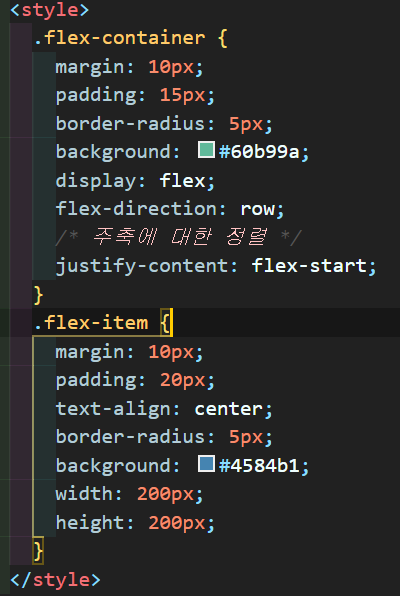
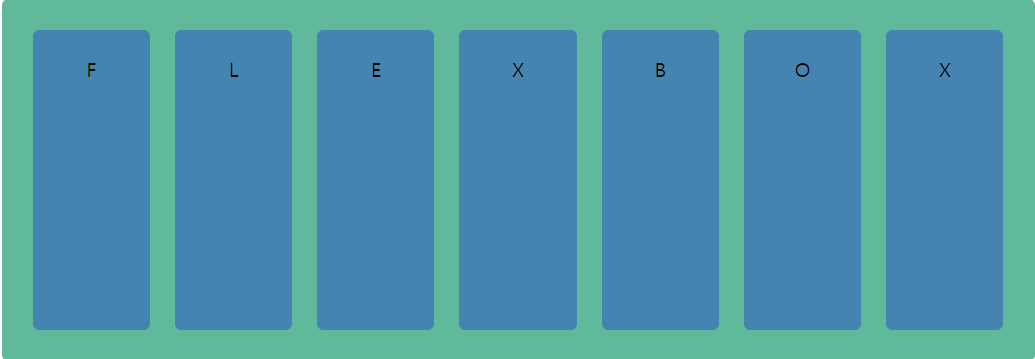
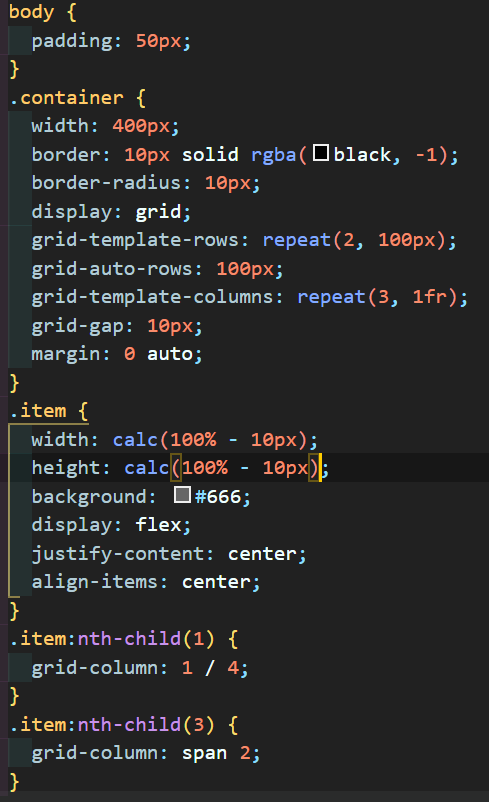
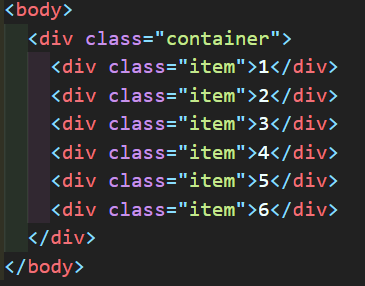
**1.FLEX레이아웃과 GRID 레이아웃을 설명하고 예제를 만드시오.**

FLEX 레이아웃 : 일명 flexbox라 불리는 Flexible Box module은 flexbox 인터페이스 내의 아이템 간 공간 배분과 강력한 정렬 기능을 제공하기 위한 1차원적인 레이아웃 모델로 설계되었다. Flexbox를 보통 1차원이라 칭하는데, 이는 레이아웃을 다룰 때 한 번에 하나의 차원(행이나 열) 만을 다룬다는 뜻이다.



Grid Layout : 페이지를여러 주요 영역으로 나누거나, 크기와 위치 및 문서 계층 구조의 관점에서 HTML 기본 요소로 작성된 콘트롤 간의 관계를 정의하는 데 아주 탁월하다. 세로 열과 가로 행을 기준으로 요소를 정렬할 수 있다.

   
  
출처 : <https://poiemaweb.com/css3-flexbox>

<https://heropy.blog/2019/08/17/css-grid/>

**2.ES6문법을 10가지이상 설명하고 각 예제를 만드시오.**

**\* const / let 블록 스코프**

var의 변수 스코프는 function 단위, const / let은 block 단위, const는 상수, let은 변수

ex)

var a = 1;

let b = 2;

if (true) {

var a = 11;

let b = 22;

console.log(‘a = ‘ + a); // 11

console.log(‘b = ‘ + b); // 22

}

console.log(‘a = ‘ + a); // 11

// a는 var로 선언되어 function scope이므로 if 문 안에서 global variable a의 값이 11로 변경된다.

console.log(‘b = ‘ + b); // 2

function func( ) {

var a = 111;

let b = 222;

console.log(‘a = ‘ + a); // 111

console.log(‘b = ‘ + b); // 222

}

func();

console.log(‘a = ‘ + a); // 11

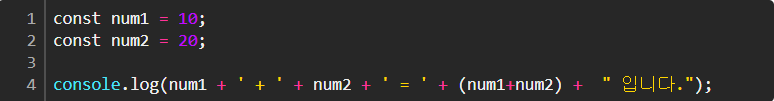
// var은 function scope이므로 func( ) 내부에 존재하는 변수 a 의 값은 그 함수 내부에서만 존재한다.

console.log(‘b = ‘ + b); // 2

* **템플릿 리터럴(백틱 적용)**

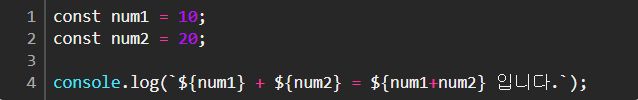
백틱(`)을 사용하면 ${ }를 사용해서 문자열과 변수를 적절히 같이 사용할 수 있다.

이전 코드



// 10 + 20 = 30입니다.

백틱 적용 코드



문자열 사이에 변수만 따로 구분할 수 있게 해서 좀 더 직관적으로 볼 수 있다.

뿐만 아니라, 백틱(`) 내에서 줄이 바뀌면 알아서 줄 바꿈 출력이 된다.

* 화살표 함수 표현(arrow function expression)

Function expression에 비해 구문이 짧고 자신의 this, arguments, super 또는 new.target을 바인딩하지 않는다. 화살표 함수는 항상 익명이다.

ex)

const materials = [

'Hydrogen',

'Helium',

'Lithium',

'Beryllium'

];

console.log(materials.map(material => material.length));

// expected output: Array [8, 6, 7, 9]

* 클래스

Prototype 기반의 대체재로 쓰임.

ex)

class Shape {

constructor( ) { }

};

class Rectangle extends Shape {

constructor(w, h) {

super(w, h);

this.w = 20;

this.h = 10;

}

getArea(w, h) {

return w \* h;

}

}

let rect = new Rectangle( );

console.log(rect.getArea(30, 20)); // 600

* 모듈

현재 웹에서는 해당 페이지에 필요한 모든 파일을 미리 불러와야 하고, 그 파일들이 사용하는 변수가 겹치지 않나 잘 살펴봐야 한다. 또한 자바스크립트를 로딩하는 순서도 중요하다. 자바스크립트는 패키지와 파일끼리 서로 의존하고 있는 경우가 많다. 의존성 관리를 하기 위해서 모듈 시스템이 만들어졌는데 모듈 시스템은 파일 첫 부분에 이 파일은 어떤 패키지를 필요로 한다고 선언한다. 그러고 나서 선언한 패키지만 그 파일에서 사용한다.

Ex)

// js/main.js

export default {

init() {

console.log("main.js");

},

sum(x, y) {

return x + y;

}

};

export const PI = 3.14156265;

// app.js

import Main from './js/main.js';

document.addEventListener('DOMContentLoaded', () => {

Main.init();

console.log(Main.sum(Main.PI \* Main.PI));

});

// otherapp.js

import {sum, PI} from "./js/main";

console.log("sum: " + sum(PI, PI));

// index.html

<script type="module" src="app.js"></script>

* 구조 분해 할당(Destructuring Assignment)

구조 분해 할당 구문은 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 JavaScript 표현식이다.

Ex)

const [a, b, c] = [1, 2, 3];

console.log(a, b, c); // 1, 2, 3

const obj = { x: ‘banana’, y: ‘apple’ }

let {x, y, z} = obj;

console.log(x, y, z); // x: ‘banana’, y: ‘apple’, z: undefined

function f({name: x}) {

console.log(x); // x: 8

}

f({name: 8});

var [u = 1] = [ ];

console.log(u === 1); // true

* Spread(…) 연산자

배열을 spread 연산자를 통해서 찍어보면 배열이 아닌 개별적인 요소가 나온다. 간단히 생각하면 배열이 각각으로 나뉜다고 생각하면 될 것같다.

Ex)

const arr = [1, 2, 3, 4, 5];

console.log(arr); // [1, 2, 3, 4, 5]

console.log(…arr); // 1, 2, 3, 4, 5

console.log(1, 2, 3, 4, 5); // 1, 2, 3, 4, 5

const a = […arr];

// 이 코드는 arr을 spread 연산자로 전개한 것을 다시 a의 배열로 넣는 것이다.

console.log(a); // [1, 2, 3, 4, 5];

이렇게 배열을 전개하는 것이 spread 연산자이다.

* Rest 파라미터

Rest 파라미터도 마찬가지로 …를 통해 사용할 수 있다. Rest 파라미터 구문은 정해지지 않은 수(an indefinite number, 부정수) 인수를 배열로 나타낼 수 있게 합니다.

Rest 파라미터는 쉽게 말해 spread 연산의 반대라고 생각하면 편하다. Spread 연산은 배열을 개발적으로 전개하지만, Rest 파라미터는 개별을 배열로 묶어준다.

Ex)

function func(…param) {

console.log(param);

}

func(1, 2, 3); // [1, 2, 3]

// param은 배열이기 때문에 관련 메서드를 사용할 수 있다.

출처 : <https://velog.io/@decody/ES6-Sheetsheet>

<https://leeborn.tistory.com/entry/JavaScript-ES2015-%EB%B0%B1%ED%8B%B1%EA%B3%BC-%ED%85%9C%ED%94%8C%EB%A6%BF-%EB%AC%B8%EC%9E%90%EC%97%B4>  
<https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Functions/%EC%95%A0%EB%A1%9C%EC%9A%B0_%ED%8E%91%EC%85%98>  
<https://www.zerocho.com/category/ECMAScript/post/579dca4054bae71500727ab9>

<https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment>

<https://velog.io/@chlwlsdn0828/Js-Spread-%EC%97%B0%EC%82%B0%EC%9E%90-Rest-%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0>

**3. props 와 state는 언제 쓰는가.**

Props는 데이터를 컴포넌트 간 전달하기 위해 쓰여진다.

State는 데이터를 유지하고 저장한다. 데이터를 유지한다라는 측면에서 보면 Props와 같다. 차이점은 State는 컴포넌트에서 생성하고 할당한다. 그리고 데이터를 변경할 수 있다.