

CODE QUALITY

I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code



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](#)

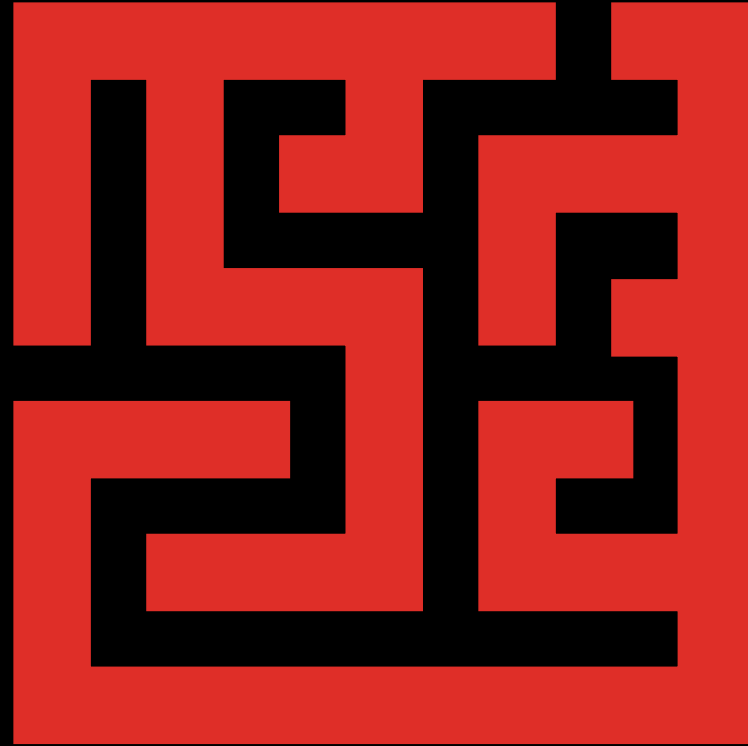


BAD CODE EXAMPLE

```
2
3 def a(x, y):
4     return x + y
5
6 def s(x, y):
7     return x - y
8
9 def m(x, y):
10    return x * y
11
12 def d(x, y):
13     if y == 0:
14         return "Error"
15     return x / y
16
17 def c():
18     print("Simple Calc")
19     while True:
20         c = input("cmd: ").lower()
21         if c == "exit":
22             break
23         if c not in ["add", "sub", "mul", "div"]:
24             print("Bad cmd")
25             continue
26         try:
27             n1 = float(input("n1: "))
28             n2 = float(input("n2: "))
29         except:
30             print("Bad input")
31             continue
32         if c == "add":
33             r = a(n1, n2)
34         elif c == "sub":
35             r = s(n1, n2)
36         elif c == "mul":
37             r = m(n1, n2)
38         elif c == "div":
39             r = d(n1, n2)
40         print(f"Result: {r}")
41
42 if __name__ == "__main__":
43     c()
44
```

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

IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.



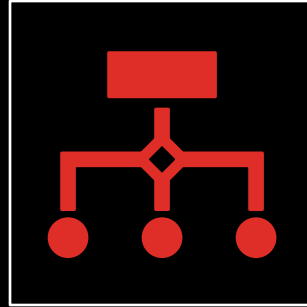
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.



OKAY, I'LL READ A STYLE GUIDE.



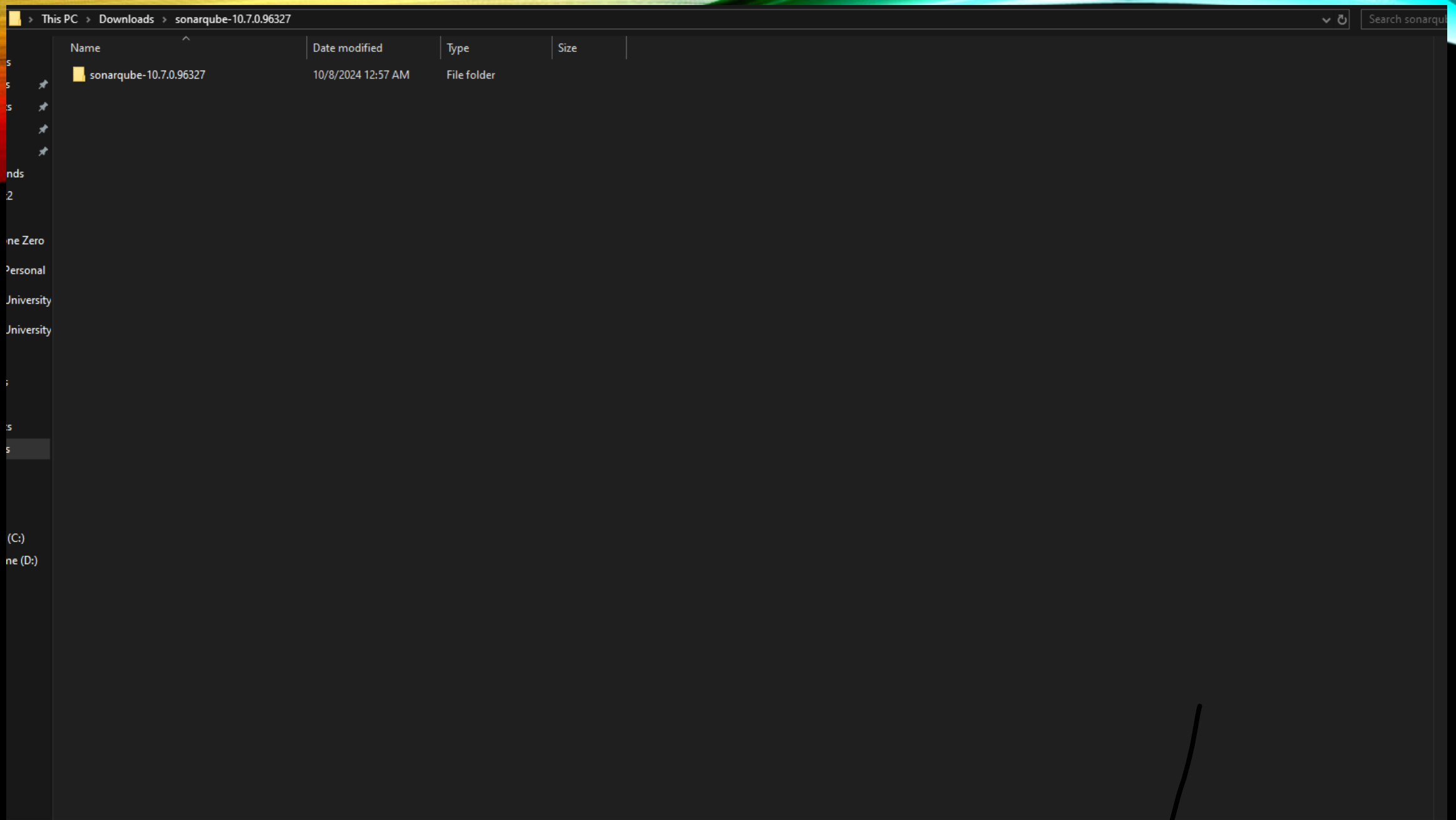
SonarQube is an effective static code analysis tool that examines code for bugs, vulnerabilities, code smells, and adherence to coding standards. It supports various programming languages and offers comprehensive insights into code quality and maintainability.



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](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

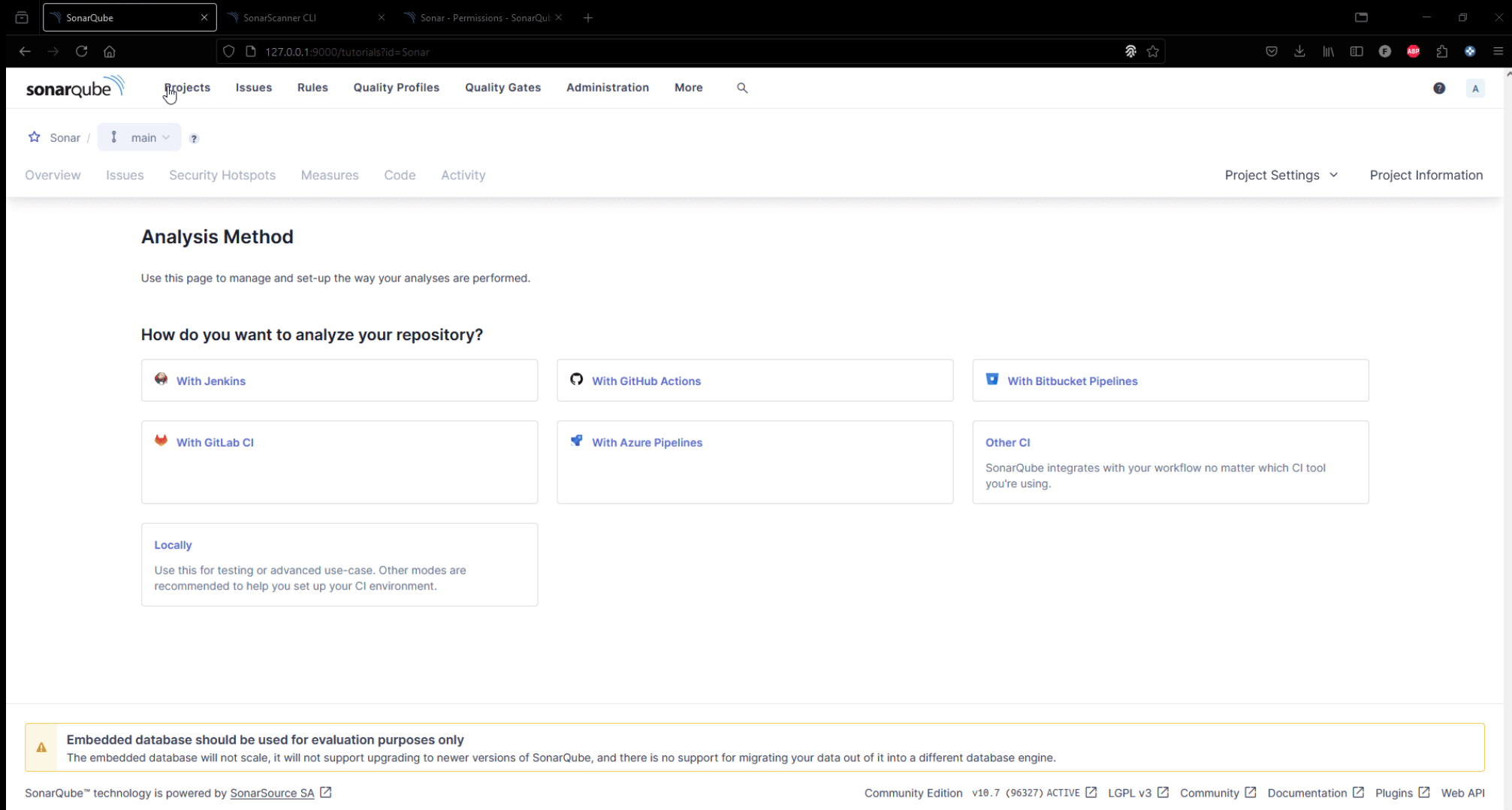


RUNNING CODE ANALYSIS

<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



The screenshot shows the SonarQube web interface. The browser tabs include 'SonarQube', 'SonarScanner CLI', and 'Sonar - Permissions - SonarQube'. The address bar shows '127.0.0.1:9000/tutorials?id=Sonar'. The SonarQube logo is in the top left, and the 'Projects' menu item is highlighted. The breadcrumb trail shows 'Sonar / main'. The navigation bar includes 'Overview', 'Issues', 'Security Hotspots', 'Measures', 'Code', 'Activity', 'Project Settings', and 'Project Information'. The main content area is titled 'Analysis Method' and contains the text 'Use this page to manage and set-up the way your analyses are performed.' Below this is the section 'How do you want to analyze your repository?' with several options: 'With Jenkins', 'With GitHub Actions', 'With Bitbucket Pipelines', 'With GitLab CI', 'With Azure Pipelines', and 'Other CI'. The 'Other CI' option includes the text 'SonarQube integrates with your workflow no matter which CI tool you're using.' At the bottom, there is a warning message: 'Embedded database should be used for evaluation purposes only. The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.' The footer shows 'SonarQube™ technology is powered by SonarSource SA' and 'Community Edition v10.7 (96327) ACTIVE'.

Analysis Method

Use this page to manage and set-up the way your analyses are performed.

How do you want to analyze your repository?

- With Jenkins**
- With GitHub Actions**
- With Bitbucket Pipelines**
- With GitLab CI**
- With Azure Pipelines**
- Other CI**
SonarQube integrates with your workflow no matter which CI tool you're using.
- Locally**
Use this for testing or advanced use-case. Other modes are recommended to help you set up your CI environment.

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by [SonarSource SA](#)

Community Edition v10.7 (96327) ACTIVE [LGPL v3](#) [Community](#) [Documentation](#) [Plugins](#) [Web API](#)

INTEGRATING WITH CI/CD:

Update build.yml · msan37/Sonar

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

140%

msan37 / Sonar

Type to search

CodeIssuesPull requestsActionsProjectsWikiSecurityInsightsSettings

You have successfully requested the workflow to be canceled.

Java CI with Maven

Update build.yml #5

Re-run jobsLatest #2

Summary

Jobs

build

Run detailsUsageWorkflow file

Re-run all jobs

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

build

Enable debug logging

CancelRe-running...

Set up JDK 110s

Build with Maven1s

Post Set up JDK 110s

Post Run actions/checkout@v40s

Complete job0s

Search logs

2s1s0s1s0s0s0s

GOOD CODE EXAMPLE

```
3 def add(x, y):
4     """Adds two numbers."""
5     return x + y
6
7 def subtract(x, y):
8     """Subtracts two numbers."""
9     return x - y
10
11 def multiply(x, y):
12     """Multiplies two numbers."""
13     return x * y
14
15 def divide(x, y):
16     """Divides two numbers. Checks for division by zero."""
17     if y == 0:
18         return "Error: Cannot divide by zero."
19     return x / y
20
21 def calculator():
22     """Runs the command-line calculator."""
23     print("Simple Calculator App")
24     print("Available commands: add, subtract, multiply, divide, exit")
25
26     while True:
27         # Get the user input
28         command = input("Enter command (add/subtract/multiply/divide/exit): ").lower()
29
30         if command == "exit":
31             print("Exiting calculator. Goodbye!")
32             break
33
34         if command not in ["add", "subtract", "multiply", "divide"]:
35             print("Invalid command. Please try again.")
36             continue
37
38         try:
39             # Get the numbers from the user
40             num1 = float(input("Enter first number: "))
41             num2 = float(input("Enter second number: "))
42         except ValueError:
43             print("Invalid input. Please enter numeric values.")
44             continue
45
46         # Perform the calculation
47         if command == "add":
48             result = add(num1, num2)
49         elif command == "subtract":
50             result = subtract(num1, num2)
51         elif command == "multiply":
52             result = multiply(num1, num2)
53         elif command == "divide":
54             result = divide(num1, num2)
55
56         print(f"Result: {result}")
57
58 if __name__ == "__main__":
59     calculator()
60
```

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 etiquette)

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.

Contents

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code Lay-out
 - Indentation
 - Tabs or Spaces?
 - Maximum Line Length
 - Should a Line Break Before or After a Binary Operator?
 - Blank Lines
 - Source File Encoding
 - Imports
 - Module Level Dunder Names
- String Quotes
- Whitespace in Expressions and Statements
 - Pet Peeves
 - Other Recommendations
- When to Use Trailing Commas
- Comments
 - Block Comments
 - Inline Comments
 - Documentation Strings
- Naming Conventions
 - Overriding Principle
 - Descriptive: Naming Styles
 - Prescriptive: Naming Conventions
 - Names to Avoid
 - ASCII Compatibility
 - Package and Module Names
 - Class Names
 - Type Variable Names
 - Exception Names
 - Global Variable Names
 - Function and Variable Names
 - Function and Method Arguments
 - Method Names and Instance

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.)

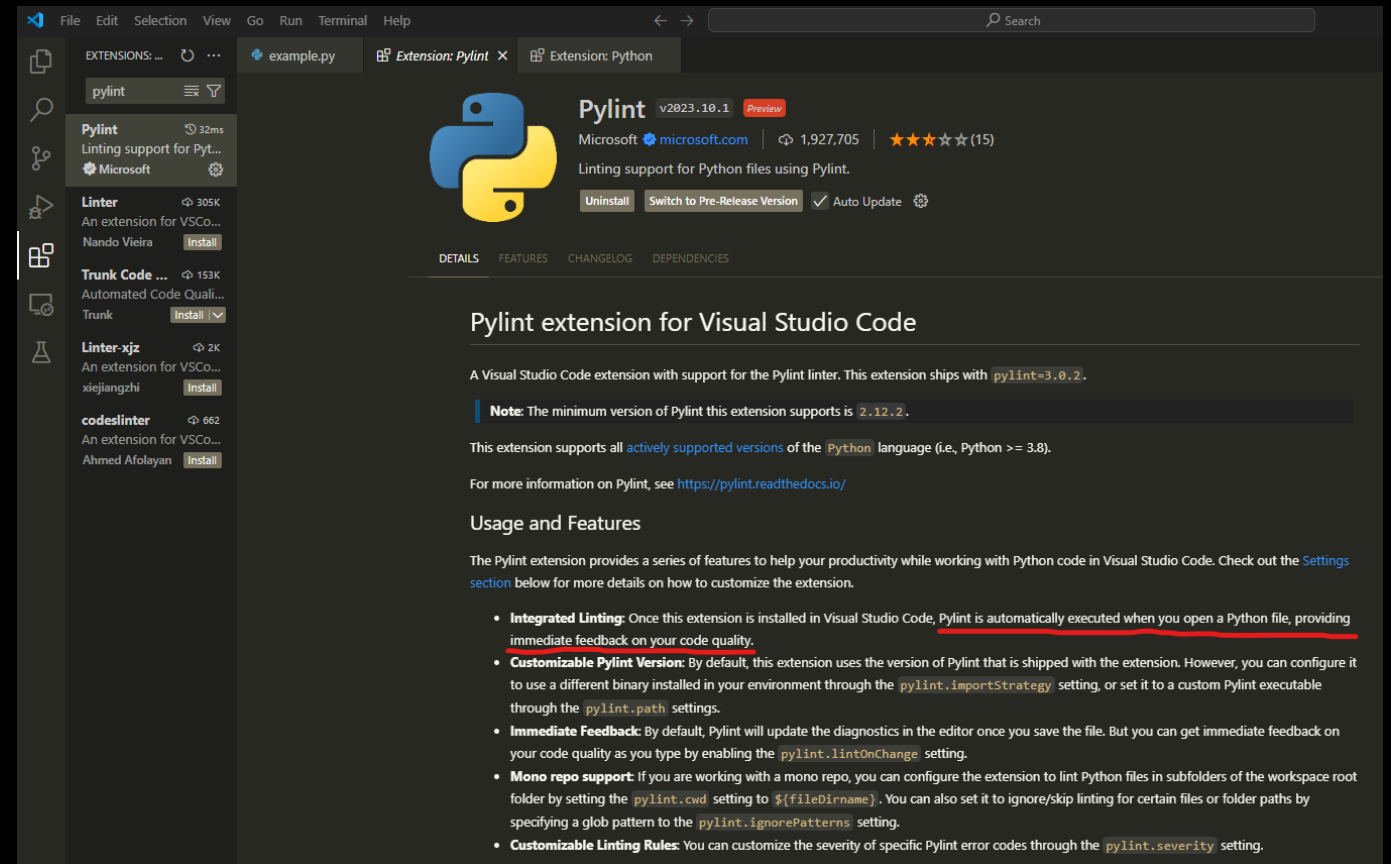
- Now, we install Pylint with `pip install pylint`

```
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



The screenshot shows the Visual Studio Code interface. On the left, the 'EXTENSIONS' sidebar is open, displaying a list of extensions. 'Pylint' by Microsoft is highlighted, showing its icon, name, version (v2023.10.1), and a red 'Preview' badge. Below it, other extensions like 'Linter', 'Trunk Code ...', 'Linter-xjz', and 'codeslinter' are listed. The main editor area displays the 'Pylint' extension page. It features the Python logo, the extension name 'Pylint', version 'v2023.10.1', and a red 'Preview' badge. It also shows the publisher 'Microsoft', a link to 'microsoft.com', and statistics: 1,927,705 downloads and 15 stars. The description states 'Linting support for Python files using Pylint.' and includes buttons for 'Uninstall', 'Switch to Pre-Release Version', and 'Auto Update'. Below this, the 'DETAILS' tab is selected, showing the title 'Pylint extension for Visual Studio Code'. The description states: 'A Visual Studio Code extension with support for the Pylint linter. This extension ships with `pylint=3.0.2`.' A note specifies: 'Note: The minimum version of Pylint this extension supports is 2.12.2.' It also mentions: 'This extension supports all actively supported versions of the Python language (i.e., Python >= 3.8).' and provides a link for more information: 'https://pylint.readthedocs.io/'. The 'Usage and Features' section lists several features: 'Integrated Linting' (Pylint is automatically executed when you open a Python file, providing immediate feedback on your code quality), 'Customizable Pylint Version' (By default, this extension uses the version of Pylint that is shipped with the extension. However, you can configure it to use a different binary installed in your environment through the `pylint.importStrategy` setting, or set it to a custom Pylint executable through the `pylint.path` settings), 'Immediate Feedback' (By default, Pylint will update the diagnostics in the editor once you save the file. But you can get immediate feedback on your code quality as you type by enabling the `pylint.lintOnChange` setting), 'Mono repo support' (If you are working with a mono repo, you can configure the extension to lint Python files in subfolders of the workspace root folder by setting the `pylint.cwd` setting to `${fileDirname}`. You can also set it to ignore/skip linting for certain files or folder paths by specifying a glob pattern to the `pylint.ignorePatterns` setting), and 'Customizable Linting Rules' (You can customize the severity of specific Pylint error codes through the `pylint.severity` setting).

EXAMPLE CODE

example.py X

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

```
1  def add_numbers(a, b):  
2      return a + b  
3  
4  result = add_numbers(10, 20)  
5  print(result)  
6
```

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

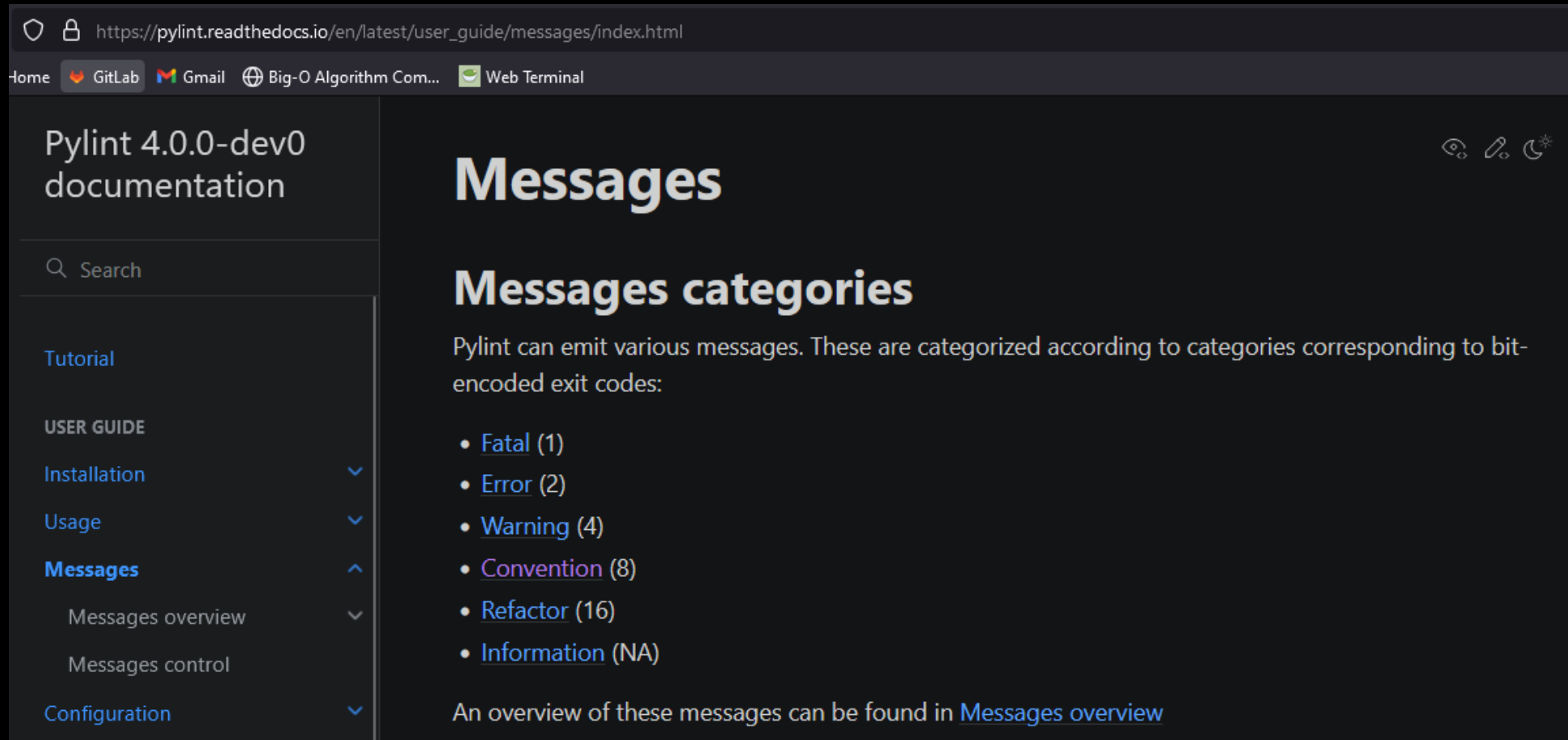
```
PS C:\Users\rambe\Documents\_sweng\workshop> pylint example.py
***** Module example
example.py:1:0: C0114: Missing module docstring (missing-module-docstring)
example.py:1:0: C0116: Missing function or method docstring (missing-function-docstring)
example.py:4:0: C0103: Constant name "result" doesn't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

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

But what do the warnings mean?

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

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



The screenshot shows a web browser window with the URL `https://pylint.readthedocs.io/en/latest/user_guide/messages/index.html`. The browser's address bar and tabs are visible at the top. The page has a dark theme. On the left is a sidebar with the title "Pylint 4.0.0-dev0 documentation". It contains a search bar and a list of navigation links: "Tutorial", "USER GUIDE", "Installation", "Usage", "Messages" (which is highlighted), "Messages overview", "Messages control", and "Configuration". The main content area is titled "Messages" and "Messages categories". It explains that Pylint messages are categorized by bit-encoded exit codes. A list of categories is provided: Fatal (1), Error (2), Warning (4), Convention (8), Refactor (16), and Information (NA). The "Convention" category is highlighted in purple. A link to "Messages overview" is also present.

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

- 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?

```
example.py ●
C: > Users > rambe > Documents > _sweng > workshop > example.py > ...
1  """This module demonstrates a simple function for adding two numbers."""
2  def add_numbers(a, b):
3      return a + b
4
5  result = add_numbers(10, 20)
6  print(result)
7
```

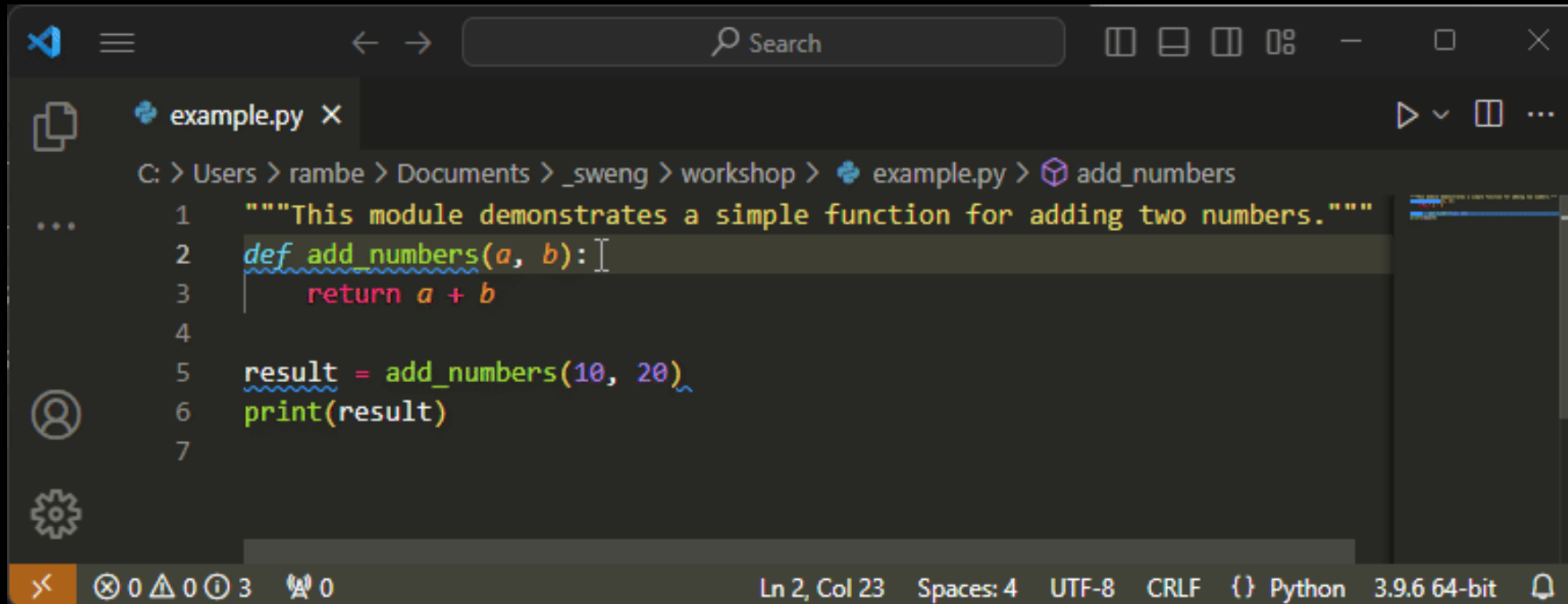
CLEANING UP THE CODE

- 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



```
example.py X
C: > Users > rambe > Documents > _sweng > workshop > example.py > add_numbers
1  """This module demonstrates a simple function for adding two numbers."""
2  def add_numbers(a, b):
3      return a + b
4
5  result = add_numbers(10, 20)
6  print(result)
7

Ln 2, Col 23  Spaces: 4  UTF-8  CRLF  {} Python  3.9.6 64-bit
```

- 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
1  """This module demonstrates a simple function for adding two numbers."""
2  def add_numbers(a, b):
3      """
4      Adds two numbers and returns the result.
5
6      Parameters:
7      a (int): The first number.
8      b (int): The second number.
9
10     Returns:
11     int: The sum of the two numbers.
12     """
13     return a + b
14
15     result = add_numbers(10, 20)
16     print(result)
17
```

- 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 X
C: > Users > rambe > Documents > _sweng > workshop > example.py > ...
1  """This module demonstrates a simple function for adding two numbers."""
2  def add_numbers(a, b):
3      """
4      Adds two numbers and returns the result.
5
6      Parameters:
7      a (int): The first number.
8      b (int): The second number.
9
10     Returns:
11     int: The sum of the two numbers.
12     """
13     return a + b
14
15     result = add_numbers(10, 20) # pylint: disable=invalid-name
16     print(result)
17
```

CLEANING UP THE CODE

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

We did
it!

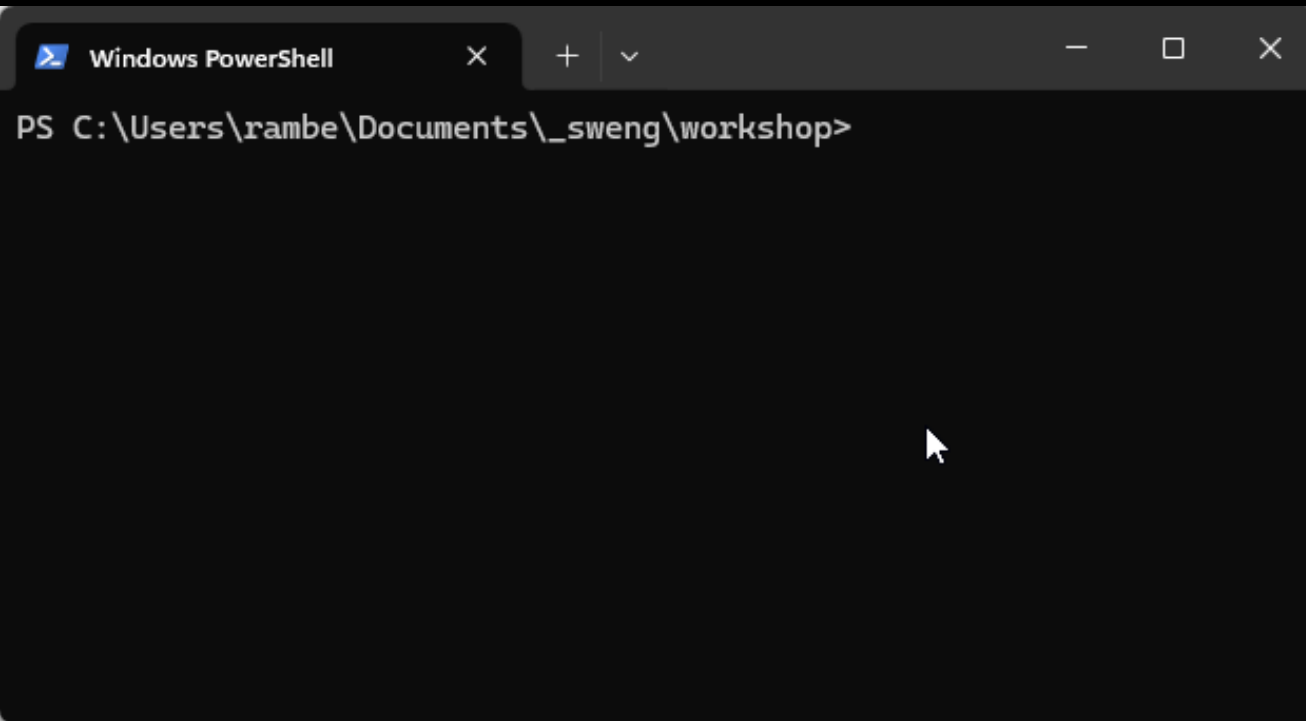
CLEANED UP THE CODE

- 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

A screenshot of a Windows PowerShell terminal window. The title bar at the top reads "Windows PowerShell" and includes standard window controls (close, maximize, and a dropdown menu). The terminal content shows the prompt "PS C:\Users\rambe\Documents_sweng\workshop>" followed by a mouse cursor pointing at the end of the line.

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

- 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
Aphelia



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.

The screenshot shows the Visual Studio Code Extensions Marketplace interface. On the left, a list of search results for 'Prettier' is displayed. The top result is 'Prettier - Code formatter' by Prettier, with 49.7M downloads and a 3.5-star rating. Below it are other extensions like 'Prettier ESLint', 'Prettier SQL V...', 'Prettier-Now', 'Prettier-Code...', 'Prettier-Stand...', and 'Prettier Java'. The main panel on the right shows the details for the 'Prettier - Code formatter' extension. It includes the extension's icon, name, version (v11.0.0), publisher (Prettier), and a list of supported languages: JavaScript, TypeScript, Flow, JSX, JSON, CSS, SCSS, Less, HTML, Vue, Angular, HANDLEBARS, Ember, Glimmer, GraphQL, Markdown, and YAML. The extension is marked as a 'Sponsor' and has an 'Auto Update' checkbox. The 'Installation' section at the bottom states: 'Install through VS Code extensions. Search for Prettier - Code formatter'.

EXTENSIONS: MARKETPLACE

prettier

Prettier - Co... 49.7M ★ 3.5
Code formatter using prettier
Prettier [Install](#)

Prettier ESLint 2.7M ★ 4
A Visual Studio Extension to...
Rebecca Vest [Install](#)

Prettier SQL V... 1.7M ★ 3
VSCode Extension to format...
inferrinizzard [Install](#)

Prettier-Now 444K ★ 4
VS Code plugin for Prettier ...
Remi Marsal [Install](#) ⚠

Prettier-Code... 474K ★ 2.5
Code formatter using prettier
Simon Siefke [Install](#) ⚠

Prettier-Stand... 316K ★ 5
VS Code plugin for prettier ...
numso [Install](#)

Prettier - Java... 154K ★ 5
VS Code plugin for jlongster...
Bastian Kistner [Install](#)

Prettier Java 112K ★ 2.5

Prettier - Code formatter v11.0.0
Prettier [prettier.io](#) | 49,707,006 | ★★★★★ (463) | ❤ Sponsor
Code formatter using prettier
[Install](#) ☒ Auto Update ⚙

DETAILS FEATURES CHANGELOG

Prettier Formatter for Visual Studio Code

Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.

JavaScript · TypeScript · Flow · JSX · JSON
CSS · SCSS · Less
HTML · Vue · Angular HANDLEBARS · Ember · Glimmer
GraphQL · Markdown · YAML
Your favorite language?

Main **passing** downloads **240M** installs **50M** code style **prettier** follow prettier

Installation
Install through VS Code extensions. Search for **Prettier - Code formatter**

Categories
Formatters

Resources
[Marketplace](#)
[Issues](#)
[Repository](#)
[License](#)
[Prettier](#)

More Info
Published 2017-01-10, 13:52:02
Last released 2024-08-14, 10:16:38
Identifier esbnp-prettier-



```
1 import React from "react";
2
3 export const test =
4
5   () => {
6     return;
7     <div>b
8
9     lah blah blah
10
11     </div>;
12
13
14
15
16
17
```

VS Code interface details: Explorer sidebar on the left shows 'test.js' under 'PRETTIER'. The main editor area shows the code. The status bar at the bottom indicates 'Ln 11, Col 5', 'Spaces: 2', 'UTF-8', 'LF', 'JavaScript', 'ESLint', and 'Prettier'. The macOS dock is visible at the bottom with various application icons.



Q&A TIME

Any questions?