7월 3일 기술면접 문제

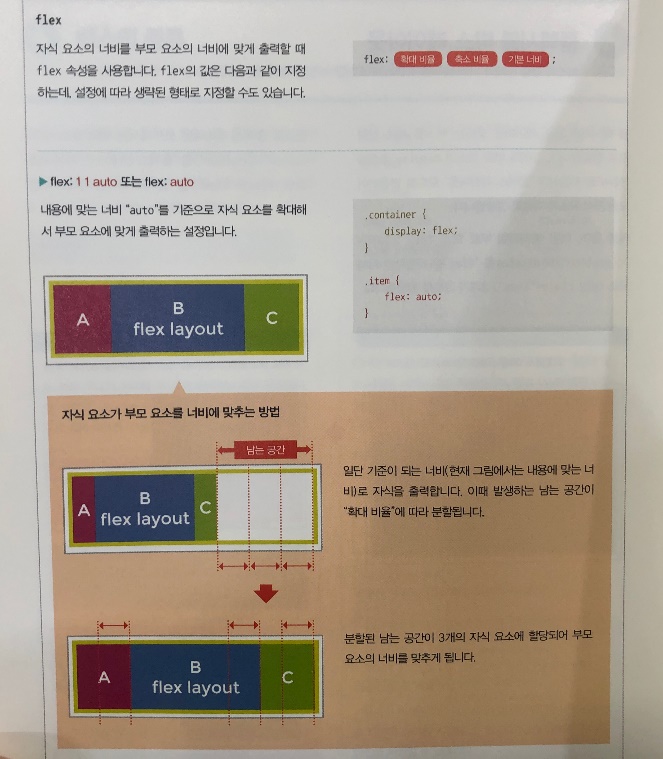
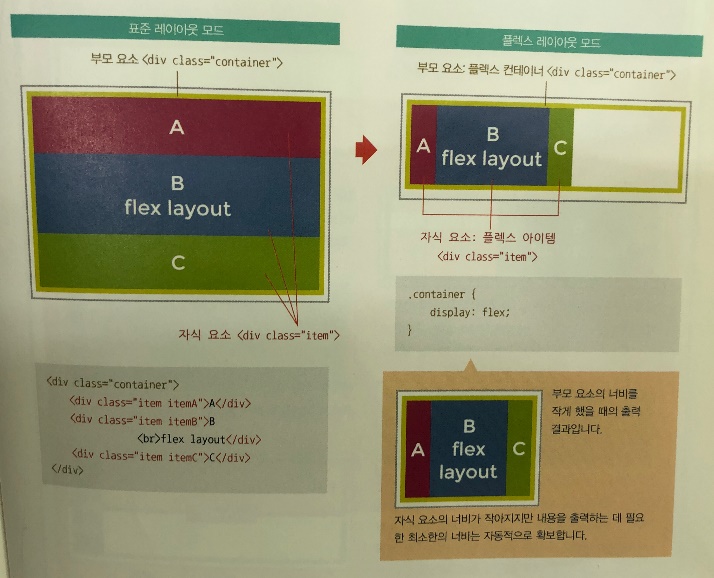
1. FLEX레이아웃과 GRID 레이아웃을 설명하고 예제를 만드시오.

- Flex 레이아웃:

일명 flexbox라 불리는 Flexible Box module, 즉 Flex 레이아웃은 flexbox 인터페이스 내의 아이템 간 공간 배분과 강력한 정렬 기능을 제공하기 위한 1차원적인 레이아웃 모델로 설계되었습니다. flexbox를 1차원이라 칭하는 것은, 레이아웃을 다룰 때 한 번에 하나의 차원(행이나 열)만을 다룬다는 뜻입니다. 이는 행과 열을 함께 조절하는 CSS 그리드 레이아웃의 2차원 모델과는 대조됩니다.

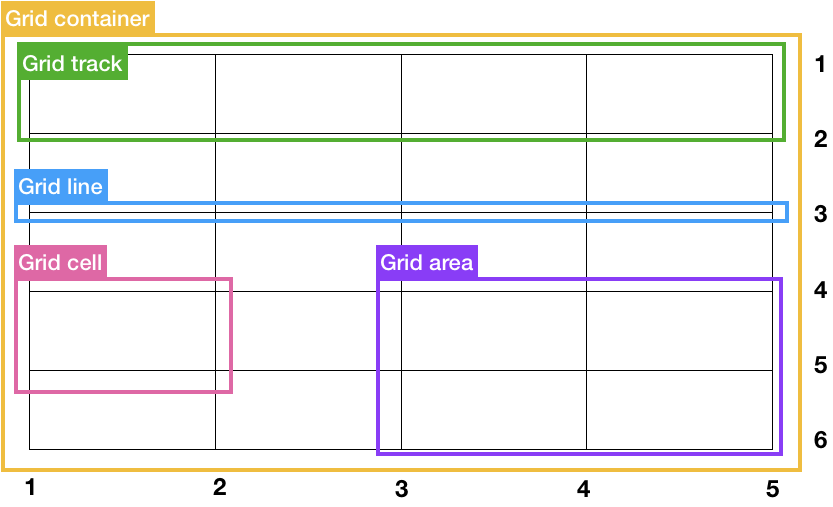
이 레이아웃을 형성하는 가장 기본적인 모델은 플렉스 컨테이너(flex container)와 관련이 있습니다. 이 컨테이너의 콘텐츠를 플렉스 항목(flex items)이라고 하며, 이 플렉스 항목의 가장 기본적인 기능은 크기를 "유연"하게 조절하는 능력입니다. 즉, 공간에 따라서 가로 또는 세로의 길이를 조절할 수 있으며 가용 공간에 대한 비율을 기반으로 크기를 조절할 수도 있습니다.

플렉스 항목이 자랑하는 또 하나의 인상적인 기능은 시각적인 순서와 방향을 정할 수 있다는 것입니다. 간단히 말해서 플렉스박스 레이아웃 모듈은 우리가 원하는 순서나 방향에 따라 플렉스 항목들이 나타나게 할 수 있고 실제 마크업된 순서와 상관없이 별개로 동작하기 때문에 접근성에 제약을 받지 않는다는 점입니다.

또한 플렉스 박스 모듈은 정렬을 제어하는 능력을 가지고 있습니다. 너비와 간격을 대등하게 하고 또 단의 높이를 동일하게 할 수 있으며 수직 정렬(vertical align)도 간단히 제어할 수 있습니다. 

- Grid 레이아웃:

CSS Grid는 CSS Flex와 같이 Container(컨테이너)와 Item(아이템)이라는 두 가지 개념으로 구분되어 있습니다. Container는 Items를 감싸는 부모 요소이며, 그 안에서 각 Item을 배치할 수 있습니다.



출처:

<https://webclub.tistory.com/628>

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

<https://armadillo-dev.github.io/css/understanding-grid-layout/>

<최신 표준 HTML+CSS 디자인> EBISUCOM 저

2. ES6문법을 10가지이상 설명하고 각 예제를 만드시오.

1. let

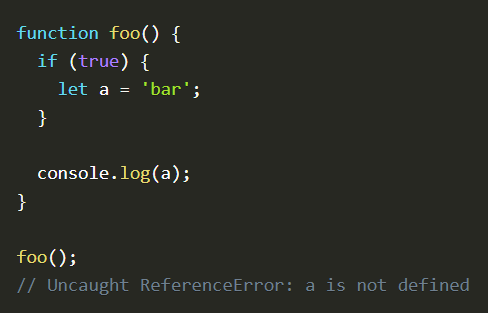
- 기존의 var를 대체하는 키워드

- 기존의 함수 스코프 변수인 var는 전역 범위로 스크립트 내 어디서건 참조 가능하다.

- 반면에 블록 스코프 변수인 let은 자신을 정의한 블록에서만 접근 가능하다.

- 동일한 변수명으로 변수를 재선언시 let은 'Type Error’를 발생시킨다.

- 보다 명확한 변수사용으로 디버깅시 오해의 소지가 줄어들며 개발자의 생산성을 향상시켜준다.



↑ 하지만 let 으로 선언한 변수는 if문의 scope 내에서만 유효하기 때문에 해당 scope 밖에서 a 에 접근할 때, 오류가 발생한다.

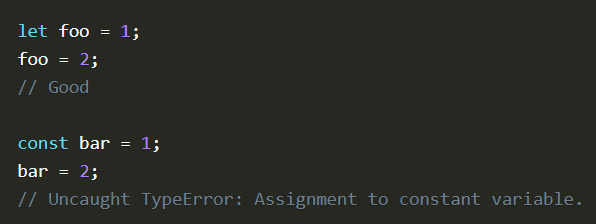
2. const

- ES6에서는 const를 사용하여 상수형으로 코딩이 가능하다.

- 담긴 값이 불변을 뜻하는 게 아니라, 단지 변수의 식별자가 재할당 될 수 없다.

- 변하지 않음을 확신할 수 있는 변수는 const 로 선언하는 것이 바람직하다.

- 변수 스코프는 let과 동일하다.



↑ let 으로 선언된 변수는 오류가 없지만, const 로 선언된 상수는 값을 변경할 수 없다는 오류가 발생한다.

3. 모듈 (export / import)

기존의 자바스크립트에서는 모듈화를 통한 분리가 불가능했다. 하지만, ES6 에서는 드디어 export 와 import 키워드로 모듈화 구현할 수 있게 되었다. 모듈화를 하였을때, 각 파일은 각자의 스코프를 갖게 된다.

- Export: 반대로 스크립트 내의 특정 함수, 객체, primitives를 내보내는 키워드

- Import: 다른 스크립트의 특정 함수, 객체, primitives를 사용하기 위해 들여오는 키워드

다른 파일에서 어떤 객체에 접근하고 싶다면, export 를 통해 외부에 공개한다. Export 하는 방법도 크게 두가지가 있다.

1) Named Export : 이름을 지정해서 export 하는 방법이다. import 할때도 해당 이름을 사용하여 불러온다.

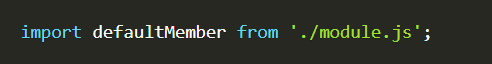
2) Default Export: 모듈 당 단 한번 만 할 수 있는 Export 이다. Named Export 는 import 할때도 이름을 지정해야하지만, default 로 export 해준 값은 간단하게 import 할 수 있다.



↑ Named Export 방식으로 export 된 모듈을 불러오는 가장 간단한 방법이다. module.js 에서 export 된 name1 과 name2 를 포함하려고 하는 자바스크립트 파일의 동일한 변수에 할당된다.



↑ export … as … 처럼 import 할 때도 별명을 설정해 줄 수 있다. module.js 의 name1 의 값이 index.js 의 newName1 에 할당된다.



export default 로 export 된 멤버를 defaultMember 에 할당한다.

4. Arrow function (화살표 함수)

- 기존의 function보다 빠르며 간결한 구문을 보여주는 함수이다.

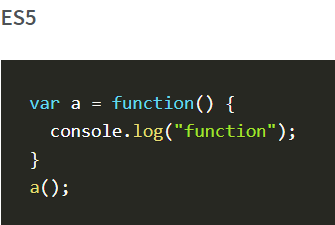
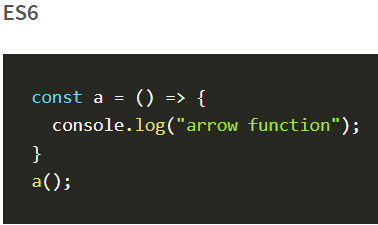
- 항상 익명함수이다.

- 생성자를 사용할 수 없다.

- 가장 중요한 특징 중 첫번째는 간결한 구문이다.

- 두번째는 arrow function의 this값은 해당 스코프의 this값과 같다.

- 기존의 ES5에서의 this는 주로 self(that)나 bind를 사용하여 this를 속박하고 있었다. ES6에서는 이러한 this의 번거로움을 줄이고 해당 arrow function을 감싸고 있는 블록을 this로 가리킨다.

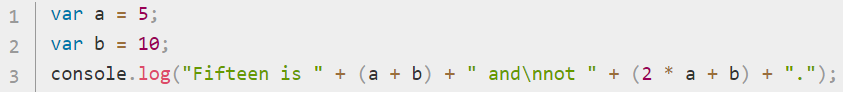
 VS 

5. Template literal (템플릿 리터럴)

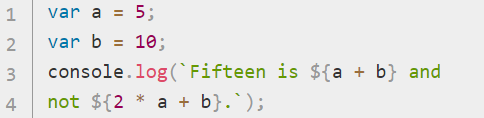
- 내장된 표현식을 허용하는 문자열 리터럴을 뜻한다.

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

- 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.



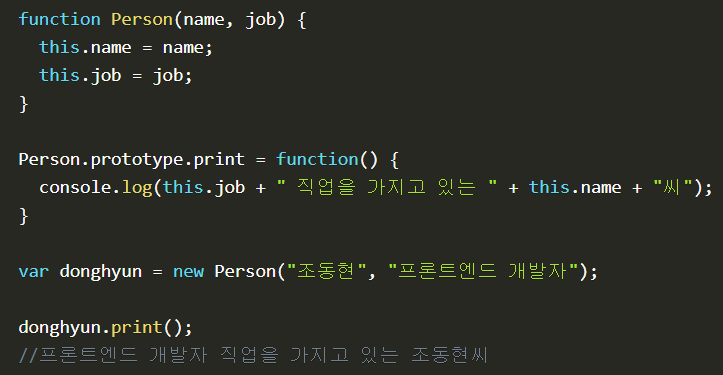
ES5(위) VS ES6(아래)



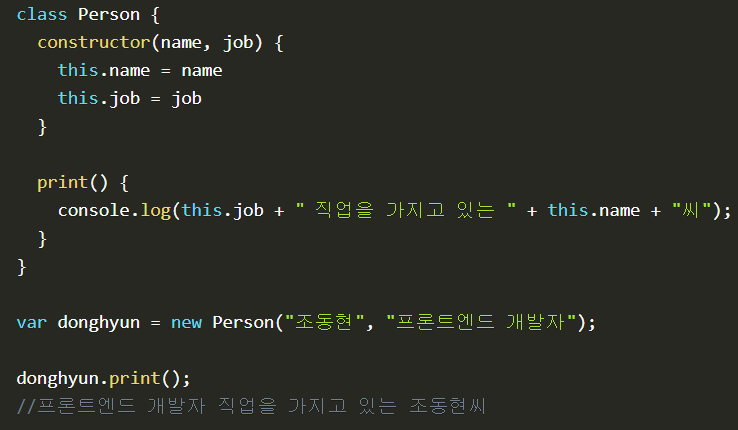
6. 클래스

- 자바스크립트의 객체는 프로토타입이 기반으로써 이 프로토타입 기반 객체지향 패턴을 더 쉽게 사용할 수 있는 대체재입니다. 클래스 패턴 생성을 더 쉽고 단순하게 생성할 수 있어서 사용하기도 편하고 상호운용성도 증가됩니다.

- 더 익숙한 형태로 클래스와 그 안의 메서드를 정의할 수 있다. 하지만, 자바스크립트의 객체 모델이 바뀐 것은 아니며, 내부에서는 그대로 프로토타입 기반으로 작동한다. 이렇게 내부 동작은 동일하지만, 구현 방식에 맞춘 새로운 문법을 문법적 설탕 (Syntactic sugar) 라고 한다.



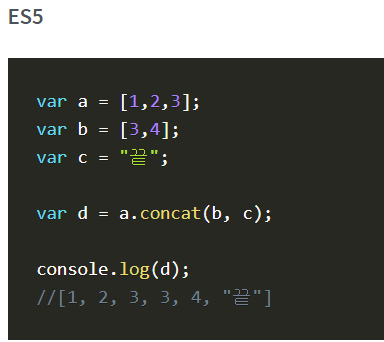
ES5(위) VS ES6(아래)



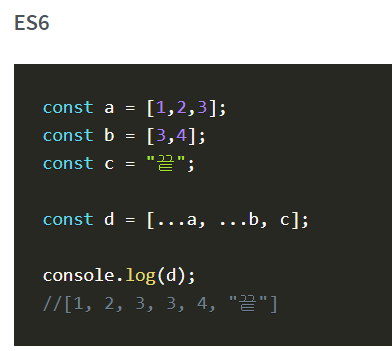
7. Spread operator (전개 연산자)

- 전개 연산자는 점 세개 (…) 로 이루어져 있는 연산자이며, 여러가지 역할을 담당하고 있다.

- ES5 에서는 여러 배열의 내용을 합쳐 새로운 배열을 만들기 위해 concat 메서드를 사용해왔다.



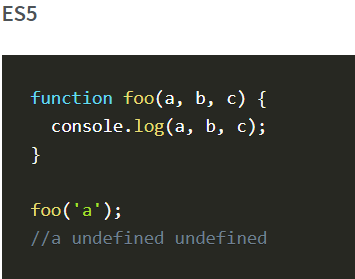
↑ a 배열에서 concat 메서드를 사용해서, 배열 b와 문자열 c를 배열에 이어붙였다. ES6 에서는 전개 연산자를 사용해서 직관적이고 간결하게 표현할 수 있다.



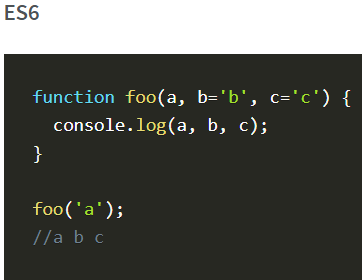
↑ a의 모든요소 …a 와 b의 모든요소 …b 그리고 c를 넣어 새로운 배열 d 를 만든 모습이다. concat 을 사용한 코드보다 훨씬 간결하고, 가독성도 개선되었다.

8. Default Parameter (기본 파라미터)

- ES6 이전엔 파라미터의 값이 들어오지 않으면 무조건 undefined 가 됐지만, ES6 부터 그 기본값을 설정할 수 있다.



↑ 파라미터 a 의 값은 들어왔지만, b 와 c 는 값을 할당받지 못해 undefined 가 된 모습이다. 기본값을 설정하기 위해 파라미터가 undefined 인지 일일이 체크하고, 값을 할당해줘야 했다.



↑ 편하게 기본 파라미터를 설정할 수 있게 되었다.

9. Destructuring assignment (비구조화 할당)

- 비구조화 할당(혹은 구조분해 할당이라고도 한다)은 객체안의 필드를 손쉽게 꺼내어 변수로 대입할 수 있는 문법이다.



↑ 새로운 name, job 두개의 변수를 선언하고, . 키워드로 객체의 필드에 직접 접근하여 값을 가져온다. ES6 에서는 이런 복잡한 과정을 거칠 필요가 없다.



↑ 조금 낯선 문법이 보인다. 변수 선언시에 { } 를 사용한다. 해당 코드는 hudi의 name, job 필드를 같은 이름을 가진 변수에 바로 대입을 해준다.

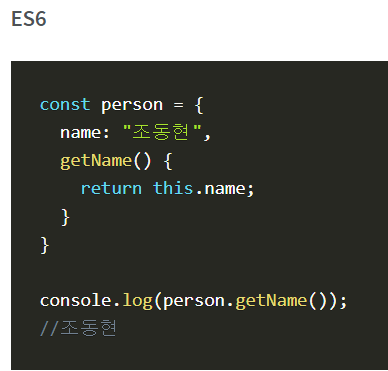
10. Enhanced object literal (향상된 객체 리터럴)

- 단축된 프로퍼티 초기화: 기존 ES5 에서 객체를 생성할 때, 필드명과 대입할 변수명이 같은 상황에서 다음과 같이 코드를 작성하였다.

 VS

↑ key: value 형태에서 단순히 변수명만 작성해주면 변수명과 동일한 필드가 생성되며, 그 변수값이 대입된다.

- 간결한 메서드: ES6 에서는 더 이상 객체 메서드를 정의하기 위해 function 키워드를 사용하지 않아도 된다.

 VS

↑ foo: function(…) { … } 형태로 객체 메서드를 정의해야 하는 ES5와 달리 ES6 에서는 function 키워드가 없어지고, 더욱 간결한 문법을 확인할 수 있다.

출처:

<https://haviyj.tistory.com/3>

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

<https://hudi.kr/es6-%EB%AC%B8%EB%B2%95%EC%9C%BC%EB%A1%9C-%EB%8B%A4%EC%8B%9C-%EC%8B%9C%EC%9E%91%ED%95%98%EB%8A%94-%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8/>

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

3. props 와 state는 언제 쓰는가.

- State: 제품을 구현하는 데에 필요한 내부적 조작장치라고 말할 수 있다. props의 값에 따라 내부에 필요한 데이터들이라 말하는데, 컴포넌트 내부에서 선언하며 내부에서 값을 변경할 수 있다. *즉, 컴포넌트에서 관리하는 상태 값으로 유동적인 데이터를 다룰 때, state를 사용한다.* state는 변경이 가능하고 변경할 때는 setState메서드를 사용해 상태를 변경한다.

- Props: 사용자가 제품을 조작하는 장치라고 말할 수 있다. 사용자가 컴포넌트를 사용하는 입장에서 중요한 것인데, 부모 컴포넌트가 자식 컴포넌트에게 주는 값이다. 자식 컴포넌트에서는 props를 받아오기만 하고, 받아온 props를 직접 수정할 수는 없다. *즉, 컴포넌트에서 사용할 데이터 중 변경되지 않는 데이터를 다룰 때 사용한다.*

출처:

https://velog.io/@dev\_glennn/생활코딩-React-State-Props

https://velog.io/@kyusung/리액트-교과서-React-컴포넌트와-상태-객체