KB

It's Your Life

with

구조 분해 할당

(비구조화 할당)

구조 분해 할당

- 객체 또는 배열의 각각의 값을 분해하여 변수에 넣어 사용하는 방법

```
const user = {
  id: 1,
  name: "tetz",
  email: "xenosign@naver.com",
};

const { id, name, email } = user;

console.log(id);
console.log(name);
console.log(email);

const fruits = ["사과", "딸기", "망고", "수박"];
const [a, b, c, d] = fruits;
console.log(a, b, c, d);
```

전개 연산자

전개 연산자

- 배열의 값을 , 단위로 구분 하여 전개시켜주는 연산자

```
const fruits = ["사과", "바나나", "수박"];
console.log(fruits);
console.log(...fruits);
// console.log("사과", "바나나", "수박");

function conLog(a, b, c) {
  console.log(a, b, c);
}

conLog(fruits[0], fruits[1], fruits[2]);
conLog(...fruits);
```

나머지 연산자(매개 변수일 때)

- 매개 변수에 너무 많은 값이 들어올 때 나머지 연산자를 사용하여 한꺼번에 처리 가능

```
const fruits = ["사과", "바나나", "수박", "망고",
"딸기"];

function conLog(a, b, ...c) {
  console.log(a, b, c);
}

conLog(...fruits);
```

```
const fruits = ["사과", "바나나", "수박", "망고", "딸기"];

function conLog(...rest) {
  rest.forEach((element) => {
    console.log(element);
  });
}

conLog(...fruits);
```

문자열을 배열로!

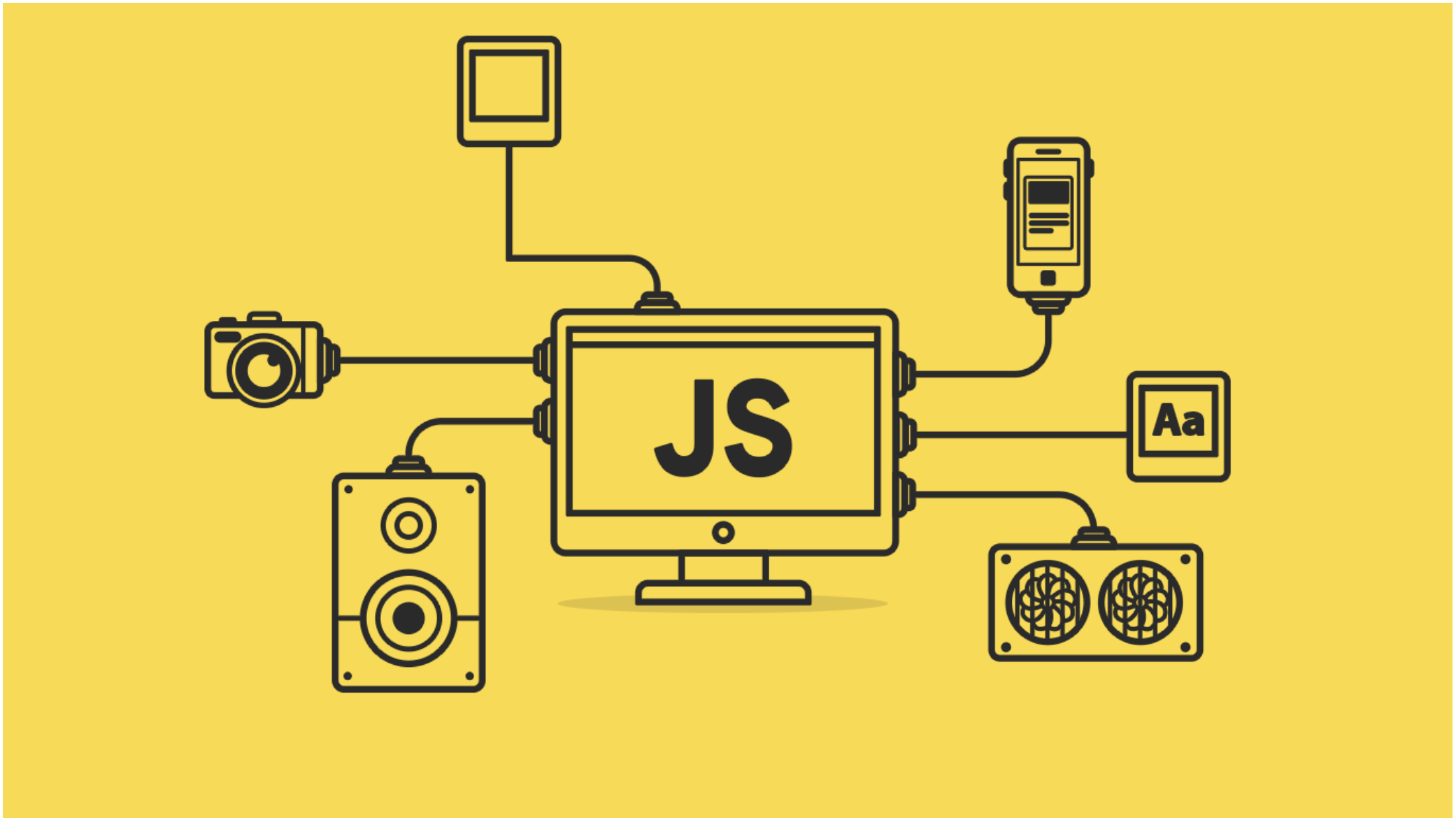
```
let string = "apple";
let strToArr = [...string];
console.log(strToArr);

let strToArr2 = str.split("");
console.log(strToArr2);
```

- 전개 연산자(…) 사용으로 “apple” 을 “a”, “p”, “p”, “l”, “e” 로 변환
- 그것을 [] 안에 넣어서 배열로 변환!

Module

사용하기!

JS Module


JS
Aa

JS 에서 Module 사용하기

- JS 도 Module 을 지원 합니다!
- 다른 사람이 만든 기능을 활용 할 때, 매번 코드를 받아서 붙여 넣기는 귀찮겠죠?
- 이럴 땐, 파일로 받아서 사용하는 편이 더 편할 겁니다!
- 그래서 JS도 파일을 Module 로 불러와서 사용이 가능합니다!
- 그런데 방법이 2가지가 있습니다!
  - CommonJS 방식 / ES6 방식

CommonJS

방식

CommonJS 방식

- Node.js 에서 사용되는 모듈 방식 입니다!
- 키워드로는 require / exports 를 사용합니다.
- Require 로 모듈을 불러 올 때, 코드 어느 곳에서나 불러 올 수 있습니다!

CommonJS 방식

- 전체 모듈로써 내보내고, 전체를 하나의 Obj 로 받아서 사용하는 방법

```
const animals = ['dog', 'cat'];

function showAnimals() {
  animals.map((el) => console.log(el));
}

module.exports = {
  animals,
  showAnimals,
};
```

Animal.js

```
const animals = require('./animals');

console.log(animals);
animals.showAnimals();
```

Module.js

CommonJS 방식

- 내보내고 싶은 것(변수, 함수, 클래스 등등)에 exports 를 붙여서 내보내고, 각각을 따로 선언해서 가져 오는 방식

```
const animals = ['dog', 'cat'];

exports.animals = animals;

exports.showAnimals = function showAnimals() {
  animals.map((el) => console.log(el));
};
```

Animal.js

```
const { animals, showAnimals } = require('./animals');

console.log(animals);
showAnimals();
```

Module.js

CommonJS 방식

- 하나의 객체(or 클래스)에 전부를 넣어놓고 그 객체 자체를 내보내는 방식!

```
const animals = {
  animals: ["dog", "cat"],
  showAnimals() {
    this.animals.map((el) => console.log(el));
  },
};

module.exports = animals;
```

Animal.js

```
const { animals, showAnimals } = require('./animals');

console.log(animals);
showAnimals();
```

Module.js

ES6 방식

ES6 방식

- 2015년에 ES6 가 업데이트 되면서 추가 된 방식입니다.
- 사실 ES6 는 브라우저에서 구동되는 JS에 대한 문법이라서 Node.js 에서는 사용하려면 별도의 처리가 필요합니다!
- 예전에는 파일 확장자를 ~~.mjs 로 처리해서 구분
- 요즘에는 package.json 에 아래의 문구를 추가해 주면 사용이 가능합니다!

```
"type": "module",
```

- ES6 는 CommonJS 처럼 전체를 내보내는 방식이 존재하지 않으며, 필요한 모듈을 설정해서 원하는 것만 내보내고 받는 방식을 씁니다!(사실 CommonJS도 그렇게 변경이 되었습니다!)

ES6 방식 - export

- 선언부에 Export 사용하기

```
export const animals = ['dog', 'cat'];

export function showAnimals() {
  animals.map((el) => console.log(el));
}
```

Animal.js

```
import { animals, showAnimals } from './animals.js';

console.log(animals);
showAnimals();
```

Module.js

ES6 방식 - export

- 마지막에 Export 사용하기

```
const animals = ['dog', 'cat'];

function showAnimals() {
  animals.map((el) => console.log(el));
}

export { animals, showAnimals };
```

Animal.js

```
import { animals, showAnimals } from './animals.js';

console.log(animals);
showAnimals();
```

Module.js

ES6 방식 - import

- 가져올 것들이 많으면 * as 를 사용

```
import * as animals from './animals.js';

console.log(animals);
animals.showAnimals();
```

Module.js

- 단, 보통의 경우는 가져올 것을 확실히 명시하는 편이 좋습니다!
  - 메모리 효율 및 처리 속도가 올라갑니다!

ES6 방식 – 모듈 이름 바꾸기(import)

- 모듈 이름을 바꾸고 싶으면 모듈이름 as 새로운모듈이름 으로 변경이 가능 (import, export 동시 적용 가능)

```
import { animals as ani, showAnimals as show } from './animals.js';

console.log(ani);
show();
```

```
export const animals = ['dog', 'cat'];

export function showAnimals() {
  animals.map((el) => console.log(el));
}
```

ES6 방식 – 모듈 이름 바꾸기(export)

- 모듈 이름을 바꾸고 싶으면 모듈이름 as 새로운모듈이름 으로 변경이 가능 (import, export 동시 적용 가능)

```
import { ani, show } from './animals.js';

console.log(ani);
show();
```

```
const animals = ['dog', 'cat'];

function showAnimals() {
  animals.map((el) => console.log(el));
}

export { animals as ani, showAnimals as show };
```

ES6 방식 – export default

- Export default 로 내보내진 모듈을 Import 할 때에는 {} 를 사용 X

```
const animal = {
  animals: ['Dog', 'Cat'],
  showAnimals: function () {
    for (let i = 0; i < this.animals.length; i++) {
      console.log(this.animals[i]);
    }
  },
};

export default animal;
```

```
import animal from './animal.mjs';

console.log(animal.animals);
console.log(animal.showAnimals());
```

CommonJS 와 ES6 의 차이점

CommonJS 와 ES6 의 차이점

- CommonJS 는 Node.js 에서 사용되고 require / exports 사용
- ES6 는 브라우저에서 사용되고 import / export 사용
- ES6를 Node.js 에서 사용하고 싶으면 package.json 에 `"type": "module"` 추가 필요
- Require 는 코드 어느 지점에서나 사용 가능 / Import 는 코드 최상단에서만 사용 가능
- 하나의 파일에서 둘 다 사용은 불가능!
- 일반적으로 ES6 문법이 필요한 모듈만 불러오는 구조를 가지기에 성능이 우수, 메모리 절약 → 그런데 CommonJS 도 해당 문법이 추가 되었음!

실습, 모듈 사용!

- 각각 human.js 와 student.js 를 만들기
- human.js 는 [‘철수’, ‘영희’] 데이터와 데이터를 전부 출력하는 showName() 메소드가 있음
- student.js 는 [‘세호‘, ‘재석’] 데이터와 데이터를 전부 출력하는 showStudent() 메소드가 있음
- common.js 파일에서는 human.js 를 commonJS 방식으로 모듈을 불러서 데이터를 출력
- es6.js 파일에서는 student.js 를 ES6 방식으로 모듈을 불러서 데이터를 출력