(https://js.foundation/about/donate)

SUPPORT THE PROJECT

JavaScript Style Guide

(/repo-maintainers-guide/)

Linting

Use JSHint to detect errors and potential problems. Every jQuery project has a Grunt task for linting all JavaScript files: `grunt jshint`. The options for JSHint are stored in a `.jshintrc` file; many repositories will have multiple `.jshintrc` files based on the type of code in each directory.

Each `.jshintrc` file follows a specific format. All options must be alphabetized and grouped:

```
{
    "common1": true,
    "common2": true,

    "repoSpecific1": true,
    "repoSpecific2": true,

    "globals": {
        "repoGlobal1": true,
        "repoGlobal2": false
    }
}
```

The following common options must be used in all projects:

```
{
    "boss": true,
    "curly": true,
    "eqeqeq": true,
    "eqnull": true,
    "expr": true,
    "immed": true,
    "noarg": true,
    "quotmark": "double",
    "smarttabs": true,
    "trailing": true,
    "undef": true,
    "unused": true
}
```

If the project supports browsers which do not implement ES5, then the `es3` option must be included with the repo-specific options.

Spacing

In general, the jQuery style guide encourages liberal spacing for improved human readability. The minification process creates a file that is optimized for browsers to read and process.

Indentation with tabs.

No whitespace at the end of line or on blank lines.

Lines should be no longer than 80 characters, and must not exceed 100 (counting tabs as 4 spaces). There are 2 exceptions, both allowing the line to exceed 100 characters:

If the line contains a comment with a long URL.

If the line contains a regex literal. This prevents having to use the regex constructor which requires otherwise unnecessary string escaping.

`if` / `else` / `for` / `while` / `try` always have braces and always go on multiple lines.

Unary special-character operators (e.g., `!`, `++`) must not have space next to their operand.

Any `,` and `;` must not have preceding space.

Any `;` used as a statement terminator must be at the end of the line.

Any `:` after a property name in an object definition must not have preceding space.

The `?` and `:` in a ternary conditional must have space on both sides.

No filler spaces in empty constructs (e.g., `{}`, `[]`, `fn()`)

New line at the end of each file.

If the entire file is wrapped in a closure, the function body is not indented. See full file closures for examples.

Bad Examples

```
1 // Bad
2 if(condition) doSomething();
3 while(!condition) iterating++;
4 for(var i=0;i<100;i++) object[array[i]] = someFn(i);
```

Good Examples

```
var i = 0;

if ( condition ) {
    doSomething();
} else if ( otherCondition ) {
    somethingElse();
} else {
    otherThing();
}

while ( !condition ) {
    iterating++;
}

for ( ; i < 100; i++ ) {
    object[ array[ i ] ] = someFn( i );
}

try {

    // Expressions
} catch ( e ) {

    // Expressions
}

array = [ "*" ];

array = [ a, b ];

foo( arg );

foo( options, object[ property ] );

foo( [ a, b ], "property", { c: d } );

foo( { a: "alpha", b: "beta" } );

foo( [ a, b ] );

foo( {
    a: "alpha",
    b: "beta"
} );

foo( function() {

    // Do stuff
}, options );

foo( data, function() {

    // Do stuff
} );
```

Object and Array Expressions

Object and array expressions can be on one line if they are short (remember the line length limits). When an expression is too long to fit on one line, there

must be one property or element per line, with the opening and closing braces each on their own lines. Property names only need to be quoted if they are reserved words or contain special characters:

```
// Bad
map = { ready: 9,
    when: 4, "you are": 15 };

array = [ 9,
    4,
    15 ];

array = [ {
    key: val
} ];

array = [ {
    key: val
}, {
    key2: val2
} ];

// Good
map = { ready: 9, when: 4, "you are": 15 };

array = [ 9, 4, 15 ];

array = [ { key: val } ];

array = [ { key: val }, { key2: val2 } ];

array = [
    { key: val },
    { key2: val2 }
];

// Good as well
map = {
    ready: 9,
    when: 4,
    "you are": 15
};

array = [
    9,
    4,
    15
];

array = [
    {
        key: val
    }
];

array = [
    {
        key: val
    },
    {
        key2: val2
    }
];
```

Multi-line Statements

When a statement is too long to fit on one line, line breaks must occur after an operator.

```
// Bad
var html = "<p>The sum of " + a + " and " + b + " pl
    + " is " + ( a + b + c );

// Good
var html = "<p>The sum of " + a + " and " + b + " pl
    " is " + ( a + b + c );
```

Lines should be broken into logical groups if it improves readability, such as splitting each expression of a ternary operator onto its own line even if both will fit on a single line.

```
var baz = firstCondition( foo ) && secondCondition(
    qux( foo, bar ) :
    foo;
```

When a conditional is too long to fit on one line, successive lines must be indented one extra level to distinguish them from the body.

```
    if ( firstCondition() && secondCondition() &&
            thirdCondition() ) {
        doStuff();
    }
```

Chained Method Calls

When a chain of method calls is too long to fit on one line, there must be one call per line, with the first call on a separate line from the object the methods are called on. If the method changes the context, an extra level of indentation must be used.

```
elements
    .addClass( "foo" )
    .children()
        .html( "hello" )
    .end()
    .appendTo( "body" );
```

Full File Closures

When an entire file is wrapped in a closure, the body of the closure is not indented.

```
( function( $ ) {

// This doesn't get indented

} )( jQuery );
```

```
module.exports = function( grunt ) {

// This doesn't get indented

};
```

The same applies to AMD wrappers.

```
define( [
    "foo",
    "bar",
    "baz"
], function( foo, bar, baz ) {

// This doesn't get indented

} );
```

For UMD, the factory is indented to visually differentiate it from the body.

```
( function( factory ) {
    if ( typeof define === "function" && define.amd

        // AMD. Register as an anonymous module.
        define( [
            "jquery"
        ], factory );
    } else {

        // Browser globals
        factory( jQuery );
    }
}( function( $ ) {

// This doesn't get indented

} ) );
```

Constructors

Constructor functions should always be invoked with argument lists, even when such lists are empty.

```
throw new Error();
when = time || new Date();
```

When property access or method invocation is immediately performed on the result of a constructor function, clarify precedence with wrapping parentheses.

```
1 detachedMode = ( new TemplateFactory( settings ) ).n
2 match = ( new RegExp( pattern ) ).exec( input );
```

Equality

Strict equality checks (`===`) must be used in favor of abstract equality checks (`==`). The *only* exception is when checking for `undefined` and `null` by way of `null`. The use of `== null` is also acceptable in cases where only one of `null` or `undefined` may be logically encountered, such as uninitialized variables.

```
1 // Check for both undefined and null values, for som
2 undefOrNull == null;
```

Type Checks

String: `typeof object === "string"`

Number: `typeof object === "number"`

Boolean: `typeof object === "boolean"`

Object: `typeof object === "object"`

Plain Object: `jQuery.isPlainObject( object )`

Function: `jQuery.isFunction( object )`

Array: `jQuery.isArray( object )`

Element: `object.nodeType`

null: `object === null`

null or undefined: `object == null`

undefined:

- Global Variables: `typeof variable === "undefined"`
- Local Variables: `variable === undefined`
- Properties: `object.prop === undefined`

Comments

Comments are always preceded by a blank line. Comments start with a capital first letter, but don't require a period at the end, unless you're writing full sentences. There must be a single space between the comment token and the comment text.

Single line comments go **over** the line they refer to:

```
// We need an explicit "bar", because later in the c
var foo = "bar";

// Even long comments that span
// multiple lines use the single
// line comment form.
```

Multi-line comments are only used for file and function headers.

Inline comments are allowed as an exception when used to annotate special arguments in formal parameter lists or when needed to support a specific development tool:

```
function foo( types, selector, data, fn, /* INTERNAL

    // Do stuff
}
```

Do not write API documentation in comments. API documentation lives in its own repository.

Quotes

jQuery uses double quotes.

```
var double = "I am wrapped in double quotes";
```

Strings that require inner quoting must use double outside and single inside.

```
var html = "<div id='my-id'></div>";
```

Semicolons

Use them. Never rely on ASI.

Naming Conventions

Variable and function names should be full words, using camel case with a lowercase first letter. Names should be descriptive but not excessively so. Exceptions are allowed for iterators, such as the use of `i` to represent the index in a loop. Constructors do not need a capital first letter.

Global Variables

Each project may expose at most one global variable.

DOM Node Rules

`.nodeName` must always be used in favor of `.tagName`.

`.nodeType` must be used to determine the classification of a node (not `.nodeName`).

Switch Statements

The usage of `switch` statements is generally discouraged, but can be useful when there are a large number of cases - especially when multiple cases can be handled by the same block, or fall-through logic (the `default` case) can be leveraged.

When using `switch` statements:

Use a `break` for each case other than `default`.

Align `case` statements with the `switch`.

```
switch ( event.keyCode ) {
case $.ui.keyCode.ENTER:
case $.ui.keyCode.SPACE:
    x();
    break;
case $.ui.keyCode.ESCAPE:
    y();
    break;
default:
    z();
}
```

Twitter (https://twitter.com/jquery)

IRC (https://irc.jquery.org/)

GitHub (https://github.com/jquery)