

COBOL C#

3 CSS

 \bowtie Flex

Go -GO

5 HTML

Java

JavaScript JS

Kotlin

Objective C

PHP

PL/I

PL/SQL

Python

RPG

Ruby

Scala

N. Swift

Terraform

Text 月

TS 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)

6 Vulnerability (29)

R Bug (55)

Tags

Security Hotspot (31)

Search by name...

Code Smell (101)

runctions snould not have too many lines of code

A 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

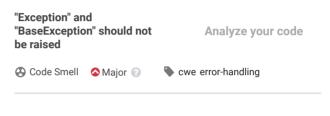
Parentheses should not be used after certain keywords

A 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



Raising instances of Exception and BaseException will have a negative impact on any code trying to catch these exceptions.

First, the only way to handle differently multiple Exceptions is to check their message, which is error-prone and difficult to maintain.

What's more, it becomes difficult to catch only your exception. The best practice is to catch only exceptions which require a specific handling. When you raise Exception or BaseException in a function the caller will have to add an except Exception or except BaseException and re-raise all exceptions which were unintentionally caught. This can create tricky bugs when the caller forgets to re-raise exceptions such as SystemExit and the software cannot be stopped.

It is recommended to either:

- raise a more specific Built-in exception when one matches. For example TypeError should be raised when the type of a parameter is not the one expected.
- create a custom exception class deriving from Exception or one of its subclasses. A common practice for libraries is to have one custom root exception class from which every other custom exception class inherits. It enables other projects using this library to catch all errors coming from the library with a single "except" statement

This rule raises an issue when Exception or BaseException are raised.

Noncompliant Code Example

```
def process1():
   raise BaseException("Wrong user input for field X")
def process2():
    raise BaseException("Wrong configuration") # Nonco
def process3(param):
    if not isinstance(param, int):
        raise Exception("param should be an integer")
def caller():
   try:
         process1()
         process2()
         process3()
    except BaseException as e:
        if e.args[0] == "Wrong user input for field X":
            # process error
            pass
        elif e.args[0] == "Wrong configuration":
            # process error
```

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

```
pass
else:
    # re-raise other exceptions
    raise
```

Compliant Solution

```
class MyProjectError(Exception):
    """Exception class from which every exception in th
         It enables other projects using this library t
         from the library with a single "except" statem
   pass
class BadUserInputError(MyProjectError):
   """A specific error""
   pass
class ConfigurationError(MyProjectError):
    """A specific error""
   pass
def process1():
   raise BadUserInputError("Wrong user input for field
   raise ConfigurationError("Wrong configuration")
def process3(param):
   if not isinstance(param, int):
        raise TypeError("param should be an integer")
def caller():
   try:
         process1()
         process2()
        process3()
   except BadUserInputError as e:
        # process error
       pass
    except ConfigurationError as e:
        # process error
        pass
```

See

- PEP 352 Required Superclass for Exceptions
- Python Documentation Built-in exceptions
- MITRE, CWE-397 Declaration of Throws for Generic Exception

Available In:

sonarlint ⊖ | sonarcloud ☆ | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved. Privacy Policy