Variables

Most of the time, a JavaScript application needs to work with information. Here are two examples:

1. An online shop – the information might include goods being sold and a shopping cart.
2. A chat application – the information might include users, messages, and much more.

Variables are used to store this information.

A variable

A variable is a "named storage" for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the `let` keyword.

The statement below creates (in other words: *declares*) a variable with the name "message":

```
let message;
```

Now, we can put some data into it by using the assignment operator `=`:

```
let message;

message = 'Hello'; // store the string
```

The string is now saved into the memory area associated with the variable. We can access it using the variable name:

```
let message;
message = 'Hello!';

alert(message); // shows the variable content
```

To be concise, we can combine the variable declaration and assignment into a single line:

```
let message = 'Hello!'; // define the variable and assign the value

alert(message); // Hello!
```

We can also declare multiple variables in one line:

```
let user = 'John', age = 25, message = 'Hello';
```

That might seem shorter, but we don't recommend it. For the sake of better readability, please use a single line per variable.

The multiline variant is a bit longer, but easier to read:

```
 let user = 'John';
let age = 25;
let message = 'Hello';
```

Some people also define multiple variables in this multiline style:

```
let user = 'John',
  age = 25,
  message = 'Hello';
```

…Or even in the "comma-first" style:

```
let user = 'John'
  , age = 25
  , message = 'Hello';
```

Technically, all these variants do the same thing. So, it's a matter of personal taste and aesthetics.

> ## `var` instead of `let`
>
> In older scripts, you may also find another keyword: `var` instead of `let`:
>
> ```
> var message = 'Hello';
> ```
>
> The `var` keyword is *almost* the same as `let`. It also declares a variable, but in a slightly different, "old-school" way.
>
> There are subtle differences between `let` and `var`, but they do not matter for us yet. We'll cover them in detail in the chapter The old "var".

A real-life analogy

We can easily grasp the concept of a "variable" if we imagine it as a "box" for data, with a uniquely-named sticker on it.

For instance, the variable `message` can be imagined as a box labeled `"message"` with the value `"Hello!"` in it:
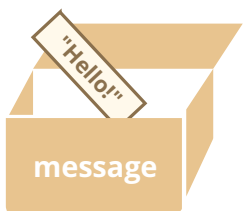
We can put any value in the box.

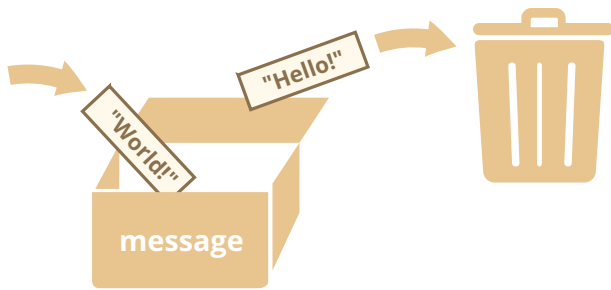We can also change it as many times as we want:

```
 let message;

message = 'Hello!';

message = 'World!'; // value changed

alert(message);
```

When the value is changed, the old data is removed from the variable:

We can also declare two variables and copy data from one into the other.

```
let hello = 'Hello world!';

let message;

// copy 'Hello world' from hello into message
message = hello;

// now two variables hold the same data
alert(hello); // Hello world!
alert(message); // Hello world!
```

Declaring twice triggers an error

A variable should be declared only once.

A repeated declaration of the same variable is an error:

```
let message = "This";

// repeated 'let' leads to an error
let message = "That"; // SyntaxError: 'message' has already been declared
```

So, we should declare a variable once and then refer to it without `let`.

Functional languages

It's interesting to note that there exist functional programming languages, like Scala or Erlang that forbid changing variable values.

In such languages, once the value is stored "in the box", it's there forever. If we need to store something else, the language forces us to create a new box (declare a new variable). We can't reuse the old one.

Though it may seem a little odd at first sight, these languages are quite capable of serious development. More than that, there are areas like parallel computations where this limitation confers certain benefits. Studying such a language (even if you're not planning to use it soon) is recommended to broaden the mind.

Variable naming

There are two limitations on variable names in JavaScript:

1. The name must contain only letters, digits, or the symbols `$` and `_`.
2. The first character must not be a digit.

Examples of valid names:

```
let userName;
let test123;
```

When the name contains multiple words, camelCase is commonly used. That is: words go one after another, each word except first starting with a capital letter: `myVeryLongName`.

What's interesting – the dollar sign `'$'` and the underscore `'_'` can also be used in names. They are regular symbols, just like letters, without any special meaning.

These names are valid:

```
let $ = 1; // declared a variable with the name "$"
let _ = 2; // and now a variable with the name "_"

alert($ + _); // 3
```

Examples of incorrect variable names:

```
let 1a; // cannot start with a digit

let my-name; // hyphens '-' aren't allowed in the name
```

> **Case matters**
>
> Variables named `apple` and `AppLE` are two different variables.

> **Non-Latin letters are allowed, but not recommended**
>
> It is possible to use any language, including cyrillic letters or even hieroglyphs, like this:
>
> ```
> let имя = '...';
> let 我 = '...';
> ```
>
> Technically, there is no error here. Such names are allowed, but there is an international convention to use English in variable names. Even if we're writing a small script, it may have a long life ahead. People from other countries may need to read it some time.

> **Reserved names**
>
> There is a list of reserved words, which cannot be used as variable names because they are used by the language itself.
>
> For example: `let`, `class`, `return`, and `function` are reserved.
>
> The code below gives a syntax error:
>
> ```
> let let = 5; // can't name a variable "let", error!
> let return = 5; // also can't name it "return", error!
> ```

> **An assignment without `use strict`**
>
> Normally, we need to define a variable before using it. But in the old times, it was technically possible to create a variable by a mere assignment of the value without using `let`. This still works now if we don't put `use strict` in our scripts to maintain compatibility with old scripts.
>
> ```
> // note: no "use strict" in this example
>
> num = 5; // the variable "num" is created if it didn't exist
>
> alert(num); // 5
> ```

This is a bad practice and would cause an error in strict mode:

```
"use strict";

num = 5; // error: num is not defined
```

Constants

To declare a constant (unchanging) variable, use `const` instead of `let` :

```
const myBirthday = '18.04.1982';
```

Variables declared using `const` are called "constants". They cannot be reassigned. An attempt to do so would cause an error:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

When a programmer is sure that a variable will never change, they can declare it with `const` to guarantee and clearly communicate that fact to everyone.

Uppercase constants

There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution.

Such constants are named using capital letters and underscores.

For instance, let's make constants for colors in so-called "web" (hexadecimal) format:

```
 const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Benefits:

- `COLOR_ORANGE` is much easier to remember than `"#FF7F00"` .
- It is much easier to mistype `"#FF7F00"` than `COLOR_ORANGE` .
- When reading the code, `COLOR_ORANGE` is much more meaningful than `#FF7F00` .

When should we use capitals for a constant and when should we name it normally? Let's make that clear.

Being a "constant" just means that a variable's value never changes. But there are constants that are known prior to execution (like a hexadecimal value for red) and there are constants that are *calculated* in run-time, during the execution, but do not change after their initial assignment.

For instance:

```
const pageLoadTime = /* time taken by a webpage to load */;
```

The value of `pageLoadTime` is not known prior to the page load, so it's named normally. But it's still a constant because it doesn't change after assignment.

In other words, capital-named constants are only used as aliases for "hard-coded" values.

Name things right

Talking about variables, there's one more extremely important thing.

A variable name should have a clean, obvious meaning, describing the data that it stores.

Variable naming is one of the most important and complex skills in programming. A quick glance at variable names can reveal which code was written by a beginner versus an experienced developer.

In a real project, most of the time is spent modifying and extending an existing code base rather than writing something completely separate from scratch. When we return to some code after doing something else for a while, it's much easier to find information that is well-labeled. Or, in other words, when the variables have good names.

Please spend time thinking about the right name for a variable before declaring it. Doing so will repay you handsomely.

Some good-to-follow rules are:

- Use human-readable names like `userName` or `shoppingCart`.
- Stay away from abbreviations or short names like `a`, `b`, `c`, unless you really know what you're doing.
- Make names maximally descriptive and concise. Examples of bad names are `data` and `value`. Such names say nothing. It's only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
- Agree on terms within your team and in your own mind. If a site visitor is called a "user" then we should name related variables `currentUser` or `newUser` instead of `currentVisitor` or `newManInTown`.

Sounds simple? Indeed it is, but creating descriptive and concise variable names in practice is not. Go for it.

> **Reuse or create?**
>
> And the last note. There are some lazy programmers who, instead of declaring new variables, tend to reuse existing ones.
>
> As a result, their variables are like boxes into which people throw different things without changing their stickers. What's inside the box now? Who knows? We need to come closer and check.
>
> Such programmers save a little bit on variable declaration but lose ten times more on debugging.
>
> An extra variable is good, not evil.
>
> Modern JavaScript minifiers and browsers optimize code well enough, so it won't create performance issues. Using different variables for different values can even help the engine optimize your code.

Summary

We can declare variables to store data by using the `var`, `let`, or `const` keywords.

- `let` – is a modern variable declaration.
- `var` – is an old-school variable declaration. Normally we don't use it at all, but we'll cover subtle differences from `let` in the chapter The old "var", just in case you need them.
- `const` – is like `let`, but the value of the variable can't be changed.

Variables should be named in a way that allows us to easily understand what's inside them.

Tasks

Working with variables

importance: 2

1. Declare two variables: `admin` and `name`.
2. Assign the value `"John"` to `name`.
3. Copy the value from `name` to `admin`.
4. Show the value of `admin` using `alert` (must output "John").

Giving the right name

importance: 3

1. Create a variable with the name of our planet. How would you name such a variable?
2. Create a variable to store the name of a current visitor to a website. How would you name that variable?

Uppercase const?

importance: 4

Examine the following code:

```
const birthday = '18.04.1982';

const age = someCode(birthday);
```

Here we have a constant `birthday` date and the `age` is calculated from `birthday` with the help of some code (it is not provided for shortness, and because details don't matter here).

Would it be right to use upper case for `birthday`? For `age`? Or even for both?

```
const BIRTHDAY = '18.04.1982'; // make uppercase?

const AGE = someCode(BIRTHDAY); // make uppercase?
```

Previous lesson	Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)

about the project

contact us

terms of usage