ECMA 스크립트

- ECMA
  - European Computer Manufactures Association International : 유럽 컴퓨터 시스템 표준화 기구
- ECMA-262
  - 자바스크립트의 표준
  - 브라우저간의 호환성 문제 해소. 각 브라우저 개발사들이 ECMAScript 표준을 따라 브라우저를 구현
- 버전
  - ES5 = 2009년
  - ES6 = ES2015
    - let,const/Arrow function/for~of/default parameter/spread operator(…)/template literal/ Destructuing Assignment/promise/Map/Set/Module/Symbol/class
  - ES8 = ES2017
    - async,await/String padding
- 바벨(Bebel)
  - 구 브라우저에서도 최신 자바스크립트 코드를 작동하도록 변환해주는 트랜스파일러
- 폴리필(Polyfill)
  - 기능을 지원하지 않는 웹 브라우저에 최신 표준의 자바스크립트 기능을 구현해주는 호환성 구현 라이브러리 코드

서버프로그램을 위한 자바스크립트

- 변수선언자
- Arrow Function
- Array 내장 함수
- Template Literals
- Spread 연산자
- Object Destructing
- Array Destructing
- Default Function Paramter
- Rest Parameter
- Promise
- Async/Await
- class
- Regular Expression

https://nodejs.dev/en/learn/how-much-javascript-do-you-need-to-know-to-use-nodejs
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements

https://developer.mozilla.org/ko/docs/Learn/JavaScript/First_steps/Arrays
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array
https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Indexed_collections

- 객체리터럴
  - 함수연결할 때 콜론과 function 생략 가능
  - 속성명과 변수명이 겹치는 경우 생략

```
let sayNode = function () {
  console.log("node");
};
```

```
let oldObject = {
  sayJS: function () {
    console.log("js old");
  },
  sayNode: sayNode,
};
```

```
let newObject = {
  sayJS() {
    console.log("js new");
  },
  sayNode,
};
```

- 모듈
  - 특정기능을 하는 하나의 코드 묶음 단위
- 캡슐화
  - 모듈 안의 모든 기능들은 모듈 안에서만 동작하며, 모듈 밖에서는 접근이 허용된 속성이나 메서드만 사용가능
- 모듈시스템 규칙
  - 모듈은 파일 단위로 구성
  - 모듈의 변수, 함수 클래스 등은 export 키워드로 노출하고 import로 가져다 사용한다.
  - 모듈 이름은 중복 안됨
  - 모듈은 순환 참조를 할 수 없음
  - 모듈도 하나의 객체로서 임포트 시점에 모듈객체의 참조 주소를 변수에 할당
  - export 키워드로 내보낼 수 있는 모듈은 var. let, const, function, class 임

- app.html

```
 <script type="module" src="./main.js"></script>
```

SyntaxError: Cannot use import statement outside a module

- main.js

```
import { module } from "./module.js";
module("module run");
```

- module.js

```
export function module(msg) {
  console.log("msg:" + msg);
}
```

내장 모듈

- process
- url
- fs
- console

Process 객체

- process 객체
  - 현재 실행되고 있는 Node.js 프로세스에 대한 정보와 제어를 제공
  - 전역 객체 사용

```
let args = process.argv
console.log(args)
```

  - import나 require로 명시적으로 선언

```
import { argv } from 'node:process';

// print process.argv
argv.forEach((val, index) => {
  console.log(`${index}: ${val}`);
});
```

- process events
  - beforeExit, exit 등 이벤트가 발생할 때마다 리스너 등록

```
import process from "process";
process.on("beforeExit", (code) => {
  console.log("2. 이벤트 루프에 등록된 작업이 모두 종료된 후 노드 프로세스를 종료하기전", code  );
});
process.on("exit", (code) => {
  console.log("3. 노드 프로세스기 종료될 때", code);
});

console.log("1. 콘솔에 출력되는 첫번째 메시지");
```

- process evn
  - 사용자 환경을 포함하는 객체를 반환

```
process.env
```

URL 모듈

- URL
  - 인터넷 주소에 해당하는 url을 다루기 위한 모듈
  - url은 구조화된 문자열

```
protocol      auth            host                  path              hash
         username password                       pathname  search
```

```
http://  user : pass       @sub.example.com    /path/a/b ?querystring    #hash
```

- fs
  - 파일 읽기, 쓰기, 삭제 등과 같은 파일 처리와 관련된 작업을 위한 모듈로서 비동기, 동기 둘 다 제공
- fs.readFie(path, [options], callback)
  - path에 지정된 파일을 옵션으로 지정한 문자 인코딩(utf8)을 사용하여 읽은 후 결과를 callback() 함수로 전달하는 비동기 방식 함수

```
import fs from 'fs';
fs.readFile('./sample/text.txt', 'utf8', (err, data) => {
  if(err) {   throw err;  }
  console.log(data);
})
```

- fs.readFileSync(path, [options])
  - path에 지정된 파일을 읽은 후 결과를 반환하는 동기 방식 함수

```
const fs = require('fs')
var text = fs.readFileSync('./sample/index.html', 'utf8')
console.log(text);
```

외부 모듈(라이브러리)

- json-server
  - JSON 기반으로 가상의 REST API 서버 구축

```
D:\vs_work\node_week1\json-server>json-server  --watch  db.json

  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:3000/posts
  http://localhost:3000/comments
  http://localhost:3000/prifle
```