JavaScript Intro…

Maktab Sharif Front-End Bootcamp
Winter - 2018
Alireza Riahi

JavaScript Introduction

JavaScript Programs

Mostly, JavaScript runs in your web browser alongside HTML and CSS, and can be added to any web page using a script tag. The script element can either contain JavaScript directly (internal) or link to an external resource via a src attribute (external).

A browser then runs JavaScript line-by-line, starting at the top of the file or script element and finishing at the bottom (unless you tell it to go elsewhere).

Internal Scripts

JavaScript programs can be inserted in any part of an HTML document with the help of the `<script>` tag.

```
<!DOCTYPE HTML>
<html>
<body>
  <p>Before the script...</p>
  <script>
    alert( 'Salam JavaScript!' );
  </script>
  <p>...After the script.</p>
</body>
</html>
```

JavaScript Introduction

External Scripts

An external JavaScript resource is a text file with a `.js` extension. To add a JavaScript file to your page, you just need to use a script tag with a `src` attribute pointing to the file. So, if your file was called `script.js` and sat in the `/path/to/` directory as your HTML file, your script element would look like this:

```
<script src="/path/to/script.js"></script>
```

Note:

The old standard HTML4 required a script to have a type. Usually it was `type="text/javascript"`. The modern HTML standard assumes this `type` by default. No attribute is required.

As a rule, only the simplest scripts are put into `HTML`. More complex ones reside in separate files. The benefit of a separate file is that the browser will download it and then store in its `cache`. After this, other pages that want the same script will take it from the cache instead of downloading it. So the file is actually downloaded only once. That saves traffic and makes pages faster.

JavaScript Code Structure

Statements:

Statements are syntax constructs and commands that perform actions. We' ve already seen a statement `alert('Salam JavaScript!')`, which shows the message. We can have as many statements in the code as we want. Another statement can be separated with a semicolon.

comments:

As time goes on, the program becomes more and more complex. It becomes necessary to add comments which describe what happens and why. Comments can be put into any place of the script. They don' t affect the execution, because the engine simply ignores them.

One-line comments start with two forward slash characters `//`.

Multiline comments start with a forward slash and an asterisk `/*` and end with an asterisk and a forward slash `*/`.

```
/* Commenting out the code
alert('Don't Show!!!');
*/
alert('Salam');
alert('Java Script'); // This comment follows the statement
```

JavaScript Variables

Variables and Data

Storing data so we can use it later is one of the most important things when writing code. Fortunately, JavaScript can do this!

Variables is declaring for store some data! its meaning variables have data! So in programming language variables datas called value!

Each variable has a name. We call and use variables with their names.

So now we know a variable has a name and a value!

Declaration is declaring a variable to exist. To declare a variable in JavaScript, we need to use the `let` keyword. The statement below declare (in other words: create or defines) a variable with the name message:

```
let message;
```

JavaScript Variables

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

```
let message;

message = 'Salam'; // 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 = 'Salam!';
alert(message); // shows the variable content
```

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

```
let message = 'Salam!'; // define the variable and assign the value
alert(message); // Salam!
```

We can also declare multiple variables in one line:

```
let user = 'Alireza', age = 30, message = 'Salam!';
```

That might seem shorter, but it’ s not recommended. 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 = 'Alireza';
let age = 30;
let message = 'Salam!';
```

JavaScript Variables

Variables and Data

There are two limitations for a variable name in JavaScript:

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

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

```
let $ = 1; // declared a variable with the name "$"
let _ = 2; // and now a variable with the name "_"
alert($ + _); // 3
```

Note:

Java script is case sensitive. That means variables named `apple` and `Apple` are two different variables.

JavaScript Variables

Constant

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

```
const myBirthday = 1988;
```

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

```
const myBirthday = 1988;
myBirthday = 1992; // error, can't reassign the constant!
```

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.

```
const MY_BIRTHDAY = 1988;
```

JavaScript Variables

Naming

Variable naming is one of the most important and complex skills in programming. A quick glance at variable names can reveal which code is written by a beginner and which by an experienced developer. In a real project, most of the time is spent on modifying and extending the existing code base, rather than writing something completely separate from scratch. And when we return to the code after some time of doing something else, it's much easier to find information that is well-labeled. Or, in other words, when the variables have good names. Spend some time thinking about the right name for a variable before declaring it. This will repay you a lot.

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 the name maximally descriptive and concise. Examples of bad names are `data` and `value`. Such a name says nothing. It is only ok to use them if it's exceptionally obvious from the context which data or value is meant.
- 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 like `currentUser` or `newUser`, but not `currentVisitor` or a `newManInTown`.

JavaScript Data types

A variable in JavaScript can contain any data. A variable can at one moment be a string and later receive a numeric value:

```
let name = "Hasan";
name = 123456;
```

Programming languages that allow such things are called dynamically typed, meaning that there are data types, but variables are not bound to any of them.

Number

The number type serves both for integer and floating point numbers.

```
let num = 123;
num = 12.345;
```

Besides regular numbers, there are so-called "special numeric values" which also belong to that type: `Infinity`, `-Infinity` and `NaN`.

```
alert( 1 / 0 ); // Infinity
alert( "not a number" / 2 ); // NaN, such division is erroneous
```

JavaScript Data types

String

A string in JavaScript must be quoted. In JavaScript, there are 3 types of quotes.

1. Double quotes: `"Hello"`.
2. Single quotes: `'Hello'`.
3. Backticks: `` `Hello` ``.

Double and single quotes are "simple" quotes. There's no difference between them in JavaScript. Backticks are "extended functionality" quotes. They allow us to embed variables and expressions into a string by wrapping them in `${…}`, for example:

```
let name = "Mohammad";
// embed a variable
alert( `Salam bar ${name}!` ); // Salam bar Mohammad!
// embed an expression
alert( `the result is ${1 + 2}` ); // the result is 3
```

Note:

There is no character type. In some languages, there is a special "character" type for a single character. For example, in the C language and in Java it is `char`. In JavaScript, there is no such type. There's only one type: `string`. A string may consist of only one character or many of them.

JavaScript Data types

Boolean

The boolean type has only two values: `true` and `false`. This type is commonly used to store yes/no values: `true` means "yes, correct", and `false` means "no, incorrect".

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
```

Boolean values also come as a result of comparisons:

```
let isGreater = 4 > 1;
alert( isGreater ); // true (the comparison result is "yes")
```

Null

The special `null` value does not belong to any type of those described above. It forms a separate type of its own, which contains only the `null` value. In JavaScript `null` is not a "reference to a non-existing object" or a "null pointer" like in some other languages. It's just a special value which has the sense of "nothing", "empty" or "value unknown".

JavaScript Data types

Undefined

The special value `undefined` stands apart. It makes a type of its own, just like `null`. The meaning of `undefined` is "value is not assigned". If a variable is declared, but not assigned, then its value is exactly `undefined`:

```
let name;
alert(name); // shows "undefined"
```

Technically, it is possible to assign `undefined` to any variable, But it's not recommended to do that. Normally, we use `null` to write an "empty" or an "unknown" value into the variable, and `undefined` is only used for checks, to see if the variable is assigned or similar.

typeof

The `typeof` operator returns the type of the argument. It's useful when we want to process values of different types differently, or just want to make a quick check. It supports two forms of syntax:

1. As an operator: `typeof someVariable`.
2. Function style: `typeof(someVariable)`.

In other words, it works both with parentheses or without them. The result is the same.

JavaScript Data types

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof true // "boolean"
typeof "ali" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

Note:

We learn about symbol, object and functions later…

JavaScript Type Conversions

Most of the time, operators and functions automatically convert a value to the right type. That's called "type conversion". For example, `alert` automatically converts any value to a string to show it. Mathematical operations convert values to numbers. There are also cases when we need to explicitly convert a value to put things right.

ToString:

String conversion happens when we need the string form of a value. We can also use a call `String(value)` function for that:

```
let value = true;
alert(typeof value); // boolean
value = String(value); // now value is a string "true"
alert(typeof value); // string
```

String conversion is mostly obvious. A `false` becomes `"false"`, `null` becomes `"null"` etc.

JavaScript Type Conversions

ToNumber:

Numeric conversion happens in mathematical functions and expressions automatically. For example, when division `/` is applied to non-numbers:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

We can use a `Number(value)` function to explicitly convert a value:

```
let value = "123";
alert(typeof value); // string
value = Number(value); // becomes a number 123
alert(typeof value); // number
```

If the string is not a valid number, the result of such conversion is `NaN`, for instance:

```
alert(Number(“You can’t convert me!”)); // NaN
alert(Number(null)); // 0
alert(Number(undefined)); // NaN
alert(Number(true)); // 1
alert(Number(false)); // 0
```

Note:

Almost all mathematical operations convert values to numbers. With a notable exception of the addition `+`. If one of the added values is a string, then another one is also converted to a string. Then it joins them:

```
alert( 1 + '2' ); // '12' (string to the right)
alert( '1' + 2 ); // '12' (string to the left)
```

JavaScript Type Conversions

ToBoolean:

It happens in logical operations (later we' ll meet condition tests and other kinds of them), but also can be performed manually with the call of `Boolean(value)`. The conversion rule:

- Values that are intuitively "empty", like `0`, an empty string, `null`, `undefined` and `NaN` become `false`.
- Other values become `true`.

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false
alert( Boolean("salam") ); // true
alert( Boolean("") ); // false
```

Note:

The string with zero `"0"` is `true` all mathematical operations convert values to numbers.

Some languages (namely PHP) treat `"0"` as `false`. But in JavaScript a non-empty string is always `true`.

```
alert( Boolean("0") ); // true
alert( Boolean(" ") ); // spaces, also true (any non-empty string
is true)
```

JavaScript Practice

Some Practice:

1- Search about web catch…

2- What happen if you set src in script tag and write java script code inside this?

3- What happen if you use neatest multiline comments?

```
/*
  /* nested comment ?!? */
*/
```

4- What different between `let` and `var` ?

5- what is FP (Functional Programming)?

6- why you can’t use `let`, `function`, `return` to variable name’s?

7- What different between `-true` and `!true`?

JavaScript Operators

Operand:

An operand – is what operators are applied to. For instance in multiplication `5 * 2` there are two operands: the left operand is `5`, and the right operand is `2`. Sometimes people say arguments instead of operands.

Unary:

An operator is unary if it has a single operand. For example, the unary negation `-` reverses the sign of the number:

```
let x = 1;
x = -x;
```

binary:

An operator is binary if it has two operands. The same minus exists in the binary form as well:

```
let x = 1, y = 3;
alert( y - x );
```

JavaScript Operators

Strings concatenation, binary + :

The binary + is applied to strings, it merges (concatenates) them:

```
let s = "my" + "string";
alert(s); // mystring
```

Note that if any of the operands is a string, then the other one is converted to a string too. For example:

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

See, it doesn’ t matter whether the first operand is a string or the second one. The rule is simple: if either operand is a string, then convert the other one into a string as well.

String concatenation and conversion is a special feature of the binary plus +. Other arithmetic operators work only with numbers. They always convert their operands to numbers.

```
alert( 2 - '1' ); // 1
alert( '6' / '2' ); // 3
```

JavaScript Operators

Numeric conversion, unary + :

The plus + exists in two forms. The binary form that we used above and the unary form. The unary plus or, in other words, the plus operator + applied to a single value, doesn’ t do anything with numbers, but if the operand is not a number, then it is converted into it.

```
// No effect on numbers
let x = 1;
alert( +x ); // 1
let y = -2;
alert( +y ); // -2
// Converts non-numbers
alert( +true ); // 1
alert( +"" );   // 0
```

It actually does the same as `Number(...)`, but is shorter.

```
let apples = "2";
let oranges = "3";
alert( apples + oranges ); // "23", the binary plus concatenates strings
// both values converted to numbers before the binary plus
alert( +apples + +oranges ); // 5
// the longer variant
// alert( Number(apples) + Number(oranges) ); // 5
```

From a mathematician’ s standpoint the abundance of pluses may seem strange. But from a programmer’ s standpoint, there’ s nothing special: unary pluses are applied first, they convert strings to numbers, and then the binary plus sums them up.

JavaScript Practice

Some Practice:

1- `alert(2 + 2 + '1' ); // ?`

2- `alert('1' + 2 + 2); // ?`

Finished

for deep information you can visit

www.tutorialspoint.com

www.w3schools.com

Alireza Riahi
ali.r.riahi@gmail.com

Maktab Sharif - Winter 2018