Seoul Software ACademy

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

With. 팀 뤼쳐드

구조 분해 할당

구조분해 할당 (Destructuring assignment)

- 구조분해 할당 배열이나 객체의 속성을 해체해 그 값을 개별변수에 담는 것
- JavaScript에서 많이 쓰이는 자료구조인 배열과 객체를 편하게 사용하기 위함
  - 즉, 객체나 배열에 저장된 데이터의 일부를 가져오고 싶을 때 주로 사용
  - 배열 구조 분해
  - 객체 구조 분해

배열의 구조 분해 할당

- arr[0], arr[1] 처럼 접근하는 것이 아닌 각각의 배열 요소를 변수의 이름으로 접근하기 위해서 사용

```
const arr5 = ['tomato', 'kiwi', 'banana'];
const [tomato, kiwi, banana] = arr5;
console.log(tomato); //'tomato'
```

  - 아래와 같음, 직접 할당하는 것보다 간단함

```
let tomato = arr5[0]
let kiwi = arr5[1]
let banana = arr5[2]
```

  - 변수를 선언한 순서대로 배열의 요소가 값으로 할당됨

배열 구조 분해 할당

const [변수] = 배열;

- 각 변수에 배열의 인덱스 순으로 값 대응
- 구조분해 시 변수의 값이 undefined 일 때 기본값 할당 가능
- 구조분해 없이 두 변수의 값 교환도 가능

배열 구조 분해 할당

```
let lists = ['apple', 'grape'];
[item1, item2, item3 = 'peach'] = lists;

console.log( "item1 : ", item1 );
console.log( "item2 : ", item2 );
console.log( "item3 : ", item3 );
```

```
let x = 1, y = 3;
[x,y] = [y,x];
console.log( x, y );
```

오브젝트(객체)의 구조 분해 할당

- 배열의 구조 분해 할당처럼 객체의 속성값을 오브젝트의 key로 접근하는 것이 아닌 변수로 접근하기 위해서 사용

```
const obj = {
  title: '제목',
  content: '내용',
  num: 0,
};
const { title, num, content } = obj
console.log(content); //'내용'
```

```
const title = obj.title;
const num = obj.num;
const content = obj.content;
```

  - 배열의 구조분해 할당과 달리 변수 선언 순서에 따라서 값이 할당되는 것이 아닌 key의 이름에 따라서 변수에 값이 할당됨

객체 구조 분해 할당

const { 변수 } = 객체;

- 객체 안의 속성을 변수명으로 사용
- 콜론(:) 이용해 새 변수명을 선언하고, 원래의 값을 새 변수명에 할당 가능

객체 구조 분해 할당

```
let obj = {
    key1: 'one',
    key2: 'two'
};
let { key1: newKey1, key2, key3 = 'three' } = obj;
console.log( "key1 : ", key1 );
console.log( "newKey1 : ", newKey1 );
console.log( "key2 : ", key2 );
console.log( "key3 : ", key3 );
```

```
let { a, b } = { a: 10, b:20 };
console.log( "a : ", a );
console.log( "b : ", b );

let { c, d, ...rest } = { c: 30, d: 40, e: 50, f: 60 };
console.log( "c : ", c );
console.log( "d : ", d );
console.log( "rest : ", rest );
```

… 연산자

전개구문 (spread) ...

```
const a = [1, 2, 3];
const b = [4, 5];
const spread = [...a, ...b];
console.log(spread);

const c = [..."Hello"];
console.log(c);
```

전개구문 (spread) ...

- 반복 가능한 객체에 사용하는 문법 → 배열, 유사 배열, 문자열 등에 사용 가능
- 객체의 요소에 접근해서 요소들을 하나씩 분리해서 전개요소에 접근해서 반환
- 연산자 ... 사용

- 예를 들어, `[1, 2, 3, 4, 5]` 라는 배열의 요소를 전개하기 위해서는

  `...[1, 2, 3, 4, 5]` 처럼 사용

전개구문 (spread) ...

```
const arr1 = [1, 2, 3, 4, 5];
const arr2 = ["a", "b", "c"];
```

- 두 배열을 이용해서 `[1, 2, 3, 4, 5, "a", "b", "c"]` 처럼 합치려면 어떻게 해야 할까요?

- arr3 이라는 변수에 초기화 시키려면

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

전개구문 (spread) ...

- arr3 이라는 변수에 초기화 시키려면

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

전개구문을 사용해서 쉽게 요소에 접근할 수 있습니다.

실습. Spread 연산자 사용하기

- const word1= “abc”;
  const word2 =“xyz”;
  로 선언

- 두 개의 문자열을 합쳐서 배열로 만들기

- 결과물 `["a","b","c","x","y","z"]`

rest 파라미터

```
const values = [10, 20, 30];

function get(a, ...rest) {
    console.log(rest); //결과는 [20, 30]
}
get(...values);
```

spread vs. rest

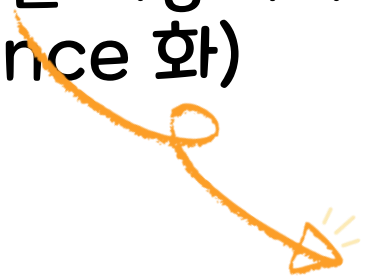
- spread 파라미터
  - 호출하는 함수의 파라미터에 사용
- rest 파라미터
  - 호출받는 함수의 파라미터에 사용
  - 호출하는 함수의 파라미터 순서에 맞춰 값 설정 후 남은 파라미터 값을 배열로 설정

클래스

클래스

- ES6 부터 등장한 오브젝트(객체)를 만드는 방법
- 오브젝트(객체)를 만들 수 있는 '틀'(template)
- 정해진 틀로 같은 규격의 오브젝트를 여러 개 만들 수 있음
  - 재사용할 때 유리함

- new 키워드를 이용해서 미리 만들어둔 클래스 형태의 오브젝트를 만들 수 있음 (instance 화)



사용해본 적 있습니다!
const today = new Date()
Date 객체로 today라는 변수를 선언!
JavaScript에 미리 만들어져 있는 Date 클래스를 사용하는 것

클래스

- Car 라는 클래스를 하나 만들어서 Audi, Nissan, Volvo 등의 여러 개의 오브젝트를 만들 수 있어요.

클래스 생성

- 클래스 생성시 클래스의 이름은 PascalCase

```
class House {
  // 생성자 함수, 객체의 속성(내부에서 사용할 변수) 부여
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }

//  객체 메소드
  age() {
    console.log(`건축한지 ${2023 - this.year}년 되었어요`);
  }
}
```

클래스 상속!

- extends 라는 키워드 사용해서 '상속'을 받을 수 있어요.
- 상속을 이용하면, 기존에 있던 클래스의 속성과 메소드를 받아와서 사용하되, 추가적인 속성과 메소드를 더 정의할 수 있습니다.

```
class Apartment extends House {
  ...
}
```

  - 미리 만들어둔 House 클래스의 속성(변수)과 메소드를 사용할 수 있어요.
  - Apartment만의 속성과 메소드를 추가할 수 있어요.

실습. Shape 클래스 만들기

- Shape(직사각형) 클래스의 속성: 가로와 세로
- Shape 클래스의 메소드 getArea()
  - 넓이 반환하는 메소드(가로 * 세로)

```
let rec1 = new Shape(3,4);
console.log(rec1.getArea()); → 12 가 나오는지 확인
```

실습. 클래스 상속 (선택)

1. Rectangle(직사각형) 클래스만들기
    - Shape 클래스 상속
    - 사각형의 넓이 구하는 메소드 getArea()
    - 직사각형의 대각선 길이 구하는 메소드 추가 (Math.sqrt() 이용)

Math.sqrt(n**2) == n입니다.
Math.sqrt(9)==3
제곱근 구하는 함수

2. Triangle(삼각형) 클래스 만들기
    - 넓이 반환하는 메소드 getArea()
3. Circle (원) 클래스 만들기
    - Shape 클래스를 상속
    - width, height 이외에 radius 생성자 추가
    - getArea() 메소드는 원의 넓이를 리턴

실습. 클래스 상속 (선택)

- Shape를 상속받은 각각의 클래스 Rectangel, Triangle, Circle 클래스를 이용해서 인스턴스 하나씩 생성
- getArea로 사각형, 삼각형, 원의 넓이가 잘 나오는지 확인하기

모듈

- 모듈 이름은 PascalCase
- 기능 단위로 모아둔 함수, 변수 등
- /node_modules ← Node.js 의 모듈을 모아놓은 폴더!

- 모듈은 누구나 만들 수 있고, 다른 사람이 만든 모듈을 우리가 사용할 수도 있습니다

ES2015 모듈

- 자바스크립트 자체 모듈 시스템 문법
- 노드 모듈 시스템과 방식이 약간 다름
- package.json 에 “type”: “module” 을 추가해 사용

```
"main": "index.js",
"type": "module",
▷ Debug
"scripts": {
```

ES6 모듈

export : 모듈 내보내기

```
const a = "a 변수";
const b = "b 변수";

module.exports = {
    a,
    b
};
```

```
const a = "a 변수";
const b = "b 변수";

export { a, b };
```

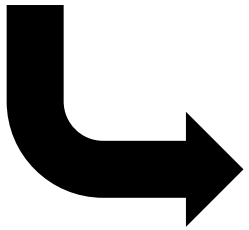
```
function connect() {
    return a + b;
}

module.exports = connect;
```

```
function connect() {
    return a + b;
}
export default connect;
```

ES6 모듈

import ~ from ~ : 모듈 가져오기

```
const { a, b } = require("./var.js" );
const returnString = require("./func.js");
```



```
import { a, b } from './var.js';
import returnString from './func.js';
```

노드 모듈 불러올 때 주의할 점

- const { } 로 가져올 때는 객체 구조분해해 가져오기에 **이름이 동일**해야 함

```
const math = require('./math2');
console.log(math);

const { add, E, PI } = require('./math2');
console.log(add(E, PI));
```

- 하나만 내보낸 모듈은 이름이 달라져도 불러올 수 있음

```
const add = require('./math');
console.log(add);

const onlyOne = require('./math');
console.log(onlyOne);
```

common JS vs. ES6 (ES2015)

- common JS
  이전에 사용되던 자바스크립트 표준(~ES5)

- ES6
  - ECMAScript version6
  - 가장 최근에 출시된 자바스크립트 표준 (2015)
  - let, const, 템플릿 리터럴(`${}`), 화살표 함수, 구조분해 할당, for of, 클래스 .. 등 가장 최근에 등장한 JS 표준
  - ES6 에서 모듈을 사용하는 방법도 재정의됨
    - export & import

https://github.com/tc39

ES6 이전에서의 모듈 사용 (Common JS)

- require/ exports 사용
  - 만든 모듈을 exports 시켜 외부에서도 사용할 수 있도록 하고,
  - require 을 통해서 사용하고 싶은 모듈을 불러옵니다.
- Node.js 에서 사용하는 방식!

1. 하단에서 한 번에 module.exports ={}; 시키는 방법
2. 내보내고 싶은 변수, 함수, 클래스 등에 exports.~ 키워드 붙이기

ES6 방식 모듈

- import/export 키워드 사용
  - 만든 모듈을 export 시켜서 다른 곳에서 사용할 수 있도록
  - 만들어진 모듈을 import 를 이용해서 사용

1. 하단에서 한번에 export {}; 시키는 방법
2. 외부에서 사용하고 싶은 데이터와 메소드에만 export 키워드 붙이는 방법

ES6 방식 모듈 : export default

- 여러 개의 메소드, 객체 등을 export 할 것이 아니고 하나의 큰 기능을 담당하는 function, class, .. 등을 export 할 때 사용!
- 해당 파일에는 export 할 개체가 하나만 있다고 알려주는 키워드

```
export default function sayHi() {
  console.log('hi');
}
```

~.js 에는 sayHi라는 function 만 export 할 것이다!

- Export default 로 내보내진 모듈을 Import 할 때에는 {} 를 사용하지 않고 불러옵니다.

```
import sayHi from './05_module.js';
sayHi();
```