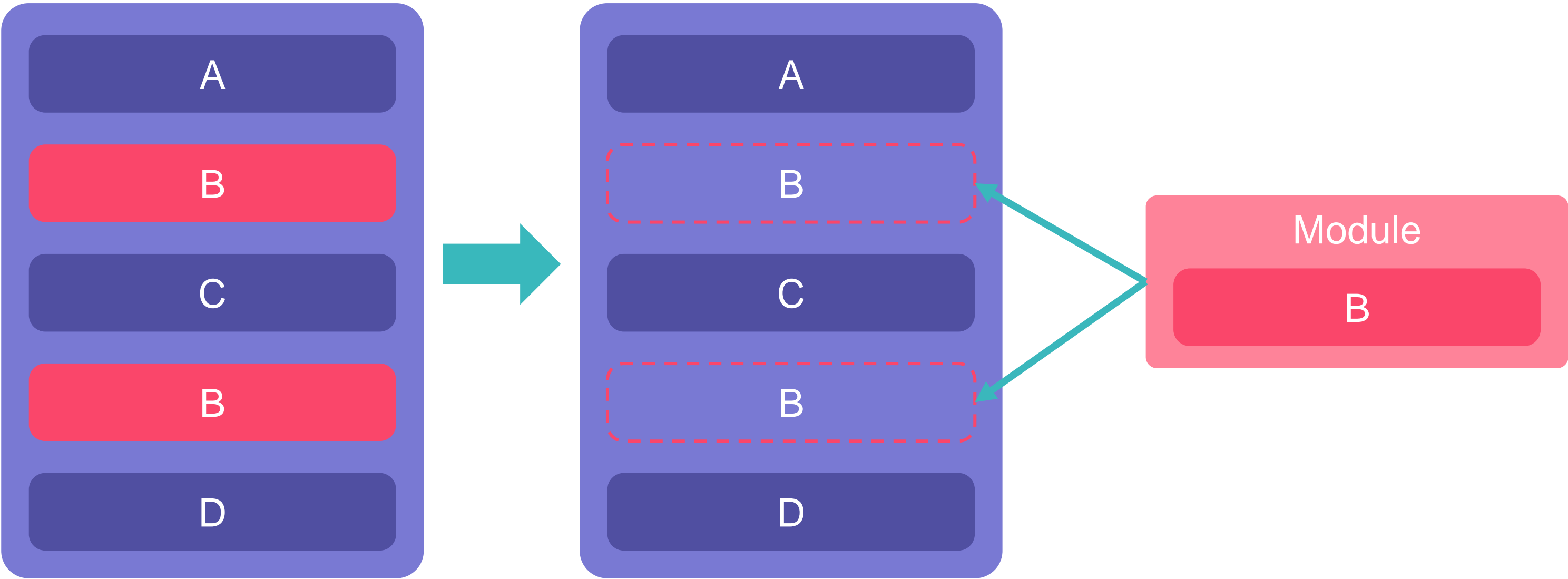
# Node.js의 모듈



## ✓ 모듈이란?

간단한 프로그램이라면 파일 하나로도 가능  
프로젝트가 커지면 **기능에 맞게 코드를 분리**하는 것이 중요  
모듈은 **코드를 분리하기 위한 방법**

✔ 모듈 사용 예



반복되는 B라는 코드를 모듈로 분리하여 사용

## ✓ 모듈과 패키지

패키지는 모듈의 모음

npm 패키지들은 많은 모듈을 포함하고 있는 코드 모음

## ✓ Node.js의 기본 제공 모듈

Node.js는 다양한 모듈을 기본적으로 제공함  
기본 제공 모듈은 직접 작성하기 매우 어렵거나  
복잡한 로직을 포함한 모듈이 있으므로  
자주 사용되는 기본 제공 모듈을 학습해야 함

## ✓ Node.js의 기본 제공 모듈 - console

브라우저에서 제공되는 console과 유사한 **디버깅 도구**

log, warn, error 함수로 **로그 레벨** 표시

time, timeLog, timeEnd 함수로 **시간 추적**

## ✓ Node.js의 기본 제공 모듈 - process

현재 실행프로세스 관련 기능 제공

arch, argv, env 등 **실행 환경 및 변수** 관련 값 제공

abort, kill, exit 등 **프로세스 동작** 관련 함수 제공

## ✓ Node.js의 기본 제공 모듈 - fs

**파일 입출력**을 하기 위해 사용

readFile, writeFile 함수로 **파일 읽기, 쓰기**

-Sync 함수 제공. **동기 동작**

watch로 파일/디렉터리 **변경 이벤트 감지**



## ✓ Node.js의 기본 제공 모듈 - http

**http** 서버, 클라이언트를 위해 사용

createServer 함수로 **서버 생성**

Request 함수로 **http 요청 생성**

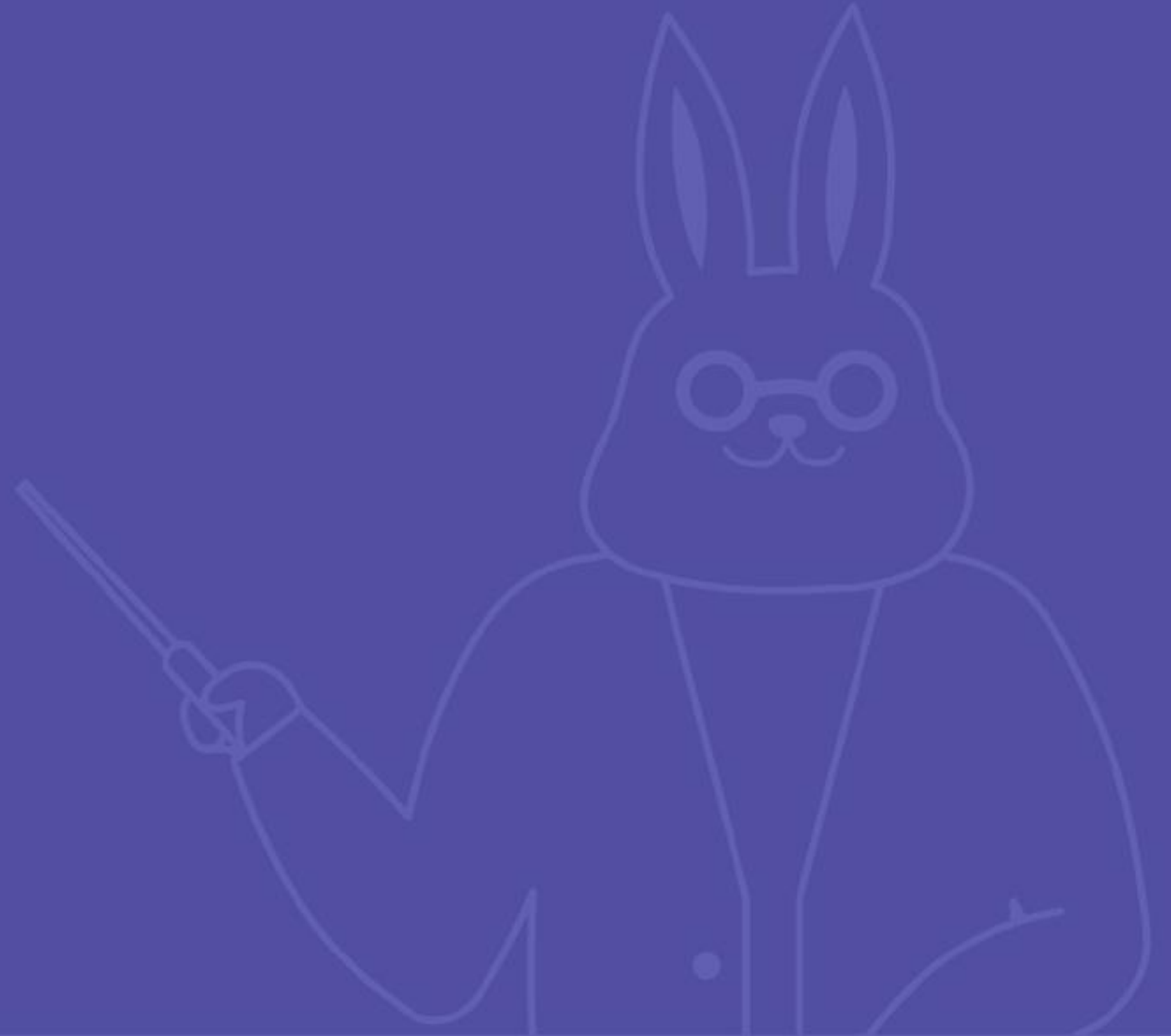
✔ 기타 기본제공 모듈

url	url 파싱
os	운영체제 정보 - cpu, memory, type ...
Path	디렉터리 string 관련 작업 - 서로 다른 운영체제 간 공통된 로직
crypto	암호화, hash 관련 함수 제공
	.
	.
	.

<https://nodejs.org/dist/latest-v14.x/docs/api/>

05

# 모듈의 작성과 사용



## ✓ 모듈의 기본적인 작성법

module

```
// elice.js
const name = 'elice';
const age = 5;
const nationality = 'korea';

module.exports = {
  name,
  age,
  nationality ,
};

---
const student = require('./elice');
// student 출력값 { name: 'elice', age: 5, nationality: 'korea'}
```

## ✓ 모듈의 기본적인 작성법

module

```
module.exports = {  
  name,  
  age,  
  nationality,  
};
```

모듈이 load될 때 사용될 값을 module.exports로 내보냄.

## ✓ 변수명으로 export 하는 모듈 작성법

### named-module

```
// elice.js
const name = 'elice';
const age = 5;
const nationality = 'korea';

exports.name = name;
exports.age = age;
exports.nationality = nationality;

---

const student = require('./elice');
// student 출력값 { name: 'elice', age: 5, nationality: 'korea' }
```

## ✓ 변수명으로 export 하는 모듈 작성법

named-module

```
exports.name = name;  
exports.age = age;  
exports.nationality = nationality;
```

모듈을 object로 만들고, 각 key - value를 지정해서 내보냄

## ✓ 함수를 export하는 모듈 작성법

function-module

```
// elice.js
module.exports = (name, age, nationality) => {
  return {
    name,
    age,
    nationality,
  };
}

---

const student = require('./elice')('elice', 5, 'korea') ;
// student 출력값 { name: 'elice', age: 5, nationality: 'korea' }
```



## ✓ 함수를 export하는 모듈 작성법

function-module

```
module.exports = (name, age, nationality) => {  
  return {  
    name,  
    age,  
    nationality ,  
  };  
}
```

모듈을 함수로 만들어서 모듈 사용 시에 값을 정할 수 있게 내보냄

## ✓ 모듈의 사용 방법

**require** 함수를 통해 모듈을 load 할 수 있음

C에서 **include**, Java에서 **import**와 유사

의존성 패키지, 직접 작성한 모듈 사용 가능

## ✓ 모듈의 사용 방법 - require 동작의 이해

require 할 때 **모듈 코드가 실행됨**

Node.js의 모듈은 **첫 require 시에 cache**, 두 번 실행하지 않음

모듈 코드를 **여러 번 실행하기 위해선 함수 모듈로 작성**

## ✓ 모듈의 사용 방법 - npm 패키지

require-npm

```
const dayjs = require('dayjs');  
console.log(dayjs());
```

의존성 패키지들은 **require('package-name')**로 load 할 수 있음  
패키지를 사용하려면 **node\_modules**에 내려받아야져 있어야 함

## ✓ 모듈의 사용 방법 - 직접 작성한 모듈

require-my-module

```
const myModule = require('./my-module');  
console.log(myModule);
```

직접 작성한 모듈은 **현재 파일과의 상대 디렉터리**로 load

my-module이 .js 파일인 경우 해당 파일 load

my-module이 디렉터리인 경우 my-module/**index.js** 파일 load

## ✓ 모듈의 사용 방법 - 함수형 모듈

require-function-module

```
const myFunctionModule = require('./my-function-module');  
console.log(myFunctionModule(name, age, nationality));
```

함수형 모듈은 load한 경우 **모듈이 바로 실행되지 않음**  
필요한 시점에 load된 함수를 실행하여 모듈을 사용할 수 있음

## ✓ 모듈의 사용 방법 - json 파일

require-my-data

// my-data.json 을 가지고 있음

```
const myData = require('./my-data');  
console.log(myData);
```

require로 **json 파일**도 load 가능  
**object**로 자동파싱

## ✔ 모듈의 작성과 사용 요약

**module.exports**를 사용하여 모듈을 작성할 수 있음

**require**를 사용하여 의존성 패키지, 모듈, json 파일을 사용할 수 있음

모듈은 첫 **require** 시에만 실행하고 **cache** 되므로

**여러 번 실행할 모듈은 함수형**으로 작성해야 함



# 연락처

TEL

070-4633-2015

WEB

<https://elice.io>

E-MAIL

[contact@elice.io](mailto:contact@elice.io)

