12월 1일 (수)

1. Codecademy JavaScript - Modules

Article 1. Introduction to JavaScript Runtime Environments

- **What is a Runtime Environment?**

런타임 환경 이란? 너의 프로그램이 실행되는 곳을 말한다.

이번 아티클에서는 두 가지 환경을 다룰 것이다.

- 1 - 브라우저 런타임 환경 (like Chrome, Firefox)

- 2 - 노드 런타임 환경

- **A Browser's Runtime Environment**

자바스크립트 코드가 실행되는 곳은 대부분 브라우저이다. 브라우저에서 실행되고 브라우저을 위해 생성된 어플리케이션은 프론트엔드 어플리케이션이라고 불린다. 오랫동안 자바스크립트는 오직 브라우저에서만 실행되었다. 벡엔드 어플리케이션을 위해서는 자바나 php 를 써야만 했다. 하지만 Node.js 가 생긴 이후로는 백엔드에서도 자바스크립트 코드를 사용할 수 있게 되었다. 브라우저 런타임 환경은 오직 window 객체에만 접근이 가능하다. window 객체는 자바스크립트의 모든 객체를 담고 있는 가장위에 존재하는 부모객체와 같다.

- **The Node Runtime Environment**

2009년 노드 런타임 환경이 개발되었다. 이는 자바스크립트 코드가 브라우저 없이 실행시킬 수 있는 기능을 갖게 된 것이다.

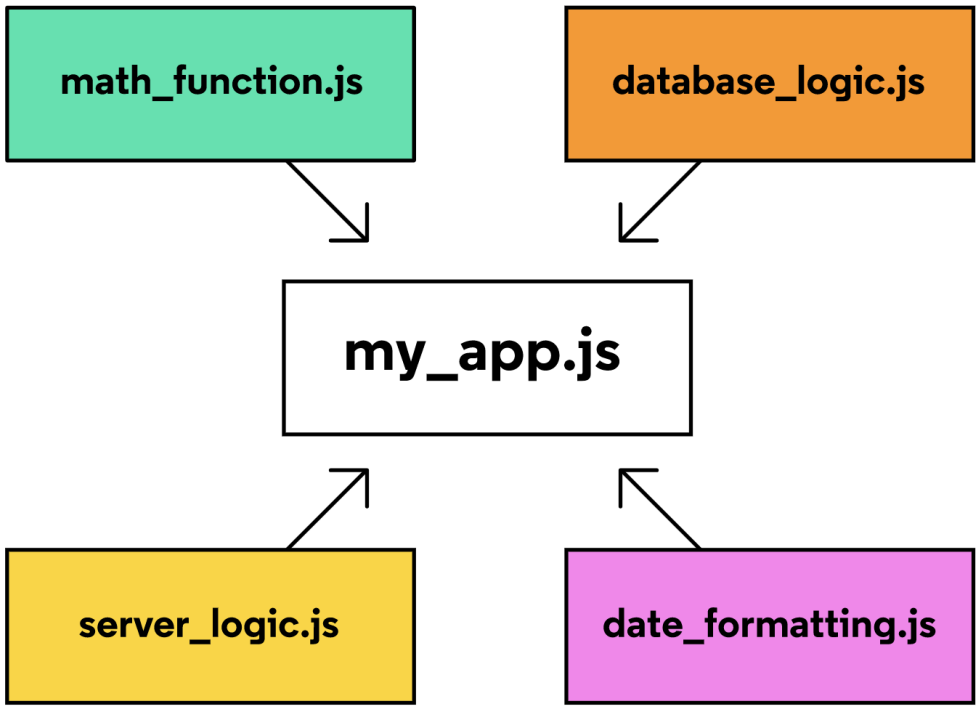
노드 런타임 환경과 브라우저 런타임 환경은 완전히 다르다. 그러므로 브라우저 런타임의 데이터 값과 함수는 사용할 수 없다. 대신에 노드 런타임 환경은 즉, Node.js 는 브라우저에서 불가능한 서버 파일시스템, 데이터베이스 , 네트워크를 다루는 어플리케이션을 개발할 수 있다.

이를 사용하기 위해서 노드js를 설치하고 터미널에서 실행해야한다.

Article 2. Implementing Modules in Node

- **What are Modules?**

모듈은 재사용할 수 있는 하나의 파일 안에 있는 코드의 모음이다. 다른 사용자에게 공유할 수도 있고 받아 올 수도 있다. 모듈러 프로그램은 복잡한 시스템을 만들기 위해 컴포넌트가 각각 분리되고, 사용되고 재조합되는 것이다.



모듈과 파일, 패키지는 모두 같은 뜻이다.

my_app.js 안에 각각 기능들을 하는 다른 js 파일 즉, 모듈들이 모두 들어가서 조화를 이루고 있다.

모듈의 장점:

⇒ 디버그 코드를 발견하고 고치는데 훨씬 더 쉬워진다.

⇒ 다른 어플리케이션에 정의된 로직의 재사용과 재활용이 가능하다.

⇒ 다른 모듈로부터 정보를 유지하고 보호받는다.

⇒ global namespace 의 오염과 잠재적인 네이밍 충돌을 예방한다.

- **Implementations of Modules in JavaScript: Node.js vs ES6 (모듈 구현하기)**

런타임 환경에 의존하는 모듈 구현에는 다양한 방법이 있다. 자바스크립트에서는 두 가지 방법있다.

⇒ 1. 노드에서는 module.exports 그리고 require( ) 문법

⇒ 2. 브라우저 ES6 는 import, export 문법

- **Implementing Modules in Node.js (노드에서 모듈 구현하기)**

노드에서 실행되는 모든 자바스크립트 파일은 ㄷ분명한 모듈로서 대한다. 각각 모듈에 정의된 함수나 데이터는 다른 모듈에 사용될 수 있다.

- **module.exports 객체**

```
/* converters.js */
function celsiusToFahrenheit(celsius) {
  return celsius * (9/5) + 32;
}

module.exports.celsiusToFahrenheit = celsiusToFahrenheit;

module.exports.fahrenheitToCelsius = function(fahrenheit) {
  return (fahrenheit - 32) * (5/9);
};
```

위와 같이 module.exports 객체 내에 메소드로써 celsiusToFahrenheit 을 넣었다. 이로써 다른 모듈에서 이 함수를 사용할 수 있게 된다. module.exports 는 노드js 런타임 환경 빌트인 객체이다.

- **require( )**

이 함수는 인자를 스트링으로 받는다. 스트링은 내가 import 하고 싶은 모듈의 file path 를 입력해야한다.

```
/* water-limits.js */
const converters = require('./converters.js');

const freezingPointC = 0;
const boilingPointC = 100;

const freezingPointF = converters.celsiusToFahrenheit(freezingPointC);
const boilingPointF = converters.celsiusToFahrenheit(boilingPointC);

console.log(`The freezing point of water in Fahrenheit is ${freezingPointF}`);
console.log(`The boiling point of water in Fahrenheit is ${boilingPointF}`);
```

converters.js 의 함수를 water-limits.js 에 require( ) 해서 사용할 수 있다.

' ./ ' 는 converters.js 는 water-limits 와 같은 폴더에 저장되어 있다는 것을 의미한다. require( )을 이용할 때, 전체의 module.exports 객체는 converters 변수에 반환되고 저장된다. 한마디로 converters 도 객체이며 그 안에 받아온 메서드를 포함한다.

이것은 .celsiusToFahrenheit() 와 .fahrenheitToCelsius() 메서드 모두 이 프로그램에서 사용될 수 있다는 것이다.

- **Using Object Destructuring to be more Selective With require( )**

모듈은 많은 함수들을 export 하곤한다. 이럴 경우 너는 필요한 것 한 두개만 뽑아서 export 할 수 있다. 객체 디스트럭쳐링을 사용하는 것이다 .

```
/* celsius-to-fahrenheit.js */
const { celsiusToFahrenheit } = require('./converters.js');

const celsiusInput = process.argv[2];
const fahrenheitValue = celsiusToFahrenheit(input);

console.log(`${celsiusInput} degrees Celsius = ${fahrenheitValue} degrees
Fahrenheit`);
```

.celsiusToFahrenheit() 만 require 하고 .fahrenheitToCelsius() 는 남긴다.

- **실습**

```
/* shape-area.js */
const PI = Math.PI;

// Define and export circleArea() and squareArea() below
const circleArea = (radiusLength) => {return PI * radiusLength * radiusLength;}

const squareArea = sideLength => {return sideLength * sideLength;}

// 함수들 export 하기
module.exports.circleArea = circleArea;

module.exports.squareArea = squareArea;
```

모듈 export 과정

```
1   /* app.js */
2
3   const radius = 5;
4   const sideLength = 10;
5
6   // Option 1: shape-area.js 의 모든 모듈을 import
    하기
7   const areaFunctions = require('./shape-area.
    js');
8
9   // Option 2: object destructuring 을 사용해서
    원하는 함수 circleArea , squareArea 만 import 하기
10
11  const { circleArea, squareArea } = require("./
    shape-area.js")
12
```

```
12
13  // 임포트된 .circleArea() 와 .squareArea()
    메서드 사용하기
14  const areaOfCircle = areaFunctions.
    circleArea(radius);
15
16  const areaOfSquare = areaFunctions.
    squareArea(sideLength);
17
```

- **Review**

이번 아티클에서는

⇒ 모듈 프로그램 구현의 이점들

⇒ Node.js 의 module.exports 객체 사용법

⇒ Node.js 의 require( ) 메서드 사용법

⇒ 원하는 컴포넌트만 불러오는 객체 디스트럭쳐링 사용법

Article 3. Implementing Modules using ES6 Syntax

- **Implementing Modules in the Browser**

브라우저에서의 모듈을 사용하면 코드의 재사용성이 늘 수 있다.

기존에 있는 함수만 변경하면 쓰이고 있는 모든 곳이 자동으로 업데이트 된다. 유지보수에 아주 좋다!

```
<!-- secret-messages.html -->
<html>
  <head>
    <title>Secret Messages</title>
  </head>
  <body>
    <button id="secret-button"> Press me... if you dare </button>
    <p id="secret-p" style="display: none"> Modules are fancy! </p>
    <script src="./secret-messages.js"> </script>
  </body>
</html>
```

```
/* secret-messages.js */
const buttonElement = document.getElementById('secret-button');
const pElement = document.getElementById('secret-p');

const toggleHiddenElement = (domElement) => {
    if (domElement.style.display === 'none') {
      domElement.style.display = 'block';
    } else {
      domElement.style.display = 'none';
    }
}

buttonElement.addEventListener('click', () => {
  toggleHiddenElement(pElement);
});
```

secret-messages.js 의 toggleHiddenElement 메서드를 export 한다.

```html
<!-- secret-image.html -->
<html>
  <head>
    <title>Secret Image</title>
  </head>
  <body>
    <button id="secret-button"> Want to see something cool? </button>
    <img id="secret-img" src="imageURL.jpg" style="display: none">
    <script src="./secret-image.js"> </script>
  </body>
</html>
```

.. and the JavaScript might look like this:

```javascript
/* secret-image.js */
const buttonElement = document.getElementById('secret-button');
const imgElement = document.getElementById('secret-img');

const toggleHiddenElement = (domElement) => {
    if (domElement.style.display === 'none') {
      domElement.style.display = 'block';
    } else {
      domElement.style.display = 'none';
    }
}

buttonElement.addEventListener('click', () => {
  toggleHiddenElement(imgElement);
});
```

export 된 모듈 toggle 메서드를 secret-imge.js 에서도 사용이 가능하다.

- **ES6 Named Export Syntax**

먼저 파일시스템 안 어디에 새로운 모듈이 포함될 지 고려해보자. 여러 프로젝트에 쓰일 모듈은 상호작용할 수 있는 장소에 넣어야한다.

```
secret-image/
|-- secret-image.html
|-- secret-image.js
secret-messages/
|-- secret-messages.html
|-- secret-messages.js
modules/
|-- dom-functions.js    <-- new module file
```

```
export { resourceToExportA, resourceToExportB, ...}
```

export Syntax 의 문법이다.

```
/* dom-functions.js */
const toggleHiddenElement = (domElement) => {
    if (domElement.style.display === 'none') {
      domElement.style.display = 'block';
    } else {
      domElement.style.display = 'none';
    }
}

const changeToFunkyColor = (domElement) => {
  const r = Math.random() * 255;
  const g = Math.random() * 255;
  const b = Math.random() * 255;

  domElement.style.background = `rgb(${r}, ${g}, ${b})`;
}

export { toggleHiddenElement, changeToFunkyColor };
```

export 할 파일에서 메서드를 export 하는 예시이다. 이제 import 로 해당 함수들을 사용할 수 있다.

```
/* dom-functions.js */
export const toggleHiddenElement = (domElement) => {
  // logic omitted...
}

export const changeToFunkyColor = (domElement) => {
  // logic omitted...
}
```

변수도 또한 expert가 가능하다.

- **ES6 Import Syntax**

export 한 모듈을 불러오는 것 또한 비슷한 문법을 지닌다.

```
import { exportedResourceA, exportedResourceB } from '/path/to/module.js';
```

```
/* secret-messages.js */
import { toggleHiddenElement, changeToFunkyColor } from '../modules/dom-
functions.js';

const buttonElement = document.getElementById('secret-button');
const pElement = document.getElementById('secret-p');

buttonElement.addEventListener('click', () => {
  toggleHiddenElement(pElement);
  changeToFunkyColor(buttonElement);
});
```

```
<!-- secret-messages.html -->
<html>
  <head>
    <title>Secret Messages</title>
  </head>
  <body>
    <button id="secret-button"> Press me... if you dare </button>
    <p id="secret-p" style="display: none"> Modules are fancy! </p>
    <script type="module" src="./secret-messages.js"> </script>
  </body>
</html>
```

모듈을 import 하게 되면 HTML 의 script 태그에 type="module" 을 무조건 써줘야한다!!

- **실습**

```
/* main.js */
import {changeText, changeToFunkyColor} from './module.js';
// import the functions here, then uncomment the code below

const header = document.getElementById("header");

// call changeText here
changeText(header, 'I did it!');
setInterval(()=> {
  changeToFunkyColor(header);
  // call changeToFunkyColor() here

}, 200);
```

Run

I did it!

- **Renaming Imports to Avoid Naming Collisions**

```
/* inside greeterEspanol.js */
const greet = () => {
  console.log('hola');
}
export { greet };

/* inside greeterFrancais.js */
const greet = () => {
  console.log('bonjour');
}
export { greet };
```

위와 같이 모듈 이름이 겹칠 수도 있다.

```
import { greet } from 'greeterEspanol.js';
import { greet } from 'greeterFrancais.js';
```

The code above will throw an error on line 2 due to the fact that the identifier `greet` has already been defined on line 1. Thankfully, ES6 includes syntax for renaming imported resources by adding in the keyword `as`. It looks like this:

```
import { exportedResource as newlyNamedResource } from '/path/to/module'
```

as 키워드를 이용하여 모듈의 이름을 바꾸자 !

```
/* main.js */
import { greet as greetEspanol } from 'greeterEspanol.js';
import { greet as greetFrancais } from 'greeterFrancais.js';

greetEspanol(); // Prints: hola
greetFrancais(); // Prints: bonjour
```

- **Default Exports and Imports**

지금까지 네이밍된 export 만 진행했지만 이번엔 default export 에 대해서 알아보자. 항상 은 아니지만 종종 default export 는 여러 함수나 데이터를 포함한

객체에 많이 쓴다.

```
const resources = {
  valueA,
  valueB
}
export { resources as default };
```

```
const resources = {
  valueA,
  valueB
}
export default resources;
```

export default 속기법도 쓸 수 있다.

- **Importing default values**

디폴트 모듈을 불러오자.

```
import importedResources from 'module.js';
```

디폴트 값 import에서는 { } 가 없어도 된다. 하지만 import { default as importedResources } from~ 으로 써도 된다.

```
// This will work...
import resources from 'module.js'
const { valueA, valueB } = resources;

// This will not work...
import { valueA, valueB } from 'module.js'
```

디폴트 객체는 객체내 밸류를 바로 부를 수 없다. import 후에 변수 지정을 해야한다.

```
/* dom-functions.js */
const toggleHiddenElement = (domElement) => {
    if (domElement.style.display === 'none') {
      domElement.style.display = 'block';
    } else {
      domElement.style.display = 'none';
    }
}

const changeToFunkyColor = (domElement) => {
  const r = Math.random() * 255;
  const g = Math.random() * 255;
  const b = Math.random() * 255;

  domElement.style.background = `rgb(${r}, ${g}, ${b})`;
}

const resources = {
  toggleHiddenElement,
  changeToFunkyColor
}
export default resources;
```

```
import domFunctions from '../modules/dom-functions.js';

const { toggleHiddenElement, changeToFunkyColor } = domFunctions;

const buttonElement = document.getElementById('secret-button');
const pElement = document.getElementById('secret-p');

buttonElement.addEventListener('click', () => {
  toggleHiddenElement(pElement);
  changeToFunkyColor(buttonElement);
});
```

객체 디스트럭쳐링으로 resource 디폴트 내 값의 메소드를 불러왔다.

- **Review**

⇒ 모듈러 프로그램 구현의 이점들

⇒ ES6 export, import 문 사용법

⇒ 리네이밍 as 키워드

⇒ 디폴트 밸류 export, import 하기

Article 4. Module Reference

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

Article 5. What are CJS, AMD, UMD, and ESM in JavaScript ?

모듈을 활용하기 위한 다양한 방법을 배워보자

https://dev.to/iggredible/what-the-heck-are-cjs-amd-umd-and-esm-ikm

초기에는 자바스트립트는 import , export 모듈을 가지고 있지 않았다. 이것은 큰 문제로 하나의 파일에 너의 앱에 대한 코드를 작성한다고 생각해봐라 너무 끔찍할 것이다.

- CJS (Common JS)

```
//importing
const doSomething = require('./doSomething.js');

//exporting
module.exports = function doSomething(n) {
  // do something
}
```

⇒ 우리가 node.js 에서 보통 사용하는 형식이다.

⇒ CJS 는 모듈을 동기적으로 임포트한다.

⇒ node_modules 라이브러리 나 local dir 에서 임포트할 수 있다.

⇒ CJS 임포트를 할 때, 임포트된 객체를 카피해서 너에게 전해줄 것이다.

⇒ CJS 는 브라우저에서 작동하지 않는다. 그것은 트랜스파일, 번들이 되어야한다.

- AMD (Asynchronous Module Definition 비동기적 모듈 정의)

```
define(['dep1', 'dep2'], function (dep1, dep2) {
    //Define the module value by returning a value.
    return function () {};
});
```

or

```
// "simplified CommonJS wrapping" https://requirejs.org/docs/whyamd.
define(function (require) {
    var dep1 = require('dep1'),
        dep2 = require('dep2');
    return function () {};
});
```

⇒ AMD 는 모듈을 비동기적으로 임포트한다.

⇒ AMD 는 프론트엔드를 위해 제작되었다. (CJS 는 백엔드용)

⇒ AMD 문법은 CJS 보다 덜 직관적이다.

- UMD (Universal Module Definition 보편적인 모듈 정의)

```
(function (root, factory) {
    if (typeof define === "function" && define.amd) {
        define(["jquery", "underscore"], factory);
    } else if (typeof exports === "object") {
        module.exports = factory(require("jquery"), require("undersc
    } else {
        root.Requester = factory(root.$, root._);
    }
}(this, function ($, _) {
    // this is where I defined my module implementation

    var Requester = { // ... };

    return Requester;
}));
```

⇒ 프론트와 백엔드 모두 작동한다.

⇒ CJS or AMD 와 달리, UMD는 여러 모듈 시스템을 설계하기 위한 하나의 패턴과 같다.

⇒ UMD 는 Rollup이나 Webpack 과 같은 번들러를 사용할 때, 보통 하나의 대체 모듈로서 사용된다. (ESM이 작동하지 않을 때 대체품으로 쓴다.)

- ESM (ES Modules)

```
import React from 'react';
```

Other sightings in the wild:

```
import {foo, bar} from './myLib';

...

export default function() {
  // your Function
};
export const function1() {...};
export const function2() {...};
```

ESM은 표준 모듈 시스템을 구성하는 js 의 제안이다. 아마 익숙할 것이다.

⇒ 많은 현대의 브라우저에서 작동한다.

⇒ 두 세계(프론트, 백)에서 가장 베스트이다. 즉, CJS와 같은 간단한 문법과 AMD의 비동기성을 가지고 있다.

⇒ ES6의 정적 모듈구조 때문에 Tree-shakeable 특성을 가지고 있다.

⇒ ESM 은 Rollup 과 같은 번들러가 불필요한 코드를 제거할 수 있도록 하며, 사이트가 적은 코드를 통해 더 빠르게 로드될 수 있도록 한다.

⇒ HTML 에서 아래와 같이 불러낼 수 있다.

```
<script type="module">
  import {func1} from 'my-lib';

  func1();
</script>
```

2. Codecademy Front End Course - Semantic HTML

- **Introduction to Semactic HTML**

의미를 가진 태그들을 사용하는 것

시맨틱 html 은 SEO 를 증가시킨다. (Search Engine Optimization)

이해하기 쉬워진다.

접근성이 좋아진다.