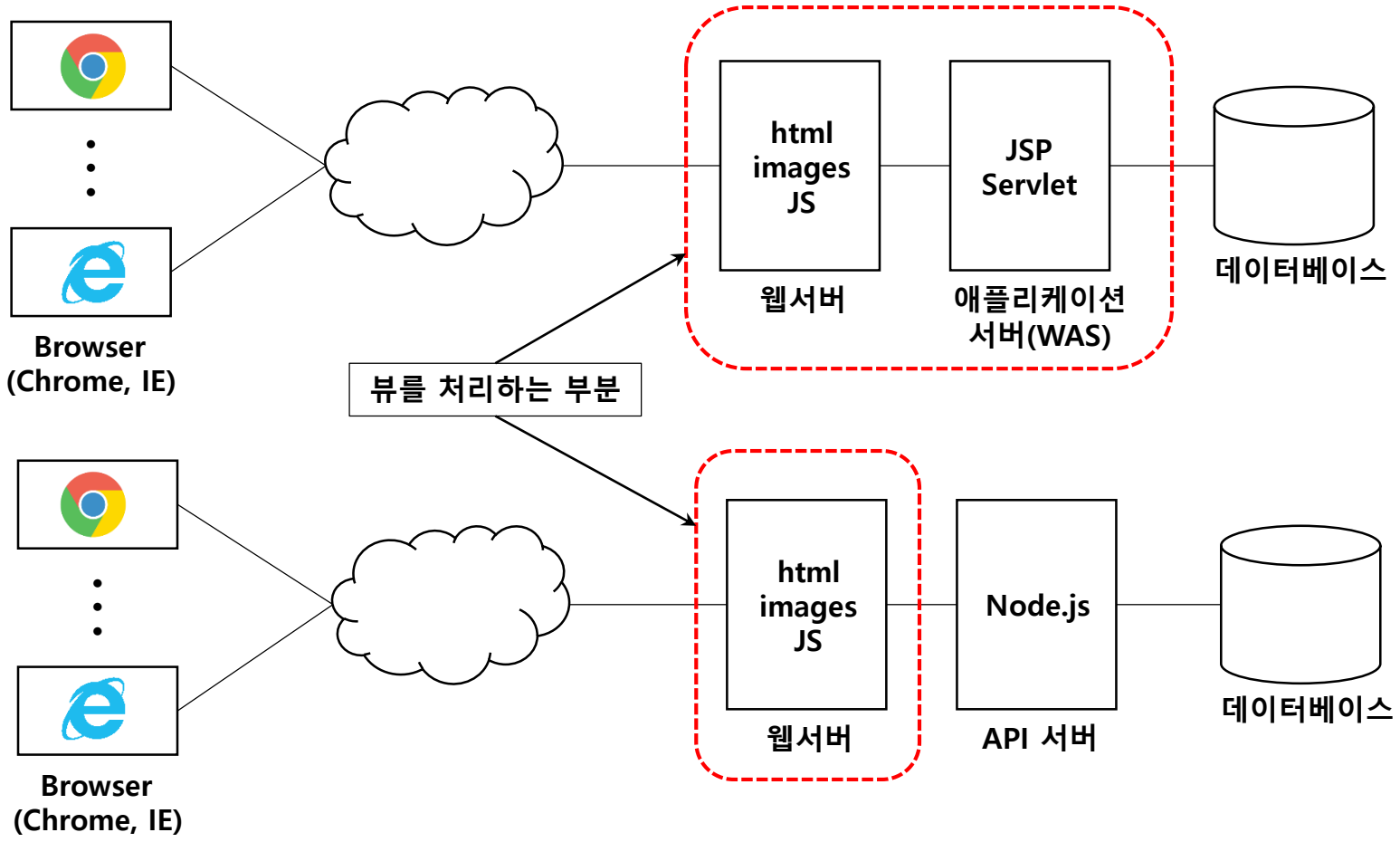

React

1. Front-end 자바스크립트

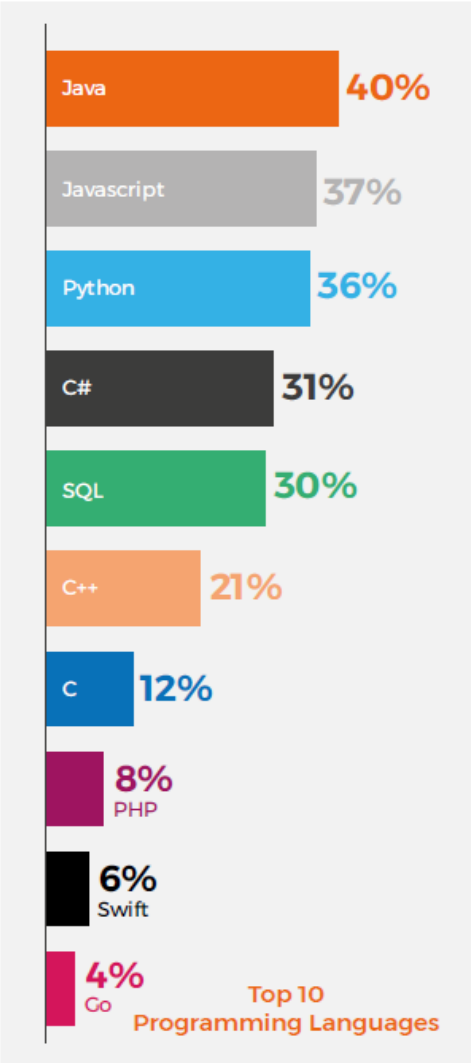


1. Front-end 자바스크립트

1.1 자바스크립트 라이브러리/프레임워크

- 브라우저 스크립트에서 벗어나 Front/Back-end에서 다양하게 활용
- 자바스크립트만으로 얼마든지 시스템 구축이 가능

  https://github.com/collections/front-end-javascript-frameworks



(Skill Up 2018)

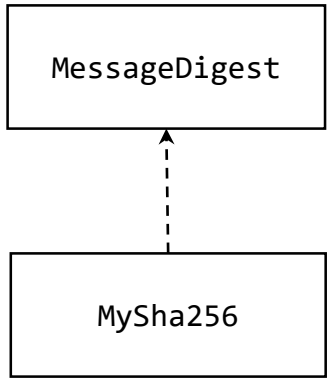
1. Front-end 자바스크립트

1.2 자바스크립트 모듈화

- **자바스크립트 애플리케이션 개발에 사용되는 다양한 라이브러리 모듈과 모듈 간의 의존성 관리 필요**

```
package com.foo.test;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class MySha256 {

    public static void main(String[] args) {

        String str = "Alice gave 0.015BTC to EDIYA";
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] encodedhash = digest.digest(str.getBytes(StandardCharsets.UTF_8));

            StringBuffer sb = new StringBuffer();
            for(int i = 0 ; i < encodedhash.length ; i++){
                sb.append(Integer.toString((encodedhash[i]&0xff) + 0x100, 16).substring(1));
            }
            System.out.println(sb.toString());

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

“encapsulation”
“dependency”



- 일반적인 프로그래밍에서는 처음부터 모든 것을 구현하기 보다는 목적에 맞는 라이브러리나 이미 구현된 기능들을 참조하여 코드를 작성한다.
- 사용하는 라이브러리들 사이의 관계 – 의존성을 관리해주는 다양한 빌드 도구들이 제공된다. (예) Maven, Gradle

1. Front-end 자바스크립트

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Foo</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="./assets/css/normalize.css">
    <link rel="stylesheet" href="./assets/css/bootstrap.min.css">
    <link rel="stylesheet" href="./assets/css/font-awesome.min.css">
    <link rel="stylesheet" href="./assets/css/themify-icons.css">
    <link rel="stylesheet" href="./assets/css/flag-icon.min.css">
    <link rel="stylesheet" href="./assets/css/cs-skin-elastic.css">
    <link rel="stylesheet" href="./assets/scss/style.css">
</head>
<body>

<script src="web3.min.js"></script>
<script src="./assets/js/vendor/jquery-2.1.4.min.js"></script>
<script src="./assets/js/popper.min.js"></script>
<script src="./assets/js/plugins.js"></script>
<script src="./assets/js/main.js"></script>
<script src="./assets/js/lib/peitychart/jquery.peity.min.js"></script>

<!-- script init-->
<script src="./assets/js/lib/peitychart/peitychart.init.js"></script>
<!-- script init-->

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

- 고전적인 방식에서는 `<script>` 태그를 사용하여 필요한 자바스크립트를 순서에 맞게 나열하는 형태로 dependency를 관리한다.
- 이 경우에는 스크립트 라이브러리들 사이의 의존성은 파악되지 않고 개발자만이 그 관계를 알고 있는 경우가 많다.

1. Front-end 자바스크립트

1.2 자바스크립트 모듈화

- 자바스크립트의 “모듈화 방식(modular pattern)” : 즉시실행함수(IIFE), CommonJS, RequireJS(AMD, Asynchronous Module Definition), UMD(Universal Module Definition)
- ES6 부터는 자바스크립트 언어 레벨에서 모듈화를 지원(export, import)
- Node.js 에서는 CommonJS 형식을 사용하여 모듈을 지원(`require, module.exports`)
- Node.js는 npm으로 모듈의 의존성을 관리 – package.json, package-lock.json

1. Front-end 자바스크립트

1.2 자바스크립트 모듈화

npm init

```
PS C:\Users\foo\hello-react> dir

    디렉터리: C:\Users\foo\hello-react

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----     2018-09-04   오후  4:21                .idea
d-----     2018-09-02   오후  2:18                node_modules
d-----     2018-09-03   오후  9:51                public
d-----     2018-09-04   오전 11:50                src
-a----     2018-09-02   오전 11:02            285 .gitignore
-a----     2018-09-02   오후  2:19         422273 package-lock.json
-a----     2018-09-02   오후  2:19            382 package.json
-a----     2018-09-02   오전 11:02         121322 README.md
```

해당 프로젝트가 의존하는 모듈들이 들어 있는 디렉토리, require( )으로 참조

모듈들의 전체 의존성 트리(의존성 재현)

설치되는 모듈 정보, 자동화 명령어

1. Front-end 자바스크립트

1.3 모듈 번들링

- 모듈을 브라우저에서 쓰려면 require함수를 인식할 수 있도록 변환해주어야 한다.

myScript.js

```
const d = require("./myModule.js");
const _ = require("underscore");
...
```

browserify myScript.js -o bundle.js

"bundling"

bundle.js

```
(function(){function r(e,n,t){...})() ...
```

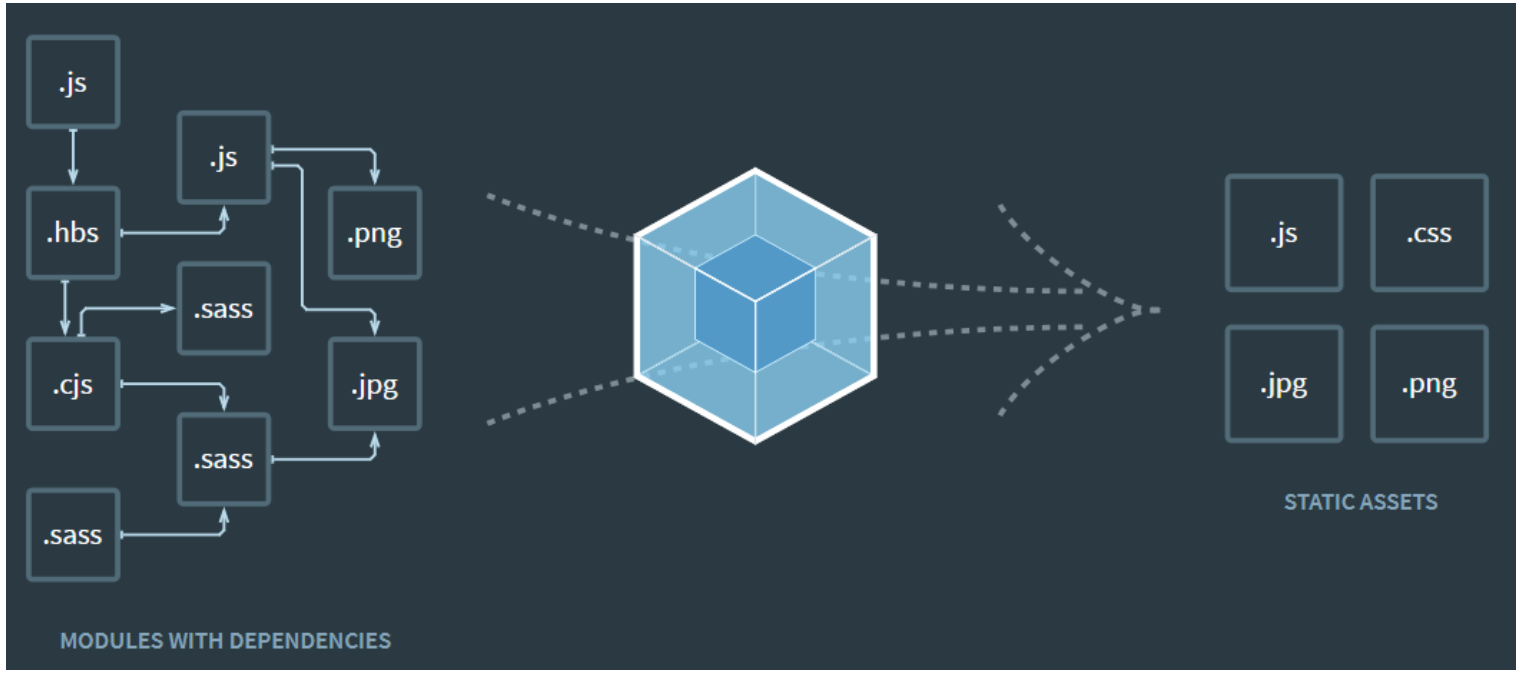
myModule

underscore

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

</head>
<body>
...
</body>
</html>
```

1. Front-end 자바스크립트

- **webpack**



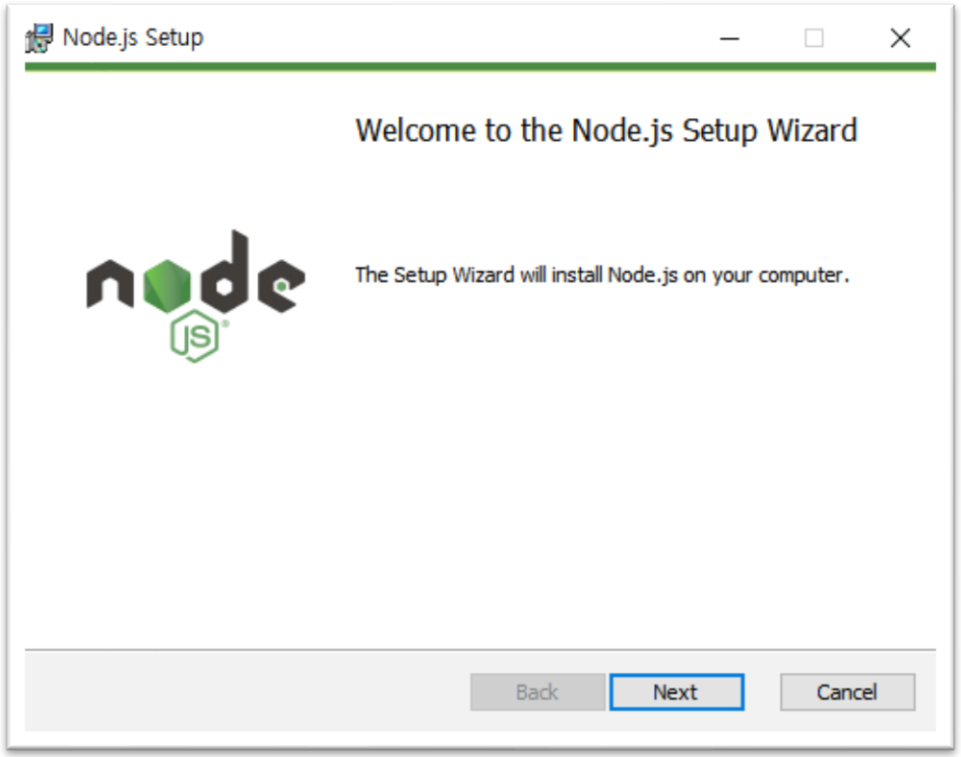
1. Front-end 자바스크립트

1.4 트랜스파일러(transpiler)

- **스크립트 → 스크립트 (source-to-source)**
- **다양하게 변형(또는 개선?)된 자바스크립트를 브라우저 호환성에 맞추어 “plain” 자바스크립트로 변환**
- **Backwards compatible**
- **Babel ES6(ES2015) → ES5(2009), JSX**
  https://babeljs.io/

2. 실습환경

2.1 Node.js 설치



- Node.js는 웹 브라우저 오픈 소스인 V8 기반(크롬)으로 만들어진 자바스크립트 런타임 환경(server-side)으로 최신 자바스크립트 표준(ES6, ES2017 등)을 지원한다.
- Node.js LTS(Long Term Support) 버전을 설치한다.
- Node.js를 설치하면 패키지 매니저인 npm도 함께 설치된다.

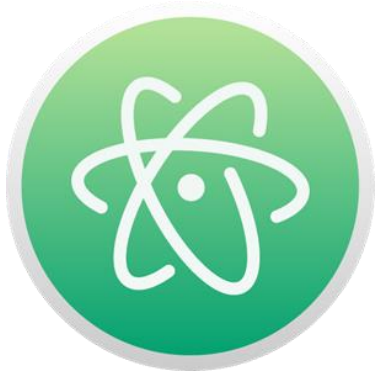
```
PS C:\Users\foo> node -v
v8.11.3
PS C:\Users\foo> npm -v
5.6.0
```

https://nodejs.org

2. 실습환경

2.2 소스파일 에디터 – 아톰(Atom)

https://atom.io/



패키지(플러그인) 설치

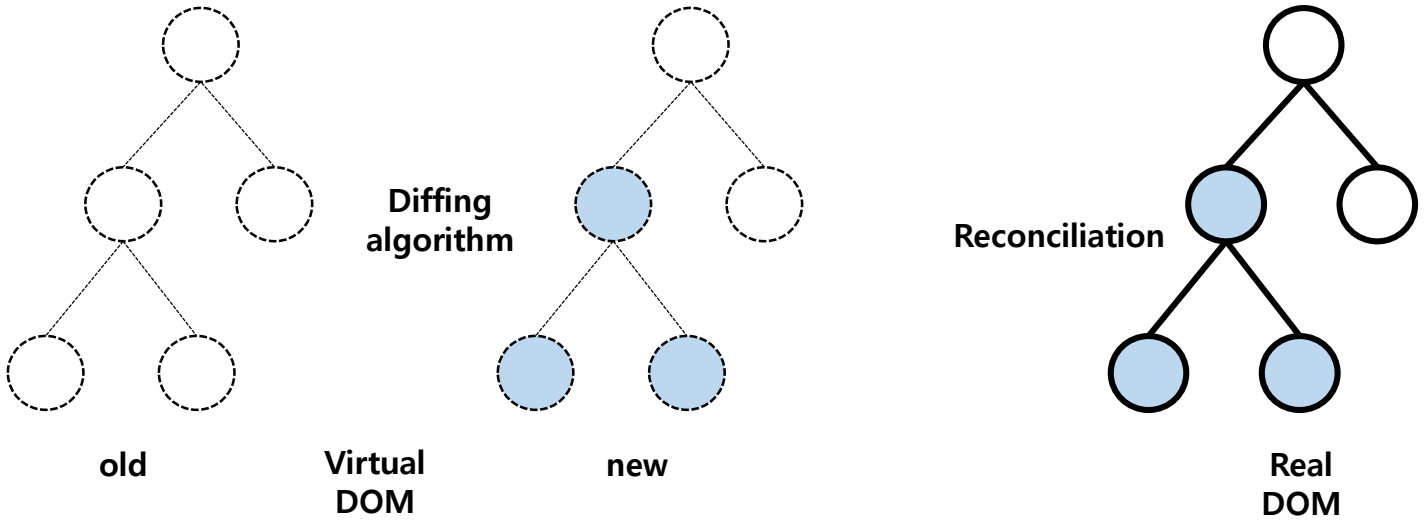
Settings > Install

- **platformio-ide-terminal Ctrl+`**
- **react**
- **autoclose-html**
- **open-in-browser Ctrl+Shift+Q**
- **react-snippets**

3. React – View를 처리하는 라이브러리

3.1 Virtual DOM

- 웹브라우저가 화면을 렌더링할 때 많은 연산과 처리를 수행(HTML, CSS, 변경된 DOM 업데이트)
- React는 Virtual DOM을 비교하여 변경된 부분만 Real DOM에 업데이트 → “Reconciliation”
- 부모 컴포넌트의 상태가 변경되면 자식 컴포넌트도 다시 렌더링



3. React – View를 처리하는 라이브러리

- **React**
  “declarative” programming – 상태만을 변경(“specifying what is to be done, not how”)
  DOM을 직접적으로 변경하지 않는다.

- **jQuery**
  “imperative” programming – 수행 명령을 순서대로 작성(“specifying how”)

3. React

3.2 React 컴포넌트

- ES6(ES2015)에서 새로 도입된 class를 사용하여 컴포넌트를 작성
- create-react-app(CRA)으로 부터 프로젝트 생성
- HTML태그와 유사한 JSX로 화면 요소들을 표현 → Babel에서 컴파일

```
react
react-dom
babel-preset-react
webpack
webpack-cli
css-loader
style-loader
file-loader
url-loader
babel-core
babel-loader
babel-preset-es2015
...
```

```
create-react-app
```

3. React

CRA로 프로젝트 생성, 컴포넌트 작성

- npx create-react-app hello-react 또는 create-react-app hello-react
- 컴포넌트 작성하기(class, function)

```
npm install –g truffle
```

```
npm install --global --production windows-build-tools
--vs2015
```

```
import Web3 from "web3";

const getWeb3 = () =>
  new Promise((resolve, reject) => {

    window.addEventListener("load", async () => {
      // Modern dapp browsers...
      if (window.ethereum) {
        const web3 = new Web3(window.ethereum);
        try {

          await window.ethereum.enable();
          resolve(web3);

        } catch (error) {
          reject(error);
        }
      }
...
```