




 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... 🔍

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

"arguments" should not be accessed directly

Analyze your code

Code Smell

Major

api-design es2015

The magic of JavaScript is that you can pass arguments to functions that don't declare parameters, and on the other side, you can use those passed-in arguments inside the no-args function.

But just because you can, that doesn't mean you should. The expectation and use of arguments inside functions that don't explicitly declare them is confusing to callers. No one should ever have to read and fully understand a function to be able to use it competently.

If you don't want to name arguments explicitly, use the ... syntax to specify that an a variable number of arguments is expected. Then inside the function, you'll be dealing with a first-class array, rather than an array-like structure.

Noncompliant Code Example

```
function concatenate() {
  let args = Array.prototype.slice.call(arguments); // Nonc
  return args.join(' ');
}

function doSomething(isTrue) {
  var args = Array.prototype.slice.call(arguments, 1); // No
  if (!isTrue) {
    for (var arg of args) {
      ...
    }
  }
}
```

Compliant Solution

```
function concatenate(...args) {
  return args.join(' ');
}

function doSomething(isTrue, ...values) {
  if (!isTrue) {
    for (var value of values) {
      ...
    }
  }
}
```

Available In:

sonarlint

sonarcloud

sonarqube

https://rules.sonarsource.com/typescript/RSPEC-3513

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>

© 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](#)