




 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



TypeScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your TYPESCRIPT code

All rules279

Vulnerability27

Bug51

Security Hotspot43

Code Smell158

Quick Fix50

Tags ▾

Search by name... 🔍

Code Smell

Primitive types should be omitted from initialized or defaulted declarations

Code Smell

Code Smell

Non-null assertions should not be used

Code Smell

Code Smell

"undefined" should not be assigned

Code Smell

Code Smell

Trailing commas should not be used

Code Smell

Code Smell

Array constructors should not be used

Code Smell

Code Smell

Quotes for string literals should be used consistently

Code Smell

Code Smell

Statements should end with semicolons

Code Smell

Code Smell

Comments should not be located at the end of lines of code

Code Smell

Code Smell

Loops should not contain more than a single "break" or "continue" statement

Code Smell

Code Smell

Variable, property and parameter names should comply with a naming convention

Code Smell

Code Smell

Lines should not end with trailing whitespaces

Code Smell

"===" and "!== " should be used instead of "==" and "!="

Analyze your code

Code Smell

Major

suspicious

The == and != operators do type coercion before comparing values. This is bad because it can mask type errors. For example, it evaluates ' \t\r\n' == 0 as true.

It is best to always use the side-effect-less === and !== operators instead.

Noncompliant Code Example

```
if (var == 'howdy') {...} // Noncompliant
```

Compliant Solution

```
if (var === 'howdy') {...}
```

Exceptions

Even if testing the equality of a variable against null doesn't do exactly what most JavaScript developers believe, usage of == or != is tolerated in such context. In the following case, if foo hasn't been initialized, its default value is not null but undefined. Nevertheless undefined == null, so JavaScript developers get the expected behavior.

```
if(foo == null) {...}
```

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](#)

https://rules.sonarsource.com/typescript/RSPEC-1440

1/2

<div>Files should contain an empty newline at the end</div> <div> Code Smell</div>
<div>An open curly brace should be located at the end of a line</div> <div> Code Smell</div>
<div>Tabulation characters should not be used</div> <div> Code Smell</div>
<div>Function and method names should comply with a naming convention</div> <div> Code Smell</div>