JavaScript Naming Conventions

OCTOBER 06, 2019 BY ROBIN WIERUCH - EDIT THIS POST

A JavaScript naming conventions introduction by example – which gives you the common sense when it comes to naming variables, functions, classes or components in JavaScript. No one is enforcing these naming convention rules, however, they are widely accepted as a standard in the JS community.

JAVASCRIPT NAMING CONVENTIONS: VARIABLES

JavaScript variables are case sensitive. Therefore, JavaScript variables with lowercase and uppercase characters are different:

```
var Name = 'Dennis Wieruch';

var NAME = 'Thomas Wieruch';

console.log(name);
// "Robin Wieruch"

console.log(Name);
// "Dennis Wieruch"

console.log(NAME);
// "Thomas Wieruch"
```

A **JavaScript variable should be self-descriptive**. It shouldn't be necessary to add a comment for additional documentation to the variable:

```
// bad
var value = 'Robin';

// bad
var val = 'Robin';

// good
var firstName = 'Robin';
```

Most often you will find JavaScript variables declared with a camelCase variable name with a leading lowercase character:

```
// bad
var firstname = 'Robin';

// bad
var first_name = 'Robin';

// bad
var FIRSTNAME = 'Robin';

// bad
var FIRST_NAME = 'Robin';

// good
var firstName = 'Robin';
```

There are exceptions for JavaScript constants, privates, and classes/components – which

A brief overview about the different case styles:

- camelCase (used in JS)
- PascalCase (used in JS)
- snake_case
- kebab-case

JAVASCRIPT NAMING CONVENTIONS: BOOLEAN

A prefix like *is*, *are*, or *has* helps every JavaScript developer to distinguish a boolean from another variable by just looking at it:

```
// bad
var visible = true;

// good
var isVisible = true;

// bad
var equal = false;

// good
var areEqual = false;

// bad
var encryption = true;

// good
var hasEncryption = true;
```

In contrast to strings and integers, you can see it as another *soft rule* for a JavaScript boolean naming convention besides being written in camel case.

JAVASCRIPT NAMING CONVENTIONS: FUNCTION

```
// bad
function name(firstName, lastName) {
  return `${firstName} ${lastName}`;
}

// good
function getName(firstName, lastName) {
  return `${firstName} ${lastName}`;
}
```

This verb as prefix can be anything (e.g. *get*, *fetch*, *push*, *apply*, *calculate*, *compute*, *post*). It's yet another *soft rule* to consider for having more self-descriptive JavaScript variables.

JAVASCRIPT NAMING CONVENTIONS: CLASS

A JavaScript class is declared with a PascalCase in contrast to other JavaScript data structures:

```
class SoftwareDeveloper {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}

var me = new SoftwareDeveloper('Robin', 'Wieruch');
```

Every time a JavaScript constructor is called to instantiate a new instance of a class, the name of the class should appear in Pascal Case, because the class has been declared with Pascal Case in the first place.

JAVASCRIPT NAMING CONVENTIONS: COMPONENT

Components are not everywhere in JavaScript, but commonly found in frontend frameworks like React. Since a component is kinda instantiated – but appended to the DOM

```
// bad
function userProfile(user) {
  return (
    <div>
      <span>First Name: {user.firstName}</span>
      <span>Last Name: {user.lastName}</span>
    </div>
  );
}

// good
function UserProfile(user) {
  return (
    <div>
      <span>First Name: {user.firstName}</span>
      <span>Last Name: {user.lastName}</span>
    </div>
  );
}
```

When a component gets used, it distinguishes itself from native HTML and web components, because its first letter is always written in uppercase.

```
<div>
  <UserProfile
    user={{ firstName: 'Robin', lastName: 'Wieruch' }}
  />
</div>
```

JAVASCRIPT NAMING CONVENTIONS: METHODS

Identical to JavaScript functions, a method on a JavaScript class is declared with camelCase:

```
class SoftwareDeveloper {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getName() {
}

var me = new SoftwareDeveloper('Robin', 'Wieruch');

console.log(me.getName());
// "Robin Wieruch"
```

Here the same rules as for JavaScript functions apply – e.g. adding a verb as a prefix –, for making the method name more self-descriptive.

JAVASCRIPT NAMING CONVENTIONS: PRIVATE

Rarely you will find an underscore (_) in front of a variable/function/method in JavaScript. If you see one, it is *intended* to be *private*. Even though it cannot be really enforced by JavaScript, declaring something as private tells us about how it should be used or how it should not be used.

For instance, a private method in a class should only be used internally by the class, but should be avoided to be used on the instance of the class:

```
class SoftwareDeveloper {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.name = _getName(firstName, lastName);
  }

  _getName(firstName, lastName) {
    return `${firstName} ${lastName}`;
  }
}

var me = new SoftwareDeveloper('Robin', 'Wieruch');

// good
var name = me.name;
console.log(name);
// "Robin Wieruch"

// bad
name = me._getName(me.firstName, me.lastName);
console.log(name);
// "Robin Wieruch"
```

A private variable/function can occur in a JavaScript file as well. This could mean that the variable/function shouldn't be used outside of this file but only internally to compute further business logic for other functions within the same file..

JAVASCRIPT NAMING CONVENTIONS: CONSTANT

Last but not least, there are constants – intended to be non-changing variables – in JavaScript which are written in capital letters (UPPERCASE):

```
var SECONDS = 60;
var MINUTES = 60;
var HOURS = 24;

var DAY = SECONDS * MINUTES * HOURS;
```

If a variable has more than one word in its variable declaration name, it makes use of an underscore (_):

```
var DAYS_UNTIL_TOMORROW = 1;
```

Usually JavaScript constants are defined at the top of a JavaScript file. As hinted before, no one enforces one to not change the variable here, except a const declaration of the variable for primitive data structures, but it's capitalized naming suggests avoiding it.

JAVASCRIPT NAMING CONVENTIONS: GLOBAL VARIABLE

A JavaScript variable is globally defined, if all its context has access to it. Often the context is defined by the JavaScript file where the variable is declared/defined in, but in smaller JavaScript projects it may be the entire project. There are no special naming conventions for global JavaScript variables.

- A global JavaScript variable is declared at the top of a project/file.

- A global JavaScript variable is written in UPPERCASE if it is immutable.

JAVASCRIPT NAMING CONVENTIONS: UNDERSCORE

So what about the underscore and dash in JavaScript variable namings? Since camelCase and PascalCase are primarily considered in JS, you have seen that the underscore is only rarely used for private variables or constants. Occasionally you will find underscores when getting information from third-parties like databases or APIs. Another scenario where you might see an underscore are unused function parameters, but don't worry about these yet if you haven't seen them out there ;-)

JAVASCRIPT NAMING CONVENTIONS: DASH

A dash in a JavaScript variable isn't common sense as well. It just makes things more difficult; like using them in an object:

```
// bad
var person = {
  'first-name': 'Robin',
  'last-name': 'Wieruch',
};

var firstName = person['first-name'];

// good
var person = {
  firstName: 'Robin',
  lastName: 'Wieruch',
};

var firstName = person.firstName;
```

It's even not possible to use a dash directly for a variable declaration:

```
var first-name = 'Robin';
// Uncaught SyntaxError: Unexpected token '-'
```

That's why it's better to avoid them.

JAVASCRIPT NAMING CONVENTIONS: FILES

There are two strategies of naming files in JavaScript: PascalCase and kebab-case. In JavaScript frontend applications, you will often see PascalCase for naming components (e.g. React components).

```
- components/
--- user/
----- UserProfile.js
----- UserList.js
----- UserItem.js
--- ui/
----- Dialog.js
----- Dropdown.js
----- Table.js
```

In contrast, in JavaScript backend application, kebab-case is the common sense:

```
- routing/
--- user-route.js
--- messages-route.js
```

You will also see camelCase namings, but similar to PascalCase (sorry frontend applications), there is a risk that operating systems are handling them differently which may lead to bugs. That's why sticking to kebab-case should be the norm for file names in JavaScript.

. . .

If you want to learn more about JavaScript code style and formatting, which isn't discussed here for the sake of naming conventions, you should definitely check out ESLint and Prettier for JavaScript.

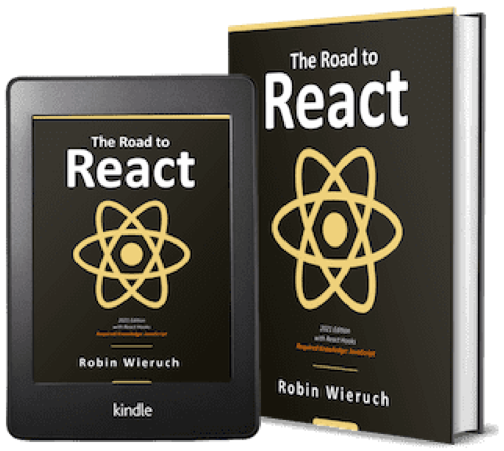
Show Comments

JAVASCRIPT CLOSURE BY EXAMPLE

Eventually you will come across the concept of a JavaScript Closure. I want to give you a step by step walkthrough on how to implement a JavaScript Closure. Along the way, you will find out yourself...

JAVASCRIPT VARIABLE TUTORIAL FOR BEGINNERS

In every programming language, you will find variables. This also holds true for JavaScript. Essentially variables are used to carry information. Without them, it wouldn't be really possible to code...



rwieruch ABOUT HIRE BLOG COURSES RSS

GET THE BOOK >

Get it on Amazon.

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

✓ Join 50.000+ Developers

✓ Learn Web Development

✓ Learn JavaScript

✓ Access Tutorials, eBooks and Courses

✓ Personal Development as a Software Engineer

SUBSCRIBE

View our Privacy Policy.