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...

Regular expressions should not be too complicated
Code Smell

Builtins should not be shadowed by local variables
Code Smell

Implicit string and byte concatenations should not be confusing
Code Smell

Identity comparisons should not be used with cached typed
Code Smell

Expressions creating sets should not have duplicate values
Code Smell

Expressions creating dictionaries should not have duplicate keys
Code Smell

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

Constants should not be used as conditions

Analyze your code

Code Smell | Critical | suspicious

When a constant is used as a condition, either it has no effect on the execution flow and it can be removed, or some code will never be executed and it is a bug.

This rule raises an issue when a constant expression is used as a condition in an `if`, `elif`, a conditional expression or other boolean expressions.

Noncompliant Code Example

```
def func(param = None):
    param = (1,)
    if param:  # Noncompliant. var is always set to (1,), th
        return sum(param)
    else:
        return None

var2 = 1 if func else 2  # Noncompliant. "func" will always
var3 = func and 1 else 2  # Noncompliant.
```

Compliant Solution

```
def func(param = None):
    if param is None:
        param = (1,)
    if param:
        return sum(param)
    else:
        return None

var2 = 1 if func() else 2
var3 = func() and 1 else 2
```

See

- PEP 285 - Adding a bool type
- Python documentation - Truth Value Testing

Available In:

sonarlint | sonarcloud | sonarqube

☢ 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