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...

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

"SystemExit" should be re-raised

Analyze your code

Code Smell | Critical | error-handling bad-practice suspicious

`SystemExit` is raised when `sys.exit()` is called. This exception is expected to propagate up until the application stops. It is ok to catch it when a clean-up is necessary but it should be raised again immediately.

A bare `except:` statement, i.e. an `except` without any exception class, is equivalent to `except BaseException`. Both statements will catch every exception, including `SystemExit`. It is recommended to catch instead a specific exception. If it is not possible, the exception should be raised again.

Note that it is also a good idea to reraise the `KeyboardInterrupt` exception.

This rule raises an issue when a bare `except:`, an `except BaseException` or an `except SystemExit` don't reraise the exception caught.

Noncompliant Code Example

```
try:
    open("foo.txt", "r")
except SystemExit:  # Noncompliant
    pass
except KeyboardInterrupt:  # No issue raised but be car
    pass

try:
    open("bar.txt", "r")
except BaseException:  # Noncompliant
    pass
except:  # Noncompliant
    pass
```

Compliant Solution

```
try:
    open("foo.txt", "r")
except SystemExit:
    # clean-up
    raise
except KeyboardInterrupt:
    # clean-up
    raise

try:
    open("bar.txt", "r")
except BaseException as e:
    # clean-up
```

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

```
    raise e
except: # Noncompliant
    # clean-up
    raise

# or use a more specific exception

try:
    open("bar.txt", "r")
except FileNotFoundError:
    # process the exception
```

See

- PEP 352 - Required Superclass for Exceptions
- Python Documentation - Built-in exceptions
- Python Documentation - The `try` statement
- MITRE, CWE-391 - Unchecked Error Condition

Available In:

sonarlint | sonarcloud | sonarqube