ES6 for PWA & Vue.js

개요

- ES6 의 여러가지 특징 중 Vue + PWA 앱 구현에 필요한 기능들 위주로 학습
- Fat Arrow, Const + Let, Methods, Modules 학습 및 실습

목차

ES6 란?

- ECMAScript 2015 와 동일한 용어
- 2015년은 ES5 (2009) 이래로 진행한 첫 메이저 업데이트가 승인된 해
- 최신 Front-End Framework (React, Vue 등) 에서 권고하는 언어 형식
- ES5 에 비해 문법이 간결해져서 익숙해지면 편해지는 부분이 있고, 그렇지 않고 복잡한 부분도 존재

Babel

- 구 버전 브라우저 중에서는 ES6 의 기능을 지원하지 않는 브라우저가 있으므로 transpiling 이 필요
- ES6 의 문법을 각 브라우저의 호환 가능한 ES5 로 변환하는 컴파일러

```
module: {
  loaders: [{
    test: /\.js$/,
    loader: 'babel-loader',
    query: {
      presets: ['es2015']
    }
  }]
},
```

Fat Arrow

ES5 에서 콜백으로 사용하는 함수 (인자로 함수를 받음) 의 문법을 간결화

```
// ES5
var arr = ["a", "b", "c"];
arr.forEach(function (value) {
  console.log(value); // a , b , c
});

// ES6
var arr = ["a", "b", "c"];
arr.forEach((value) => console.log(value)); // a , b , c
```

실습 #1 - Fat Arrow

아래 ES5 문법을 Fat Arrow 를 이용하여 변환해보세요. jsbin 또는 브라우저 콘솔 사용.

```
// #1
var arr = [1, 2, 3, 4];
var sum = 0;
arr.forEach(function (data) {
  sum += data;
});

// #2
$('#modal').click(function(event) {
  alert('show modal');
});
```

Const & Let VS Var

- `{ }` 단위로 변수의 범위가 제한되었음
- `const` : 한번 선언한 값에 대해서 변경 불가능 (상수 개념)
- `let` : 한번 선언한 값에 대해서 다시 선언 불가능

위 특징들에 대해서 자세히 알아보기 전에 ES5 특징 2가지를 리뷰하겠습니다.

ES5 특징 리뷰 - 변수의 Scope

- 기존 자바스크립트 (ES5) 는 `{ }` 에 제한되지 않는 변수 범위를 가짐

```
var sum = 0;
for (var i = 0; i < 5; i++) {
  sum = sum + i;
}
console.log(sum); // 10
console.log(i); // 5
```

ES5 **특징 리뷰** - Hoisting

- Hosting 이란 선언한 함수와 변수를 해석기가 가장 상단에 있는 것처럼 인식한다.
- js 컴파일러는 코드의 라인 순서와 관계 없이 함수선언식과 변수를 위한 메모리 공간을 먼저 확보한다.
- 따라서, `function a()` 와 `var` 는 코드의 최상단으로 끌어 올려진 것 (hoisted) 처럼 보인다.

```
function willBeOveridden() {
    return 10;
}

willBeOveridden(); // 5

function willBeOveridden() {
    return 5;
}
```

자바스크립트 컴파일러의 Hoisting

아래와 같은 코드를 실행할 때 어떻게 자바스크립트 컴파일러가 코드 순서를 재조정할까요?

```
var sum = 5;
sum = sum + i;

function sumAllNumbers() {
  // ...
}

var i = 10;
```

```
// #1 - 함수 선언식과 변수 선언을 hoisting
var sum;
function sumAllNumbers() {
  // ...
}
var i;

// #2 - 변수 대입 및 할당
sum = 5;
sum = sum + i;
i = 10;
```

ES6 - `{ }` 단위로 변수의 범위가 제한

```
let sum = 0;
for (let i = 0; i < 5; i++) {
  sum = sum + i;
}
console.log(sum); // 10
console.log(i); // 0
```

ES6 - `const` 로 지정한 값 변경 불가능

```
const a = 10;
a = 20; // Uncaught TypeError: Assignment to constant variable
```

ES6 - `let` 선언한 값에 대해서 다시 선언 불가능

```
let a = 10;
let a = 20; // Uncaught SyntaxError: Identifier 'a' has already been declared
```

ES6 - const, let

```
function f() {
  {
    let x;
    {
      // 새로운 블락안에 새로운 x 의 스코프가 생김
      const x = "sneaky";
      x = "foo"; // 위에 이미 const 로 x 를 선언했으므로 다시 값을 대입하면 에러 발생
    }
    // 이전 block 범위로 돌아왔기 때문에 `let x` 에 해당하는 메모리에 값을 대입
    x = "bar";
    let x = "inner"; // Uncaught SyntaxError: Identifier 'x' has already been de
  }
}
```

실습 #2 - Const, Let

아래 코드에서 ES5 의 non-block-scope 로 인해 발생되는 문제점들을 찾아
ES6 의 변수로 선언하여 수정해보세요.

```
let fw = ["jquery", "angular", "react", "vue"],
    result = "";

// log all frameworks
for (var i = 0; i < fw.length; i++) {
  result = result + fw[i] + " ";
}

// ...

// add version number to frameworks
while (i < fw.length) {
  fw[i] += "1.0";
  i++;
}
```

실습 #2 - Const, Let

1. 아래 코드를 돌리기 전에 오류가 예상되는 지점을 한번 찾아보세요 (2군데)
2. 어떤 오류가 발생하였는지 해당 라인의 맨 우측에 주석으로 오류를 적어보세요.
3. let 과 const 를 var 로 변환해서 돌려보고 어떤 차이점이 있는지 확인해보세요.

```
function f() {
  {
    let x;
    {
      const x = "sneaky";
      x = "foo";
    }
    x = "bar";
    let x = "inner";
  }
}
```

Methods

- 기존 선언 방식에서 `: function` 을 제외하고 선언 가능

```
var dictionary = {
  words: 100,
  // ES5
  lookup: function() {
    console.log("find words");
  },
  // ES6
  lookup() {
    console.log("find words");
  }
};
```

Modules

- 자바스크립트 모듈 로더 라이브러리 (AMD, Commons JS) 기능을 js 언어 자체에서 지원
- 호출되기 전까지는 코드 실행과 동작을 하지 않는 특징이 있음

```
// libs/math.js
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;

// main.js
import {sum} from 'libs/math.js';
sum(1, 2);
```

- Vue.js 에서 마주칠 `default` export

```
// util.js
export default function (x) {
  return console.log(x);
}

// main.js
import util from `util.js`;
console.log(util); // function (x) { return console.log(x); }
util("hi");

// app.js
import log from 'util.js';
console.log(log);
log("hi");
```

실습 #3 - Module loading

두 숫자를 합하는 함수를 export 하는 `math.js` 와
해당 함수를 import 하여 사용하는 `app.js` 를 구현하세요.

실습절차

1. webpack 기본 프로젝트 구성
2. babel 플러그인 [설치] : `babel-loader babel-core babel-preset-es2015`
3. `webpack.config.js` 파일 설정
4. `math.js` 구현
5. `app.js` 구현
6. `index.html` 에서 로딩 및 동작 확인

참고

- Babel
- Modules, default export
- export, MDN

끝