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

**FLEX 레이아웃**

기존에는 시맨틱에 맞지않는 방식을 사용했었습니다. <table>태그를 이용하여 의미와는 다르게 사용하거나, float 속성을 이용해서 레이아웃을 잡았습니다.

표준을 지키면서 레이아웃을 편하게 잡을 수 있도록 flex 라는 것을 지원하게 되었습니다.

<container>

<item>

</item>

</containler>

flex는 부모와 자식 구조여야 합니다.

flex의 시작은 부모 태그( container )에 display:flex 스타일을 부여하는 것으로 시작이 됩니다.

**예제**











위와같이 부모에게 display:flex 를 주었을때 자식들은 block 구조에서 inline 구조로 바뀌며 부모에 의해 구조들이 변경됩니다.

**출처** : <https://victorydntmd.tistory.com/188>

**GRID 레이아웃**

CSS Grid는 2차원(행과 열)의 레이아웃 시스템을 제공합니다.

Flexible Box도 훌륭하지만 비교적 단순한 1차원 레이아웃을 위하며, 좀 더 복작한 레이아웃을 위해 CSS Grid를 사용할 수 있습니다.

CSS Grid는 Flex와 같이 Container와 Item 이라는 두 가지 개념으로 구분되어 있습니다.

Container는 Items를 감싸는 부모 요소이며, 그 안에서 각 Item을 배치할 수 있습니다.

**grid-template-rows, grid-template-columns**

명시적 행 , 열(Track)의 크기를 정의합니다.

동시에 라인(Line)의 이름도 정의할 수 있습니다.

fr(fraction, 공간 비율) 단위를 사용할 수 있습니다.

repeat() 함수를 사용할 수 있습니다.





각 라인은 행(Row, Track)과 열(Column, Track)의 개수대로 숫자(양수/음수) 라인 이름이 자동으로 지정되어 있어서, 꼭 필요한 경우가 아니면 라인 이름을 정의할 필요가 없습니다.

**예제**

grid를 사용하여 정사각형을 3 x 3 구조로 만드시오

item인 정사각형의 width와 height는 100px 입니다.

박스의 전체 크기는 width,height 500px 입니다.

**풀이**





**grid-template의 columns는** 가로를 담당합니다. 100px 씩 총 3개를 입력했기에 부모의 가로길이가 아무리 길어도 100px씩 3개의 요소가 자리잡고 나머지는 밑으로 내려오게 됩니다.

**grid-template의 rows는** 세로를 담당합니다. 100px 씩 총 3개를 입력했기에 세로로 3칸까지는 동일한 100px값이 입력됩니다. 단, items의 갯수가 9개 이상일경우엔 처음 입력해둔 3개를 제외하고는 나머지 요소들은 공간을 나눠갖습니다.

**grid-template의 rows ex)**



**2. ES6문법을 10가지이상 설명하고 각 예제를 만드시오.**

**Arrows**

Arrows(화살표) 함수는 => 문법을 사용하는 축약형 함수 입니다. C#, JAVA 8, CoffeeScript의 해당 기능과 문법적으로 유사합니다. Arrows는 표현식의 결과 값을 반환하는 표현식 본문 뿐만 아니라 상태 블럭 본문도 지원합니다. 하지만 일반 함수의 자신을 호출하는 객체를 가리키는 dynamic this와 달리 arrows 함수는 코드의 상위 스코프를 가리키는 lexical this를 가집니다.

function 표현에 비해 구문이 짧고 자신의 this, arguments, super 또는 new.target을 바인딩하지 않습니다. 화살표 함수는 항상 익명 입니다. 이 함수 표현은 메소드 함수가 아닌 곳에 가장 적합합니다. 그래서 생성자로서 사용할 수 없습니다.

**예제**

const materials = [ 'Hydrogen', 'Heilum', 'Lithium', 'Beryllium' ];

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

**결과** : [8, 6, 7, 9]

**주관적인 해설** : map 함수를 사용하여 배열을 재정의하였습니다. Arrows함수를 사용하여 function (){내용} 의 절차를 생략하고 map을 사용할때 주어지는 인자값을 materials(변수명) 를 가지고 길이를 출력하도록 표현되었습니다.

출처 :

MDN : <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>

es6 번역 : <https://itstory.tk/entry/JavaScript-ES6-%EB%AC%B8%EB%B2%95-%EC%A0%95%EB%A6%AC>

**Classes**

ES6 클래스는 프로토타입 기반 객체지향 패턴을 더 쉽게 사용할 수 있는 대체재입니다. 클래스 패턴 생성을 더 쉽고 단순하게 생성할 수 있어서 사용하기도 편하고 상호운용성도 증가됩니다.

class 표현식은 ECMAScript 2015(ES6)에서 클래스를 정의하는 방법입니다. function 식과 비슷하게, class 식은 유명(named) 또는 익명(unnamed)일 수 있습니다. 유명인 경우, 클래스명은 클래스 몸통(body)에서만 지역(local)입니다. JavaScript 클래스는 프로토타입(원형) 기반 상속을 사용합니다.

**예제**

const Rectangle = class {

constructor(height, width) {

this.height = height;

this.width = width;

}

area() {

return this.height \* this.width;

}

};

console.log(new Rectangle(5, 8).area());

결과 : 40

출처 :

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

es6 번역 : <https://itstory.tk/entry/JavaScript-ES6-%EB%AC%B8%EB%B2%95-%EC%A0%95%EB%A6%AC>

**template literal (Template Strings)**

템플릿 리터럴은 내장된 표현식을 허용하는 문자열 리터럴입니다. 여러 줄로 이뤄진 문자열과 문자 보간기능을 사용할 수 있습니다. 이전 버전의 ES2015사양 명세에서는 "template strings" (템플릿 문자열) 라고 불려 왔습니다.

템플릿 리터럴은 이중 따옴표 나 작은 따옴표 대신 백틱(` `) (grave accent) 을 이용합니다. 템플릿 리터럴은 또한 플레이스 홀더를 이용하여 표현식을 넣을 수 있는데, 이는 $와 중괄호( $ {expression} ) 로 표기할 수 있습니다. 플레이스 홀더 안에서의 표현식과 그 사이의 텍스트는 함께 함수로 전달됩니다.

console.log("string text line 1\n"+ "string text line 2");

"string text line 1 string text line 2"

위와같이 사용하던 **Multi-line strings** 방식에서 아래와같이

**template literal로** 사용할 수 있습니다.

console.log(`string text line 1 string text line 2`);

"string text line 1 string text line 2"

**출처**

MDN : <https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals>

ES6 번역 : <https://jsdev.kr/t/es6/2944>

**Let + Const**

블록 유효 범위를 갖는 새로운 변수 선언 방법을 지원합니다. **let은 var와 유사하게 동작**합니다. **const는 재할당 및 재선언이 불가능**합니다.

**var의 유효 범위는 전체 외부 함수**까지이지만 **let은 변수를 선언한 블록과 그 내부 블록들에서 유효**합니다.



**출처**

ES6 번역 : <https://jsdev.kr/t/es6/2944>

**Symbols**

심볼(Symbol)은 객체 상태의 접근 제어를 가능하게 합니다. Symbol은 새로운 원시 타입으로 이름 충돌의 위험 없이 속성(property)의 키(key)로 사용할 수 있습니다. 옵션 파라미터인 description는 디버깅 용도로 사용되며 식별 용도는 아닙니다. Symbol은 고유(unique)하며, Object.getOwnPropertySymbols와 같은 reflection 기능들로 접근할 수 있기 때문에 private 하진 않습니다(for in나 Object.keys()로는 접근 불가).

7번째 타입(type)

1997년 JavaScript가 처음 표준화된 이래로, JavaScript는 6개의 타입을 갖고 있었습니다. ES6가 발표되기 전까지, JS 프로그램 안의 모든 값은 반드시 다음 6개 타입들 중 하나였습니다.

Undefined

Null

Boolean

Number

String

Object

심볼(Symbol)은 프로그램이 이름 충돌의 위험 없이 속성(property)의 키(key)로 쓰기 위해 생성하고 사용할 수 있는 값입니다.

**출처**

ES6 번역 : <https://jsdev.kr/t/es6/2944> , <http://hacks.mozilla.or.kr/2015/09/es6-in-depth-symbols/>

**Binary and Octal**

2진법 (b), 8진법 (o) numeric 리터럴 형식이 추가되었습니다.



**Promises**

Promise는 비동기 프로그래밍을 위한 라이브러리입니다. Promise는 미래에 생성되는 값을 나타내는 일급 객체입니다. Promise는 현존하는 많은 JavaScript 라이브러리에 사용되고 있습니다.

Promise 객체는 비동기 작업이 맞이할 미래의 완료 또는 실패와 그 결과 값을 나타냅니다

Promise는 프로미스가 생성될 때 꼭 알 수 있지는 않은 값을 위한 대리자로, 비동기 연산이 종료된 이후의 결과값이나 실패 이유를 처리하기 위한 처리기를 연결할 수 있도록 합니다. 프로미스를 사용하면 비동기 메서드에서 마치 동기 메서드처럼 값을 반환할 수 있습니다. 다만 최종 결과를 반환하지는 않고, 대신 프로미스를 반환해서 미래의 어떤 시점에 결과를 제공합니다.

Promise는 다음 중 하나의 상태를 가집니다.

대기(pending): 이행하거나 거부되지 않은 초기 상태.

이행(fulfilled): 연산이 성공적으로 완료됨.

거부(rejected): 연산이 실패함.

대기 중인 프로미스는 값과 함께 이행할 수도, 어떤 이유(오류)로 인해 거부될 수 있습니다. 이행이나 거부될 때, 프로미스에 연결한 처리기는 그 프로미스의 then 메서드에 의해 대기열에 오릅니다. 이미 이행했거나 거부된 프로미스에 연결한 처리기도 호출하므로, 비동기 연산과 처리기 연결 사이에 경합 조건race condition은 없습니다.

Promise.prototype.then() 및 Promise.prototype.catch() 메서드의 반환 값은 다른 프로미스이므로, 서로 연결할 수 있습니다.



**출처**

ES6 번역 : <https://jsdev.kr/t/es6/2944> , <https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Promise>

**Tail Calls**

마지막에 호출되는 함수가 호출 스택이 초과되게 하지 않습니다. 재귀 알고리즘을 매우 큰 입력 값에서도 안전하게 만듭니다.



**Modules**

언어 차원에서 컴포넌트 정의를 위한 모듈을 지원합니다. 유명한 JavaScript 모듈 로더들(AMD, CommonJS)의 패턴을 적용시켰습니다. 런타임 동작은 호스트에 정의된 기본 로더에 의해 정의됩니다. 묵시적 비동기 형태로 요구되는 모듈들이 정상적으로 로드되기 전까지 코드가 실행되지 않습니다.







export default와 export \* 문법도 제공합니다.





react 처럼 사용할 함수나 변수들을 export 하고 사용할 위치에서 import 하여 사용합니다.

**Destructuring assignment**

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

Object 구조



Array 구조



react에서 useState를 사용할 때 변수에 값을 담는 방식이 구조 분해 할당인지 궁금해졌습니다.

react ex) [data, setData] = useState(");

출처

<https://sang12.co.kr/228/const-%7B%7D-%3D-props-es6-%EB%AC%B8%EB%B2%95-%EA%B5%AC%EC%A1%B0%EB%B6%84%ED%95%B4%ED%95%A0%EB%8B%B9%28Destructuring-assignment%29%EC%9D%B4%EB%9E%80>

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

**3. props 와 state는 언제 쓰는가.**

데이터를 다룰때 사용합니다.

**props** 는 부모 컴포넌트가 자식에게 주는 데이터 값이며, 말 그대로 자식에게 값을 넘겨줄때 사용합니다. 단, 값의 수정은 허용하지 않습니다. 읽기전용.

**state** 는 컴포넌트 내부에서 선언하며 내부에서 값을 변경할 수 있습니다.

동적인 데이터를 사용할때 주로 사용합니다.

본인 컴포넌트 내부에서 선언 과 사용을 할 수도 있지만, 보통 props와 동일하게 상위 컴포넌트에서 선언하고 관리하며 값을 자식 컴포넌트로 내려보냅니다.