- ECMA 스크립트
- 서버프로그램을 위한 자바스크립트
- Node.js 내장 모듈과 객체

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/Literals

https://nodejs.org/en/learn/getting-started/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

Array 내장함수

- sort()

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();

const array1 = [1, 30, 4, 21, 100000];
array1.sort();
array1.sort(function(a,b){return a-b});
```

- filter()
  - 특정 조건을 만족하는 배열의 요소만을 찾아서 새로운 배열로 반환

```
const words = ['spray', 'elite', 'exuberant',
               'destruction', 'present'];
const result = words.filter((word) =>
word.length > 6);
```

- map()
  - 배열 내의 모든 요소 각각에 대하여 주어진 함수를 호출한 결과를 모아 새로운 배열을 반환

```
const array1 = [1, 4, 9, 16];
const map1 = array1.map((x) => x * 2);
```

- reduce()
  - 배열의 각 요소에 대해 주어진 리듀서 (reducer) 함수를 실행하고, 하나의 결과값을 반환

```
const array1 = [1, 2, 3, 4];// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator +
currentValue,   initialValue,
);
```

- splice()
  - 배열의 기존 요소를 삭제 또는 교체하거나 새 요소를 추가하여 배열의 내용을 변경

```
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
```

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array

리터럴

- 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,
};
```

- 템플릿 리터럴(문자열)
  - 변수에 할당된 문자열을 하나의 문자열로 병합할 때 ,+ 를 사용하지 않고 하나의 문자열로 표현
  - 문자열을 템플릿 리터럴로 표현하려면 역따옴표(``)로 묶음

```
let name = "hong";
let letter = `Dear ${hong}
Lorem   ${hong}`;
```

■ lorem ipsum

- 로렘 입숨((내용보다 디자인 요소를 강조하기 위해 사용되는 텍스트))
- 공간 채움을 위한 의미 없는 글.

사용방법

1. scode에서 마켓플레이스를 연후 'Lorem Ipsum'을 검색
2. F1(명령 팔레트)를 실행 후 'Lorem Insum'을 검색하고 선택항목 3가지 중 필요한 항목을 선택

* HTML 페이지에서는 Lorem*20

- spread 연산자

```
let arr1 = ['March', 'Jan'];
let arr2 = ['Feb', 'Dec'];

let arr3 = [... arr1, ...arr2];
```

- rest 파라미터

```
const add = (first, ...nums) => {  }
add(5,4,3,2,1);
```

- 구조 분해 할당
  - Object destructuring

```
const obj = {a: 1, b: 2, c: 3}
const {c, b, a} = obj // 3 2 1
```

  - Array Destructuring

```
//모든 요소 분해
const number = [ 1, 2, 3]
const [first, second, third ] = numbers
```

```
//필요 요소 분해
const chars = ['a', 'b', 'c', 'd', 'e', 'f']
const [a, b, c, ...rest] = chars
```

- 비동기 처리
  - Node.js에서 Promise는 파일 쓰기, 데이터베이스 트랜잭션 처리등 비동기 함수를 실행할 때 사용
  - 코드가 완료될 때까지 대기하지 않고 바로 다음 코드를 실행 할 수 있도록 해주며 비동기 함수 실행이 완료되면 then()함수에서 결과 코드를 실행

```
function delay() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {  resolve("hello");  }, 4000);
  });
}

delay().then((res) => {
  console.log("then", res);
});

console.log("then after");
```

- async – 비동기로 실행
- await – 결과가 올 때까지 대기

```
function delay() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {   resolve("hello");   }, 4000);
  });
}

async function f() {
  let result = await delay(); // 프라미스가 이행될 때까지 기다림
  console.log("await", result);
}

f();
console.log("await after");
```

모듈

- 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
  - 모듈 이름은 중복 안됨
  - 모듈은 순환 참조를 할 수 없음
  - 모듈도 하나의 객체로서 임포트 시점에 모듈객체의 참조 주소를 변수에 할당
  - ES5에서는 모듈 개념이 없어서 즉시실행함수를 사용했음
  - 변수, 함수, 클래스(var. let, const, function, class) 등은 export 키워드로 노출하고 import로 가져다 사용한다.

- 모듈 장점
  - 유지보수 용이 - 기능들이 모듈화가 잘 되어 있는 경우, 의존성을 줄일 수 있기 때문에 기능을 개선하거나 수정이 용이
  - 네임스페이스화 - 코드의 양이 많아질수록 전역 스코프에 존재하는 변수명이 겹치는 경우가 존재. 모듈만의 네임스페이스글 지정하여 문제를 해결
  - 재사용성 - 같은 코드를 반복하지 않고 모듈로 분리시켜서 필요할 때마다 재활용
- 모듈 시스템의 종류
  - AMD - 가장 오래된 시스템 중 하나로 require.js라는 라이브러리를 통해 처음 개발되었습니다.
  - CommonJS - NodeJs 환경을 위해 만들어진 모듈 시스템 입니다.
  - UMD - AMD와 NodeJs와 같은 다양한 모듈 시스템을 함께 사용하기 위해 만들어졌습니다.
  - ES Module - ES6(Es2015)에 도입된 자바스크립트 모듈 시스템입니다.

`CommonJS`

- require가 동기로 이루어지므로 promise를 return 하지 않는다.
- CommonJs는 실행을 해보아야 import, export 에러를 감지할 수 있다.
- 디폴트 값으로 적용된다.
- top-level await가 불가능하다.

`ES Module`

- 모듈을 비동기 환경에서 다운로드하며, import export 구문을 찾아 파싱한다.
- EsModules는 실행해보지 않아도 import,export 에러를 감지할 수 있다.
- config를 type='module'로 세팅 해주어야 사용할 수 있다.
- top-level await가 가능하다.

CommonJS와 비교하여 ESModules는 비동기로 동작하여 속도가 빠르고
실제 사용되는 부분만을 import(tree shaking)하여 메모리를 적게 차지하며, 가독성이 좋고
순환의존성을 지원한다는 이점이 있습니다.

- ES Module
  - 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 객체

- 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
```