




 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




Non-null assertions should not be used




"undefined" should not be assigned




Trailing commas should not be used




Array constructors should not be used




Quotes for string literals should be used consistently




Statements should end with semicolons




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




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



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




Lines should not end with trailing whitespaces




Wrapper objects should not be used for primitive types


Analyze your code




 Code Smell



 Minor



 Quick Fix



 pitfall

The use of wrapper objects for primitive types is gratuitous, confusing and dangerous. If you use a wrapper object constructor for type conversion, just remove the new keyword, and you'll get a primitive value automatically. If you use a wrapper object as a way to add properties to a primitive, you should re-think the design. Such uses are considered bad practice, and should be refactored.


Noncompliant Code Example


```
let x = new Number("0");
if (x) {
  alert('hi'); // Shows 'hi'.
}
```


Compliant Solution

```
let x = Number("0");
if (x) {
  alert('hi');
}
```

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-1533

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>