




 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 rules 279

Vulnerability 27

Bug 51

Security Hotspot 43

Code Smell 158

Quick Fix 50

Tags ▾

Search by name... 🔍

Dependencies should be explicit

Code Smell

"this" should not be assigned to variables

Code Smell

The "any" type should not be used

Code Smell

"for in" should not be used with iterables

Code Smell

Functions should use "return" consistently

Code Smell

"arguments" should not be accessed directly

Code Smell

Comparison operators should not be used with strings

Code Smell

Private properties that are only assigned in the constructor or at declaration should be "readonly"

Code Smell

Property getters and setters should come in pairs

Code Smell

JavaScript parser failure

Code Smell

The ternary operator should not be used

Code Smell

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

Code Smell

Ternary operators should not be nested

Analyze your code

Code Smell

Major

confusing

Just because you *can* do something, doesn't mean you should, and that's the case with nested ternary operations. Nesting ternary operators results in the kind of code that may seem clear as day when you write it, but six months later will leave maintainers (or worse - future you) scratching their heads and cursing.

Instead, err on the side of clarity, and use another line to express the nested operation as a separate statement.

Noncompliant Code Example

```
function getReadableStatus(job) {
  return job.isRunning() ? "Running" : job.hasErrors() ? "Failed" : "Succeeded";
}
```

Compliant Solution

```
function getReadableStatus(job) {
  if (job.isRunning()) {
    return "Running";
  }
  return job.hasErrors() ? "Failed" : "Succeeded";
}
```





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-3358

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>