Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

Python static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your PYTHON code

All rules 216 | Vulnerability 29 | Bug 55 | Security Hotspot 31 | Code Smell 101

Tags

Search by name...

Functions should not have too many lines of code
Code Smell

Track uses of "NOSONAR" comments
Code Smell

Track comments matching a regular expression
Code Smell

Statements should be on separate lines
Code Smell

Functions should not contain too many return statements
Code Smell

Files should not have too many lines of code
Code Smell

Lines should not be too long
Code Smell

Methods and properties that don't access instance data should be static
Code Smell

New-style classes should be used
Code Smell

Parentheses should not be used after certain keywords
Code Smell

Track "TODO" and "FIXME" comments that do not contain a reference to a person
Code Smell

Module names should comply with a naming convention

Unused class-private methods should be removed

Analyze your code

Code Smell | Major | unused

"Class-Private" methods that are never executed inside their enclosing class are dead code: unnecessary, inoperative code that should be removed. Cleaning out dead code decreases the size of the maintained codebase, making it easier to understand the program and preventing bugs from being introduced.

Python has no real private methods. Every method is accessible. There are however two conventions indicating that a method is not meant to be "public":

- methods with a name starting with a single underscore (ex: `_mymethod`) should be seen as non-public and might change without prior notice. They should not be used by third-party libraries or software. It is ok to use those methods inside the library defining them but it should be done with caution.
- "class-private" methods have a name which starts with at least two underscores and ends with at most one underscore. These methods' names will be automatically mangled to avoid collision with subclasses' methods. For example `__mymethod` will be renamed as `_classname__mymethod`, where `classname` is the method's class name without its leading underscore(s). These methods shouldn't be used outside of their enclosing class.

This rule raises an issue when a class-private method (two leading underscores, max one underscore at the end) is never called inside the class. Class methods, static methods and instance methods will all raise an issue.

Noncompliant Code Example

```
class Noncompliant:

    @classmethod
    def __mangled_class_method(cls):  # Noncompliant
        print("__mangled_class_method")

    @staticmethod
    def __mangled_static_method():  # Noncompliant
        print("__mangled_static_method")

    def __mangled_instance_method(self):  # Noncomplian
        print("__mangled_instance_method")
```

Compliant Solution

```
class Compliant:

    def __init__(self):
        Compliant.__mangled_class_method()
```

Code Smell

Comments should not be located at the end of lines of code

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Files should contain an empty newline at the end

Code Smell

Long suffix "L" should be upper case

Code Smell

```
        Compliant.__mangled_static_method()
        self.__mangled_instance_method()

    @classmethod
    def __mangled_class_method(cls):
        print("__mangled_class_method")

    @staticmethod
    def __mangled_static_method():
        print("__mangled_static_method")

    def __mangled_instance_method(self):
        print("__mangled_instance_method")
```

See

- Python documentation – Private Variables
- PEP8 – Designing for Inheritance

Available In:

sonarlint | sonarcloud | sonarqube