




 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



JavaScript static code analysis

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

All rules285

Vulnerability29

Bug62

Security Hotspot43

Code Smell151

Quick Fix41

Tags ▾

Search by name... 🔍

Object literal shorthand syntax should be used

Code Smell

Strings and non-strings should not be added

Code Smell

Object literal syntax should be used

Code Smell

"undefined" should not be assigned

Code Smell

Trailing commas should not be used

Code Smell

Array constructors should not be used

Code Smell

Quotes for string literals should be used consistently

Code Smell

Statements should end with semicolons

Code Smell

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

Code Smell

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

Code Smell

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

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Values not convertible to numbers should not be used in numeric comparisons

Analyze your code

Code Smell

Major ?

In a Zen-like manner, NaN isn't equal to anything, even itself. So comparisons (`>`, `<`, `>=`, `<=`) where one operand is NaN or evaluates to NaN always return `false`. Specifically, `undefined` and objects that cannot be converted to numbers evaluate to NaN when used in numerical comparisons.

This rule raises an issue when there is at least one path through the code where one of the operands to a comparison is NaN, `undefined` or an `Object` which cannot be converted to a number.

Noncompliant Code Example

```
var x; // x is currently "undefined"
if (someCondition()) {
  x = 42;
}

if (42 > x) { // Noncompliant; "x" might still be "undefined"
  doSomething();
}

var obj = {prop: 42};
if (obj > 24) { // Noncompliant
  doSomething();
}
```




Compliant Solution

```
var x;
if (someCondition()) {
  x = 42;
} else {
  x = foo();
}

if (42 > x) {
  doSomething();
}

var obj = {prop: 42};
if (obj.prop > 24) {
  doSomething();
}
```

Available In:

sonarlint  | sonarcloud  | sonarqube 

https://rules.sonarsource.com/javascript/RSPEC-3758

1/2

Files should contain an empty newline at the end

 Code Smell

An open curly brace should be located at the end of a line

 Code Smell

Tabulation characters should not be used

 Code Smell

Function and method names should comply with a naming convention

 Code Smell

© 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](#)