




 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... 🔍

 Code Smell


Functions should use "return" consistently




Variables and functions should not be declared in the global scope




Arithmetic operators should only have numbers as operands




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




Arithmetic operations should not result in "NaN"




"arguments" should not be accessed directly




Comparison operators should not be used with strings




Property getters and setters should come in pairs




JavaScript parser failure



The ternary operator should not be used





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





Literals should not be thrown

Analyze your code

 Code Smell

 Major

 Quick Fix

 error-handling api-design

It is a bad practice to throw something that's not derived at some level from `Error`. If you can't find an existing `Error` type that suitably conveys what you need to convey, then you should extend `Error` to create one.

Specifically, part of the point of throwing `Errors` is to communicate about the conditions of the error, but literals have far less ability to communicate meaningfully than `Errors` because they don't include stacktraces.

Noncompliant Code Example

```
throw 404; // Noncompliant
throw "Invalid negative index."; // Noncompliant
```

Compliant Solution

```
throw new Error("Status: " + 404);
throw new Error("Invalid negative index.");{code}
```

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/javascript/RSPEC-3696

1/2

<div>Functions should not have too many lines of code</div> <div> Code Smell</div>
<div>Track comments matching a regular expression</div> <div> Code Smell</div>
<div>Statements should be on separate lines</div> <div> Code Smell</div>
<div>Magic numbers should not be used</div> <div> Code Smell</div>