



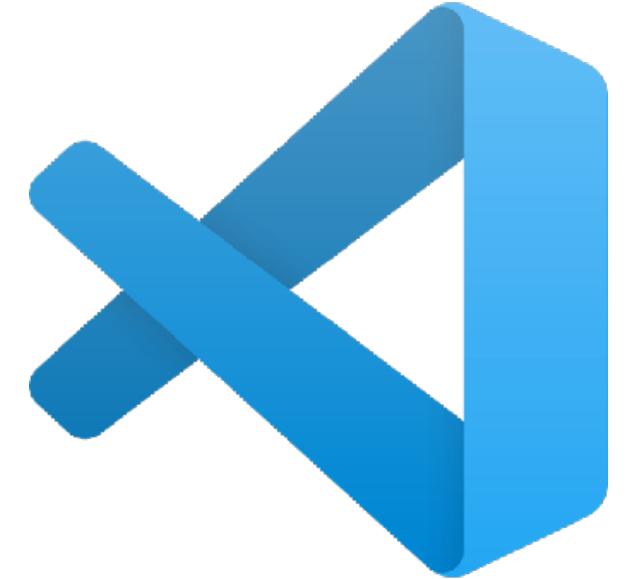
PLURALSIGHT



salesforce

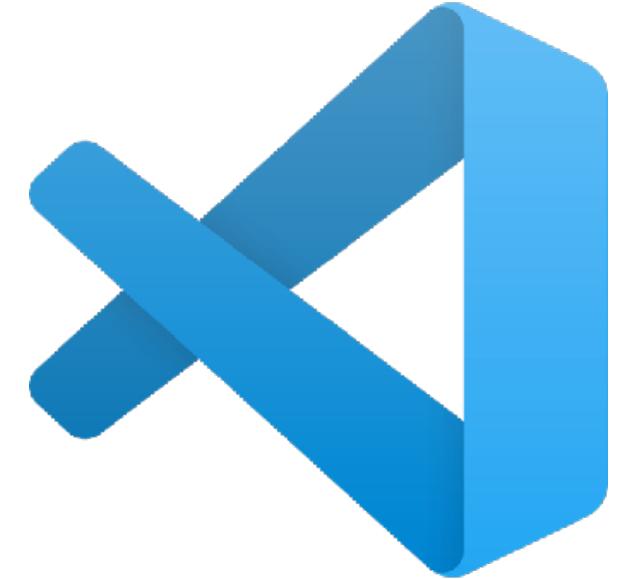


Introduction to VSCode



Visual Studio Code Benefits

- Frictionless builds
- IntelliSense completion
- Easy Customization
- Immense Community of Extensions



Visual Studio Architecture

- Built on Electron
- Uses web technologies such as JavaScript and Node.js
- Operates at the speed and flexibility of native apps
- VS Code uses a tools service architecture that enables it to integrate with many of the same technologies that power Visual Studio



Online Environments



Gitpod.io

- Create VSCode Environments Online
- Integrated with Github, Gitlab, Bitbucket, or other repositories
- Based on Cloud Development Environments (CDEs), which are on-demand development environments pre-configured with all the necessary tools, libraries, and dependencies for writing and reviewing code.



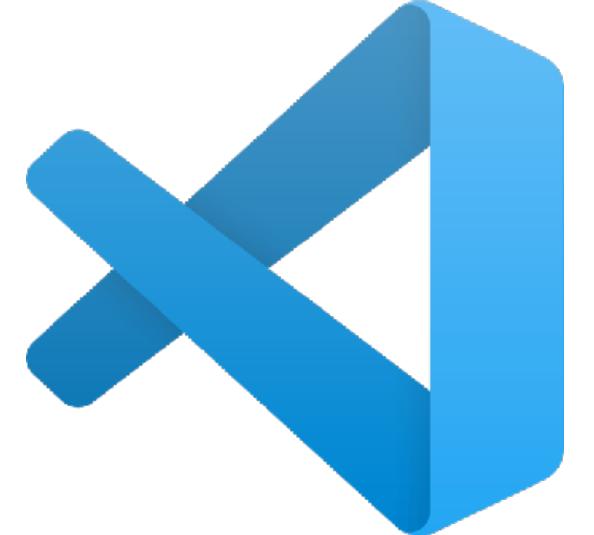
Github Codespaces

- Create VSCode Environments Online, same as Gitpod.io
- Integrated directly with Github, as opposed to [gitpod.io](#) which can use other platforms
- Seamless integration with Github
- Has templates that you can get started on immediately including Jupyter Notebooks



Development Containers

- Github Codespaces uses Dev Containers to ensure that proper tools are installed
- A development container (or dev container for short) allows you to use a container as a full-featured development environment.
- It can be used to run an application, to separate tools, libraries, or runtimes needed for working with a codebase, and to aid in continuous integration and testing.
- Dev containers can be run locally or remotely, in a private or public cloud, in a variety of supporting tools and editors.



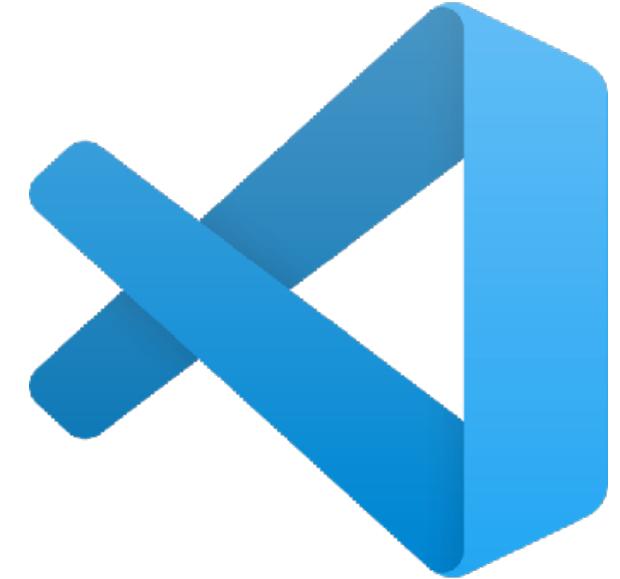
Trying out the Environments



- In this lab we will try out either Java, JavaScript, Python projects in various environments
 - Github's Codespaces
 - gitpod.io

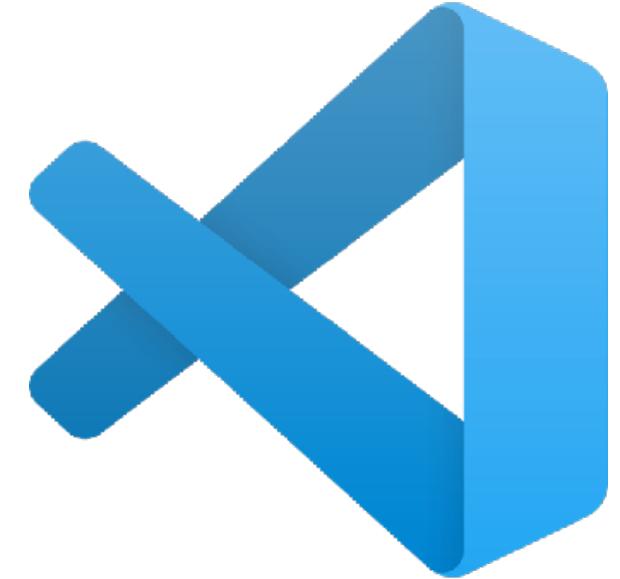


Language Servers



Language Servers

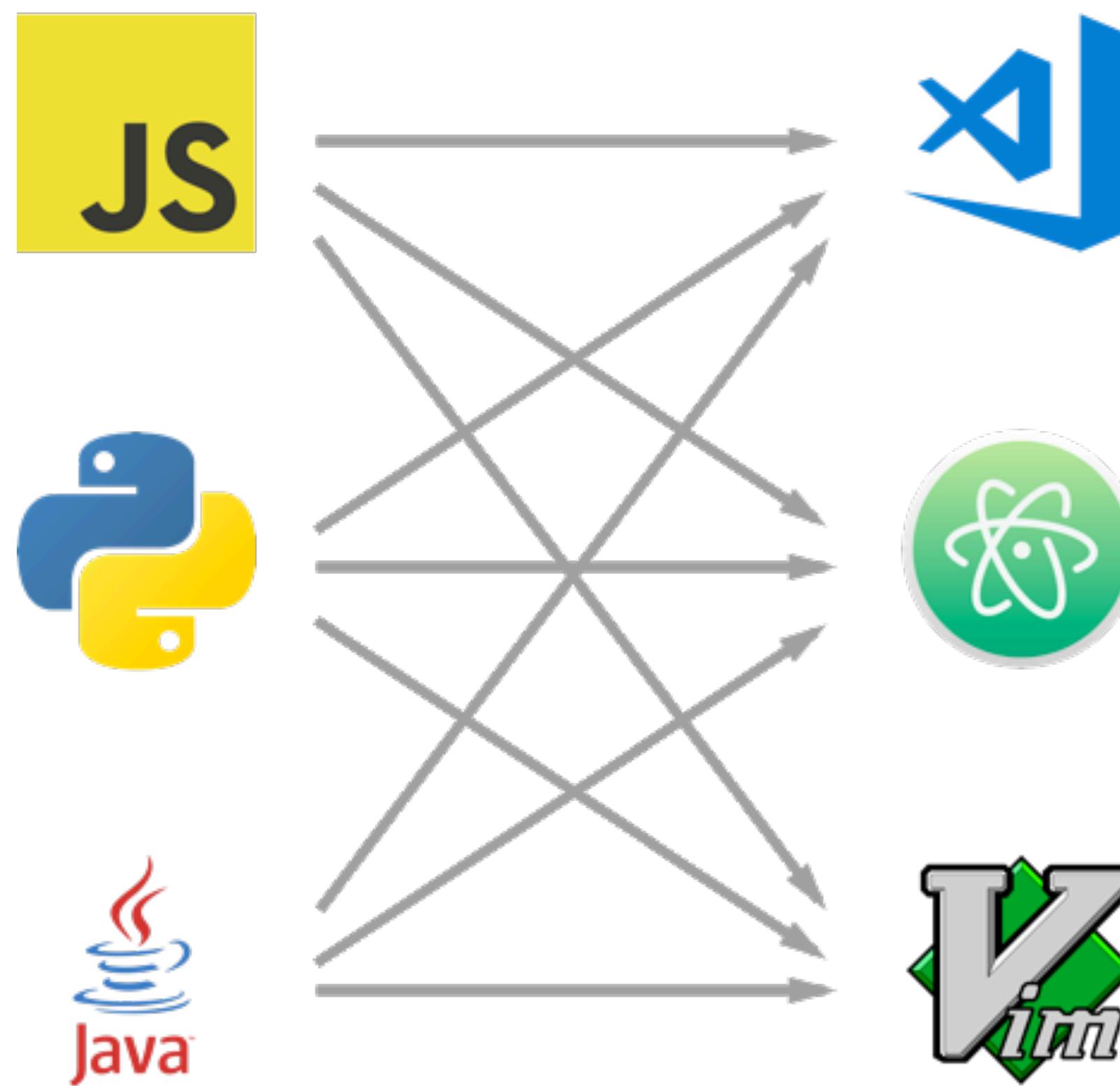
- Language Server is a special kind of Visual Studio Code extension that powers the editing experience for many programming languages.
- Solves the common problem with implementing different languages in an IDE
 - Language Servers are usually implemented in their native programming languages. VSCode is a Node.js runtime
 - Language features can be resource intensive. VSCode wanted to ensure that VS Code's performance remains unaffected
 - Integrating multiple language toolings with multiple code editors could involve significant effort. From language toolings' perspective, they need to adapt to code editors with different APIs. This causes an $M \times N$ issue. M languages in N code editors



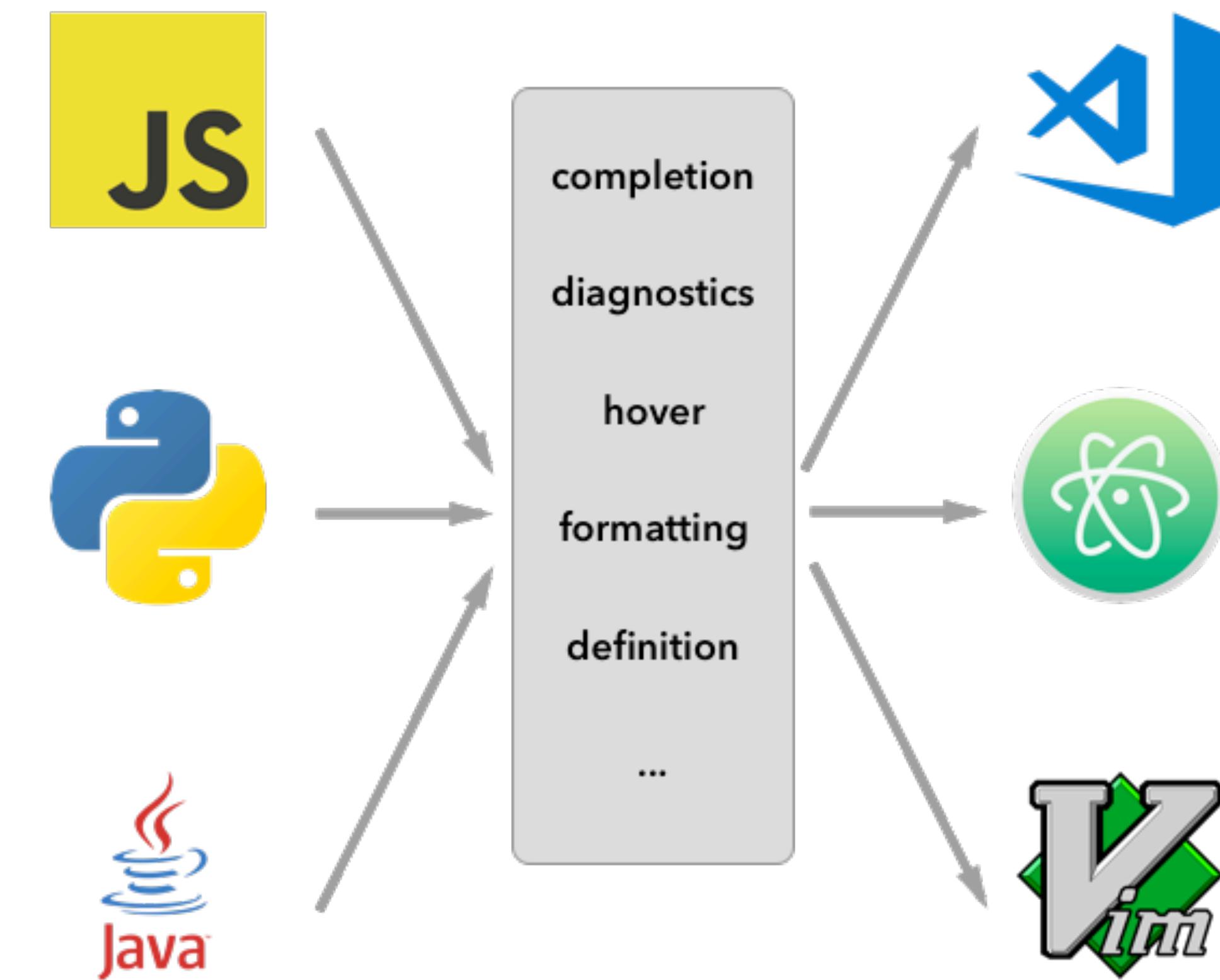
Language Server Protocol

- Microsoft specified [Language Server Protocol](#), which standardizes the communication between language tooling and code editor.
- Language Servers can be implemented in any language and run in their own process to avoid performance cost, as they communicate with the code editor through the Language Server Protocol.
- Any LSP-compliant language toolings can integrate with multiple LSP-compliant code editors, and any LSP-compliant code editors can easily pick up multiple LSP-compliant language toolings.

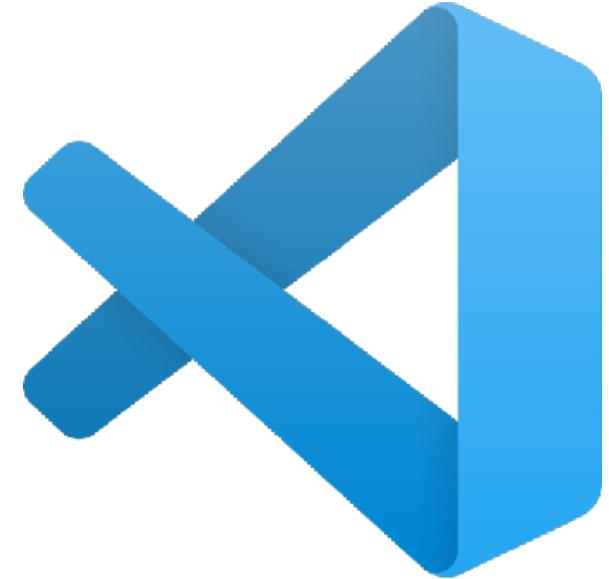
NO LSP



LSP

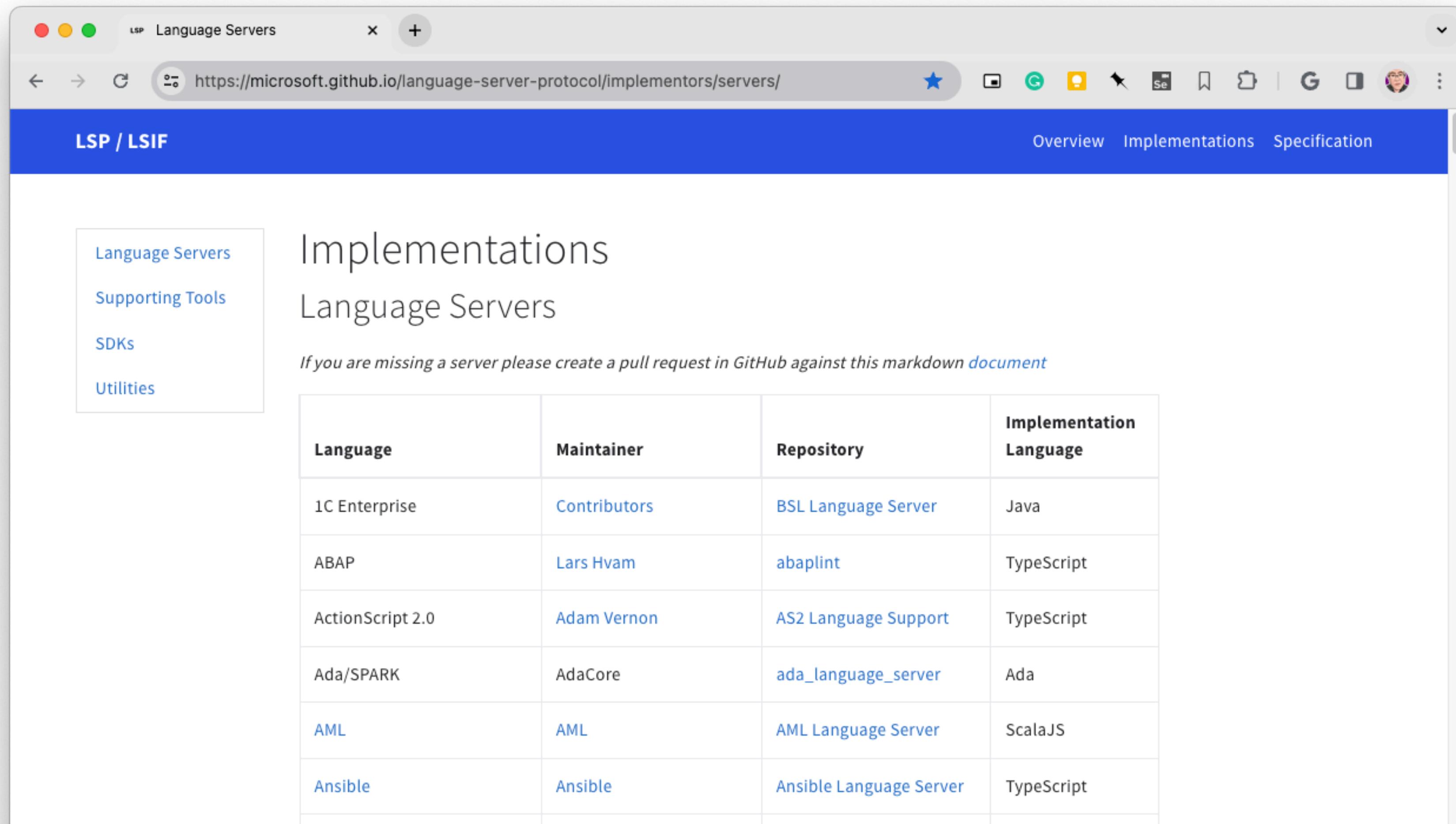


View the Language Server Website on [how to implement a Language Server](#)



List of Language Servers

<https://microsoft.github.io/language-server-protocol/implementors/servers/>

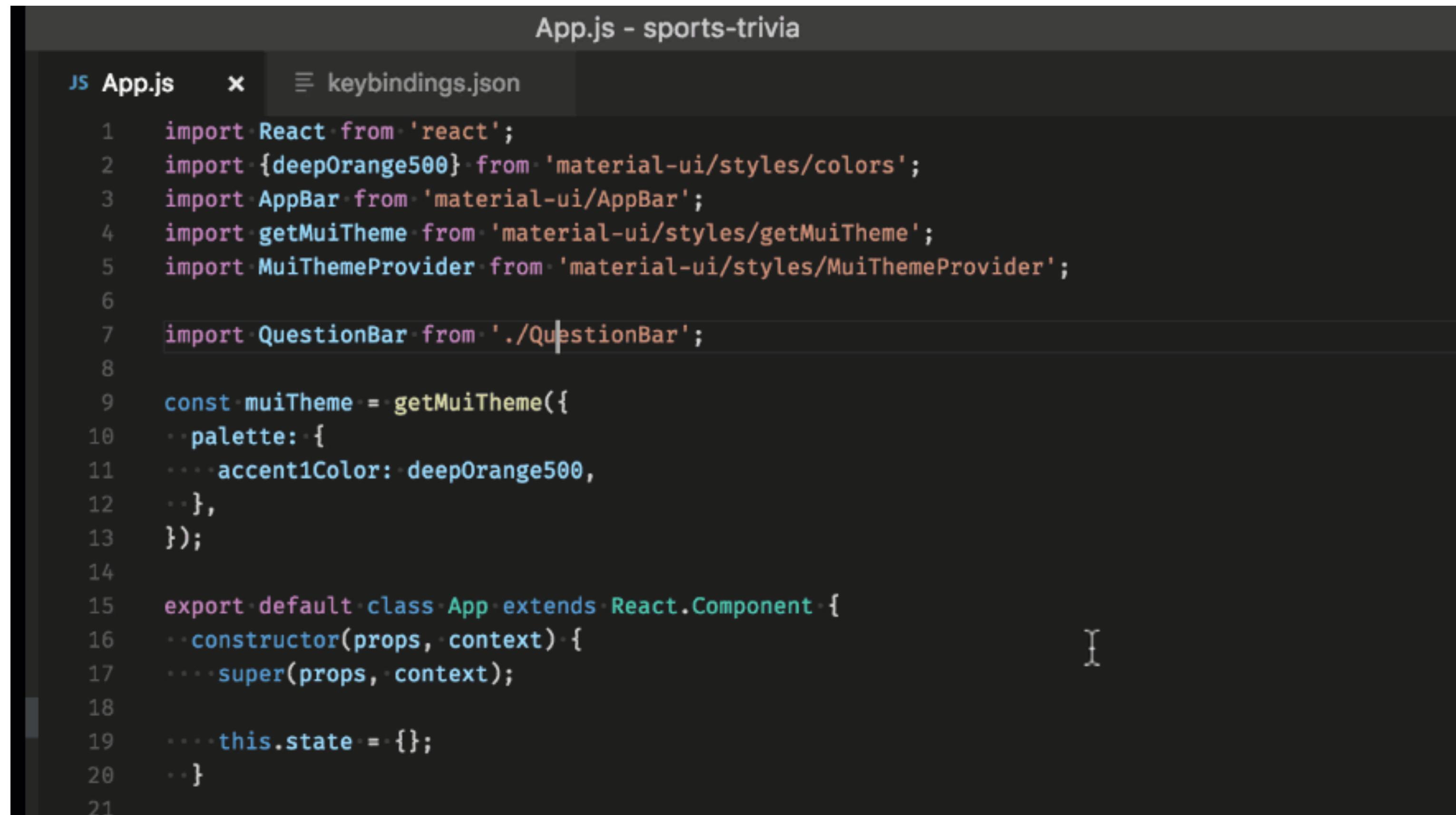
A screenshot of a web browser displaying the Microsoft GitHub page for Language Servers. The page has a blue header with the text "LSP / LSIF". Below the header, there's a sidebar with links to "Language Servers", "Supporting Tools", "SDKs", and "Utilities". The main content area is titled "Implementations" and "Language Servers". It includes a note: "If you are missing a server please create a pull request in GitHub against this markdown document". A table lists various language servers with their maintainers, repositories, and implementation languages.

Language	Maintainer	Repository	Implementation Language
1C Enterprise	Contributors	BSL Language Server	Java
ABAP	Lars Hvam	abaplint	TypeScript
ActionScript 2.0	Adam Vernon	AS2 Language Support	TypeScript
Ada/SPARK	AdaCore	ada_language_server	Ada
AML	AML	AML Language Server	ScalaJS
Ansible	Ansible	Ansible Language Server	TypeScript



The Ubiquitous and Essential Command Palette

Command Palette



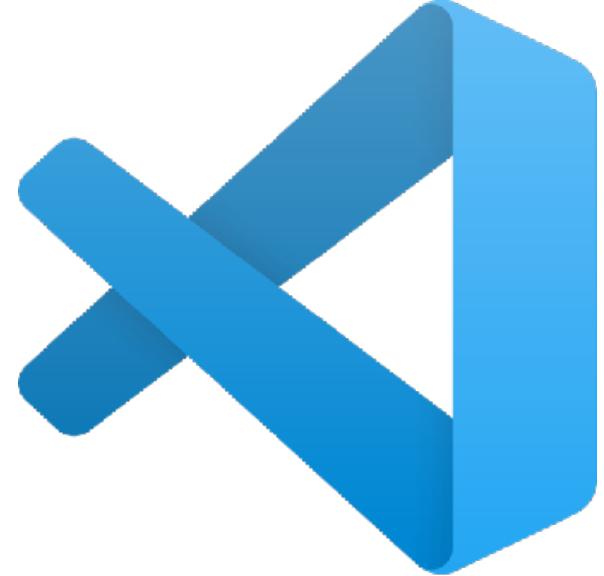
The screenshot shows the VS Code interface with the title bar "App.js - sports-trivia". A floating Command Palette is displayed, listing key combinations such as ⌘P (Windows: Ctrl+Shift+P) and F1. The main editor area shows a portion of an App.js file with code related to React and Material-UI.

```
JS App.js    x  keybindings.json
1 import React from 'react';
2 import {deepOrange500} from 'material-ui/styles/colors';
3 import AppBar from 'material-ui/AppBar';
4 import getMuiTheme from 'material-ui/styles/getMuiTheme';
5 import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';
6
7 import QuestionBar from './QuestionBar';
8
9 const muiTheme = getMuiTheme({
10   palette: {
11     accent1Color: deepOrange500,
12   },
13 });
14
15 export default class App extends React.Component {
16   constructor(props, context) {
17     super(props, context);
18
19     this.state = {};
20   }
21 }
```

The Command Palette (**⌘P**) (**CTRL+SHIFT+P**) or **F1** is your all-around import key command, it brings up all key combinations in case you didn't remember



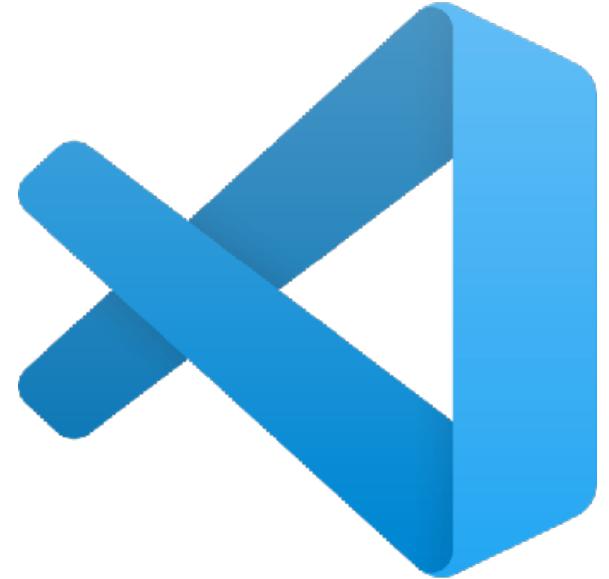
Opening and Managing Projects



Starting VS Code from the Terminal

```
> code .
```

- Ensure that you are in the root directory of your project
- This directory will typically have the build file, or at the minimum, where the project files are located
- If you need to open another project, run the above from a different folder. Yes, you will have multiple VSCode windows for different projects, that is normal.



Other VS Code Terminal Options

Open the current directory in the most recently used code window

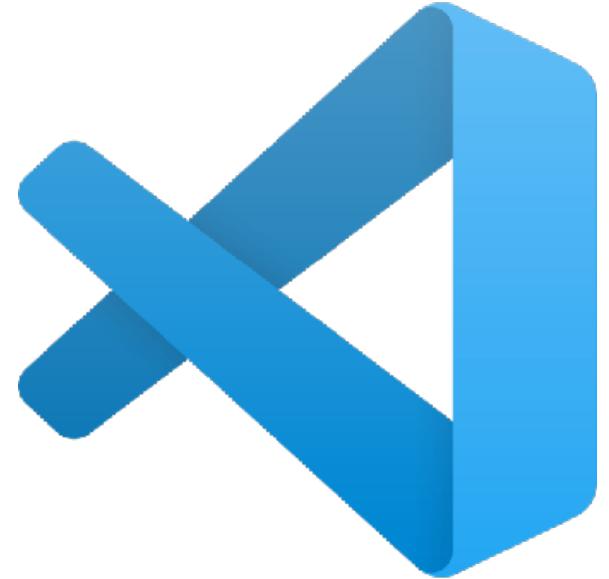
```
> code -r .
```

Create a new window

```
> code -n
```

Change the language

```
> code --locale=es
```



Other VS Code Terminal Options

Open a diff editor. Perfect for git

```
> code --diff <file1> <file2>
```

Open file at specific line and column

```
> code --goto <file:line[:character]>
> code --goto package.json:10:5
```

See Help Option

```
> code --help
```

Disable all extensions

```
> code --disable-extensions .
```



What if it isn't there?

A screenshot of the VS Code Command Palette. The input field at the top contains the text ">Install Code". Below the input field, a green bar displays the text "Shell Command: Install 'code' command in PATH" in white. To the right of this text is a gear icon. Below the green bar, there are several suggestions: ".NET Install Tool: Report an issue with the .NET Install Tool.", "Configure Display Language", "Remote-SSH: Kill VS Code Server on Host...", and "Rest Client: Generate Code Snippet". On the far right of the palette, there are three small icons: a file icon, a terminal icon, and a settings icon. The entire interface has a light gray background.

You can type in the Command Palette “**Install ‘code’ command in PATH**” to install that on your PATH

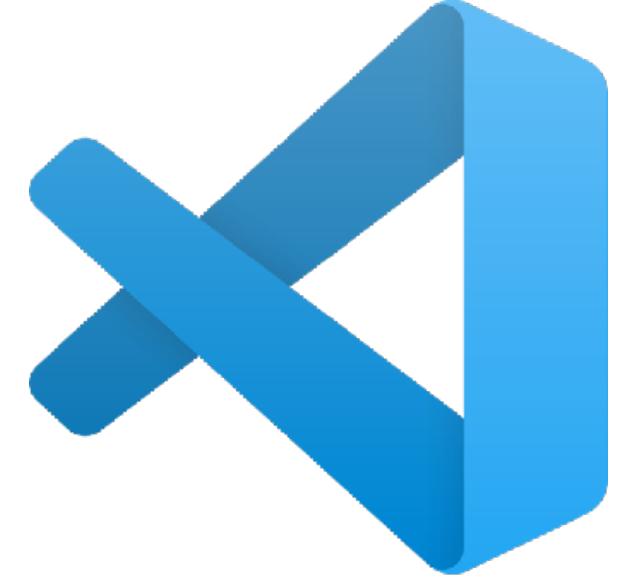


Workspaces



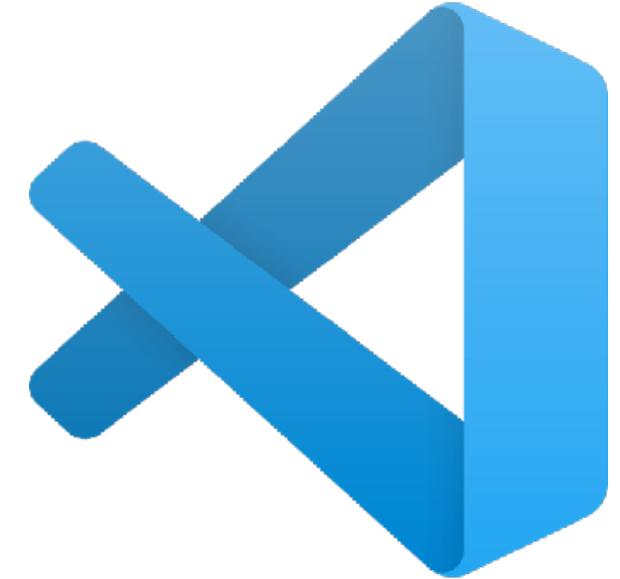
Workspaces

- A Visual Studio Code "workspace" is the collection of one or more folders that are opened in a VS Code window (instance)
- In most cases, you will have a single folder opened as the workspace.
- You can include more than one folder, using an advanced configuration called *Multi-root workspaces*



Workspaces vs. User Settings

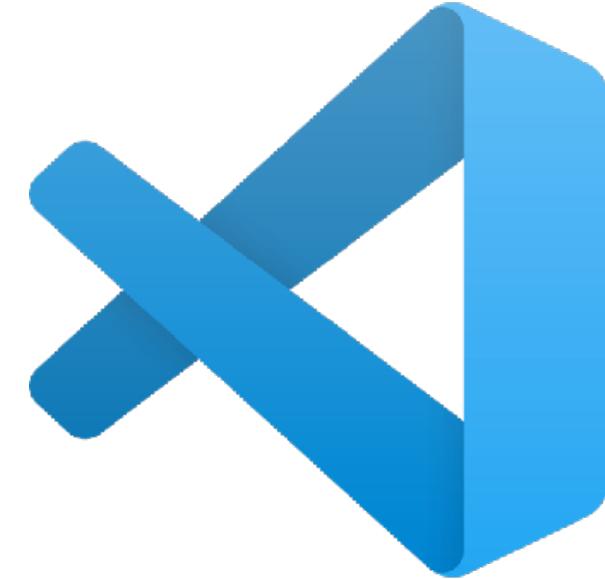
- **User Settings** - Settings that apply globally to any instance of VS Code you open.
- **Workspace Settings** - Settings stored inside your workspace and only apply when the workspace is opened.



Workspace Concepts

The concept of a workspace enables VS Code to:

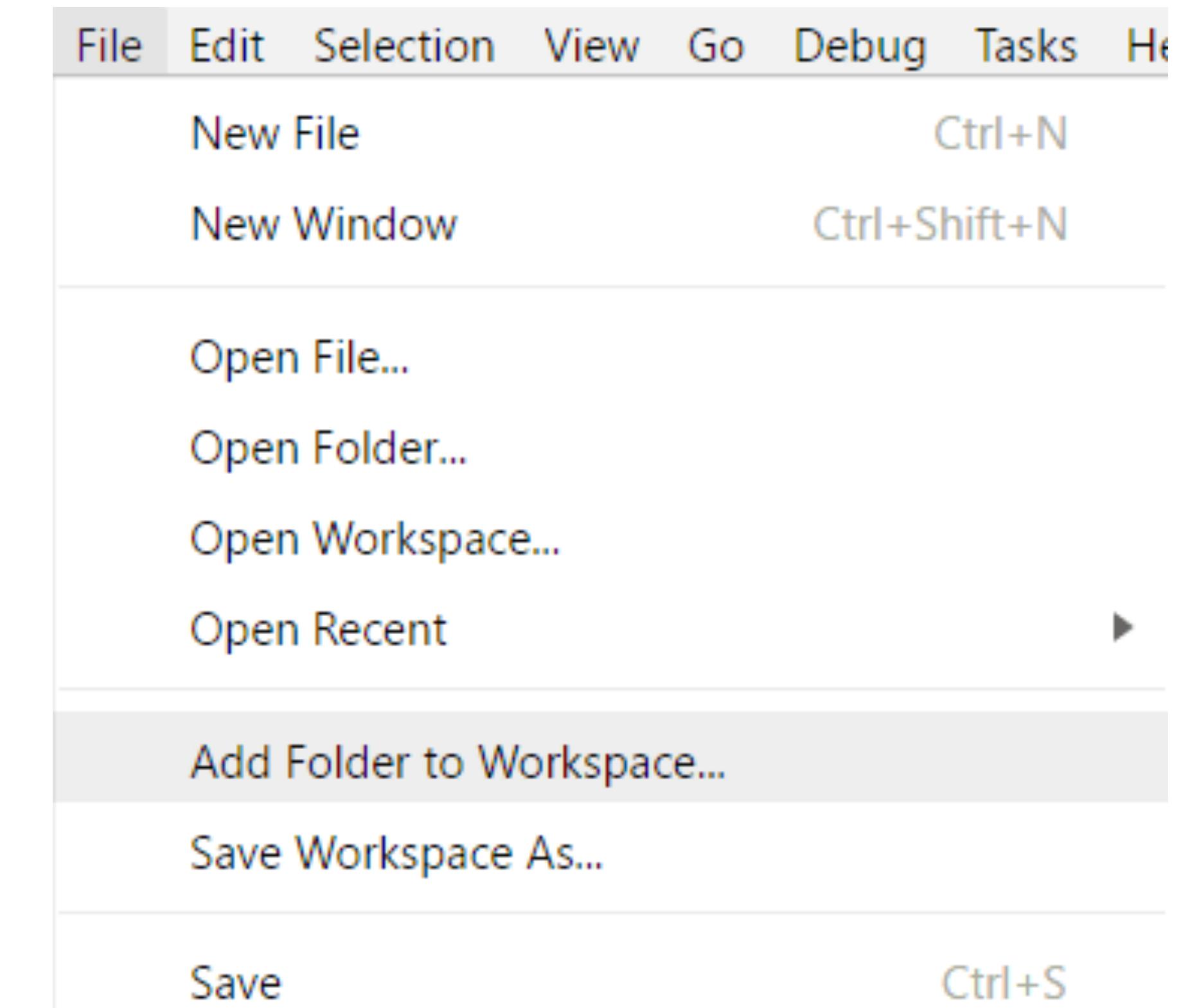
- Configure settings that only apply to a specific folder or folders but not others.
- Persist task and debugger launch configurations that are only valid in the context of that workspace.
- Store and restore UI state associated with that workspace (for example, the files that are opened).
- Selectively enable or disable extensions only for that workspace.



Adding Folder to Workspace

Multi-root workspaces have the following purposes:

- You can run multiple project folders in one session of VSCode.
- Typically this is done with several *related projects*
- There are multiple way to add a workspaces, one common way is **File > Add Folder to Workspace**





Adding Folder Alternatives

Other ways you can include other folders:

- Drag and Drop:
 - You can use drag and drop to add folders to a workspace. Drag a folder to the File Explorer to add it to the current workspace. You can even select and drag multiple folders.
 - Dropping a single folder into the editor region of VS Code will still open the folder in single folder mode. If you drag and drop multiple folders into the editor region, a new multi-root workspace will be created.
- Opening multiple folders with your platform's native file open dialog will create a multi-root workspace.
- You also create multiroot workspaces with: `code --add vscode vscode-docs`



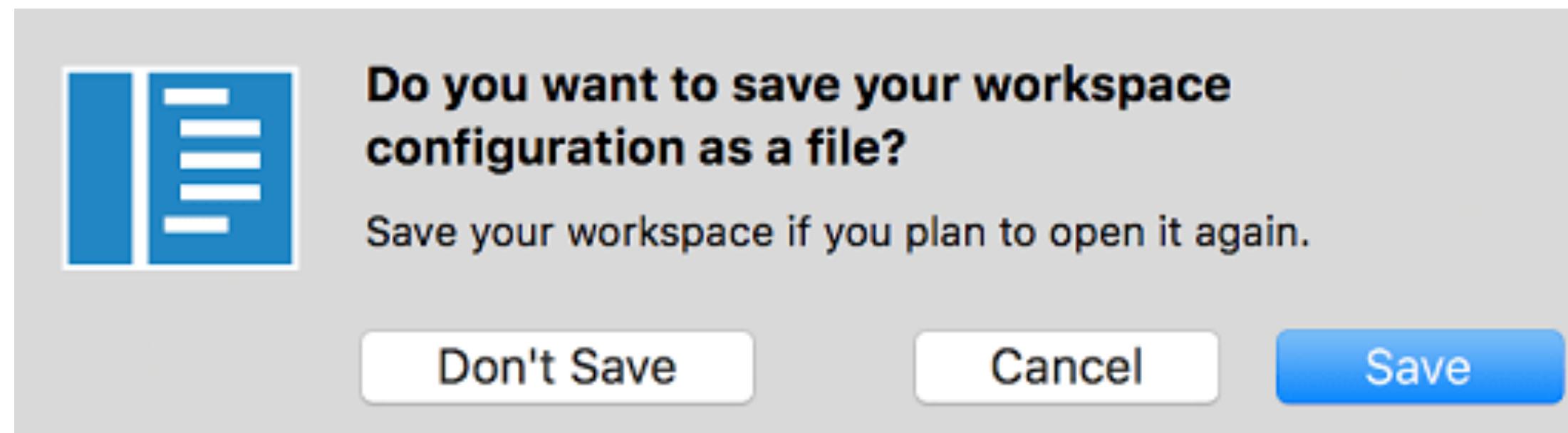
Removing Folders

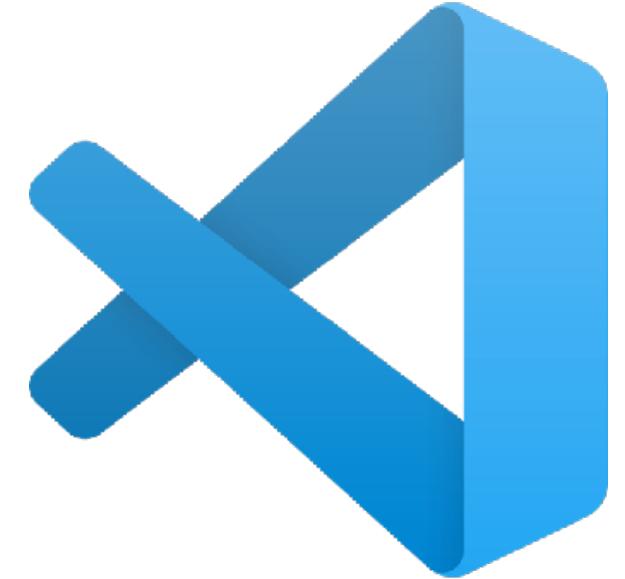
- You can remove a folder from a Workspace with the Remove Folder from Workspace context menu command.



Workspace Files

- When you add multiple folders, they are initially placed in a Workspace titled **UNTITLED WORKSPACE** and that name will remain until you save the workspace.
- You do not need to save a Workspace until you want to have it in a permanent location, for example, on your Desktop, or somewhere else meaningful
- Untitled Workspaces are present as long as the VS Code instance they are using is open.
- Once you completely close an instance with an untitled workspace, you will be asked to save it if you plan to open it again in the future:





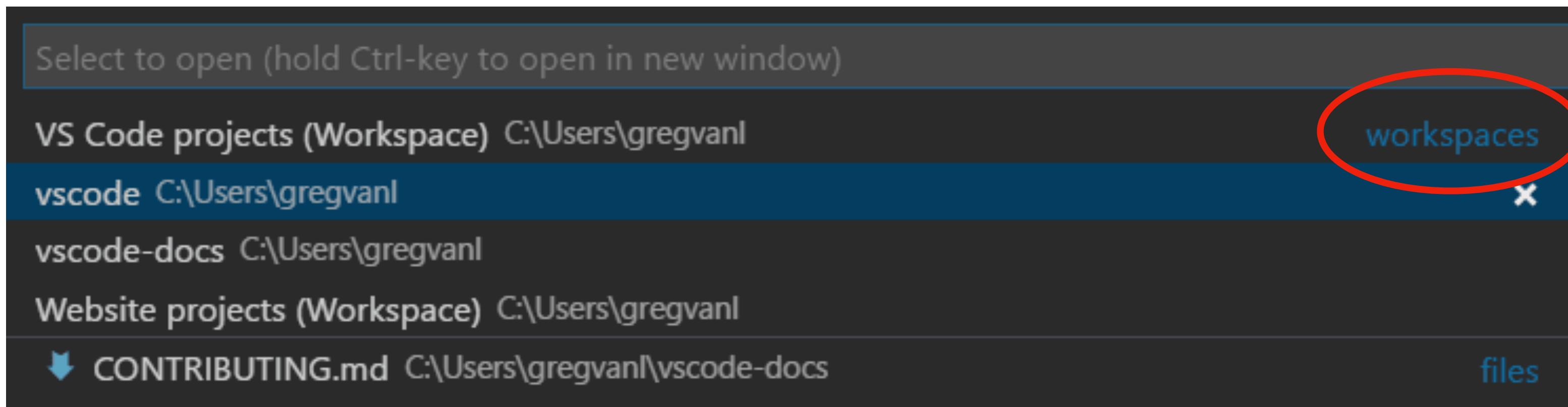
Workspace Files Details

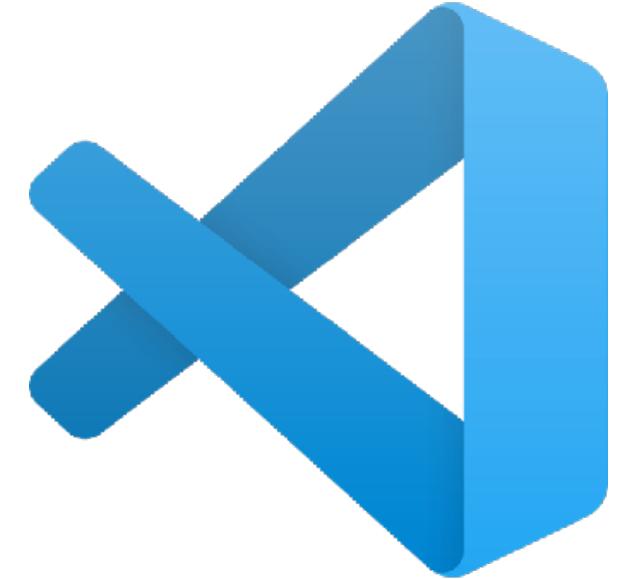
- When you save your workspace, it will create a `.code-workspace` file and the file name will be displayed in the File Explorer.
- If you want to move your Workspace file to a new location, you can use the **File > Save Workspace As** command, which will automatically set the correct folder paths relative to the new Workspace file location.



Opening Workspace Files

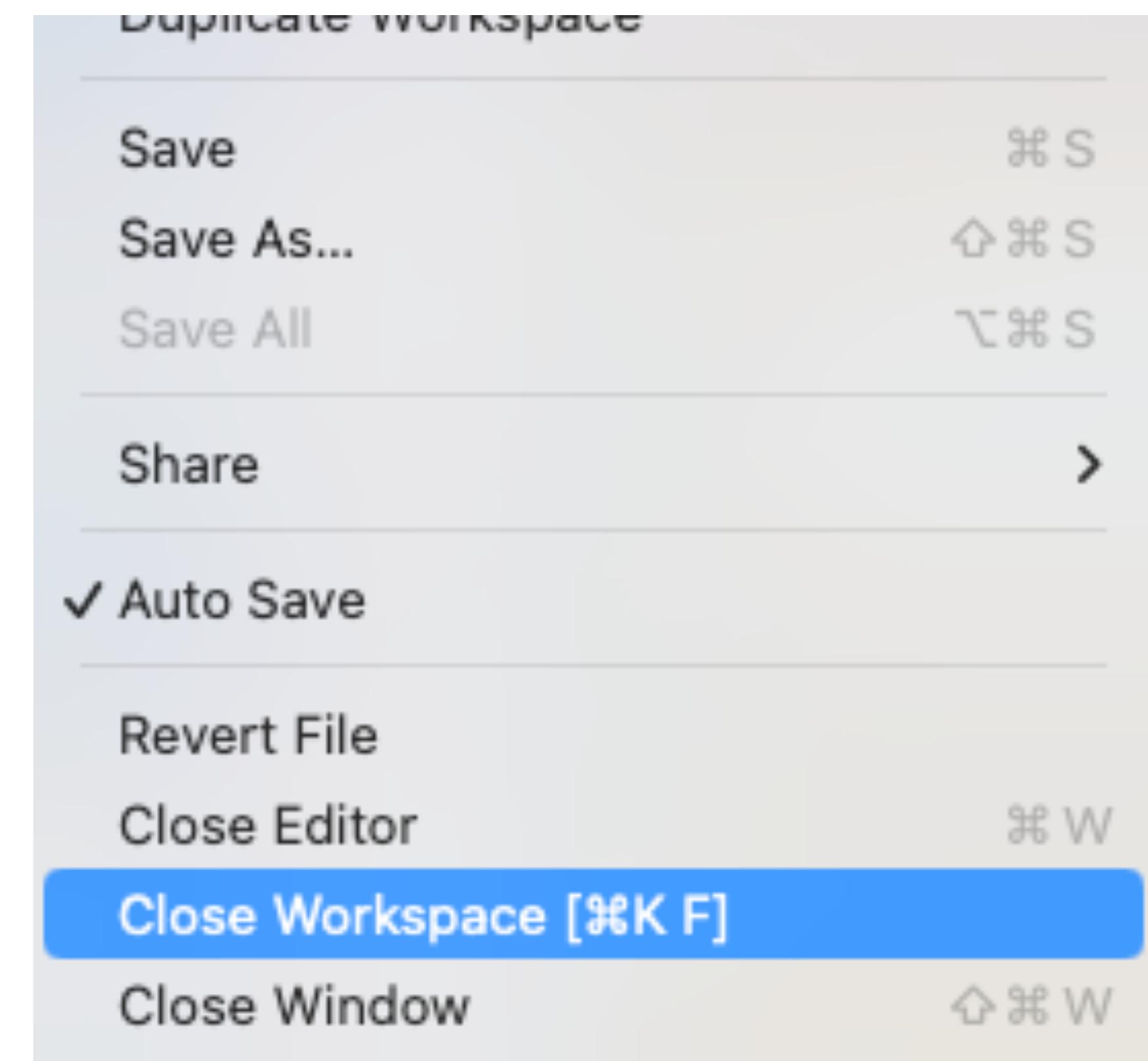
- Double-click the .code-workspace file in your platform's Explorer.
- Use the **File > Open Workspace** command and select the Workspace file.
- Select the Workspace from the **File > Open Recent (^R)** list.
- Workspaces have a (Workspace) suffix to differentiate them from folders in the UI





Closing Workspace Files

- Just like Close Folder when a single folder is open in VS Code, there is a Close Workspace (`⌘K F`) command to close the active Workspace.





User Interface



VSCode User Interface

The screenshot illustrates the VSCode user interface with several key components labeled:

- A Activity Bar**: The vertical bar on the left containing icons for Explorer, Search, Find, Files, and Settings.
- B Side Bar**: The sidebar on the left side of the main editor area, listing files and folders.
- C Editor Groups**: Multiple code editors grouped together.
- D Panel**: The bottom panel showing the Problems, Output, Debug Console, and Terminal tabs, with the Terminal tab active.
- E Status Bar**: The status bar at the bottom showing the current file path, commit status, and other system information.

Editor Groups (C):

```
/* Copyright (c) Microsoft Corporation. All rights reserved.
 * Licensed under the MIT License. See License.txt in the project root for license information.
 */
'use strict';

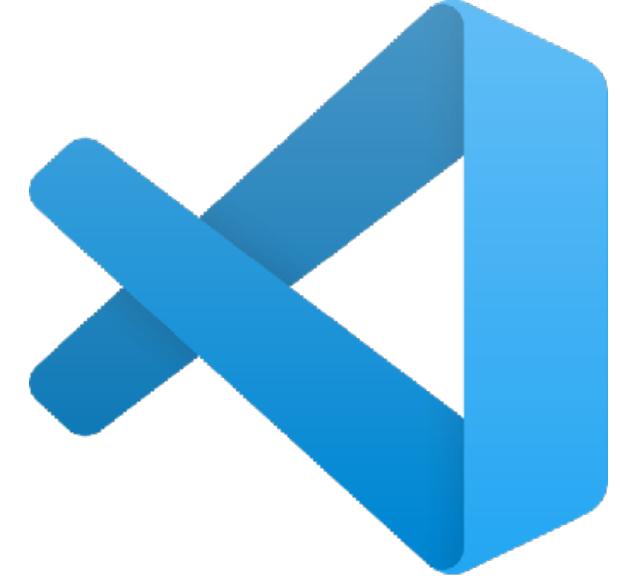
import * as dom from 'vs/base/browser/dom';
import { Widget } from 'vs/base/browser/ui/widget';
import { IKeybindingService } from 'vs/platform/keybinding/common/keybinding';
import { ICodeEditor, IOliverayWidget, IOliverayWidgetP
import { FIND_IDS } from 'vs/editor/contrib/find/find';
import { FindReplaceState } from 'vs/editor/contrib/f
import { CaseSensitiveCheckbox, WholeWordsCheckbox, R
import { RunOnceScheduler } from 'vs/base/common/asyn
import { IThemeService, ITheme, registerThemingPartic
import { inputActiveOptionBorder, editorWidgetBackgro
export class FindOptionsWidget extends Widget impleme
```

Terminal (Panel D):

```
PS C:\Users\gregvanl\vscode> [REDACTED]
```

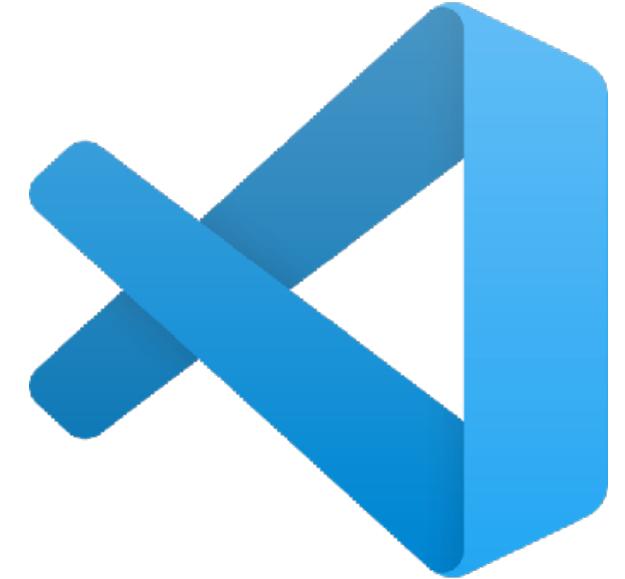
Status Bar (E):

```
master 1710 0 0 0
Ln 1, Col 1 Spaces: 2 UTF-8 LF JSON ☀
```



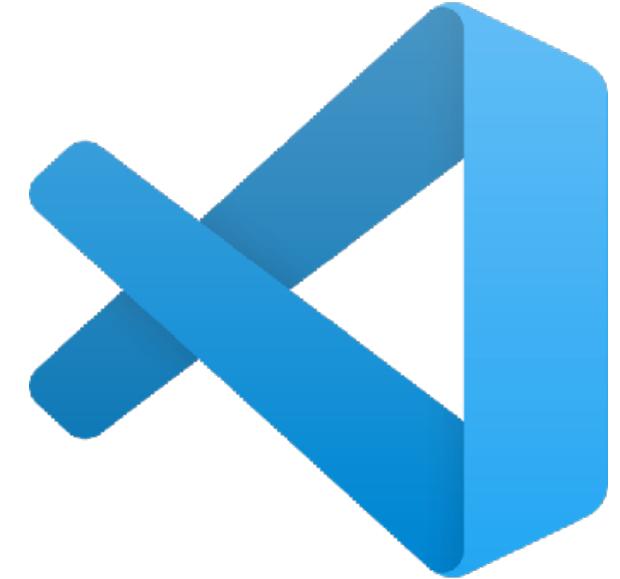
Basic Layout

- **Editor** - The main area to edit your files. You can open as many editors as you like side by side vertically and horizontally.
- **Primary Side Bar** - Contains different views like the Explorer to assist you while working on your project.
- **Status Bar** - Information about the opened project and the files you edit.
- **Activity Bar** - Located on the far left-hand side, this lets you switch between views and gives you additional context-specific indicators, like the number of outgoing changes when Git is enabled.
- **Panel** - An additional space for views below the editor region. By default, it houses output, debug information, errors and warnings, and an integrated terminal. Panel can also be moved to the left or right for more vertical space.
- A Secondary Side Bar is also available to display views opposite the Primary Side Bar. 



Splitting Editor Panes

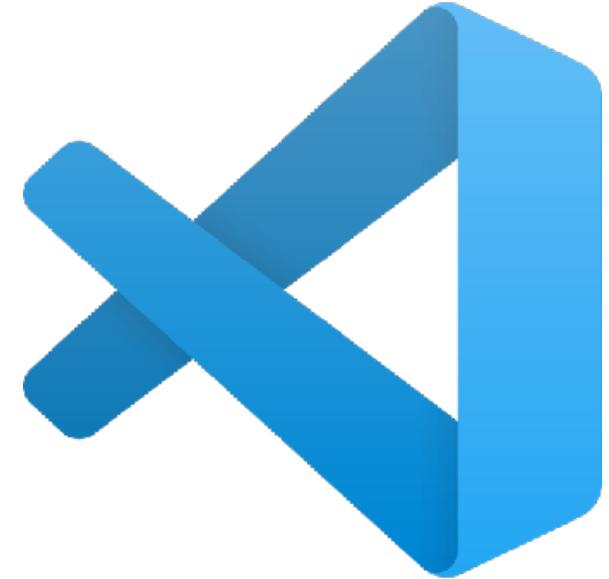
- **OPTION(⌥) + CLICK** Open file to open in split pane
- **COMMAND(⌘) + ** Split an active editor into two
- **CTRL(^) + ENTER** from the Explorer context menu on a file
- Click the Split Editor Button on the Upper Right of the Editor
- Drag and drop a file to any side of the editor region.
- **COMMAND(⌘) + ENTER** in the Quick Open List (**COMMAND(⌘) + P**)



Minimap

- A Minimap (code outline) gives you a high-level overview of your source code, which is useful for quick navigation and code understanding.
- A file's minimap is shown on the right side of the editor.
- You can click or drag the shaded area to quickly jump to different sections of your file.





Indent Guides

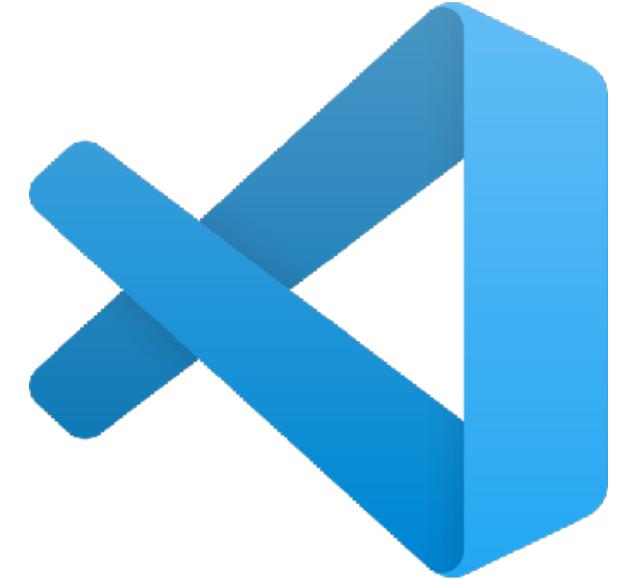
Indent Guides are light, typically grey, line that can help you match indentation levels



A screenshot of a code editor displaying an XML configuration file named log4j2.xml. The code is as follows:

```
src > main > resources > log4j2.xml
You, 4 days ago | 1 author (You)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration status="WARN">
3   <Appenders>
4     <Console name="Console" target="SYSTEM_OUT">
5       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
6     </Console>
7   </Appenders>
8   <Loggers>
9     <Root level="${sys:log.level:-INFO}">
10      <AppenderRef ref="Console"/>
11    </Root>
12  </Loggers>
13 </Configuration>
14 |
```

The code editor uses light grey vertical lines (indent guides) to indicate the indentation levels of the XML tags, helping the developer match them correctly. The file path, last modified date, and author information are visible at the top left of the editor window.



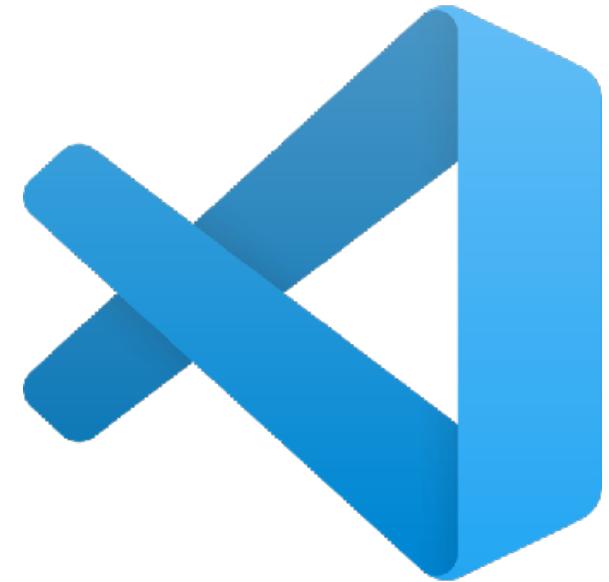
Breadcrumbs

- The editor has a navigation bar above its contents called **Breadcrumbs**.
- It shows the current location and allows you to quickly navigate between folders, files, and symbols.
- Breadcrumbs always show the file path and if the current file type has language support for symbols, the symbol path up to the cursor position.
- You can disable breadcrumbs with the **View > Show Breadcrumbs** toggle command.

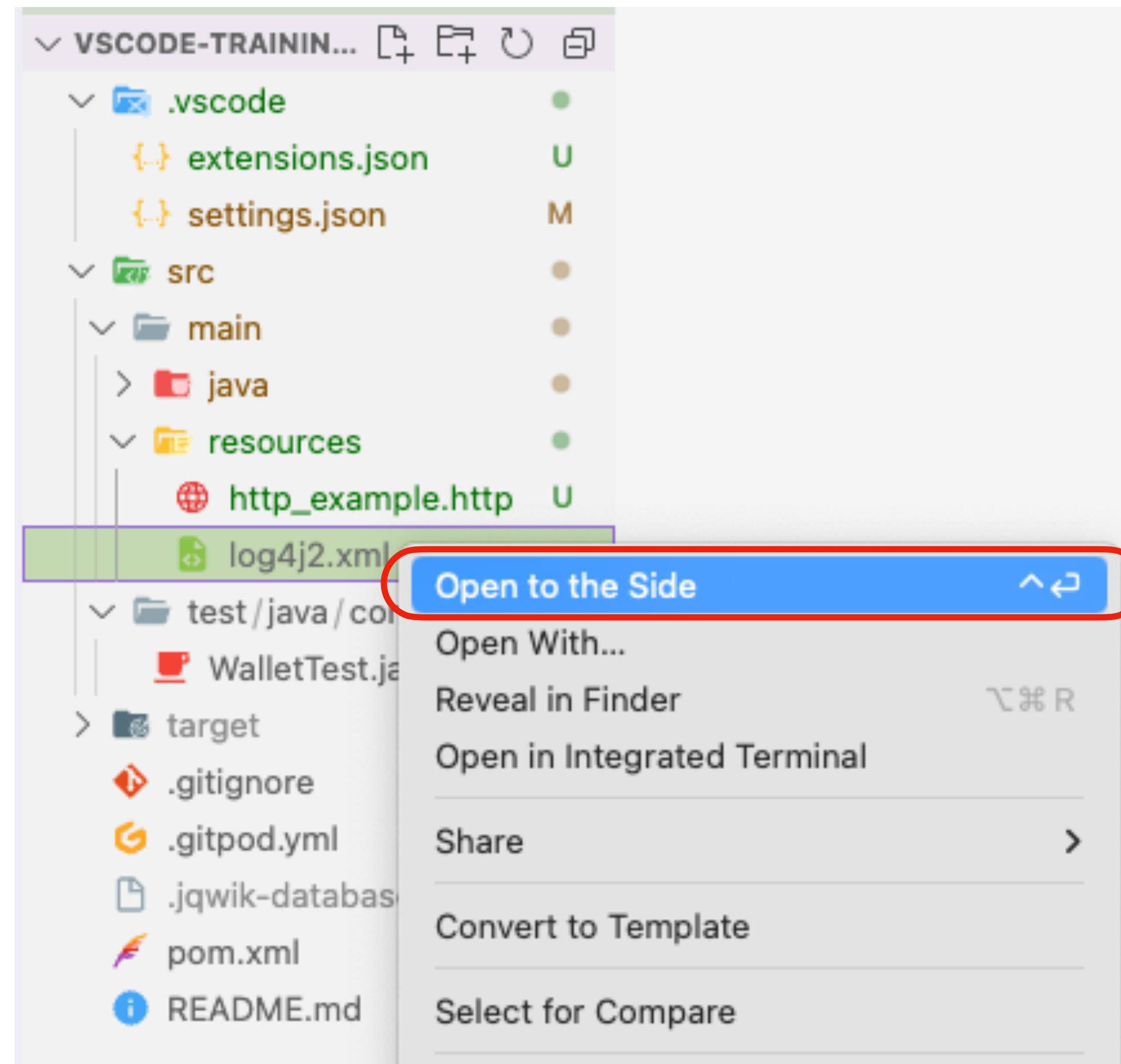


A screenshot of the Visual Studio Code interface. At the top, there is a navigation bar with the text "src > main > resources > log4j2.xml". This entire text is enclosed in a red oval. Below the bar, the code editor displays an XML file named "log4j2.xml". The code is as follows:

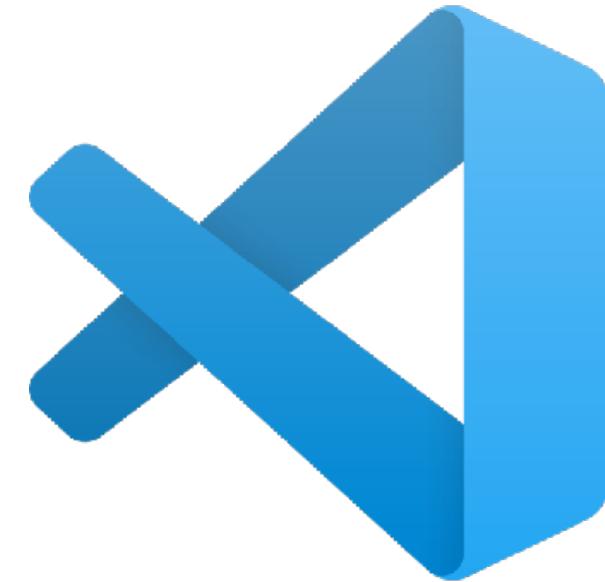
```
src > main > resources > log4j2.xml
You, 4 days ago | 1 author (You)
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Configuration status="WARN">
3      <Appenders>
4          <Console name="Console" target="SYSTEM_OUT">
5              <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
6          </Console>
7      </Appenders>
8      <Loggers>
9          <Root level="${sys:log.level:-INFO}">
10             <AppenderRef ref="Console"/>
11         </Root>
12     </Loggers>
13 </Configuration>
14
```



Editor Groups



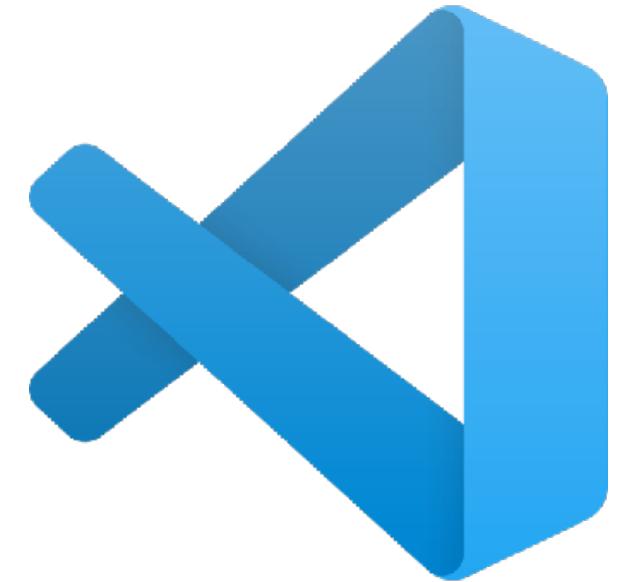
- When you split an editor (using the **Split Editor** or **Open to the Side** commands), a new editor region is created which can hold a group of items.
- You can open as many editor regions as you like side by side vertically and horizontally.
- You can Drag and Drop editor groups on the workbench, move individual Tabs between groups and quickly close entire groups (Close All).



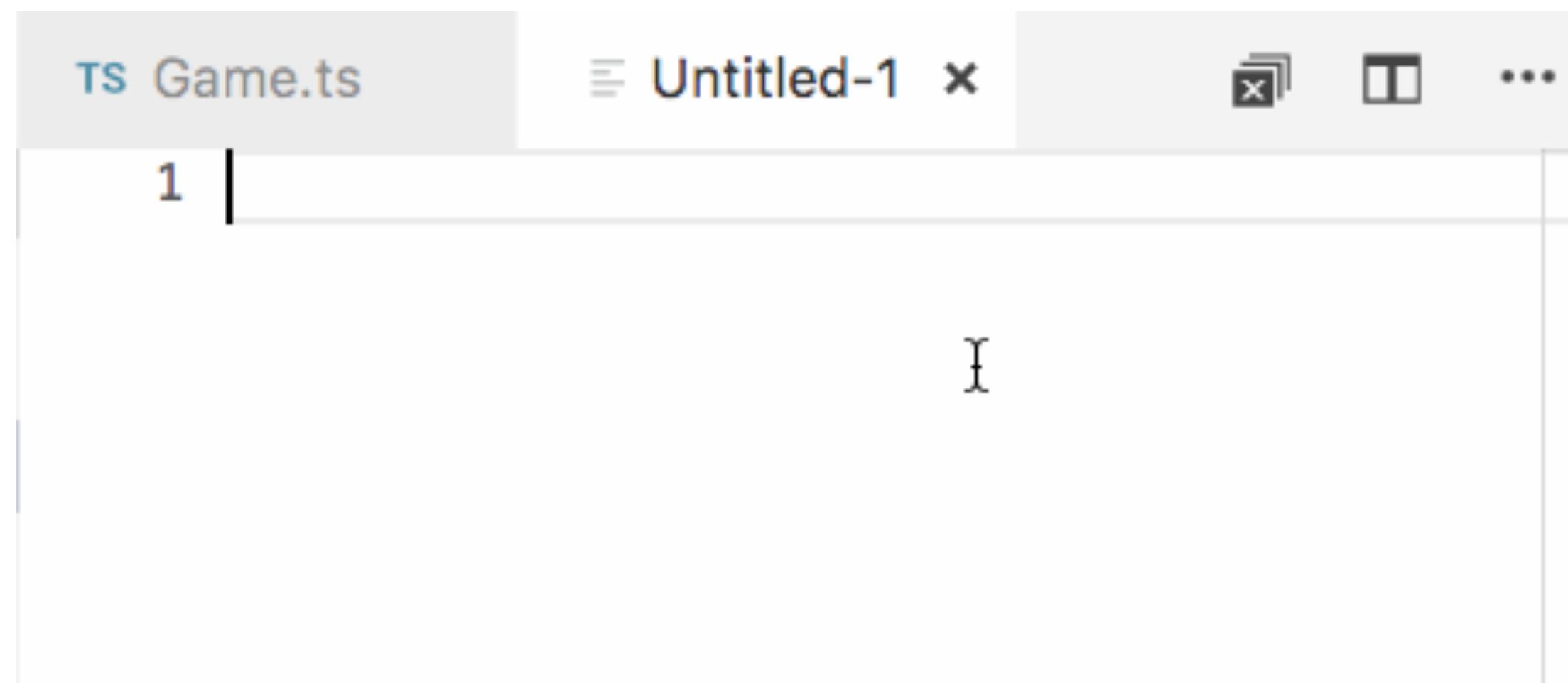
Grid Editor Layout

- By default, editor groups are laid out in vertical columns (for example when you split an editor to open it to the side).
- You can easily arrange editor groups in any layout you like, both vertically and horizontally

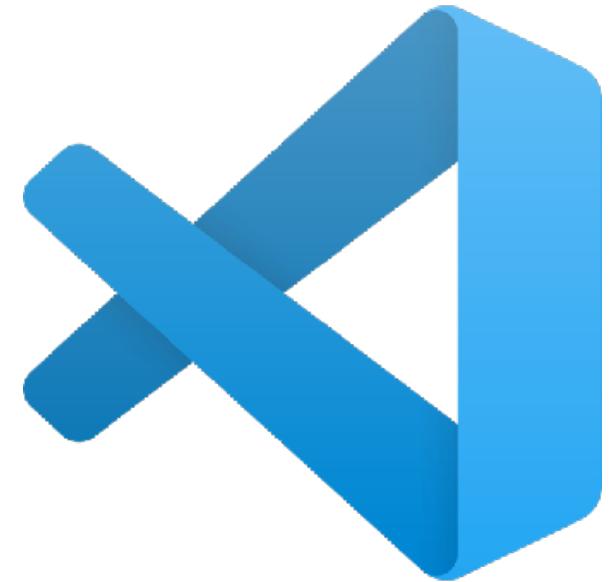
A screenshot of the Visual Studio Code interface demonstrating a horizontal grid editor layout. The window is divided into four quadrants by a yellow grid line. The top-left quadrant contains the Explorer, Terminal, and Status bar. The bottom-left quadrant contains the Outline, Timeline, Projects, and Run Configuration views. The top-right quadrant contains the Editor view with three files: 'WalletTest.java', 'extensions.json', and 'com.xml'. The bottom-right quadrant contains the Editor view with two files: '.gitpod.yml' and 'http_example.http'. The status bar at the bottom indicates 'Last synced: 42 secs ago'.



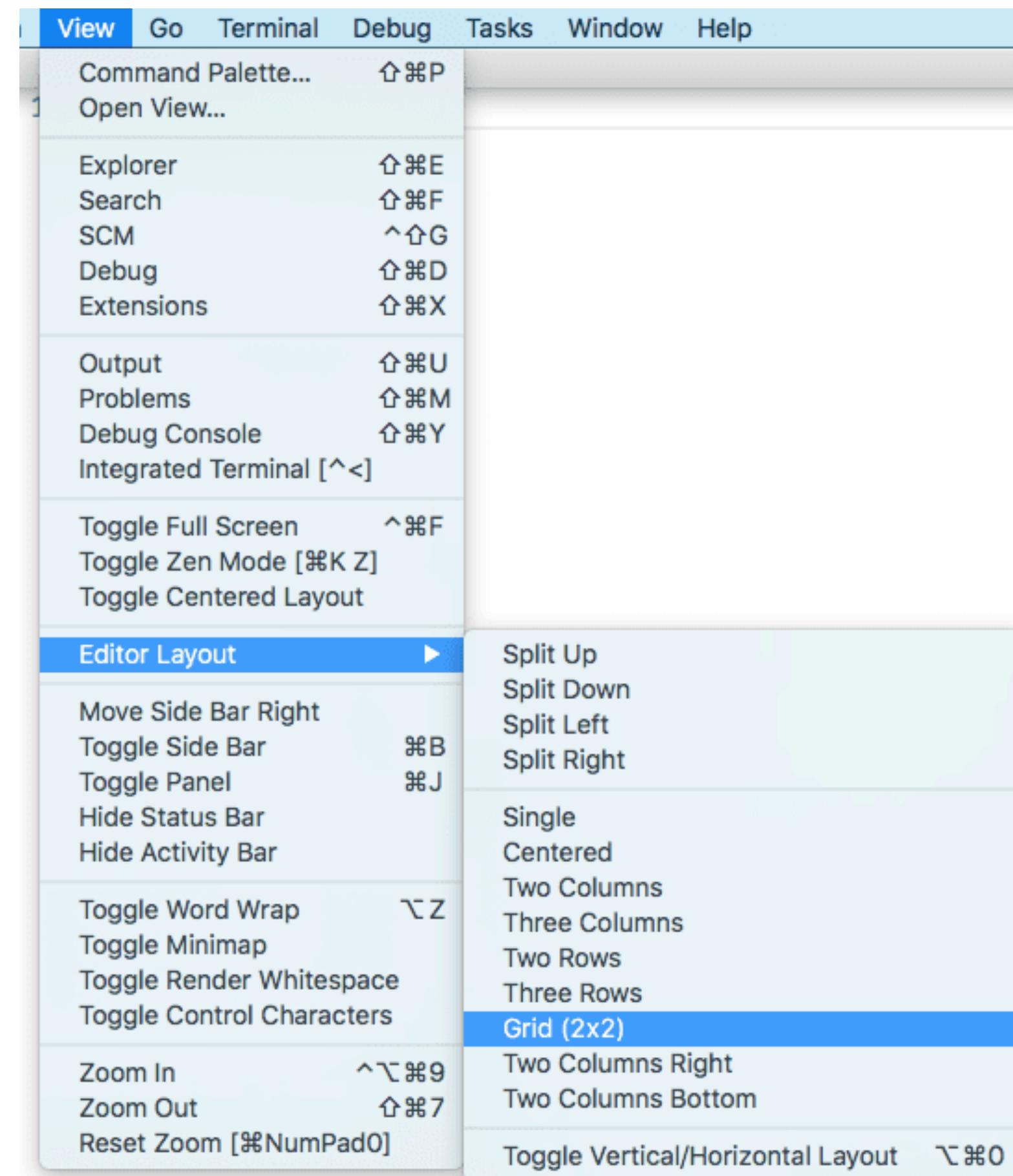
Holding the Option Key



You can hold the **Alt** or **Option** (↖) Key and open a new terminal in what format you would like



Predefined Editor Layouts



There are a predefined set of editor layouts in the new **View > Editor Layout** menu

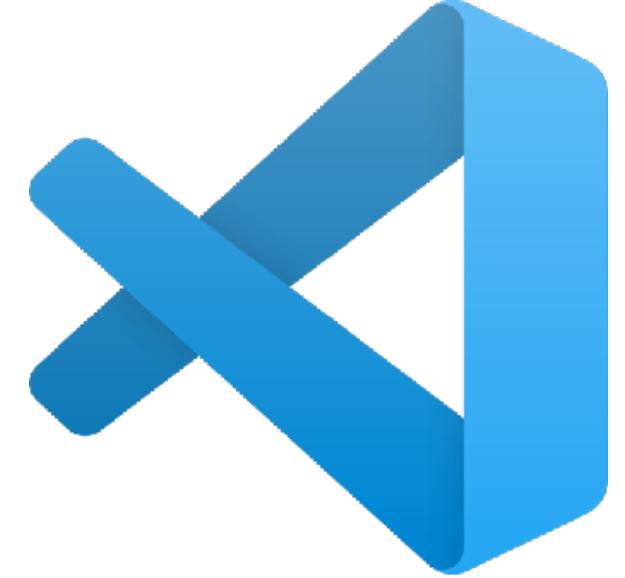


Extensions



Extensions

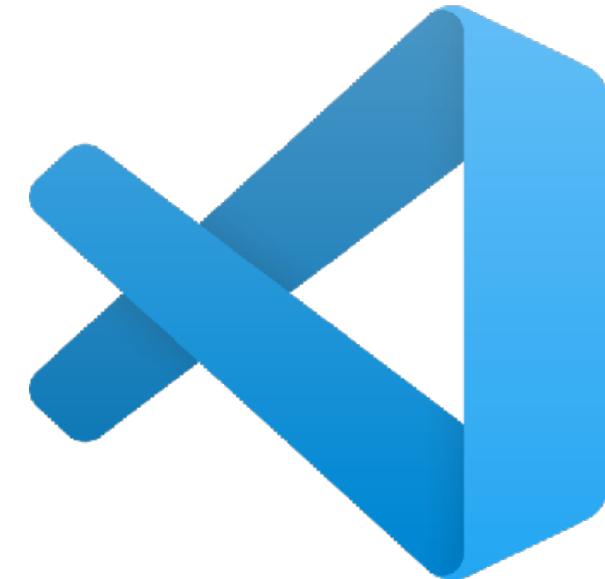
- The features that Visual Studio Code includes out-of-the-box are just the start. VS Code extensions let you add languages, debuggers, and tools to your installation to support your development workflow
- VS Code's rich extensibility model lets extension authors plug directly into the VS Code UI and contribute functionality through the same APIs used by VS Code.
- You can go to the Extensions by going to the Extensions view ()



Finding Extensions

Find Extensions:

- In the VS Code [Marketplace](#).
- Search inside VS Code in the Extensions view.
- View extension recommendations
- Community curated extension lists, such as [awesome-vscode](#).



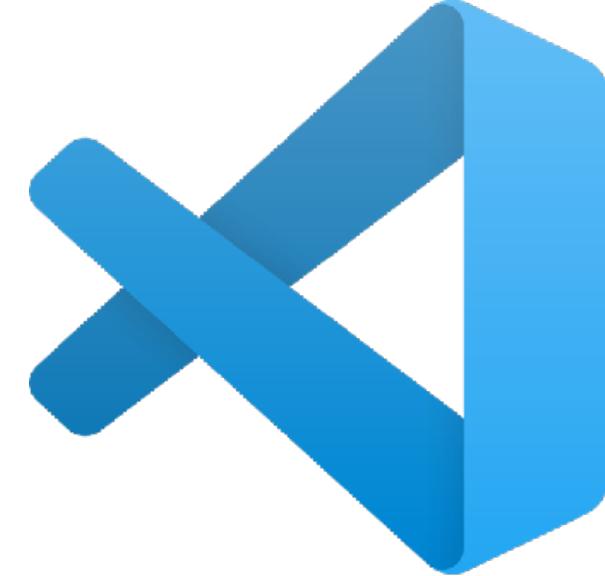
Filtering Extensions

Find Extensions:

- In the VS Code Marketplace.
- Search inside VS Code in the Extensions view using the language name

The image displays three separate screenshots of the VS Code Marketplace interface, each showing a search results page for a specific language. The first screenshot shows results for 'JavaScript', the second for 'Java', and the third for 'Python'. Each screenshot includes a search bar at the top with the language name, a list of extension cards below it, and a 'Install' button on the right side of each card.

Language	Extension	Description	Developer	Installs	Rating
JavaScript	JavaScript (ES6) co...	Code snippets for JavaScript in ES6...	charalampos karypidis	14.9M	4.5
	Babel JavaScript	VSCode syntax highlighting for toda...	Michael McDermott	2.9M	4.5
	JavaScript and Type...	Enables typescript@next to power ...	Microsoft	3.3M	2.5
Java	Extension Pack for Java	Popular extensions for Java develop...	Microsoft	25ms	—
	Debugger for Java	A lightweight Java debugger for Vis...	Microsoft	7ms	—
	Test Runner for Java	Run and debug JUnit or TestNG test...	Microsoft	9ms	—
	Project Manager for Java	Manage Java projects with ease	Microsoft	17ms	—
Python	Python	Python language support with exten...	Microsoft	117K	5
	Python Debugger	Python Debugger extension using d...	Microsoft	—	—
	Python Indent	Intelligent Python code indentation	shiro	8.4M	4.5



Using VSIX Files

- You can manually install a VS Code extension packaged in a .vsix file.
- Using the Install from VSIX command in the Extensions view command dropdown, or the Extensions:
 - Install from VSIX command in the Command Palette, point to the .vsix file.
 - You can also install using the VS Code `--install-extension` command-line switch providing the path to the `.vsix` file.

```
> code --install-extension myextension.vsix
```

- You can include multiple `--install-extension` tags as needed



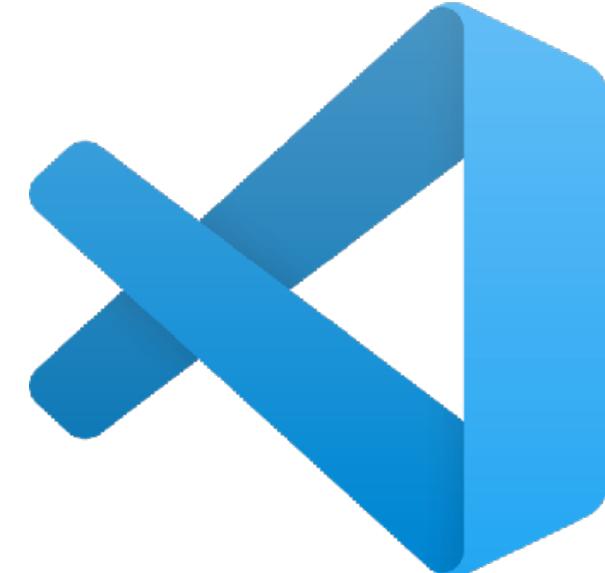
Where are Extensions installed?

- Extensions are installed in a per user extensions folder. Depending on your platform, the location is in the following folder:
 - Windows: `%USERPROFILE%\.\vscode\extensions`
 - MacOSX: `~/.vscode/extensions`
 - Linux: `~/.vscode/extensions`
- You can change the location by launching VS Code with the `--extensions-dir <dir>` command-line option.



Creating your own extensions

- View the [Extension API Documentation](#)
- View the [Contribution Point Documentation](#)
 - Contribution Points are how your extensions will interact with Visual Studio Code
 - Contribution points are placed in the package.json of your Extension Manifest



Are Extensions safe?

<https://www.aquasec.com/blog/can-you-trust-your-vscode-extensions/>



Platform

Solutions

Resources

Company



Get Started

◀ Aqua Blog

Can You Trust Your VSCode Extensions?



Ilay Goldman, Yakir Kadkoda

January 6, 2023





Are Extensions Safe?

- Maybe? (How is that for confidence?)
- Companies should have their own scanning of the source of the extensions
- All Extensions via VSIX files are just Zip File and can be scanned
- Most plugins are typically on the internet and can be audited
- Still, not everything is secure so take care.

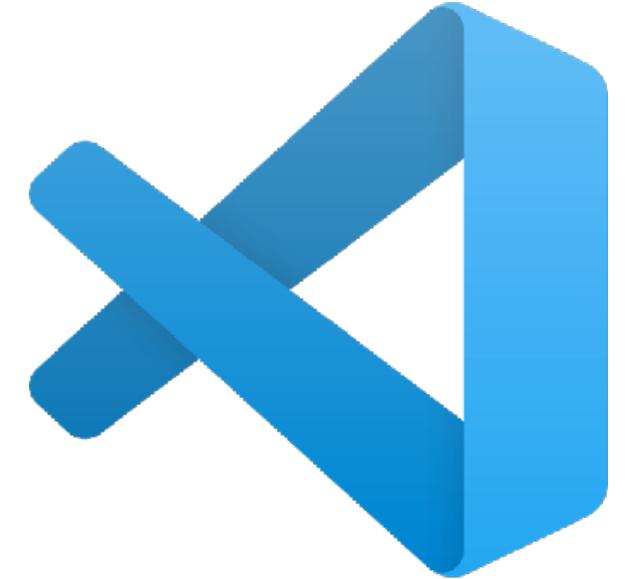


Basic Usages



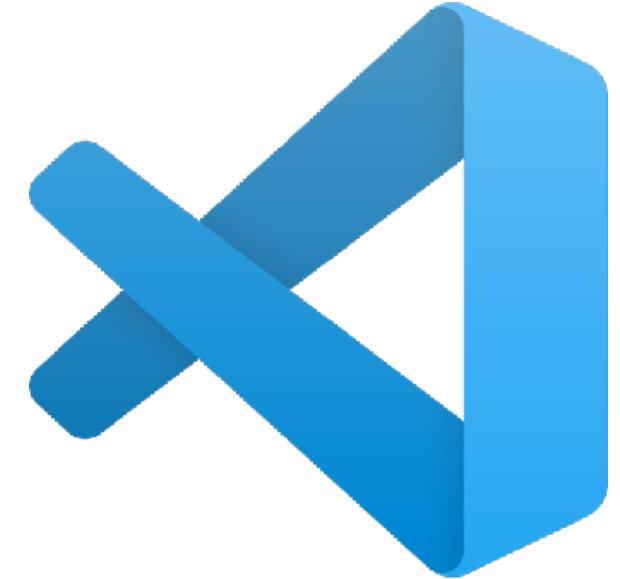
Quick Open

- Quick Opening of a File - `⌘P`
- Once in the view; you can select `?` to view command suggestions
- Repeat the Quick Open keyboard shortcut to cycle quickly between recently opened files.
- You can open multiple files from **Quick Open** by **pressing the Right arrow key**. This will open the currently selected file in the background and you can continue selecting files from Quick Open.
- Open **Recent command** or `^R` - Displays a Quick Pick dropdown with the list from **File > Open Recent** with recently opened folders and workspaces followed by files.



Other Navigation Options

- ⌘P will let you navigate to any file or symbol by typing its name
- ⌘Tab will cycle you through the last set of files opened
- ⌘O will let you navigate to a specific symbol in a file
- ⌘G will let you navigate to a specific line in a file



Zen Mode

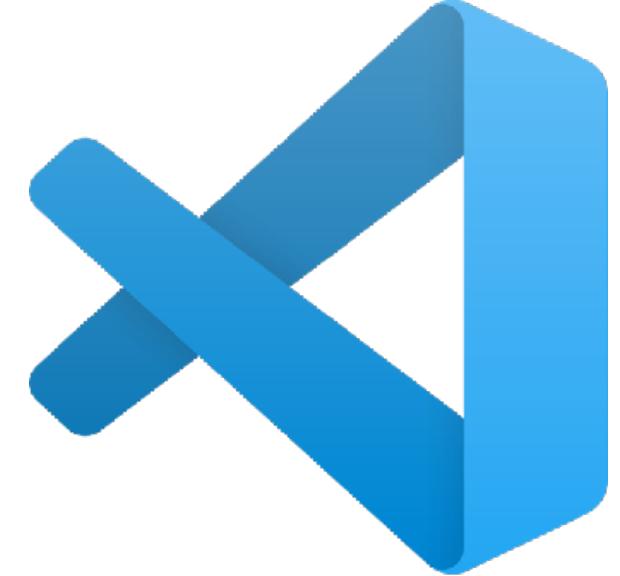
- Zen Mode lets you focus on your code by hiding all UI except the editor (no Activity Bar, Status Bar, Side Bar and Panel), going to full screen and centering the editor layout
- Zen mode can be toggled using the **View** menu, **Command Palette** or by the shortcut **⌘K Z**.
- Double **Esc** exits Zen Mode.



Customizing VSCode



Themes



Changing your Theme and Icons

- Change your theme using `⌘K ⌘T`
- You can install more themes from the VS Code extension [Marketplace](#).
- You can also install icon themes which changes icon looks and feels
- You can install more icon themes from the VS Code extension [Marketplace](#)
 - Filter the list of extensions using: `tag:icon-theme`



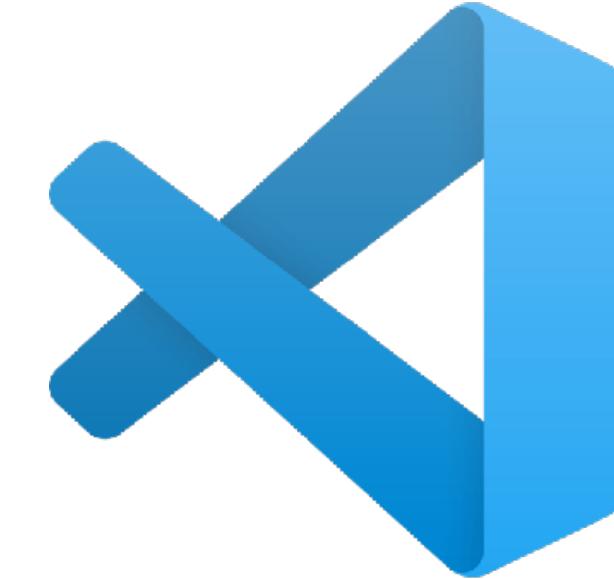
Adding your themes



- Do what makes you happy
- Change icons and themes to your liking



Settings



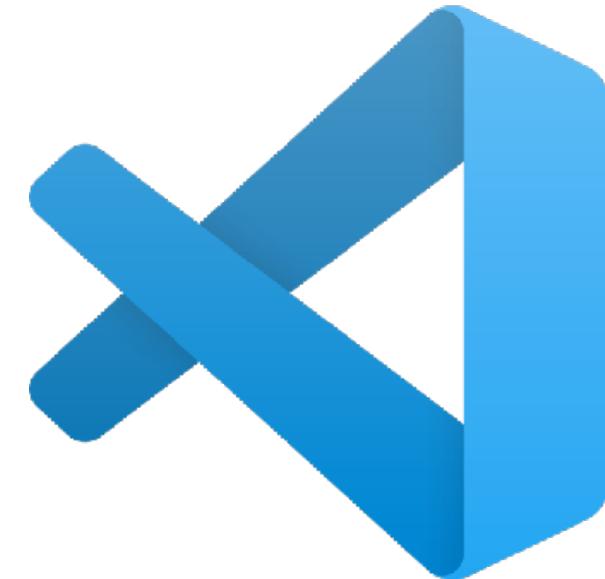
Settings

- Nearly every part of VS Code's editor, user interface, and functional behavior has options you can modify.
- VS Code provides several different scopes for settings. When you open a workspace, you will see at least the following two scopes:
 - **User Settings** - Settings that apply globally to any instance of VS Code you open.
 - **Workspace Settings** - Settings stored inside your workspace and only apply when the workspace is opened.
- There are the core settings in VSCode, but every extension also has its own settings that can *also be manipulated*



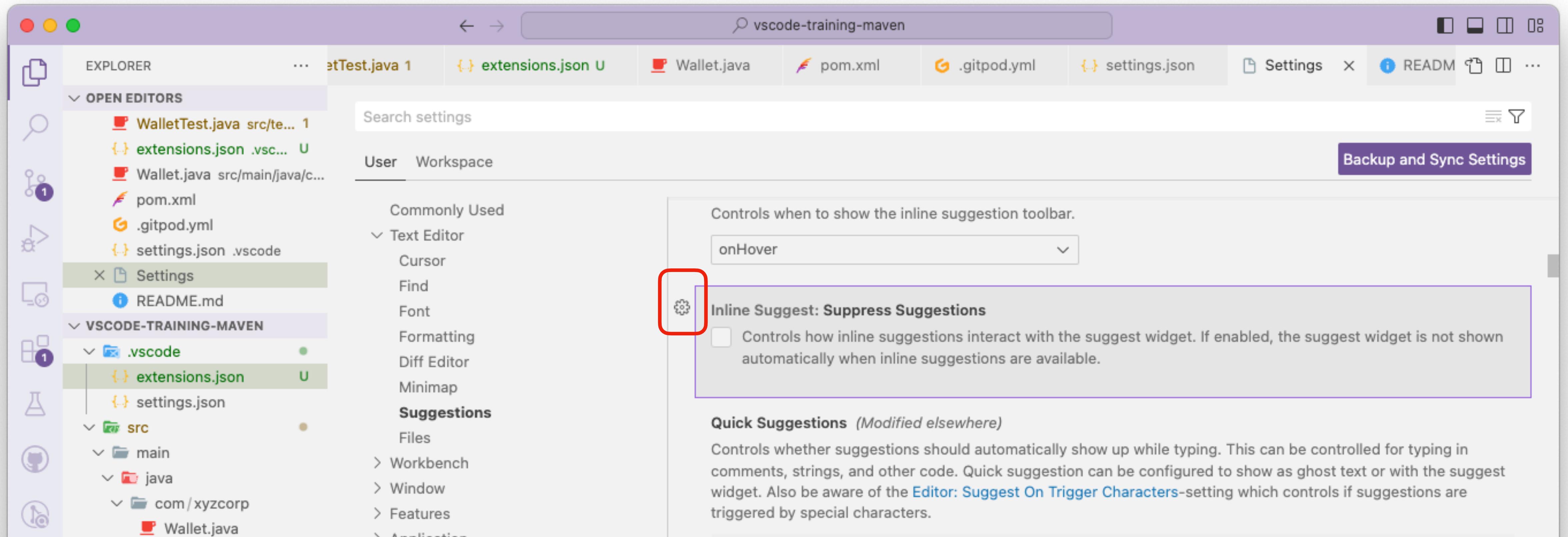
Settings Editor

- **Code > Settings > Settings**
- Command Pallette (with Preferences: Open Settings)
- Use the Standard () to go into your Settings



Exploring Settings

- You will have a selection line when a particular setting is active
- Each selection will also have a gear icon to view more information on your particular setting
- Every setting is either a checkbox, input or dropdown



A screenshot of the Visual Studio Code interface showing the settings editor. The title bar says "vscode-training-maven". The left sidebar shows the file tree with files like "WalletTest.java", "extensions.json", "Wallet.java", "pom.xml", ".gitpod.yml", and "settings.json". The "Settings" file is currently selected. The main area shows the settings configuration. A specific setting, "Inline Suggest: Suppress Suggestions", is highlighted with a red box around its gear icon. The setting description says: "Controls how inline suggestions interact with the suggest widget. If enabled, the suggest widget is not shown automatically when inline suggestions are available." Below it, another section titled "Quick Suggestions (Modified elsewhere)" is partially visible.

Search settings

User Workspace

Commonly Used

Text Editor

Cursor

Find

Font

Formatting

Diff Editor

Minimap

Suggestions

Files

> Workbench

> Window

> Features

> Application

Controls when to show the inline suggestion toolbar.

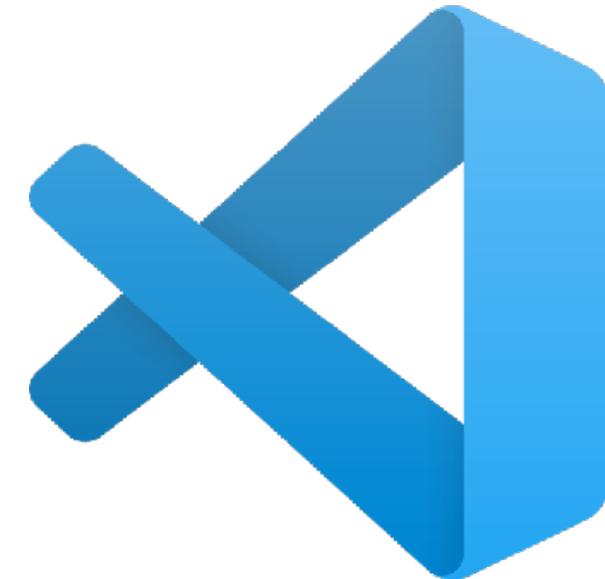
onHover

Inline Suggest: Suppress Suggestions

Controls how inline suggestions interact with the suggest widget. If enabled, the suggest widget is not shown automatically when inline suggestions are available.

Quick Suggestions (Modified elsewhere)

Controls whether suggestions should automatically show up while typing. This can be controlled for typing in comments, strings, and other code. Quick suggestion can be configured to show as ghost text or with the suggest widget. Also be aware of the [Editor: Suggest On Trigger Characters](#)-setting which controls if suggestions are triggered by special characters.



Setting Groups

- Settings are represented in groups so that you can navigate them easily.
- There is a **Commonly Used** group at the top, which shows popular customizations.

The screenshot shows the VS Code settings interface. On the left, the Explorer sidebar displays a project structure with files like `WalletTest.java`, `extensions.json`, `pom.xml`, `.gitpod.yml`, and `settings.json`. A red box highlights the `Commonly Used` group under the `Text Editor` section. The main panel shows settings for the `Text Editor` group, including `Cursor`, `Find`, `Font`, `Formatting`, `Diff Editor`, `Minimap`, `Suggestions`, and `Files`. Below this, there are sections for `Inline Suggest: Suppress Suggestions` and `Quick Suggestions`.

Commonly Used

- Text Editor
 - Cursor
 - Find
 - Font
 - Formatting
 - Diff Editor
 - Minimap
 - Suggestions**
 - Files

Controls when to show the inline suggestion toolbar.

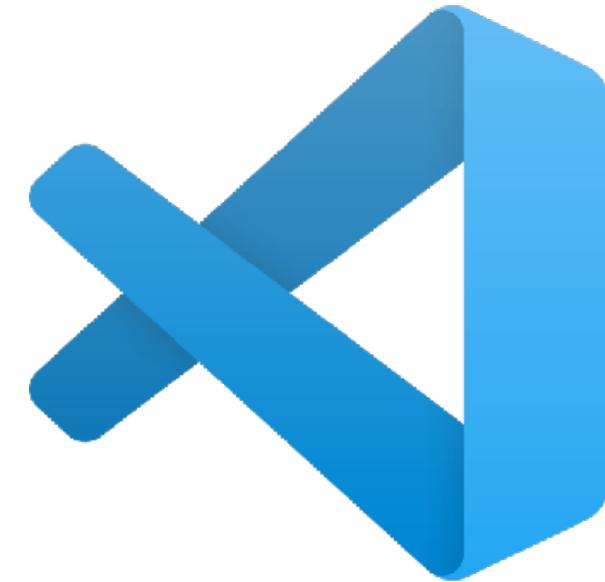
onHover

Inline Suggest: Suppress Suggestions

Controls how inline suggestions interact with the suggest widget. If enabled, the suggest widget is not shown automatically when inline suggestions are available.

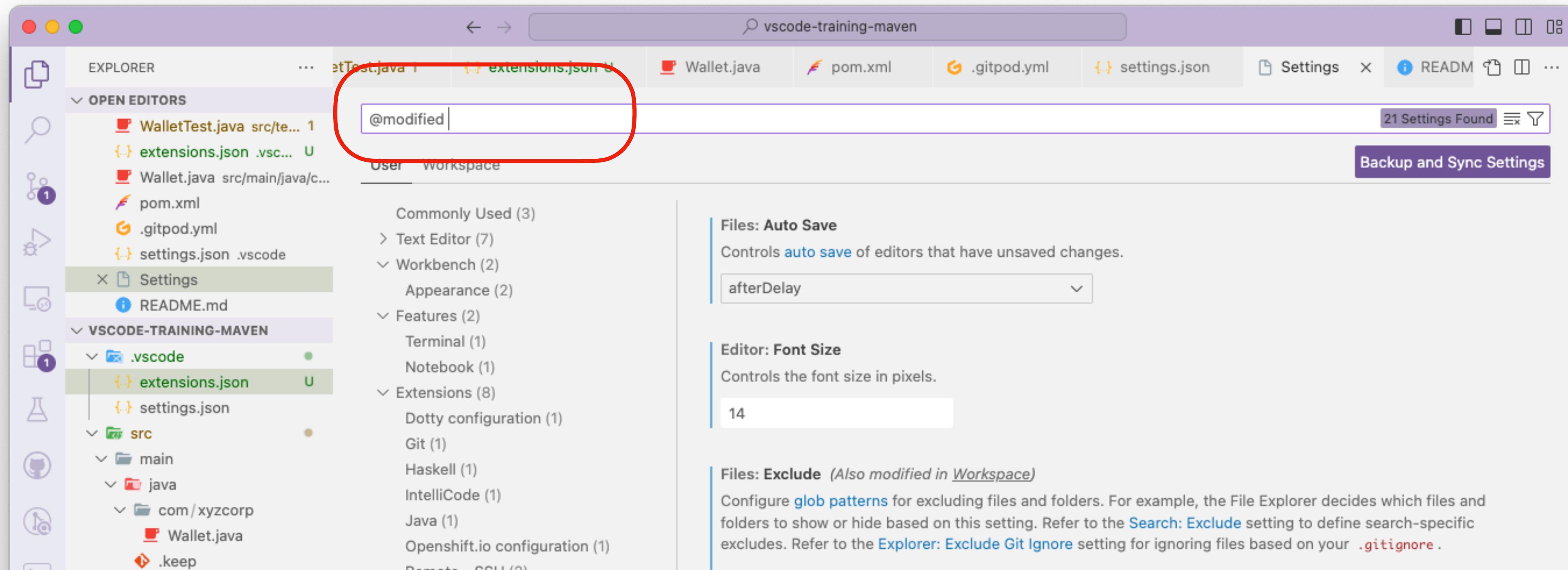
Quick Suggestions (Modified elsewhere)

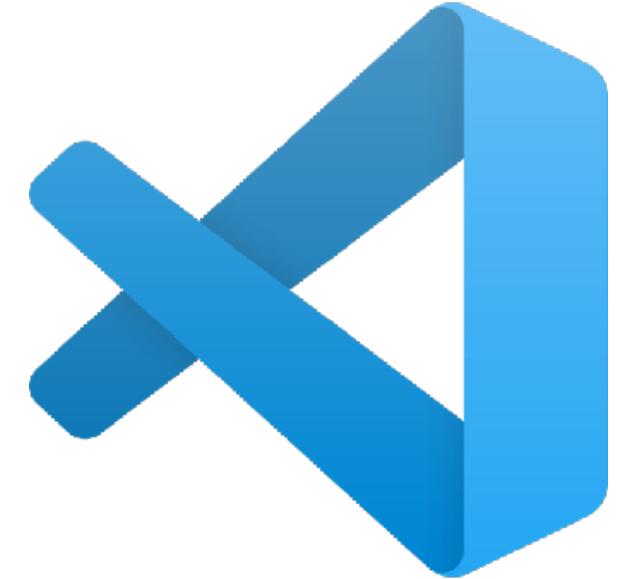
Controls whether suggestions should automatically show up while typing. This can be controlled for typing in comments, strings, and other code. Quick suggestion can be configured to show as ghost text or with the suggest widget. Also be aware of the [Editor: Suggest On Trigger Characters](#)-setting which controls if suggestions are triggered by special characters.



Filtering Settings

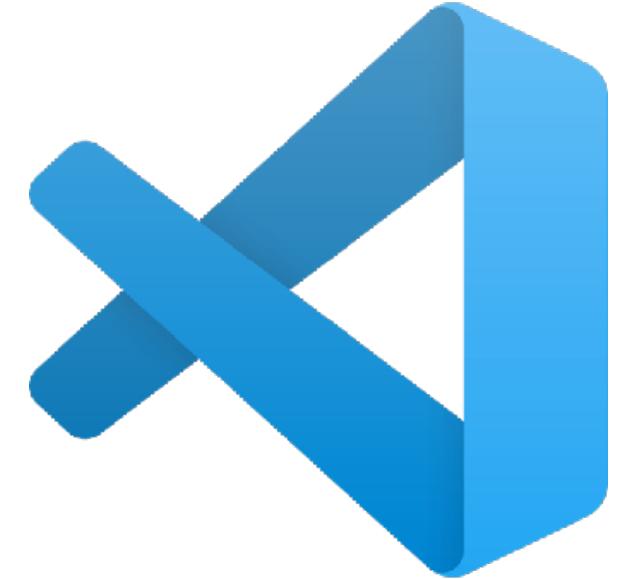
- Settings editor Search bar has several filters to make it easier to manage your settings.
- To the right of the Search bar is a filter button with a funnel icon that provides some options to easily add a filter to the Search bar
- You can also filter settings with keyword that begin with an @





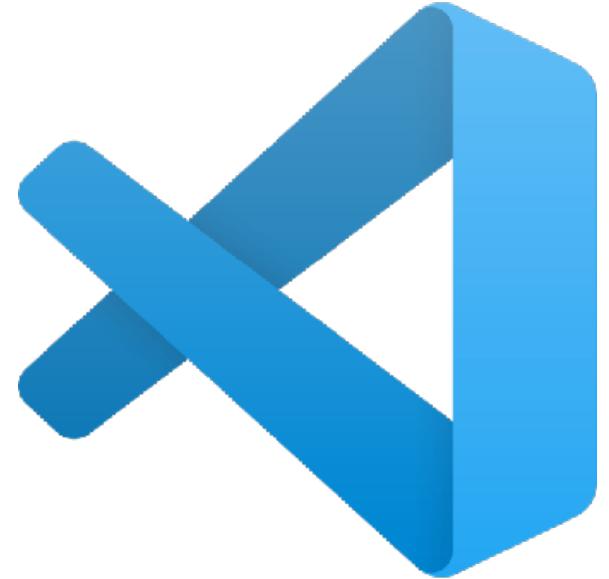
Example Settings

- **@ext** - Settings specific to an extension. You provide the extension ID such as `@ext:ms-python.python`.
- **@feature** - Settings specific to a Features subgroup. For example, `@feature:explorer` shows settings of the File Explorer.
- **@id** - Find a setting based on the setting ID. For example, `@id:workbench.activityBar.visible`.
- **@lang** - Apply a language filter based on a language ID. For example, `@langtypescript`. See Language-specific editor settings for more details.
- **@tag** - Settings specific to a system of VS Code. For example, `@tag:workspaceTrust` for settings related to Workspace Trust, or `@tag:accessibility` for settings related to accessibility.



Tuning your Settings

- You can also open from your command palette (↑⌘P) and finding **Open User Settings (JSON) command**
- You can also change your default settings editor with the `workbench.settings.editor` setting in your settings (⌘,). Use “json” to force changes to settings to always take place in your `settings.json` file.
- Settings can be:
 - Per user (depending on operating system)
 - Per workspace (`.vscode/settings`)
 - Your `settings.json` file has intellisense and smart completions with error highlighting



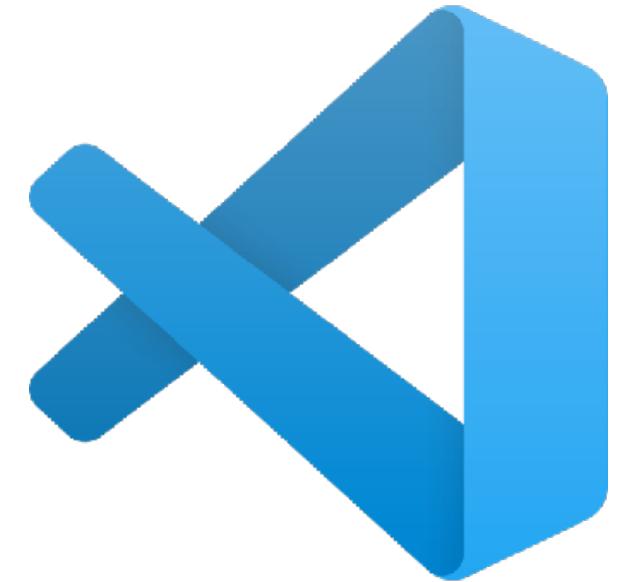
settings.json Example

```
// Main editor
"editor.fontSize": 18,
// Terminal panel
"terminal.integrated.fontSize": 14,
// Output panel
"[Log]": {
    "editor.fontSize": 15
}
```



User Settings

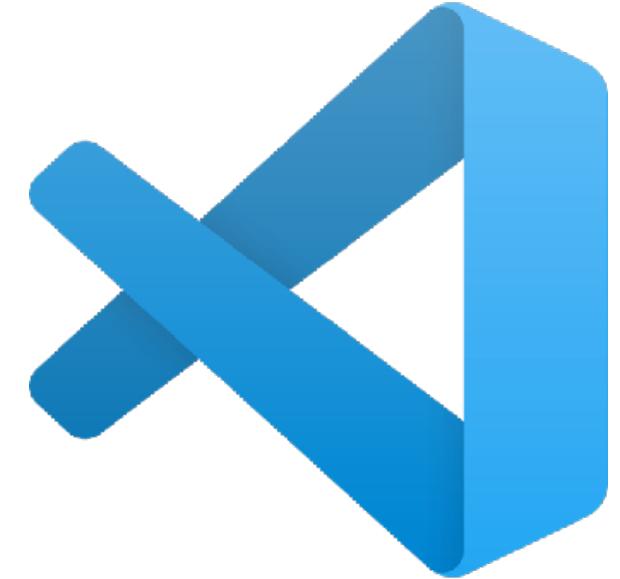
- Depending on the platform that you are using, you can find your user settings in the following locations
 - **Windows** `%APPDATA%\Code\User\settings.json`
 - **MacOSX** `$HOME/Library/Application\ Support/Code/User/settings.json`
 - **Linux** `$HOME/.config/Code/User/settings.json`



Workspace Settings

- Workspace settings are specific to a project and can be shared across developers on a project.
- **Workspace settings override user settings.**

The screenshot shows the VS Code interface with the title bar "vscode-training-maven". The left sidebar shows the "OPEN EDITORS" section with files like "WalletTest.java", "extensions.json", "Wallet.java", "pom.xml", ".gitpod.yml", and "settings.json". The "Settings" file is currently selected. The main editor area shows a search result for "@modified" with tabs for "User" and "Workspace". The "Workspace" tab is circled in red. Below it, the "Commonly Used" section lists "Text Editor", "Features", and "Extensions". To the right, a panel titled "Files: Exclude" provides instructions on configuring glob patterns to exclude files and folders from the file explorer. A list of patterns is shown, including: "**/.git", "**/.svn", "**/.hg", "**/CVS", "**/.DS_Store", and "**/Thumbs.db".



Workspace Settings

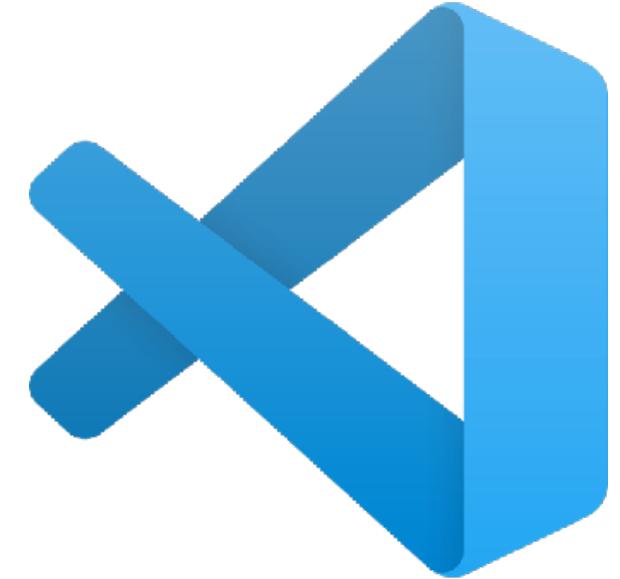
- Workspaces are just your project root folder
- Workspace settings as well as [debugging](#) and [task](#) configurations are stored at the root in a [.vscode](#) folder.
- All features of the Settings editor such as settings groups, search, and filtering behave the same for Workspace settings.
- Not all User settings are available as Workspace settings.
- For example, application-wide settings related to updates and security can not be overridden by Workspace settings.
- For a Multi-root Workspace, workspace settings are located inside the workspace configuration file.



Editing Workspace Files

- For multi-root project files, you can manually edit the path to the projects that you wish to use
- Every configuration is a JSON, and the workspace will be no different, Workspace Files can be edited either:
 - Opening the `.code-workspace` file within VSCode
 - Using the Command Palette (which we will see later) and calling **Preferences Open Workspace Settings (JSON)**

```
{  
  "folders": [  
    {  
      "path": "project-1"  
    },  
    {  
      "path": "project-2"  
    }  
,  
  "settings": {...}  
}
```



Minimap Settings

- You can move the minimap to the left hand side or disable it completely by respectively setting `"editor.minimap.side": "left"` or `"editor.minimap.enabled": false` in your user or workspace settings.



Indent Guide Settings

If you would like to disable indent guides, you can set "editor.guides.indentation": **false** in your user or workspace settings.



A screenshot of the Microsoft Visual Studio Code (VS Code) code editor. The editor shows an XML configuration file named 'log4j2.xml'. The code is color-coded and uses vertical green lines to indicate indentation levels. The XML content includes sections for Configuration, Appenders, and Loggers. The status bar at the bottom right of the editor window displays the text 'Indent Guide Settings'.

```
src > main > resources > log4j2.xml
You, 4 days ago | 1 author (You)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration status="WARN">
3   <Appenders>
4     <Console name="Console" target="SYSTEM_OUT">
5       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
6     </Console>
7   </Appenders>
8   <Loggers>
9     <Root level="${sys:log.level:-INFO}">
10      <AppenderRef ref="Console"/>
11    </Root>
12  </Loggers>
13 </Configuration>
14 |
```



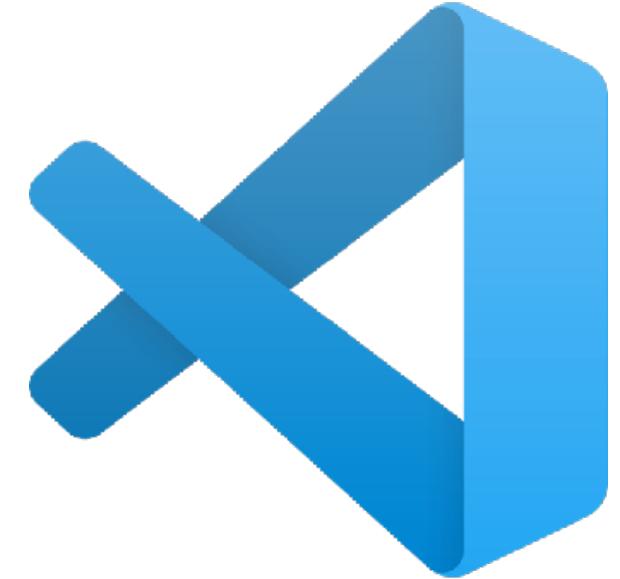
Change some settings



- Try to disable the minimap for either your workspace, or user settings or both
- Change the font size for your current workspace
- View the list of settings in both JSON and GUI style

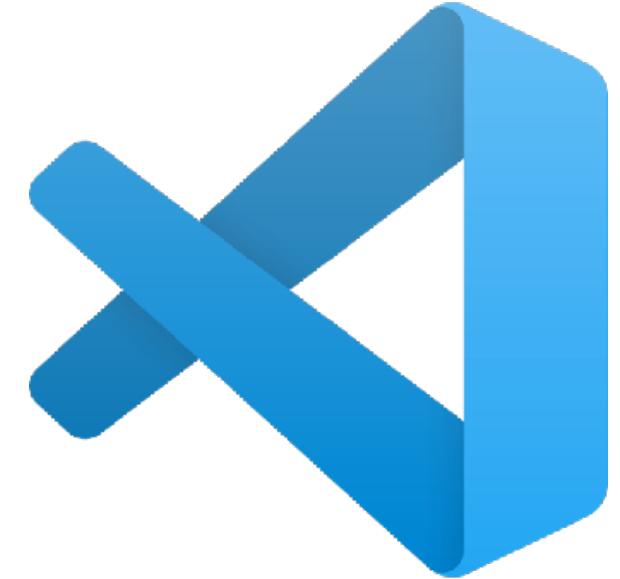


Keybindings



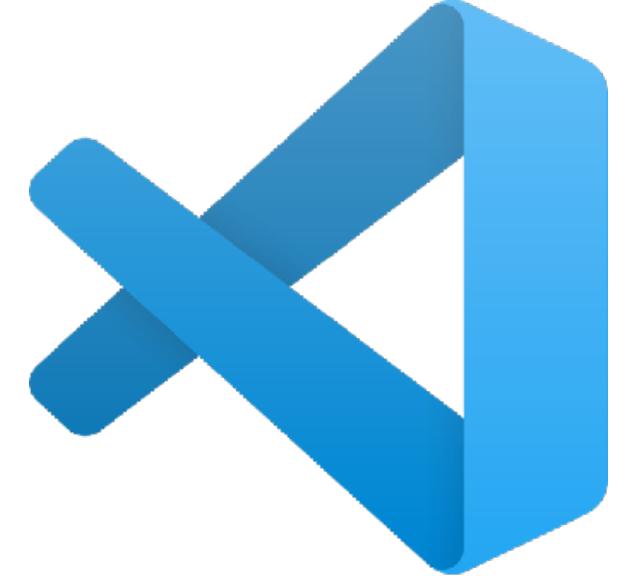
Accessing Cheatsheets

- Always have cheatsheets for your applicable operating system handy
- [Windows](#), [MacOSX](#), and [Linux](#)
- You can also find the keyboard shortcuts by going to the Command Palette ($\text{Shift} + \text{Command} + \text{P}$), and selecting **Help: Keyboard Shortcuts Reference** or ($\text{Shift} + \text{Command} + \text{K} + \text{R}$)
- These key commands are the default key combinations, they can also be customized to your liking



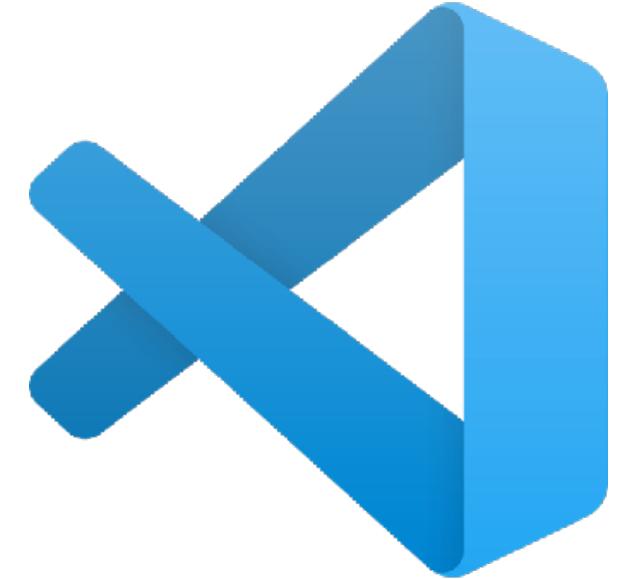
Keyboard Shortcuts Editor

- Visual Studio Code provides a rich and easy keyboard shortcuts editing experience using **Keyboard Shortcuts** editor.
- It lists all available commands with and without keybindings and you can easily change / remove / reset their keybindings using the available actions.
- It also has a search box on the top that helps you in finding commands or keybindings.
- Keyboard layouts are set based on your Keyboard layout
- **Code > Settings > Keyboard Shortcuts.**



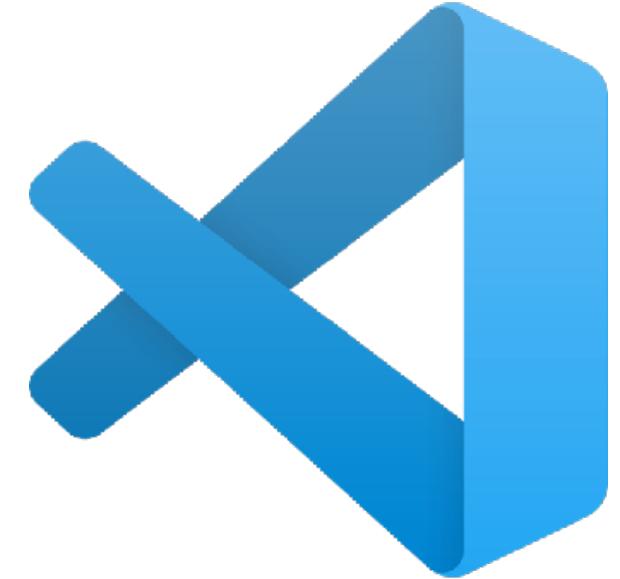
Migrating Keymaps

- Keyboard shortcuts are vital to productivity and changing keyboarding habits can be tough.
- To help with this, **Code > Settings > Migrate Keyboard Shortcuts from...** shows you a list of popular Keymap extensions
- These extensions modify the VS Code shortcuts to match those of other editors so you don't need to learn new keyboard shortcuts.
- There is also a [Keymaps category](#) of extensions in the Marketplace.



Detecting Conflicts

- If you have many extensions installed or you have [customized](#) your keyboard shortcuts, you can sometimes have keybinding conflicts where the same keyboard shortcut is mapped to several commands.
- The **Keyboard Shortcuts** editor has a context menu command **Show Same Keybindings**, which will filter the keybindings based on a keyboard shortcut to display conflicts.



Troubleshooting Keybindings

- To troubleshoot keybindings problems, you can execute the command **Developer: Toggle Keyboard Shortcuts Troubleshooting**.
- This will activate logging of dispatched keyboard shortcuts and will open an output panel with the corresponding log file.
- You can then press your desired keybinding and check what keyboard shortcut VS Code detects and what command is invoked.



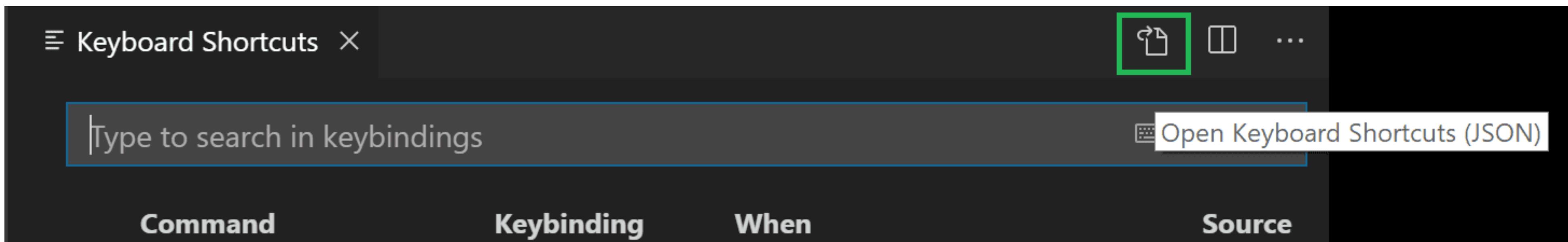
Viewing Modified Keybindings

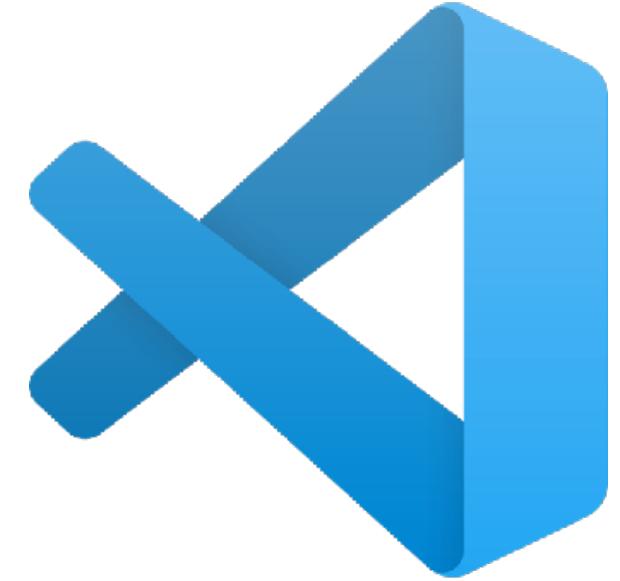
- You can view any user modified keyboard shortcuts in VS Code in the **Keyboard Shortcuts** editor with the **Show User Keybindings** command in the **More Actions (...)** menu.
- This applies the `@source:user` filter to the **Keyboard Shortcuts** editor (**Source is 'User'**)



Advanced Editing of Keyboard Shortcuts

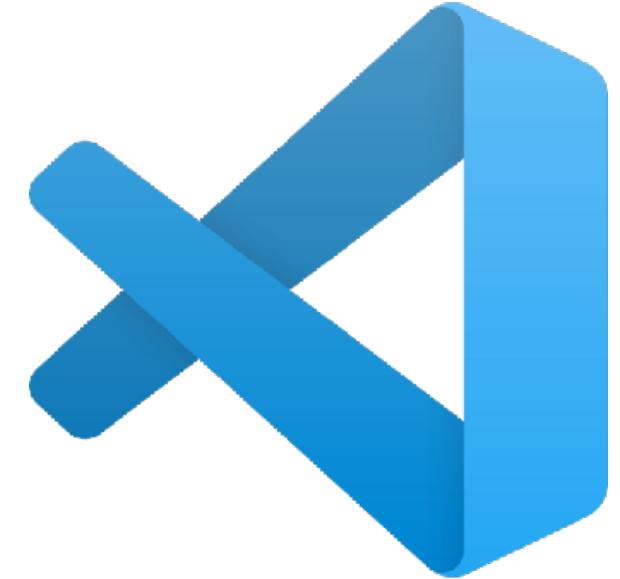
- All keyboard shortcuts in VS Code can be customized via the `keybindings.json` file.
- To configure keyboard shortcuts through the JSON file, open **Keyboard Shortcuts** editor and select the **Open Keyboard Shortcuts (JSON)** button on the right of the editor title bar. Or from Command Palette ($\text{↑ } \text{⌘ } \text{P}$) with the Preferences: **Open Keyboard Shortcuts (JSON)** command.
- This will open your `keybindings.json` file where you can overwrite the Default Keyboard Shortcuts.





Keyboard Rules

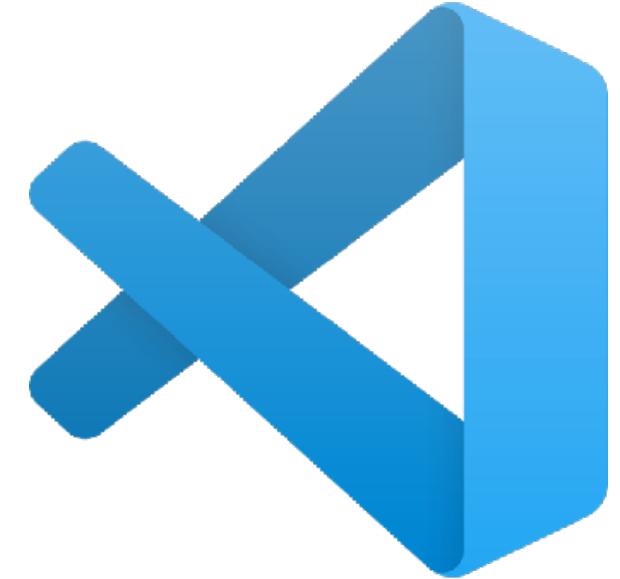
- Each Rule Consists of:
 - a *key* that describes the pressed keys.
 - a *command* containing the identifier of the command to execute.
 - an *optional when clause* containing a boolean expression that will be evaluated depending on the current context.
- Chords (two separate keypress actions) are described by separating the two keypresses with a space. For example, **Ctrl+K Ctrl+C**.



When a Key is Pressed

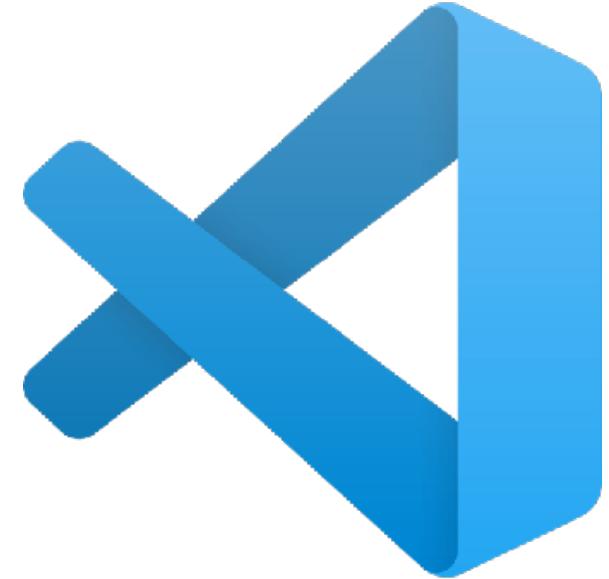
When a key is pressed:

- The rules are evaluated from bottom to top.
- The first rule that matches, both the key and in terms of when, is accepted.
- No more rules are processed.
- If a rule is found and has a command set, the command is executed.



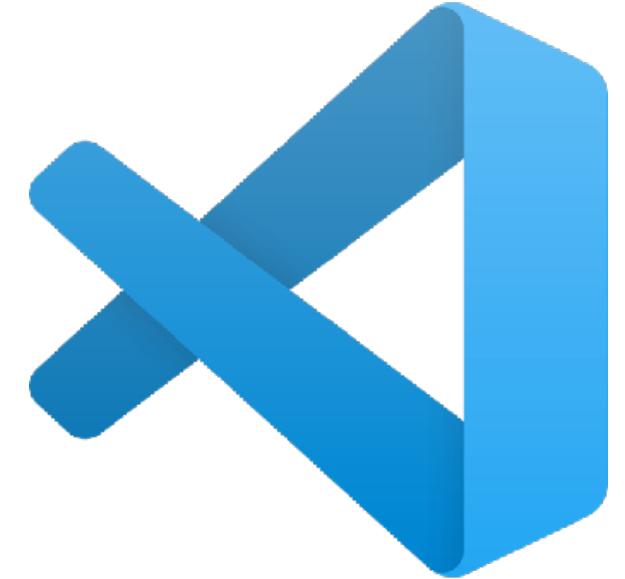
Additional Rules

- The additional **keybindings.json** rules are appended at runtime to the bottom of the default rules, thus allowing them to overwrite the default rules.
- The **keybindings.json** file is watched by VS Code so editing it while VS Code is running will update the rules at runtime.



Platform Accepted Keys

Platform	Modifiers
macOS	<code>Ctrl+</code> , <code>Shift+</code> , <code>Alt+</code> , <code>Cmd+</code>
Windows	<code>Ctrl+</code> , <code>Shift+</code> , <code>Alt+</code> , <code>Win+</code>
Linux	<code>Ctrl+</code> , <code>Shift+</code> , <code>Alt+</code> , <code>Meta+</code>



General Accepted Keys

- f1-f19, a-z, 0-9
- ` , - , = , [,] , \ , ; , ' , , . , /
- left, up, right, down, pageup, pagedown, end, home
- tab, enter, escape, space, backspace, delete
- pausebreak, capslock, insert
- numpad0-numpad9, numpad_multiply,
- numpad_add, numpad_separator
- numpad_subtract, numpad_decimal, numpad_divide



Format of Action

```
{  
  "key": "enter",  
  "command": "type",  
  "args": { "text": "Hello World" },  
  "when": "editorTextFocus"  
}
```

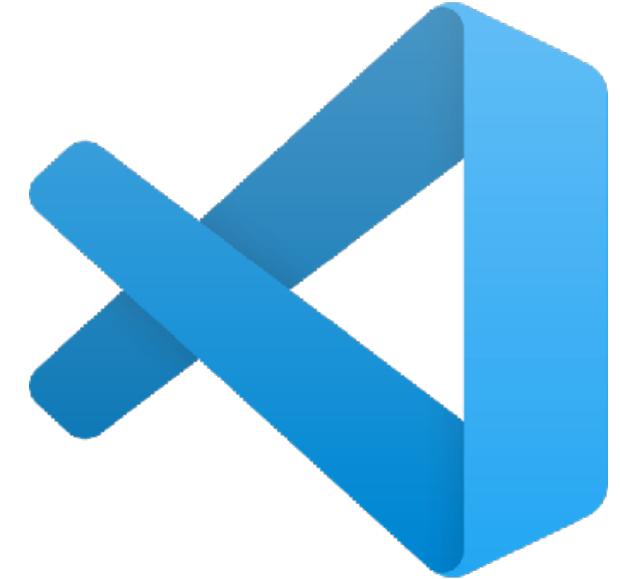


References for Crafting

- [Built in Commands](#)
- [When Contexts](#)
- Remember to view all the possible commands use **Preferences: Default Keyboard Shortcuts (JSON)** for the complete listing

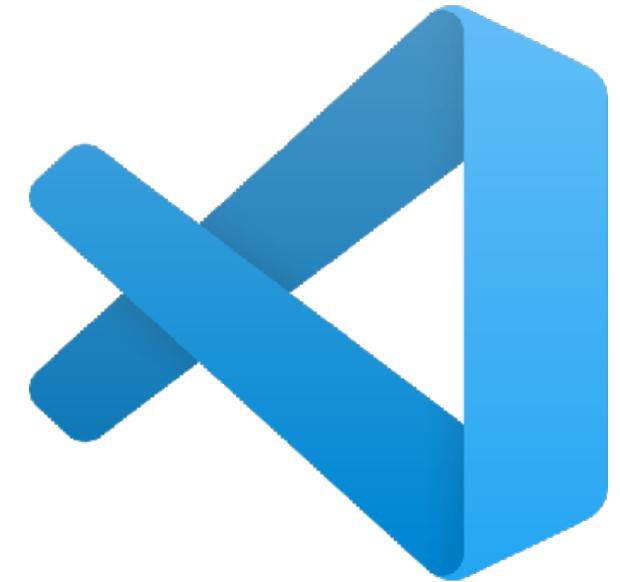


Language Specific Settings



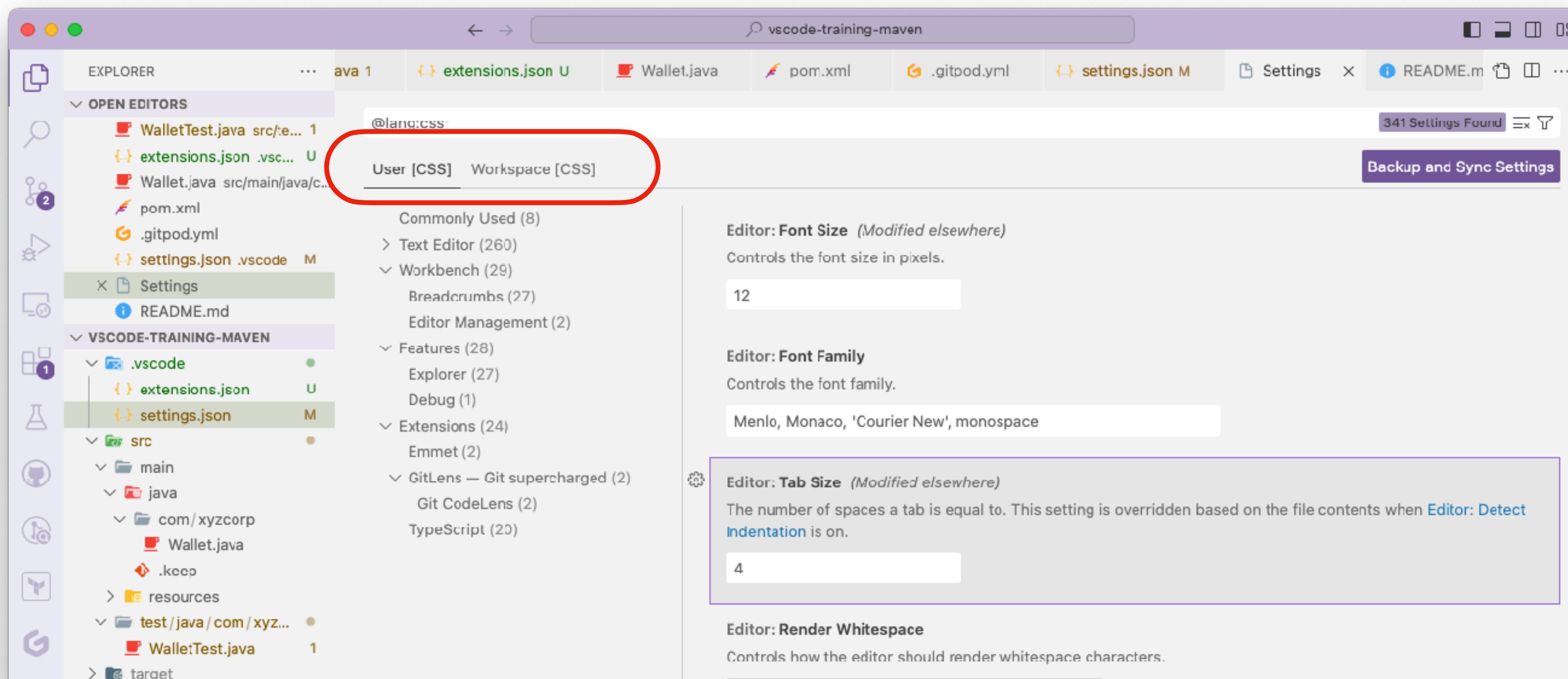
Language Specific Settings

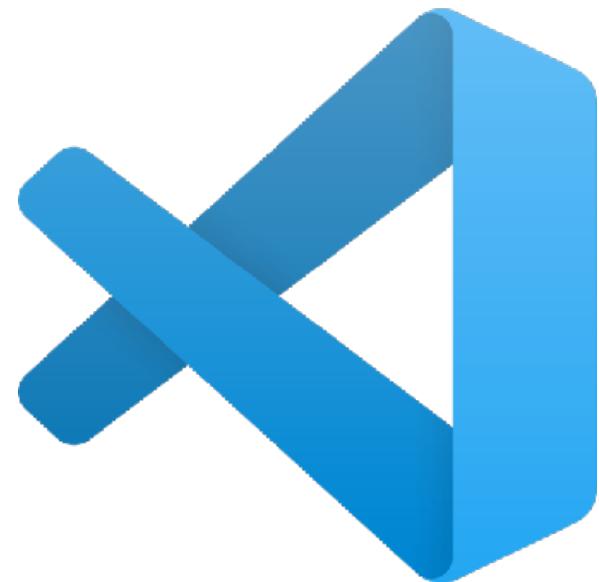
- One way to customize language-specific settings is by opening the Settings editor language option to add a language filter.
- Alternatively, one can directly type a language filter of the form `@lang:languageId` into the search widget.
- The settings that show up will be configurable for that specific language, and will show the setting value specific to that language, if applicable.



Language Specific Settings

- Settings, once filtered, will be only applied to that language
- You can also review the changes in the your `settings.json` file





Language Specific Settings

- Settings, once filtered, will be only applied to that language
 - You can also review the changes in the your `settings.json` file

The screenshot shows the VS Code interface with the following details:

- Title Bar:** The title bar displays "vscode-training-maven".
- File Explorer:** On the left, the file tree shows a project structure under "VSCODE-TRAINING-MAVEN". It includes a ".vscode" folder containing "extensions.json" and "settings.json", and a "src" folder containing "main", "java", "com/xyzcorp" (with "Wallet.java" and ".keep"), "resources", and "test/java/com/xyz...".
- Open Editors:** The "OPEN EDITORS" section shows several files: "WalletTest.java", "extensions.json", "Wallet.java", "pom.xml", ".gitpod.yml", and "settings.json". "settings.json" is the active editor.
- Code Editor:** The main area displays the "settings.json" file content:

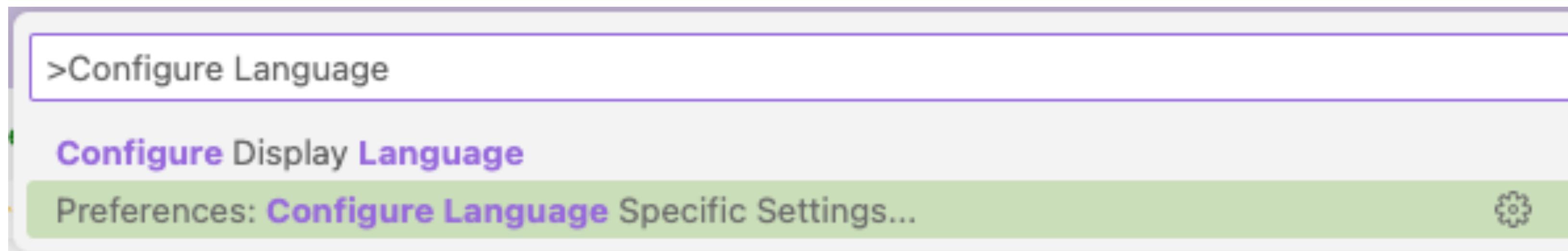
```
{
    "java.configuration.updateBuildConfiguration": "interactive",
    "java.configuration.runtimes": [
        {
            "name": "JavaSE-17",
            "path": "~/.sdkman/candidates/java/17.0.9-tem/"
        }
    ],
    "[css]": {
        "editor.fontSize": 16
    }
}
```

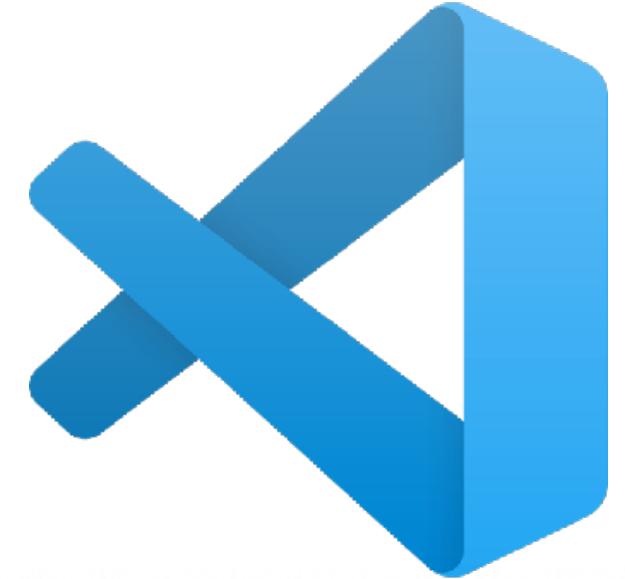
A red oval highlights the CSS editor configuration block (lines 25-27).
- Bottom Status Bar:** The status bar at the bottom indicates "You, 4 days ago added .vscode files".



Language Specific Settings

- You can also set specific settings per language by the global command **Preferences: Configure Language Specific Settings** (command ID: `workbench.action.configureLanguageBasedSettings`) from the Command Palette (⇧⌘P) which opens the language picker
- Select the language you want.
- Then, the Settings editor opens with a language filter for the selected language, which allows you to modify language-specific settings for that language.
- Though, if you have the `workbench.settings.editor` setting set to `json`, then the `settings.json` file opens with a new language entry where you can add applicable settings.





Language Settings in Editor

The screenshot shows the VS Code interface with a Java file open in the editor. The status bar at the bottom right shows 'Java'. A context menu is open in the top right corner, with the 'Configure Language based settings...' option highlighted and circled in red. Another red circle highlights the 'Java' entry in the status bar.

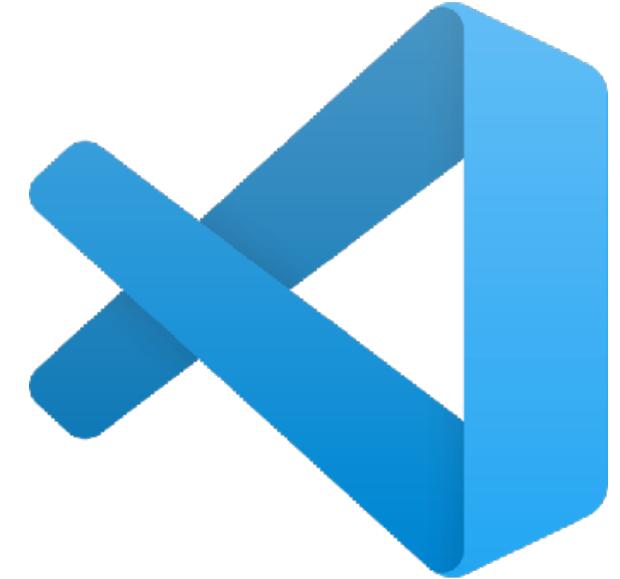
```
public boolean isEmpty() {
    logger.debug(format("Querying isEmpty in wallet {}, which is {}", uuid, balance));
    return balance == 0;
}

public void addFunds(int amount) {
    logger.debug(format("Adding {} to wallet {}", amount, uuid));
    balance += amount;
}

public void spendFunds(int amount) {
    logger.debug(format("Removing amount {} from balance {} in wallet {}", amount, balance, uuid));
    balance -= amount;
}

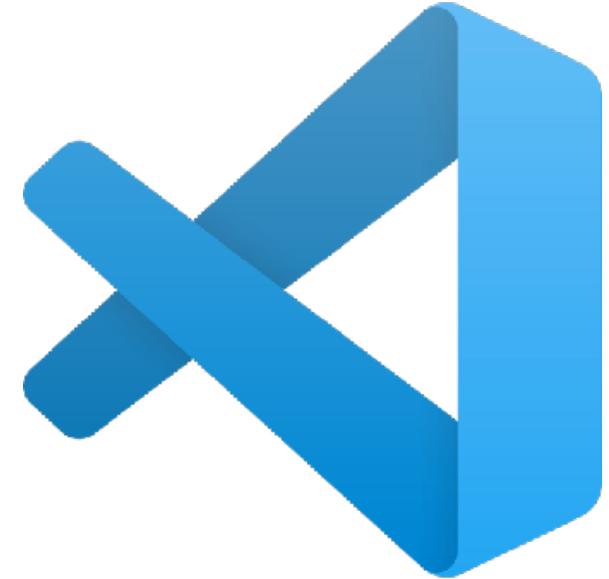
public UUID getUUID() {
    return uuid;
}
```

- If you have a file open and you want to customize the editor for this file type, select the Language Mode in the Status Bar to the bottom-right of the VS Code window.
- This opens the Language Mode picker with an option **Configure 'language_name' language based settings**.
- Selecting this opens your user **settings.json** with the language entry where you can add applicable settings.



Language Precedence

- Language-specific editor settings always override non-language-specific editor settings, even if the non-language-specific setting has a narrower scope.
- For example, language-specific user settings override non-language-specific workspace settings.
- You can scope language-specific settings to the workspace by placing them in the workspace settings just like other settings.
- **If you have settings defined for the same language in both user and workspace scopes, then they are merged by giving precedence to the ones defined in the workspace.**
- View More in the [VS Code Language Precendence documentation](#)



Multilanguage Settings

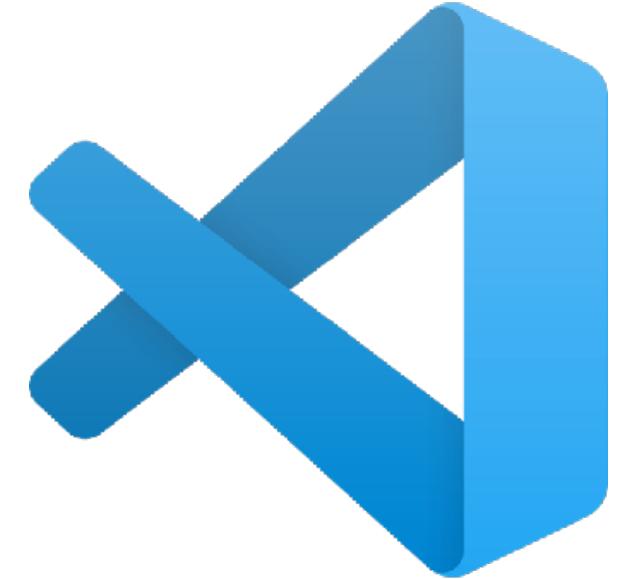
```
{  
  "[typescript)": {  
    "editor.formatOnSave": true,  
    "editor.formatOnPaste": true  
  },  
  "[markdown)": {  
    "editor.formatOnSave": true,  
    "editor.wordWrap": "on",  
    "editor.renderWhitespace": "all",  
    "editor.acceptSuggestionOnEnter": "off"  
  }  
}
```

NOTE: You can review default language-specific settings already set in [defaultSettings.json](#) by running the **Preferences: Open Default Settings** command.



Merging Multilanguage Settings

```
"[javascript][typescript)": {  
    "editor.maxTokenizationLineLength": 2500  
}
```



Syncing your Settings

- You can share your user settings across your VS Code instances with the **Settings Sync** feature.
- This feature lets you share settings, keyboard shortcuts, and installed extensions across your VS Code installs on various machines.
- You can enable Settings Sync via the **Turn on Settings Sync** command on the right of the Settings editor or on the **Accounts Activity Bar** context menu.



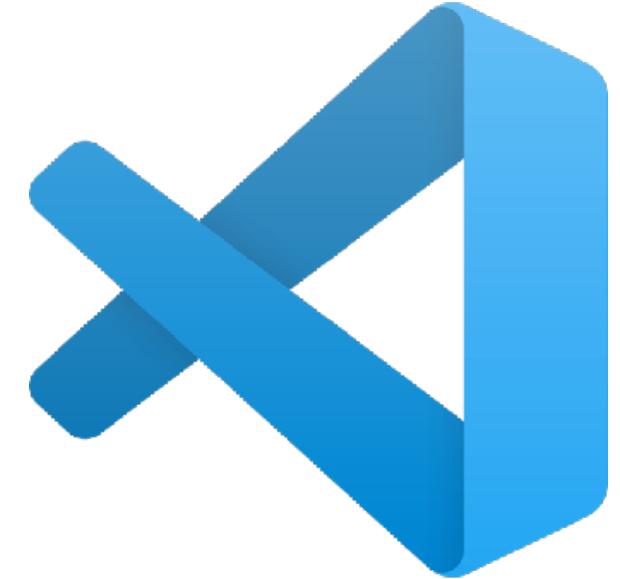
Review your Keybindings



- In this lab let's just review your keybindings
- Change any keybindings that you require or want to change.



Snippets



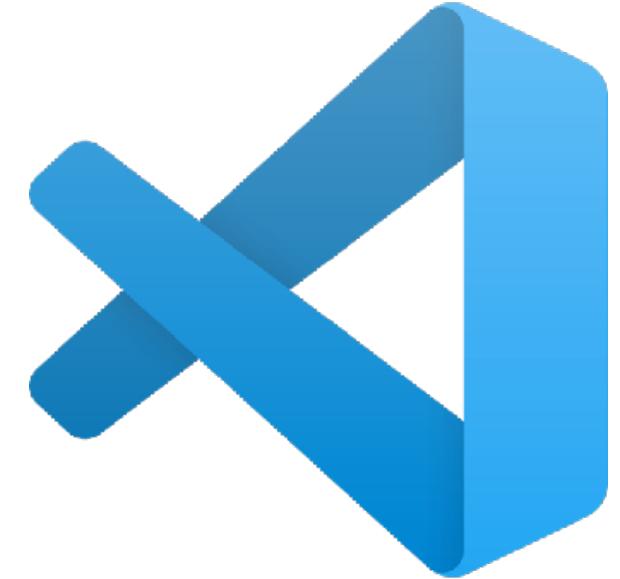
Snippets

- Code snippets are templates that make it easier to enter repeating code patterns, such as loops or conditional-statements.
- In Visual Studio Code, snippets appear in IntelliSense ([^Space](#)) mixed with other suggestions, as well as in a dedicated snippet picker ([Insert Snippet](#) in the Command Palette).
- There is also support for tab-completion: Enable it with `"editor.tabCompletion": "on"`, type a **snippet prefix** (trigger text), and press [Tab](#) to insert a snippet.
- You can see the available snippets for a language by running the [Insert Snippet](#) command in the Command Palette to get a list of the snippets for the language of the current file



Finding More Snippets

- Many extensions on the VS Code Marketplace include snippets.
- You can search for extensions that contains snippets in the Extensions view ([⬆️🔍X](#)) using the `@category:"snippets"` filter.
- Restart VSCode once, installed



Creating Your Own

- To create or edit your own snippets, select **Configure User Snippets** under **Code > Settings**, and then select the language (by language identifier) for which the snippets should appear
- For global snippets for common items select **New Global Snippets file** option if they should appear for all languages.
- VS Code manages the creation and refreshing of the underlying snippets file(s) for you.
- Snippets files are written in JSON, support C-style comments, and can define an unlimited number of snippets.
- Snippets intelligently format whitespace based on the insertion context, and allow easy multiline editing.



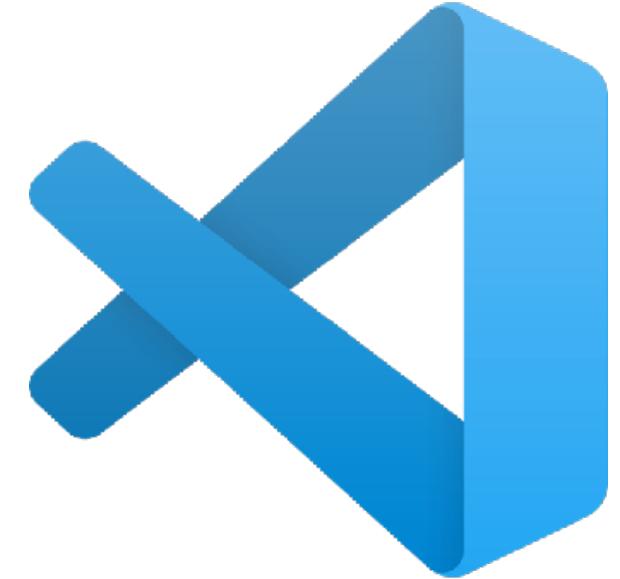
Anatomy of a Snippet

```
// in file 'Code/User/snippets/javascript.json'  
{  
  "For Loop": {  
    "prefix": ["for", "for-const"],  
    "body": ["for (const ${2:element} of ${1:array}) {}", "\t$0", "}],  
    "description": "A for loop."  
  }  
}
```

The diagram illustrates the structure of a snippet definition in `javascript.json`. It highlights several key components:

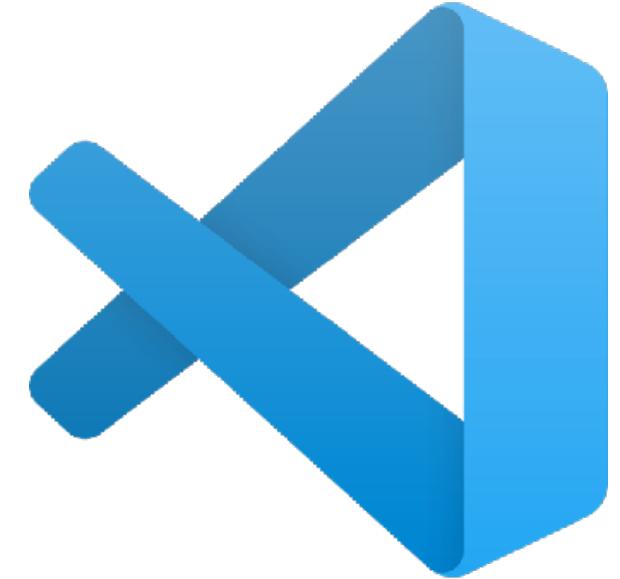
- Prefix Triggers**: The triggers for this snippet, indicated by arrows pointing to the values in the `prefix` field: `"for"` and `"for-const"`.
- Default Values**: The default values for variables in the body, indicated by arrows pointing to the values in the `body` field: `{}
\t$0
[]`.
- Traversal Order**: The order in which variables are traversed, indicated by arrows pointing to the variable names in the `body` field: `${2:element}
${1:array}`.
- Description for Help**: The description provided for this snippet, indicated by an arrow pointing to the value in the `description` field: `"A for loop."`.
- Body**: The main body of the snippet, indicated by an arrow pointing to the outermost curly braces of the JSON object.

Find more advanced syntax snippet settings in the [VSCode Documentation](#)



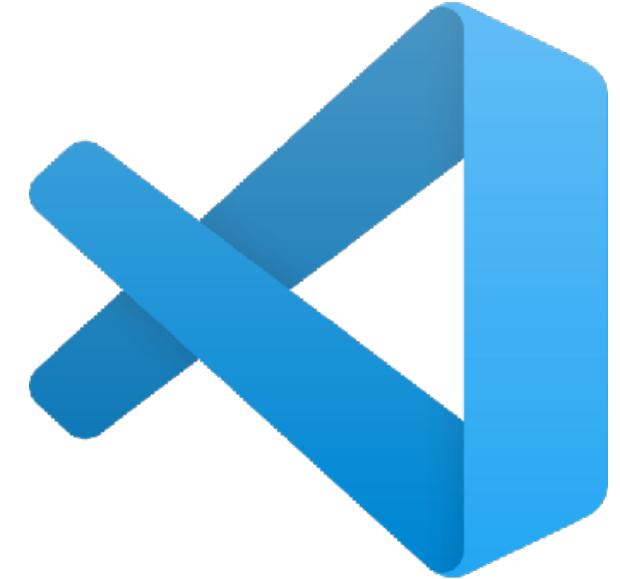
File Snippet

- You can add the `isFileTemplate` attribute to your snippet's definition if the snippet is intended to populate or replace a file's contents.
- File template snippets are displayed in a dropdown when you run the Snippets: **Populate File from Snippet** command in a new or existing file.



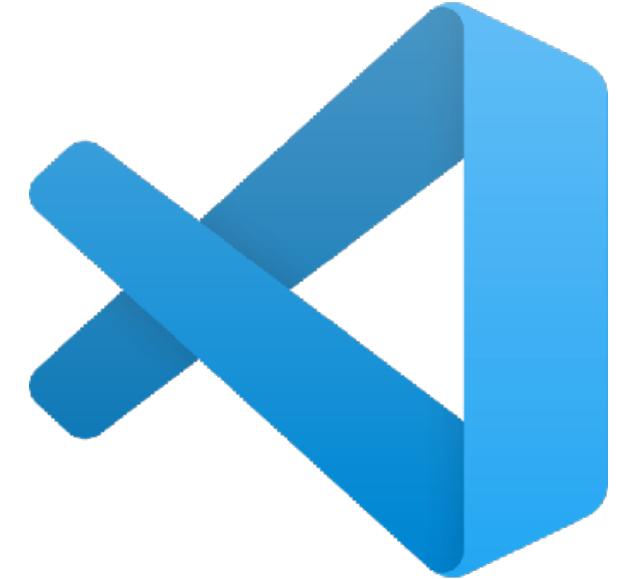
Snippet Scope

- Snippets are scoped so that only relevant snippets are suggested. Snippets can be scoped by either:
 - the language(s) to which snippets are scoped (possibly all)
 - the project(s) to which snippets are scoped (probably all)
- Every snippet is scoped to one, several, or all ("global") languages based on whether it is defined in:
 - a language snippet file
 - a global snippet file



Single and Multilanguage

- Single-language user-defined snippets are defined in a specific language's snippet file (for example `javascript.json`), which you can access by language identifier through **Snippets: Configure User Snippets**.
 - A snippet is only accessible when editing the language for which it is defined.
- Multi-language and global user-defined snippets are all defined in "global" snippet files (JSON with the file suffix `.code-snippets`), which is also accessible through **Snippets: Configure User Snippets**.
 - In a global snippets file, a snippet definition may have an additional `scope` property that takes one or more language identifiers, which makes the snippet available only for those specified languages.
 - If no `scope` property is given, then the global snippet is available in all languages.



Project Snippet Scope

- You can also have a global snippets file (JSON with file suffix `.code-snippets`) scoped to your project.
- Project-folder snippets are created with the **New Snippets file for "... option** in the **Snippets: Configure User Snippets** dropdown menu and are located at the root of the project in a `.vscode` folder.
- Project snippet files are useful for sharing snippets with all users working in that project since it is a file that will be typically version controlled.
- Project-folder snippets are similar to global snippets and can be scoped to specific languages through the scope property.



Assigning Key and Lang

Bindings in `keybindings.json`

```
{  
  "key": "cmd+k 1", ← Key Binding  
  "command": "editor.action.insertSnippet",  
  "when": "editorTextFocus",  
  "args": {  
    "snippet": "console.log($1)$0" ← Snippet  
  }  
}
```



Assigning Key and LangRef

```
{  
  "key": "cmd+k 1", ← Key Binding  
  "command": "editor.action.insertSnippet",  
  "when": "editorTextFocus",  
  "args": {  
    "langId": "csharp",  
    "name": "myFavSnippet" ← Snippet Reference  
  }  
}
```

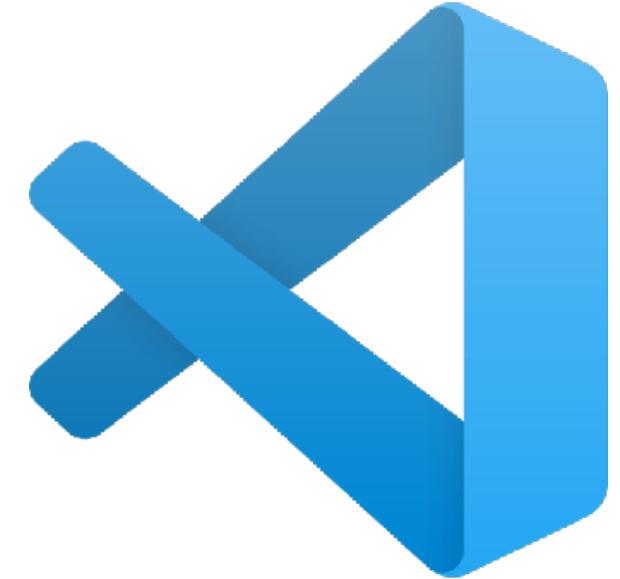


Tasks



Tasks

- Lots of tools exist to automate tasks like linting, building, packaging, testing, or deploying software systems.
- Examples include the TypeScript Compiler, linters like ESLint and TSLint as well as build systems like Make, Ant, Gulp, Jake, Rake, and MSBuild.
- Tasks in VS Code can be configured to run scripts and start processes so that many of these existing tools can be used from within VS Code without having to enter a command line or write new code.
- Workspace or folder specific tasks are configured from the `tasks.json` file in the `.vscode` folder for a workspace
- To run any tasks that have been configured, use `⇧⌘B`

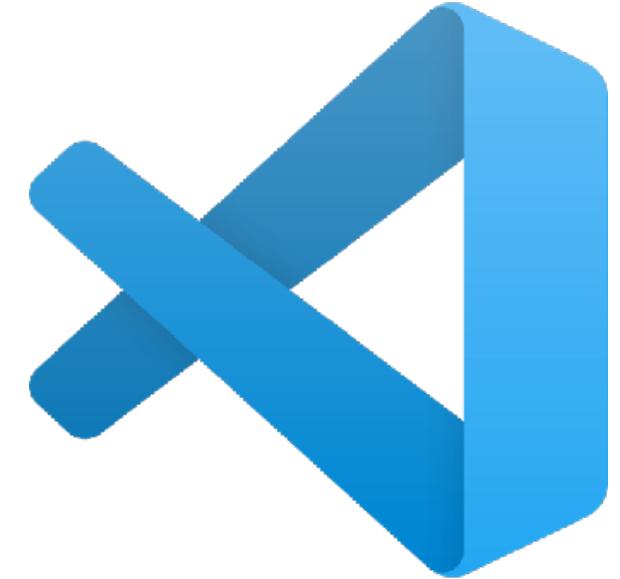


Creating Custom Tasks

- Lots of tools exist to automate tasks like linting, building, packaging, testing, or deploying software systems.
- Examples include the TypeScript Compiler, linters like ESLint and TSLint as well as build systems like Make, Ant, Gulp, Jake, Rake, and MSBuild.
- Tasks in VS Code can be configured to run scripts and start processes so that many of these existing tools can be used from within VS Code without having to enter a command line or write new code.
- Workspace or folder specific tasks are configured from the `tasks.json` file in the `.vscode` folder for a workspace
- To run any tasks that have been configured, use ⌘B

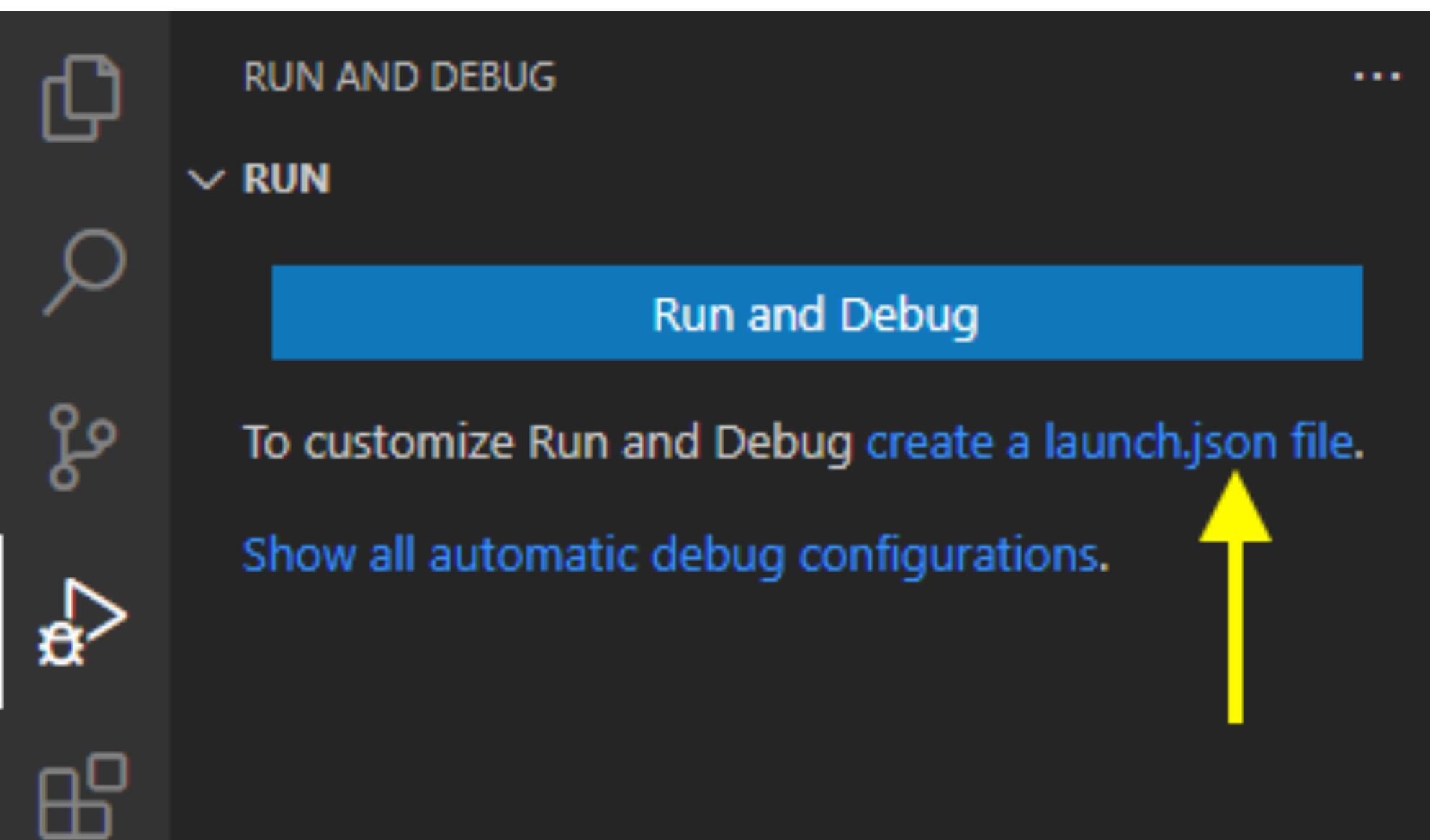


Launch



Launch

- For most language any debugger and main class is automatically configured and handled by VSCode
- If you would like to customize and persist your launch configuration, you can select the **create a launch.json file** link in the **Run and Debug** view.
- The **launch.json** file is located in a **.vscode** folder in your workspace (project root folder).



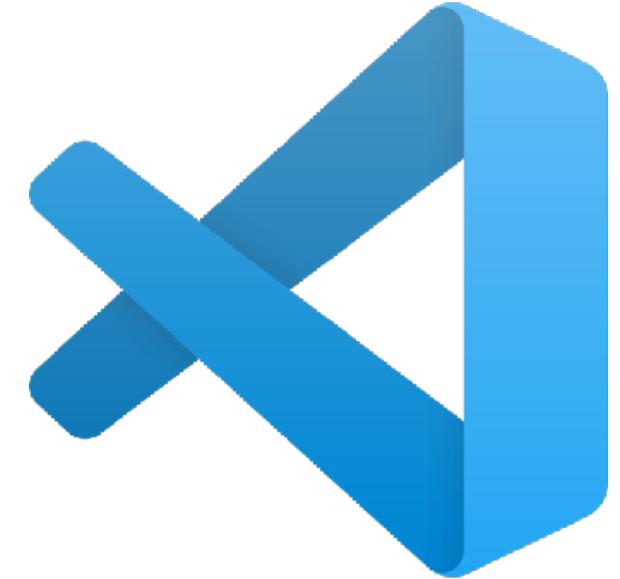


Determining Launch Environments

- You can then select the launch environments that you wish to create from a list of options
- This will in turn create a `launch.json` that you can configure
- For debugging, In order to start a debug session, first select the configuration named **Launch Program** using the Configuration dropdown in the **Run and Debug** view. Once you have your launch configuration set, start your debug session with **F5**.
- Alternatively, you can run your configuration through the Command Palette (**⬆️⌘P**) by filtering on **Debug: Select and Start Debugging** or typing '`debug`' and selecting the configuration you want to debug.

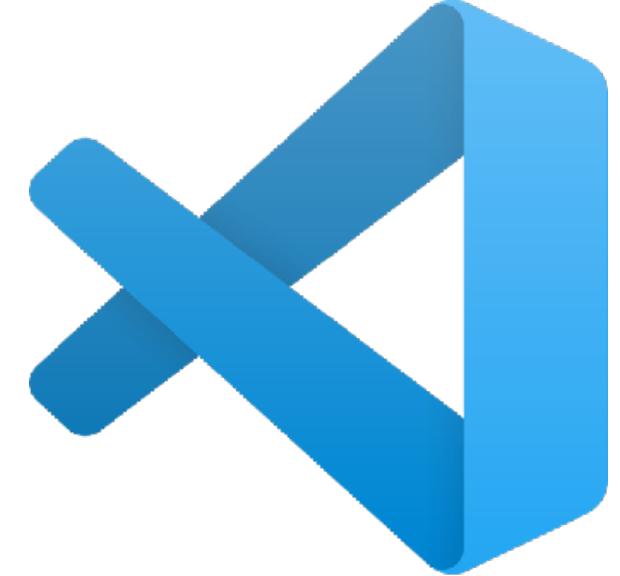


Profiles



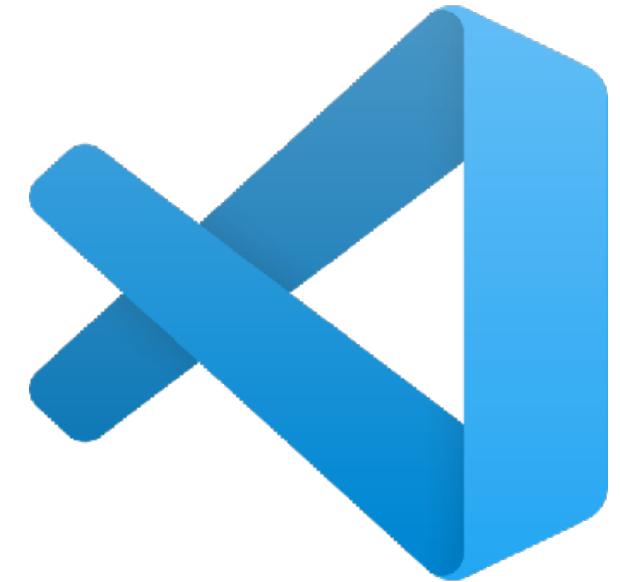
Profiles

- Visual Studio Code has hundreds of [settings](#), thousands of [extensions](#), and innumerable ways to adjust the UI layout to customize the editor.
- **VS Code Profiles** let you create sets of customizations and quickly switch between them or share them with others.



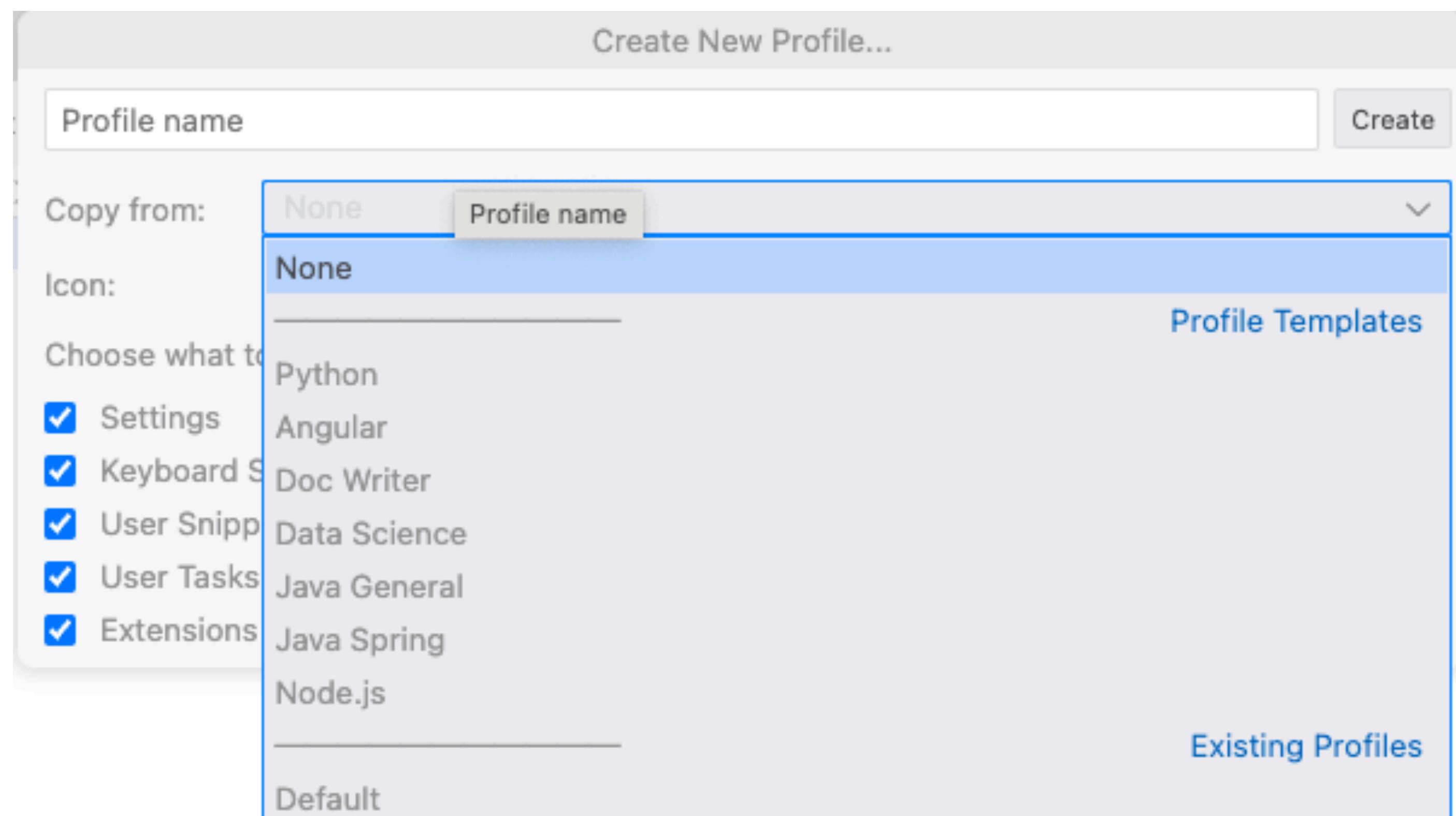
Creating a Profile

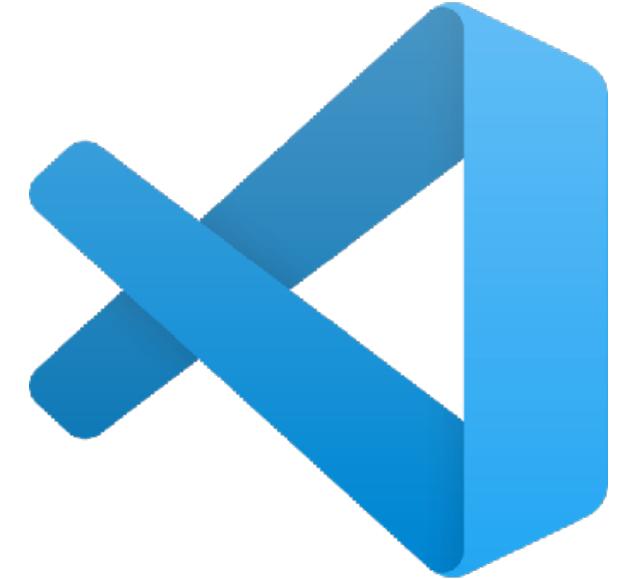
- VS Code treats your current configuration as the **Default Profile**. As you modify settings, install extensions, or change UI layout by moving views, these customizations are tracked in the **Default Profile**.
- To create a new profile, you can use the **Code > Settings > Profiles > Create Profile** menu item.
- You can create a new profile based on the current profile (**Profiles: Create from Current Profiles**) or create an **Empty Profile**. An Empty Profile includes no user customizations (settings, extensions, snippets, etc.).
- You can also access the Profile command menu via the **Manage** gear button in the lower right of the Activity bar.
- It is also available in the Command Palette as always (**⇧⌘P**)



Profile Template

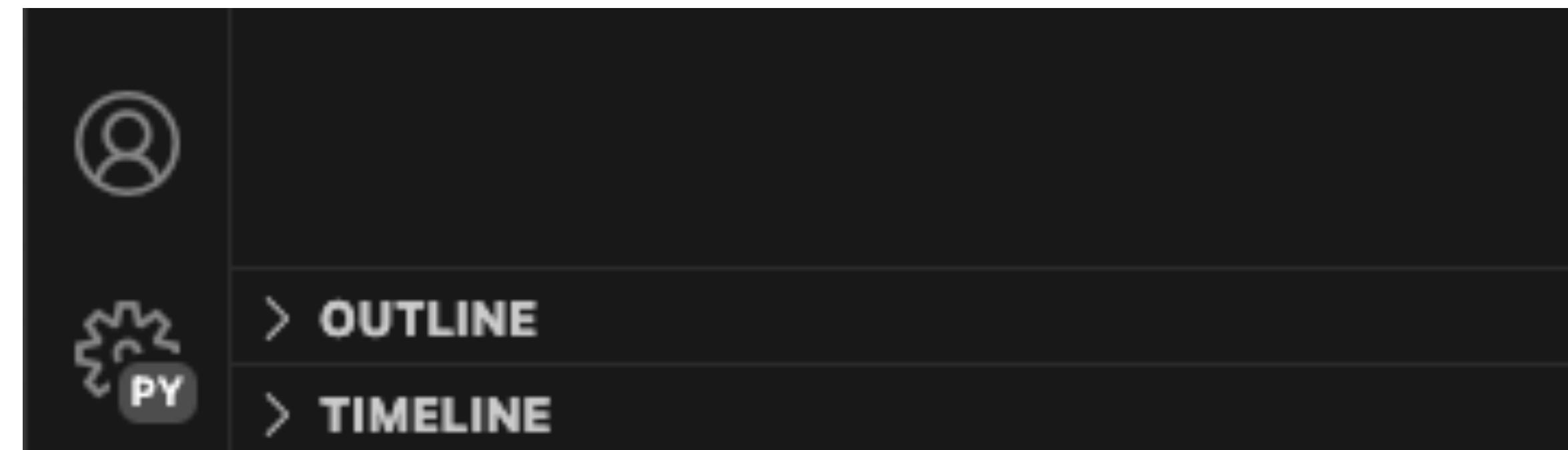
You can choose to create a new profile by copying from a Profile Template or from an existing profile or create an Empty Profile in the Profile creation dialog.

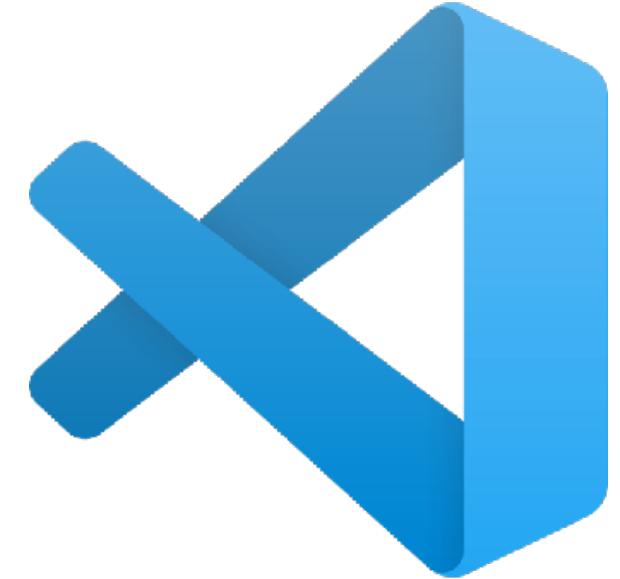




Partial Profile

- You can choose whatever icon you would like for your profile
- But, if you leave it as a “gear”, the **Manage** gear button displays a badge with the first two letters of the active profile so you can quickly check which profile you are running





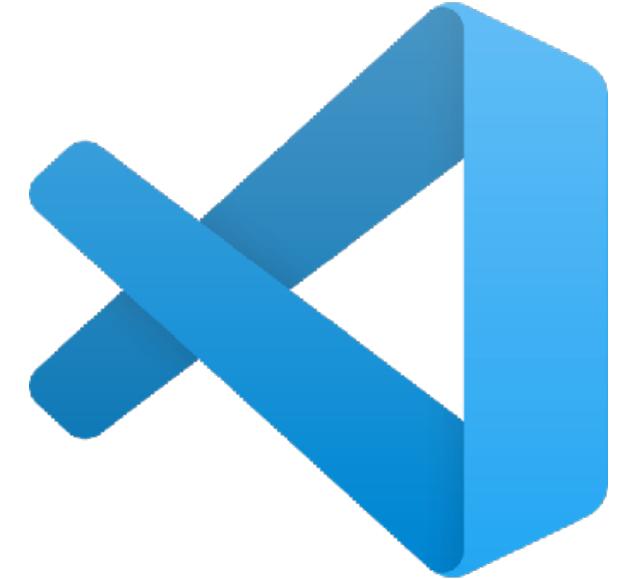
Configure a Profile

- You can configure a profile just as you would normally change any VS Code configuration.
- You can install/uninstall/disable extensions, change settings, and adjust the editor's UI layout (for example, moving and hiding views) like normal.
- These changes are stored in your currently active profile.



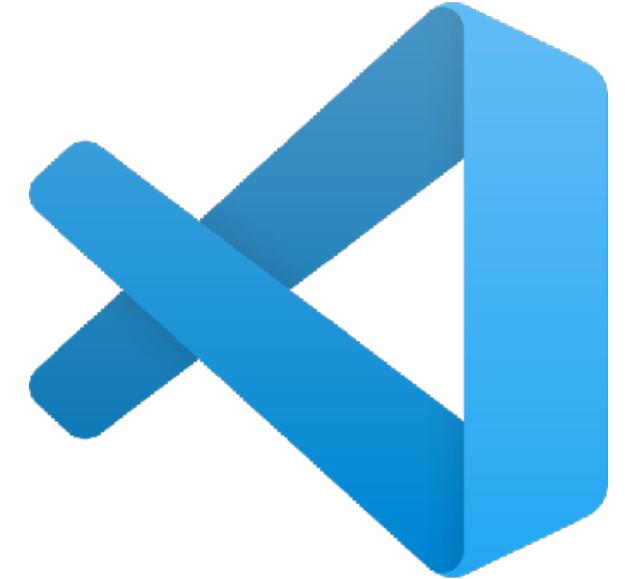
Workspace Associations

- When you create or select an existing profile, it is associated with the current workspace and whenever you open that folder, the workspace's profile is active.
- If you open another folder, the profile switches to that folder's associated profile if one has been set or remains on the last used profile.



Viewing Profile Contents

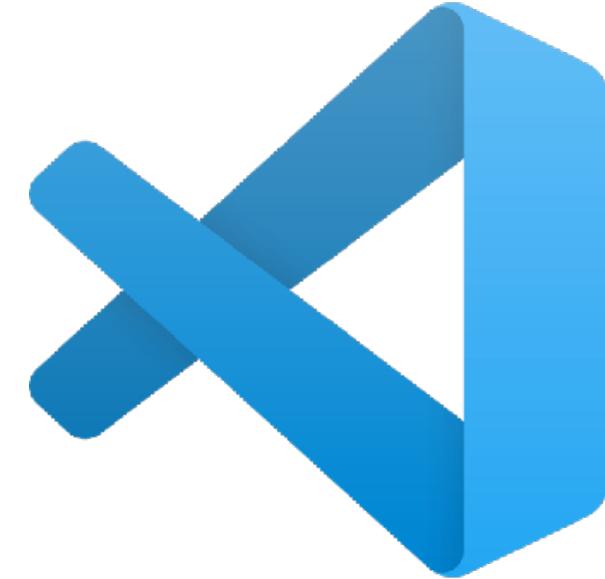
- The **Profiles: Show Contents** command (available in the Command Palette or Profiles menus) brings up the Profiles view, where you can review the customizations for the profile.
- A profile can include:
 - Settings - In a profile-specific `settings.json` file.
 - Extensions - The list of extensions included in the current profile.
 - UI state - View layout (positions), visible views and actions.
 - Keybindings - In a profile-specific `keybindings.json` file.
 - Snippets - In a profile-specific `{language}.json` files.
 - User Tasks - In a profile-specific `tasks.json` file.



Apply a Setting to All Profiles

A setting can be applied to all profiles using the **Apply Setting to all Profiles** action available in the Settings editor.

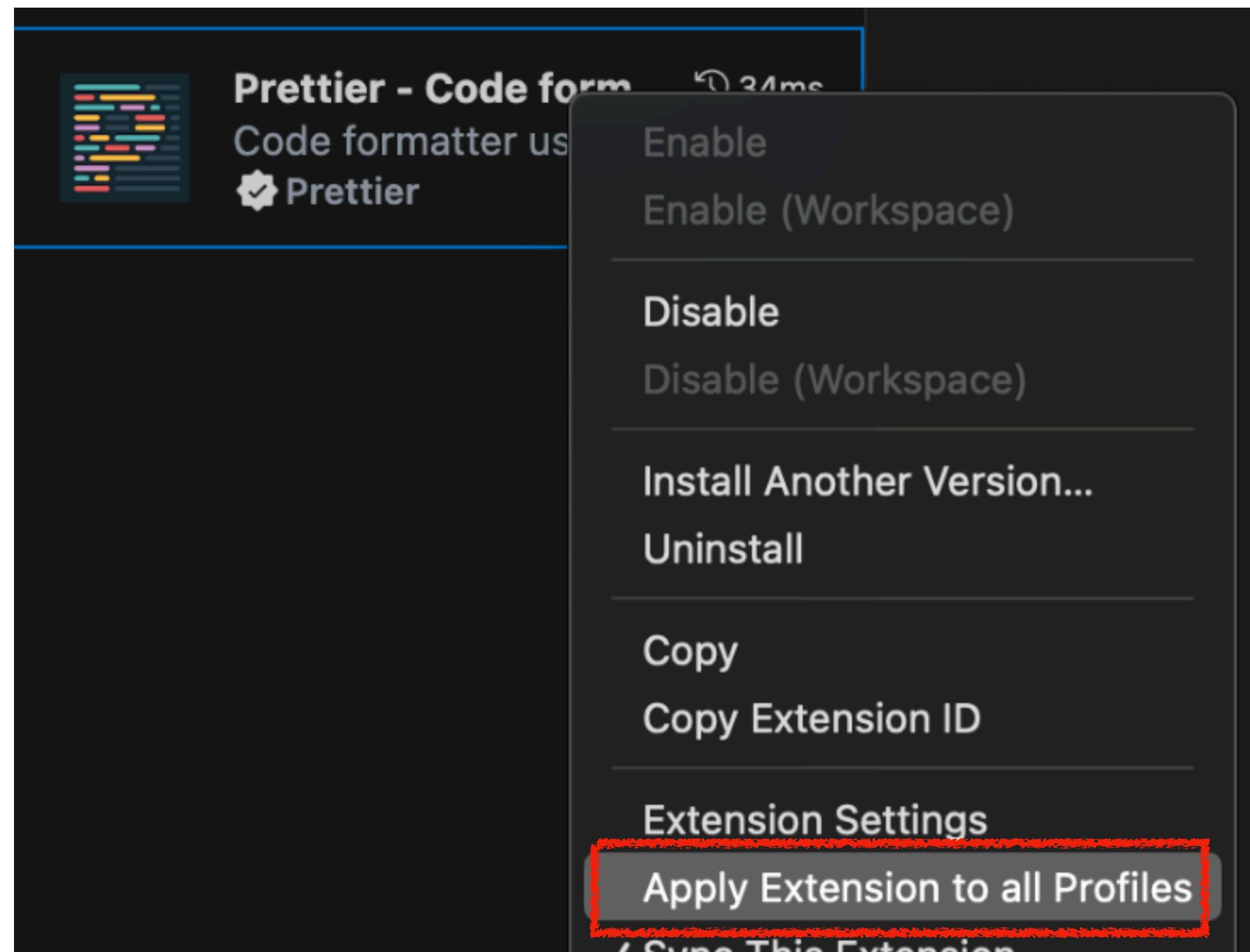
The screenshot shows the VS Code Settings editor interface. The title bar says "Settings". Below it is a search bar with the placeholder "Search settings". A "User" tab is selected, indicated by an underline. On the left, there's a sidebar titled "Commonly Used" with items: Text Editor, Workbench, Window, Features, Application, Security, and Extensions. The main area shows a "Commonly Used" section with a gear icon and the heading "Files: Auto Save". A context menu is open over the "Files: Auto Save" section, listing: Reset Setting, Copy Setting ID, Copy Setting as JSON, **Apply Setting to all Profiles** (which is highlighted with a red border), and Sync This Setting.

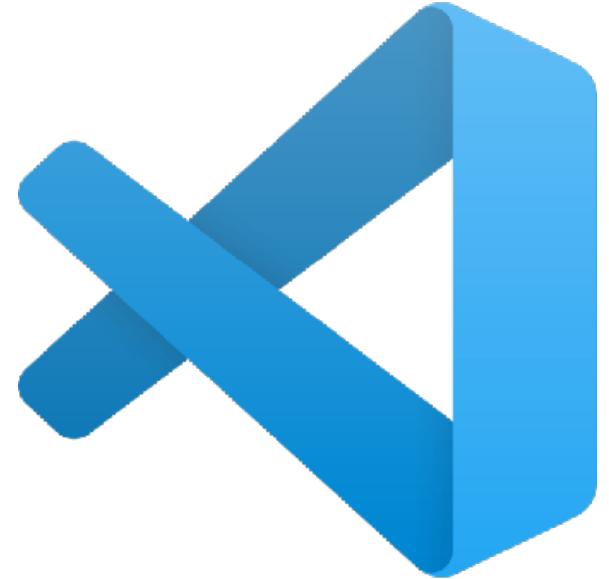


Apply an Extension to All Profiles

An extension can be applied to all profiles by clicking on the **Apply Extension to all Profiles** action in the Extensions view.

You can always revert this by unchecking the **Apply Extension to all Profiles** action.





Exporting and Importing

Exporting Profiles:

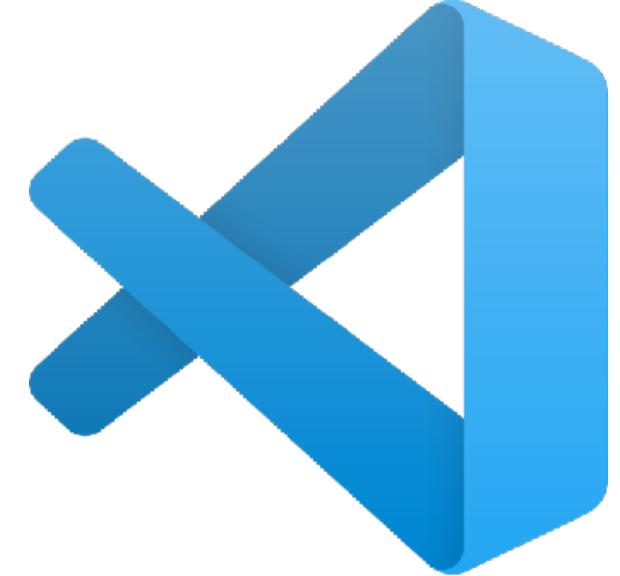
- You can export a profile in order to save it or share it with others.
- The **Export Profile** command displays the Profiles view with the contents of the active profile and an **Export** button
- Export can be save as a local file or as a Github Gist

Importing Profiles

- To import an existing profile, run the **Import Profiles** command.
- You are prompted for the URL of a GitHub gist or the file location of a profile via an **Import Profile** dialog.
- Once you have selected the profile, the Profiles view opens and displays the profile to import.
- You can unselect some profile elements if you don't want to import them.
- Select the **Import Profile** button

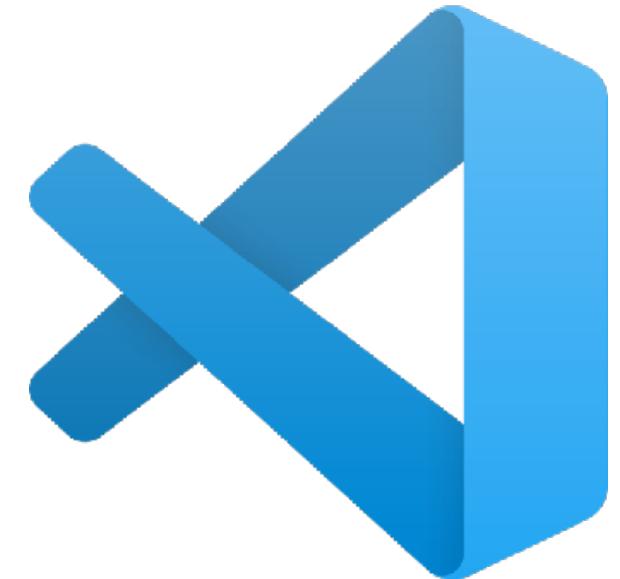


Projects

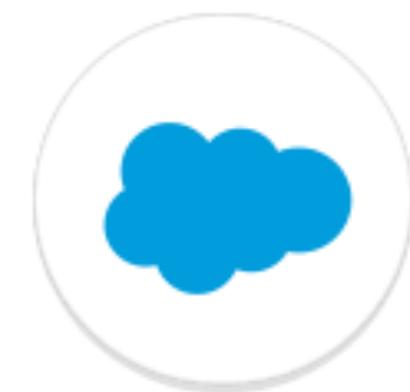


Salesforce Projects

- Salesforce Workspaces
 - Not be confused with VSCode Workspaces
 - Workspaces allows to select a code repository and bring that in as a VSCode Session
 - This includes an online version, much like [gitpod.io](#) and Codespaces
 - Currently only used for Salesforce Core
- Code Genie is the Salesforce AI Engine for Developers
 - It is available to *all Salesforce engineers*
 - *Trained on Salesforce codebases*
- Mulesoft also will have their own Workspaces



Salesforce Extension Pack



Salesforce Extension Pack v60.2.3

Salesforce [salesforce.com](#) | ↗ 1,268,445 | ★★★★★(55)

Extensions for developing on the Salesforce Platform

[Install](#)

[DETAILS](#) [FEATURES](#) [CHANGELOG](#)

Extension Pack (9)



Salesforce CLI Integration

Provides integration with the Salesforce CLI

[Install](#)

Apex

Provides code-editing features for the Ap...

[Install](#)

Aura Components

Provides code-editing features for Aura C...

[Install](#)

Categories

[Extension Packs](#)

Resources

[Marketplace](#)

[Issues](#)

[Repository](#)

[License](#)

[Salesforce](#)

More Info

Published 2017-09-21,
10:31:10

Last released 2024-02-28,
12:14:14

Identifier [salesforce.salesforce](#)
vscode

Salesforce Extensions for Visual Studio Code

This extension pack includes tools for developing on the Salesforce platform in the lightweight, extensible VS Code editor. These tools provide features for working with development orgs (scratch orgs, sandboxes, and DE orgs), Apex, Aura components, and Visualforce.



Mulesoft Extension Pack



Anypoint Extension Pack v1.0.7

Salesforce [salesforce.com](#) | ⚡ 7,511 | ★★★★★ (11)

API design, implementation, and integration development with Anypoint Code Builder

[Install](#)

[DETAILS](#) [FEATURES](#) [CHANGELOG](#)

Extension Pack (7)



Anypoint Code Builder - Dependencies ...

Contains Anypoint Code Builder depende...

Salesforce

[Install](#)

Anypoint Code Builder - Platform Exten...

Provides common Anypoint Code Builder f...

Salesforce

[Install](#)

Anypoint Code Builder - Integration Ext...

Create integration applications that run on...

Salesforce

[Install](#)

Categories

[Extension Packs](#)

Resources

[Marketplace](#)

[License](#)

[Salesforce](#)

More Info

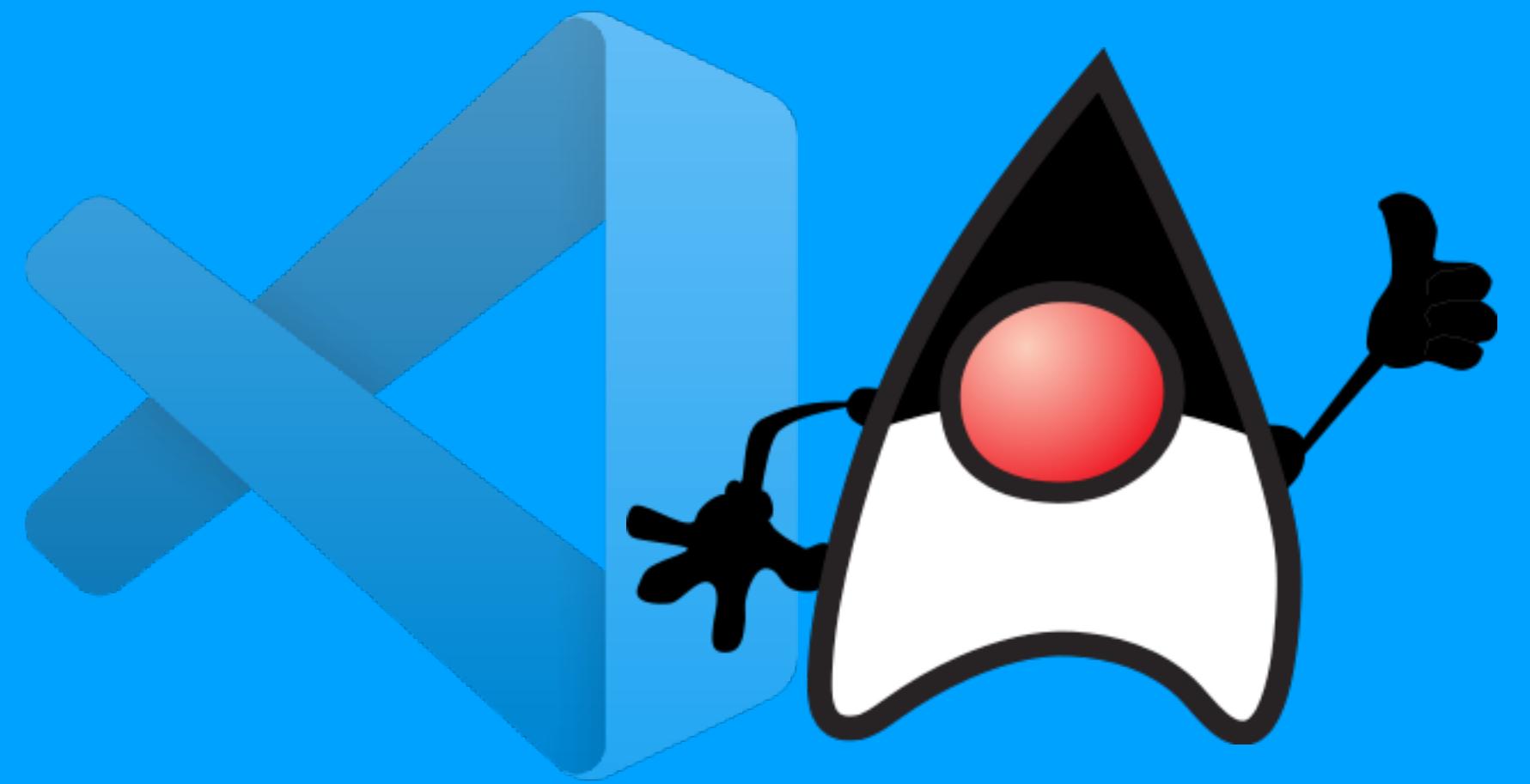
Published 2023-08-09,
19:58:52

Last released 2024-02-14,
07:53:51

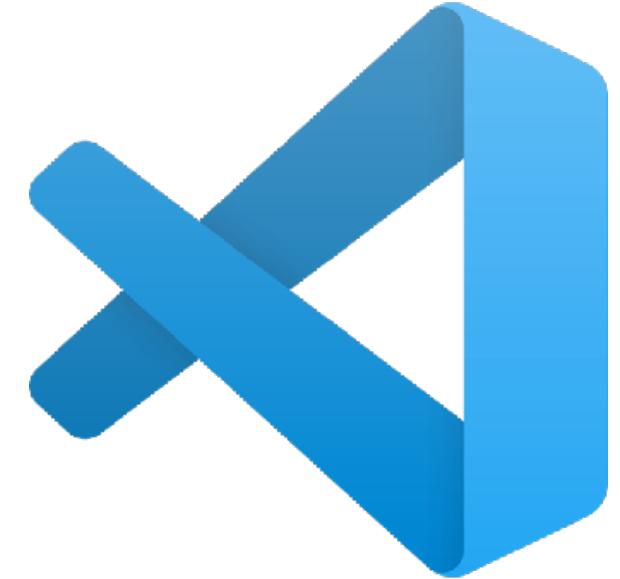
Identifier salesforce.mule-
dx-extension-pack

Anypoint Extensions for Visual Studio Code

This extension pack includes tools for developing on the Anypoint platform in the desktop version of the extensible VS Code editor. These tools provide features for API design, API implementation, integration development, debugging, and testing.

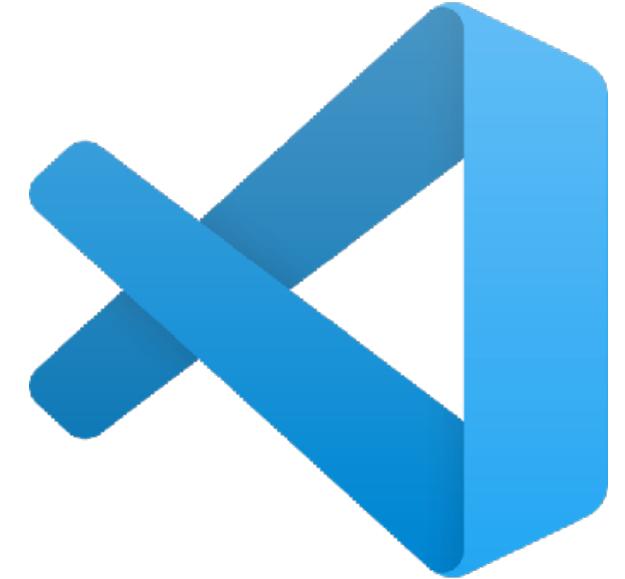


Java



Java and VSCode

- VS Code provides essential language features such as code completion, refactoring, linting, formatting, and code snippets along with convenient debugging and unit test support.
- VS Code also integrates with tooling and frameworks such as Maven, Tomcat, Jetty, and Spring Boot.
- Leveraging the power of Visual Studio Code, Java developers get an excellent tool for both quick code editing and also the full debugging and testing cycle.
- Underneath VSCode Uses an Eclipse Engine



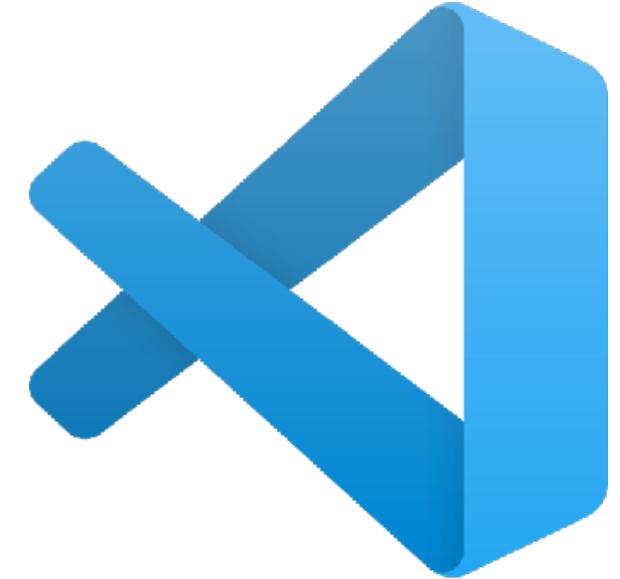
Java Coding Pack

- To help you set up quickly, there is a **Coding Pack for Java**, which is the bundle of VS Code, the Java Development Kit (JDK), and a collection of suggested extensions by Microsoft.
- You can also just use your own JDK, which you have installed either from Brew, SDKMan, or direct download and configured.



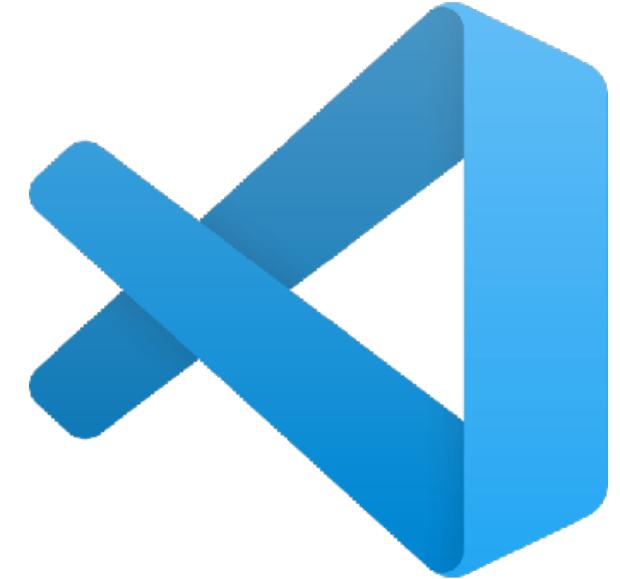
Recommended Plugins

- If you have already installed VS Code and want to add Java support to it, we recommend using the Extension Pack for Java, a collection of extensions suggested by Microsoft:
 - [Language Support for Java™ by Red Hat](#)
 - [Debugger for Java](#)
 - [Test Runner for Java](#)
 - [Maven for Java](#)
 - [Project Manager for Java](#)
 - [Visual Studio IntelliCode](#)
- You can also install the [Java Extension Pack](#)



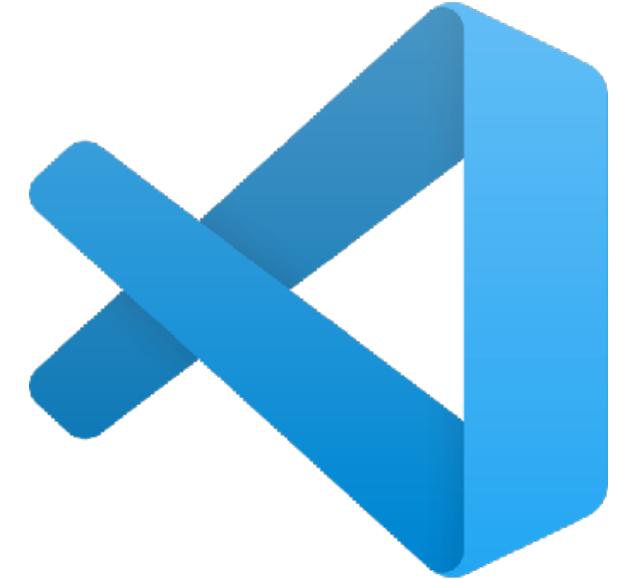
VS Code Workspaces

- You can have:
 - Folder Workspace - when you open a filesystem folder (directory) in VS Code
 - Multiroot Workspaces - can refer to multiple folders (directories) from disparate parts of the file system and VS Code displays the contents of the folder(s) of the workspace together in the [File Explorer](#).
 - Single Java File Editing - if you need to quickly enter and manipulate a single Java file



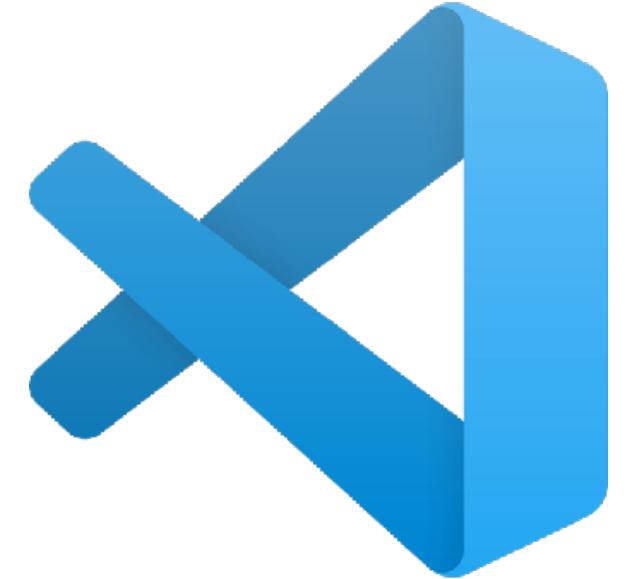
VS Code Modes

- VS Code for Java supports two modes, **lightweight** and **standard**
- With **lightweight mode**, only source files and JDK are resolved by the language server.
 - Great for quick-start and lightweight environment to work with your source files, for example, reading source code, navigating among source code and JDK, viewing outline and Javadoc, and detecting and fixing syntax errors.
- With **standard mode**, imported dependencies are resolved and the project is built by the language server.
 - Great for full development experience: running, debugging, refactoring, linting, or detecting semantic errors



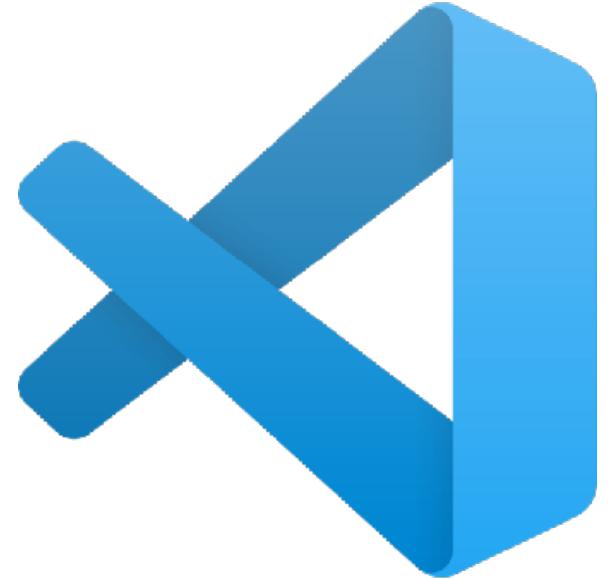
VS Setting Code Modes

- You can control which mode to launch with by configuring `java.server.launchMode` with the options below:
- **Hybrid** (default) - Firstly, a workspace is opened with lightweight mode. You will be asked whether to switch to standard mode if your workspace contains unresolved Java projects. If you choose Later, it will stay in lightweight mode. You can click the language status item on the Status bar to manually switch to standard mode.
- **Standard** - A workspace is opened with standard mode.
- **LightWeight** - A workspace is opened with lightweight mode. You can click the language status item on the Status bar to manually switch to standard mode.



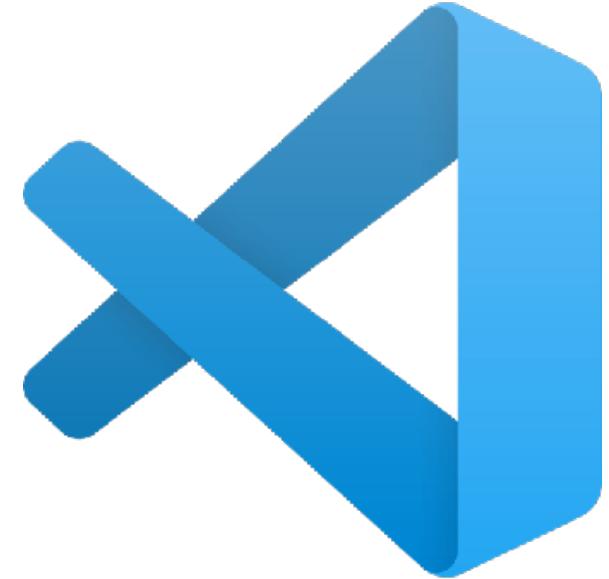
Configuring Java Runtimes

- As Java evolves, it's common that developers work with multiple versions of JDK.
- You can map them to your local installation paths via the setting: `java.configuration.runtimes`.
- You can also set the default runtime to "default": `true`



Configuring Runtimes

```
"java.configuration.runtimes": [
    {
        "name": "JavaSE-1.8",
        "path": "/usr/local/jdk1.8.0_201"
    },
    {
        "name": "JavaSE-11",
        "path": "/usr/local/jdk-11.0.3",
        "sources": "/usr/local/jdk-11.0.3/lib/src.zip",
        "javadoc": "https://docs.oracle.com/en/java/javase/11/docs/api",
        "default": true
    },
    {
        "name": "JavaSE-12",
        "path": "/usr/local/jdk-12.0.2"
    },
    {
        "name": "JavaSE-13",
        "path": "/usr/local/jdk-13"
    }
]
```



Selecting your Runtime

- To see which JDKs are used for your projects, you can trigger the command Java:
 - **Configure Java Runtime in Command Palette (⇧⌘P).**
 - This command opens a view displaying the runtime information for your projects:

A screenshot of the 'Configure Java Runtime' view in Visual Studio Code. The title bar says 'Configure Java Runtime'. The main area has a dark background with white text. It shows a table of projects and their runtime configurations. There are three rows: 1) 'spring-petclinic' (Maven), Java Version 1.8, edit icon. 2) 'junit5-jupiter-starter-gradle' (Gradle), Java Version 11, edit icon. 3) 'unmanaged-folder' (Unmanaged folder), Java Version 11, edit icon. Below the table, it says 'Manage Java runtime for your projects. If you don't have a valid Java runtime, you can [download](#) one.'



Run Dependency Plugin

- If you are using Maven, and dependencies are not resolved you can try the following

```
> mvn dependency:sources
```

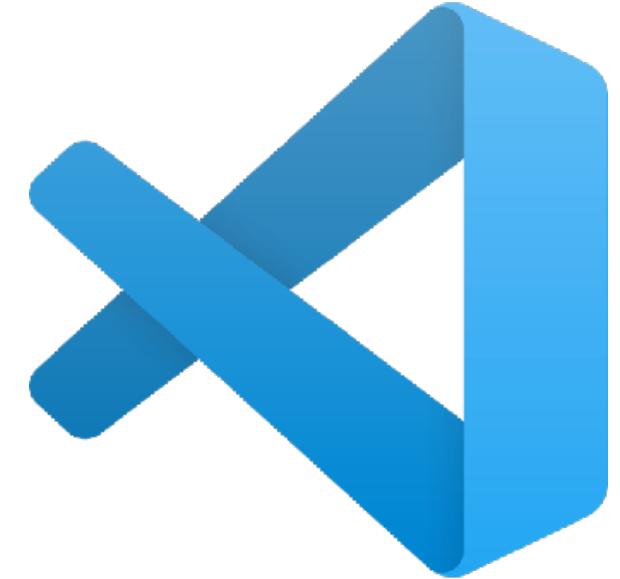
```
> mvn dependency:resolve -Dclassifier=javadoc
```

- Combining both the above commands into one

```
> mvn dependency:sources dependency:resolve -Dclassifier=javadoc
```



Opening Maven Projects



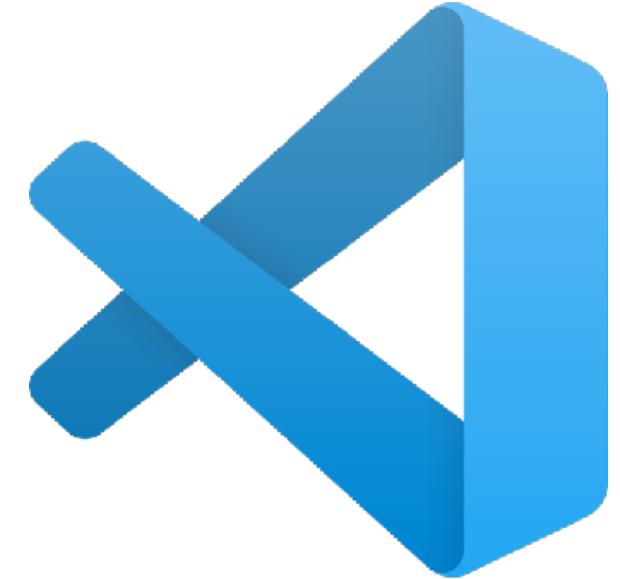
Maven in VSCode

- Maven is a software tool that helps you manage Java projects and automate application builds.
- The Maven for Java extension for Visual Studio Code provides fully integrated Maven support, allowing you to explore Maven projects, execute Maven commands, and perform the goals of build lifecycle and plugins.
- It is recommended to install the Extension Pack for Java, which includes Maven support and other important Java development features.



Resolving the Artifact

- The Maven extension also supports searching Maven Central to resolve unknown types in your source code.
- You can do this by selecting the **Resolve unknown type** link shown on hover over the code

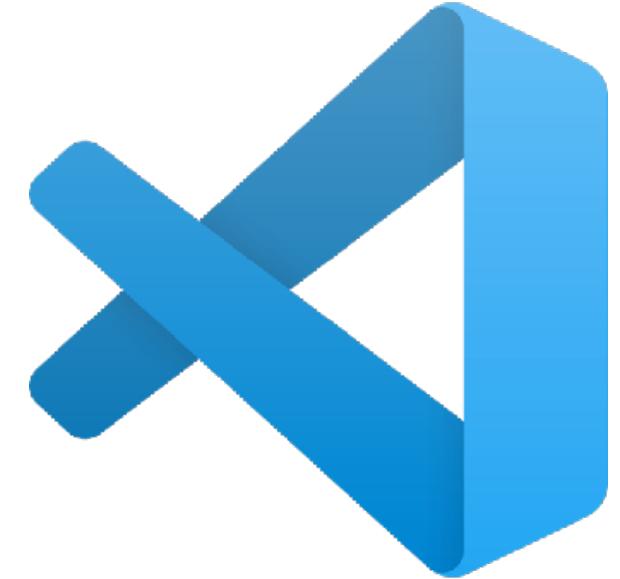


More Maven Features

- The Maven extension also supports searching Maven Central to resolve unknown types in your source code.
- You can do this by selecting the **Resolve unknown type** link shown on hover over the code
- You can also add a dependency using the **Maven: Add a Dependency** (or `maven.project.addDependency`) command
- Autocompletion is available where you can enter the dependencies that you exactly require.
- You can right click on the maven tool window and select “Show Effective POM” to view the Effective POM.

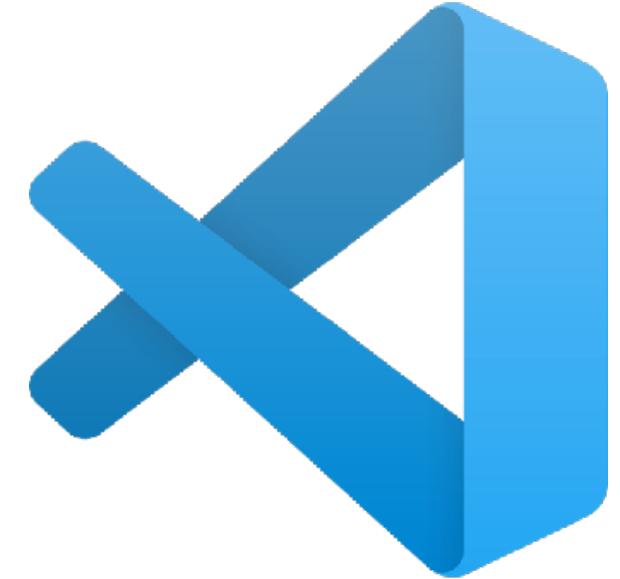


Opening Gradle Projects



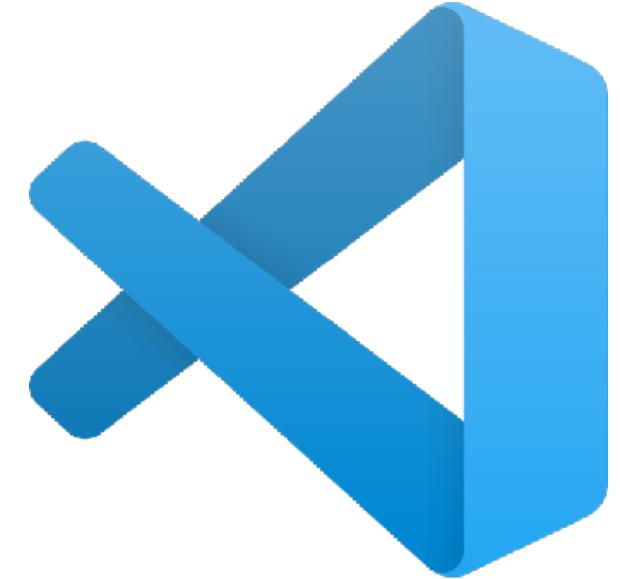
Gradle in VSCode

- VS Code supports Gradle Java project (not including Android) via the Gradle for Java extension.
- The extension provides a visual interface for your Gradle build, you can use this interface to view Gradle Tasks and Project dependencies, or run Gradle Tasks as VS Code Tasks.
- The extension also offers a better Gradle build file authoring experience including syntax highlighting, error reporting, and auto-completion.
- Be sure to run the Extension “Gradle for Java”



Working with Gradle Tasks

- When you open a Gradle project in VSCode, you can find some useful Gradle views by clicking the Gradle Side Bar item.
- **Gradle Projects** view lists all the Gradle projects found in the workspace.
- You can view, run, or debug Gradle tasks here.
- When there are many Gradle tasks in the workspace, it might be hard to find a specific task.
- The extension offers a Pinned Tasks view to help you pin your favorite tasks so that you can easily find them in a separate view.
- You can also see recently executed tasks in the Recent Tasks view.



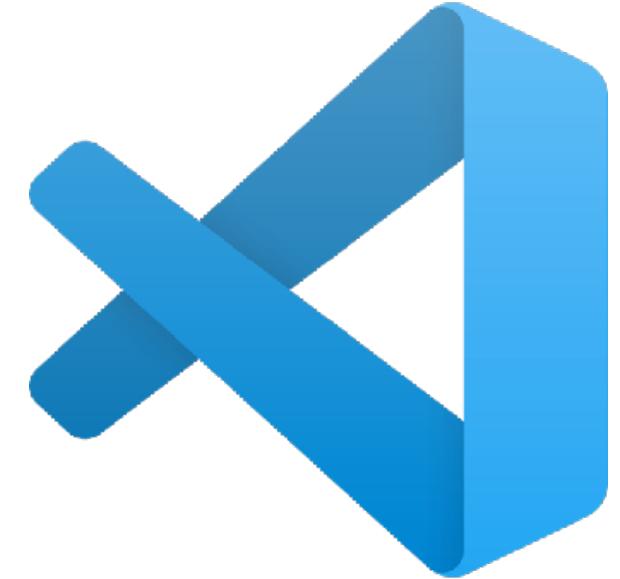
Managing Dependencies

- In the **Gradle Projects** view, you can find a **Dependencies** item under each Gradle project item.
- It includes all the dependencies in your specified configuration, you can easily check the dependency status of your project.



Managing Gradle Daemons

- The **Gradle Daemons** view shows the daemon status of the current workspace.
- It lists all the running Gradle daemons in the same version as the workspace.
- You can choose to stop a specific one or all the daemons in this view.

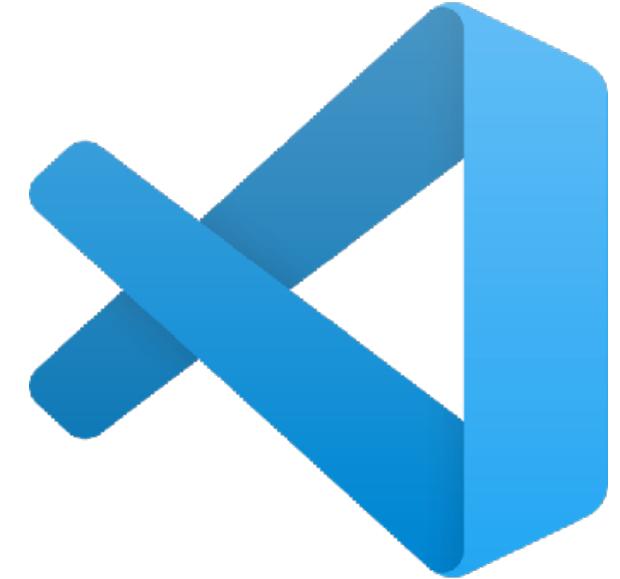


More Gradle Features

- Extensions provides syntax highlight and auto completions for Gradle build files
- Gradle build file analysis
- In the **Outline view**, the extension provides the document symbols of the opened Gradle file, which can help you to navigate to any part of the file easily.
- If there is any syntax error (missing characters, type not found, etc.) in the opened Gradle file, you can find them in the **Problems view**.
- You can declare new dependencies, and it offer dependency candidate



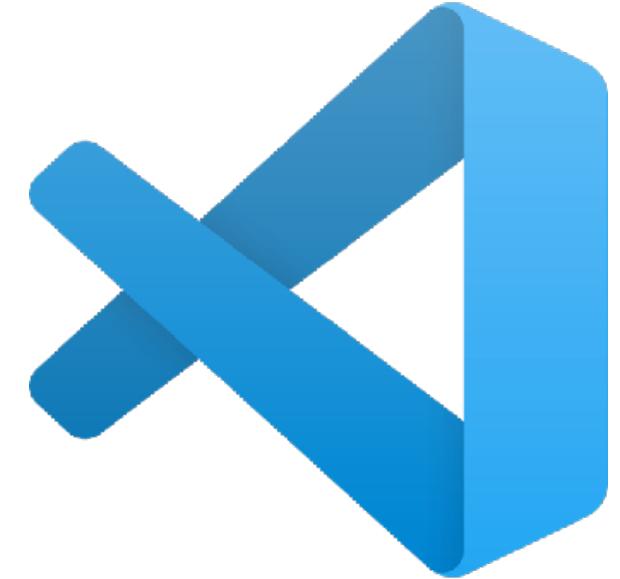
Opening Bazel Projects



Bazel in VSCode

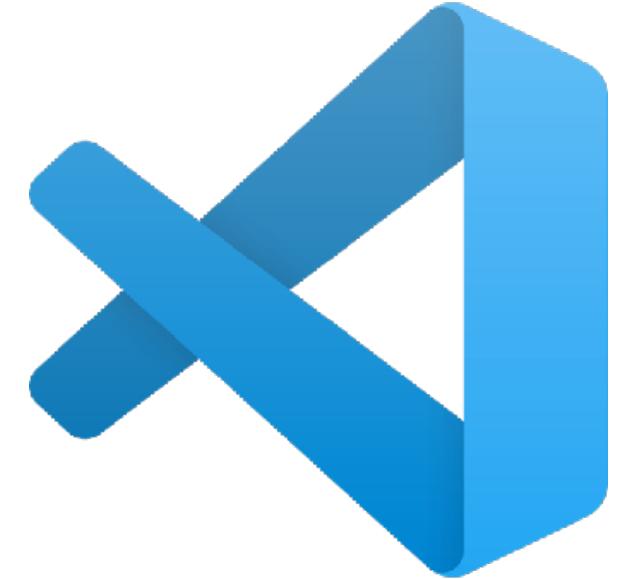
- Bazel is a Google based build tool for varying languages
- Bazel build files use Starlark, a Python based language used for configuration and runs on a JDK
- Incremental based and modular





Workspaces

- Before you can build a project, you need to set up its workspace.
- A workspace is a directory that holds your project's source files and Bazel's build outputs.
- It also contains files that Bazel recognizes as special
- The `WORKSPACE` file, which identifies the directory and its contents as a Bazel workspace and lives at the root of the project's directory structure
- To designate a directory as a Bazel workspace, create an empty file named `WORKSPACE` in that directory



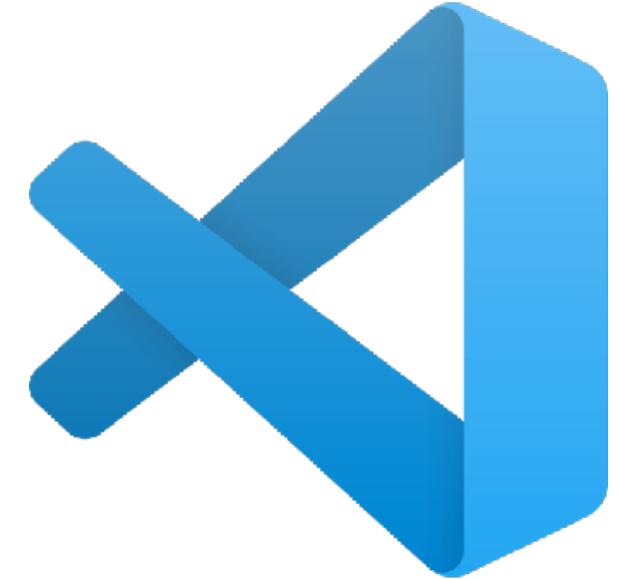
Build Files

- One or more BUILD files, which tells Bazel how to build different parts of the project.
- A directory within the workspace that contains a BUILD file is a package.
- This provides the modularity of your builds



Running Bazel Commands

- One or more BUILD files, which tells Bazel how to build different parts of the project.
- A directory within the workspace that contains a BUILD file is a package.
- This provides the modularity of your builds



Salesforce Bazel Extension

- Salesforce leads in development of a Bazel Extension
- Integration with VSCode and works much like the Maven and Gradle equivalents
- Uses the Redhat Java Extension and is a dependency
- A new version will be release in December 2023, but also check for new updates as they become available.
- Ensure that you use the pre-release version
- This is Salesforce invested project and you are welcome to become involved.



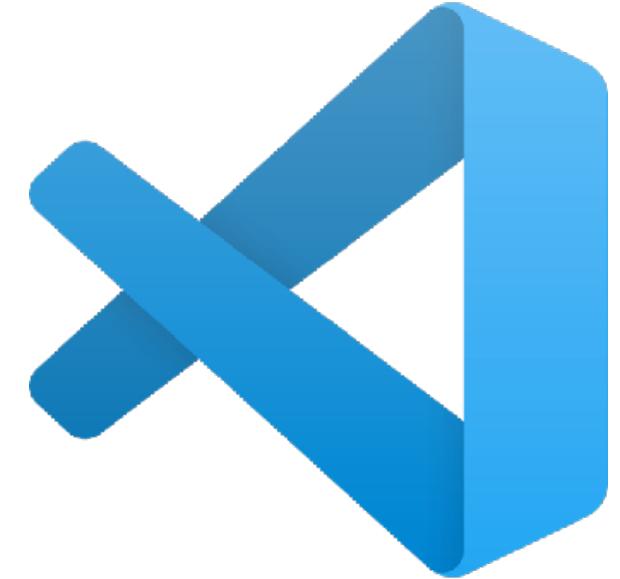
Using Bazel in VSCode



- In this lab, let's run the [vscode-training-bazel](#) project
- Review the contents
- Understand the relationships between BUILD and WORKSPACE
- Add modules through [.bazelproject](#)
- Run an application within VSCode in both the editor and the terminal

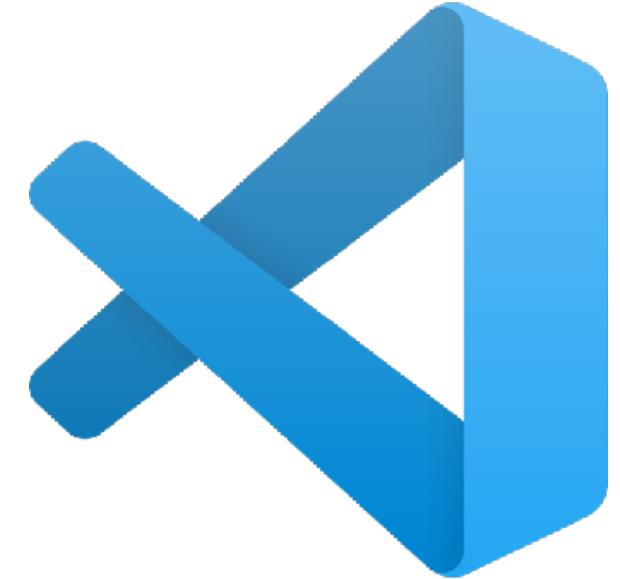


Formatting and Linting



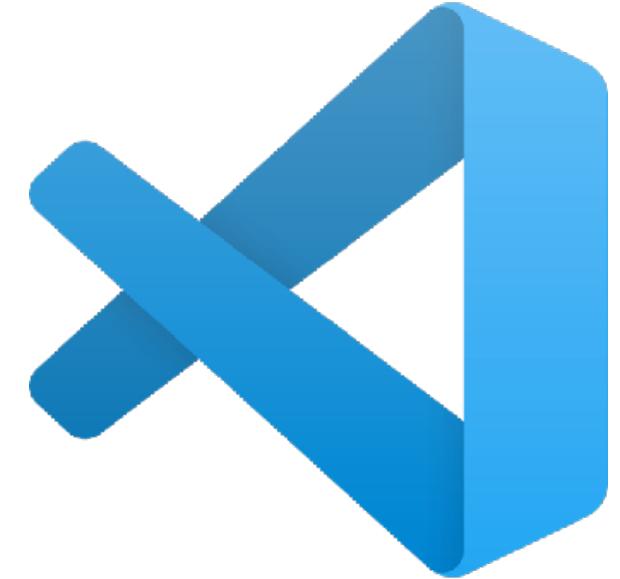
Java Formatting

- Language Support for Java™ by Red Hat also provides formatting settings.
- You can export an Eclipse formatter file and then use it for your project in VS Code.
- You can also use [Checkstyle for Java](#) and [SonarLint](#) extensions



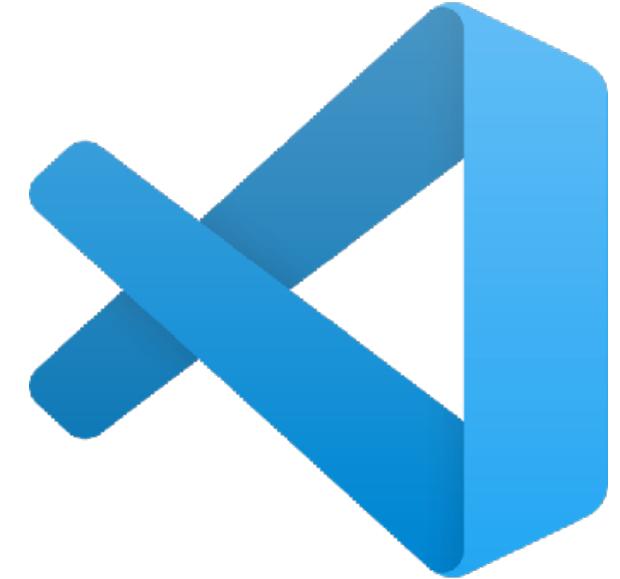
Java Formatting

- You can use Format Document command to format a Java file. If you didn't specify a formatter profile before, the Java file will be formatted using default settings.
- You can also apply other styles to your liking; like GoogleStyle
- You can open the editor with the command **Java: Open Java Formatter Settings with Preview**. In the editor, you can change the formatter settings and preview the effects. After saving the current editor, the changes will be saved to the formatter profile.



SolarLint

- SonarLint is an easy-to-use extension that helps you find and fix bugs and security issues as you code.
- The extension runs in the background and, just like a spell checker, highlights source code issues that pose a quality or security concern.
- The extension not only tells you what the issue is but also provides in-context guidance on why it's harmful and how to fix it, with examples.
- The extension supports over 500+ Java rules and includes several Quick Fixes to automatically fix certain quality issues.

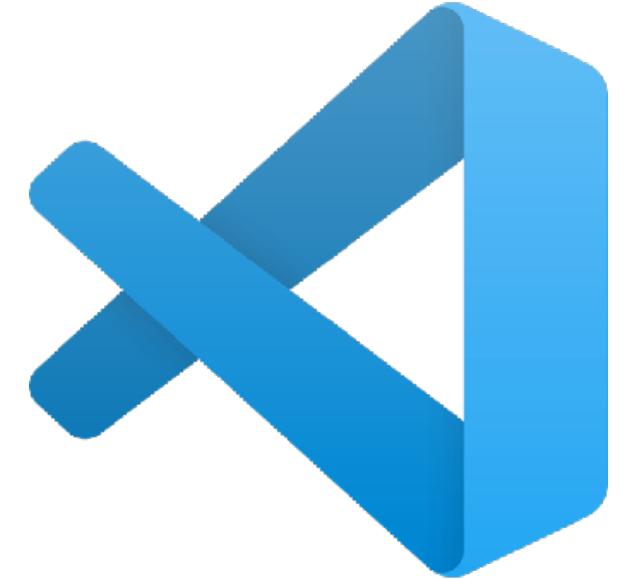


Checkstyle

- With the Checkstyle for Java extension, you can use either existing checkstyle configurations (Google's or Sun's Check) or your own customized files for your project.
- When editing a Java file, the extension will check the file format and provide Quick Fixes if possible on the fly.
- Set Checkstyle configuration file using the **Checkstyle: Set the Checkstyle Configuration File** command and selecting the **Checkstyle file** from the dropdown.

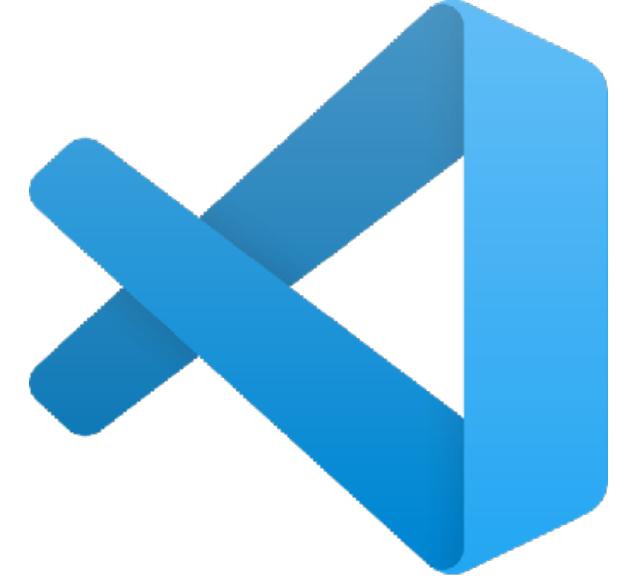


Spring and Spring Boot



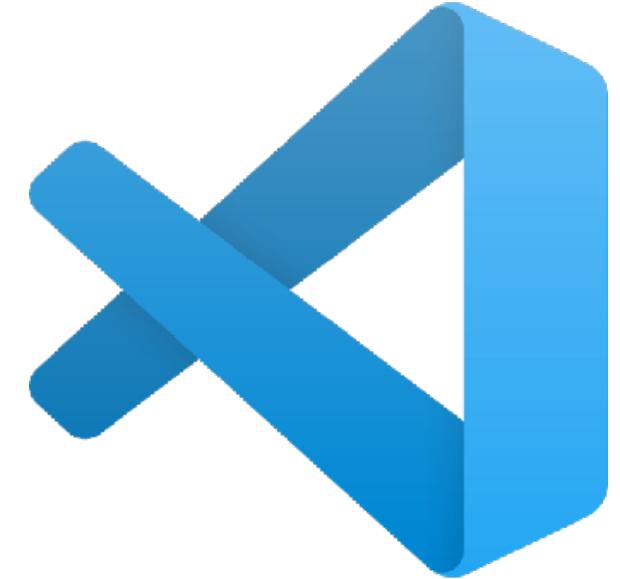
Spring Boot in VSCode

- Visual Studio Code is an ideal lightweight development environment for Spring Boot application developers and there are several useful VS Code extensions including:
 - Spring Boot Tools
 - Spring Initializr
 - Spring Boot Dashboard
- Recommended to use the [Spring Boot Extension Pack](#) that includes all of the extensions above.



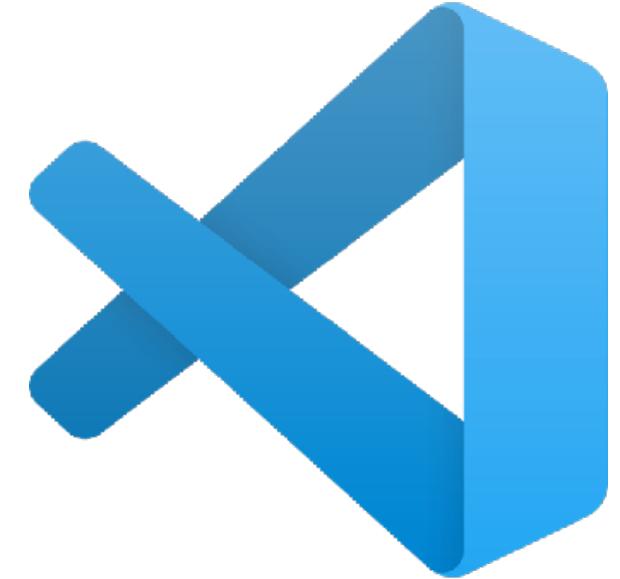
Creating a Spring Boot Project

- To install, launch VS Code and from the Extensions view ([\$\text{↑⌘X}\$](#)), search for `vscode-spring-initializr`.
- Once you have the extension installed, open the Command Palette ([\$\text{↑⌘P}\$](#)) and type `Spring Initializr` to start generating a Maven or Gradle project and then follow the wizard.



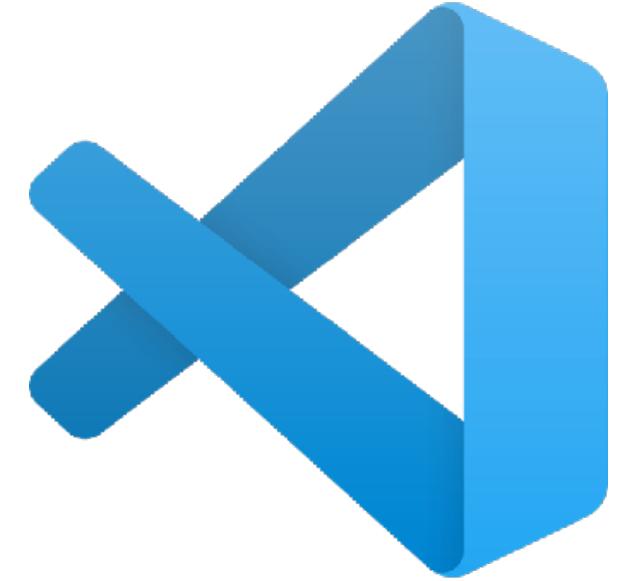
Editing a Spring Boot Project

- The Spring Initializr extension allows you to add dependencies after generating a new Spring Boot project.
- Navigate to your pom.xml file and right-click to select **Add starters...**
- A dropdown will show the dependencies you already have beginning with a ✓.
- You can search for other dependencies you want to add to your project. Or you can click on the existing dependencies to remove them.



Developing a Spring Boot Project

- The Spring Boot Tools extension includes rich language support for working with Spring Boot [application.properties](#), [application.yml](#), and [.java](#) files.
- The extension supports the following features:
 - Quickly navigate to a Spring element in your workspace
 - Smart code completion for Spring specific components
 - Quick access to running Spring apps
 - Live application information
 - Code templates
- Similar code completion and validation features are also available for `.properties` and `.yml` files.



Running a Spring Boot Project

- In addition to using **F5** to run your application, there's the Spring Boot Dashboard extension
- Spring Boot Dashboard lets you view and manage all available Spring Boot projects in your workspace as well as quickly start, stop, or debug your project.



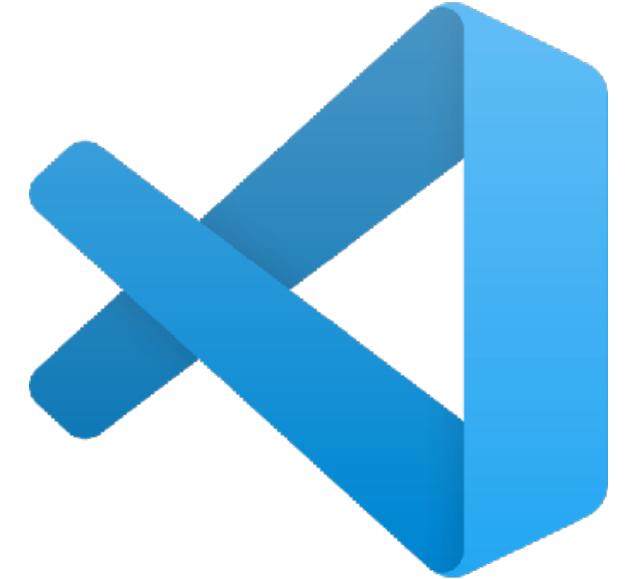
Opening PetClinic in VSCode



- In this lab, let's open the Petclinic Application in VSCode
- Review the contents
- Make some changes to the code

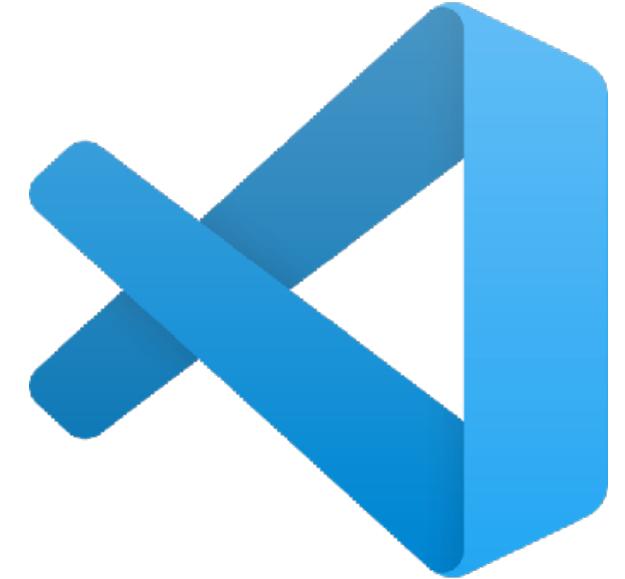


Python



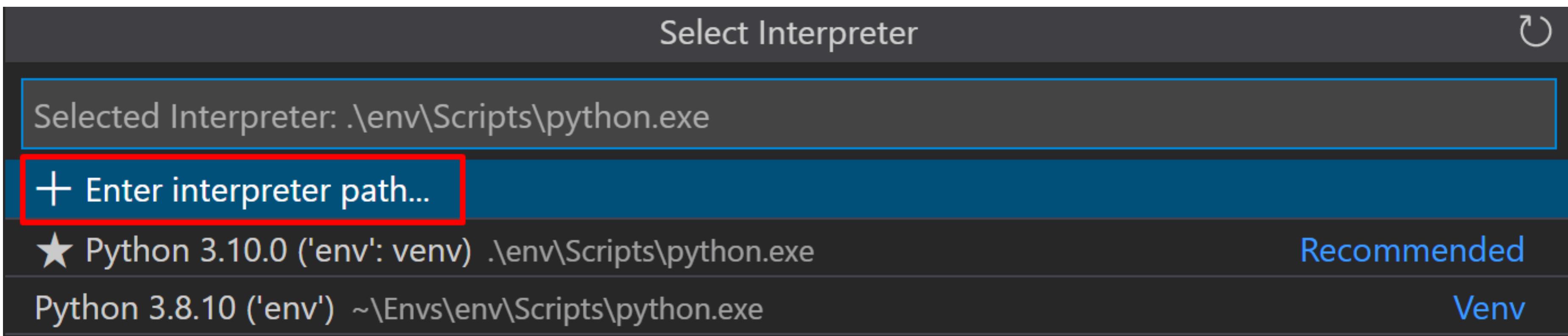
Python and VSCode

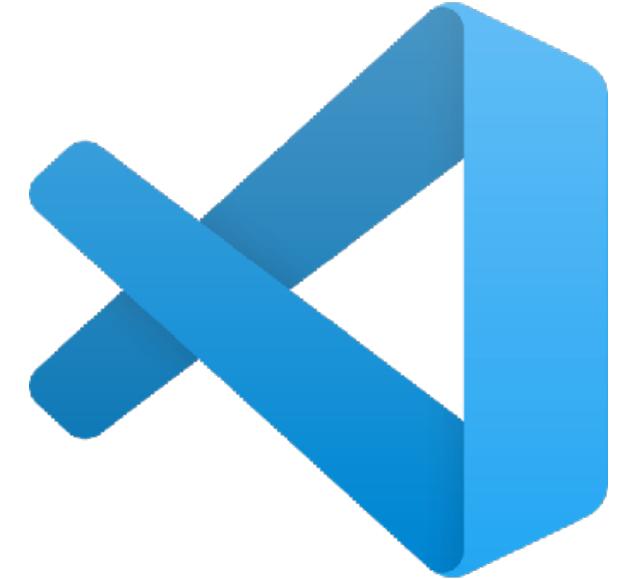
- Python uses the Microsoft Python extension
- It leverages all of VS Code's power to provide auto complete and IntelliSense, linting, debugging, and unit testing, along with the ability to easily switch between Python environments, including virtual and conda environments.



Using Python in VSCode

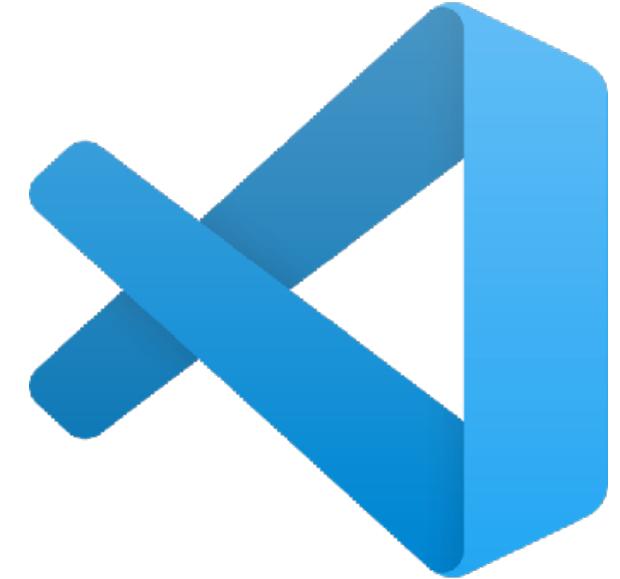
- Once you have a version of Python installed, select it using the **Python: Select Interpreter** command.
- If VS Code doesn't automatically locate the interpreter you're looking for, open the **Command Palette**, and select **Python: Select Interpreter** command and select the **Enter interpreter path...** option that shows on the top of the interpreters list:





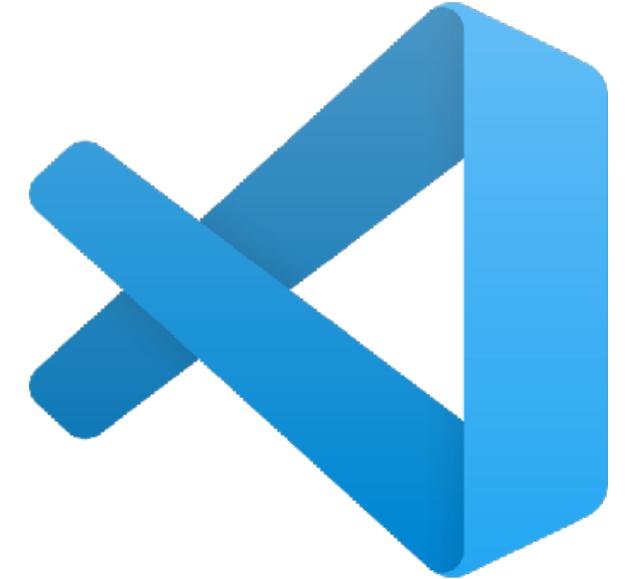
Python Environment Manager Extension

- Plugin that manages multiple Python Environments
 - Quickly change workspace Python Environments
 - Viewing Environments specific to a Workspace
 - Viewing all of your Python environments grouped by their type (Conda, PyEnv, etc)
 - Creating a terminal with the environment activated.
 - Create/delete Conda and Virtual Environments.
 - Install specific versions of Python from PyEnv
 - View, install, update and uninstall packages in your environments, and more
- View more at the [extension website](#)



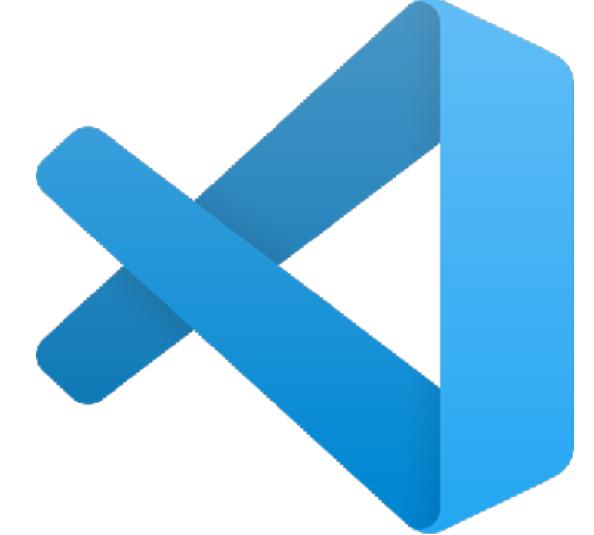
Interactive Python

- [Jupyter](#) (formerly IPython Notebook) is an open-source project that lets you easily combine Markdown text and executable Python source code on one canvas called a **notebook**.
- Visual Studio Code supports working with Jupyter Notebooks natively, as well as through Python code files.
- To work with Jupyter notebooks, you must activate an Anaconda environment in VS Code, or another Python environment in which you've installed the Jupyter package.
- To select an environment, use the **Python: Select Interpreter** command from the **Command Palette** ().



Recommended Plugins

- If you have already installed VS Code and want to add Python support use the [the Python Extension](#)
 - [autoDocString](#)
 - [Error Lens](#)
 - [Python Test Explorer for Visual Studio Code](#)



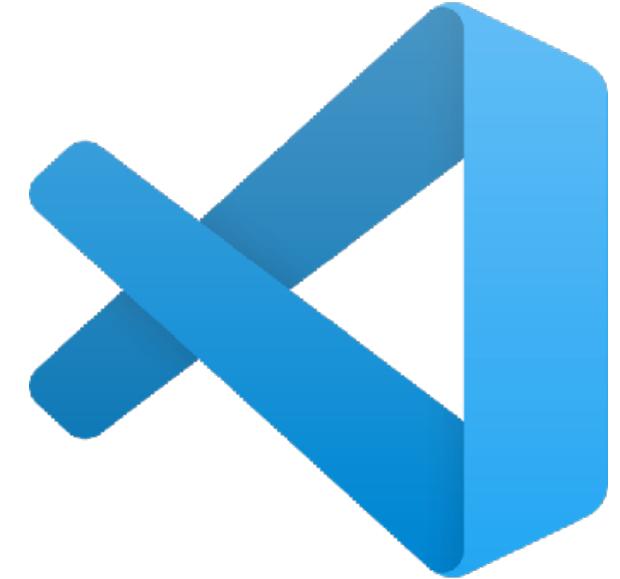
Using Python in VSCode



- In this lab, let's run the [`vscode-training-python`](#) project
- Review the contents

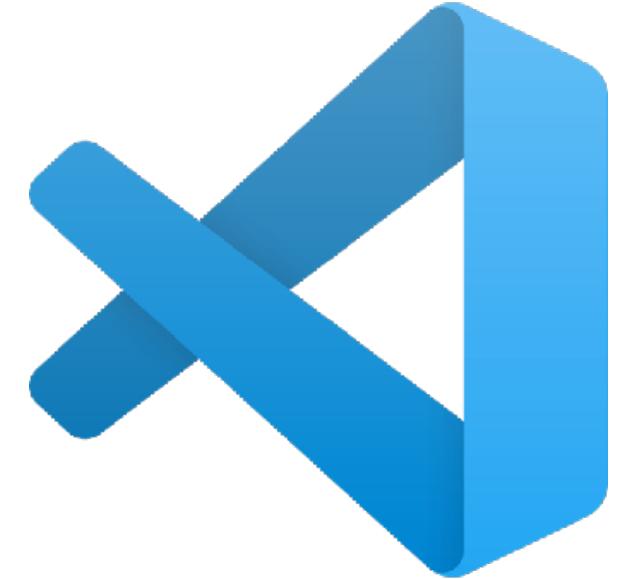


JavaScript Editing



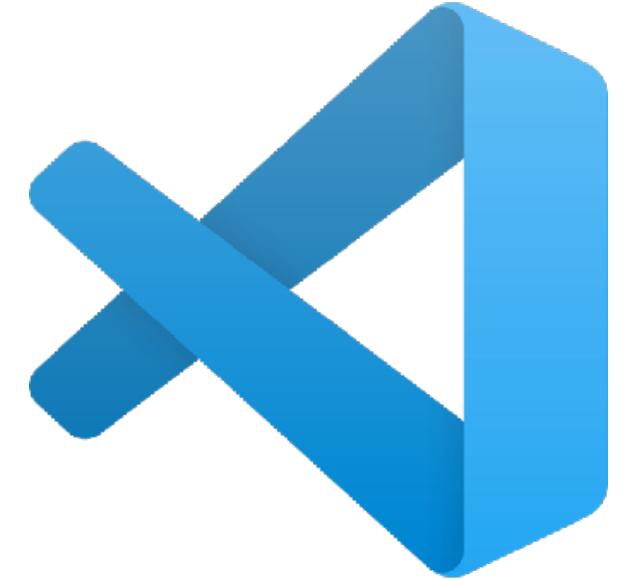
JavaScript in VSCode

- JavaScript Support includes IntelliSense and other common features in VSCode
- Node.js Support included
- While IntelliSense should just work for most JavaScript projects without any configuration, you can make IntelliSense even more useful with JSDoc or by configuring a [jsconfig.json](#) project.



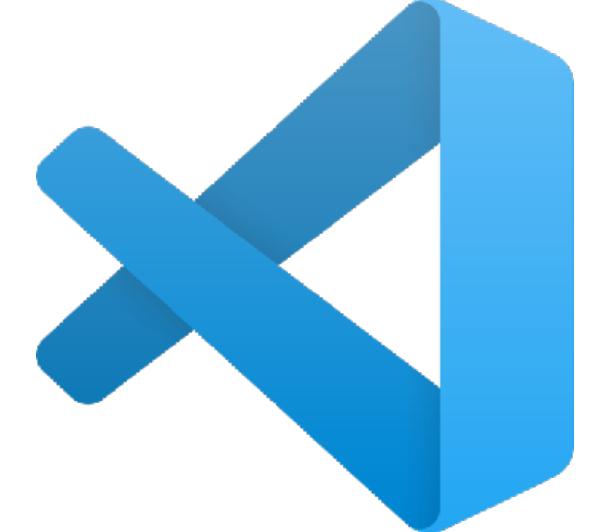
JSConfig.json

- A `jsconfig.json` file defines a JavaScript project in VS Code.
- `jsconfig.json` files are not required, you will want to create one for the following reasons:
 - Removing files from the Intellisense's reach
 - Ensure that a subset of JavaScript file is treated as one problem
 - Multi-root project resolutions
 - Using TypeScript compiler to down-level compile JavaScript source code.
- Find more details here [at the website](#)



Recommended Plugins

- Visual Studio Code includes built-in JavaScript IntelliSense, debugging, formatting, code navigation, refactorings, and many other advanced language features. The exception are for linters
- Some other fine choices include
 - [JavaScript Code Snippets](#)
 - [Babel JavaScript](#)



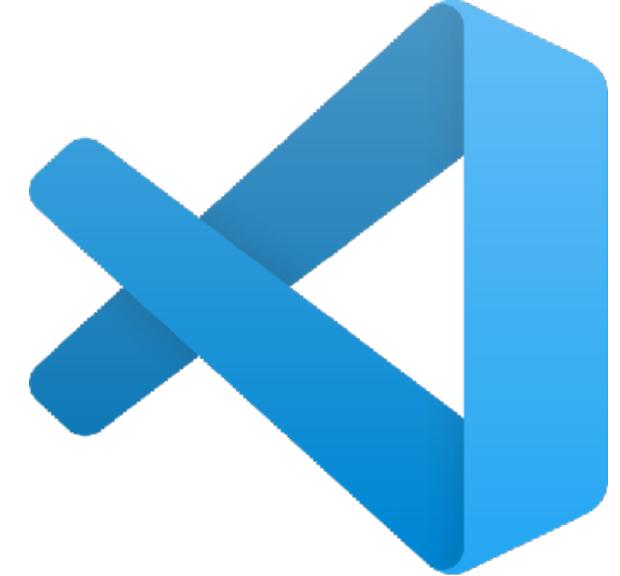
Using JavaScript in VSCode



- In this lab, let's run the [vscode-training-node](#) project
- Review the contents



Programming Maneuvers

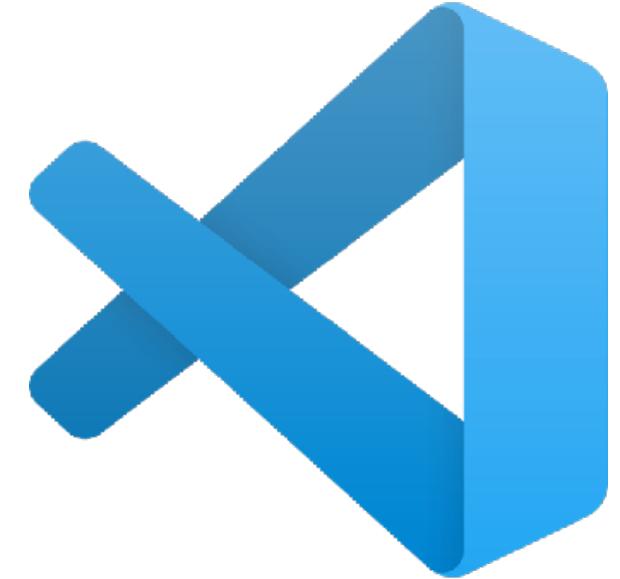


Standard Maneuvers

- These are some standard maneuvers within VSCode
- Some may or may not be available depending on the extensions provided

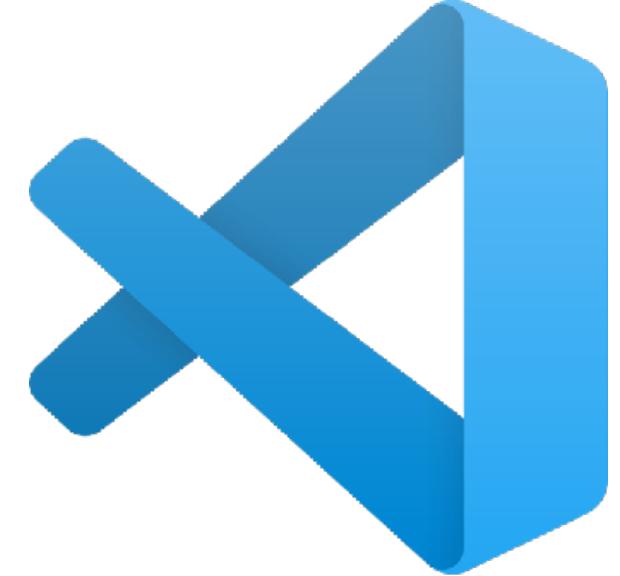


Navigation



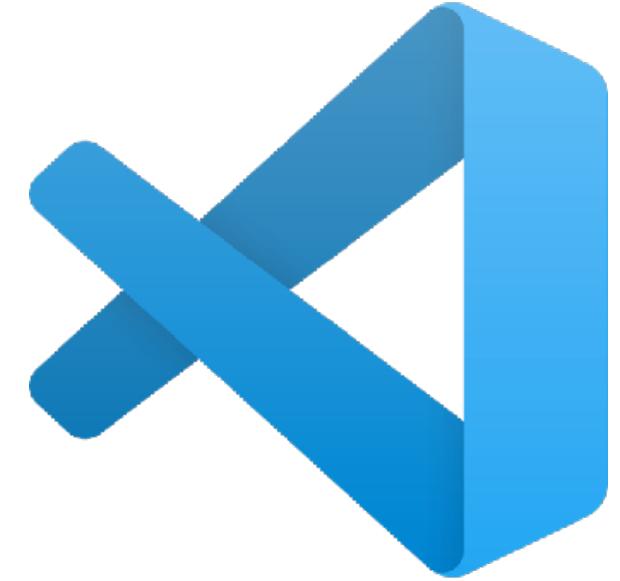
Code Navigation

- With the **Outline** view, you can conveniently navigate the members within the current file.
- Projects view also provide a great overview of your project.
- For Java, it also supports Call Hierarchy, Type Hierarchy, Definition Navigation, Search Types in Workspace, etc.



Search for Workspace Symbols

- To search for a symbol in the current workspace, start by pressing `⌘T`, then enter the name of the symbol.
- A list of potential matches will appear as before. If you choose a match that was found in a file that's not already open, the file will be opened before navigating to the match's location.
- Alternatively, you can also use Quick Open (`⌘P`) then enter the '#' command to search the current workspace. `⌘T` is just the shortcut for the '#' commands, so everything works the same.



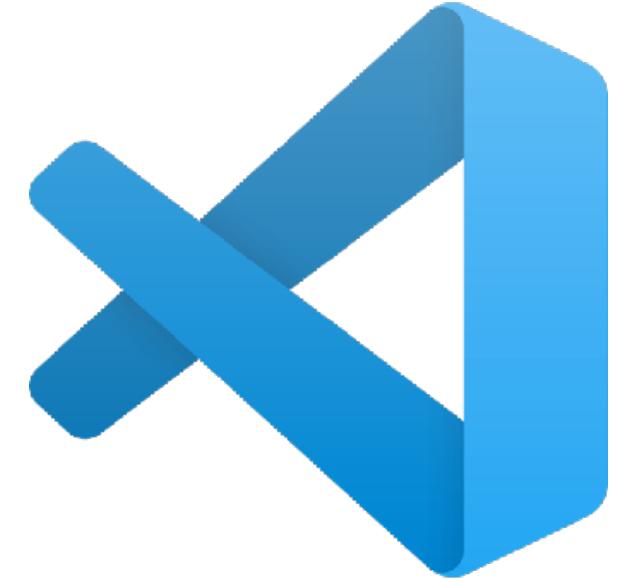
Search for Current File Symbols

- To search for a symbol in the current file, use Quick Open (`⌘P`) then enter the '`@`' command
- Enter the name of the symbol you're looking for.
- A list of potential matches will appear and be filtered as you type. Choose from the list of matches to navigate to its location.
- You can also go to a symbol in a file by performing (`⇧⌘O`)



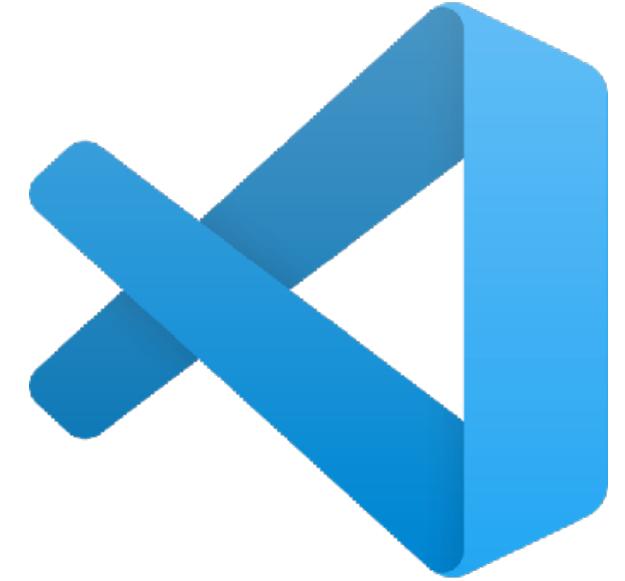
Definition Commands

- Peek Definition (`\F12`)
- Goto Definition (`F12`)
- Super Implementation (Click and Hover on Class)
- Goto References (`\F12`)
- Find All References (`\F12`)



Hierarchy Commands

- A Call Hierarchy view shows all calls from or to a function and allows you to drill into callers of callers and call of calls.
- Right-click on a function and select **Peek > Peek Call Hierarchy**.
- You can also right-click in a function body and pick **Show Call Hierarchy**.
- A Type Hierarchy view shows the inheritance relationships between Java Objects. You can right-click on a type and pick **Show Type Hierarchy**.

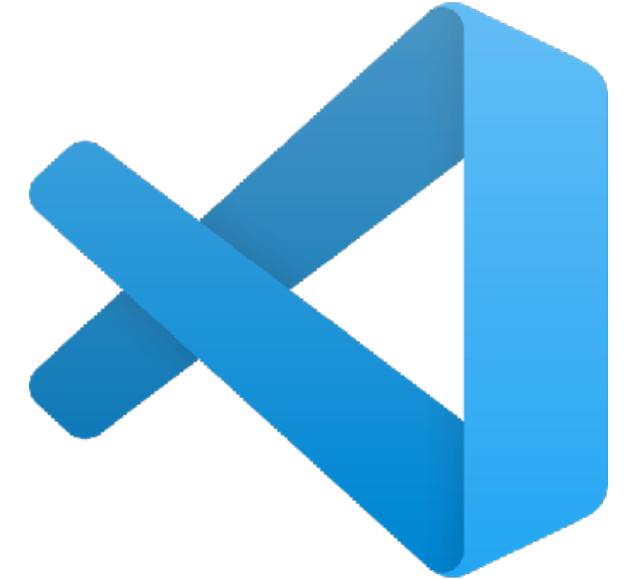


Other Navigation Commands

- Select current line (`⌘L`)
- Navigate to beginning and end of file (`⌘↑`) and (`⌘↓`)



IntelliSense

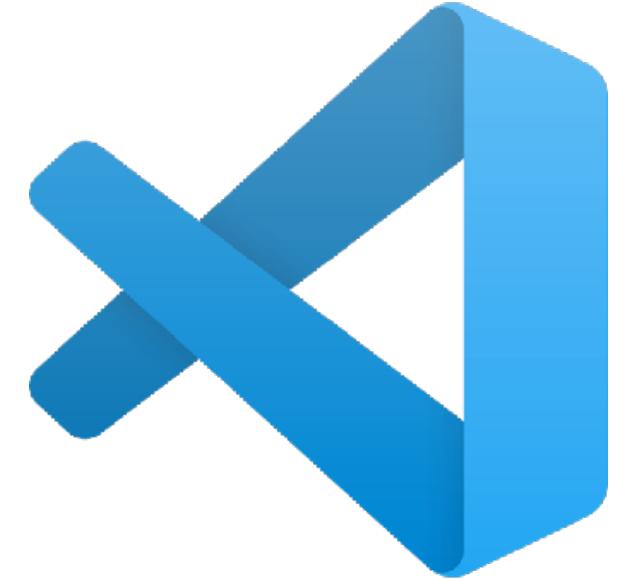


IntelliSense

- IntelliSense is a suggestions widgets meant to help you code
- `^Space` to trigger the Suggestions widget.



Errors and Warning



Errors and Warnings

- Show Problems ([`⌃M`](#))
- Cycle through Problems ([`⌃F8`](#)) ([`F8`](#))



Increase + Decrease Selections



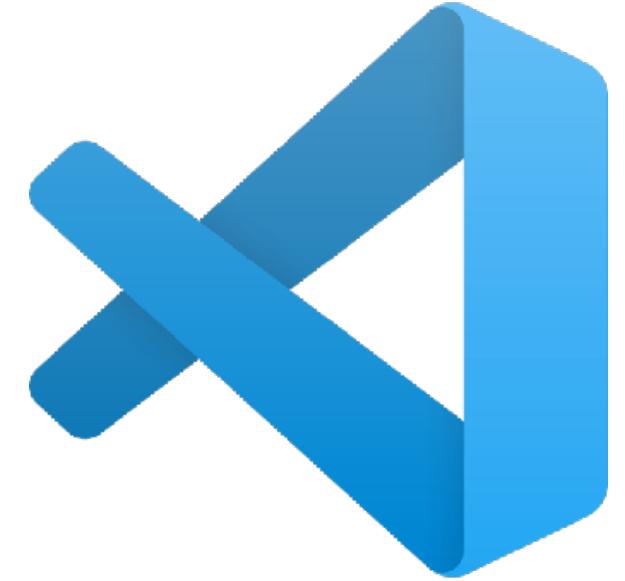
TONY LYN presents

THE BLOB

starring

STEVE McQUEEN,
ANETA CORSAUT, STEPHEN CHASE,
EARL ROWE, OLIN HOWLIN,
JOHN BENSON & GEORGE KARAS

12

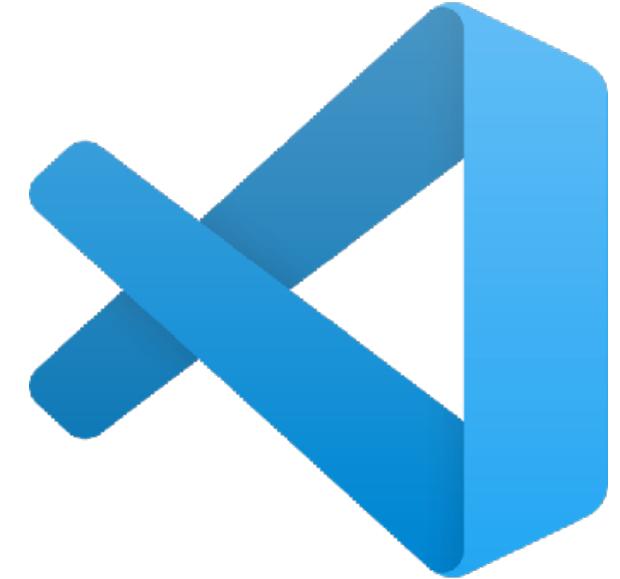


Increase and Decrease Selections

- Increase Selection (^↑⌘←)
- Decrease Selection (^↑⌘→)



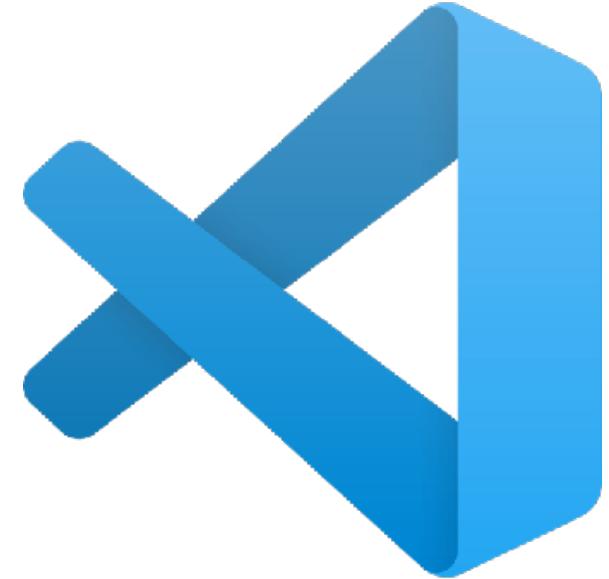
Semantic Highlighting



Semantic Highlighting

- Semantic highlighting enriches the syntax coloring based on symbol information from a language service that has the full understanding of the project.
- Based on this understanding each identifier gets colored & styled with the color of the symbol it resolves to.
- Built-in themes show semantic highlighting. Other themes need to opt-in to semantic highlighting by a new property in the theme file

```
"semanticHighlighting": true
```



Overriding Semantic Highlighting

Users can override that setting in the user settings

```
"editor.semanticTokenColorCustomizations": {  
    "[Atom One Dark)": {  
        "enabled": true  
    }  
}
```



Code Formatting

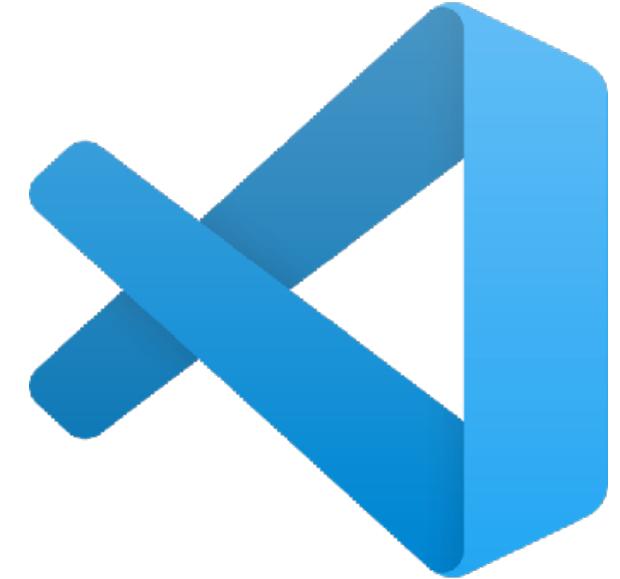


Code Formatting

- Currently selected source code only (**⌘K ⌘F**)
- Whole document format: **⇪ ⌘F**



Code Folding

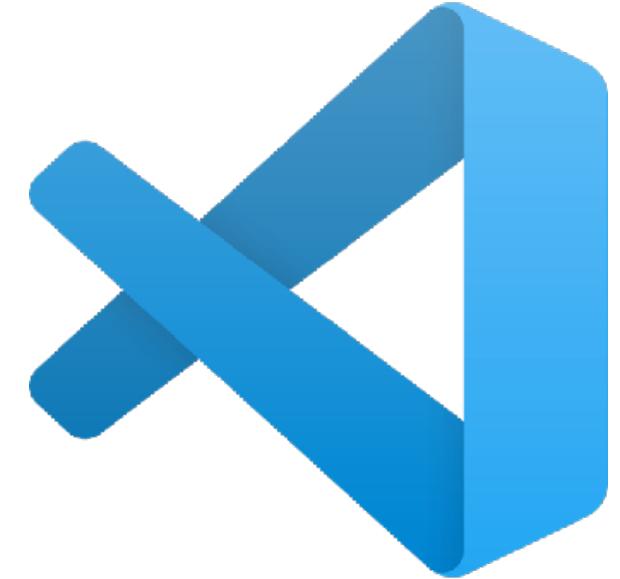


Code Folding

- Keyboard Shortcut: (`⌘⌘[`) and (`⌘⌘]`)
- You can also fold/unfold all regions in the editor with Fold All (`⌘K ⌘O`) and Unfold All (`⌘K ⌘J`).
- You can fold all block comments with Fold All Block Comments (`⌘K ⌘/`).

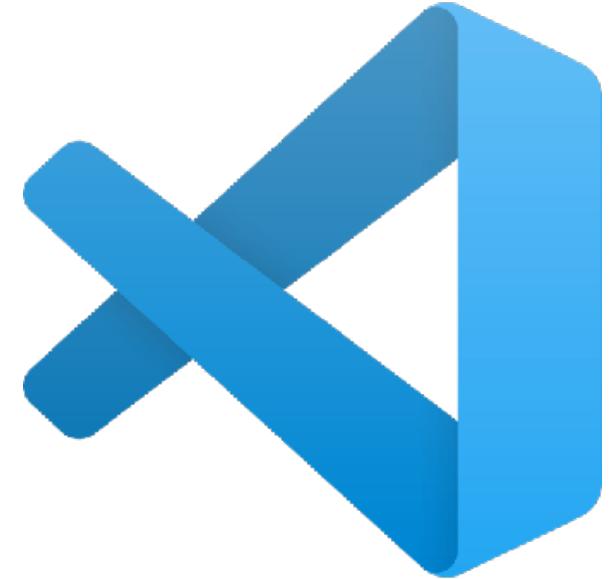


Refactoring



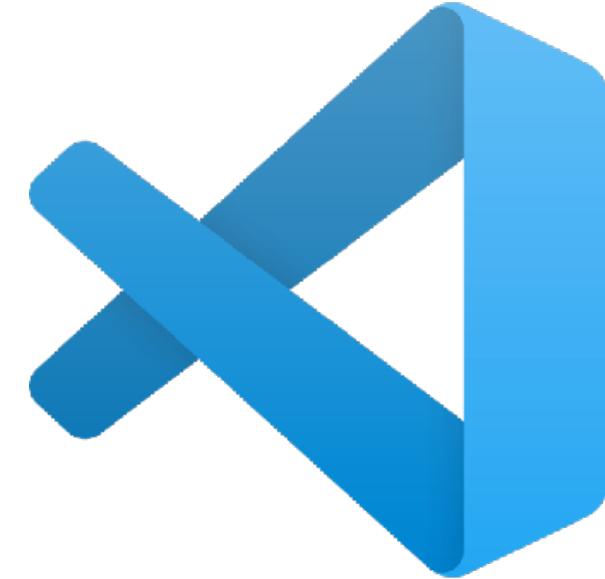
Refactoring in VSCode

- Visual Studio Code provides many options to refactor your source code as well as Source Actions to generate code and fix issues while you're coding.
- To access them, click on the light bulb whenever you see it.
- Or right-click the editor view and pick **Source Action...**



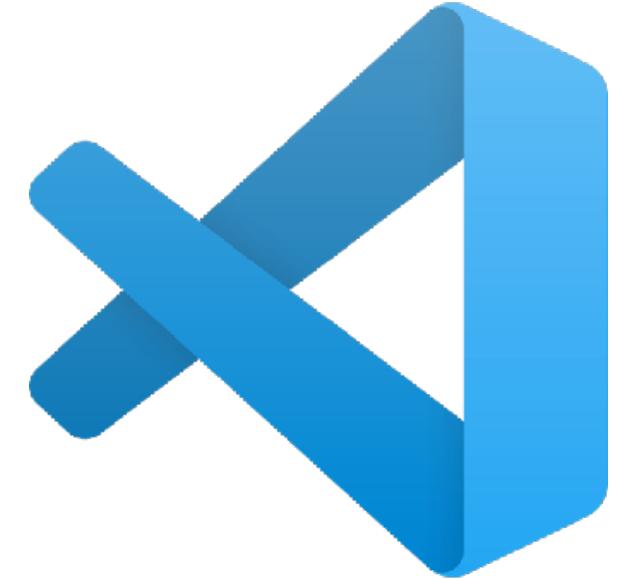
Supported Code Actions

- Assign to variable
- Convert anonymous to nested class
- Convert to anonymous class creation
- Convert to enhanced for loop
- Convert to lambda expression
- Convert to static import
- Extract to constant
- Extract to field
- Extract to method
- Extract to local variable
- Inline constant
- Inline local variable
- Inline method



Supported Code Actions

- Invert conditions
- Invert local variable
- Move
- Rename ([F2](#))
- Change resolved type to var type
- Change var type to resolved type
- Generate constructors
- Generate delegate methods
- Override/implement methods
- Organize imports
- Generate getters and setters
- Generate hashCode() and equals()
- Generate toString()
- Change modifiers to final where possible



Supported Code Actions

- Fix non-accessible reference
- Create non-existing package
- Create unresolved types
- Remove the final modifier
- Remove unnecessary cast
- Remove redundant interfaces
- Add missing case labels in switch statements
- Jump to definition on break/continue
- Correct access to static elements



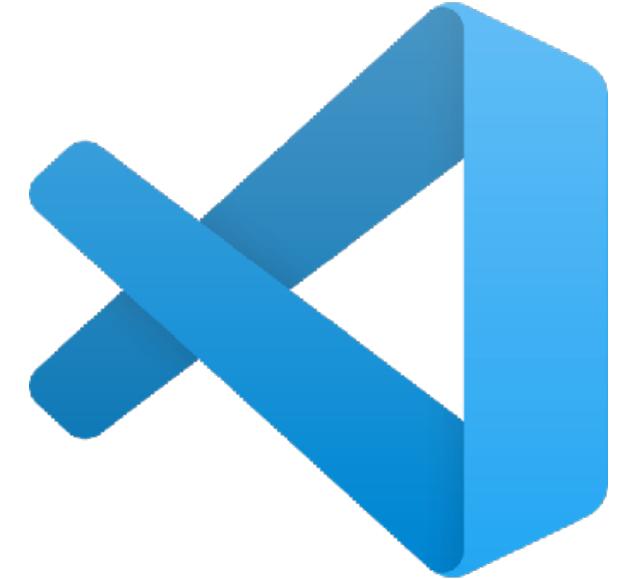
Refactoring



- In this lab, let's try refactoring
- Open Ted Young's blackjack game and let's try some refactoring. You can also work on your own repository
- Create methods, move methods, etc.



Find and Replace

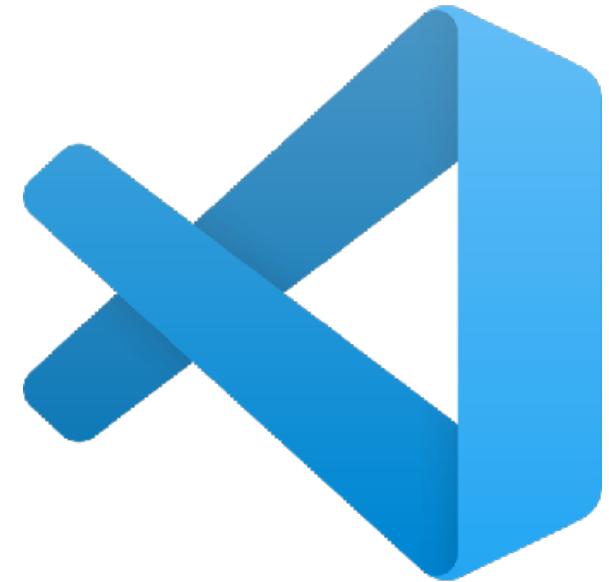


Find and Replace

- Besides searching and replacing expressions, you can also search and reuse parts of what was matched, using regular expressions with capturing groups.
- Enable regular expressions in the search box by clicking the **Use Regular Expression (*)** button (`⌘R`)
- You can then write a regular expression and use parenthesis to define groups.
- You can then reuse the content matched in each group by using `$1`, `$2`, etc. in the Replace field.



Debugging



Debugging

Start debugging

Pause, step over, step in/out, restart, stop

The screenshot shows the Visual Studio Code interface with the following highlights:

- Launch Program button:** Located in the top left of the toolbar, highlighted with a red box.
- Debug toolbar icons:** A row of icons for pausing, resuming, stepping over, stepping in, stepping out, restarting, and stopping the debugger, also highlighted with a red box.
- Code editor:** Shows a file named "app.js" with code for an Express application. Line 10 is highlighted with a red box.
- DEBUG CONSOLE:** A panel at the bottom showing the command "C:\Program Files\nodejs\node.exe .\bin\www".
- Debug side bar:** A sidebar on the left containing sections for RUN, VARIABLES, WATCH, CALL STACK, LOADED SCRIPTS, and BREAKPOINTS. The BREAKPOINTS section has three items: Caught Exceptions, Uncaught Exceptions, and app.js, with app.js checked and highlighted with a red box.
- Debug console panel:** A panel on the right labeled "Debug console panel".

File Edit Selection View Go Run Terminal Help

app.js - myExpressApp - Visual Studio Code

```
1 var createError = require('http-errors');
2 var express = require('express');
3 var path = require('path');
4 var cookieParser = require('cookie-parser');
5 var logger = require('morgan');
6
7 var indexRouter = require('./routes/index');
8 var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'nunjucks');
```

DEBUG CONSOLE

C:\Program Files\nodejs\node.exe .\bin\www

master □ 0 △ 0 Launch Program (myExpressApp)

Ln 10, Col 11 Spaces: 2 UTF-8 LF JavaScript ⌂ ⌂

Debug side bar

Debug console panel



Run and Debug View

- To bring up the **Run and Debug** view, select the **Run and Debug** icon in the Activity Bar on the side of VS Code. You can also use the keyboard shortcut [`⇧⌘D`](#).
- If running and debugging is not yet configured (no `launch.json` has been created), VS Code shows the Run start view.
- The top-level Run menu has the most common run and debug commands



Launch Configurations

- To run or debug a simple app in VS Code, select **Run and Debug** on the Debug start view or press **F5** and VS Code will try to run your currently active file.
- However, for most debugging scenarios, creating a launch configuration file is beneficial because it allows you to configure and save debugging setup details.
- VS Code keeps debugging configuration information in a [launch.json](#) file located in a [.vscode](#) folder in your workspace (project root folder) or in your user settings or workspace settings.
- To create a [launch.json](#) file, click the create a [launch.json](#) file link in the **Run** start view.

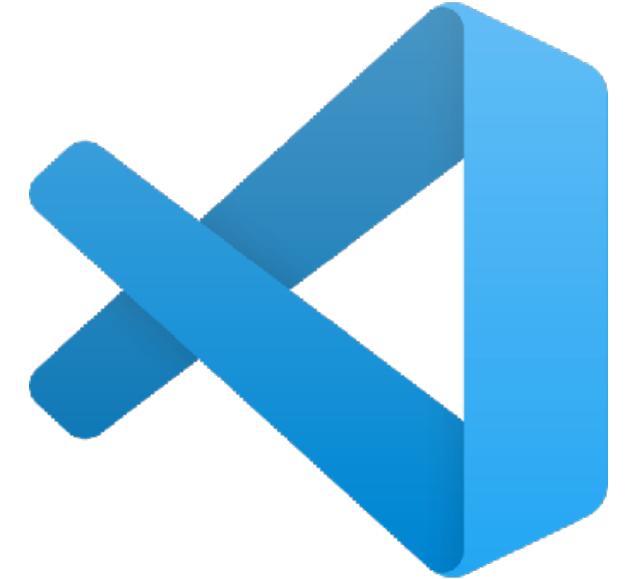


Debugging Keymap

- **F9** Toggle breakpoint
- **F5** Start/Continue
- **F11 / ⌘F11** Step into/ out
- **F10** Step over
- **⌃F5** Stop
- **⌘K ⌘I** Show hover



Testing



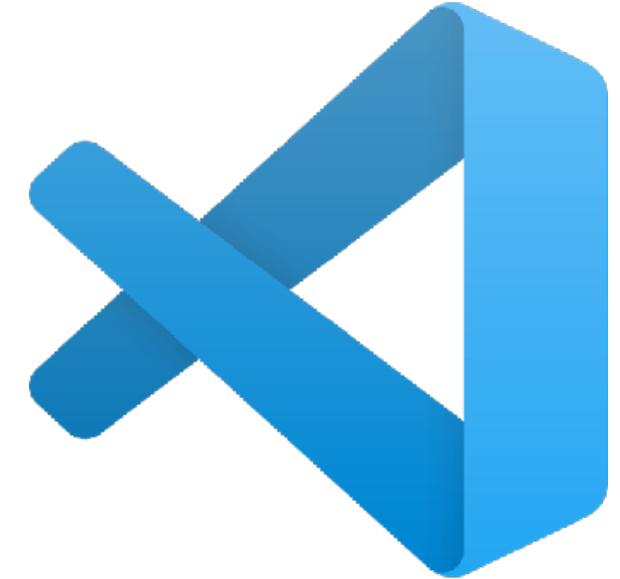
Testing

- Install the Test Runner for Java extension
- This will generate shortcuts (the green play button) on the left side of the class and method definition.
- To run the target test cases, select the green play button. You can also right-click on it to see more options.



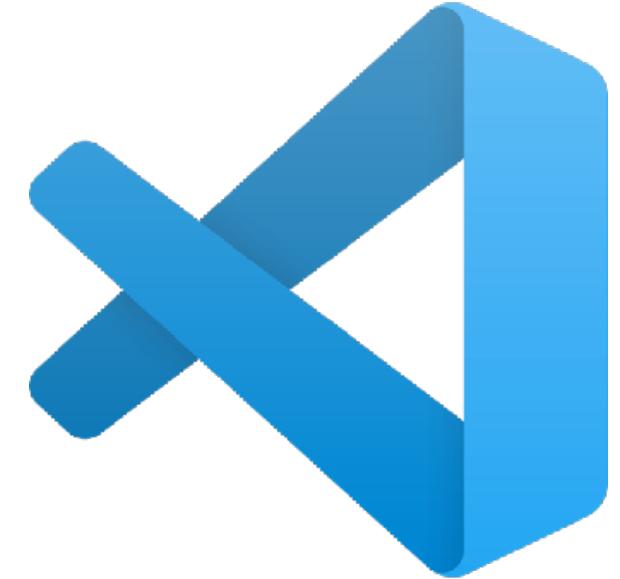
Testing Explorer

- The Testing Explorer is a tree view to show all the test cases in your workspace.
- You can select the beaker button on the left-side Activity bar of Visual Studio Code to open it.
- You can also run/debug your test cases and view their test results from there.



View Test Results

- After running/debugging the test cases, the state of the related test items will be updated in both editor decorations and the Testing Explorer.
- You can trigger the command Test: Peek Output to peek the results view. You can select the links in the stack trace to navigate to the source location.



Generating Tests

- The extension provides features to help you scaffold test cases. You can find the entry in the editor context menu. Select **Source Action...** and then choose **Generate Tests....**
- If you trigger this source action from your main source code (test subject), you will be asked the test class's fully qualified name and the methods you want to test.
- The extension will then generate the test code for you
- If you trigger the source action from your test source code, you will be asked which kinds of test methods you want to add. Including the lifecycle methods and the test method



Test Navigation

- The extension provides features to help you navigate between your tests and test subjects.
- If your source code is contained in `src/main/java` or `src/test/java`, you can find the entry named **Go to Test** or **Go to Test Subject** in the editor context menu
- You can also find the command in the Command Palette (↑⌘P) by searching for **Java: Go to Test**.
- There are other testing commands (for example, **Run Tests in Current File**) that can be found by searching for '**Test:**' in the Command Palette (↑⌘P).



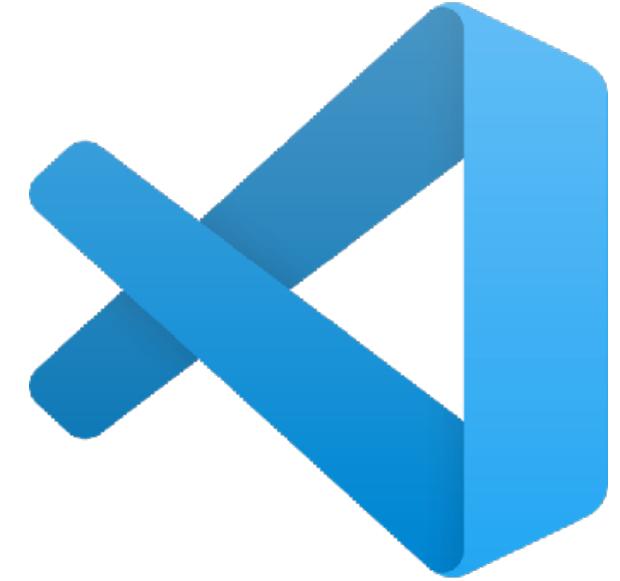
Running Tests



- In this lab we will try out Test Driven Development by adding a test for our Wallet.
- We will create a balance method that will return the balance of the wallet
- View the reports if successful



Multicursors

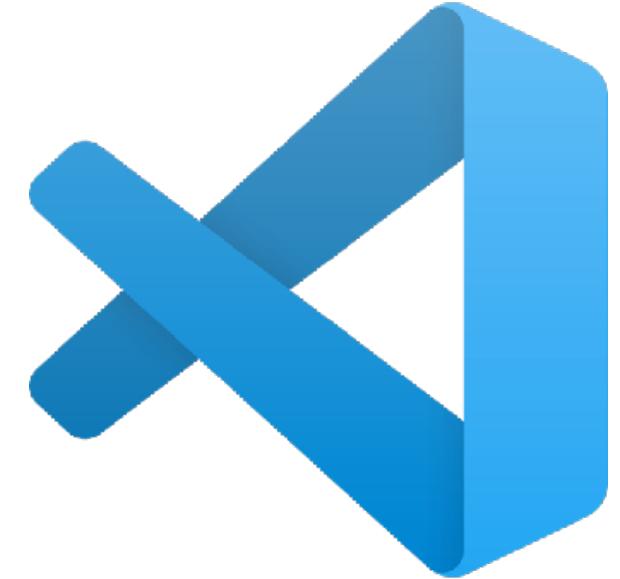


Multicursor Support

- To add cursors at arbitrary positions, select a position with your mouse and use **Alt+Click** (**⌥+Click** on macOS).
- To set cursors above or below the current position use: Keyboard Shortcut: **↖⌘↑** or **↖⌘↓**
- You can add additional cursors to all occurrences of the current selection with **⬆⌘L**.
- To select one occurrence at a time you can use **⌘D** instead
- You can select blocks of text by holding **Shift+Alt** (**⇧+↖** on macOS) while you drag your mouse. A separate cursor will be added to the end of each selected line.

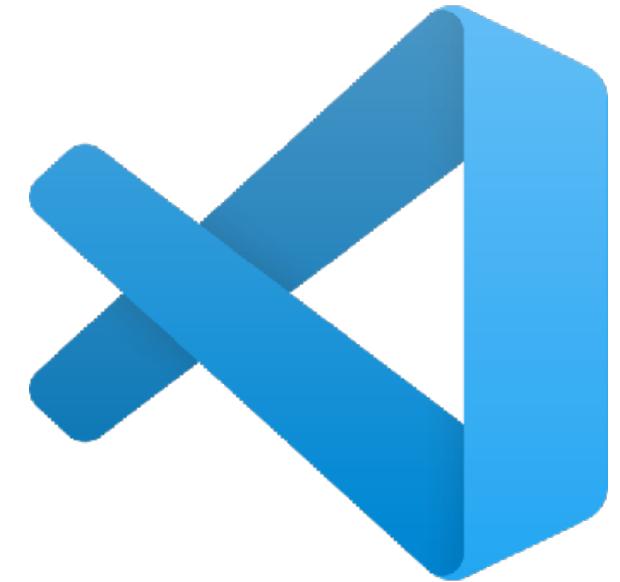


XML Editing



XML Editing on VSCode

- Depending on the Extension used, XML editing might be maintained by that particular extension
- If you require more power then consider either [RedHat's XML](#) or [VSCode-XML](#)

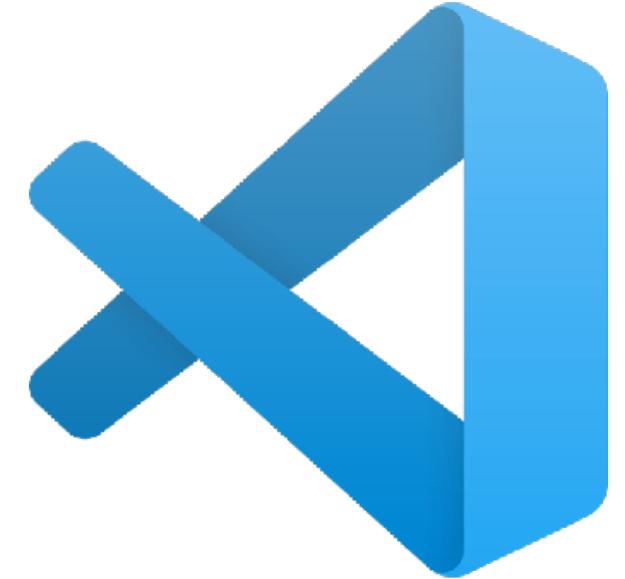


Emmet

- Emmet helps you expand HTML/XML and at times CSS tags using a concise abbreviation
- VSCode has Emmet support built in
- Use the cheatsheet for examples on using Emmet for efficient editing

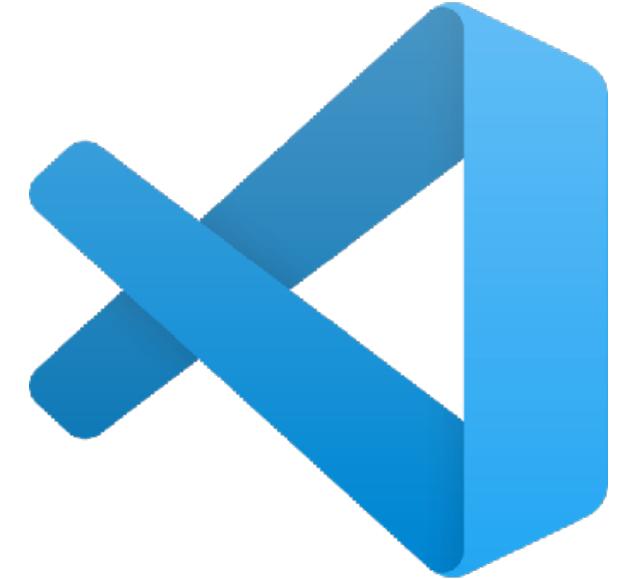


HTTP Editing with RestClient



HTTP Editing with Rest Client

- You don't always have to use Postman. Reasons include:
 - Your HTTP calls are not in your project
 - It's not configuration as code
- Go to Extensions and use the [REST Client Extension](#) by Huachao Mao
- Create a file with an .http extension
- Has a huge amount of features including variable interpolation, security, syntax highlighting, multiple language support, and more.
- A competitive extension is the [Thunder Client Extension](#)



Sending a Request

- Send your requests by either:
 - Click the **Send Request** link above the request. This link will appear automatically if the file's language mode is set to HTTP.
 - You can also use the shortcut **Ctrl+Alt+R** (**⌘ ⌘R** for macOS)
 - You can right-click in the editor and select Send Request from the context menu
 - Open the Command Palette (**⇧ ⌘P**) and select/type **Rest Client: Send Request**.
- The response will be previewed in a separate webview panel inside Visual Studio Code.
- If you prefer to use the full power of searching, selecting, or manipulating in Visual Studio Code, you can preview the response in an untitled document by setting **rest-client.previewResponseInUntitledDocument** to **true**.



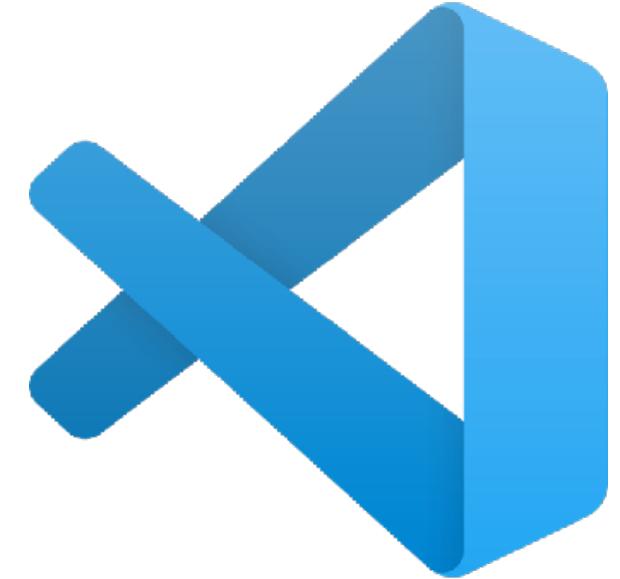
Using HTTP Editing in VSCode



- In this lab, let's create an http file and run it calling the service <https://postman-echo.com>
- Review the contents
- Try other calls like GET, POST, PUT if time remaining

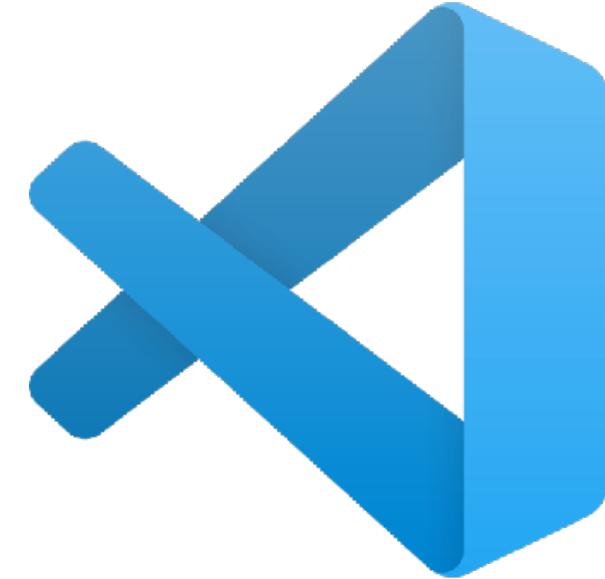


Terminals



Integrated Terminals

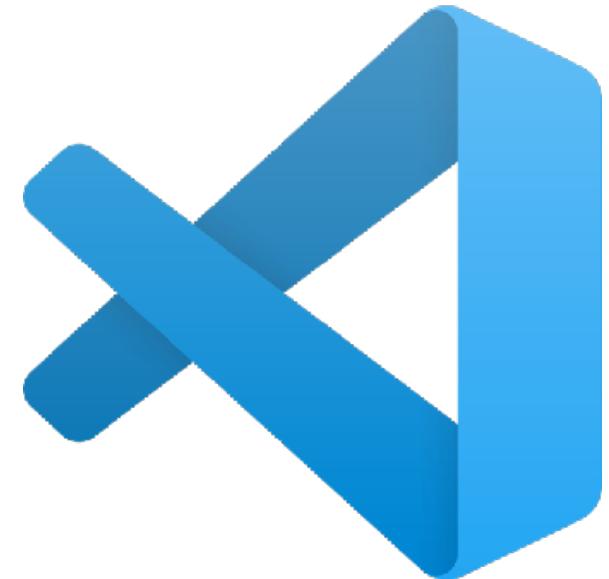
- Visual Studio Code includes a full featured integrated terminal that starts at the root of your workspace.
- It provides integration with the editor to support features like links and error detection.
- The integrated terminal can run commands such as `mkdir` and `git` just like a standalone terminal.
- Some other information on the terminal settings: <https://www.growingwiththeweb.com/2017/03/mastering-vscode-terminal.html>



Opening the Terminals

You can open the terminals in the following ways:

- From the menu, use the **Terminal > New Terminal** or **View > Terminal** menu commands.
- From the **Command Palette** ($\text{Shift} + \text{Command} + \text{P}$), use the **View: Toggle Terminal** command.
- In the Explorer, you can use the **Open in Integrated Terminal** context menu command to open a new terminal from a folder.
- To toggle the terminal panel, use the (\wedge) keyboard shortcut.
- To create a new terminal, use the $(\wedge \text{Shift} + \wedge)$ keyboard shortcut.
- If you prefer to work outside VS Code, open an external terminal with the $(\text{Shift} + \text{Command} + \text{T})$ keyboard shortcut



Viewing Status of Terminal Commands

TEST RESULTS TERMINAL OUTPUT DEBUG CONSOLE PORTS GITLENS

```
modified:  .vscode/settings.json
modified:  src/main/java/com/xyzcorp/Wallet.java

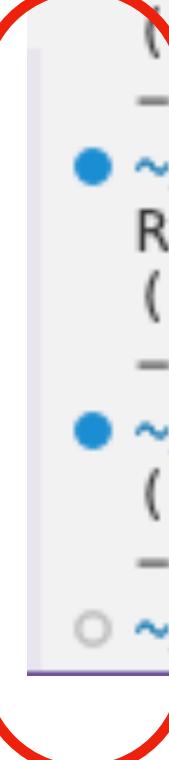
Untracked files:
(use "git add <file>..." to include in what will be committed)
  .vscode/extensions.json

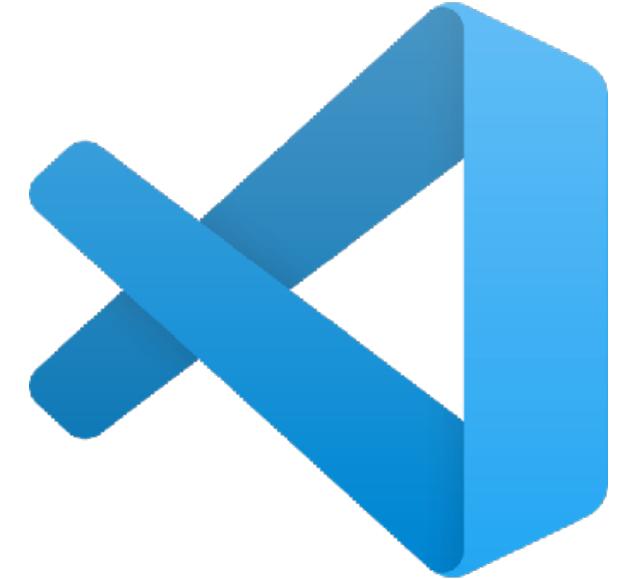
no changes added to commit (use "git add" and/or "git commit -a")
(base) -----
```

● ~/Development/mine/vscode-training-maven (main*) » ls danno@MacBook-Pro
README.md pom.xml src target

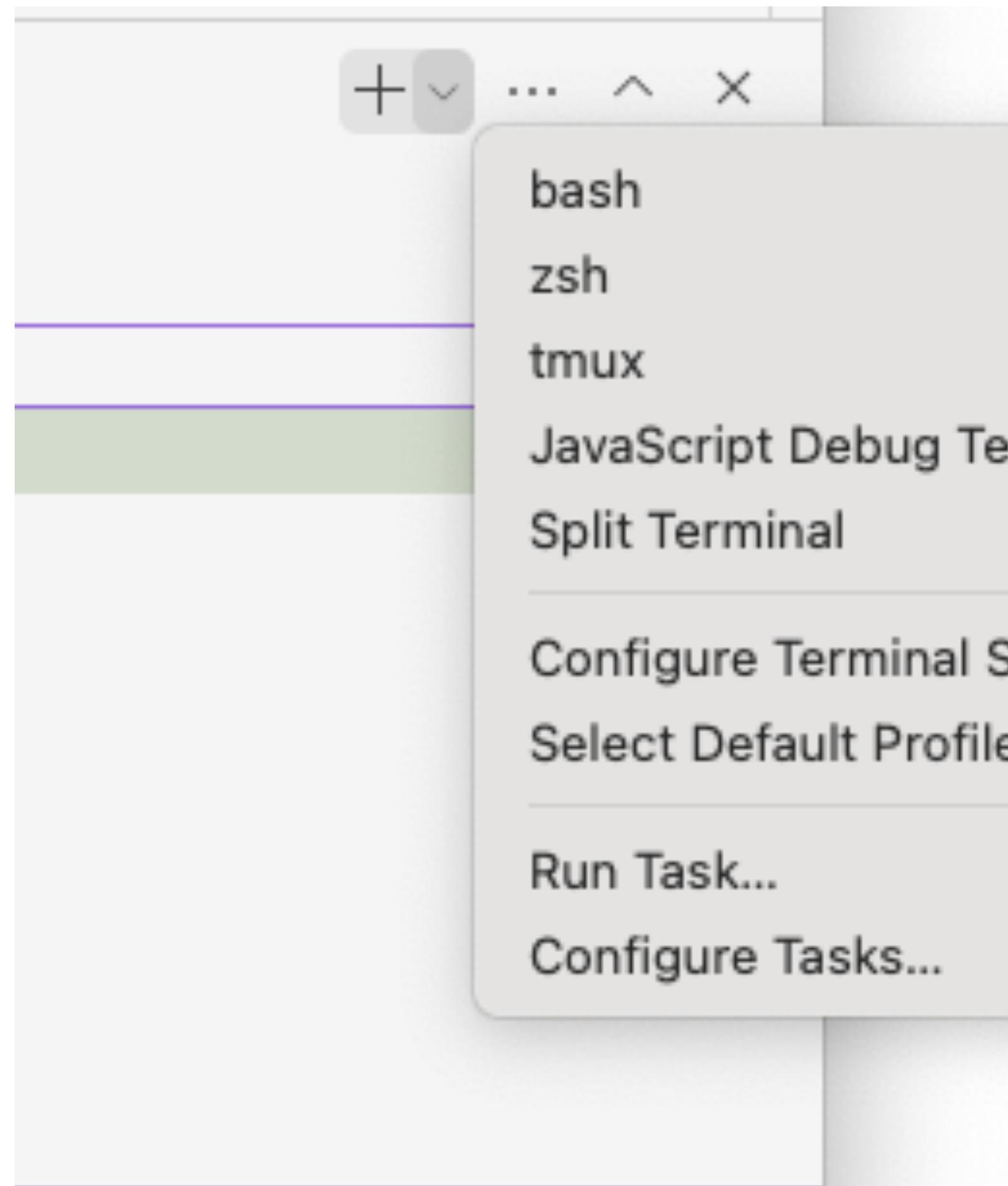
● ~/Development/mine/vscode-training-maven (main*) » git add .vscode/settings.json

○ ~/Development/mine/vscode-training-maven (main*) » █ danno@MacBook-Pro

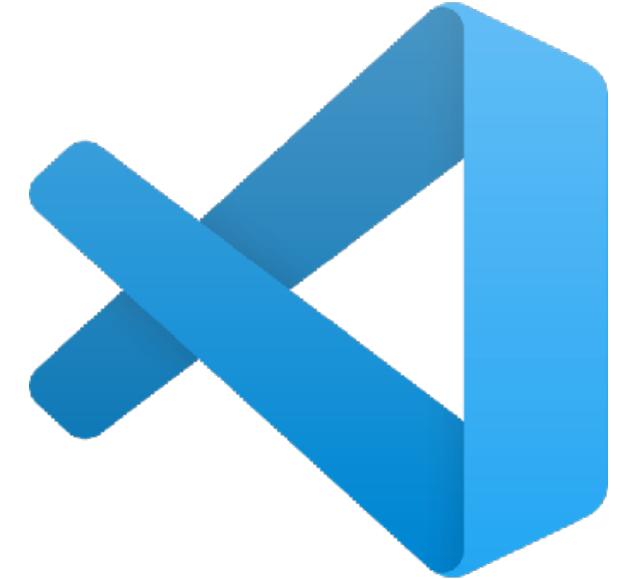




Various Terminal Shells

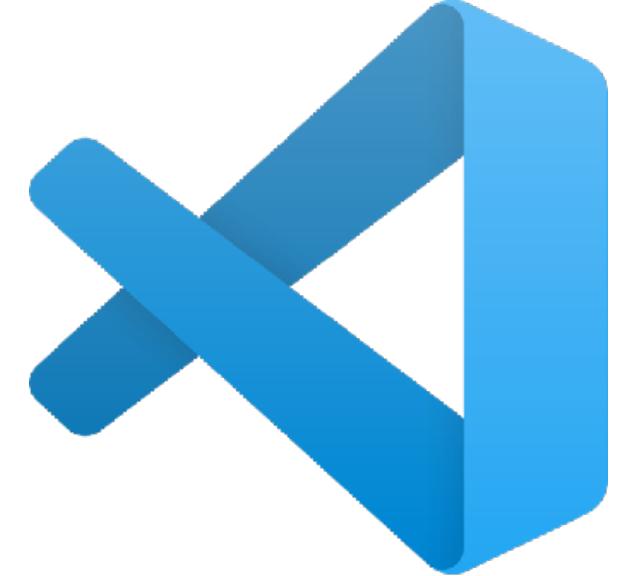


- The integrated terminal can use various shells installed on your machine, with the default being pulled from your system defaults.
- Shells are detected and presented in the terminal profiles dropdown.



Adding Terminals

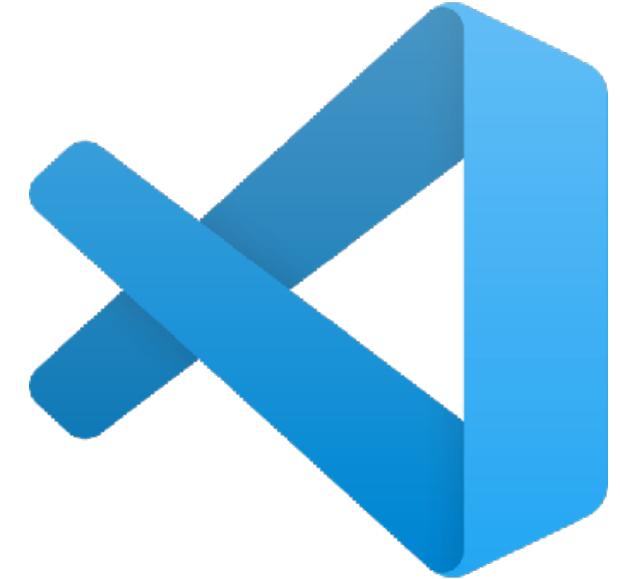
- The terminal tabs UI is on the right side of the terminal view. Each terminal has an entry with its name, icon, color, and group decoration (if any).
- Add terminal instances by selecting the `+` icon on the top-right of the **TERMINAL** panel, selecting a profile from the terminal dropdown, or by triggering the `^↑`` command.
- This action creates another entry in the tab list associated with that terminal.
- Icons may appear to the right of the terminal title on the tab label when a terminal's status changes. Some examples are a bell (macOS) and for tasks, displaying a check mark when there are no errors and an X otherwise.
- Hover the icon to read status information, which may contain actions.



Removing Terminals

Remove terminal instances by:

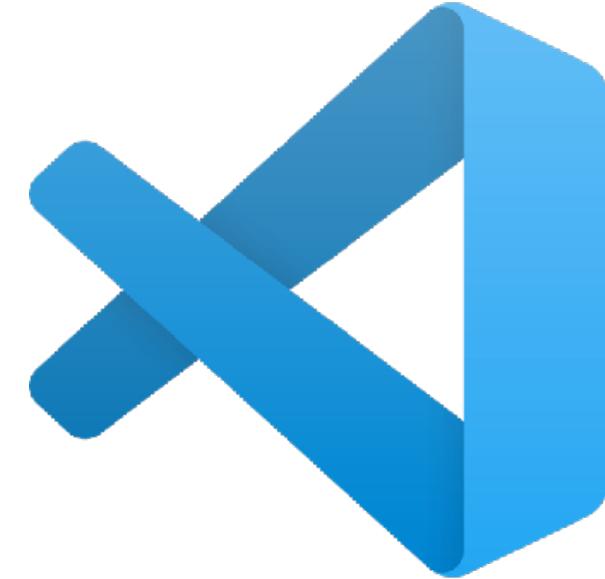
- Hovering a tab and selecting the **Trash Can** button
- Selecting a tab item and pressing **Delete**
- Using **Terminal: Kill the Active Terminal Instance** command
- Via the right-click context menu.



Terminal Groups

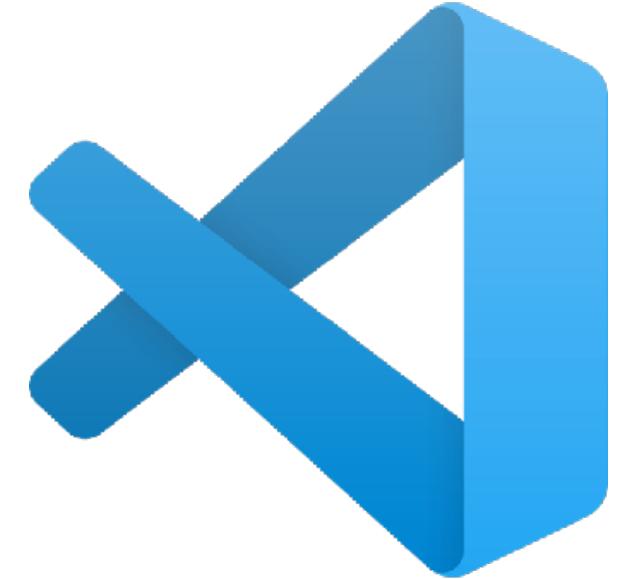
Place multiple terminals side-by-side and create a group by splitting a terminal:

- Hover over a entry in the list of terminals on the right and select the inline split button.
- Right-click the context menu and selecting the Split menu option.
- **Alt** and click on a tab, the **+** button, or the single tab on the terminal panel.
- Trigger the **⌘** command.



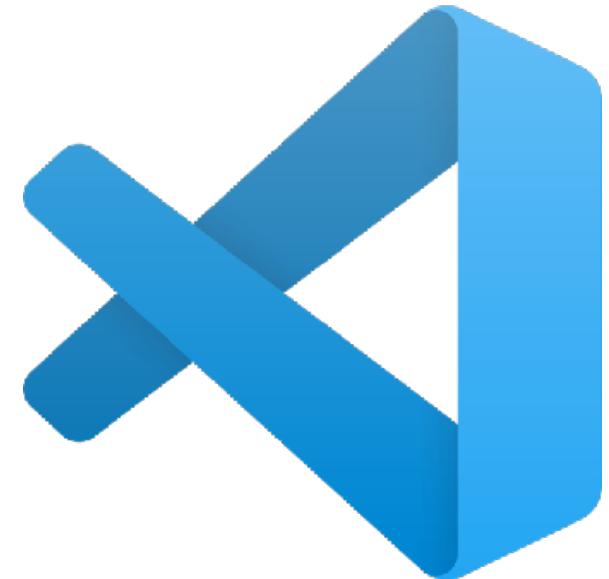
Navigating Terminal Groups

- Navigate between terminal groups using focus next `⬆⌘[` and focus previous `⬆⌘[.`
- Navigate between terminals in a group by focusing the previous pane, `↖⌘←`, or the next pane, `↖⌘→`
- Dragging and dropping tabs in the list rearranges them. Dragging a tab into the main terminal area allows moving a terminal from one group to another.
- Moving a terminal into its own group can be done with the **Terminal: Unsplit Terminal command** through the Command Palette or in the right-click context menu.



Editor Area Terminals

- You can open terminals in the editor area (terminal editors) with the
 - **Terminal: Create New Terminal in Editor Area** command
 - **Terminal: Create New Terminal in Editor Area to the Side** command
- Dragging a terminal from the terminal view into the editor area.
- Terminal editors are presented like regular editor tabs



Editor Area Terminals

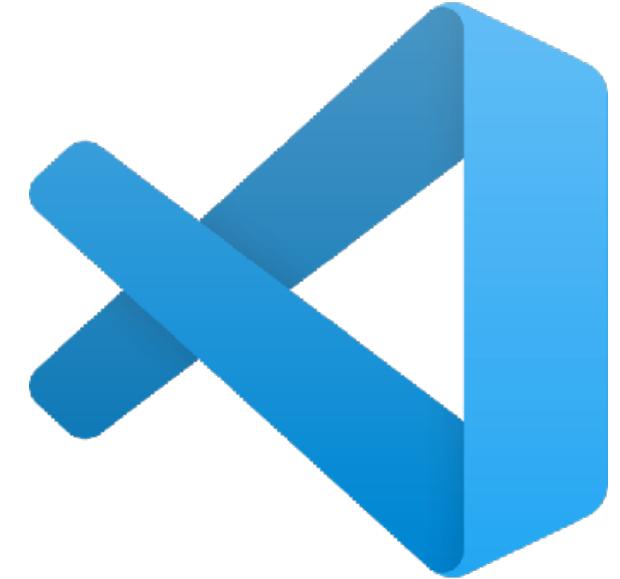
The screenshot shows the Visual Studio Code interface with a red circle highlighting the terminal area. The terminal window displays the output of a Maven build command:

```
no changes added to commit (use "git add" and/or "git commit -a")
(base) -----
~/.Development/mine/vscode-training-maven (main*) » ls
danno@MacBook-Pro
README.md pom.xml src target
(base) -----
(base) -----
~/.Development/mine/vscode-training-maven (main*) » mvn compile danno@MacBook-Pro
[INFO] Scanning for projects...
[INFO]
[INFO] ----- com.evolutionnext:vscode-training-maven -----
[INFO] Building basic-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- jacoco:0.8.11:prepare-agent (default-prepare-agent) @ vscode-training-maven ---
[INFO] argLine set to -javaagent:/Users/danno/.m2/repository/org/jacoco/org.jacoco.agent/0.8.11/org.jacoco.agent-0.8.11-runtime.jar=destfile=/Users/danno/Development/mine/vscode-training-maven/target/jacoco.exec
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ vscode-training-maven ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] skip non existing resourceDirectory /Users/danno/Development/mine/vscode-training-maven/src/main/resources-filtered
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ vscode-training-maven ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 0.623 s
[INFO] Finished at: 2023-11-26T12:12:14-07:00
[INFO]
(base) -----
~/.Development/mine/vscode-training-maven (main*) » danno@MacBook-Pro
```

The terminal also shows the output of a git status command:

```
no changes added to commit (use "git add" and/or "git commit -a")
```

The Explorer sidebar on the left shows the project structure, including files like Wallet.java, pom.xml, .gitpod.yml, settings.json, and README.md.



Navigating the Buffer

- The content in the terminal is called the buffer, with the section right above the bottom viewport being called "scrollback".
- The amount of scrollback kept is determined by the `terminal.integrated.scrollback` setting and defaults to 1000 lines.
- There are various commands available to navigate around the terminal buffer:
 - Scroll up a line - `↑⌘PageUp`
 - Scroll down a line - `↓⌘PageDown`
 - Scroll up a page - `PageUp`
 - Scroll down a page - `PageDown`
 - Scroll to the top - `⌘Home`
 - Scroll to the bottom - `⌘End`



Scrolling Previous Commands

- Command navigation is also available:
 - Scroll to the previous command - ⌘↑
 - Scroll to the next command - ⌘↓
- Scrolling will happen instantaneously, but can be configured to animate over a short duration with the `terminal.integrated.smoothScrolling` setting.

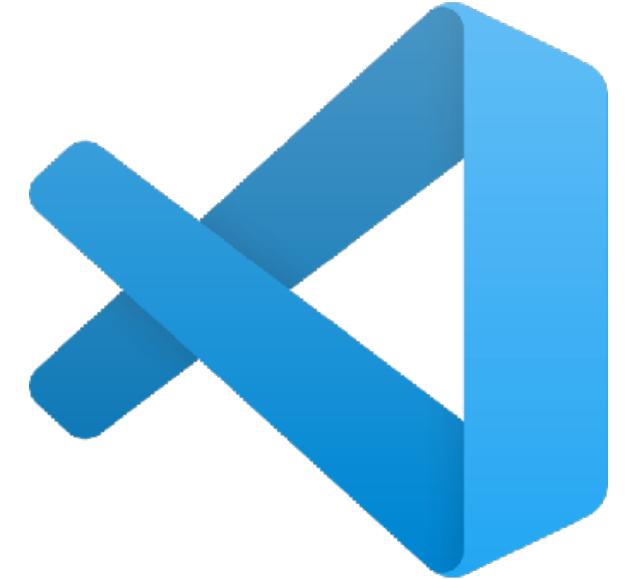


Other Features

- Various Link Handlers are available for URLs, Files, Folders, and Words. You can hover over a link and perform ([`⌘+Click`](#))
- You can perform the **Open Detected Link** command ([`⇧ ⌘ O`](#)) can be used to access links via the keyboard
- You can find text in your terminal using ([`⌘ F`](#))
- You can drag a file into the terminal and it will input the path.
- You can automate terminals with tasks in the [`.vscode/tasks.json`](#) file, which will be a part of your project workspace and shared as configuration as code.
- On local systems, you can change the fonts and style on your terminal as well

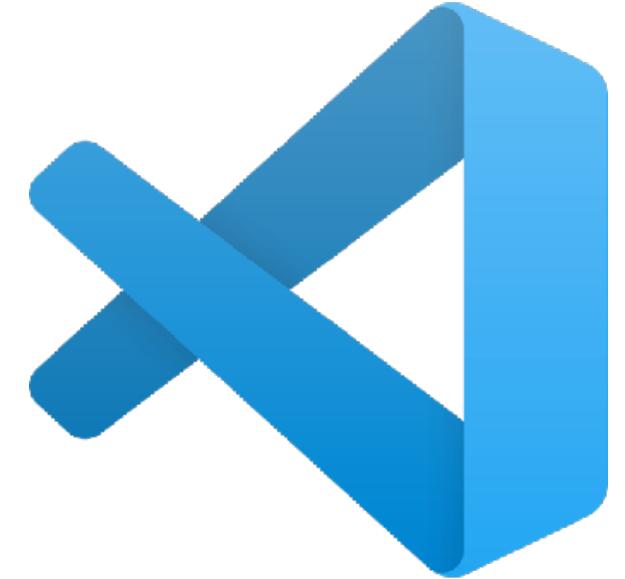


Docker



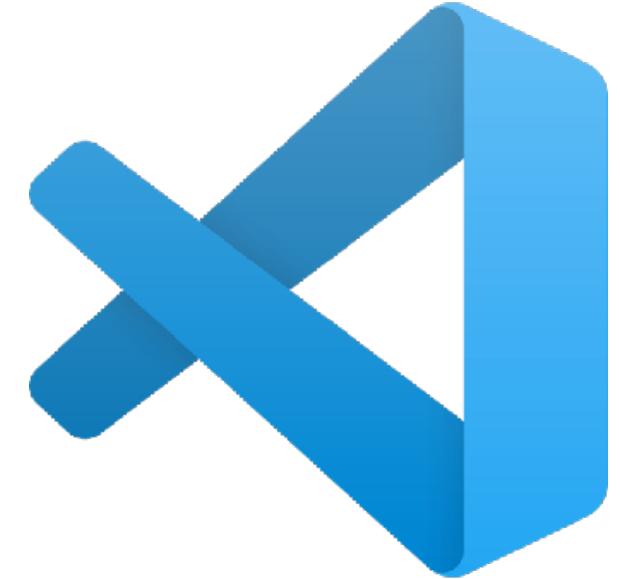
Docker in VSCode

- The Docker extension makes it easy to build, manage, and deploy containerized applications in Visual Studio Code.
- Ensure that Docker is installed
- Install the Docker extension



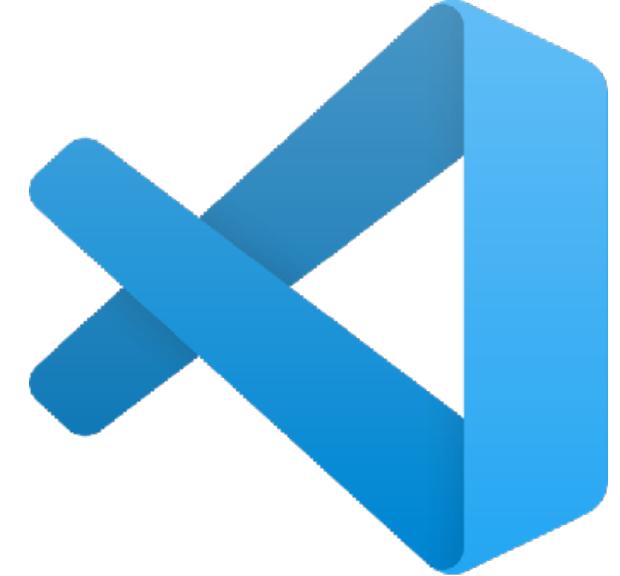
Features in Docker Files

- You can get IntelliSense by clicking `^Space` when editing your `Dockerfile` and `docker-compose.yml` files, with completions and syntax help for common commands.
- In addition, you can use the Problems panel (`⇧⌘M`) to view common errors for `Dockerfile` and `docker-compose.yml` files.



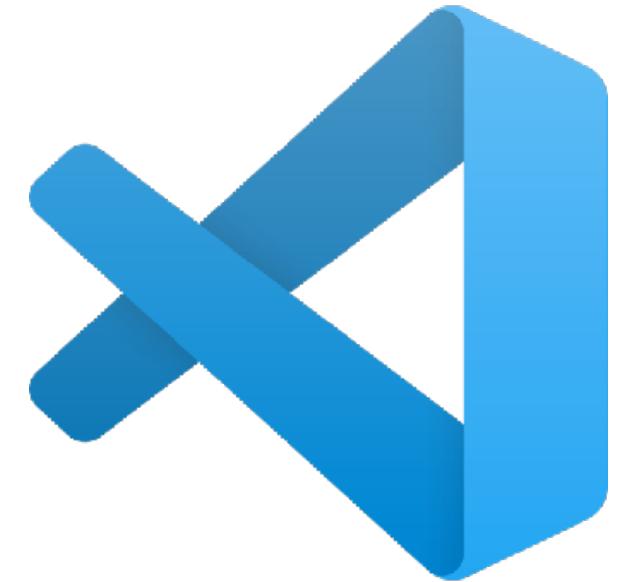
Generating Docker Files

- You can add Docker files to your workspace by opening the Command Palette (⇧⌘P) and using Docker: Add Docker Files to Workspace command.
- The command will generate `Dockerfile` and `.dockerignore` files and add them to your workspace.
- The command will also ask you if you want to add Docker Compose files as well, but this is optional.
- The extension can scaffold Docker files for most popular development languages (C#, Node.js, Python, Ruby, Go, and Java) and customizes the generated Docker files accordingly.
- When these files are created, it creates the necessary artifacts to provide debugging support for Node.js, Python, and .NET (C#).



Docker Explorer

- The Docker extension contributes a Docker Explorer view to VS Code.
- The Docker Explorer lets you examine and manage Docker assets: containers, images, volumes, networks, and container registries.
- You can rearrange the Docker Explorer panes by dragging them up or down with a mouse and use the context menu to hide or show them.



Docker Commands Available

>docker containers

Docker Containers: Refresh

Docker Containers: Attach Shell

Docker Containers: Configure Explorer...

Docker Containers: Inspect

Docker Containers: Prune...

Docker Containers: Remove...

Docker Containers: Restart

Docker Containers: Start

Docker Containers: Stop

Docker Containers: View Logs

Docker: Focus on **Containers** View

>docker images

Docker Images: Build Image...

Docker Images: Run Azure CLI

Docker Images: Configure Explorer...

Docker Images: Inspect

Docker Images: Prune...

Docker Images: Push

Docker Images: Refresh

Docker Images: Remove...

Docker Images: Run

Docker Images: Run Interactive

Docker Images: Tag...

Docker Registries: Deploy **Image** to Azure App **Service**...

Docker: Focus on **Images** View

>docker registries

Docker Registries: Connect Registry...

Docker Registries: Copy Image Digest

Docker Registries: Delete Image...

Docker Registries: Deploy Image to Azure App Service...

Docker Registries: Disconnect

Docker Registries: Log In to Docker CLI

Docker Registries: Log Out of Docker CLI

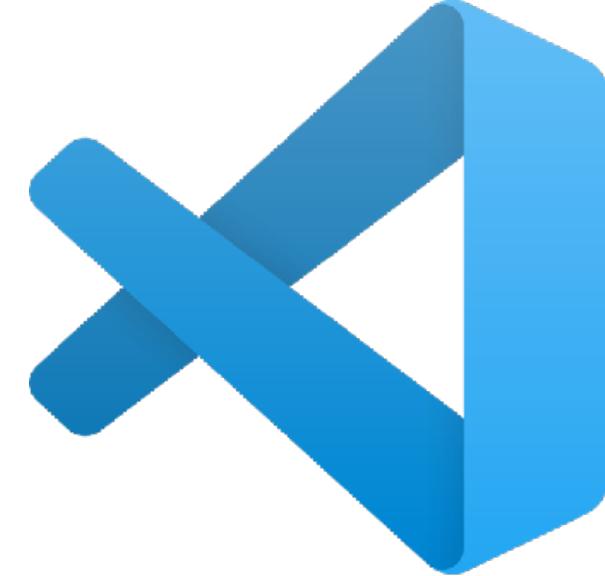
Docker Registries: Pull Image

Docker Registries: Pull Repository

Docker Registries: Refresh

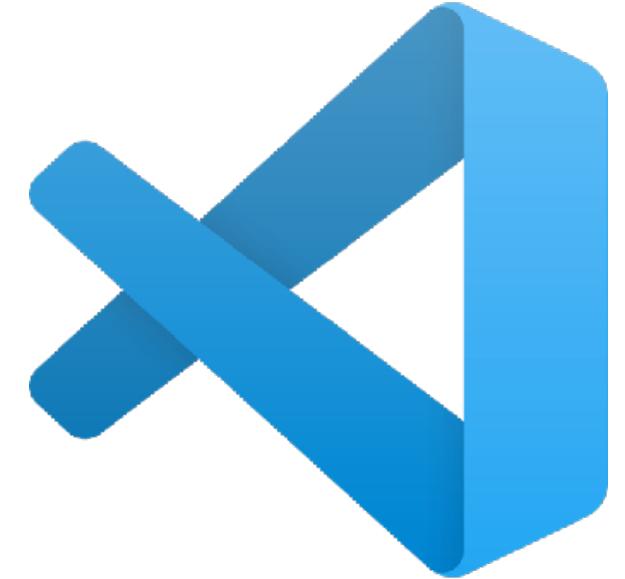
Docker Registries: Set as Default

Docker: Focus on **Registries** View



Docker Compose

- Docker Compose lets you define and run multi-container applications with Docker.
- IntelliSense and tab completions when authoring `docker-compose.yml` files.
Press `^Space` to see a list of valid Compose directives.
- Tooltips are provided when hovered over certain element.
- You can use `Compose Up` allows you to run all of your services at once
- You can use `Compose Up - Select Services` lets you select any combination of the services you want to run.
- You can use `Compose Down` to bring down various containers launch by Docker Compose
- When run docker images can run in specific ports and exposed through gitpod.io or Codespaces



Docker Registries

- You can display the content and push, pull, or delete images from Azure Container Registry, Docker Hub, GitLab, and more
- You can pull images directly from docker registries



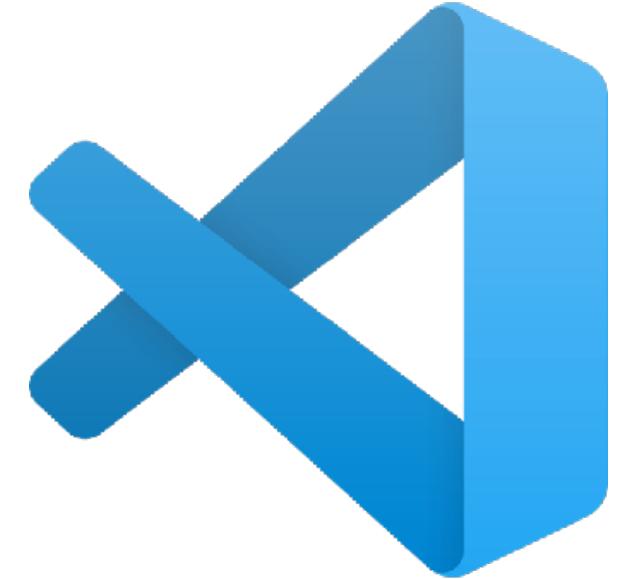
Using Docker in VSCode



- In this lab, we will run Docker in VSCode
 - Running and Pulling Images
 - Programming against a service
 - Opening Ports to the location



SQL Editing



SQLTools in VSCode

- SQLTools provides connections to many of the most commonly used databases, making it easier to work with your data.
- Connects to MySQL, PostgreSQL, Microsoft SQL Server and many more...
 - Beautifier and formatter for SQL code
 - Query runner, history and bookmarks
 - Connection explorer
 - Generator for INSERT queries
 - Pluggable driver architecture
- To install **Click the Extensions view** (in the left bar), then search for [@tag:sqltools-driver](#).
- This lists all the drivers available for SQLTools, choose the appropriate one for your use.



Using SQL Tools

- Click the SQLTools icon (in the left bar)
- Create a connection to your database. To do this, hover over **CONNECTIONS** in the SQLTools pane.
- Click the "**Add New Connection**" icon and choose the driver and enter the connection parameters.



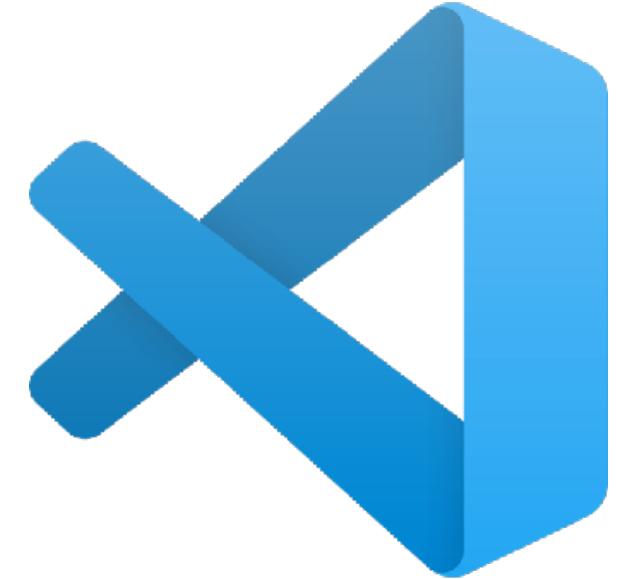
Trying SQL in VSCode



- Let's use the Sakila Database within VSCode after we load it up with Docker inside of VSCode
- Run some examples using SQLTools

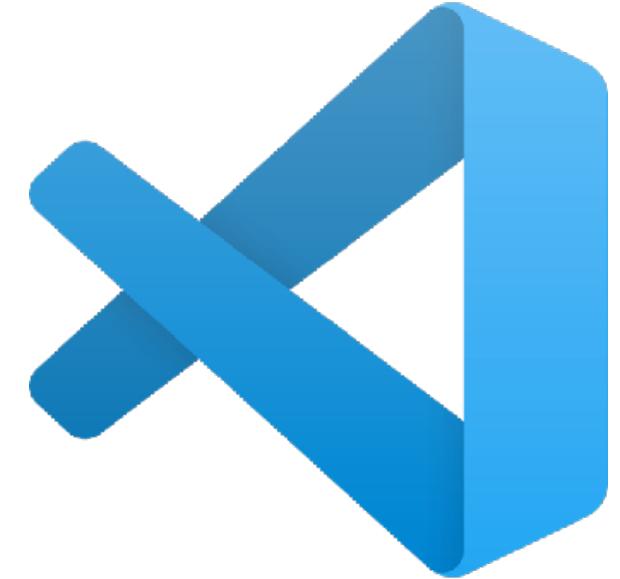


Git



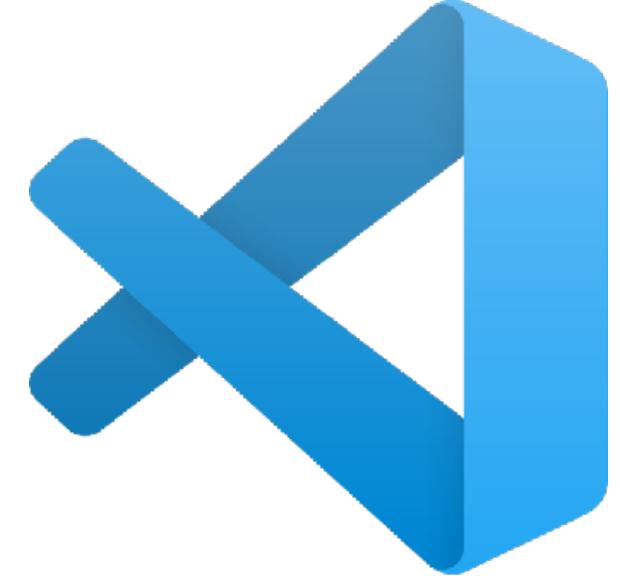
Git in VSCode

- Easily manage your source code and collaborate with others
- Supports common Git actions like pushing and pulling code, creating and merging branches, and committing code changes
- Actions are straight from the editor



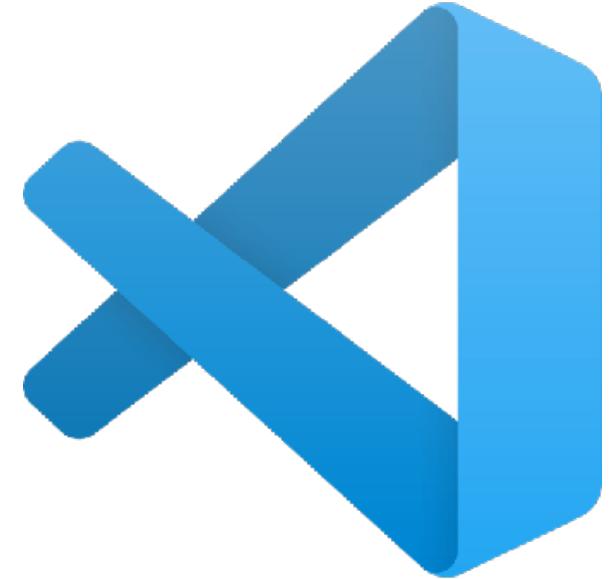
Git Requirements

- To use Git and GitHub in VS Code, first make sure you have Git installed on your computer.
- If Git is missing, the Source Control view shows instructions on how to install it. Make sure to restart VS Code afterwards.
- Additionally you can sign into VS Code with your GitHub account in the Accounts menu in the lower right of the Activity bar to enables additional features like Settings Sync, but also cloning and publishing repositories from GitHub.

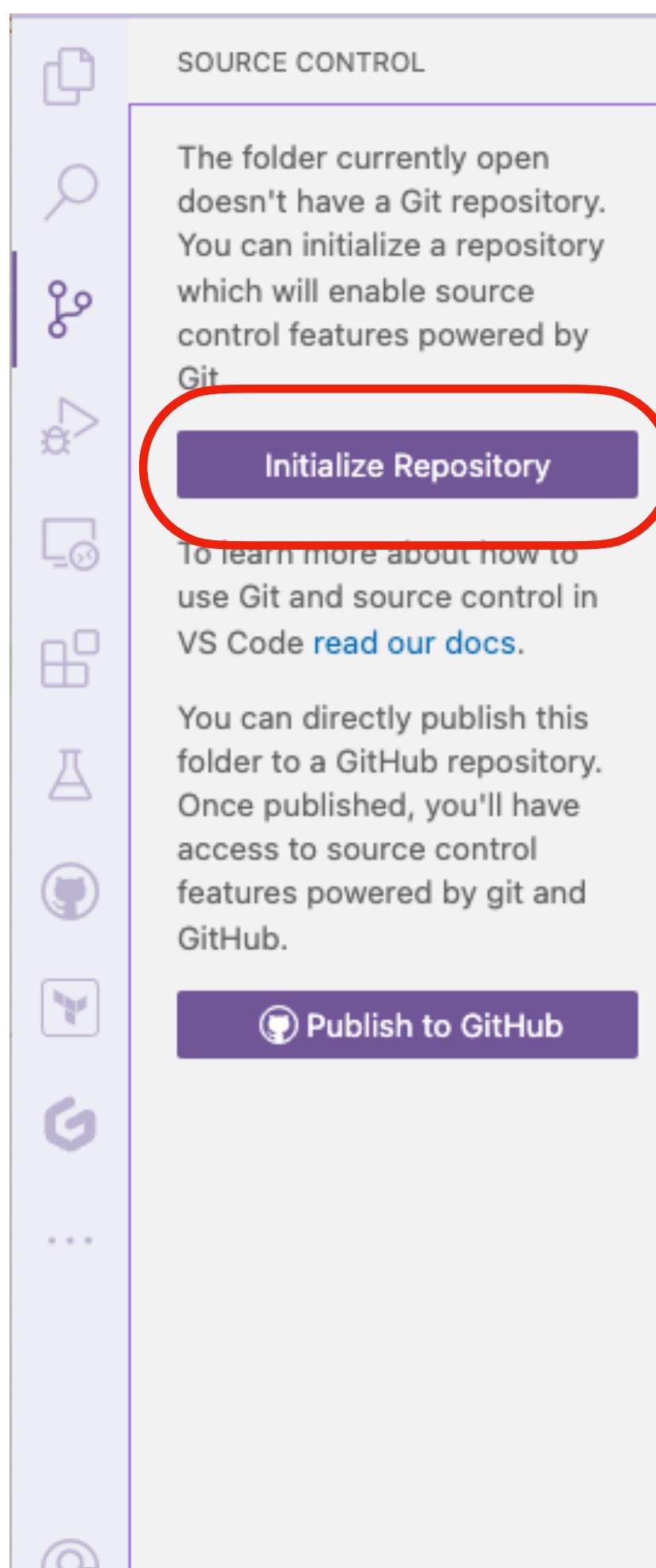


Cloning a Repository

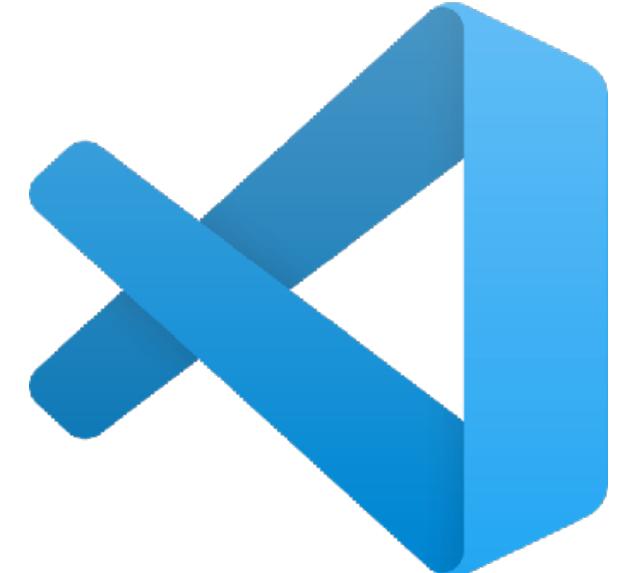
- To clone a repository from GitHub, execute the **Git: Clone** command or select the **Clone Repository** button in the Source Control view.
- If you clone from GitHub, VS Code will prompt you to authenticate with GitHub. This allows you to search all available repositories and clone private repositories.
- For other Git providers, enter the repository URL and select **Clone** and pick a folder.
- VS Code opens the folder once the repository is cloned on your local machine.



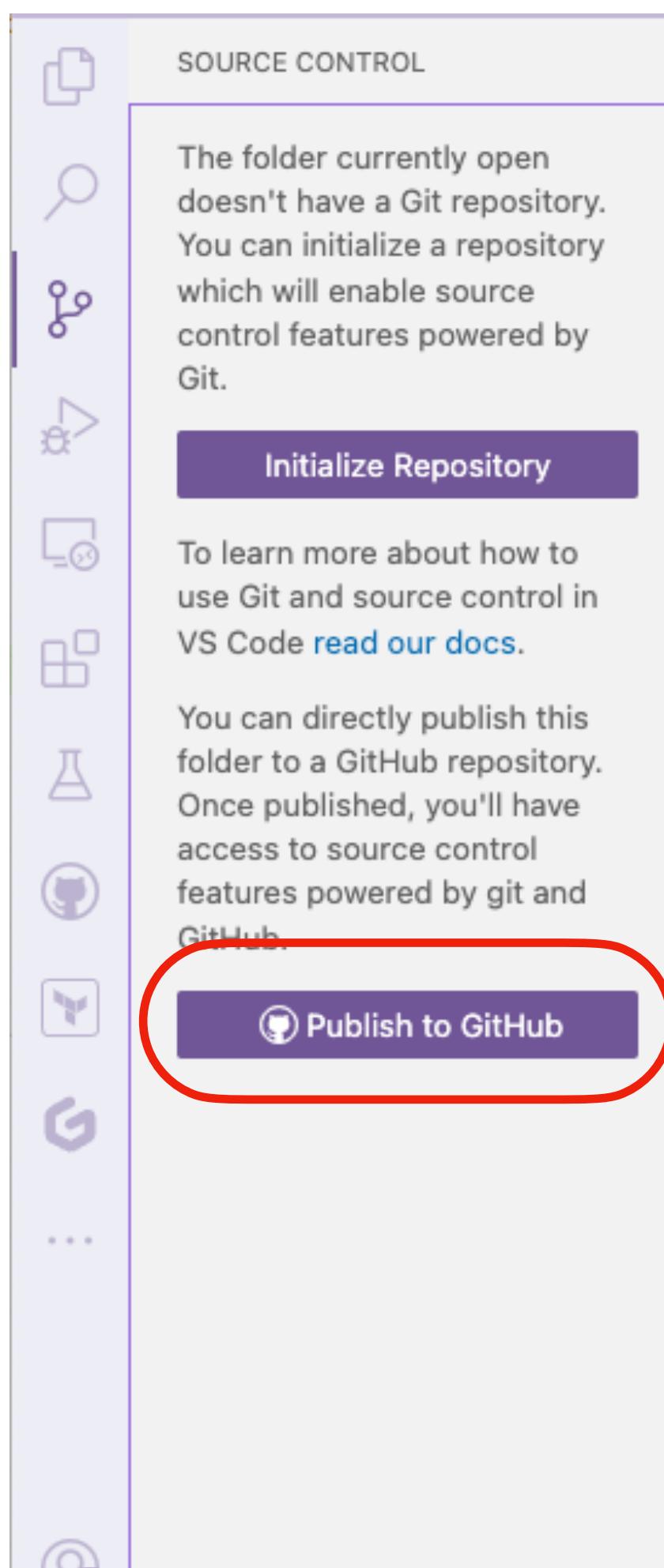
Initialize a Local Folder



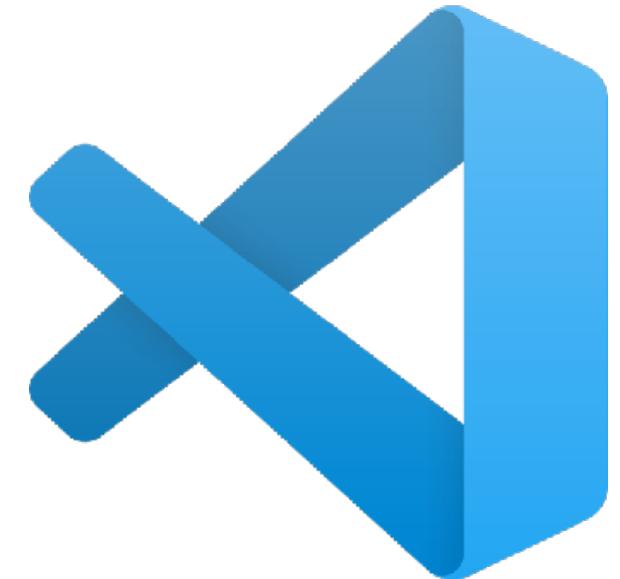
- To initialize a new local repository, pick an existing or new folder on your computer and open it in VS Code.
- In the Source Control view, select the Initialize Repository button.
- This creates a new Git repository in the current folder, allowing you to start tracking code changes.



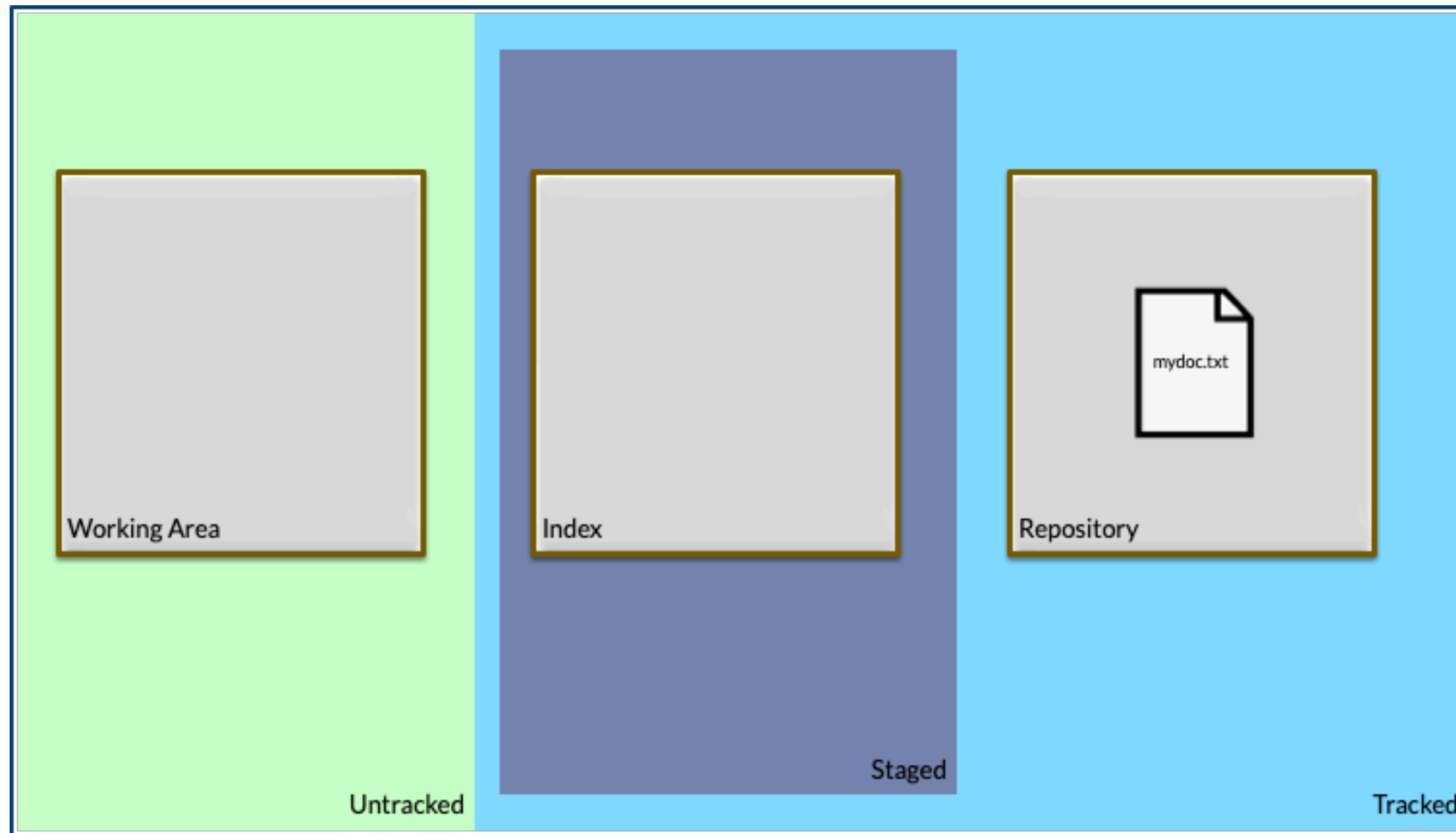
Publish Local to Github

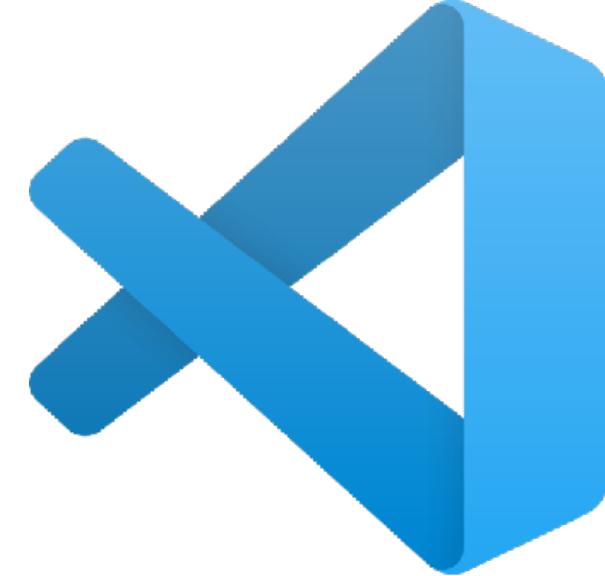


- Once you have a local Git repository set up, you can publish it to GitHub.
- This will create a new repository on your GitHub account, and push your local code to the remote repository.
- Use the Publish to GitHub command button in the Source Control view. You can then choose a name and description for the repository, and whether to make it public or private. Once the repository has been created, VS Code will push your local code to the remote repository.



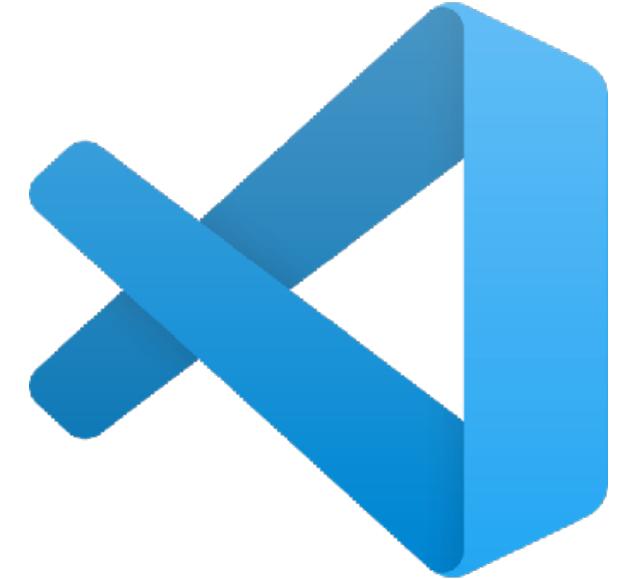
Git Workflow Reminder





Staging and Committing Code Changes

- To stage a file, select the **+** (plus) icon next to the file in the **Source Control** view (**⌃↑G**).
- This will add the file to the **Staged Changes** section, indicating that it will be included in the next commit.
- Staged changes can also be discarded by selecting the **-** (minus) icon next to the file.
- To commit your staged changes, type a commit message in the upper text box and select the **Commit** button.
- This saves your changes to the local Git repository, allowing you to revert to previous versions of your code if needed.
- You can navigate through and review all local file changes and commits in the **Timeline** view available in the bottom of the Explorer.



Pushing and Pulling

- Once you have made commits to your local Git repository, you can push them to the remote repository.
- The Sync Changes button indicates how many commits are going to be pushed and pulled.
- Selecting the Sync Changes button downloads (pull) any new remote commits and uploads (push) new local commits to the remote repository.
- You can enable the **Git: Autofetch** setting to always get an up-to-date remote commit indicator.



Using Branches

- In Git, branches allow you to work on multiple versions of your codebase simultaneously. This is useful for experimenting with new features or making large code changes without affecting the main codebase.
- The branch indicator in the Status bar shows the current branch and lets you switch to new and existing branches.
- To create a new branch, select the branch indicator and choose to create it from the current branch or another local one
- Type a name for the new branch, and confirm. VS Code creates a new branch and switches to it, allowing you to make changes to your code without affecting the main branch.
- To push the branch to the remote repository, select **Publish Branch** in the **Source Control** view. This will create a new branch on the remote repository, allowing you to collaborate with others in that branch.



Using VSCode for Git Merge & Git Diff

```
$ git config --global diff.tool vscode  
$ git config --global difftool.vscode.cmd 'code --wait  
--diff $LOCAL $REMOTE'
```

```
$ git config --global merge.tool vscode  
$ git config --global mergetool.vscode.cmd 'code --wait  
$MERGED'
```



Using Version Control in VSCode



- In this lab we will try out managing Git repositories
 - Committing
 - Pushing & Pulling
 - Conflict Resolution



Conclusion