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...

Special method "__exit__" should not re-raise the provided exception

Code Smell

Unused scope-limited definitions should be removed

Code Smell

Functions and methods should not have identical implementations

Code Smell

Unused private nested classes should be removed

Code Smell

String formatting should be used correctly

Code Smell

Conditional expressions should not be nested

Code Smell

Loops without "break" should not have "else" clauses

Code Smell

Doubled prefix operators "not" and "~" should not be used

Code Smell

The "print" statement should not be used

Code Smell

"<>" should not be used to test inequality

Code Smell

Two branches in a conditional structure should not have exactly the same implementation

Code Smell

Unused assignments should be

"self" should be the first argument to instance methods

Analyze your code

Code Smell Critical convention confusing suspicious

Instance methods, i.e. methods not annotated with `@classmethod` or `@staticmethod`, are expected to have at least one parameter. This parameter will reference the object instance on which the method is called. By convention, this first parameter is named "self".

Naming the "self" parameter differently is confusing. It might also indicate that the "self" parameter was forgotten, in which case calling the method will most probably fail.

Note also that creating methods which are used as static methods without the `@staticmethod` decorator is a bad practice because calling these methods on an instance will raise a `TypeError`. Either move the method out of the class or decorate it with `@staticmethod`.

This rule raises an issue when the first parameter of an instance method is not called "self".

Noncompliant Code Example

```
class MyClass:
    def send_request(request):  # Noncompliant. "self" was p
        print("send_request")

class ClassWithStaticMethod:
    def static_method(param):  # Noncompliant
        print(param)
ClassWithStaticMethod().static_method(42)  # Method is avail
```

Compliant Solution

```
class MyClass:
    def send_request(self, request):
        print("send_request")

class ClassWithStaticMethod:
    @staticmethod
    def static_method(param):
        print(param)
ClassWithStaticMethod().static_method(42)
```

Exceptions

This rule will also accept "cls" or "mcs" as first parameter's name for metaclasses' methods.

No issue will be raised for methods called `__init_subclass__`, `__class_getitem__` or `__new__` as these methods' first parameter is a class.

You can also disable issues on methods decorated with a specific decorator. Add these decorators to this rule's "ignoreDecorators" parameter.

removed

Code Smell

A field should not duplicate the name of its containing class

Code Smell

Function names should comply with a naming convention

Code Smell

Functions and lambdas should not reference variables defined in enclosing loops

Code Smell

With "ignoredDecorators" set to "abstractmethod"

```
from abc import abstractmethod, ABC

class MyClass(ABC):
    @abstractmethod
    def method():  # No issue, even if it is better in this
        pass
```

See

- Python documentation - Method Objects
- PEP8 - Function and Method Arguments

Available In:

sonarlint | sonarcloud | sonarqube