

# 코딩 컨벤션





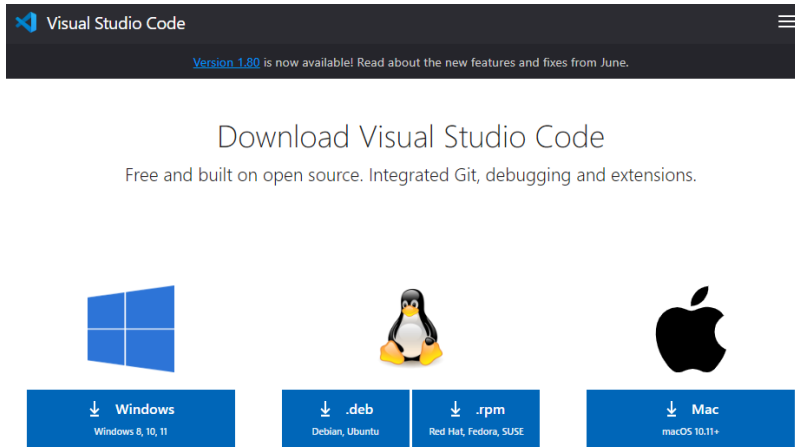
- ✓ 다른 개발자가 작성한 코드를 조금 더 쉽게 읽을 수 있도록, 코드의 복잡도가 높아지지 않도록 정하는 것



- ✓ 개발하면서 확인 및 점검을 하기 위해 사용하는 도구, 툴
- ✓ 여러 명의 개발자들이 같이 작업할 때 코딩 규칙을 정하여 가독성을 높이고 효율적인 개발을 진행하기 위해 사용
- ✓ 코딩 컨벤션이나 오류를 지속적으로 확인해주고 미리 알려주는 플러그인 or 프로그램
- ✓ Javascript, Python, Ruby, Java, Swift, HTML, CSS, YAML, 심지어 Markdown까지 대부분의 문법을 지원

# Development Environment Settings

IDE 를 다운받은 뒤 개발에 사용할 프로그래밍 언어에 맞춰 Code Convention 을 적용한다.  
대표적으로 MS의 Visual Studio Code 와 JetBrains 의 IntelliJ IDEA 가 있으며, Apache Foundation의 Eclipse 가 대표적.



Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

Windows 8, 10, 11

MacOS 10.11+

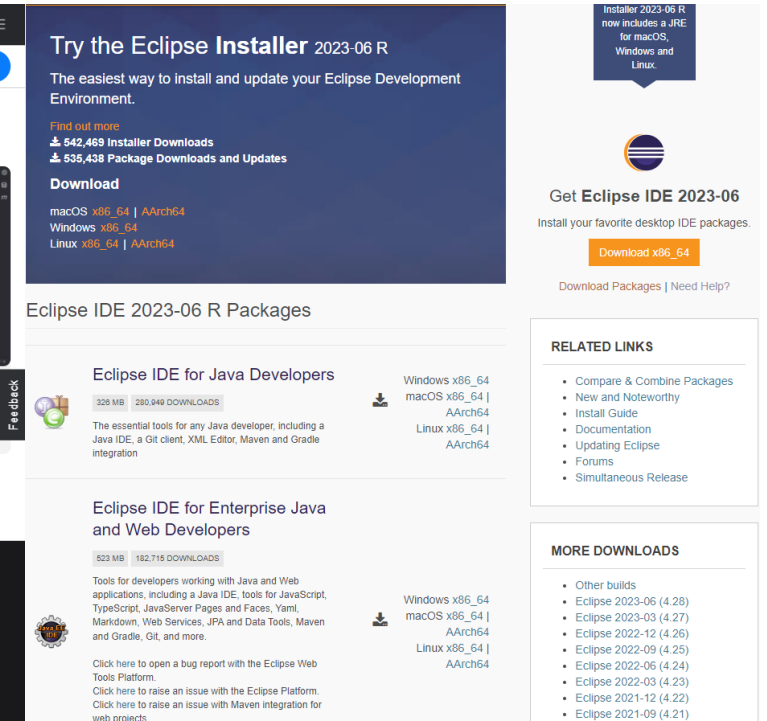
Linux (Debian, Ubuntu, Red Hat, Fedora, SUSE)

JetBrains IntelliJ IDEA Ultimate

Java 및 Kotlin용 최고의 IDE

다운로드

30일 무료 평가판



Try the Eclipse Installer 2023-06 R

The easiest way to install and update your Eclipse Development Environment.

Find out more

542,469 Installer Downloads

535,438 Package Downloads and Updates

Download

macOS x86\_64 | AArch64

Windows x86\_64

Linux x86\_64 | AArch64

Get Eclipse IDE 2023-06

Install your favorite desktop IDE packages.

Download x86\_64

Download Packages | Need Help?

RELATED LINKS

- Compare & Combine Packages
- New and Noteworthy
- Install Guide
- Documentation
- Updating Eclipse
- Forums
- Simultaneous Release

MORE DOWNLOADS

- Other builds
- Eclipse 2023-06 (4.28)
- Eclipse 2023-03 (4.27)
- Eclipse 2022-12 (4.26)
- Eclipse 2022-09 (4.25)
- Eclipse 2022-06 (4.24)
- Eclipse 2022-03 (4.23)
- Eclipse 2021-12 (4.22)
- Eclipse 2021-09 (4.21)

JetBrains는 멋진 도움을 주고 있는 커뮤니티에 보답하고자 하며, 이것이 IntelliJ IDEA Community Edition을 완전 무료로 제공하는 이유입니다.

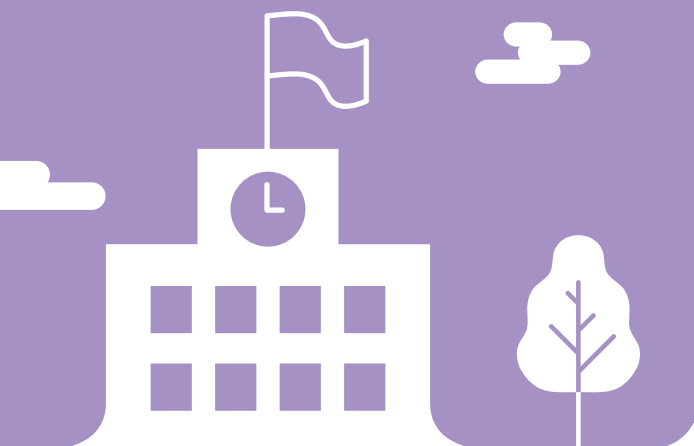
[MS의 Visual Studio Code](#)

[JetBrain 의 IntelliJ IDEA](#)

[Apache Foundation의 Eclipse](#)

# 코드 컨벤션 적용하기

## I Java





# Intelli J

## Naver Convention



## | Java - IntelliJ - Naver

1. Formatter 설정 파일을 다운로드해 준다.

# naver-intellij-formatter.xml

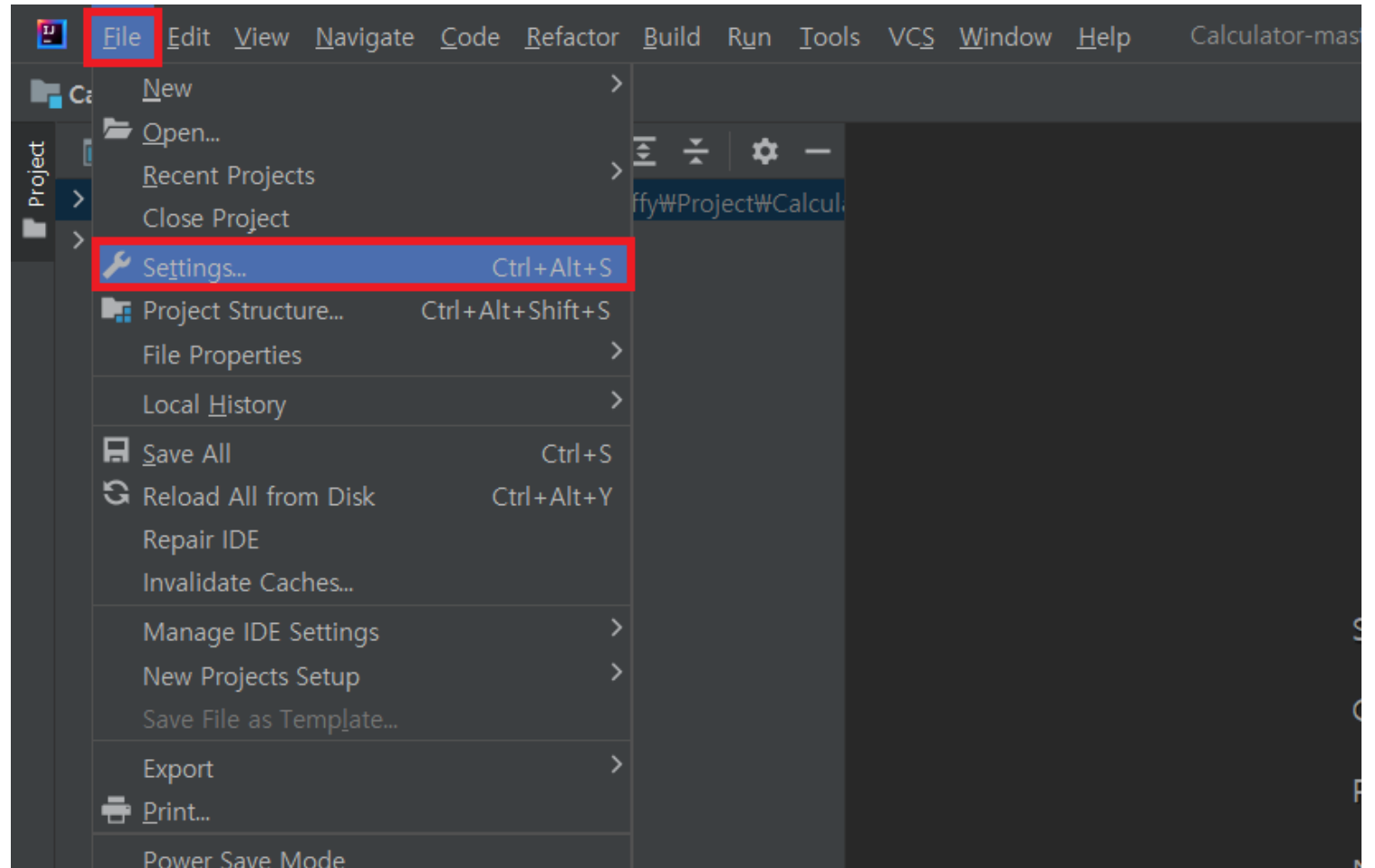
<https://github.com/naver/hackday-conventions-java/blob/master/rule-config/naver-intellij-formatter.xml>



## Java - IntelliJ - Naver

### 2. Formatter 설정을 해 준다.

#### File-Settings





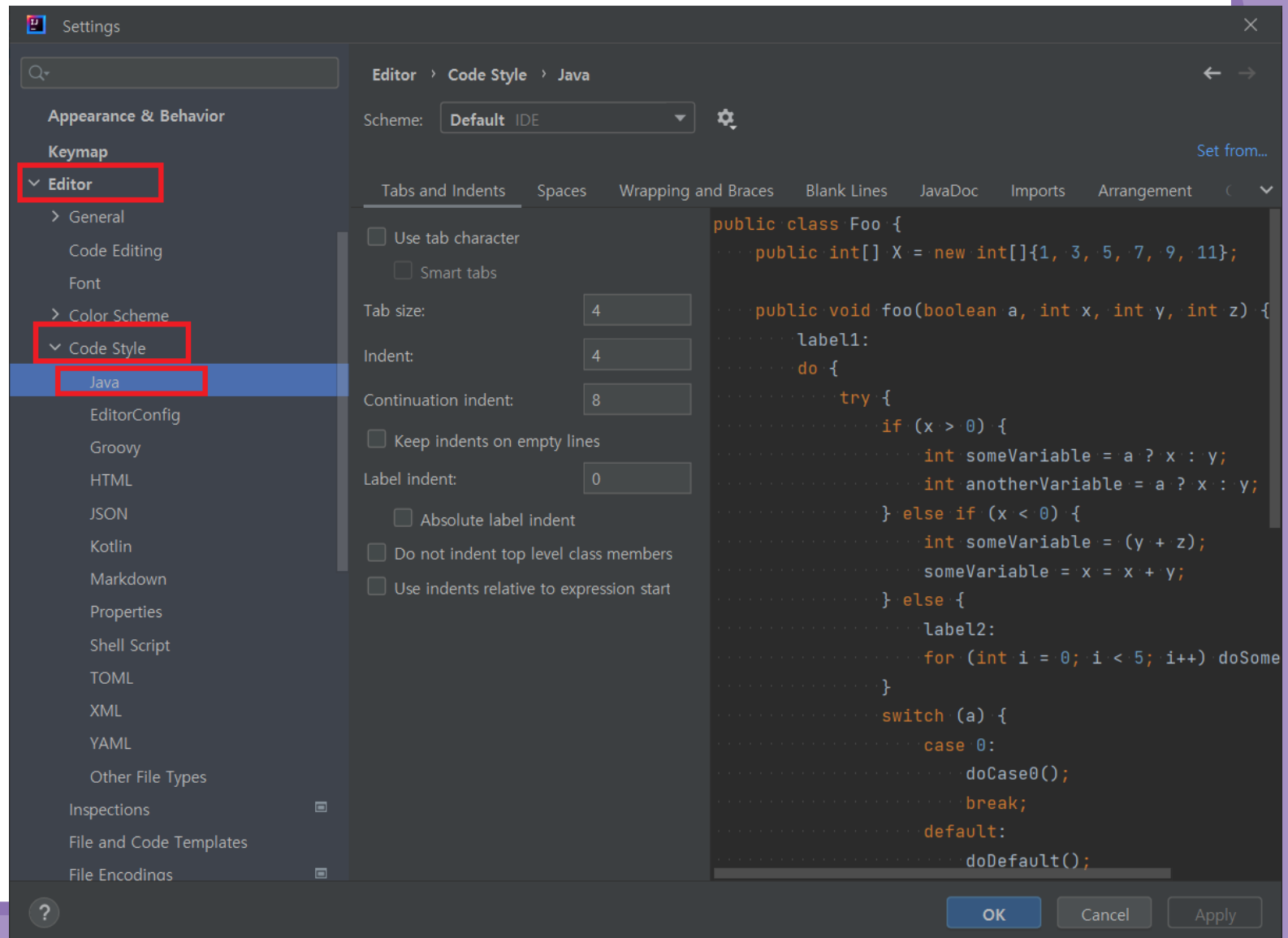
# Code Convention

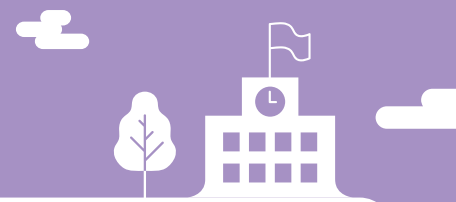


## Java - IntelliJ - Naver

### 2. Formatter 설정을 해 준다.

#### Editor- Code Style - Java



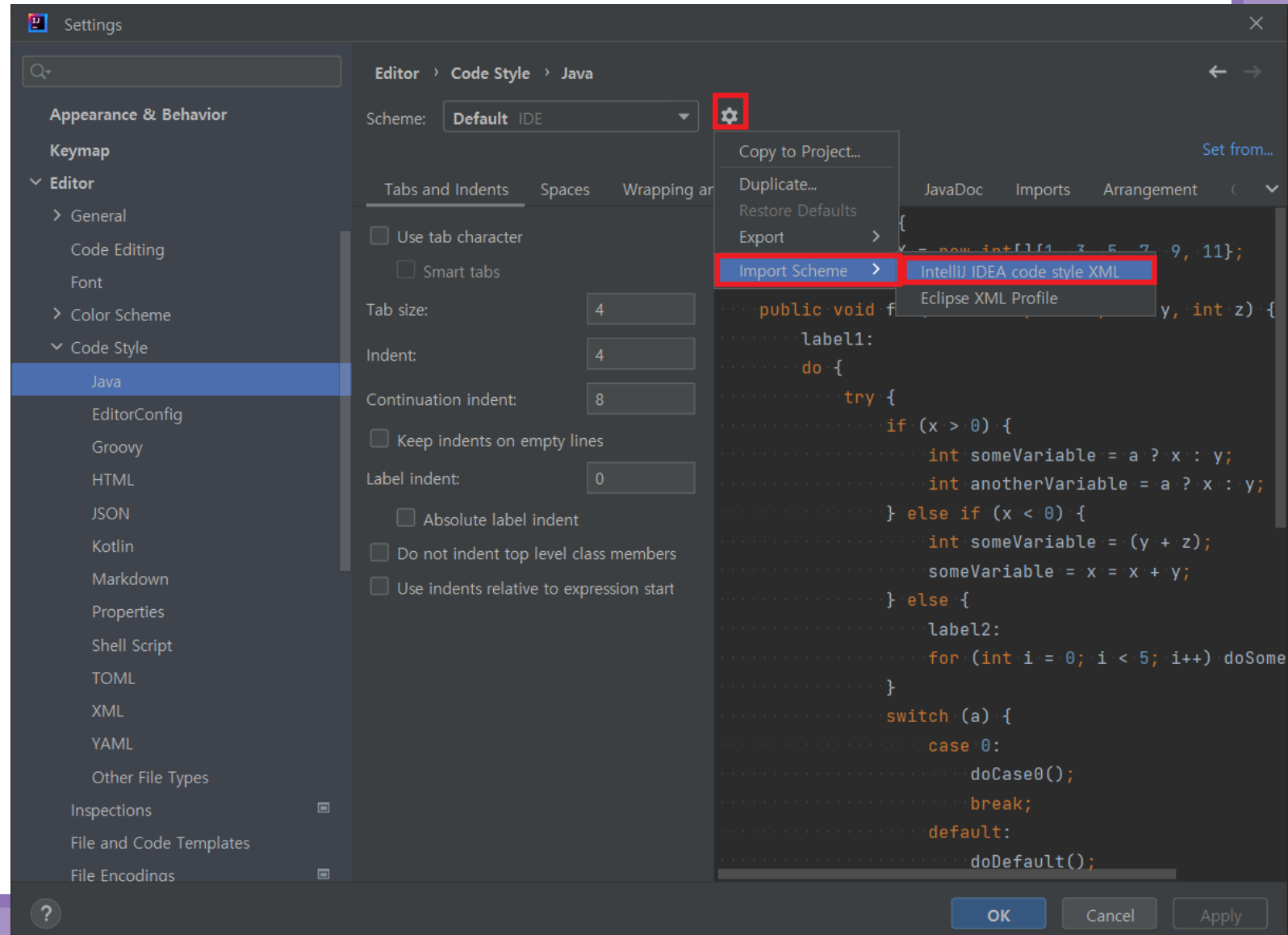


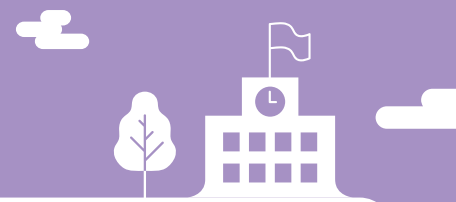
## Java - IntelliJ - Naver

### 2. Formatter 설정을 해 준다.

설정(톱니바퀴) - Import Scheme

- IntelliJ IDEA code style XML

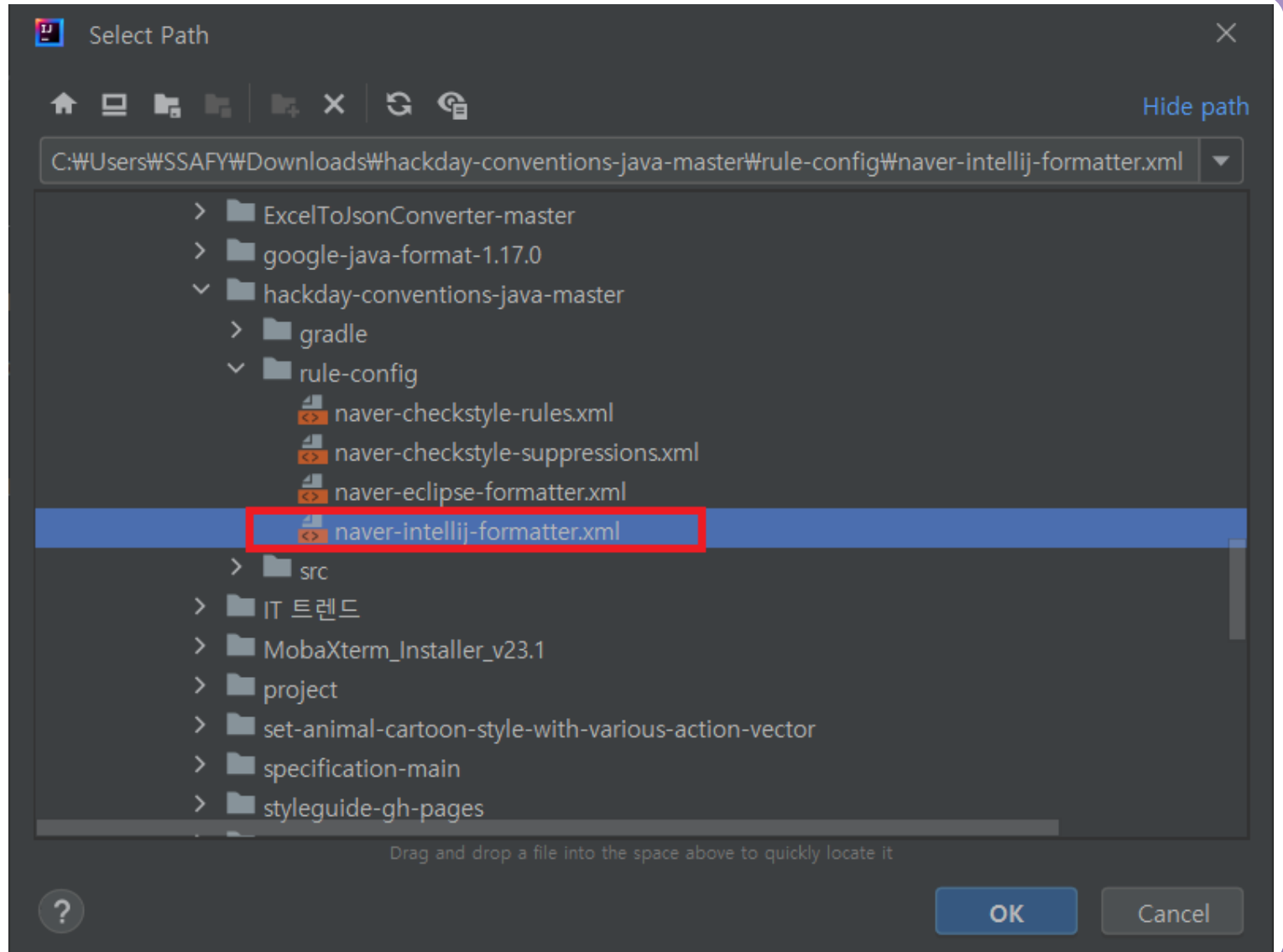


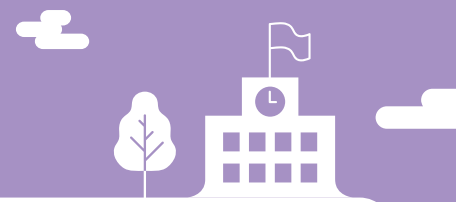


## Java - IntelliJ - Naver

### 2. Formatter 설정을 해 준다.

다운 받은 formatter.xml 세팅

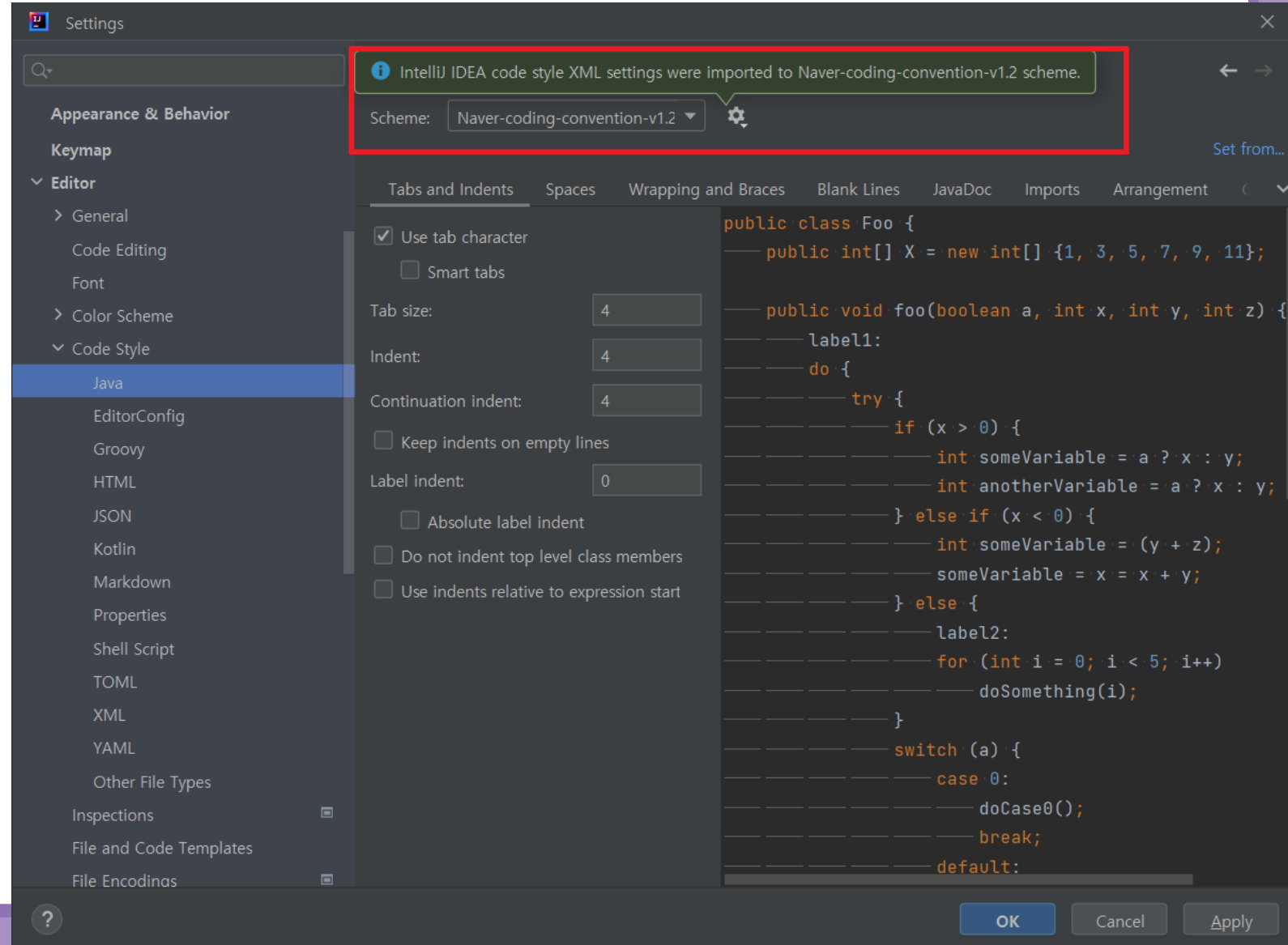




## Java - IntelliJ - Naver

### 2. Formatter 설정을 해 준다.

변경된 Naver Codeing Convention





# Intelli J

## Google Convention



## | Java - IntelliJ - Google

1. Formatter 설정 파일을 다운로드해 준다.

# intellij-java-google-style.xml

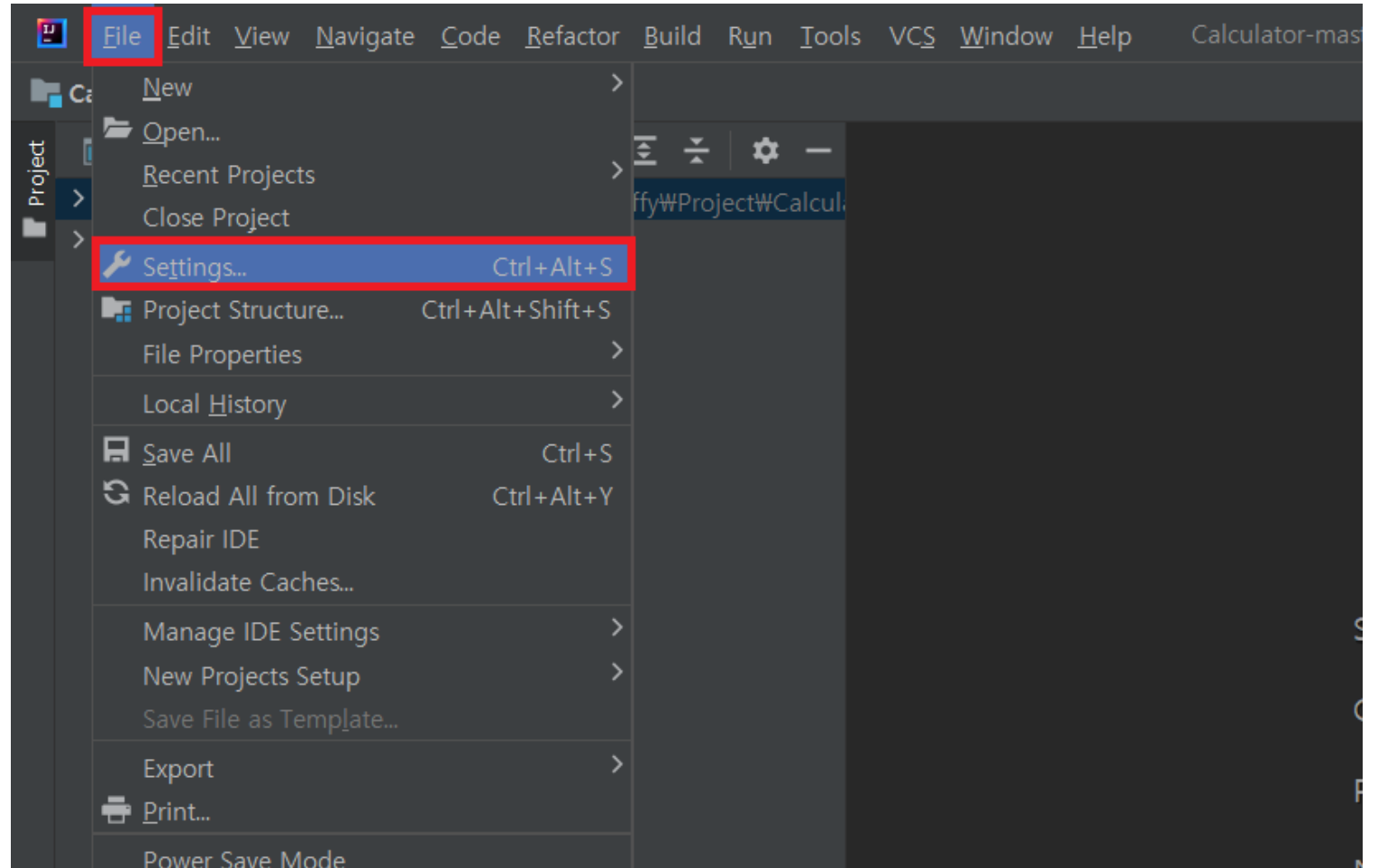
<https://github.com/google/styleguide>



## Java - IntelliJ - Google

### 2. Formatter 설정을 해 준다.

Window - Preferences



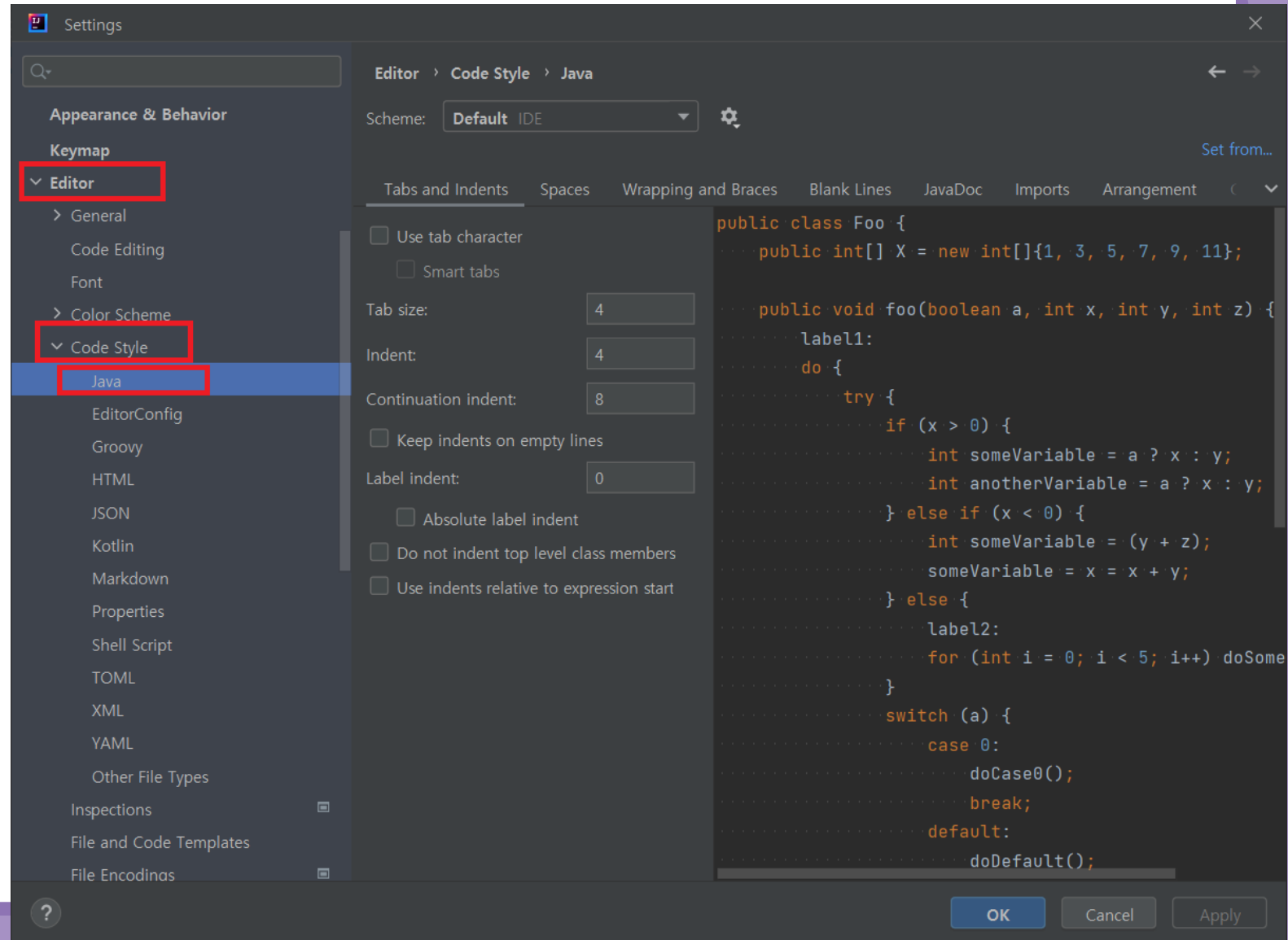
# Code Convention



## Java - IntelliJ - Google

### 2. Formatter 설정을 해 준다.

### Java - Code Style - Formatter





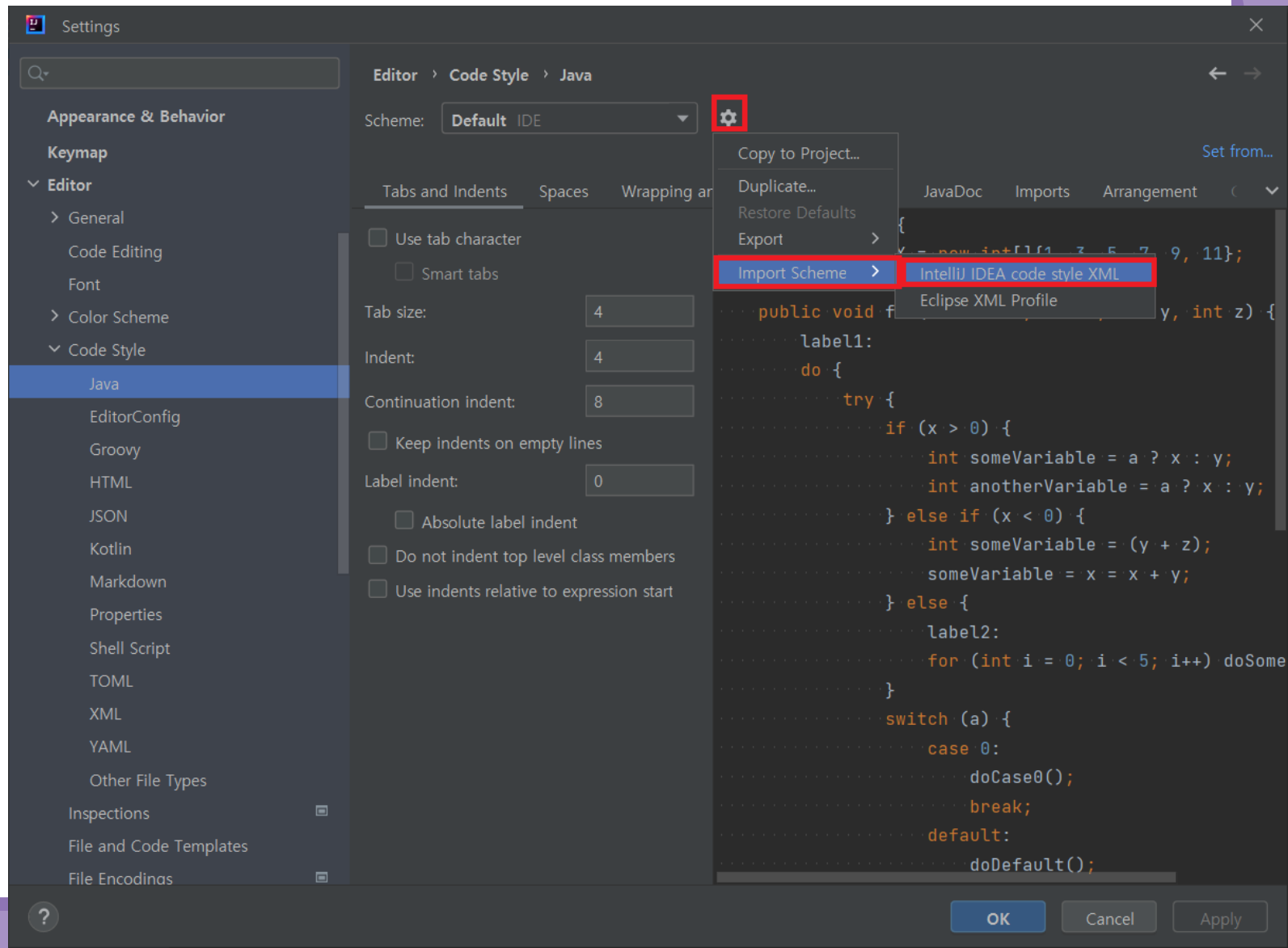
# Code Convention



## Java - IntelliJ - Google

### 2. Formatter 설정을 해 준다.

Import 버튼 클릭



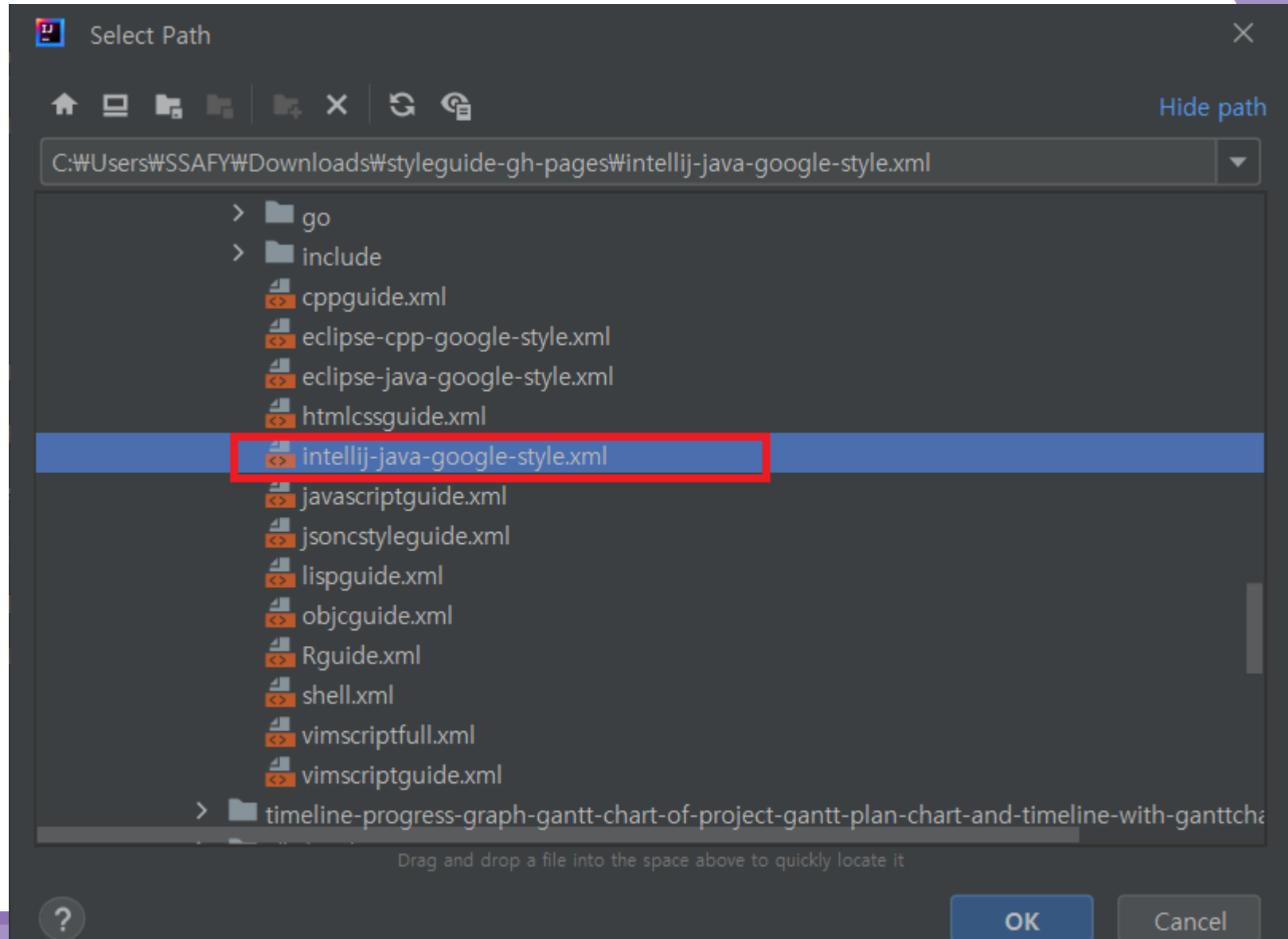
# Code Convention



## Java - IntelliJ - Google

### 2. Formatter 설정을 해 준다.

다운 받은 formatter.xml 세팅



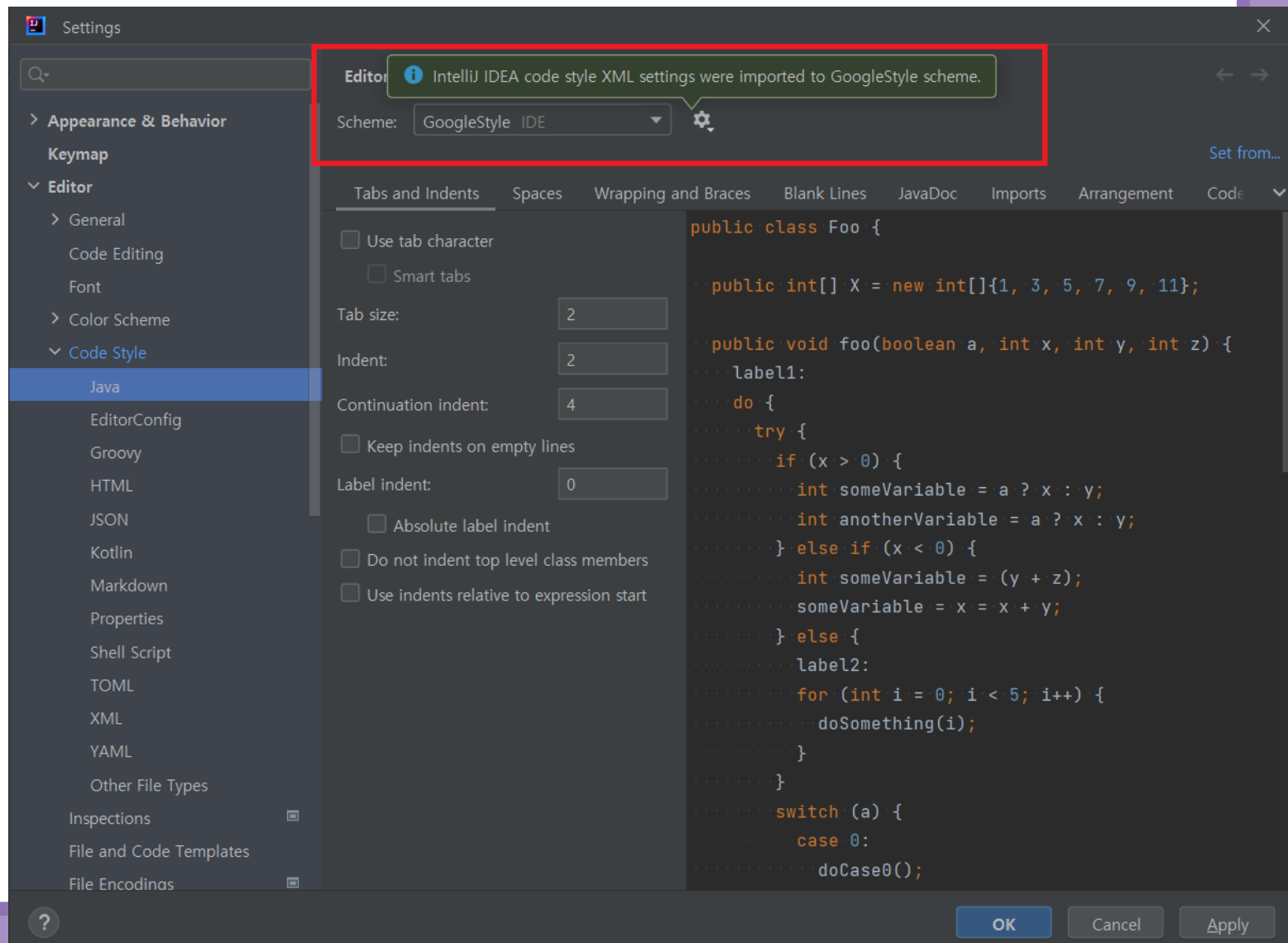
# Code Convention



## Java - IntelliJ - Google

### 2. Formatter 설정을 해 준다.

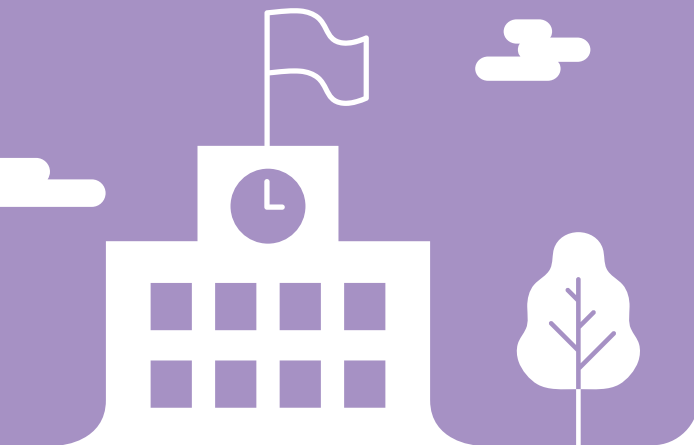
변경된 GoogleStyle 확인 가능



# 코드 컨벤션 적용하기

I Javascript

II Typescript



# Code Convention - Javascript



## | Javascript 코드 컨벤션 적용

- ✓ IDE - Visual Studio Code
- ✓ VScode Extends - ESLint, Prettier



## | ESLint

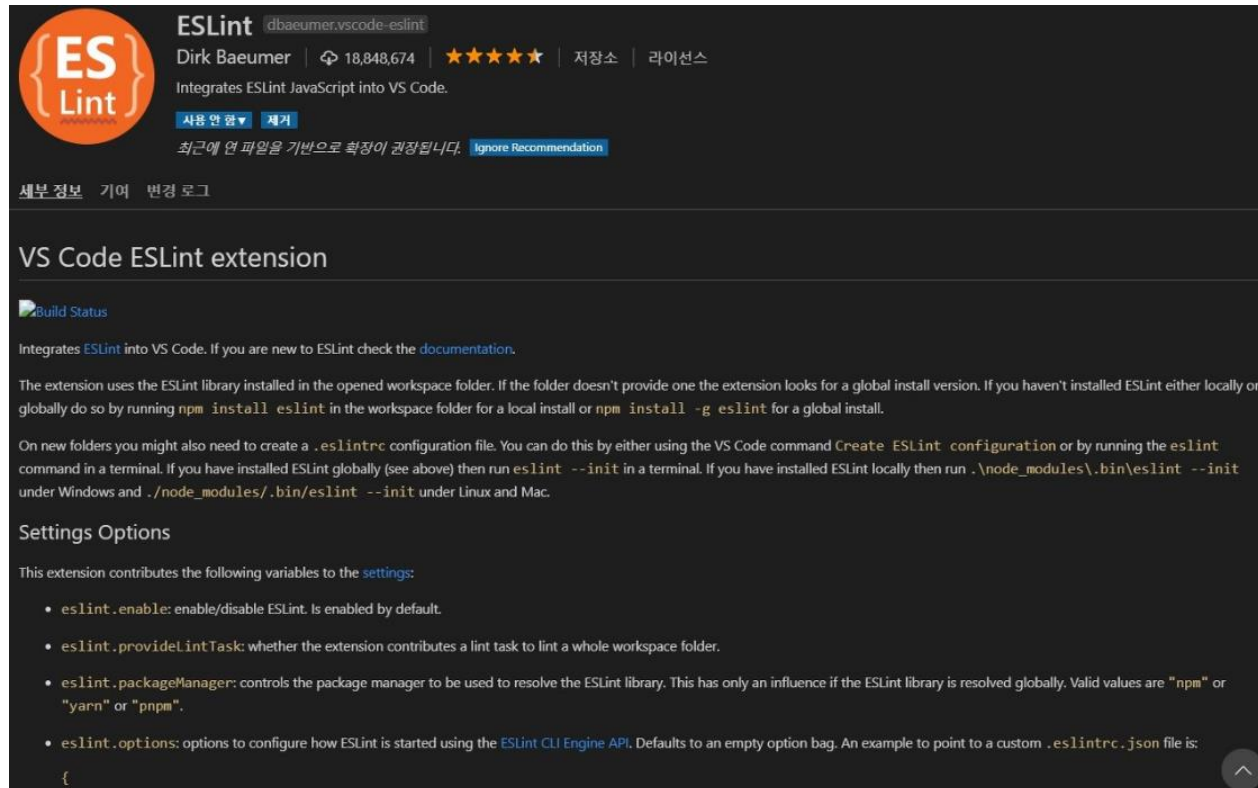
- ✓ JavaScript, JSX의 정적 분석 도구로 오픈 소스 프로젝트
- ✓ 커스터마이징이 쉽고 확장성이 뛰어나 많이 쓰이고 있음.

# Code Convention - Javascript



## ESLint

### VSCode 확장기능에서 ESLint를 설치



**ESLint** `dbaeumer.vscode-eslint`  
Dirk Baeumer | 18,848,674 | ★★★★★ | 저장소 | 라이선스  
Integrates ESLint JavaScript into VS Code.  
[사용 안 함](#) [재거](#)  
[최근에 연 파일을 기반으로 확장이 권장됩니다.](#) [Ignore Recommendation](#)

[세부 정보](#) [기여](#) [변경 로그](#)

### VS Code ESLint extension

Build Status

Integrates [ESLint](#) into VS Code. If you are new to ESLint check the [documentation](#).

The extension uses the ESLint library installed in the opened workspace folder. If the folder doesn't provide one the extension looks for a global install version. If you haven't installed ESLint either locally or globally do so by running `npm install eslint` in the workspace folder for a local install or `npm install -g eslint` for a global install.

On new folders you might also need to create a `.eslintrc` configuration file. You can do this by either using the VS Code command `Create ESLint configuration` or by running the `eslint` command in a terminal. If you have installed ESLint globally (see above) then run `eslint --init` in a terminal. If you have installed ESLint locally then run `./node_modules/.bin/eslint --init` under Windows and `./node_modules/.bin/eslint --init` under Linux and Mac.

### Settings Options

This extension contributes the following variables to the [settings](#):

- `eslint.enable`: enable/disable ESLint. Is enabled by default.
- `eslint.provideLintTask`: whether the extension contributes a lint task to lint a whole workspace folder.
- `eslint.packageManager`: controls the package manager to be used to resolve the ESLint library. This has only an influence if the ESLint library is resolved globally. Valid values are "npm" or "yarn" or "pnpm".
- `eslint.options`: options to configure how ESLint is started using the [ESLint CLI Engine API](#). Defaults to an empty option bag. An example to point to a custom `.eslintrc.json` file is:  
{

# Code Convention - Javascript



## ESLint

노드js 설치 - <https://nodejs.org/>

The screenshot shows the Node.js website with a dark blue header containing navigation links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. The main content area is dark blue and features the Node.js logo, a description of Node.js as an open-source, cross-platform JavaScript runtime environment, and a green banner announcing new security releases available June 20th, 2023. Below this, the 'Download' section offers two options: '18.16.0 LTS' (Recommended For Most Users) and '20.3.0 Current' (Latest Features). Links for 'Other Downloads', 'Changelog', and 'API Docs' are provided for each version. At the bottom, a link to the 'release schedule' is included.

nodejs

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

Node.js® is an open-source, cross-platform JavaScript runtime environment.

New security releases to be made available June 20th, 2023

Download

18.16.0 LTS  
Recommended For Most Users

20.3.0 Current  
Latest Features

Other Downloads | Changelog | API Docs    Other Downloads | Changelog | API Docs

For information about supported releases, see the [release schedule](#).



# Code Convention - Javascript



## | ESLint

"Ctrl + ~"로 터미널 실행, 터미널에서 아래 명령어 입력

```
npm install -g eslint
```

```
문제  출력 디버그 콘솔 터미널 1: powershell + - X ^
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\JS> npm install -g eslint
C:\Users\Dong\AppData\Roaming\npm\eslint -> C:\Users\Dong\AppData\Roaming\npm\node_modules\eslint\bin\eslint.js
+ eslint@5.15.3
updated 1 package in 3.204s
PS C:\JS> |
```

# Code Convention - Javascript



## ESLint

계속 터미널에서 아래 명령어 입력

`npm init`

입력을 요구하는데 쪽 엔터로 넘어가고 마지막에 'yes' 입력

그러면 기본값으로 `package.json` 파일이 생성

문제 출력 디버그 콘솔 터미널

```
PS C:\JS> 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>` afterwards to install a package and
save it as a dependency in the package.json file.
```

```
Press ^C at any time to quit.
package name: (js)
version: (1.0.0)
description:
entry point: (Test.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\JS\package.json:
```

```
{
  "name": "js",
  "version": "1.0.0",
  "description": "",
  "main": "Test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

```
Is this OK? (yes) yes
PS C:\JS> 
```

# Code Convention - Javascript



## | ESLint

터미널에서 마지막으로 아래 명령어 입력

`eslint --init`

어떤 설정을 사용할 것인지 물어본다.

자신이 사용하는 것을 위주로 선택 (추후 수정 가능)

중간에 설치 질문 나오는 것은 'yes' 선택

설치를 종료하고 나면 '`eslinttrc.js`' 파일이 생성

이 파일에 자신이 원하는 규칙을 rules에 입력

규칙들은 ESLint 사이트 참조

<https://eslint.org/docs/rules/>

```
문제  출력  디버그 콘솔  터미널
PS C:\JS> 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>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (js)
version: (1.0.0)
description:
entry point: (Test.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\JS\package.json:

{
  "name": "js",
  "version": "1.0.0",
  "description": "",
  "main": "Test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes) yes
PS C:\JS> 
```

# Code Convention - Javascript



## | ESLint - settings.json 수정

VSCode에서 F1키 눌러서 기본 설정 클릭

>|

기본 설정: 사용자 설정 열기

Preferences: Open User Settings

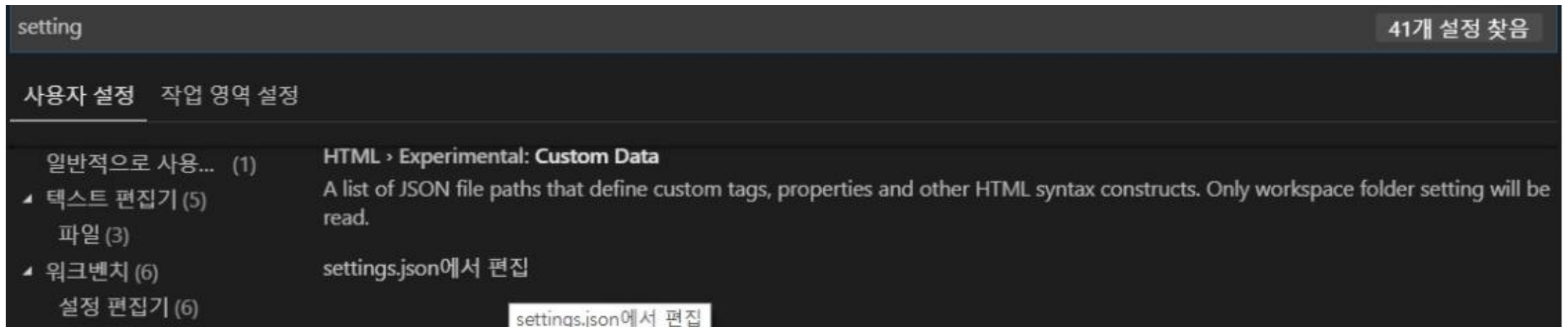
최근에 사용한 항목

# Code Convention - Javascript



## ESLint - settings.json 수정

setting를 치고 스크롤을 내리다보면 보이는 settings.json에서 편집하기 클릭



# Code Convention - Javascript



## | ESLint - settings.json 수정

settings.json 파일에 아래 설정 저장, 자신이 사용하는 언어가 있으면 추가하면 됨

```
"eslint.validate": [  
  {"language": "javascript"},  
  {"language": "html"},  
],
```

# Code Convention - Javascript



## | ESLint - settings.json 수정

.eslintrc.js 파일 수정 (자신이 최근에 사용하던 폴더에 생성되어 있음), 아래와 같이 설정

```
"extends": "eslint:recommended",
"rules": {
  "indent": [
    "error",
    4
  ],
  "no-unused-vars": 1,
  "no-use-before-define": 1,
  "no-redeclare": 1,
  "no-console": 0,
}
```

# Code Convention - Javascript



## ESLint - settings.json 수정

10. "\*.js"파일에서 코드 작성, 아래 문제창에서 에러 메시지 출력

```
1 function outfunction(arg1, arg2) {  
2     var local = 8;  
3     function innerfunction(innerArg) {  
4         console.log((arg1 + arg2) / (innerArg + local));  
5     }  
6     return innerfunction;  
7 }  
8 var exam = outfunction(2, 1);  
9 alert(exam);
```

문제 3 출력 디버그 콘솔 터미널

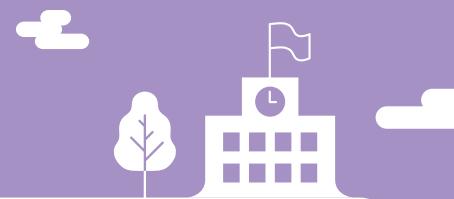
폴더입니다. 예: text, \*\*/\*.ts, /\*\*/node\_modules/\*\*

Testcode.js

- Expected indentation of 4 spaces but found 6. eslint(indent) [3, 1]
- 'outfunction' is not defined. eslint(no-undef) [8, 12]
- 'outfunction' is defined but never used. eslint(no-unused-vars) [1, 10]

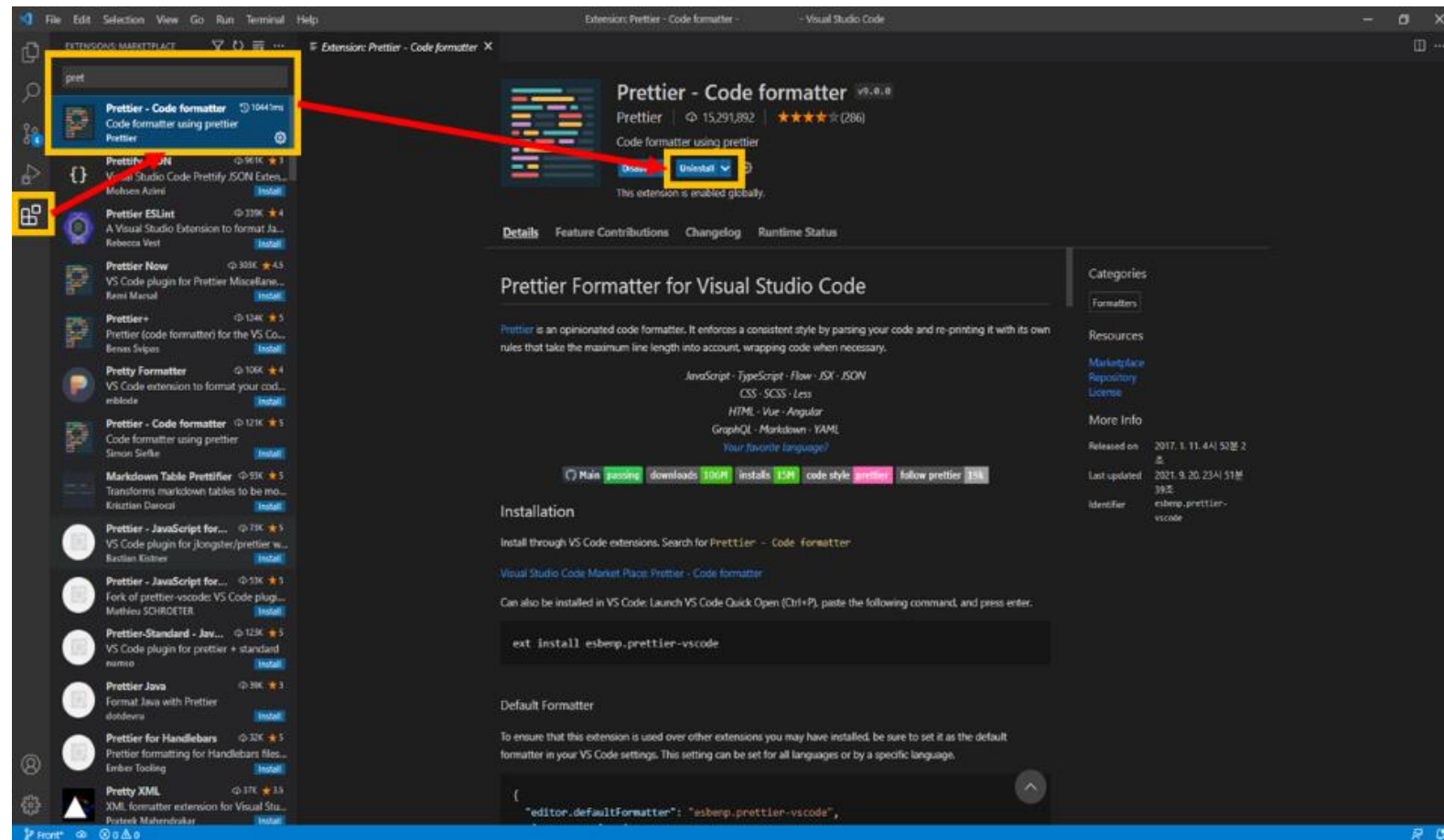


# Code Convention - Javascript

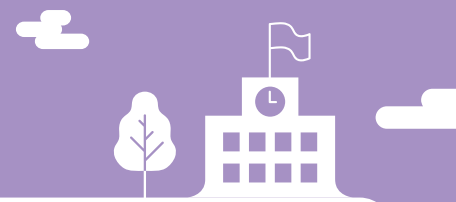


## Prettier - Prettier 설치하기

익스텐션 설치하기 (Install)

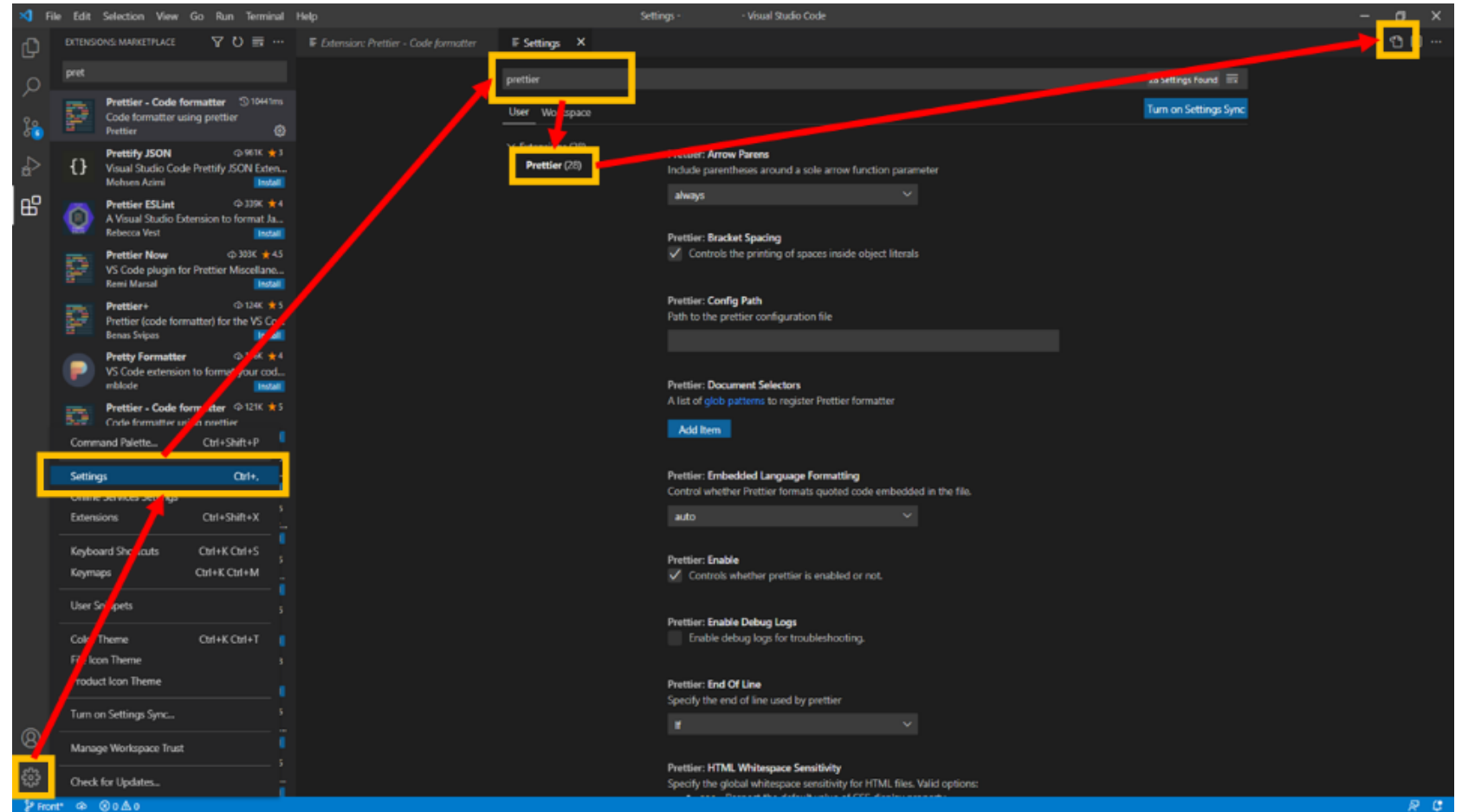


# Code Convention - Javascript



## Prettier - Prettier 설치하기

settings.json 접근하기



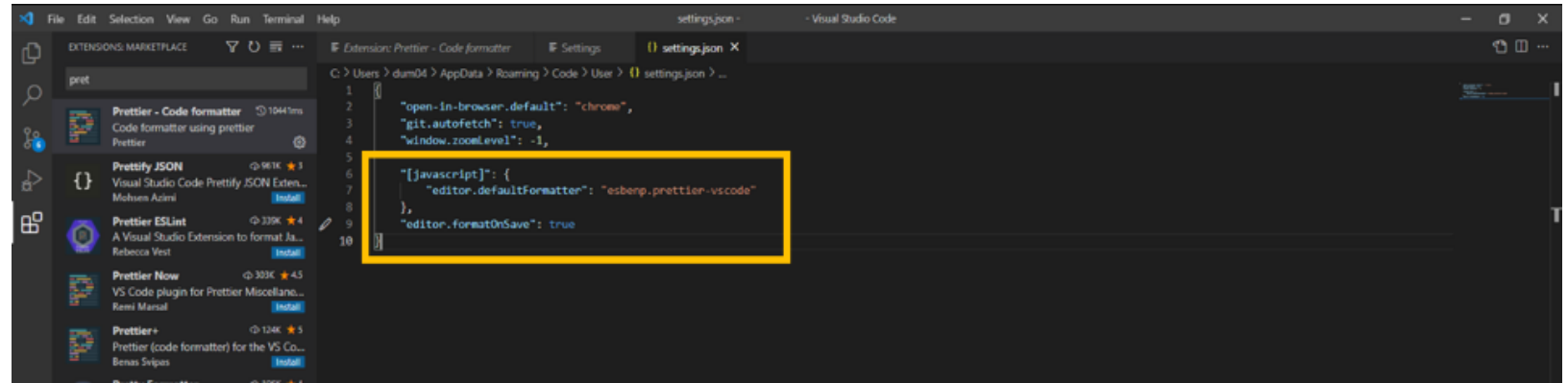
# Code Convention - Javascript



## Prettier - Prettier 설치하기

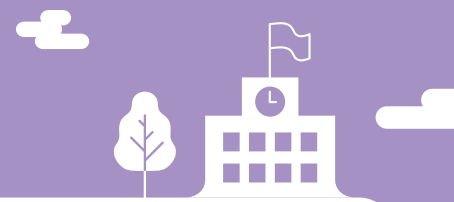
settings.json에서

기본 설정하기



```
{
  ...
  "[javascript]": {
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  },
  "editor.formatOnSave": true
}
```

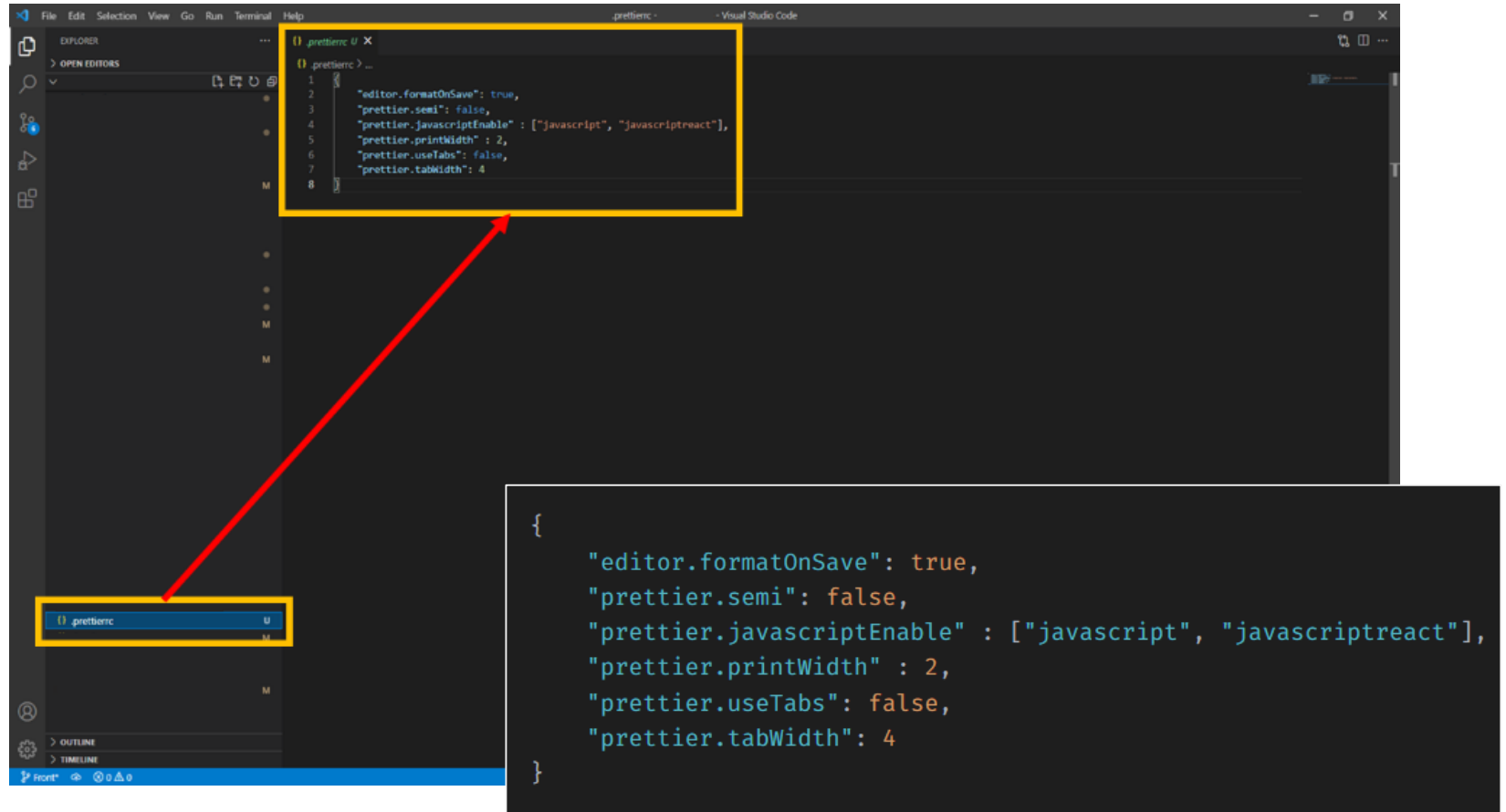
# Code Convention - Javascript



## Prettier - 커스텀 Prettier 적용하기(.prettierrc)

**.prettierrc** 파일 만들고

옵션 커스텀하기



# Code Convention - Javascript



## Prettier - 커스텀 Prettier 적용하기(.prettierrc)

### Prettier에서 설정 가능한 전체 옵션들

```
{
  "arrowParens": "avoid", // 화살표 함수 괄호 사용 방식
  "bracketSpacing": false, // 객체 리터럴에서 괄호에 공백 삽입 여부
  "endOfLine": "auto", // EoF 방식, OS별로 처리 방식이 다름
  "htmlWhitespaceSensitivity": "css", // HTML 공백 감도 설정
  "jsxBracketSameLine": false, // JSX의 마지막 `>`를 다음 줄로 내릴지 여부
  "jsxSingleQuote": false, // JSX에 single quote 사용 여부
  "printWidth": 80, // 줄 바꿈 할 폭 길이
  "proseWrap": "preserve", // markdown 텍스트의 줄바꿈 방식 (v1.8.2)
  "quoteProps": "as-needed" // 객체 속성에 quote 사용 방식
  "semi": true, // 세미콜론 사용 여부
  "singleQuote": true, // single quote 사용 여부
  "tabWidth": 2, // 탭 너비
  "trailingComma": "all", // 여러 줄을 사용할 때, 후행 쉼표 사용 방식
  "useTabs": false, // 탭 사용 여부
  "vueIndentScriptAndStyle": true, // Vue 파일의 script와 style 태그의 들여쓰기 여부 (v1.19.0)
  "parser": "babel", // 사용할 parser를 지정, 자동으로 지정됨
  "filepath": "", // parser를 유추할 수 있는 파일을 지정
  "rangeStart": 0, // 포매팅을 부분 적용할 파일의 시작 라인 지정
  "rangeEnd": Infinity, // 포매팅 부분 적용할 파일의 끝 라인 지정,
  "requirePragma": false, // 파일 상단에 미리 정의된 주석을 작성하고 Pragma로 포매팅 사용 여부 지정
  "insertPragma": false, // 미리 정의된 @format marker의 사용 여부 (v1.8.0)
  "overrides": [
    {
      "files": "*.json",
      "options": {
        "printWidth": 200
      }
    }
  ], // 특정 파일별로 옵션을 다르게 지정함, ESLint 방식 사용
}
```

# Code Convention - Typescript



## | Typescript 코드 컨벤션 적용

- ✓ 이전에 설치된 ESLint, Prettier 환경에서 TypeScript 설치 및 환경 설정 진행

# Code Convention - Typescript



## | TypeScript 설치

터미널에 다음 라인을 실행

```
# TypeScript 설치  
npm i -D typescript
```

-D 옵션은 ~~-save-dev~~와 같은 기능을 하는데, 해당 옵션을 붙여주면 `package.json` 파일의 `devDependencies`에 패키지가 저장  
개발 환경에서만 사용되는 패키지를 `devDependencies`에 설치 물론 옵션 없이 설치해도 무방

# Code Convention - Typescript



## | tsconfig.json 파일 설정

터미널에 다음 라인을 입력해주면 프로젝트에 **tsconfig.json** 파일이 생성

```
# tsconfig.json 파일 생성  
npx tsc --init
```

생성된 **tsconfig.json** 파일을 열어보면 엄청나게 많은 옵션들이 있는데

여기서는 프로젝트에 필요한 최소한의 옵션 몇 가지만 아래와 같이 설정함

```
{  
  "compilerOptions": {  
    "target": "es5",           // 컴파일 후 생성될 파일의 ECMAScript 버전  
    "module": "commonjs",     // 컴파일 후 생성될 파일이 사용하는 모듈 버전  
    "outDir": "dist/",        // 파일이 생성될 폴더  
    "esModuleInterop": true  // 'require'와 'import' 호환  
  },  
  "include": ["src/*.ts"]     // 사용할 폴더 및 파일  
}
```



# Code Convention - Typescript



## | TypeScript 적용

프로젝트의 app.js 파일 이름을 app.ts 파일로 변경해줍니다. 그리고 다음과 같이 1번째 줄을 import 문으로 변경

```
// 기존 코드: const express = require('express');  
import express from 'express'; // express 불러오기
```

변경하는 이유는, require을 사용할 경우 express의 타입 추론이 any가 되기 때문임.

이어서 package.json 파일의 script 항목에 다음과 같은 줄을 추가

```
"scripts": {  
  // .ts 파일 컴파일 후 생성된 .js 파일을 실행  
  "start": "npx tsc && node dist/app.js"  
}
```

npx tsc는 .ts 파일을 .js 파일로 컴파일하겠다는 의미이고, 그 뒤 node 명령을 통해 dist 폴더에 생성된 app.js 파일을 실행하게 됩니다. 그리고 다음과 같이 서버를 실행하면 됩니다

```
# express app 실행  
npm start
```

# Code Convention - Typescript



## | ESLint 설정 - ESLint 설치

npm을 통해 ESLint를 설치 (이미 설치 되어 있다면 생략)

```
# ESLint 설치  
npm i -D eslint
```

# Code Convention - Typescript



## ESLint 설정 - .eslintrc.json 파일 설정

터미널에서 다음 명령을 실행

```
# .eslintrc.json 파일 생성 및 설정  
npx eslint --init
```

tsconfig.json 파일을 생성할 때와는 다르게 여러 가지

사항을 물어보는데, 오른쪽과 같이 설정

설정이 모두 끝나면 해당 설정에 필요한 패키지들을

자동으로 설치함

@typescript-eslint/eslint-plugin, @typescript-eslint/parser

패키지들이 설치됨

직접 eslintrc.json 파일을 생성할 경우에는 수동으로 설치

- How would you like to use ESLint?  
**To check syntax and find problems**
- What type of modules does your project use?  
**JavaScript modules (import/export)**
- Which framework does your project use?  
**None of these**
- Does your project use TypeScript? (y/N)  
**y**
- Where does your code run? (Press space to select, a to toggle all, i to invert selection)  
**Node**
- What format do you want your config file to be in? (Use arrow keys)  
**JSON**
- The config that you've selected requires the following dependencies:  
@typescript-eslint/eslint-plugin@latest @typescript-eslint/parser@latest  
Would you like to install them now with npm? (Y/n)  
**Y**

# Code Convention - Typescript



## | ESLint 설정 - .eslintrc.json 파일 설정

패키지가 모두 설치되고 나면 프로젝트에

`.eslintrc.json` 파일이 생성됨

생성된 `.eslintrc.json` 파일에 오른쪽 같은 설정을 추가

`extends`는 ESLint에 적용할 규칙들을 정의해주는 곳으로

나중에 정의된 옵션일수록 높은 우선 순위를 가짐

`parserOptions.project`는 타입 정보를 필요로 하는

규칙들을 사용하고 싶으면 설정해야 하는 속성으로

프로젝트의 `tsconfig.json` 파일 위치를 작성

`ignorePatterns`는 ESLint가 무시할 폴더, 파일을

적어주는 옵션

```
{
  ...
  "extends": [
    "eslint:recommended",
    "plugin:@typescript-eslint/eslint-recommended",
    "plugin:@typescript-eslint/recommended",
    "plugin:@typescript-eslint/recommended-requiring-type-checking"
  ],
  "parserOptions": {
    "project": "./tsconfig.json",
    "ecmaVersion": 2018,
    "sourceType": "module"
  },
  "ignorePatterns": ["dist/", "node_modules/"]
  ...
}
```

# Code Convention - Typescript



## ESLint 설정 - ESLint 적용 확인

설정이 끝나고 나면 **app.ts** 파일에 잘못된 문법을 가진 코드를 적어, 제대로 오류가 발생하는지 확인

```
25  const a: String = "abc";
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Filter. E.g.: text, \*\*/

✓ TS app.ts src 2

- ⊗ Don't use 'String' as a type. Use string instead eslint(@typescript-eslint/ban-types) [25, 10]
- ⚠ 'a' is assigned a value but never used. eslint(@typescript-eslint/no-unused-vars) [25, 7]

위와 같이 에러 및 경고가 잘 생성되면 ESLint 설정이 잘 끝난 것임

# Code Convention - Typescript



## | Prettier 설정 - Prettier 설치

npm을 통해 패키지를 설치 (이미 설치되어 있다면 생략)

```
# Prettier 설치  
npm i -D prettier
```

ESLint와 Prettier가 서로 충돌이 되지 않고 잘 작동하도록 **eslint-config-prettier**와 **eslint-plugin-prettier**를 설치

```
# ESLint 호환  
npm i -D eslint-config-prettier eslint-plugin-prettier
```

**eslint-config-prettier**는 Prettier와 충돌되는 ESLint 규칙들을 무시하는 설정이고, **eslint-plugin-prettier**는 Prettier를 사용해 포맷팅을 하도록 ESLint 규칙을 추가하는 플러그인임

# Code Convention - Typescript



## Prettier 설정 - .eslintrc.json과 .prettierrc.json 파일 설정

프로젝트 폴더에 .prettierrc.json 파일을 생성

이 파일에서 설정할 수 있는 다양한 옵션들이 있는데, Prettier-Options를 참고해서 원하시는 스타일로 설정

아래 설정은 참고

```
{
  "printWidth": 80,           // 한 줄의 라인 수
  "tabWidth": 2,              // tab의 너비
  "useTabs": false,           // tab 사용 여부
  "semi": true,               // ; 사용 여부
  "singleQuote": true,        // 'string' 사용 여부
  "quoteProps": "consistent", // 객체 property의 따옴표 여부
  "trailingComma": "es5",      // 끝에 , 사용 여부
  "bracketSpacing": true,      // Object literal에 띄어쓰기 사용 여부 (ex: { foo: bar })
  "arrowParens": "always",     // 함수에서 인자에 괄호 사용 여부 (ex: (x) => y)
  "endOfLine": "lf"           // 라인 엔딩 지정
}
```

# Code Convention - Typescript



## Prettier 설정 - .eslintrc.json과 .prettierrc.json 파일 설정

ESLint도 설정을 해줘야 합니다. .eslintrc.json 파일에서 다음과 같이 설정을 변경해줍니다.

```
{
  ...
  "extends": [
    "eslint:recommended",
    "plugin:@typescript-eslint/eslint-recommended",
    "plugin:@typescript-eslint/recommended",
    "plugin:@typescript-eslint/recommended-requiring-type-checking",
    "plugin:prettier/recommended",
    "prettier/@typescript-eslint"
  ],
  ...
}
```

ESLint에서 Prettier를 extends에 사용하기 위해 `plugin:prettier/recommended`, `prettier/@typescript-eslint`를 추가  
extends에 추가할 때 유의할 점은, 아래에 있는 것일수록 우선 순위가 높습니다.  
따라서 Prettier 관련 규칙들이 위의 코드처럼 목록의 최하단에 위치해야 합니다.

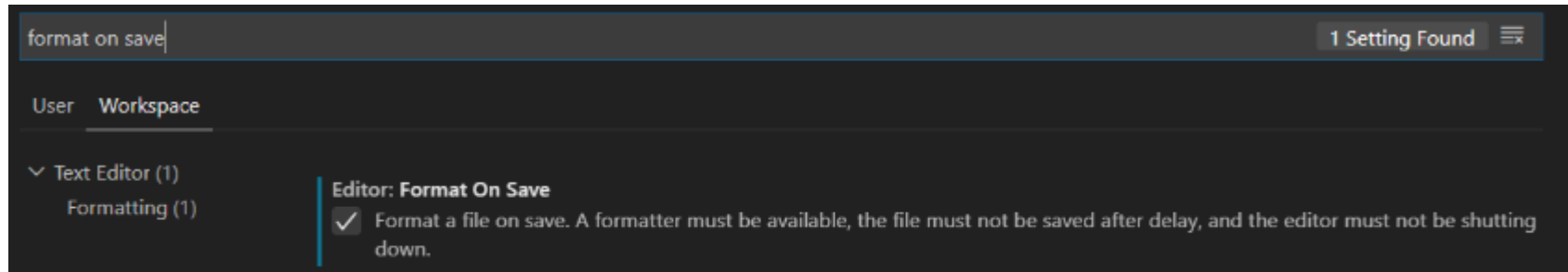


# Code Convention - Typescript



## Prettier 설정 - VSCode Format On Save 설정

Preferences > Settings > Workspace > Editor: Format On Save 옵션에 체크해주면,  
파일을 저장할 때마다 Prettier의 설정값에 맞춰 자동으로 파일이 포매팅이 됩니다.

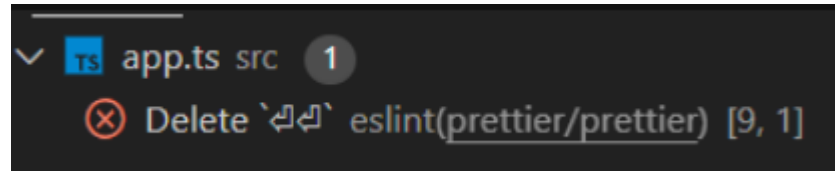


# Code Convention - Typescript



## | Prettier 설정 - Prettier 적용 확인

코드에 엔터를 여러 번 입력하면 아래 스크린샷처럼 ESLint에서 Prettier 관련 에러가 나타납니다.

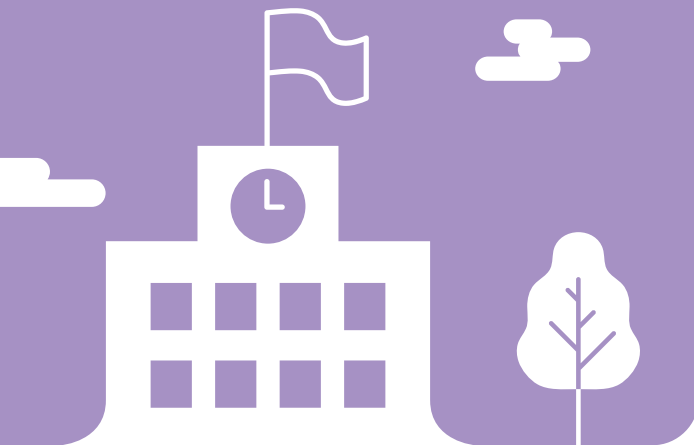


그리고 Ctrl+s를 눌러 저장을 했을 때, Enter가 자동으로 없어진다면 설정이 잘 된 것임

# 코드 컨벤션 적용하기

I Python

II C/C++





## | Python

- ✓ Python Code Convention 적용 Guide
- ✓ IDE - VSCode



## | Python Formatter

1. black tool 을 사용하여 다양한 python IDE 에서 formatter 사용 가능

Github : <https://github.com/psf/black>

가이드 문서 : <https://black.readthedocs.io/en/stable/integrations/editors.html>

2. Google code style formatter 사용



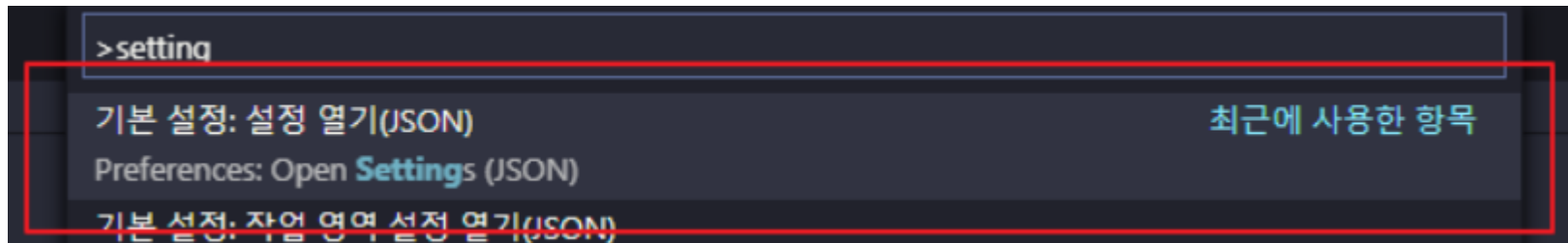
## | Python : VSCode 설정

### 1. flake8, black 설치

`pip install flake8`

`pip install black`

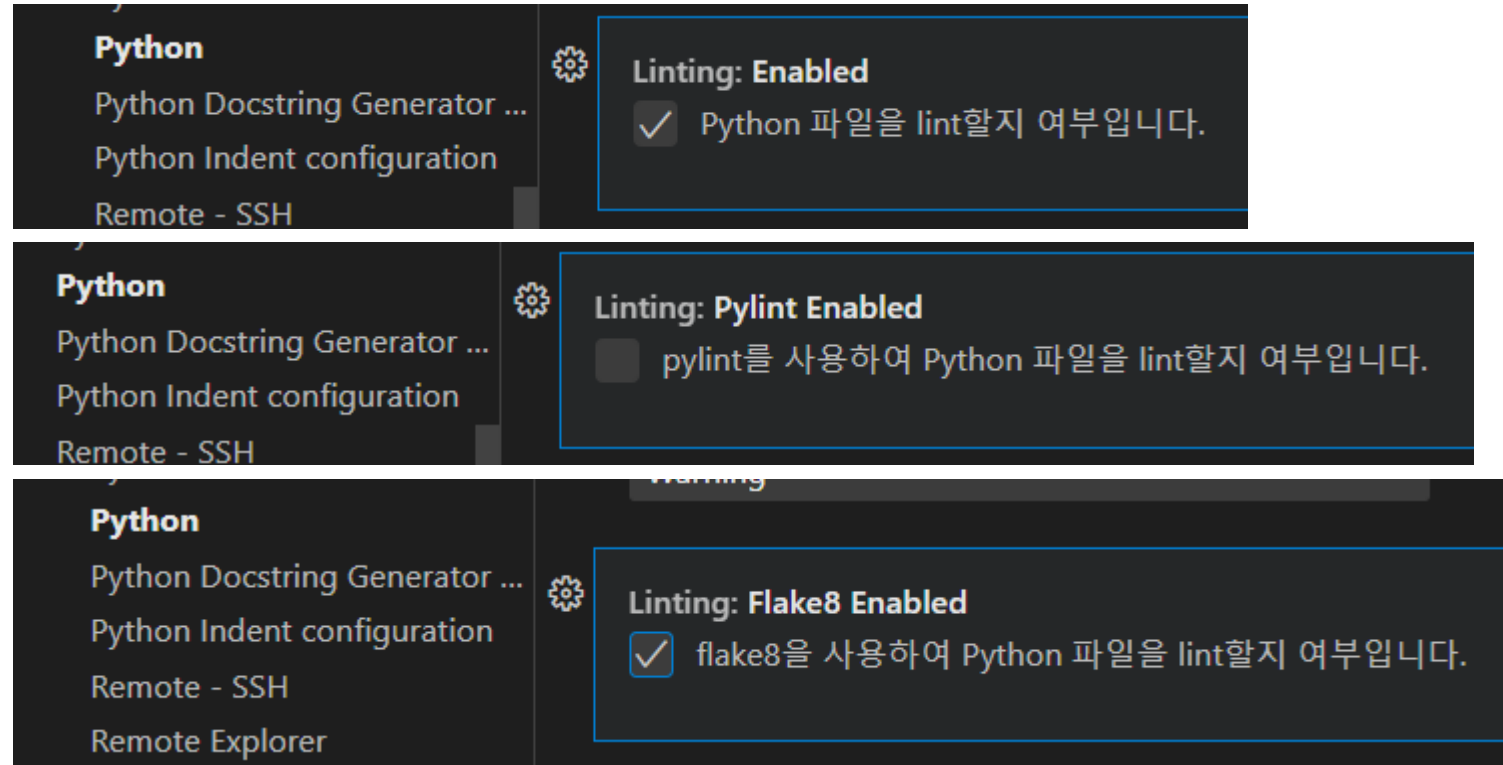
### 2. 명령 팔레트(Ctrl + Shift + p) 실행 후 사용자 설정 열기





## | Python : VSCode 설정

### 3. 사용자 설정





## | Python : VSCode 설정

### 3. 사용자 설정

The screenshot shows the VS Code settings interface. On the left, the 'Python' category is selected in the settings sidebar. The main panel displays the 'Formatting: Provider' setting. The description states: '서식을 위한 공급자입니다. 가능한 옵션에는 'autopep8', 'black' 및 'yapf'가 있습니다.' (This is the provider for formatting. Possible options are 'autopep8', 'black', and 'yapf'). A dropdown menu is open, showing the following options: 'autopep8' (selected), 'autopep8 기본값' (autopep8 default), 'black', 'none', and 'yapf'. Below this, the 'Editor: Default Formatter' setting is visible, with the description: '다른 모든 포맷터 설정보다 우선하는 기본 포맷터를 정의합니다. 포맷터를 제공하는 확장 프로그램의 식별자 여야합니다.' (Defines the default formatter that takes precedence over all other formatter settings. It must be the identifier of the extension that provides the formatter). A dropdown menu for this setting shows 'Python' selected, with 'Python ms-python.python' listed below it.

**Python**

- Python Docstring Generator ...
- Python Indent configuration
- Remote - SSH
- Remote Explorer
- Simple Browser
- TypeScript
- Vetur

**Formatting: Provider**

서식을 위한 공급자입니다. 가능한 옵션에는 'autopep8', 'black' 및 'yapf'가 있습니다.

- autopep8
- autopep8 기본값
- black**
- none
- yapf

**Editor: Default Formatter**

다른 모든 포맷터 설정보다 우선하는 기본 포맷터를 정의합니다. 포맷터를 제공하는 확장 프로그램의 식별자 여야합니다.

- Python
- Python ms-python.python





## | Python : VSCode 설정

4. setting.json 파일을 직접 열어  
오른쪽 그림과 같이 추가 또는 수정.

```
{  
    "editor.formatOnSave": true,  
    "python.linting.enabled": true,  
    "python.linting.pylintEnabled": false,  
    "python.linting.flake8Enabled": true,  
    "python.linting.lintOnSave": true,  
    "python.formatting.provider": "black",  
    "[python]": {  
        "editor.defaultFormatter": "ms-python.python"  
    }  
}
```



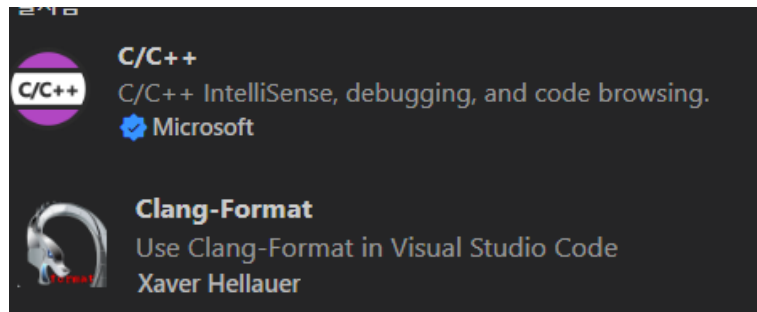
## | C++

- ✓ C++ Code Convention 적용 Guide
- ✓ IDE - VSCode

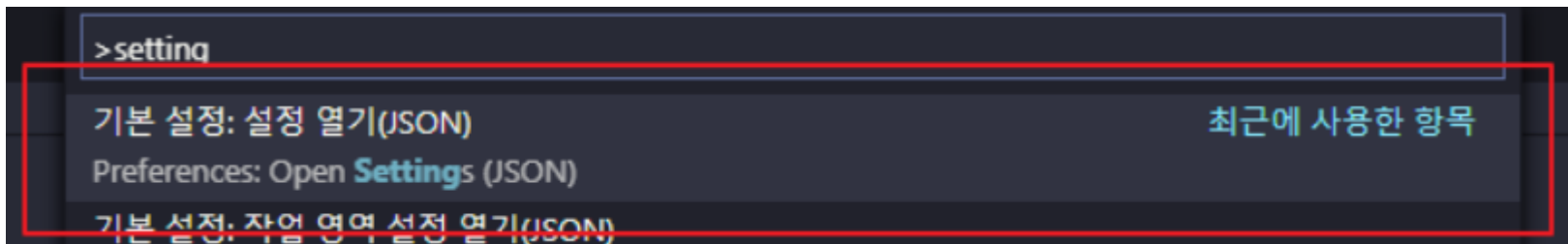


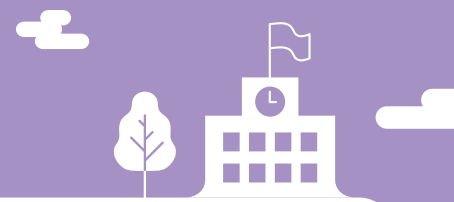
## | VSCode C++ Fomatter

1. 2가지 Extention 을 설치합니다.



2. 명령 팔레트(Ctrl + Shift + p) 실행 후 사용자 설정 열기





## C++ formatter

3. VSCode 설정 (setting.json)을  
오른쪽 그림과 같이 추가 또는 수정.

```
{  
  "editor.formatOnSave": true,  
  "editor.defaultFormatter": "ms-vscode.cpptools",  
  "C_Cpp.clang_format_fallbackStyle": "Google",  
}
```

