# CODE QUALITY


Pod 1: Miguel Sanchez, Kumar Pathak, and Jason Buras

# WHAT WE WILL LEARN TODAY

- **What is Code Quality?**
- Understanding the concept and why it is crucial for software development.
- **Why is Code Quality Important?**
- Exploring the benefits of maintaining high code quality and its impact on long-term project success.
- **Different Types of Code Quality Tools**
- An introduction to tools like SonarQube,  Pylint, and Prettier.
- **How to Use These Tools**
- A hands-on guide to installing, configuring, and integrating these tools into your development workflow.

# CASE STUDY OF BAD CODE QUALITY:

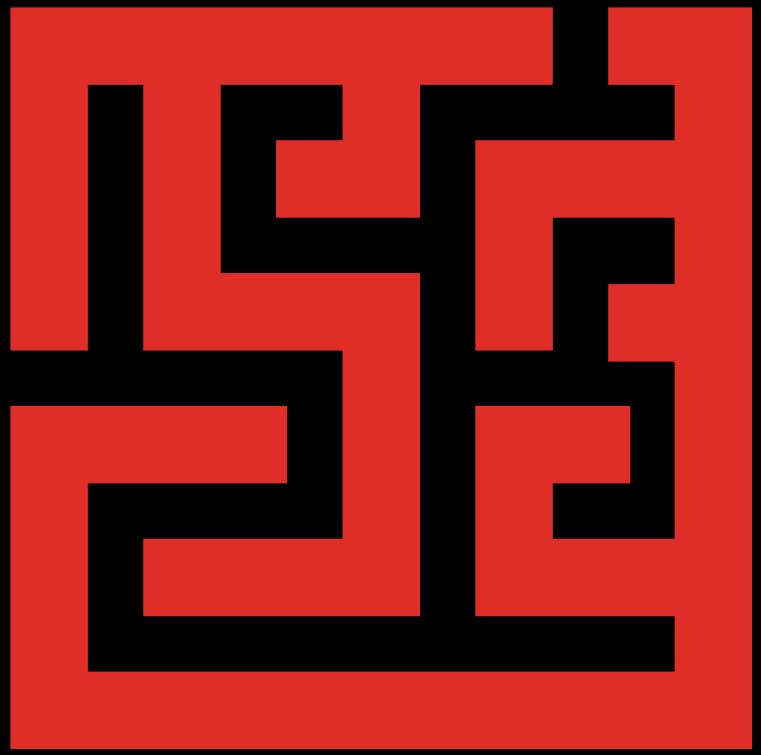
- Bad code gallery

# BAD CODE EXAMPLE

```python

def a(x, y):
    return x + y

def s(x, y):
    return x - y

def m(x, y):
    return x * y

def d(x, y):
    if y == 0:
        return "Error"
    return x / y

def c():
    print("Simple Calc")
    while True:
        c = input("cmd: ").lower()
        if c == "exit":
            break
        if c not in ["add", "sub", "mul", "div"]:
            print("Bad cmd")
            continue
        try:
            n1 = float(input("n1: "))
            n2 = float(input("n2: "))
        except:
            print("Bad input")
            continue
        if c == "add":
            r = a(n1, n2)
        elif c == "sub":
            r = s(n1, n2)
        elif c == "mul":
            r = m(n1, n2)
        elif c == "div":
            r = d(n1, n2)
        print(f"Result: {r}")

if __name__ == "__main__":
    c()

```

# WHAT IS IT FOR?

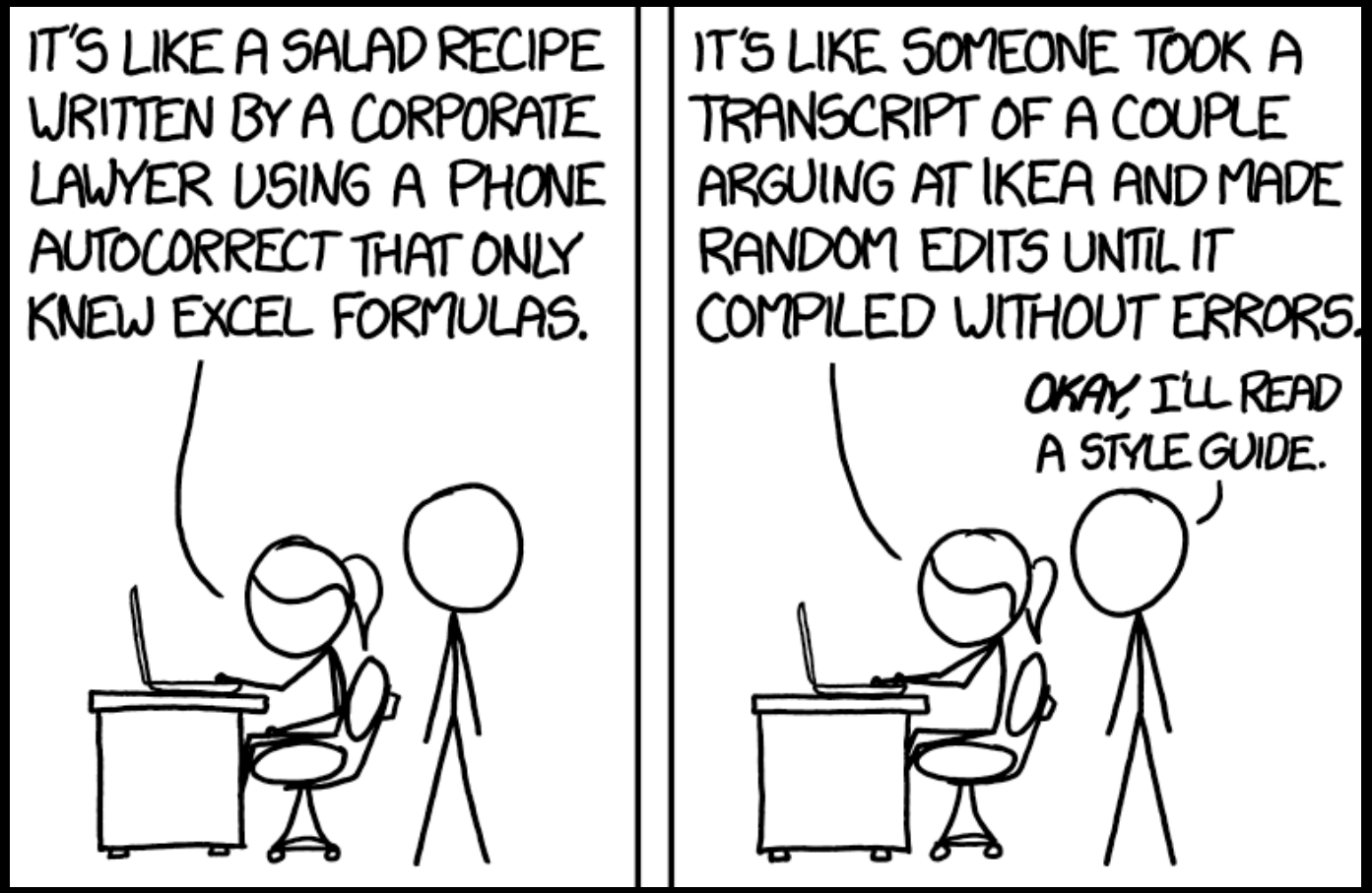
Code quality is essential for developing maintainable, readable, and error-free software. This workshop will cover tools and techniques for analyzing code quality, identifying potential issues early, and enforcing coding standards. Participants will learn how to integrate these tools into their development process, enabling continuous monitoring and improvement of code quality.



# CODE QUALITY

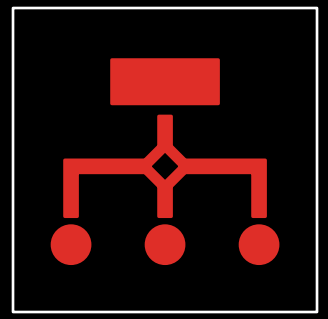
Poor code quality is an umbrella term for multiple issues with the codebase:

- Code that exhibits buggy behavior
- Slow implementation
- Messy code with high coupling and low cohesion (a.k.a. spaghetti code)
- Unmaintainable code
- Usage of obsolete libraries/frameworks
- Code repetition that leads to costly refactoring



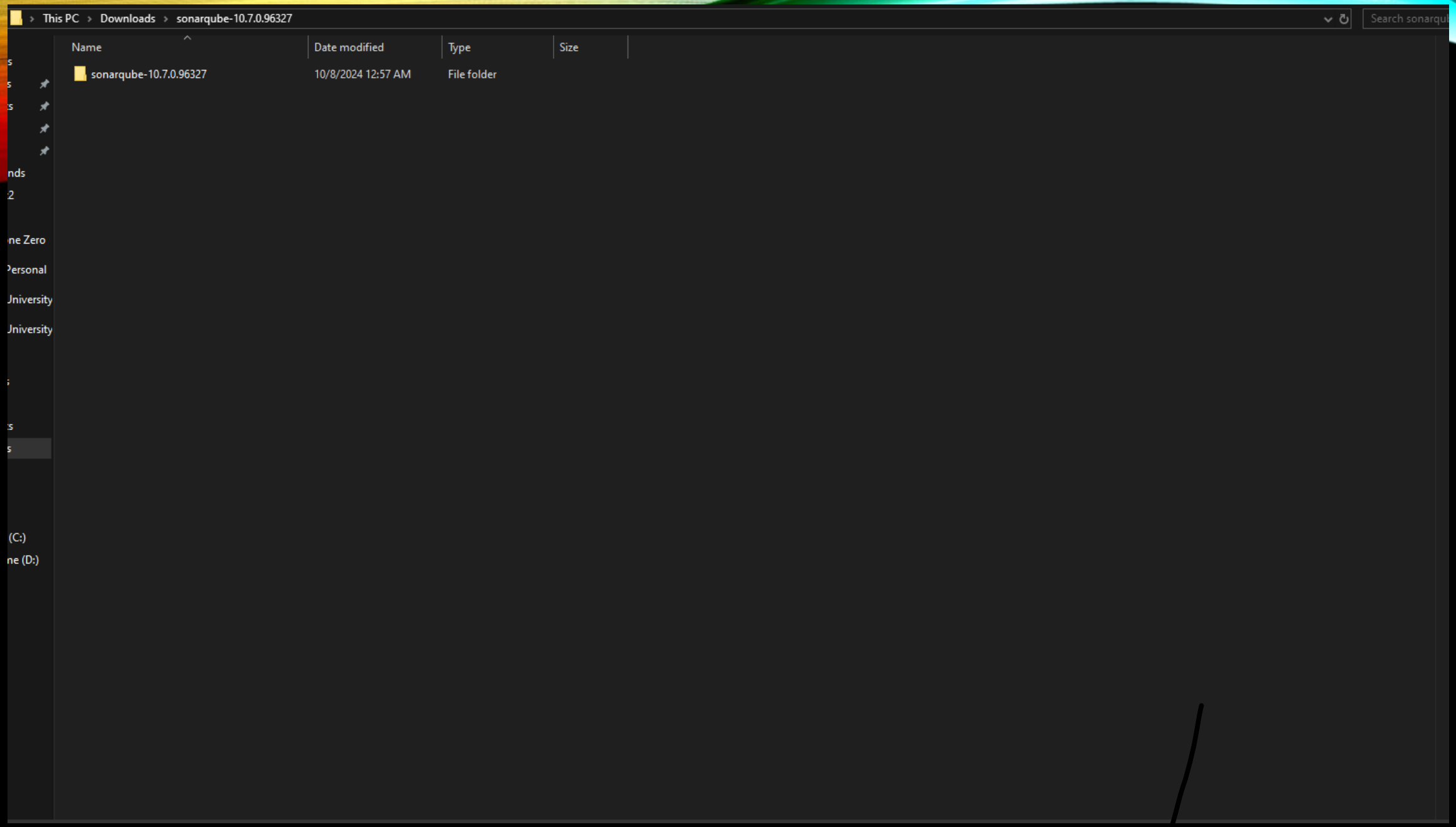
# SONARQUBE

SonarQube is an effective static code analysis tool that examines code for bugs, vulnerabilities, code smells, and adherence to coding standards. It supports various programming languages and offers comprehensive insights into code quality and maintainability.



Takes the source code from the repository or a config parameter is used to set the place from which you want the source code to be taken. SonarQube cube supports plugin control systems like git. the build tool automatically pulls the software from the database.

https://youtu.be/xeTwG9XFFTE



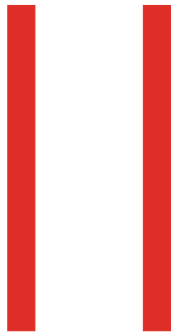

THE SONARQUBE SERVER REQUIRES JAVA JDK 17 OR HIGHER
DOWNLOAD HTTPS://WWW.ORACLE.COM/JAVA/TECHNOLOGIES/JAVASE/JDK17-ARCHIVE-DOWNLOADS.HTML

# SETTING UP SONARQUBE

We didn't use Pi's to setup, but the process should be much of the same, except on most likely a linux distro.

Resource playlist:

https://youtu.be/3F70-Yl0KWw?list=PLJRhBldgqe1oWB8kGypExY0Ru2_QP4Hvy

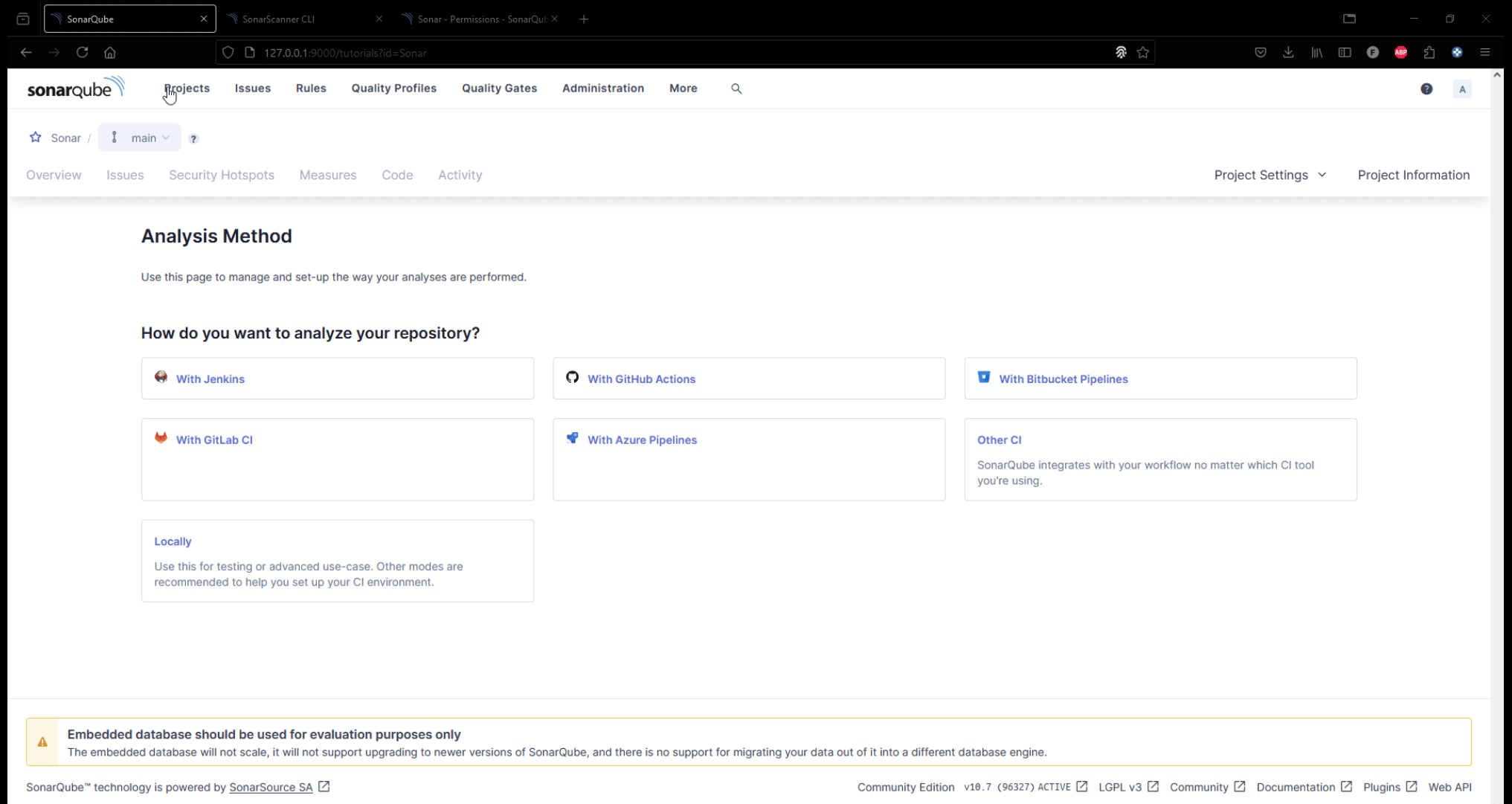


# RUNNING CODE ANALYSIS

https://youtu.be/ezMqyPbwxn4

# CUSTOMIZING QUALITY GATES

Coverage highlights which parts of the codebase are covered by tests, which are not, and which parts have partial coverage, thereby providing insights into potential areas needing better test coverage



# INTEGRATING WITH CI/CD:

Update build.yml · msan37/Son

https://github.com/msan37/Sonar/actions/runs/11233819930/job/31229661188

msan37 / Sonar

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

You have successfully requested the workflow to be canceled.

Java CI with Maven

Update build.yml #5

Re-run jobs | Latest #2

Summary

Jobs

build

Run details

Usage

Workflow file

Re-run all jobs

A new attempt of this workflow will be started, including **all the jobs**:

build

Enable debug logging

Cancel | Re-running...

Search logs

| | |
|---|---|
| Set up JDK 11 | 2s |
| Build with Maven | 1s |
| Post Set up JDK 11 | 0s |
| Post Run actions/checkout@v4 | 1s |
| Complete job | 0s |
| | 0s |

github.com

# GOOD CODE EXAMPLE

```python
def add(x, y):
    """Adds two numbers."""
    return x + y

def subtract(x, y):
    """Subtracts two numbers."""
    return x - y

def multiply(x, y):
    """Multiplies two numbers."""
    return x * y

def divide(x, y):
    """Divides two numbers. Checks for division by zero."""
    if y == 0:
        return "Error: Cannot divide by zero."
    return x / y

def calculator():
    """Runs the command-line calculator."""
    print("Simple Calculator App")
    print("Available commands: add, subtract, multiply, divide, exit")

    while True:
        # Get the user input
        command = input("Enter command (add/subtract/multiply/divide/exit): ").lower()

        if command == "exit":
            print("Exiting calculator. Goodbye!")
            break

        if command not in ["add", "subtract", "multiply", "divide"]:
            print("Invalid command. Please try again.")
            continue

        try:
            # Get the numbers from the user
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter numeric values.")
            continue

        # Perform the calculation
        if command == "add":
            result = add(num1, num2)
        elif command == "subtract":
            result = subtract(num1, num2)
        elif command == "multiply":
            result = multiply(num1, num2)
        elif command == "divide":
            result = divide(num1, num2)

        print(f"Result: {result}")

if __name__ == "__main__":
    calculator()
```

# CODE REVIEW BEST PRACTICES

Emphasize the importance of code reviews in maintaining code quality. Provide best practices for conducting effective code re-views, focusing on identifying code smells, ensuring adherence to coding standards, and improving code readability.

Good vs bad

Healthy Code Reviews (basics of communication and repo etiquite

# USING LINTERS AND FORMATTERS:

- A **linter** is a tool that analyzes code to detect potential errors, bugs, stylistic issues, and violations of coding standards. It helps developers identify and fix problems in their code early, before they become bigger issues. Linters provide feedback on code quality and enforce coding guidelines, ensuring consistency across a project.
- **Syntax checking**: Detects syntax errors or malformed code.
- **Style enforcement**: Ensures the code adheres to coding standards (like PEP8 in Python).
- **Error detection**: Identifies potential bugs or problematic code (e.g., undefined variables, unused imports).
- **Code consistency**: Helps maintain uniform code formatting across a team or project.

# DIFFERENT NAMING CONVENTIONS

Different naming conventions are used to enhance code readability and maintainability.

While you can technically name functions, variables, classes, and other code elements however you want (within the language's rules), there are well-established guidelines that have been developed over time.

These guidelines are not enforced by the programming language itself but are community-driven style standards that promote best practices for code formatting, naming conventions, and overall structure. Following them improves consistency and collaboration in a project.

# DIFFERENT NAMING CONVENTIONS

Certain languages have certain guidelines associated with them

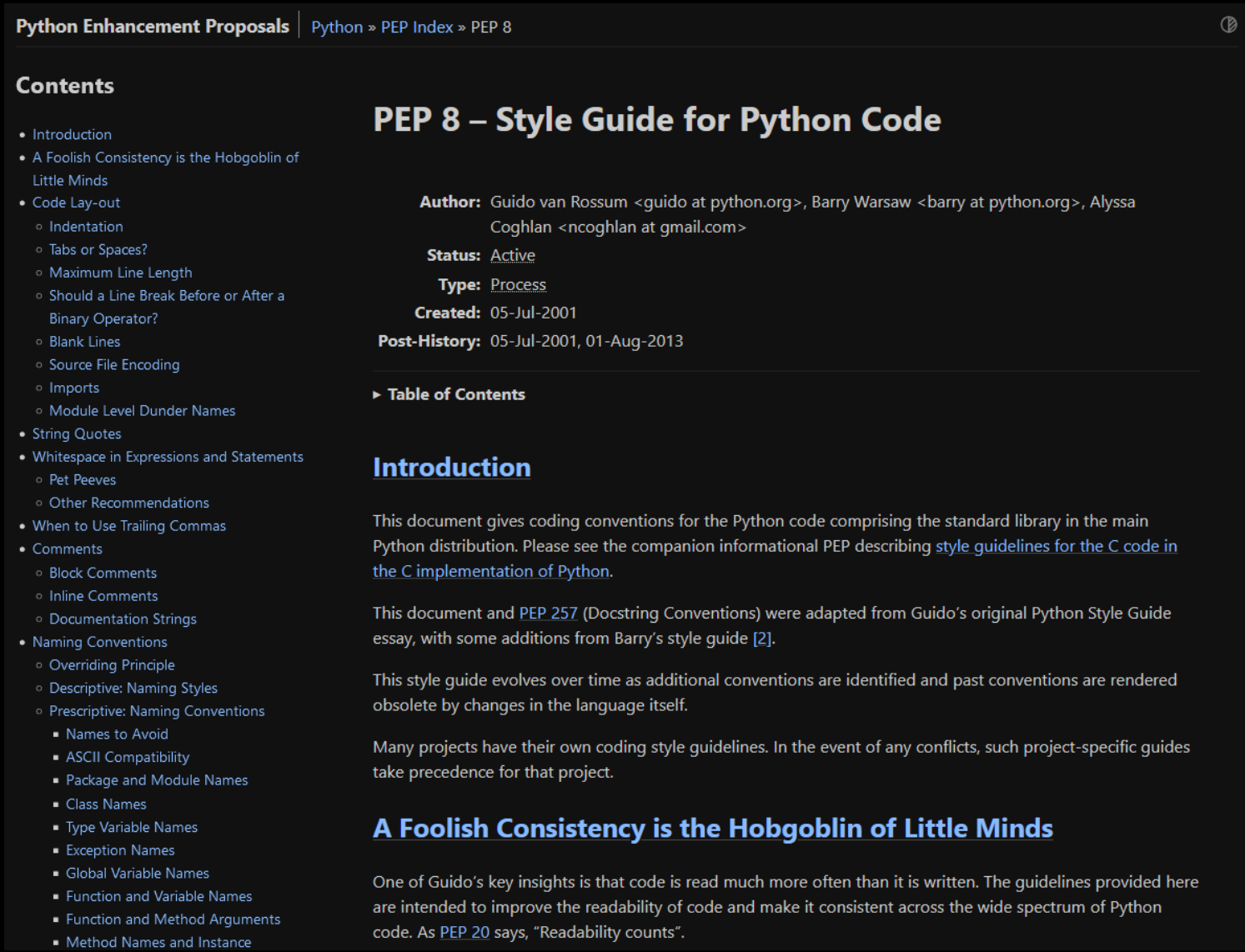
- Python: Follows the PEP8 style guide
  - snake_case for variables and function names
  - PascalCase for class names
  - SCREAMING_SNAKE_CASE for constants
- Java:
  - camelCase for variables and methods
  - PascalCase for class names
  - SCREAMING_SNAKE_CASE for constants
- We will see why this matters later

# WHAT IS THE PEP8 STYLE GUIDE?

PEP8 is the **Python Enhancement Proposal 8**, which provides a set of style guidelines and best practices for writing Python code. It is widely accepted by the Python community and aims to ensure that Python code is readable, consistent, and easy to maintain. https://peps.python.org/pep-0008/

Key Aspects of PEP8:

- **Indentation**
  - Use 4 spaces per indentation level
  - Do not use tabs for indentation
- **Maximum Line Length**
  - Limit lines to a maximum of 79 characters for code
  - For comments or docstrings, the limit is 72 characters
- **Blank Lines**
  - Use two blank lines to separate top-level function and class definitions
  - Use one blank line to separate methods within a class or functions within a module.



Python Enhancement Proposals | Python » PEP Index » PEP 8

Contents



## PEP 8 – Style Guide for Python Code


**Author:** Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>



**Status:** Active
**Type:** Process
**Created:** 05-Jul-2001
**Post-History:** 05-Jul-2001, 01-Aug-2013


▸ Table of Contents

### Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python.

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

### A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".

# WHAT IS PYLINT?

- Pylint is a Python static code analysis tool that checks your code for errors, enforces a coding standard, and looks for code smells (areas that may not necessarily be wrong but could be improved). Pylint helps improve the quality and consistency of Python code by flagging potential issues and suggesting improvements based on style guidelines

# HOW DO I INSTALL PYLINT?

You need to ensure you have Python installed:

```
PS C:\WINDOWS\system32> py --version
Python 3.9.6
```

You will also need to install pip (py –m ensurepip –default-pip)

```
PS C:\WINDOWS\system32>
```

# HOW DO I INSTALL PYLINT? (CONT.)

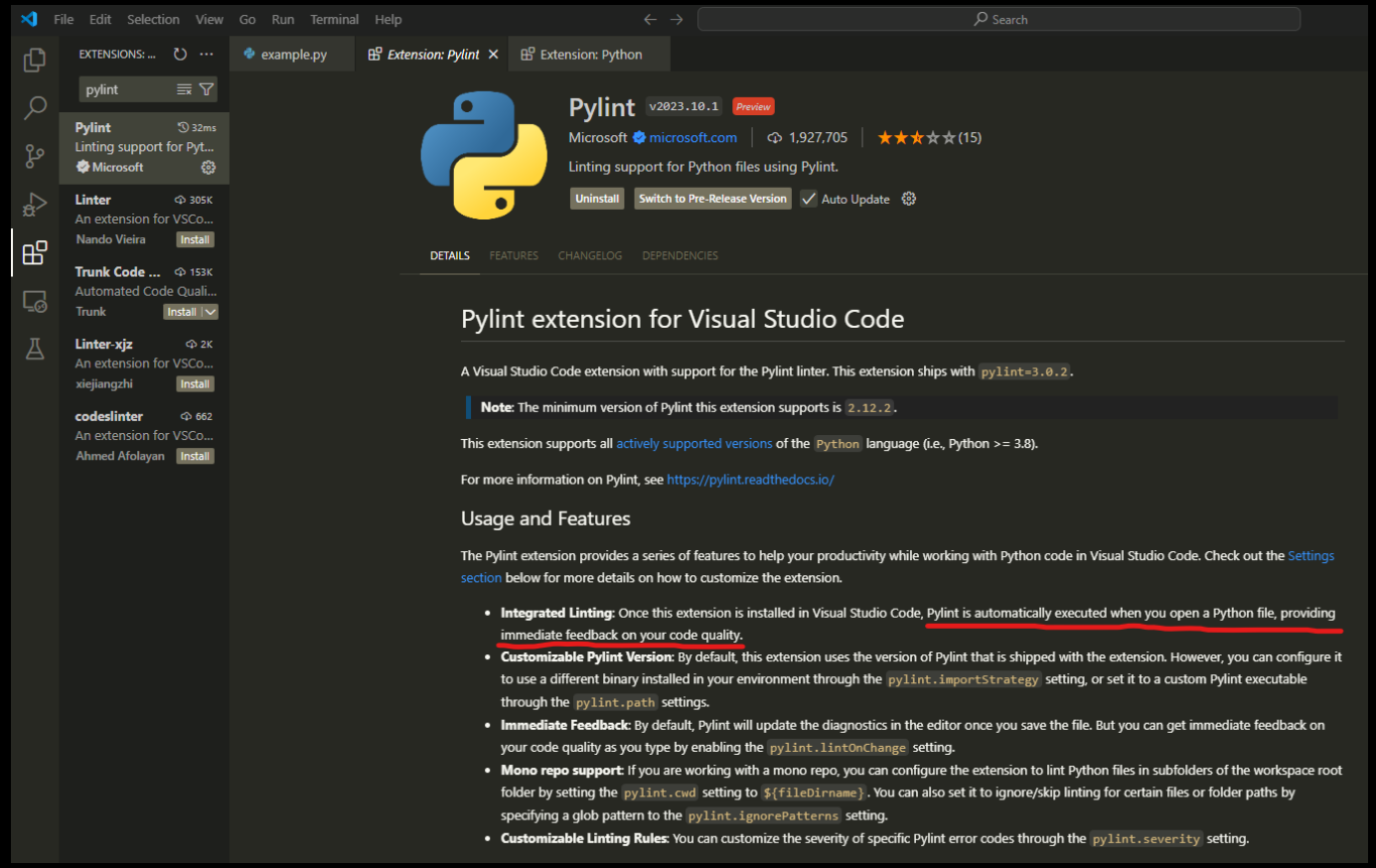
- Now, we install Pylint with pip install pylint'

```
PS C:\WINDOWS\system32> py -m pip --version
pip 21.1.3 from C:\Users\rambe\AppData\Local\Programs\Python\Python39\lib\site-
packages\pip (python 3.9)
PS C:\WINDOWS\system32>
```

# HOW DO I INSTALL PYLINT? (ALT.)

Some IDE's have a plugin/extension for Pylint. The extension for Visual Studio Code automatically checks your code while working in a Python file.

We'll see this in action later



# EXAMPLE CODE

example.py

C: > Users > rambe > Documents > _sweng > workshop > example.py >

```python
def add_numbers(a, b):
    return a + b

result = add_numbers(10, 20)
print(result)
```

```
PS C:\Users\rambe\Documents\_sweng\workshop>
```

Let's see how our code scores using `pylint example.py`

```
PS C:\Users\rambe\Documents\_sweng\workshop>
```

# YIKES, A 2.5/10

```
PS C:\Users\rambe\Documents\_sweng\workshop> pylint example.py
************* Module example
example.py:1:0: C0114: Missing module docstring (missing-module-docstring)
example.py:1:0: C0116: Missing function or method docstring (missing-function-d
ocstring)
example.py:4:0: C0103: Constant name "result" doesn't conform to UPPER_CASE nam
ing style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 2.50/10 (previous run: 2.50/10, +0.00)
```

But what do the warnings mean?
https://pylint.readthedocs.io/en/latest/user_guide/messages/index.html

# C0114: MISSING MODULE DOCSTRING (MISSING-MODULE-DOCSTRING)

https://pylint.readthedocs.io/en/latest/user_guide/messages/index.html

Home | GitLab | Gmail | Big-O Algorithm Com... | Web Terminal

Pylint 4.0.0-dev0 documentation

Search

Tutorial

USER GUIDE

Installation

Usage

Messages

Messages overview

Messages control

Configuration

## Messages

### Messages categories

Pylint can emit various messages. These are categorized according to categories corresponding to bit-encoded exit codes:

- Fatal (1)
- Error (2)
- Warning (4)
- Convention (8)
- Refactor (16)
- Information (NA)

An overview of these messages can be found in Messages overview

The 'C' tells us it's a Convention warning, so we'll go look in there

# C0114: MISSING MODULE DOCSTRING (MISSING-MODULE-DOCSTRING)

Pylint 4.0.0-dev0 documentation

Search

Tutorial

USER GUIDE

Installation

Usage

Messages

Messages overview

astroid-error / F0002

config-parse-error / F0011

fatal / F0001

method-check-failed / F0202

parse-error / F0010

old-import-error / F0401

abstract-class-instantiated / E0110

- consider-using-dict-items / C0206
- consider-using-enumerate / C0200
- consider-using-f-string / C0209
- dict-init-mutate / C3401
- disallowed-name / C0104
- docstring-first-line-empty / C0199
- empty-docstring / C0112
- import-outside-toplevel / C0415
- import-private-name / C2701
- invalid-characters-in-docstring / C0403
- invalid-name / C0103
- line-too-long / C0301
- misplaced-comparison-constant / C2201
- missing-class-docstring / C0115
- missing-final-newline / C0304
- missing-function-docstring / C0116
- missing-module-docstring / C0114
- mixed-line-endings / C0327
- multiple-imports / C0410
- multiple-statements / C0321
- non-ascii-module-import / C2403
- non-ascii-name / C2401
- single-string-used-for-slots / C0205
- singleton-comparison / C0121

## missing-module-docstring / C0114

**Message emitted:**

`Missing module docstring`

**Description:**

*Used when a module has no docstring. Empty modules do not require a docstring.*

**Problematic code:**

```
import sys  # [missing-module-docstring]


def print_python_version():
    print(sys.version)
```

**Correct code:**

```
"""Module providing a function printing python version."""

import sys


def print_python_version():
    print(sys.version)
```

Created by the basic checker.

So we're just missing a docstring explaining what the function does. Let's go add that

# CLEANING UP THE CODE

example.py

C: > Users > rambe > Documents > _sweng > workshop > example.py > ...

```python
"""This module demonstrates a simple function for adding two numbers."""
def add_numbers(a, b):
    return a + b

result = add_numbers(10, 20)
print(result)
```

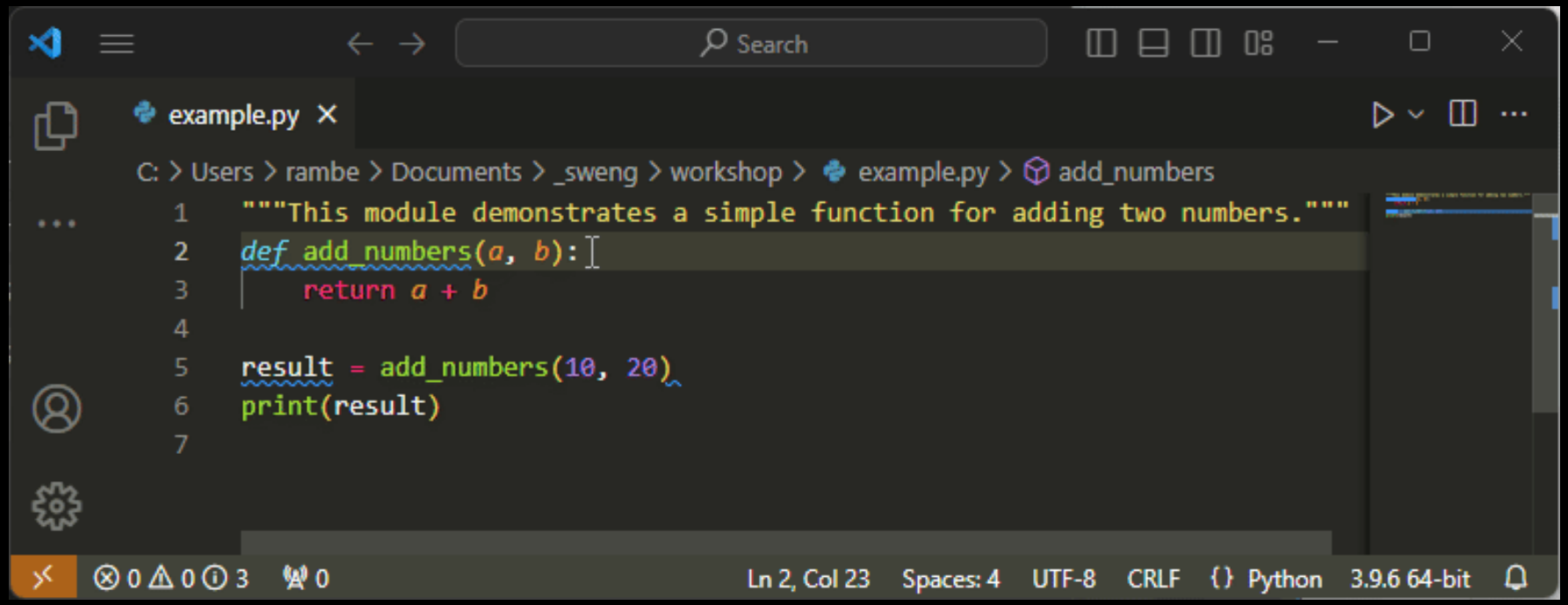
- A docstring is a special type of comment in Python that is used to document functions,classes, and modules.
- It provides a description of what the function, class, or module does, and is enclosed in triple quotes ('''docstring''')
- Given that we got a 2.5/10 last time, what do you think we'll score this run by adding this docstring?

# CLEANING UP THE CODE

- We actually jumped up to a 5/10!
- This was a +2.50 increase from the last run, and pylint will even tell you what your previous score was.
- We still have some issues, but this time we will use the extension in Visual Studio Code

```
PS C:\Users\rambe\Documents\_sweng\workshop> pylint example.py
************* Module example
example.py:3:0: C0116: Missing function or method docstring (missing-function-docstring)
example.py:6:0: C0103: Constant name "result" doesn't conform to UPPER_CASE naming style
(invalid-name)

------------------------------------------------------------------
Your code has been rated at 5.00/10 (previous run: 2.50/10, +2.50)
```

# CLEANING UP THE CODE




- You'll see the blue squiggly line under `def add_numbers` If you mouse over it, you can click on the blue hyperlink and it'll take you to the page that describes the warning.

# CLEANING UP THE CODE

Another docstring issue. We need to go and specify what the function 'add_numbers' does

```
def add_numbers(a, b):
    return a + b
```

Pylint 4.0.0-dev0 documentation

Search

Tutorial

USER GUIDE

Installation

Usage

Messages

Messages overview

astroid-error / F0002

config-parse-error / F0011

fatal / F0001

method-check-failed / F0202

parse-error / F0010

old-import-error / F0401

abstract-class-instantiated / E0110

access-member-before-definition / E0203

assigning-non-slot / E0237

## missing-function-docstring / C0116

**Message emitted:**

`Missing function or method docstring`

**Description:**

*Used when a function or method has no docstring. Some special methods like __init__ do not require a docstring.*

**Problematic code:**

```
import sys


def print_python_version():  # [missing-function-docstring]
    print(sys.version)
```

**Correct code:**

```
import sys


def print_python_version():
    """Function printing python version."""
    print(sys.version)
```

Created by the basic checker.

# CLEANING UP THE CODE

C: > Users > rambe > Documents > _sweng > workshop > example.py > add_numbers

```python
"""This module demonstrates a simple function for adding two numbers."""
def add_numbers(a, b):
    """
    Adds two numbers and returns the result.

    Parameters:
    a (int): The first number.
    b (int): The second number.

    Returns:
    int: The sum of the two numbers.
    """
    return a + b

result = add_numbers(10, 20)
print(result)
```

- Let's see how many points we gained by adding some more documentation.

# CLEANING UP THE CODE

```
example.py:1:0: C0116: Missing function or method docstring (missing-function-docstring)
example.py:4:0: C0103: Constant name "result" doesn't conform to UPPER_CASE naming style
(invalid-name)

------------------------------------------------------------------
Your code has been rated at 2.50/10 (previous run: 2.50/10, +0.00)

PS C:\Users\rambe\Documents\_sweng\workshop> pylint example.py
************* Module example
example.py:3:0: C0116: Missing function or method docstring (missing-function-docstring)
example.py:6:0: C0103: Constant name "result" doesn't conform to UPPER_CASE naming style
(invalid-name)

------------------------------------------------------------------
Your code has been rated at 5.00/10 (previous run: 2.50/10, +2.50)

PS C:\Users\rambe\Documents\_sweng\workshop>
```

- We increased our score by another +2.5! But we still have one error left.

# CLEANING UP THE CODE

```
************* Module example
example.py:15:0: C0103: Constant name "result" doesn't conform to UPPER_CASE naming style
 (invalid-name)

------------------------------------------------------------------
Your code has been rated at 7.50/10 (previous run: 5.00/10, +2.50)
```

Let's say we don't see the need for the variable name to be 'RESULT' or 'ADDED_NUMBERS'
We can use '# pylint: disable=invalid-name' behind that line to ignore that warning.

example.py ×

C: > Users > rambe > Documents > _sweng > workshop > example.py > ...

```python
"""This module demonstrates a simple function for adding two numbers."""
def add_numbers(a, b):
    """
    Adds two numbers and returns the result.

    Parameters:
    a (int): The first number.
    b (int): The second number.

    Returns:
    int: The sum of the two numbers.
    """
    return a + b

result = add_numbers(10, 20) # pylint: disable=invalid-name
print(result)
```

# CLEANING UP THE CODE

```
PS C:\Users\rambe\Documents\_sweng\workshop>
```

We did it!

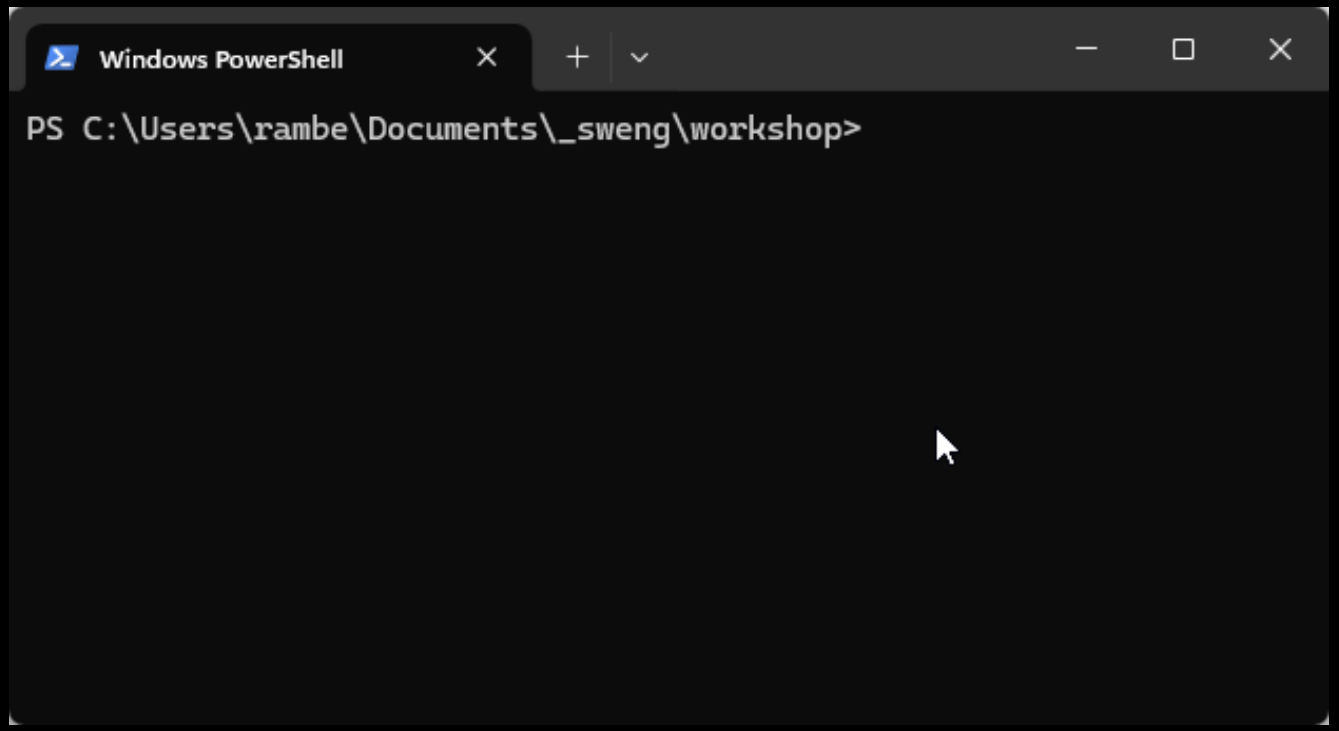
# CLEANED UP THE CODE

- We managed to score a perfect 10/10!

- And even though this is a very small example, this scales across larger projects that involve multiple developers.
- Consistency in coding style is critical to avoid confusion and document the intend purpose of each class and function.

```
PS C:\Users\rambe\Documents\_sweng\workshop> pylint example.py

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 7.50/10, +2.50)
```

# MORE ON DOCSTRING




- We can use the help() function in Python to return the docstrings for our function.
- Start by running 'python' in the terminal
- Then we need to import our module and pass our function into the help() function

# MORE ON DOCSTRING

```
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 b
Type "help", "copyright", "credits" or "license" for more information.
>>> from example import add_numbers
30
>>> help(add_numbers)
Help on function add_numbers in module example:

add_numbers(a, b)
    Adds two numbers and returns the result.

    Parameters:
    a (int): The first number.
    b (int): The second number.

    Returns:
    int: The sum of the two numbers.

>>> |
```

- This displays the docstring we defined for the add_numbers function.
- It shows its description, parameters, and return values.

# PRETTIER

# What is Prettier?

* An opinionated code formatter
* Supports many languages
* Integrates with most editors

- Prettier is a powerful opinionated code formatter that ensures a consistent code style across  entire codebase, improving readability and minimizing formatting-related merge conflicts. It takes care of formatting  code automatically, regardless of the original style, which allows developers to focus more on writing the logic of the code rather than worrying about styling inconsistencies.

# PRETTIER IS SUPPORT FOR:

- It works with multiple languages,  JavaScript, Python, HTML, a CSS , JSON, GraphQL, TypeScript, Markdown, YAML

JavaScript
JSX
Flow
TypeScript
JSON

HTML
Vue
Angular
Ember / Handlebars

CSS
Less
SCSS
styled-components
styled-jsx

GraphQL
GraphQL Schemas

Markdown
CommonMark
GitHub-Flavored Markdown
MDX v1

YAML

**Enforces Consistent Style**: Prettier automatically formats code to a single style, helping teams maintain uniformity.

**Readability Improvement**: Prettier makes code easier to read by consistently applying rules to format things like indentation, spacing, and line breaks.

**Reduces Merge Conflicts**: By maintaining the same formatting across the entire codebase, Prettier significantly reduces conflicts caused by different developers using varying code styles.

# Why?

* Your code is formatted on save
* No need to discuss style in code review
* Saves you time and energy

# PRETTIER TAKES CODE AND REPRINTS

- Take the example of the following code : Bunch of parameter in single line

JS foo(reallyLongArg(), omgSoManyParameters Untitled-1

```
1  foo(reallyLongArg(), omgSoManyParameters(), IShouldRefactorThis(), isThereSeriouslyAnotherOne(), IDKifthereifthereismore(),letmecheck());
```

- Prettier is going to do the painstaking work of reprinting and new code looks like:

JS foo(reallyLongArg(), Untitled-1

```
1  foo(reallyLongArg(),
2      omgSoManyParameters(),
3      IShouldRefactorThis(),
4      isThereSeriouslyAnotherOne(),
5      IDKifthereifthereismore(),
6      letmecheck());
```

# OTHER REASONS TO CHOOSE PRETTIER

**1. Building and Enforcing a Style Guide**:

- **Unified Code Style**: Prettier automatically enforces a consistent style guide across the project, ensuring that everyone follows the same conventions.
- **Avoid Nitpicking**: Code reviews can focus on logic and functionality rather than style. This reduces disagreements about formatting, as Prettier eliminates the need for "nitpicky" comments.

**2. Helping Newcomers**:

- **Reducing Syntax Errors**: For beginners, getting the syntax right can be challenging. Prettier helps newcomers by automatically correcting formatting issues, enabling them to focus on learning concepts and logic instead of being stuck on minor mistakes.
- **Boosts Confidence**: As Prettier helps clean up the code instantly, newcomers can better understand how well-written code looks, building their confidence.

# OTHER REASONS CONT…

**3. Writing Code Efficiently**:

- **Saves Time**: Prettier can save up to 5% of the time spent formatting, which can instead be used for writing more complex logic or even taking breaks to maintain productivity.
- **Single Command**: With a single shortcut or command, developers can format an entire codebase, allowing them to focus on the content of the code rather than worrying about style inconsistencies.

**4. Easy to Adopt**:

- **Seamless Integration**: Prettier works well with most editors and IDEs like VS Code, Atom, and Sublime, and it's simple to configure.
- **Minimal Setup**: Just a quick installation, and you can use Prettier immediately. It's designed to be easy to add to any new or existing project.

**5. Clean Up an Existing Codebase**:

- **Instant Cleanup**: Prettier can instantly reformat an entire legacy codebase, making it easier to maintain and understand.
- **Consistent Formatting Across Files**: No more old formatting styles mixed with new ones—Prettier ensures every file looks the same, regardless of when it was written.

# WHAT DO YOU THINK?

- Prettier is one of the few tools that fully automates enforcing a style guide. While it may not format every single line exactly the way we might prefer, don't you think the unique benefits, like time savings and consistency, make it worth using?

# EDITOR THAT SUPPORT PRETTIER:

**Atom**
prettier-atom
mprettier
miniprettier

**Emacs**
prettier-js
prettier.el
Apheleia

**Espresso**
espresso-prettier

**Nova**
Prettier

**Sublime Text**
JsPrettier

**Vim**
vim-prettier
neoformat
ALE
coc-prettier

**Visual Studio**
JavaScriptPrettier

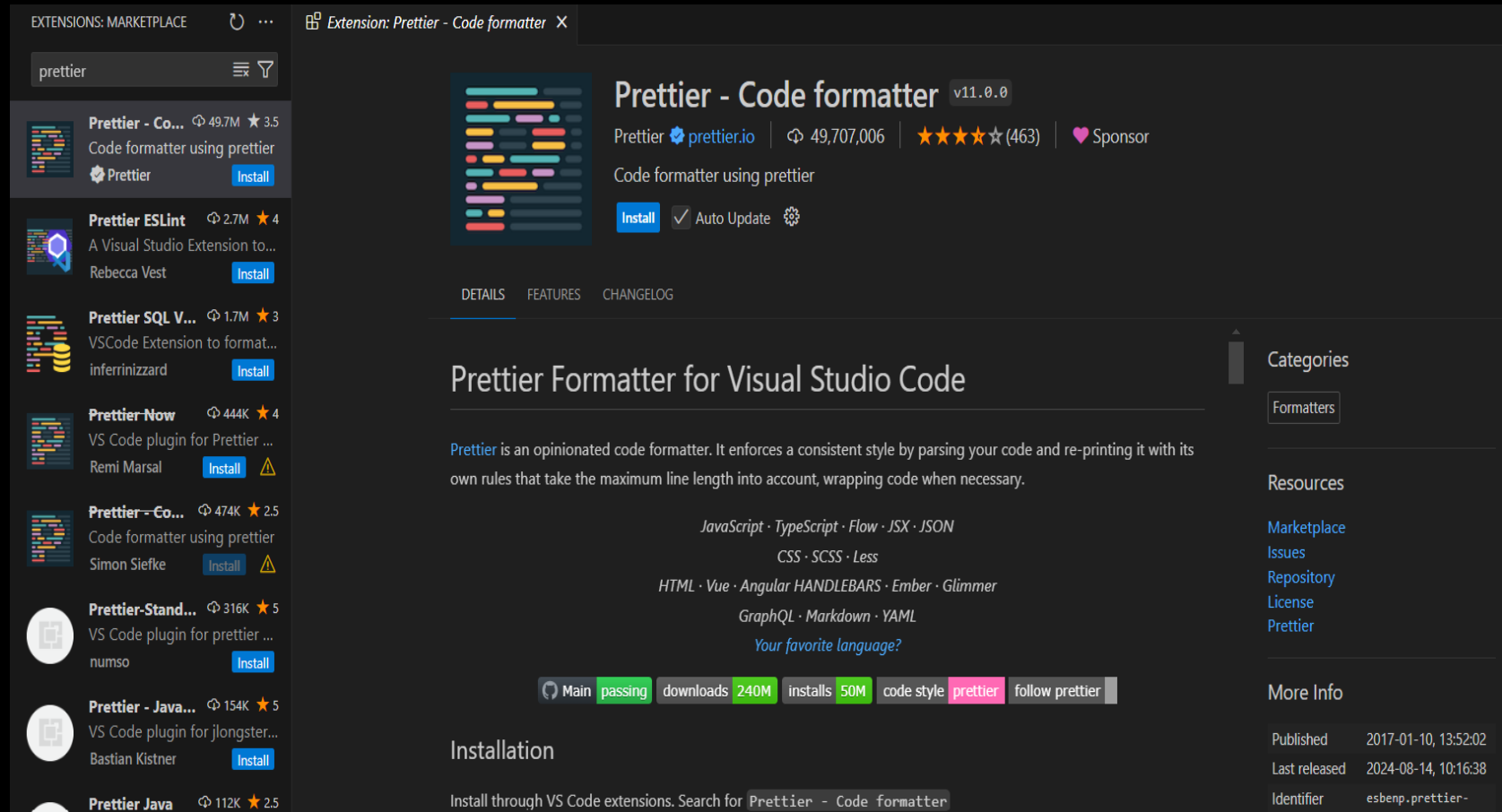
**VS Code**
prettier-vscode

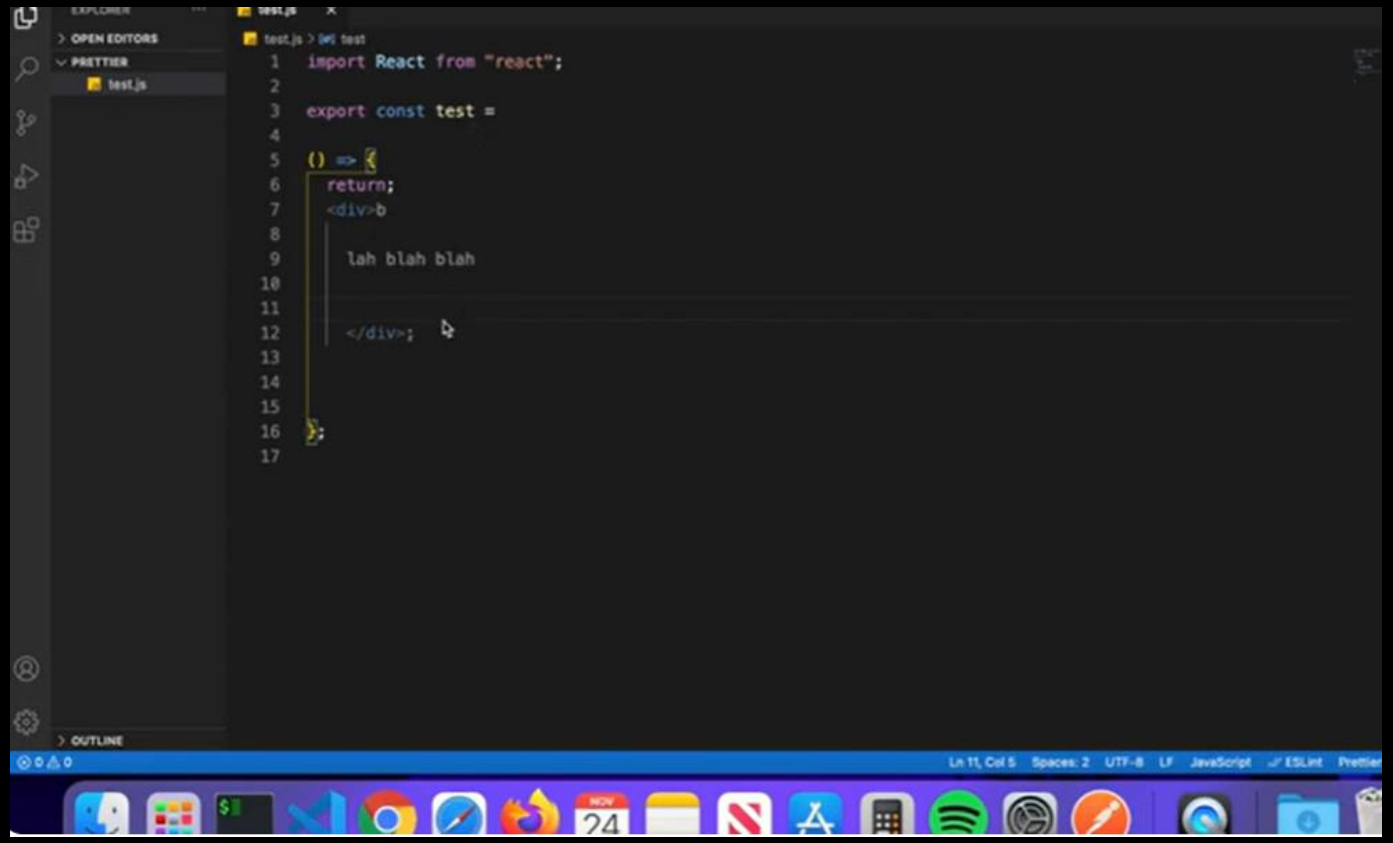
**WebStorm**
Built-in support

# INSTALLING PRETTIER

- Prettier · Opinionated Code Formatter
- 

How to install Prettier in VS code.




OPEN EDITORS
PRETTIER
test.js
OUTLINE
import React from "react";
export const test =
() => {
return;
b
lah blah blah
;
};
Ln 11, Col 5
Spaces: 2
UTF-8
LF
JavaScript
ESLint
Prettier

# Q&A TIME

Any questions?