02. script async & defer

1. WEB APIs

☞ Console API는 WEB API 중의 하나이다.

☞ WEB APIs는 JavaScript 언어 자체에 포함된 것이 아니라 Browser 가 제공하는(이해할 수 있는) 함수들이다.

2. Dev Tools

☞ Elements

☞ Console

☞ Sources

☞ Network

☞ JavaScript의 공식 사이트 => ecma-international.org

☞ 개발자들이 자주가는 사이트 => developer.mozilla.org

3. async vs defer

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="main.js"></script>
</head>

<body>
</body>
</html>
```

~~~~~~

☞ parsing HTML

fetching js executing js

parsing HTML <--page is ready

✓ <head> 안에 script를 포함하게 되면 사용자가 html 파일을 다운로드 받았을 때 브라우저가 한 줄 한줄 분석한다.

✓ 이렇게 분석한 것을 CSS와 병합하여 DOM elements 로 변환하게 된다.

✓ 단점으로는 만약 js 파일의 사이즈가 크고 인터넷 속도도 느릴 경우에는 사용자가 웹 사이트를 보는데 까지 많은 시간이 소요가 된다.

✓ 따라서 script를 <head>에 포함하는 것은 좋은 것이 아니다.
~~~~~~

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>

  <script src="main.js"></script>
</body>
</html>
```

~~~~~~

☞ parsing HTML <--page is ready
☞ fetching js executing js

✓ 다음으로 <body> 끝부분에 script를 포함하게 되면 브라우저가 html 파일을 다운로드 받아서 파싱 후 페이지가 준비가 된 다음에 script를 서버에서 받아오고 실행하게 된다.
✓ 따라서 페이지가 사용자들에 js 파일을 받기 전에도 이미 준비가 되어서 사용자가 벌써 콘텐츠를 볼 수가 있다.
✓ 그러나 단점으로는 사용자가 기본적인 html 콘텐츠를 빨리 본다는 장점은 있지만 만약 우리의 웹사이트가 JavaScript의 많이 의존적이라면, 즉 다시말하면 사용자가 의미있는 콘텐츠를 보기 위해서는 JavaScript를 이용하여 Server에 있는 데이터를 받아 온다든지 아니면 DOM 요소를 이쁘게 꾸면준다던지 하는 그런식으로 동작하는 웹 사이트라면 사용자가 정상적인 페이지를 보기 전까지는 Fetching 하는 즉, Server에서 JavaScript를 받아오는 시간도 기다려야 하고 싫어하는 시간도 기다려야 한다.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script async src="main.js"></script>
</head>
<body>
  <div></div>
</body>
</html>
```

~~~~~~

☞ parsing HTML
  fetching js executing js
  parsing HTML <--page is ready

✓ 다음으로 <head> 안에 <script>를 이용하되 async라는 속성 값을 사용하는 것이다.
✓ async는 boolean 타입의 속성값이기 때문에 선언하는 것만으로도 true로 설정되어 async 옵션

을 사용할 수 있다.

- ✓ async를 사용하게 되면 브라우저가 html을 다운로드 받아서 parsing을 하다가 async를 만나면 병렬로 js 파일을 다운로드 받자라는 명령만 한 후 다시 parsing하다가 js 파일이 다운로드가 완료가 되면 그 때 parsing하는 것을 멈추고 다운로드한 js 파일을 실행하게 된다.
- ✓ 실행을 다하고 나서 나머지 html을 parsing하게 된다.
- ✓ 장점으로는 <body> 끝에 사용하는 것보다는 fetching이 parsing하는 동안 병렬적으로 일어나기 때문에 다운로드 받는 시간을 절약할 수 있다.
- ✓ 그러나 JavaScript가 html이 parsing 되기도 전에 실행이되기 때문에 만약 JavaScript 파일에서 querySelector()를 이용하여 DOM 요소를 조작한다면 조작할려는 시점에 html이 우리가 원하는 요소가 아직 정의되어 있지 않을 수 있기 때문에 이 부분이 조금 위험할 수도 있다.
- ✓ 또 하나는 html이 parsing 하는 동안에 언제든지 JavaScript를 실행하기 위하여 멈출 수 있기 때문에 사용자가 페이지를 보는데 시간이 여전히 조금 더 걸릴 수 있다.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script defer src="main.js"></script>
</head>
<body>
  <div></div>
</body>
</html>
```

~~~~~~

☞ parsing HTML ----------------->
  fetching js <--page is ready
                executing js

- ✓ 다음으로 <head> 안에 <script>를 이용하되 defer라는 속성 값을 사용하는 것이다.
- ✓ 이 옵션은 parsing을 하다가 defer를 만나면 js 파일을 다운로드 받자는 명령만 실행시켜 놓고 나머지 html을 끝까지 parsing 하게 된다.
- ✓ 마지막에 parsing이 끝난 다음에 다운로드 되어진 JavaScript를 실행하게 된다.
- ✓ defer가 가장 좋은 옵션이다.

☞ 마지막으로 JsvaScript를 이용할 때는 제일 먼저 'use strict';를 제일 윗 부분에 정의해 주면 좋다.

☞ TypeScript를 사용하 때는 전혀 선언할 필요가 없지만 순수 Vanila JavaScript에서는 꼭 사용하도록 한다.

☞ 이유는 Brendan Eich가 JavaScript를 만들 때 빨리 만들었기 때문에 이때 JavaScript는 아주 유연한 언어로 만들어 졌다.
~~~~~~

- ☞ 그러나 이 유연함 때문에 개발자들이 많은 실수를 할 수 있기 때문에 아주 위험한 언어로 되었다.
- ☞ JavaScript에서는 선언되지 않는 변수에 값을 할당한다든지 아니면 기존의 프로토타입을 변경한다든지 이러한 비상식적인 것들을 다른 언어를 배운 개발자들이 봤을 때에는 아주 비상식적으로 보인다.
- ☞ 따라서 'use strict';는 ECMASCript 5에서 추가되었다.
- ☞ 이렇게 선언하게 되면 더 이상 비상식적인 것을 사용할 수 없게 된다.