

7일 (수)

1. Codecademy JavaScript - Class 실습 - Build a Library

```
class Media {
  constructor(title) {
    this._title = title;
    this._isCheckedOut = false;
    this._ratings = [];
  }

  get title() {
    return this._title;
  }

  get isCheckedOut() {
    return this._isCheckedOut;
  }
  set isCheckedOut(newStatus) {
    this._isCheckedOut = newStatus;
  }

  get ratings() {
    return this._ratings;
  }

  toggleCheckOutStatus() {
    if (this._isCheckedOut === false) {
      this._isCheckedOut = true;
    }
    else {
      this._isCheckedOut = false;
    }
  }

  getAverageRating() {
    const resultSum = this._ratings.reduce((currentSum, rating) => {
      return currentSum + rating
    }, 0);
    const arrayLength = this._ratings.length;

    return Math.round(resultSum / arrayLength);
  }

  addRating(rating) {
    this._ratings.push(rating)
  }
}
```

```

class Book extends Media {
  constructor(title, author, pages) {
    super(title);
    this._author = author;
    this._pages = pages;
  }

  get author() {
    return this._author;
  }

  get pages() {
    return this._pages;
  }
}

class Movie extends Media {
  constructor(title, director, runTime) {
    super(title);
    this._director = director;
    this._runTime = runTime;
  }

  get director() {
    return this._author;
  }

  get pages() {
    return this._runTime;
  }
}

const historyOfEverything = new Book('A Short History of Nearly Everything', 'Bill Bryson', 544);

historyOfEverything.toggleCheckOutStatus();

console.log(historyOfEverything.isCheckedOut);

historyOfEverything.addRating(4);
historyOfEverything.addRating(5);
historyOfEverything.addRating(5);

console.log(historyOfEverything.getAverageRating());

const speed = new Movie('Speed', 'Jan de Bont', 116);

speed.toggleCheckOutStatus();

console.log(speed.isCheckedOut);

speed.addRating(1);
speed.addRating(1);
speed.addRating(5);

console.log(speed.getAverageRating());

```

<https://www.codecademy.com/courses/introduction-to-javascript/projects/build-a-library>

2. Codecademy JavaScript - Browser Compatibility and Transpilation (브라우저 호환과 트랜스파일레이션)

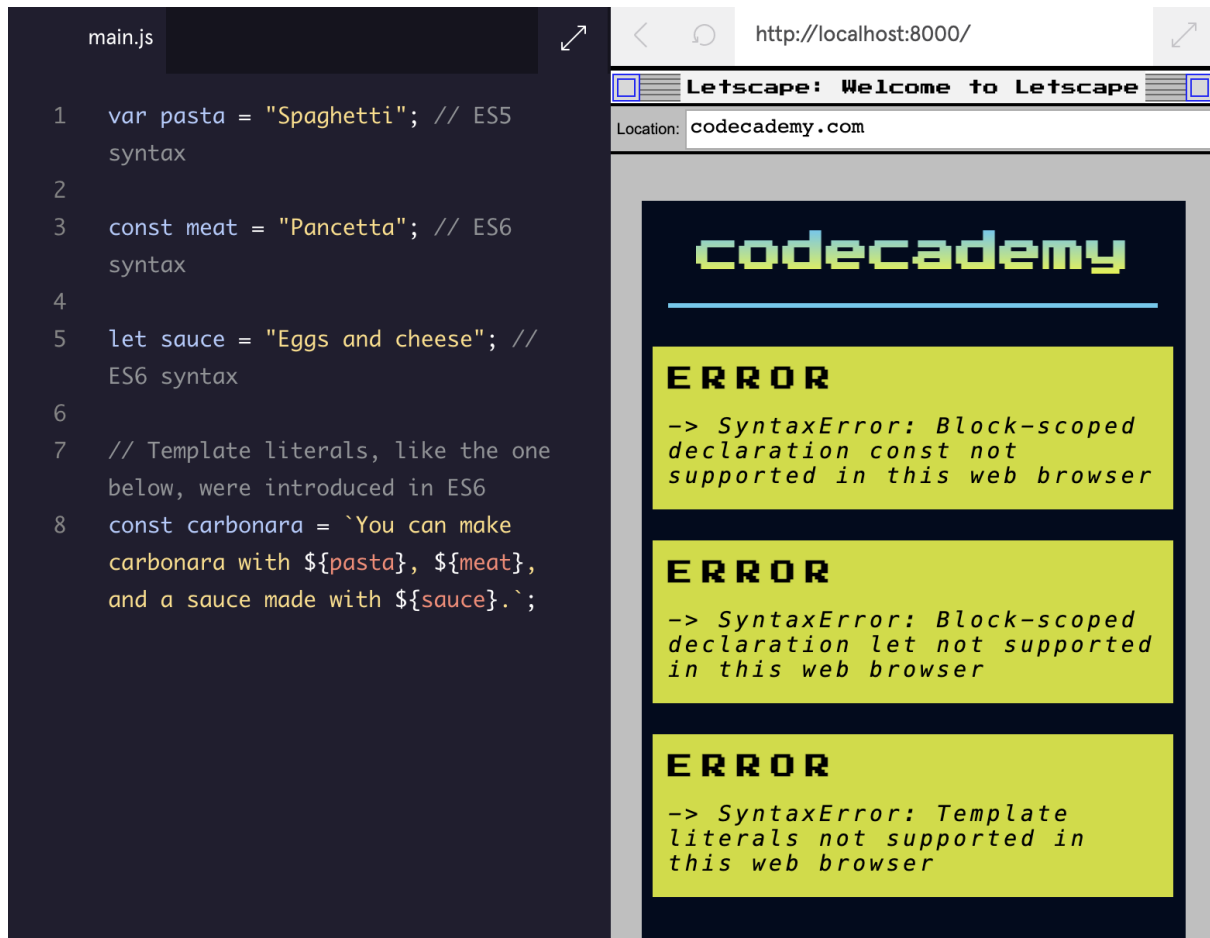
트랜스파일레이션은 특정 언어로 작성된 코드를 비슷한 다른 언어로 변환시키는 것을 말한다. 이는 모든 브라우저가 ES6+ 의 기능을 제공하지 않기때문에 이를 ES5로 변환시키는 과정이 필요하기 때문이다. ES5 는 거의 모든 브라우저에서 호환이 된다.

바벨(Babel) 이나 타입스크립트를 변환하는 타입스크립트 트랜스파일러가 있다.

- **Introduction**

ES6 를 지원하지 않는 여러 브라우저에서 호환을 시키기 어렵다. 개발자들은 브라우저 호환의 이슈를 자주 겪음 그를 해결하기 위해 트랜스파일러를 도입했다. 이번 레슨에서는 브라우저 호환 이슈를 다룰 중요한 두가지 툴을 배워볼 것이다. (tools: caniuse.com and Babel)

바벨은 자바스크립트 라이브러리이다.



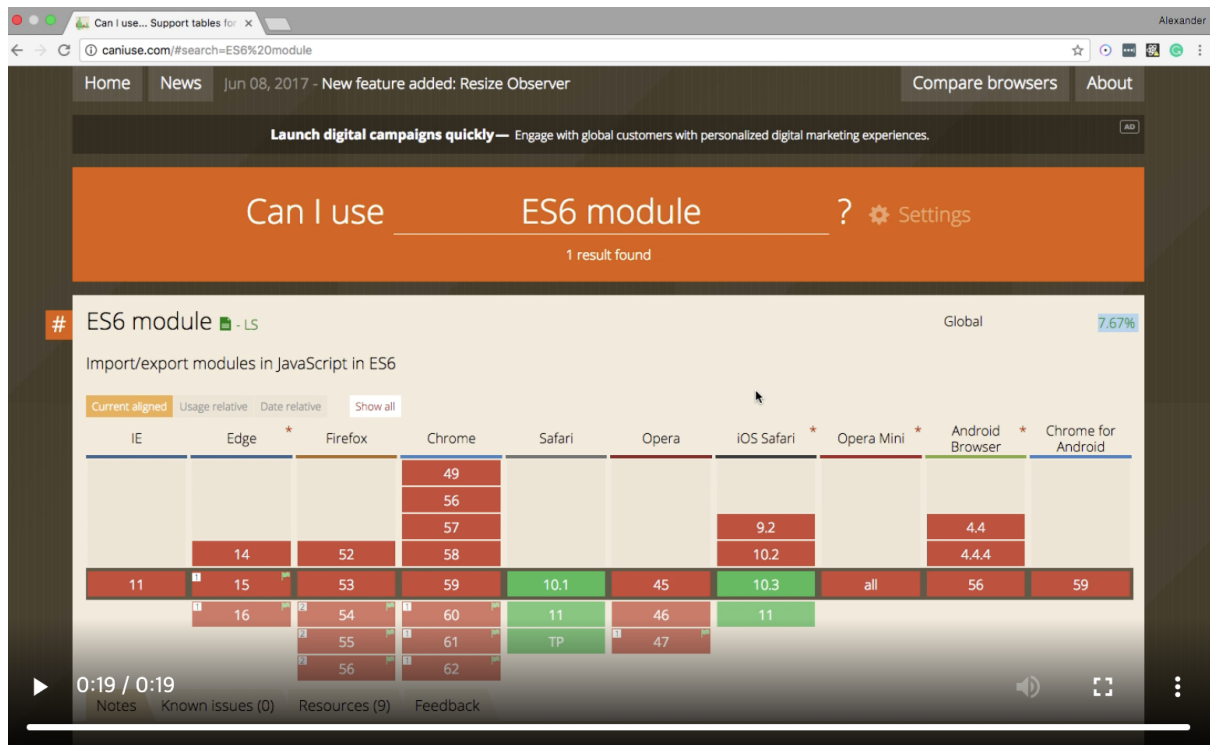
const, let, \${ } 의 사용으로 ES6 를 지원하지 않는 브라우저 호환에 이슈가 생겼다.

• caniuse 닷컴 - 1 -

호환 문제 :

1. 몇몇의 유저들이 브라우저 업데이트를 안했다.
2. ES6 의 모듈과 같은 여러 최신 기능들은 대부분의 웹브라우저에서 지원이 안된다. - 익스플로러와 같은 것

caniuse 에서 ES6에서 지원하는 키워드들이 어떤 브라우저에서 쓸 수 있는지 확인이 가능하다.



• Why ES6?

먼저 변환하는 법을 알기 전에 ECMA 가 상당한 업데이트를 진행한 이유에 대해 이해하는 것이 도움이 될 것이다.

ES5 → ES6 로 업데이트한 세가지 이유

- ⇒ for 코드 가독성과 코드 효율성 증가
- ⇒ for syntax 버그를 완화시키기 위해
- ⇒ for 다른 OOP 언어와 유사함으로 마찰을 줄이기 위해

• 트랜스파일레이션 with Babel

자동으로 코드를 바꿔주는 Babel 이다.

- ✓ 1. In the terminal window type:

```
npm install babel-cli
```

This installs one of the two required Babel packages.

- ✓ 2. In the terminal window type:

```
npm install babel-preset-env
```

This installs the second of two required Babel packages.

- ✓ 3. In the terminal, type `npm run build` and press enter.

You can view the ES5 code in **`./lib/main.js`**.

You may need to refresh to see the newly created **lib** directory.

babel 설정 방법??

- **npm init**

첫번째 단계는 src 라는 디렉토리 안에 변경할 js file을 넣는 것이다. npm 은 Node Package Manager 이다.

바벨을 설치하기전 먼저 npm 을 사용해야 한다. node package 는 다른 개발자들이 작성한 코드가 들어있는 디렉토리이다. 이러한 패키지를 사용하면 중복코드나 버그를 피할 수 있다.

우리의 프로젝트 디렉토리에 바벨을 추가하기전 먼저 npm init을 실행해야한다. npm init 은 package.json 파일을 root 디렉토리에 생성한다.

package.json 파일은 현재 자바스크립트 프로젝트 파일에 대한 정보와 다음 세가지를 포함한다.

- ⇒ 메타데이터 : 프로젝트 제목, 설명, 작성자, 등등
- ⇒ 프로젝트에 요구된 node packages 의 리스트 : 만약 다른 개발자가 너의 프로젝트를 실행하려고 하면 npm 은 package.json 의 내부를 참조하고 다운로드한다.
- ⇒ 커맨드 라인 스크립트의 키와 밸류쌍 : 이 속기 스크립트를 실행하기 위해 npm을 사용할 수 있다.

만약 노드를 너의 컴퓨터에 설치했다면, 너는 npm init 을 터미널에 타이핑하면 package.json 파일을 생성할 수있다.

터미널은 프로젝트의 메타데이터에 대한 정보를 기입하라고 유발할 것이다. 안써도 되지만 최소한 제목과 설명은 쓰는 것을 추천한다.

After you run `npm init` your directory structure will contain the following files and folders:

```
project
|_ src
|__ main.js
|_ package.json
```

npm adds the **package.json** file to the same level as the **src** directory.

실행 후 결과이다.


```
$ ls
src
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible
defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (learn-javascript-transpilation-bct-npm-init) learning-Babel
Sorry, name can no longer contain capital letters.
name: (learn-javascript-transpilation-bct-npm-init) learning-babel
version: (1.0.0)
description: Use Babel to transpile JavaScript ES6 to ES5
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/ccuser/workspace/learn-javascript-transpilation-bct-npm-init/package.json:

{
  "name": "learning-babel",
  "version": "1.0.0",
  "description": "Use Babel to transpile JavaScript ES6 to ES5",
  "main": "index.js",
  "scripts": {
```

```
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this ok? (yes)
$ ls
package.json  src
$
```

디렉토리에 package.json가 생성된 모습이다.

- **Node Packages 설치하기**

npm 의 install 커맨드로 새로운 노드 패키지를 지역적으로 설치할 수 있다. install 커맨드는 **node_modules** 라는 폴더를 생성하고 패키지 파일들을 거기에 복사한다. install 커맨드는 또한 패키지에 종속 된 모든 것들을 설치한다.

바벨을 설치하기 위해, npm install babel-cil 과 npm install babel-preset-env 이 두가지 패키지가 필요하다.

We install Babel with the following two commands:

```
$ npm install babel-cli -D  
$ npm install babel-preset-env -D
```

'babel-cli' 패키지는 커맨드 라인 바벨틀 을 포함하고 'babel -preset-env' 패키지는 어느 JavaScript ES6를 ES5 로 맵핑하는 코드를 가지고 있다.

-D 플래그는 npm에게 각각의 패키지에 package.json 에서 devDependencies 라고 불리는 프로퍼티를 추가하라고 지시한다.

일단 프로젝트의 종속들이 devDependencies 로 나열되면, 다른 개발자들은 각각 패키지를 따로 설치할 필요 없이 너의 프로젝트를 실행할 수 있다.

위와 달리, npm install 을 실행할 수 있다. 즉, 이 커맨드는 npm 에게 package.json 의 내부를 보고 devDependencies 에 나열된 모든 패키지를 다운로드하라고 시킨다.

일단 너가 npm install 한다면 너는 babel package 를 발견할 수 있고, 그것에 종속된 모든 것들을 node_modules 폴더에서 발견할 수 있다.

```
project
├─ node_modules
├── .bin
├── ...
├─ src
├── main.js
└─ package.json
```

... 은 npm 이 설치한 100개 이상의 패키지들이 설치된 것을 나타낸다.

- **.babelrc**

지금까지 babel 을 다운로드 했으니 자바스크립트 코드소스의 버전을 명시해야한다.

초기 자바스크립트 버전을 .babelrc 라는 파일 내부에 명시할 수 있다. root 디렉토리에서 touch .babelrc 를 실행하여 이 파일을 만들 수 있다.

```
project
|_ node_modules
|___ .bin
|___ ...
|_ src
|___ main.js
|_ .babelrc
|_ package.json
```

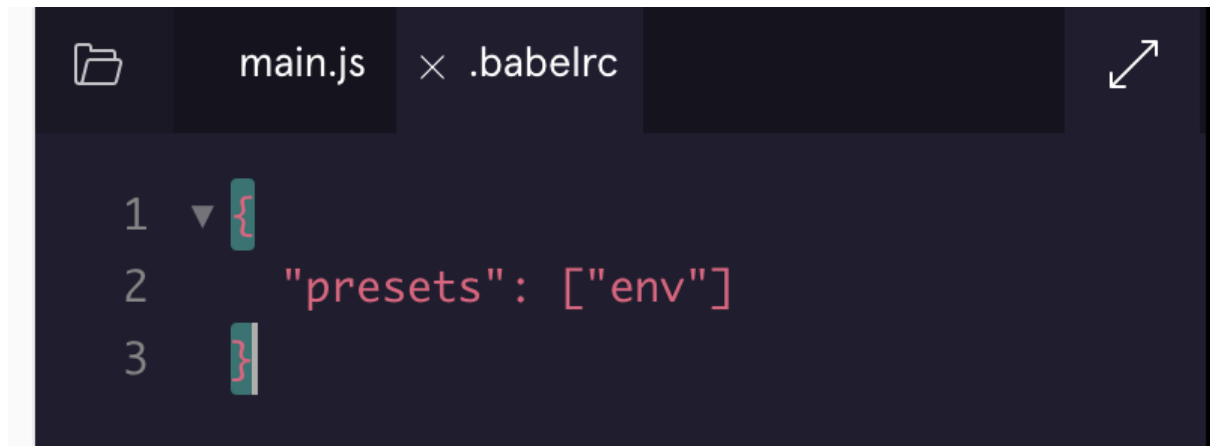
.babelrc 내부에서 나의 자바스크립트 파일 소스에 대한 preset을 정의해줘야한다. preset은 자신의 초기 자바스크립트 파일의 버전을 명시한다.

보통 우리는 ES6 에서 ES5 로 변경하길 원한다.

ES6+ 소스로 부터 우리가 코드를 트랜스파일링한다고 명시하기 위해, .babelrc 내에 다음 아래 객체를 입력해야한다.

```
{
  "presets": ["env"]
}
```

바벨을 실행할 때, babelrc 에서 초기 자바스크립트 파일의 버전을 결정한다. ["env"] 는 바벨에게 ES6+ 로 부터 코드를 트랜스파일링 하라고 지시한다.



- **Babel Source Lib**

이제 우리의 코드를 트랜스파일링 하기 위한 마지막 단계이다. 우리는 ES6+ 를 ES5 로 초기화하는 package.json 의 스크립트를 명시해야한다.

package.json 파일 내부에 커맨드라인 실행법을 지정하기 위한 객체를 갖는 script 라는 프로퍼티가 있다.

```
...  
"scripts": {  
  "test": "echo \"Error: no test  
specified\" && exit 1"  
}, ...
```

```
...  
"scripts": {  
  "test": "echo \"Error: no test  
specified\" && exit 1",  
  "build": "babel src -d lib"  
}
```

test 프로퍼티 아래에 우리는 바벨을 실행시키는 하나의 스크립트 "build"를 추가했다. 이 커맨드라인 "babel src -d lib" 는 ES6+ 코드를 ES5 코드로 변경시켜준다. 각 인자가 무엇을 의미하는지 살펴보자.

babel → 트랜스파일링 코드를 위한 바벨 커맨드 콜

src → 바벨에게 src 디렉토리 내부에 있는 모든 자바스크립트 코드를 트랜스파일하라고 지시하는 것이다.

-d → 하나의 디렉토리에 트랜스파일된 코드를 작성하라고 바벨에게 지시하는 것.

lib → 바벨이 트랜스파일된 코드를 lib 라고 하는 디렉토리에 작성하게 하는 것

```

1  ▼ {
2      "name": "learning-babel",
3      "version": "1.0.0",
4      "description": "Use Babel to
    transpile JavaScript ES6 to ES5",
5      "main": "index.js",
6  ▼  "scripts": {
7      "test": "echo \"Error: no test
    specified\" && exit 1"
8      },
9      "build": "babel src -d lib",
10     "author": "",
11     "license": "ISC",
12  ▼  "devDependencies": {
13      "babel-cli": "^6.26.0",
14      "babel-preset-env": "^1.7.0"
15  }
16  }
17

```

build 프로퍼티를 커맨드 라인과 함께 직접 추가

- **build**

이제 build 를 호출해보자.


```
npm run build
```

바벨은 ES5 코드 파일을 원본 이름과 같도록 작성해서 lib 폴더안에 저장한다.

```
project
|_ lib
|___ main.js
|_ node_modules
|___ .bin
|___ ...
|_ src
|___ main.js
|_ .babelrc
|_ package.json
```

또한 코드를 트랜스파일할때 src 안에 있는 모든 파일을 갯수에 상관없이 트랜스 파일한다. 이것은 규모가 큰 자바스크립트 프로젝트를 할 때 매우 유용하다.

- 정리 Review

- ES5 — The old JavaScript version that is supported by all modern web browsers.
- ES6 — The new(er) JavaScript version that is *not* supported by all modern web browsers. The syntax is more readable, similar to other programming languages, and addresses the source of common bugs in ES5.

- caniuse.com — a website you can use to look up HTML, CSS, and JavaScript browser compatibility information.
- Babel — A JavaScript package that transpiles JavaScript ES6+ code to ES5.
- `npm init` — A terminal command that creates a **package.json** file.
- **package.json** — A file that contains information about a JavaScript project.
- `npm install` — A command that installs Node packages.
- `babel-cli` — A Node package that contains command line tools for Babel.
- `babel-preset-env` — A Node package that contains ES6+ to ES5 syntax mapping information.
- **.babelrc** — A file that specifies the version of the JavaScript source code.
- `"build"` script — A **package.json** script that you use to transpile ES6+ code to ES5.
- `npm run build` — A command that runs the `build` script and transpiles ES6+ code to ES5.

For future reference, here is a list of the steps needed to set up a project for transpilation:

1. Initialize your project using `npm init` and create a directory called **src**
2. Install babel dependencies by running

```
npm install babel-cli -D
npm install babel-preset-env -D
```

3. Create a **.babelrc** file inside your project and add the following code inside it:

```
{
  "presets": ["env"]
}
```

4. Add the following script to your `scripts` object in **package.json**:

```
"build": "babel src -d lib"
```

5. Run `npm run build` whenever you want to transpile your code from your **src** to **lib** directories.

3. 클린 코드를 위한 5가지 팁 - by 노마드 코더


좋은 코드란 코드 그 자체로 설명이 다 되는 코드

첫번째, 검색이 가능한 이름을 써라. 숫자와 같은 변수에 어떤 숫자인지 변수로 지정하는 것이 좋다.

두번째, 함수명은 무조건 동사를 써라!! 액션중심으로 이름이 짓자. 함수는 단 한 가지 역할만 해야한다!!!

세번째, 인수는 3개 이하가 제일 좋다!! 만약 피치못하게 인수를 많이 써야한다면 한개의 **configuration object** 를 보내는 것을 추천한다!

```
function makePayment(price, productId, size, quantity, userId){  
  // process payment  
}  
  
makePayment(35, 5, "xl", 2, "니꼬")
```



```
function makePayment({ price, productId, size, quantity, userId }) {  
  // process payment  
}  
  
makePayment({  
  price: 35,  
  productId: 5,  
  size: "xl",  
  quantity: 2,  
  userId: "니꼬",  
});
```

네번째, 불리언 값을 인수로 함수에 보내는 것을 최대한 방지하자!!

인수에 불리언이 들어간다는 것은 함수안에 if else 문이 들어 간다는 것인데 이러한 경우 각 if else 를 다른 함수로 분리하는 것이 좋다! 함수는 단 한가지 역할만 해야한다는 것을 명심하자.

다섯번째, 짧은 변수명이나 (아무도 이해못하는) 축약어 쓰는 것을 피하자!

```
allUsers.forEach((u, i) => {  
  sendEmail(u);  
  addToCount(i);  
});
```



```
allUsers.forEach((user, currentNumber) => {  
  sendEmail(user);  
  addToCount(currentNumber);  
});
```



처음 코딩을 작성할때는 이쁘게 작성하려고 하지말고 일단 코드를 쓰고나서 작동되는것이 확인되고 클린코드로 바꾸면 된다! 처음부터 하려고하면 너무 혼돈이라 꼭 마지막에 작업하도록!!!!

4. 노마드코더 바닐라 JS 그림판 만들기 - 그림판 fill 구현, 그림판 저장 구현

5. Github README.md 작성 방법 팁 - by 드림코딩 엘리

https://www.youtube.com/watch?v=kMEb_BzyUqk&list=PLM14_cQ8kSmvwExMwjBRJgOxfg72JVqGh&index=5

오늘의 단어

- dire : 대단히 심각한, 엄청난, 지독한
- prompt : 유도하다. ; 즉각적인, 지체없는 ; 신속한, 시간을 엄수하는; (사람에게 어떤 결정을 내리도록, 어떤 일이 일어나도록) 하다(= provoke 촉발하다)
- vulnerabilities : 취약성
- latter : 후자의, 마지막의
- mitigate : 완화시키다.
- pitfall : (눈에 잘 안 띄는) 위험, 곤란
- friction : 마찰