# [JavaScript Style Guide](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide)

This is a modified version of [https://github.com/airbnb/javascript](https://github.com/airbnb/javascript" \t "_blank).

* + [Objects](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Objects)
  + [Arrays](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Arrays)
  + [Strings](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Strings)
  + [Functions](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Functions)
  + [Properties](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Properties)
  + [Variables](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Variables)
  + [Conditional Expressions & Equality](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-ConditionalExpressions&Equality)
  + [Blocks](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Blocks)
  + [Whitespace](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Whitespace)
  + [Commas](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Commas)
  + [Semicolons](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Semicolons)
  + [Naming Conventions](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-NamingConventions)
* [Linting](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-Linting)
* [Best Practices](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-BestPractices)
* [Code Blocks](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-CodeBlocks)
  + [Known Issues](https://code.bestbuy.com/wiki/display/FED/JavaScript+Style+Guide#JavaScriptStyleGuide-KnownIssues)

**Objects**

* Use the literal syntax for object creation.

|  |
| --- |
| // bad  var item = new Object();    // good  var item = {}; |

**Arrays**

* Use the literal syntax for array creation.

|  |
| --- |
| // bad  var items = new Array();    // good  var items = []; |

**Strings**

* Use double quotes "" for strings.

|  |
| --- |
| // bad  var name = 'Bob Parr';    // good  var name = "Bob Parr"; |

**Functions**

* Never declare a function in a non-function block (if, while, etc). Assign the function to a variable instead.

|  |
| --- |
| // anonymous function expression  var anonymous = function() {    return true;  };    // named function expression  var named = function named() {    return true;  };    // immediately-invoked function expression  (function() {    console.log('Welcome to the Internet. Please follow me.');  })(); |

**Properties**

* Use dot notation when accessing properties.

|  |
| --- |
| var luke = {    jedi: true,    age: 28  };    // bad  var isJedi = luke["jedi"];    // good  var isJedi = luke.jedi; |

**Variables**

* Always use var to declare variables. Not doing so will result in global variables. We want to avoid polluting the global namespace.

|  |
| --- |
| // bad  superPower = new SuperPower();    // good  var superPower = new SuperPower(); |

* Use one var declaration per variable. It's easier to add new variable declarations this way, and you never have to worry about swapping out a ; for a , or introducing punctuation-only diffs.

|  |
| --- |
| // bad  var items = getItems(),      goSportsTeam = true,      dragonball = 'z';    // good  var items = getItems();  var goSportsTeam = true;  var dragonball = 'z'; |

**Conditional Expressions & Equality**

* Use === and !== over == and !=.

**Blocks**

* Use braces with all blocks.

|  |
| --- |
| // bad  if (a === b) a++; b--;    // good  if (a === b) {      a++;  }  b--; |

**Whitespace**

* Use tabs not spaces (no soft tabs).
* Place 1 space before the leading brace.

|  |
| --- |
| // bad  function test(){    console.log('test');  }    // good  function test() {    console.log('test');  } |

* Set off operators with spaces.

|  |
| --- |
| // bad  var x=y+5;    // good  var x = y + 5; |

* End files with a single newline character.
* Use indentation when making long method chains. Use a leading dot, which emphasizes that the line is a method call, not a new statement.

|  |
| --- |
| // bad  $('#items').find('.selected').highlight().end().find('.open').updateCount();    // good  $('#items')    .find('.selected')      .highlight()      .end()    .find('.open')      .updateCount(); |

* Leave a blank line after blocks and before the next statement.

|  |
| --- |
| // bad  if (foo) {    return bar;  }  return baz;    // good  if (foo) {    return bar;  }    return baz; |

**Commas**

* Leading commas: **Nope.**

|  |
| --- |
| // bad  var story = [      once    , upon    , aTime  ];    // good  var story = [    once,    upon,    aTime  ]; |

* Additional trailing comma: **Nope.**

|  |
| --- |
| // bad  var hero = {    firstName: 'Kevin',    lastName: 'Flynn',  };    // good  var hero = {    firstName: 'Kevin',    lastName: 'Flynn'  }; |

**Semicolons**

* Yup.

|  |
| --- |
| // bad  myFunc()    // good  myFunc(); |

**Naming Conventions**

* Avoid single letter names. Be descriptive with your naming.

|  |
| --- |
| // bad  function q() {    // ...stuff...  }    // good  function query() {    // ..stuff..  } |

* Use camelCase when naming objects, functions, and instances.

|  |
| --- |
| // bad  var OBJEcttsssss = {};  var this\_is\_my\_object = {};  function c() {}  var u = new user({    name: 'Bob Parr'  });    // good  var thisIsMyObject = {};  function thisIsMyFunction() {}  var user = new User({    name: 'Bob Parr'  }); |

* Use PascalCase when naming constructors or classes.

|  |
| --- |
| // bad  function user(options) {    this.name = options.name;  }    var bad = new user({    name: 'nope'  });    // good  function User(options) {    this.name = options.name;  }    var good = new User({    name: 'yup'  }); |

* Use a leading underscore \_ when naming private properties.

|  |
| --- |
| // bad  this.\_\_firstName\_\_ = 'Panda';  this.firstName\_ = 'Panda';    // good  this.\_firstName = 'Panda'; |

* When saving a reference to this use \_this.

|  |
| --- |
| // bad  function() {    var self = this;    return function() {      console.log(self);    };  }    // good  function() {    var \_this = this;    return function() {      console.log(\_this);    };  } |

* Name your functions. This is helpful for stack traces.

|  |
| --- |
| // bad  var log = function(msg) {    console.log(msg);  };    // good  var log = function log(msg) {    console.log(msg);  }; |

Linting

JSHint is a program that flags suspicious usage in programs written in JavaScript. Use the following configuration as a starting point and modify as needed. For example, if you are developing a client-side application, then node should be set to false.

|  |
| --- |
| {    "camelcase": true,    "forin": false,    "immed": true,    "latedef": true,    "newcap": false,    "nonew": true,    "strict": false,    "smarttabs": true,    "devel": false,    "node": true,    "unused": false,    "expr": true,    "quotmark": "double",    "globals": {      "requirejs": false,      "require": false,      "define": false,      "sinon": false,      "should": false,      "expect": false,      "describe": false,      "it": false,      "before": false,      "after": false,      "beforeEach": false,      "afterEach": false,      "suite": false,      "test": false,      "setup": false,      "teardown": false    }  } |

Best Practices

**Design Promise-based APIs over** **callbacks.**Asynchronous functionality should give and receive promises over callbacks to avoid what is titled "Callback Hell." Promises provide easier ways to keep code flat and readable where as callbacks generally create "nested triangles" which make debugging a nightmare.

**Avoid anonymous functions when possible.**  
JavaScript will include the name of functions in stack traces including those normally declared as anonymous functions. If you name them instead of keeping them anonymous, your stack traces will be much easier to follow.

Code Blocks

**Known Issues**

|  |
| --- |
| var test = obj["key1"] && obj["key2"];  // gets re-formatted as:  var test = obj.key1 && obj.key2]; |
| {"8880044": 1}  // gets re-formatted as  {8880044: 1} |