

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- 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...

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 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

Sections of code should not be commented out

Code Smell

Unused function parameters should

Some special methods should return "NotImplemented" instead of raising "NotImplementedError"

Analyze your code

Code Smell Critical error-handling bad-practice


In Python, special methods corresponding to numeric operators, rich comparison operators and the `__length_hint__` method should return `NotImplemented` when the operation is not supported. These methods should not raise `NotImplementedError` as callers don't expect it and won't catch this exception.

For example `A + B` is equivalent to calling `A.__add__(B)`. If this binary operation is not supported by class `A`, `A.__add__(B)` should return `NotImplemented`. The interpreter will then try the reverse operation, i.e. `B.__radd__(A)`. This enables adding new operations by changing only one class instead of two.


This rule raises an issue when one of the following methods raises `NotImplementedError` instead of returning `NotImplemented`:

- `__lt__(self, other)`
- `__le__(self, other)`
- `__eq__(self, other)`
- `__ne__(self, other)`
- `__gt__(self, other)`
- `__ge__(self, other)`
- `__add__(self, other)`
- `__sub__(self, other)`
- `__mul__(self, other)`
- `__matmul__(self, other)`
- `__truediv__(self, other)`
- `__floordiv__(self, other)`
- `__mod__(self, other)`
- `__divmod__(self, other)`
- `__pow__(self, other[, modulo])`
- `__lshift__(self, other)`
- `__rshift__(self, other)`
- `__and__(self, other)`
- `__xor__(self, other)`
- `__or__(self, other)`
- `__radd__(self, other)`
- `__rsub__(self, other)`
- `__rmul__(self, other)`
- `__rmatmul__(self, other)`
- `__rtruediv__(self, other)`
- `__rfloordiv__(self, other)`
- `__rmod__(self, other)`
- `__rdivmod__(self, other)`
- `__rpow__(self, other[, modulo])`
- `__rlshift__(self, other)`
- `__rrshift__(self, other)`
- `__rand__(self, other)`
- `__rxor__(self, other)`


be removed

 Code Smell


Unused class-private methods should be removed

 Code Smell


Track uses of "FIXME" tags

 Code Smell

"Exception" and "BaseException" should not be raised

 Code Smell

Redundant pairs of parentheses should be removed

 Code Smell

- `__ror__(self, other)`
- `__iadd__(self, other)`
- `__isub__(self, other)`
- `__imul__(self, other)`
- `__imatmul__(self, other)`
- `__itruediv__(self, other)`
- `__ifloordiv__(self, other)`
- `__imod__(self, other)`
- `__ipow__(self, other[, modulo])`
- `__lshift__(self, other)`
- `__rshift__(self, other)`
- `__iand__(self, other)`
- `__ixor__(self, other)`
- `__ior__(self, other)`
- `__length_hint__(self)`

Noncompliant Code Example

```
class MyClass:
    def __add__(self, other):
        raise NotImplementedError() # Noncompliant
    def __radd__(self, other):
        raise NotImplementedError() # Noncompliant

class MyOtherClass:
    def __add__(self, other):
        return 42
    def __radd__(self, other):
        return 42

MyClass() + MyOtherClass() # This will raise NotImplem
```

Compliant Solution

```
class MyClass:
    def __add__(self, other):
        return NotImplemented
    def __radd__(self, other):
        return NotImplemented

class MyOtherClass:
    def __add__(self, other):
        return 42
    def __radd__(self, other):
        return 42

MyClass() + MyOtherClass() # This returns 42
```

See

- Python documentation - [Built-in Constants - NotImplemented](#)
- Python documentation - [Implementing the arithmetic operations](#)

Available In:

sonarlint  | sonarcloud  | sonarqube 