Node.js

■ Node.js

- Google Chrome의 **V8 JavaScript 엔진으로 빌드된 JavaScript 런타임 환경**

- 웹 브라우저 외부에서 JavaScript 코드를 실행할 수 있게 해주는 오픈 소스 플랫폼**(웹 브라우저 없이도 JavaScript 코드를 실행할 수 있는 환경을 제공)**

- 서버 개발에 주로 사용되어 JavaScript의 활용 범위를 웹 프론트엔드를 넘어 백엔드까지 확장했으며, 낮은 학습 장벽, 방대한 개발자 커뮤니티, 높은 확장성 등의 장점으로 풀스택 개발 및 다양한 네트워크 애플리케이션 개발에 널리 활용됨

- **node -v** : 노드 버전 확인, 노드가 잘 설치되었는지 확인

■ "Hello, World!"를 출력하는 코드와 간단한 웹 서버를 만드는 코드

– (1) 기본 콘솔 출력 예제 (Hello, World!)

* Node.js는 자바스크립트를 브라우저 밖에서 실행할 수 있게 해줌
* 가장 기본적인 예제는 콘솔에 텍스트를 출력하는 것임

```
// hello.js 파일로 저장
// Node.js는 브라우저 자바스크립트와 마찬가지로 console.log()를 사용하여 출력
        console.log("Hello, Node.js World!");

        // 간단한 변수 선언 및 계산도 물론 가능함
        const a = 5;
        const b = 10;
        console.log(`두 수의 합: ${a + b}`);
```

* 터미널(명령 프롬프트)에서 해당 파일이 있는 디렉토리로 이동한 후 다음 명령어를 실행
  - **node hello.js**

– (2) 간단한 HTTP 웹 서버 예제

* Node.js의 강력한 기능 중 하나는 **내장된 http 모듈을 사용하여 웹 서버를 쉽게 만들 수 있다는 것임**

```
// server.js 파일로 저장
        // 1. HTTP 모듈을 가져옴
        const http = require('http');

        // 서버가 접속을 대기할 포트 번호와 호스트 이름을 정의함
        const hostname = '127.0.0.1'; // localhost
        const port = 3000;

        // 2. 서버를 생성함
        const server = http.createServer((req, res) => {
         // 요청(req)에 응답(res)하는 로직임

            // 응답 헤더 설정: 성공(200) 상태 코드와 응답 타입(일반 텍스트)
            res.statusCode = 200;
            res.setHeader('Content-Type', 'text/plain; charset=utf-8');

            // 사용자에게 보낼 실제 응답 본문
            res.end('Node.js로 만든 아주 간단한 서버입니다!\n');
           });

        // 3. 서버를 지정된 포트와 호스트 이름에서 대기(Listen)하도록 함
        server.listen(port, hostname, () => {
        console.log(`서버가 http://${hostname}:${port}/ 주소에서 실행 중입니다.`);
        console.log('브라우저에서 이 주소로 접속해 보세요.');
            });
```

* 터미널에서 다음 명령어를 실행
  - **node server.js**
* 동작 확인
  - **터미널에 "서버가 http://127.0.0.1:3000/ 주소에서 실행 중입니다." 메시지가 출력됨**
  - 웹 브라우저를 열고 **주소창에 http://127.0.0.1:3000/을 입력하고 접속함**
  - 브라우저 화면에 "Node.js로 만든 아주 간단한 서버입니다!"라는 텍스트가 보임
  - 서버를 종료하려면 터미널에서 Ctrl + C를 누름

♣ ES 모듈(ESM)

- ESM 모듈이란?
  * ECMA Script Modules의 약자로, **자바스크립트의 공식적인 표준 모듈 시스템**을 의미
  * 이는 **코드를 재사용 가능한 단위(모듈)로 분리하고, 필요한 곳에서 불러와(import) 사용**할 수 있도록 ECMAScript 2015(ES6)에서 도입되었음

- 최신 Node.js 버전에서는 ES 모듈(ESM)을 사용하는 것이 표준화되었음
  * 이전의 require/module.exports (CommonJS) 방식 대신 import/export 문법을 사용함

♣ 최신 Node.js 환경에서 JavaScript 파일을 실행하는 가장 기본적인 두 가지 예제

– ES 모듈(ESM) 방식으로 실행 (권장)

* ES 모듈 방식을 사용하면 브라우저에서 사용하던 import/export 문법을 Node.js 환경에서 그대로 사용할 수 있음

* 1단계) 프로젝트 초기화 - 프로젝트 폴더를 만들고 초기화

```
mkdir node-esm-project
cd node-esm-project
npm init -y
```

```
C:₩CODE₩node-esm-project>npm init -y
Wrote to C:₩CODE₩node-esm-project₩package.json:

{
  "name": "esm-project",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo ₩"Error: no test specified₩" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

* 2단계) package.json 설정 (필수)

- package.json 파일에 "type": "module"을 추가하여 Node.js에게 이 프로젝트의 .js 파일들을 ES 모듈로 처리하도록 지시함

- package.json 예시

```
{
  "name": "node-esm-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",   <-- 이 부분을 추가
  "scripts": {
    "start": "node app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

* 3단계) 모듈 파일 생성

utils.js (모듈 내보내기):

```
// utils.js

// 1. 이름 있는 내보내기 (Named Export)
export const greet = (name) => {
    return `안녕하세요, ${name}님! (ESM)`;
};

// 2. 비동기 함수 (Top-level await 가능)
export async function fetchData() {
    // 최신 Node.js는 fetch API를 내장하고 있음
    const response = await
fetch('https://jsonplaceholder.typicode.com/todos/1');
    const data = await response.json();
    return data.title;
}
```

* 4단계) 실행 파일 생성 - app.js (모듈 가져오기 및 실행)

```
// app.js
// 로컬 모듈을 가져올 때는 항상 확장자(.js, .mjs 등)를 명시해야 함
import { greet, fetchData } from './utils.js';

// Top-level await 사용 (최신 Node.js 기능)
console.log('--- 1. 일반 함수 실행 ---');
const message = greet('개발자');
console.log(message);

console.log('\n--- 2. 비동기 함수 실행 (Top-level await) ---');
try {
    const todoTitle = await fetchData();
    console.log(`비동기 데이터: ${todoTitle}`);
} catch (error) {
    console.error('데이터를 가져오는 중 오류 발생:', error);
}

// Node.js 내장 모듈 가져오기 (import 'node:' 접두사 권장)
import { readFile } from 'node:fs/promises';
import { fileURLToPath } from 'node:url';
import { dirname } from 'node:path';

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

console.log('\n--- 3. Node.js 내장 모듈 사용 (ESM) ---');
console.log(`현재 파일 경로: ${__filename}`);
// ... 파일 읽기 등의 비동기 작업을 await로 바로 처리 가능
```

* 5단계) 실행

```
node app.js
```

* 결과

```
--- 1. 일반 함수 실행 ---
안녕하세요, 개발자님! (ESM)

--- 2. 비동기 함수 실행 (Top-level await) ---
비동기 데이터: delectus aut autem

--- 3. Node.js 내장 모듈 사용 (ESM) ---
현재 파일 경로: /path/to/node-esm-project/app.js
```

```
C:₩CODE₩node-esm-project>node app.js
(node:10040) [MODULE_TYPELESS_PACKAGE_JSON] Warning: Module type of file:///C:/CODE/node-esm-project/app.js is not speci
fied and it doesn't parse as CommonJS.
Reparsing as ES module because module syntax was detected. This incurs a performance overhead.
To eliminate this warning, add "type": "module" to C:₩CODE₩node-esm-project₩package.json.
(Use `node --trace-warnings ...` to show where the warning was created)
--- 1. 일반 함수 실행 ---
안녕하세요, 개발자님! (ESM)

--- 2. 비동기 함수 실행 (Top-level await) ---
비동기 데이터: delectus aut autem

--- 3. Node.js 내장 모듈 사용 (ESM) ---
현재 파일 경로: C:₩CODE₩node-esm-project₩app.js
```

– .mjs 파일 확장자를 이용한 실행

* package.json을 수정하지 않고, 파일 확장자를 .mjs로 지정하면 해당 파일은 무조건 ES 모듈로 처리됨

* 1단계) 파일 생성

calculator.mjs:

```
// calculator.mjs
export function sum(a, b) {
    return a + b;
}

export function multiply(a, b) {
    return a * b;
}
```

main.mjs:

```
// main.mjs
import { sum, multiply } from './calculator.mjs'; // .mjs 확장자 명시

console.log(`10 + 5 = ${sum(10, 5)}`);
console.log(`10 * 5 = ${multiply(10, 5)}`);
```

* 2단계) 실행

```
node main.mjs
```

* 결과

```
10 + 5 = 15
10 * 5 = 50
```