JavaScript(https://www.sitepoint.com/javascript/) - August 22, 2017 -
By Craig Buckler (https://www.sitepoint.com/author/craig-buckler/)

Truthy and Falsy: When All is Not Equal in JavaScript

Related Topics:

React (https://www.sitepoint.com/javascript/react/) Node.js (https://www.sitepoint.com/javascript/nodejs-javascript/)
APIs (https://www.sitepoint.com/javascript/apis-javascript/) AngularJS (https://www.sitepoint.com/javascript/angularjs/)
jQuery (https://www.sitepoint.com/javascript/jquery/) More... (https://www.sitepoint.com/javascript/)

JavaScript variables are loosely/dynamically typed and the language doesn't care how a value is declared or changed.

> **2017.08.22:** This article has been updated to reflect the current state of the JavaScript ecosystem.

```
 let x;
x = 1;   // x is a number
x = '1'; // x is a string
x = [1]; // x is an array
```

Seemingly different values equate to `true` when compared with `==` (loose or abstract equality) because JavaScript (effectively) converts each to a string representation before comparison:

```
// all true
1 == '1';
1 == [1];
'1' == [1];
```

A more obvious `false` result occurs when comparing with `===` (strict equality) because the type is considered:

```
// all false
1 === '1';
1 === [1];
'1' === [1];
```

Internally, JavaScript sets a value to one of six primitive data types:

- Undefined (a variable with no defined value)
- Null (a single null value)
- Boolean (true or false)
- Number (this includes `Infinity` and `NaN` – not a number!)
- String (textual data)
- Symbol (a unique and immutable primitive new to ES6/2015)

Everything else is an Object — including arrays.

Truthy and Falsy

As well as a type, each value also has an inherent boolean value, generally known as either *truthy* or *falsy*. Some of the rules are a little bizarre so understanding the concepts and effect on comparison helps when debugging JavaScript applications.

The following values are **always falsy**:

- `false`
- `0` (zero)
- `''` or `""` (empty string)
- `null`
- `undefined`
- `NaN` (e.g. the result of `1/0`)

Everything else is **truthy**. That includes:

- `'0'` (a string containing a single zero)
- `'false'` (a string containing the text “false”)
- `[]` (an empty array)
- `{}` (an empty object)

`function(){}` (an "empty" function)

A single value can therefore be used within conditions, e.g.

```
if (value) {
  // value is truthy
}
else {
  // value is falsy
  // it could be false, 0, '', null, undefined or NaN
}
```

Loose Equality Comparisons With `==`

Unexpected situations can occur when comparing *truthy* and *falsy* values using the `==` loose equality:

`==`	`true`	`false`	`0`	`''`	`null`	`undefined`	`NaN`	`Infinity`	`[]`	`{}`
`true`	true	false	false	false	false	false	false	false	false	false
`false`	false	true	true	true	false	false	false	false	true	false
`0`	false	true	true	true	false	false	false	false	true	false
`''`	false	true	true	true	false	false	false	false	true	false
`null`	false	false	false	false	true	true	false	false	false	false
`undefined`	false	false	false	false	true	true	false	false	false	false
`NaN`	false	false	false	false	false	false	false	false	false	false
`Infinity`	false	false	false	false	false	false	false	true	false	false
`[]`	false	true	true	true	false	false	false	false	true	false
`{}`	false	false	false	false	false	false	false	false	false	true

The rules:

1. `false`, zero and empty strings are all equivalent.
2. `null` and `undefined` are equivalent to themselves and each other but nothing else.
3. `NaN` is not equivalent to anything – *including another `NaN`!*
4. `Infinity` is truthy – *but cannot be compared to `true` or `false`!*
5. An empty array is truthy – *yet comparing with `true` is `false` and comparing with `false` is `true`?!*

Examples:

```
// all true
false == 0;
0 == '';
null == undefined;
[] == false;
!![0] == true;

// all false
false == null;
NaN == NaN;
Infinity == true;
[] == true;
[0] == true;
```

Strict Equality Comparisons With `===`

The situation is clearer when using a strict comparison because the value types must match:

`===`	`true`	`false`	`0`	`''`	`null`	`undefined`	`NaN`	`Infinity`	`[]`	`{}`
`true`	true	false	false	false	false	false	false	false	false	false
`false`	false	true	false	false	false	false	false	false	false	false
`0`	false	false	true	false	false	false	false	false	false	false
`''`	false	false	false	true	false	false	false	false	false	false
`null`	false	false	false	false	true	false	false	false	false	false
`undefined`	false	false	false	false	false	true	false	false	false	false
`NaN`	false	false	false	false	false	false	false	false	false	false
`Infinity`	false	false	false	false	false	false	false	true	false	false
`[]`	false	false	false	false	false	false	false	false	true	false
`{}`	false	false	false	false	false	false	false	false	false	true

The only exception is `NaN` which remains stubbornly inequivalent to everything.

Recommendations

Truthy and falsy values can catch out the most experienced developers. Those new to programming or migrating from other languages have no chance! Fortunately, there are simple steps to catch the most difficult-to-spot errors when handling truthy and falsy variables:

1. Avoid direct comparisons

It's rarely necessary to compare two truthy and falsy values when a single value will always equate to true cr false:

```
// instead of
if (x == false) // ...
// runs if x is false, 0, '', or []

// use
if (!x) // ...
// runs if x is false, 0, '', NaN, null or undefined
```

2. Use `===` strict equality

Use a `===` strict equality (or `!==` strict inequality) comparisons to compare values and avoid type conversion issues:

```
// instead of
if (x == y) // ...
// runs if x and y are both truthy or both falsy
// e.g. x = null and y = undefined

// use
if (x === y) // ...
// runs if x and y are identical...
// except when both are NaN
```

3. Convert to real Boolean values where necessary

Any value can be converted to a real Boolean value using a double-negative `!!` to be absolutely certain a false is generated only by `false`, `0`, `""`, `null`, `undefined` and `NaN`:

```
// instead of
if (x === y) // ...
// runs if x and y are identical...
// except when both are NaN

// use
if (!!x === !!y) // ...
// runs if x and y are identical...
// including when either or both are NaN
```

Conclusion

Truthy and falsy values allow you to write terse JavaScript conditions and ternary operators. However, always consider the edge cases. A rogue empty array or NaN variable could lead to many hours of debugging grief!

Meet the author

Craig Buckler (https://www.sitepoint.com/author/craig-buckler/)
(https://twitter.com/craigbuckler)
(http://plus.google.com/+CraigBuckler)
(http://www.facebook.com/craigbuckler)
(http://www.linkedin.com/in/craigbuckler)
(https://github.com/craigbuckler)

Craig is a freelance UK web consultant who built his first page for IE2.0 in 1995. Since that time he's been advocating standards, accessibility, and best-practice HTML5 techniques. He's written more than 1,000 articles for SitePoint and you can find him @craigbuckler (http://twitter.com/craigbuckler).

Stuff We Do

- Premium (/premium/)
- Versioning (/versioning/)
- Themes (/themes/)
- Forums (/community/)
- References (/html-css/css/)

About

- Our Story (/about-us/)
- Press Room (/press/)

Contact

- Contact Us (/contact-us/)
- FAQ (https://sitepoint.zendesk.com/hc/en-us)
- Write for Us (/write-for-us/)
- Advertise (/advertise/)

Legals

- Terms of Use (/legals/)
- Privacy Policy (/legals/#privacy)

Connect

(https://www.facebook.com/sitepoint) (http://twitter.com/sitepointdotcom) (/versioning/) (https://www.sitepoint.com/feed/) (https://plus.google.com/+sitepoint)