차세대 JavaScript - 요약

이 모듈에서, 저는 몇몇 핵심 차세대 자바스크립트 기능들에 대한 간략한 소개를 해 드렸습니다. 물론 이 과정에서 여러분들이 자주 보시게 될 것들에 초점을 맞추었죠. 여기 간략한 요약이 있습니다!

let & const

`let` 에 대해 더 읽어보기: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

`const`에 대해 더 읽어보기:: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const

`let` 과 `const` 는 기본적으로 `var` 를 대체합니다. 여러분은 `var` 대신 `let` 을 사용하고, `var` 대신 `const`를 사용하게 됩니다. 만약 이 "변수"를 다시 할당하지 않을 경우에 말이죠 (따라서 효과적으로 constant로 변환합니다).

ES6 Arrow Functions

더 읽어보기: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Arrow function은 JavaScript 환경에서함수를 생성하는 또 다른 방법입니다. 더 짧은 구문 외에도 `this` 키워드의 범위를 유지하는데 있 이점을 제공합니다 (여기를 보세요).

Arrow function 구문은 낯설게 보일 수 있으나 사실 간단합니다.

```
function callMe(name) {
    console.log(name);
}
```

또한 다음과 같이 작성할 수도 있습니다:

```
const callMe = function(name) {
    console.log(name);
}
```

이렇게 됩니다:

```
const callMe = (name) => {
    console.log(name);
}
```

중요:

arguments가 없는 경우, 함수 선언시 빈 괄호를 사용해야 합니다:

```
const callMe = () => {
    console.log('Max!');
}
```

정확히 하나의 argument가 있는 경우, 괄호를 생략할 수 있습니다:

```
const callMe = name => {
    console.log(name);
}
```

value를 return할 때, 다음과 같은 숏컷을 사용할 수 있습니다:

```
const returnMe = name => name
```

이것은 다음과 같습니다:

```
const returnMe = name => {
    return name;
}
```

Exports & Imports

React 프로젝트에서 (그리고 실제로 모든 최신 JavaScript에서), 모듈이라 불리는 여러 자바스크립트 파일들에 코드를 분할합니다. 이렇게 하면 각 file/ 모듈의 목적을 명확하게 하고 관리가 용이하게 합니다.

다른 파일의 기능에 계속 액세스하려면 `export` (available하게 하기 위해) 및 `import` 엑세스를 확보하기 위해) statements가 필요합니다.

두 가지 유형의 export가 있습니다: **default** (unnamed)와 **named** 입니다.

default => `export default ...;`

named => `export const someData = ...;`

default exports를 다음과 같이 import 할 수 있습니다.

`import someNameOfYourChoice from './path/to/file.js';`

놀랍게도, `someNameOfYourChoice` 전적으로 여러분에게 달려 있습니다.

Named exports는 이름으로 import되어야 합니다:

`import { someData } from './path/to/file.js';`

파일 하나는 오직 하나의 default와 무한한 named exports를 가질 수 있습니다. 하나의 default를 같은 파일 내에서 named exports와 믹스할 수 있습니다.

named exports를 import할 때, 다음 구문을 이용해 한 번에 모든 named exports를 import할 수 있습니다.

`import * as upToYou from './path/to/file.js';`

`upToYou` 는 모든 exported 변수/함수를 하나의 자바스크립트 객체에 모읍니다. 예를 들어, `export const someData = ...` (`/path/to/file.js`) 이와 같이 `upToYou` 에 액세스 할 수 있습니다: `upToYou.someData` .

Classes

Classes는 constructor 함수와 prototypes를 대체하는 기능입니다. 자바스크립트 객체에 blueprints를 정의할 수 있습니다.

예시:

```
class Person {
    constructor () {
        this.name = 'Max';
    }
}

const person = new Person();
console.log(person.name); // prints 'Max'
```

위의 예시에서, class뿐 만 아니라 해당 class의 property (=> `name`) 이 정의됩니다. 해당 구문은, property를 정의하는 "구식" 구문입니다. 최신 자바스크립트 프로젝트에서는 (이 코스에서 사용된 것처럼), 다음과 같은 보다 편리한 정의 방법을 사용해 class property를 정의합니다:

```
class Person {
    name = 'Max';
}

const person = new Person();
console.log(person.name); // prints 'Max'
```

메소드를 정의할 수도 있습니다. 다음과 같이 말이죠:

```
class Person {
    name = 'Max';
    printMyName () {
        console.log(this.name); // this is required to refer to the class!
    }
}

const person = new Person();
person.printMyName();
```

혹은 이와 같이 할 수도 있습니다:

```
class Person {
    name = 'Max';
    printMyName = () => {
        console.log(this.name);
    }
}

const person = new Person();
person.printMyName();
```

두 번째 접근 방식은 all arrow function과 같은 이점이 있습니다: `this`키워드가 reference를 변경하지 않습니다.

class 사용시 **inheritance**를 사용할 수도 있습니다.

```
class Human {
    species = 'human';
}

class Person extends Human {
    name = 'Max';
    printMyName = () => {
        console.log(this.name);
    }
}

const person = new Person();
person.printMyName();
console.log(person.species); // prints 'human'
```

Spread & Rest Operator

Spread 와 rest operator는 사실 같은 구문을 사용합니다: `...`

맞습니다, 연산자입니다 - 점 세개죠. 이것을 사용해 spread로 사용할지 rest operator로 사용할지 결정합니다.

Spread Operator 사용하기:

Spread operator는 배열에서 요소들을 가져오거나 (=> 배열을 요소들의 리스트로 분해) 객체에서 속성을 가져옵니다.

두 가지 예시가 있습니다:

```
const oldArray = [1, 2, 3];
const newArray = [...oldArray, 4, 5]; // This now is [1, 2, 3, 4, 5];
```

객체에 spread operator를 사용한 예시입니다:

```
const oldObject = {
    name: 'Max'
};
const newObject = {
    ...oldObject,
    age: 28
};
```

그러면 `newObject`는 다음이 될 것입니다.

```
{
    name: 'Max',
    age: 28
}
```

sperad operator는 배열과 객체를 복제하는데 매우 유용합니다. 둘 다 (primitives가 아닌) reference 유형이기 때문에, 안정적으로 복사를 하는게 어려울 수 있습니다. (복사된 원본에 future mutation 발생 방지). Spread operator로, 객체나 배열의 복사본 (shallow!)을 쉽게 얻을 수 있습니다.

Destructuring

Destructuring을 사용하면 배열이나 객체의 값에 쉽게 엑세스할 수 있고 변수에 할당할 수 있습니다.

한 배열의 예시입니다:

```
const array = [1, 2, 3];
const [a, b] = array;
```

```
console.log(a); // prints 1
console.log(b); // prints 2
console.log(array); // prints [1, 2, 3]
```

다음은 객체의 예시입니다:

```
const myObj = {
    name: 'Max',
    age: 28
}
const {name} = myObj;
console.log(name); // prints 'Max'
console.log(age); // prints undefined
console.log(myObj); // prints {name: 'Max', age: 28}
```

Destructuring은 인자를 가진 함수를 작업할 때 매우 유용합니다. 이 예시를 보시죠:

```
const printName = (personObj) => {
    console.log(personObj.name);
}
printName({name: 'Max', age: 28}); // prints 'Max'
```

여기서, 함수내 name만을 print하고 싶지만 함수에 완전한 person 객체를 보내고 있습니다. 당연히 이것은 문제가 되지 않지만 personObj.name을 이 함수내에서 호출해야만 합니다. 이 코드를 destructuring으로 압축시켜 보겠습니다.

```
const printName = ({name}) => {
    console.log(name);
}
printName({name: 'Max', age: 28}); // prints 'Max')
```

위와 동일한 결과를 얻지만 코드가 줄었습니다. Destructuring을 통해, `name` property를 가져와 `name` 이라는 이름의 변수/인수에 저장하고 함수 본문에서 사용할 수 있습니다.

JS Array functions

차세대 자바스크립트는 아니지만 중요합니다. 다음과 같은 자바스크립트 array 함수가 있습니다: `map()` , `filter()` , `reduce()` .

많은 React 개념이 (불변의 방식으로) 배열 작업에 의존하기 때문에 제가 그것들을 꽤 많이 사용하는 것을 보게 될 것입니다.

다음 페이지는 어레이 프로토타입에서 사용할 수 있는 다양한 방법에 대한 좋은 개요를 제공합니다. 필요에 따라 이를 클릭하고 지식을 리프레시할 수 있습니

다. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

이 코스에서 특히 중요한 사항은 다음과 같습니다:

- `map()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- `find()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find
- `findIndex()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex
- `filter()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter
- `reduce()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce?v=b
- `concat()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/concat?v=b
- `slice()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice
- `splice()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice