**Best Practices for JavaScript**

For reference you can use below link.

<https://code.tutsplus.com/tutorials/24-javascript-best-practices-for-beginners--net-5399>

<https://www.w3schools.com/js/js_best_practices.asp>

1. Use === instead of ==

EX: '' == '0' // false

0 == '' // true

0 == '0' // true

1. Always declare local variables.
2. It is good coding practice to put all declarations at the top of each script or function.

EX: // Declare at the beginning  
let firstName, lastName, price;

// Use later

firstName = "John";  
lastName = "Doe";

1. Don’t user short-hand

**Bad**: if (somethingExists)

x = false;

anotherFunctionCall();

**Better**: if (somethingExists) {

x = false;

anotherFunctionCall();

}

1. Utilize JS Lint :
   1. JSLint is a debugger its quickly scan any noticeable issues and errors in your code.
2. Comment your code.

EX: Single line comment: //

Multiline comment: /\*\*

Your comment here

\*/

1. Use let and const to create variables.

EX: const name = "John"; let age = 29

1. Avoid to create global variables.
2. Use shortcut notations:

**Bad**: If (v) {

let x = v;

} else {

let x = 10

}

**Better**: let x = v || 10

1. Use ternary operators instead of nested if else.
2. Don’t Use **new Object()**
   1. Use {} instead of new Object()
   2. Use "" instead of new String()
   3. Use 0 instead of new Number()
   4. Use false instead of new Boolean()
   5. Use [] instead of new Array()
   6. Use /()/ instead of new RegExp()
   7. Use function (){} instead of new Function()
3. Modularize code: Write smaller, generic helper function that fulfil one specific one task rather than catch-all methods, at a later stage you can also expose these functions when using the revealing module pattern to create an API to extend the main functionality.
4. Avoid to creating a lot of dependent code.
5. User async and await for the asynchronous execution.
6. Make a use of destructuring of object and array.
7. Beware of automatic type conversions
   1. Beware that number can accidentally be converted into strings or NaN (Not a Number).
   2. JavaScript is loosely typed. A variable can contain different data types, and a variable can change its data type.

EX: let x = "Hello";     // typeof x is a string

x = 5;               // changes typeof x to a number

1. Use Parameters Defaults
   1. If a function is called with a missing argument, the value of the missing argument is set to undefined.
   2. Undefined values can break your code. It is a good habit to assign default values to arguments.

EX: function myFunction(x, y) {  
  if (y === undefined) {  
    y = 0;  
  }  
}

ES5 allows default parameters in the function call:

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

1. Name your functions
   1. Name all functions, including closures and callbacks. Avoid anonymous functions. This is especially useful when profiling a node app. Naming all functions will allow you to easily understand what you're looking at when checking a memory snapshot.
2. Use naming conversions for variables, constants, functions and classes
   1. Use **lowerCamelCase** when naming constants, variables and functions and **UpperCamelCase** (capital first letter as well) when naming classes. This will help you to easily distinguish between plain variables/functions, and classes that require instantiation. Use descriptive names but try to keep them short.

EX: // for class name we use UpperCamelCase

class SomeClassExample {}

// for const names we use the const keyword and lowerCamelCase

const config = {

key: 'value'

};

// for variables and functions names we use lowerCamelCase

let someVariableExample = 'value';

function doSomething() {}

1. Require modules by folders, opposed to the files directly
   1. When developing a module/library in a folder, place an index.js file that exposes the module's internals so every consumer will pass through it. This serves as an 'interface' to your module and eases future changes without breaking the contract.

EX:

**Bad**: module.exports.SMSProvider = require('./SMSProvider/SMSProvider.js');

**Better**: module.exports.SMSProvider = require('./SMSProvider');

1. Separate your statements properly.
   1. No matter if you use semicolons or not to separate your statements, knowing the common pitfalls of improper line breaks or automatic semicolon insertion, will help you to eliminate regular syntax errors.

**Bad (throw exception)**: const m = new Map()

const a = [1, 2, 3]

[...m.values()].forEach(console.log)

**Better**: const items = [1, 2, 3]

items.forEach(console.log)

1. Start a codeblock’s curly braces on the same line.

**Bad**: function someFunction()

{

// code block

}

**Better**: function someFunction() {

// code block

}

1. Use Async Await, avoid callbacks
   1. Node 8 LTS now has full support for Async-await. This is a new way of dealing with asynchronous code which supersedes callbacks and promises. Async-await is non-blocking, and it makes asynchronous code look synchronous. The best gift you can give to your code is using async-await which provides a much more compact and familiar code syntax like try-catch.
2. Use arrow function expression
   1. Though it's recommended to use async-await and avoid function parameters when dealing with older APIs that accept promises or callbacks - arrow functions make the code structure more compact and keep the lexical context of the root function (i.e. this)

EX: const someFunction = () => {

// code block

}

**Best Practices for Node JS API**

<https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-cbe81b06f291>

1. Use nouns but not verbs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource | GET read | POST create | PUT update | DELETE |
| /cars | Returns a list of cars | Create a new car | Bulk update of cars | Delete all cars |
| /cars/711 | Returns a specific car | Method not allowed (405) | Updates a specific car | Deletes a specific car |

Do not use verbs:

/getAllCars

/createNewCar

/deleteAllRedCar

1. Use PUT, POST, DELETE methods instead of the GET method to alter the state.
2. Don’t mix up singular and plural nouns. Keep it simple and use only plural nouns for all resources.

EX: /cars instead of /car

/users instead of /user

1. Use HTTP header for serialization formats
   1. Both client and server, need to know which format is used for the communication. The format has to be specified in the HTTP-Header.

EX: Content-Type defines the request format.

Accept defines a list of acceptable response format.

1. Use HTTP header for serialization formats