

- 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

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 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

Nested blocks of code should not be left empty

Code Smell

Functions, methods and lambdas should not have too many parameters

`str.replace` should be preferred to `re.sub`

Analyze your code

Code Smell Critical ? regex performance

An `re.sub` call always performs an evaluation of the first argument as a regular expression, even if no regular expression features were used. This has a significant performance cost and therefore should be used with care.

When `re.sub` is used, the first argument should be a real regular expression. If it's not the case, `str.replace` does exactly the same thing as `re.sub` without the performance drawback of the regex.

This rule raises an issue for each `re.sub` used with a simple string as first argument which doesn't contains special regex character or pattern.

Noncompliant Code Example

```
input = "Bob is a Bird... Bob is a Plane... Bob is Supe
changed = re.sub("Bob is", "It's", input) # Noncompliant
changed = re.sub("\.\.\.", ";", changed) # Noncompliant
```

Compliant Solution

```
input = "Bob is a Bird... Bob is a Plane... Bob is Supe
changed = str.replace("Bob is", "It's", input)
changed = str.replace("...", ";", changed)
```





Or, with a regex:

```
input = "Bob is a Bird... Bob is a Plane... Bob is Supe
changed = re.sub(r"\w*\sis", "It's", input)
changed = re.sub(r"\.{3}", ";", changed)
```

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](#)

 Code Smell
<p>Collapsible "if" statements should be merged</p>  Code Smell
<p>Logging should not be vulnerable to injection attacks</p>  Vulnerability
<p>Repeated patterns in regular expressions should not match the empty string</p>  Bug
<p>Function parameters initial values should not be ignored</p> 