Javascript Coding Guidelines

- Reduce the use of Global Variables.
- Avoid the use of Inline Javascript. Do not mix HTML & Javascript. Concentrate all the Javascript into a single .js file rather than using multiple script blocks with the html view page.
- Try to include all the Javascript functions within a single Namespace
- Try to cache jQuery Selectors for non-dynamic HTML elements. For example
    - var selector = $("#selector");
    - selector.hide();
    - selector.show();
- Do not write DOM selectors inside Loops.

Variable declaration

Declare all vars at the top of their scope when possible. As JavaScript has function level scoping, not block level as in some other languages, this helps clarify where those variables are accessible.

```
function sample() {
  //the variable 'foo' is available from the start of function scope
  var foo;

  if ( condition ) {
    foo = 'bar';
  }
  return foo;
}
```

Declare all vars with one var keyword when appropriate and use preceding commas. This is more performant than multiple statements.

```
//avoid repeating var in variable declaration
var foo;
var bar;
var lorem;
var ipsum;

//comma separated variable declarations
var foo
, bar
, lorem
, ipsum;
```

Declare Variables Outside of the For Statement

When executing lengthy "for" statements, don't make the engine work any harder than it must. For example:

```
// bad
for(var i = 0; i < someArray.length; i++) {
   var container = document.getElementById('container');
   container.innerHtml += 'my number: ' + i;
   console.log(i);
}
```

```
//better
var container = document.getElementById('container');
for(var i = 0, len = someArray.length; i < len;  i++) {
  container.innerHtml += 'my number: ' + i;
  console.log(i);
}
```

Coding Template

- All JS file should start with a closure.

Please try to use the following template for writing your javascript code.

```
(function() {

    'use strict';  // Enforces Javascript strict mode, a restricted variant of
Javascript. This line should be at the top of other codes you
may write

    // All the javascript code should be written inside this block.
    // Any code written within this block won't be accessible outside. This ensures
encapsulation.

var _this, app = {
     el: {
        // All static Element Selectors should be defined within this block
        $button1: $("#button1"),
        $textbox1: $("#textbox1")
       },
     init: function() {
        _this = this;

          // This function acts as the Class Constructor

        _this.initPlugins();
        _this.initEventHandlers();
        },
        buttonClickHandler: function() {

          alert("Hello");

        },
        initEventHandlers: function() {
            var el = _this.el;
            // Registering Eventlisteners
            el.$button1.on('click', _this.buttonClickHandler);
         },
        initPlugins: function() {

            // Any Javascript/jQuery plugins can be incorporated here
            _this.textbox1.datepicker();
        }

    };
// Initialize the Application once DOM is loaded
   document.addEventListener('DOMContentLoaded', app.init.bind(app));
})();
```

- Include all the script files just above the closing body tag & the style sheets should be included inside the head tag.

Syntax Conventions

The syntax of a program is the set of rules that must be adhered to in writing the code so that it can be interpreted or compiled properly. The following table depicts guidelines where the syntax may be written in more than one way. Code Formatting: Use white spaces and indents to make the code readable.

Example:

```
for(i=0; i < 10; i++){
 statement1 ...
 statement2 ...
 }
```

Semicolons: Use a semicolon at the end of each statement. JavaScript does not require a semicolon at the end of a statement if it is on its own line. Semicolons should always be used for readability, consistency, and ease in debugging.

Example:

```
a=3; NOT a=3
b=2; b=3
```

Use === Instead of ==

Why strict comparison operators?

JavaScript has both strict and type-converting equality comparison. For strict equality the objects being compared must
have the same type and:

* Two strings are strictly equal when they have the same sequence of characters, same length, and same characters
in corresponding positions.
* Two numbers are strictly equal when they are numerically equal (have the same number value).
NaN is not equal to anything, including NaN. Positive and negative zeros are equal to one another.
* Two Boolean operands are strictly equal if both are true or both are false.
* Two objects are strictly equal if they refer to the same Object.
* Null and Undefined types are == (but not ===). [I.e. Null==Undefined (but not Null===Undefined)]

OTBS

Use "one true brace style" (1TBS or OTBS) syntax. JavaScript handles the return statement differently in the following two scenarios:

```
//OTBS
return {
  'status': 'ok'
};

//K&R
return
{
  'status': 'ok'
};
```

Don't Use new Object()

- Use `""` instead of `new String()`
- Use `0` instead of `new Number()`
- Use `false` instead of `new Boolean()`
- Use `{}` instead of `new Object()`
- Use `[]` instead of `new Array()`
- Use `/()/` instead of `new RegExp()`
- Use `function (){}` instead of `new Function()`

Use Parameter Defaults

If a function is called with a missing argument, the value of the missing argument is set to `undefined`.

Undefined values can break your code. It is a good habit to assign default values to arguments.

```
function myFunction(x, y) {

if (y === undefined) {

     y = 0;

  }

}
```

ECMAscript2015

```
function (a=1, b=1) { /*function code*/ }
```

End Your Switches with Defaults

Always end your `switch` statements with a `default`. Even if you think there is no need for it.

Avoid Using eval()

The `eval()` function is used to run text as code. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem.

Object Rules

General rules for object definitions:

- Place the opening bracket on the same line as the object name.
- Use colon plus one space between each property and its value.
- Use quotes around string values, not around numeric values.
- Do not add a comma after the last property-value pair.
- Place the closing bracket on a new line, without leading spaces.
- Always end an object definition with a semicolon.

```
const person = {
  firstName: "John",

  lastName: "Doe",

  age: 50,

  eyeColor: "blue"

};
```

Naming Conventions

Always use the same naming convention for all your code. For example:

- Variable and function names written as camelCase
- Global variables written in UPPERCASE (We don't, but it's quite common)
- Constants (like PI) written in UPPERCASE

Hyphens in HTML and CSS:

HTML5 attributes can start with data- (data-quantity, data-price).

CSS uses hyphens in property-names (font-size).

Spacing

- No whitespace at the end of line or on blank lines.
- Lines should usually be no longer than 80 characters, and should not exceed 100 (counting tabs as 4 spaces). *This is a "soft" rule, but long lines generally indicate unreadable or disorganized code.*
- `if/else/for/while/try` blocks should always use 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()`).
- There should be a new line at the end of each file.
- Any `!` negation operator should have a following space.*
- All function bodies are indented by one tab, even if the entire file is wrapped in a closure.*
- Spaces may align code within documentation blocks or within a line, but only tabs should be used at the start of a line.*

Declare Objects with const

Declaring objects with const will prevent any accidental change of type:Its depends on your use cases

```
let car = {type:"Fiat", model:"500", color:"white"};

car = "Fiat";       // Changes object to string

const car = {type:"Fiat", model:"500", color:"white"};

car = "Fiat";       // Not possible
```

Declare Arrays with const

Declaring arrays with const will prevent any accidential change of type:its depends on the usecases

```
let cars = ["Saab", "Volvo", "BMW"];

cars = 3;    // Changes array to number
```

```
const cars = ["Saab", "Volvo", "BMW"];

cars = 3;     // Not possible
```

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 + ' plus ' + c

   + ' is ' + ( a + b + c ) + '</p>';



// Good

var html = '<p>The sum of ' + a + ' and ' + b + ' plus ' + c +

   ' is ' + ( a + b + c ) + '</p>';
```

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.

```
// Acceptable

var baz = ( true === conditionalStatement() ) ? 'thing 1' : 'thing 2';



// Better
```

```
var baz = firstCondition( foo ) && secondCondition( bar ) ?

    qux( foo, bar ) :

    foo;
```

Support Libraries

These libraries provide functions, including _.each, _.map, and _.reduce, allow for efficient, readable transformations of large data sets.

- underscore.js,lodash