JavaScript

25+ JavaScript Shorthand Coding Techniques

jQuery

Michael Wanyoike, **Sam Deering**
August 25, 2019

> **This really is a must read for any JavaScript developer**. I have written this guide to shorthand JavaScript coding techniques that I have picked up over the years. To help you understand what is going on, I have included the longhand versions to give some coding perspective.

> **August 25th, 2019**: This article was updated to add new shorthand tips based on the latest specifications. If you want to learn more about ES6 and beyond, check out **JavaScript: Novice to Ninja, 2nd Edition**.

1. The Ternary Operator

This is a great code saver when you want to write an `if..else` statement in just one line.

Longhand:

```
const x = 20;
let answer;

if (x > 10) {
    answer = "greater than 10";
} else {
    answer =  "less than 10";
```

```
}
```

Shorthand:

```
const answer = x > 10 ? "greater than 10" : "less than 10";
```

You can also nest your `if` statement like this:

```
const answer = x > 10 ? "greater than 10" : x < 5 ? "less than 5" : "between 5
```

2. Short-circuit Evaluation Shorthand

When assigning a variable value to another variable, you may want to ensure that the source variable is not null, undefined, or empty. You can either write a long `if` statement with multiple conditionals, or use a short-circuit evaluation.

Longhand:

```
if (variable1 !== null || variable1 !== undefined || variable1 !== '') {
     let variable2 = variable1;
}
```

Shorthand:

```
const variable2 = variable1  || 'new';
```

Don't believe me? Test it yourself (paste the following code in **es6console**):

```
let variable1;
let variable2 = variable1  || 'bar';
console.log(variable2 === 'bar'); // prints true

variable1 = 'foo';
```

```
variable2 = variable1  || 'bar';
console.log(variable2); // prints foo
```

Do note that if you set `variable1` to `false` or `0`, the value `bar` will be assigned.

3. Declaring Variables Shorthand

It's good practice to declare your variable assignments at the beginning of your functions. This shorthand method can save you lots of time and space when declaring multiple variables at the same time.

Longhand:

```
let x;
let y;
let z = 3;
```

Shorthand:

```
let x, y, z=3;
```

4. If Presence Shorthand

This might be trivial, but worth a mention. When doing "`if` checks", assignment operators can sometimes be omitted.

Longhand:

```
if (likeJavaScript === true)
```

Shorthand:

Shorthand:

```
if (likeJavaScript)
```

> **Note:** these two examples are not exactly equal, as the shorthand check will pass as long as `likeJavaScript` is a **truthy value**.

Here is another example. If `a` is NOT equal to true, then do something.

Longhand:

```
let a;
if ( a !== true ) {
// do something...
}
```

Shorthand:

```
let a;
if ( !a ) {
// do something...
}
```

5. JavaScript For Loop Shorthand

This little tip is really useful if you want plain JavaScript and don't want to rely on external libraries such as jQuery or lodash.

Longhand:

```
const fruits = ['mango', 'peach', 'banana'];
for (let i = 0; i < fruits.length; i++)
```

Shorthand:

```
for (let fruit of fruits)
```

If you just wanted to access the index, do:

```
for (let index in fruits)
```

This also works if you want to access keys in a literal object:

```
const obj = {continent: 'Africa', country: 'Kenya', city: 'Nairobi'}
for (let key in obj)
  console.log(key) // output: continent, country, city
```

Shorthand for Array.forEach:

```
function logArrayElements(element, index, array) {
  console.log("a[" + index + "] = " + element);
}
[2, 5, 9].forEach(logArrayElements);
// a[0] = 2
// a[1] = 5
// a[2] = 9
```

6. Short-circuit Evaluation

Instead of writing six lines of code to assign a default value if the intended parameter is null or undefined, we can simply use a short-circuit logical operator and accomplish the same thing with just one line of code.

Longhand:

```
let dbHost;
if (process.env.DB_HOST) {
  dbHost = process.env.DB_HOST;
} else {
  dbHost = 'localhost';
}
```

Shorthand:

```
const dbHost = process.env.DB_HOST || 'localhost';
```

7. Decimal Base Exponents

You may have seen this one around. It's essentially a fancy way to write numbers without the trailing zeros. For example, 1e7 essentially means 1 followed by 7 zeros. It represents a decimal base (which JavaScript interprets as a float type) equal to 10,000,000.

Longhand:

```
for (let i = 0; i < 10000; i++) {}
```

Shorthand:

```
for (let i = 0; i < 1e7; i++) {}

// All the below will evaluate to true
1e0 === 1;
1e1 === 10;
1e2 === 100;
1e3 === 1000;
1e4 === 10000;
1e5 === 100000;
```

8. Object Property Shorthand

Defining object literals in JavaScript makes life much easier. ES6 provides an even easier way of assigning properties to objects. If the variable name is the same as the object key, you can take advantage of the shorthand notation.

Longhand:

```
const x = 1920, y = 1080;
const obj = { x:x, y:y };
```

Shorthand:

```
const obj = { x, y };
```

9. Arrow Functions Shorthand

Classical functions are easy to read and write in their plain form, but they do tend to become a bit verbose and confusing once you start nesting them in other function calls.

Longhand:

```
function sayHello(name) {
  console.log('Hello', name);
}

setTimeout(function() {
  console.log('Loaded')
}, 2000);

list.forEach(function(item) {
  console.log(item);
});
```

Shorthand:

```
sayHello = name => console.log('Hello', name);

setTimeout(() => console.log('Loaded'), 2000);

list.forEach(item => console.log(item));
```

It's important to note that the value of `this` inside an arrow function is determined differently than for longhand functions, so the two examples are not strictly equivalent. See **this article on arrow function syntax** for more details.

10. Implicit Return Shorthand

Return is a keyword we use often to return the final result of a function. An arrow function with a single statement will implicitly return the result its evaluation (the function must omit the braces (`{}`) in order to omit the return keyword).

To return a multi-line statement (such as an object literal), it's necessary to use `()` instead of `{}` to wrap your function body. This ensures the code is evaluated as a single statement.

Longhand:

```
function calcCircumference(diameter) {
  return Math.PI * diameter
}
```

Shorthand:

```
calcCircumference = diameter => (
  Math.PI * diameter;
)
```

11. Default Parameter Values

11. Default Parameter Values

You can use the `if` statement to define default values for function parameters. In ES6, you can define the default values in the function declaration itself.

Longhand:

```
function volume(l, w, h) {
  if (w === undefined)
    w = 3;
  if (h === undefined)
    h = 4;
  return l * w * h;
}
```

Shorthand:

```
volume = (l, w = 3, h = 4 ) => (l * w * h);

volume(2) //output: 24
```

12. Template Literals

Aren't you tired of using `' + '` to concatenate multiple variables into a string? Isn't there a much easier way of doing this? If you are able to use ES6, then you are in luck. All you need to do is use is the backtick, and `${}` to enclose your variables.

Longhand:

```
const welcome = 'You have logged in as ' + first + ' ' + last + '.'

const db = 'http://' + host + ':' + port + '/' + database;
```

Shorthand:

```
const welcome = `You have logged in as ${first} ${last}`;

const db = `http://${host}:${port}/${database}`;
```

13. Destructuring Assignment Shorthand

If you are working with any popular web framework, there are high chances you will be using arrays or data in the form of object literals to pass information between components and APIs. Once the data object reaches a component, you'll need to unpack it.

Longhand:

```
const observable = require('mobx/observable');
const action = require('mobx/action');
const runInAction = require('mobx/runInAction');

const store = this.props.store;
const form = this.props.form;
const loading = this.props.loading;
const errors = this.props.errors;
const entity = this.props.entity;
```

Shorthand:

```
import { observable, action, runInAction } from 'mobx';

const { store, form, loading, errors, entity } = this.props;
```

You can even assign your own variable names:

```
const { store, form, loading, errors, entity:contact } = this.props;
```

14. Multi-line String Shorthand

14. Multi-line String Shorthand

If you have ever found yourself in need of writing multi-line strings in code, this is how you would write it:

Longhand:

```
const lorem = 'Lorem ipsum dolor sit amet, consectetur\n\t'
    + 'adipisicing elit, sed do eiusmod tempor incididunt\n\t'
    + 'ut labore et dolore magna aliqua. Ut enim ad minim\n\t'
    + 'veniam, quis nostrud exercitation ullamco laboris\n\t'
    + 'nisi ut aliquip ex ea commodo consequat. Duis aute\n\t'
    + 'irure dolor in reprehenderit in voluptate velit esse.\n\t'
```

But there is an easier way. Just use backticks.

Shorthand:

```
const lorem = `Lorem ipsum dolor sit amet, consectetur
    adipisicing elit, sed do eiusmod tempor incididunt
    ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris
    nisi ut aliquip ex ea commodo consequat. Duis aute
    irure dolor in reprehenderit in voluptate velit esse.`
```

15. Spread Operator Shorthand

The **spread operator**, introduced in ES6, has several use cases that make JavaScript code more efficient and fun to use. It can be used to replace certain array functions. The spread operator is simply a series of three dots.

Longhand

```
// joining arrays
const odd = [1, 3, 5];
const nums = [2, 4, 6].concat(odd);
```

```
const nums = [2 ,4 , 6].concat(odd);


// cloning arrays
const arr = [1, 2, 3, 4];
const arr2 = arr.slice()
```

Shorthand:

```
// joining arrays
const odd = [1, 3, 5 ];
const nums = [2 ,4 , 6, ...odd];
console.log(nums); // [ 2, 4, 6, 1, 3, 5 ]

// cloning arrays
const arr = [1, 2, 3, 4];
const arr2 = [...arr];
```

Unlike the `concat()` function, you can use the spread operator to insert an array anywhere inside another array.

```
const odd = [1, 3, 5 ];
const nums = [2, ...odd, 4 , 6];
```

You can also combine the spread operator with ES6 destructuring notation:

```
const { a, b, ...z } = { a: 1, b: 2, c: 3, d: 4 };
console.log(a) // 1
console.log(b) // 2
console.log(z) // { c: 3, d: 4 }
```

16. Mandatory Parameter Shorthand

By default, JavaScript will set function parameters to `undefined` if they are not passed a value. Some other languages will throw a warning or error. To enforce parameter assignment, you can use an `if` statement to throw an error if `undefined`, or you can take advantage of

the 'Mandatory parameter shorthand'.

Longhand:

```
function foo(bar) {
  if(bar === undefined) {
    throw new Error('Missing parameter!');
  }
  return bar;
}
```

Shorthand:

```
mandatory = () => {
  throw new Error('Missing parameter!');
}

foo = (bar = mandatory()) => {
  return bar;
}
```

17. Array.find Shorthand

If you have ever been tasked with writing a find function in plain JavaScript, you would probably have used a `for` loop. In ES6, a new array function named `find()` was introduced.

Longhand:

```
const pets = [
  { type: 'Dog', name: 'Max'},
  { type: 'Cat', name: 'Karl'},
  { type: 'Dog', name: 'Tommy'},
]

function findDog(name) {
  for(let i = 0; i<pets.length; ++i) {
    if(pets[i].type === 'Dog' && pets[i].name === name) {
      return pets[i];
```

```
    }

  }
}
```

Shorthand:

```
pet = pets.find(pet => pet.type ==='Dog' && pet.name === 'Tommy');
console.log(pet); // { type: 'Dog', name: 'Tommy' }
```

18. Object [key] Shorthand

Did you know that `Foo.bar` can also be written as `Foo['bar']`? At first, there doesn't seem to be a reason why you should write it like that. However, this notation gives you the building block for writing re-usable code.

Consider this simplified example of a validation function:

```
function validate(values) {
  if(!values.first)
    return false;
  if(!values.last)
    return false;
  return true;
}

console.log(validate({first:'Bruce',last:'Wayne'})); // true
```

This function does its job perfectly. However, consider a scenario where you have very many forms where you need to apply the validation but with different fields and rules. Wouldn't it be nice to build a generic validation function that can be configured at runtime?

Shorthand:

```
// object validation rules
```

```
const schema = {
  first: {
    required:true
  },
  last: {
    required:true
  }
}

// universal validation function
const validate = (schema, values) => {
  for(field in schema) {
    if(schema[field].required) {
      if(!values[field]) {
        return false;
      }
    }
  }
  return true;
}


console.log(validate(schema, {first:'Bruce'})); // false
console.log(validate(schema, {first:'Bruce',last:'Wayne'})); // true
```

Now we have a validate function we can reuse in all forms without needing to write a custom validation function for each.

19. Double Bitwise NOT Shorthand

Bitwise operators are one of those features you learn about in beginner JavaScript tutorials and you never get to implement them anywhere. Besides, who wants to work with ones and zeroes if you are not dealing with binary?

There is, however, a very practical use case for the Double Bitwise NOT operator. You can use it as a replacement for `Math.floor()`. The advantage of the Double Bitwise NOT operator is that it performs the same operation much faster. You can read more about Bitwise operators **here**.

. . .

Longhand:

```
Math.floor(4.9) === 4  //true
```

Shorthand:

```
~~4.9 === 4  //true
```

20. Exponent Power Shorthand

Shorthand for a Math exponent power function:

Longhand:

```
Math.pow(2,3); // 8
Math.pow(2,2); // 4
Math.pow(4,3); // 64
```

Shorthand:

```
2**3 // 8
2**4 // 4
4**3 // 64
```

21. Converting a String into a Number

There are times when your code receives data that comes in String format but needs to processed in Numerical format. It's not a big deal, we can perform a quick conversion.

Longhand:

```
const num1 = parseInt("100");
const num2 = parseFloat("100.01");
```

```
const num2 =  parseFloat("100.01");
```

Shorthand:

```
const num1 = +"100"; // converts to int data type
const num2 =  +"100.01"; // converts to float data type
```

22. Object Property Assignment

Consider the following piece of code:

```
let fname = { firstName : 'Black' };
let lname = { lastName : 'Panther'}
```

How would you merge them into one object? One way is to write a function that copies data from the second object onto the first one. Unfortunately, this might not be what you want — you may need to create an entirely new object without mutating any of the existing objects. The easiest way is to use the `Object.assign` function introduced in ES6:

```
let full_names = Object.assign(fname, lname);
```

You can also use the object destruction notation introduced in ES8:

```
let full_names = {...fname, ...lname};
```

There is no limit to the number of object properties you can merge. If you do have objects with identical property names, values will be overwritten in the order they were merged.

23. Bitwise IndexOf Shorthand

When performing a lookup using an array, the `indexOf()` function is used to retrieve the position of the item you are looking for. If the item is not found, the value `-1` is returned. In

JavaScript, `0` is considered 'falsy', while numbers greater or lesser than `0` are considered 'truthy'. As a result, one has to write the correct code like this.

Longhand:

```
if(arr.indexOf(item) > -1) { // Confirm item IS found

}

if(arr.indexOf(item) === -1) { // Confirm item IS NOT found

}
```

Shorthand:

```
if(~arr.indexOf(item)) { // Confirm item IS found

}

if(!~arr.indexOf(item)) { // Confirm item IS NOT found

}
```

The `bitwise(~)` operator will return a truthy value for anything but `-1`. Negating it is as simple as doing `!~`. Alternatively, we can also use the `includes()` function:

```
if(arr.includes(item)) { // Returns true if the item exists, false if it doesr

}
```

24. Object.entries()

This is a feature that was introduced in ES8 that allows you to convert a literal object into a key/value pair array. See the example below:

```
const credits = { producer: 'John', director: 'Jane', assistant: 'Peter' };
const arr = Object.entries(credits);
console.log(arr);

/** Output:
[ [ 'producer', 'John' ],
  [ 'director', 'Jane' ],
  [ 'assistant', 'Peter' ]
]
**/
```

25. Object.values()

This is also a new feature introduced in ES8 that performs a similar function to `Object.entries()`, but without the key part:

```
const credits = { producer: 'John', director: 'Jane', assistant: 'Peter' };
const arr = Object.values(credits);
console.log(arr);

/** Output:
[ 'John', 'Jane', 'Peter' ]
**/
```

26. Suggest One?

I really do love these and would love to find more, so please leave a comment if you know of one!

Share This Article

Michael Wanyoike

I write clean, readable and modular code. I love learning new technologies

that bring efficiencies and increased productivity to my workflow.

Sam Deering

Sam Deering has 15+ years of programming and website development experience. He was a website consultant at Console, ABC News, Flight Centre, Sapient Nitro, and the QLD Government and runs a tech blog with over 1 million views per month. Currently, Sam is the Founder of Crypto News, Australia.

jQuery

Up Next

JavaScript Refactoring Techniques: Specific to Generic Code

Paul Wilkins

Video: Shorthand if-else Conditionals with PHP

Lami Adabonyan

An Introduction to jQuery's Shorthand Ajax Methods

Aurelio De Rosa

js ipad zoom control techniques

Sam Deering