JavaScript

JavaScript is a front-end script language. As a front-end language it does not support reading or writing to the client persistent memory (hard disk), except for the cookies.

JavaScript is not related to Java. In the past, both languages belonged to Sun, and since Java was very popular, Sun named its script using the word Java to leverage it from the marketing point of view.

JavaScript code is written between the tags `<script>...</script>` which are equivalent to `<script type="text/javascript">...</script>`.

JavaScript code can be added in the **head** or in the **body** of the **html** document.

Typically, scripts included in the head section will not affect the page. They are used to define functions that are called in the body section.

By default, the script is fetched when the HTML parser reaches the script element and is executed immediately, before the browser continues parsing the page. For large HTML documents and large JavaScript files, the result is that the page takes much longer to render.

The script may also be in separate file. In this case, it is included (in the **head** or **body**) by `<script src="path/file.js"></script>` where *path* is the absolute or relative path to the file containing the script and *file* is the filename of the file containing the script.

There must be no code between opening tag `<script src="path/file.js">` and closing tag `</script>`

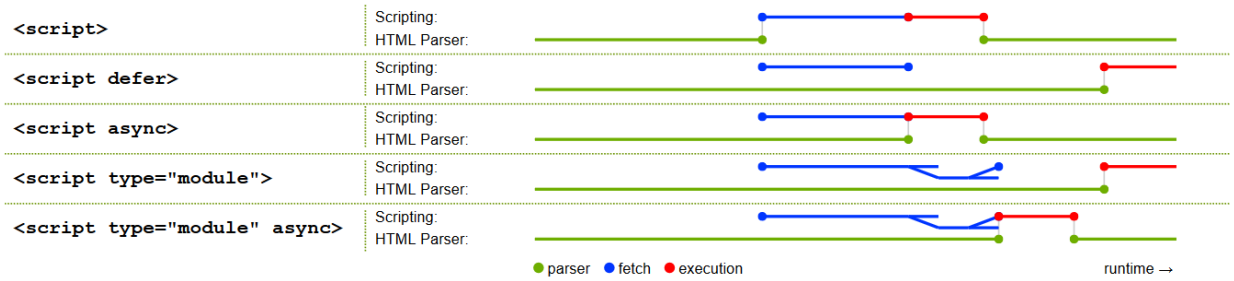
Scripts included from a separate file are called **external scripts**. Scripts whose codes are directly in the html file are called **inline scripts**.

Typically, when scripts are long, it is a good practice to code them in a separate file and include them as external scripts.

Advanced remark:

For external scripts, the order of fetch and execution can be changed by adding the **defer** or **async** to the script tag.

1. If neither **defer** nor **async** is present, then when the parser reaches the external script element, parser will stop parsing the page, the script will be fetched, and once it is fetched, it will be immediately executed. Then the parser will continue to parse the page. This is the default behaviour.
2. If **defer** is present, then when the parser reaches the external script element, the script is fetched in background while the parser continues to parse the page. The script is executed only after the page is completely parsed (but before the page is loaded).
3. If **async** is present, then when the parser reaches the external script element, the script is fetched in background while the parser continues to parse the page. Once the script is fetched, the parser stops and the script is executed, and then the parser continue to parse the page.
4. If both **async** and **defer** are present, it behaves as **async**, and if the browser (old browsers) don't support **async** then it behaves as **defer**.



For more information:
https://gist.github.com/jakub-g/385ee6b41085303a53ad92c7c8afd7a6

JavaScript Programming Language

Basic data type:

JavaScript has three types of data,

- Numbers : 123, 32.32
- Strings : “Mary”, “Today is Monday”, “123”
- Boolean : true, false

Note: All the numbers are considered as ‘floating point’.

The **strings** can be represented in several different ways:

"Every good boy deserves fudge"

'The quick brown fox jumps over the lazy dog'

'The doctor said "Today is your lucky day!" '

"I'm going to be happy when you give me the news!"

It is also possible to use the escape character (\), as in:

'The doctor said "I\'m pleased to announce that it\'s a girl!" '

"The doctor said \"I'm pleased to announce that it's a girl!\" "

Some other common escape sequences are \t to embed a tab and \n to embed a new line. You can also use \uHHHH where HHHH is a 4-digit hexadecimal code to embed a Unicode character.

You can use the plus sign (+) to represent string concatenation. The following is an example in which several strings are concatenated to produce one large string:

```
'Hickory Dickory Dock.' + "The mouse ran up the clock." +
'The clock struck one...'
```

There are several methods that apply to strings. For instance:

```
"Mary".toUpperCase()  returns  "MARY"
"Mary".toLowerCase()  returns  "mary"
"Mary".bold()  returns  "Mary"
"Mary".fontsize(14px)  returns  the string "Mary" in font size 14px.
"Mary".fontcolor("blue")  returns  the string "Mary" in font color blue.
"Mary".link("https://www.w3schools.com/js" )  returns "Mary" as  hyperlink.
```

There are other methods such as: .small() .strike().

Strings written with single or double quotes behave very much the same — the only difference is in which type of quote you need to escape inside of them.

Backtick-quoted strings, usually called template literals, can do a few more tricks. Apart from being able to span lines, they can also embed other values.

```
`half of 100 is ${100 / 2}`
```

When you write something inside ${} in a template literal, its result will be computed, converted to a string, and included at that position. The example produces “half of 100 is 50”.

The **typeof** operator

The typeof operator requires a single operand and returns a string that indicates the operand's type, as follows:

```
typeof 'Hello World'
typeof 19.5
typeof true
```

In the three examples, the first example returns 'string', the second example returns 'number', and the third example returns 'Boolean'.

Operators

Arithmetic operators

- +
- -
- *
- /
- % : modulus i.e remainder of the division
- ++ (increment)
- -- (decrement)

Assignment operators

- =
- +=
- -=
- *=
- /=
- %=

Comparison operators

- == (compare only values, allowing automatic type conversion)
- === (compare both values and type, without type conversion.)
- !=
- !==
- >
- <
- >=
- <=

Conditional operator

- ? :

e.g. ‘((a > b) ? a/b : b/a))’ i.e if ‘a>b’ then do ‘a/b’ else do ‘b/a’

Logical operators

- && (logical and)
- || (logical or)
- ! (logical not)

Bitwise operators

- & (and)
- | (or)
- ^ (xor)
- ~ (not)

Variables

Variables can be defined using keyword ‘var’ or ‘let’. Further, JavaScript is untyped language i.e. a variable can hold any type of value.

Variable names can be any word. Digits can be part of variable names—catch22 is a valid name, for example—but the name must not start with a digit. A variable name may include dollar signs ($) or underscores (_) but no other punctuation or special characters.

Words with a special meaning, such as 'let', are keywords, and they may not be used as variable names. There are also a number of words that are “reserved for use” in future versions of JavaScript, which also can’t be used as variable names. The full list of keywords and reserved words is rather long:

> *break case catch class const continue debugger default delete do else enum export extends false finally for function if implements import interface in instanceof let new package private protected public return static super switch this throw true try typeof var void while with yield*

Don’t worry about memorizing this list. When creating a variable produces an unexpected syntax error, see whether you’re trying to define as variable a reserved word.

Rules when naming JavaScript variables:

- A variable name can contain numbers, but they cannot begin with a number. Legal examples are x1, y2, gift4you. Illegal examples are 4YourEyes, 2give, 1ForAll.
- Variable names must not contain mathematical or logical operators. Illegal examples are monday-friday, boxes+bags, cost*5.
- Variable names must not contain any punctuation marks of any kind other than the underscore (_) and dollar sign ($). Legal examples are vehicle_identification, first_name, last_name, $cost, total$. Illegal examples are thisDoesn’tWork, begin;end, Many#s.
- Variable names must not contain any spaces.
- Variable names must not be JavaScript keywords, but they can contain keywords. Illegal examples are function, char, class, for, var. Legal examples are theFunction, for-Loop, myVar.

- Variable names are case-sensitive. Examples of different-case variables are MyData, myData, mydata, MYDATA.

Explicit Type Conversion

Number() – Number() is used to convert strings into numbers.
e.g. Number('3.4') is 3.4

parseInt() – parseInt() is used to convert strings or float numbers to integers
e.g. parseInt('3.4') is 3
parseInt(3.4) is 3

parseFloat() – converts strings to numbers. Similar to Number().

Automatic Type Conversion

JavaScript automatically converts any type of data to any other type. For instance:

"5" – 1 is 4 ("5" is converted to 5)

"5" + 1 is "51" (1 is converted to "1")

"5"==5 is **true**

false == 0 is **true**

false == "" is **true**

"" == 0 is **true**

If we need to compare without automatic type conversion, we must use the operator === (and its negation !==).

Special values

Note that:

"five" - 1 is NaN

When something that doesn't map to a number in an obvious way (such as "five") is converted to a number, you get the value **NaN**. Further arithmetic operations on **NaN** keep producing **NaN**, so if you find yourself getting one of those in an unexpected place, look for accidental type conversions.

Attention: the comparison NaN == NaN is false. And NaN === NaN is also false. So, the comparisons NaN != Nan and NaN !== NaN are both true.

There are two special values, written **null** and **undefined**, that are used to denote the absence of a meaningful value. They are themselves values, but they carry no information. Many operations in the language that don't produce a meaningful value yield **undefined** simply because they have to yield some value.

The difference in meaning between **undefined** and **null** is an accident of JavaScript's design, and it doesn't matter most of the time. In cases where you actually have to concern yourself with these values, we recommend treating them as mostly interchangeable.

Advanced : The difference between undefined and null is minimal, but there is a difference. A variable whose value is undefined has never been initialized. A variable whose value is null was explicitly given a value of null, which means that the variable was explicitly set to have no value. If you compare undefined and null by using the null==undefined expression, they will be equal.

Attention:

undefined == null is **true**.
undefined === null is **false**
0==null is **false**.
false == null is **false** , but inside a test (as an **if**) or inside a logical expression, null is treaded as false.
false == undefined is **false**, but inside a test (as an **if**) or inside a logical expression, undefined is treaded as false.

Remark:

We can use the (short-circuit or: ||) as a way to fall back on a default value. If you have a value that might be empty, you can put || after it with a replacement value. If the initial value can be converted to false, you'll get the replacement instead. The rules for converting strings and numbers to Boolean values state that 0, NaN, and the empty string ("") count as false, while all the other values count as true.
So 0 || -1 produces -1, and "" || "!?" yields "!?".

Basic Input / Output

JavaScript can "display" data in different ways:

Writing into an HTML element, using innerHTML.
Writing into the HTML output using document.write() and. document.writeln()
Writing into an alert box, using window.alert().
Writing into the browser console, using console.log().

Exemples:

1.

```
<p id="id1"></p>

<script>document.getElementById("id1").innerHTML = "Good Afternoon"</script>
```

2.

```
<script>
    document.writeln("Hello <br>");
    document.writeln("Hello again<br>");
</script>
```

Attention: to actually break line in the final result, we need the
. The writeln() adds a line break just to the html input to the browser.

3.

```
<script>alert("Hi");</script>
```

makes a pop-up window to show.

4.

```
<script>console.log("Point 2 reached");</script>
```

For debugging purposes, you can use the console.log() method to display data. It writes data to the console, not the page.

For basic input from the user, there is the function prompt(). For instance:

```
var x = prompt("enter a number");
document.write("2 * "+ x+ " = "+ 2*x + "<br>");
```

A pop-up window will appear due to prompt.

Control Structures

They are the standard C style control structures:

Examples:

1. **if-else**

```
age = 10;
if (age > 3 && age < 6){
        document.write("Age : " + age + "<b> go to kindergarten</b>");
}
else if ( age >=6 && age < 18){
        document.write("Age : " + age + "<b> go to school</b>");
}
else{
        document.write("Age : " + age + "<b> go to college</b>");
}
document.write("<br>");
```

2. **switch**

```
var grade = 'A';
document.write("Grade " + grade + " : ");
switch(grade){
        case 'A':
                document.write("Very good grade!");
                break;
        case 'B':
                document.write("Good grade!");
                break;
        default: // if grade is neither 'A' nor 'B'
                document.write("Consider improving in the future");
}
document.write("<br>");
```

3. **for**

```
for (i=5; i>=0; i--){
        document.write(i + " ");
}
document.write("<br>");
```

4. **while**

```
x=0;
while(x < 5){
        document.write(x + " ");
        x++;
}
document.write("<br>");
```

5. **do-while**

```
x=0;
do{
        document.write(x + " ");
        x++;
}while(x < 3);
document.write("<br>");
```

6. **for-in**

```
arr = [10, 12, 31]; // array
for (a in arr){
        document.write(arr[a] + " ");
}
document.write("<br>");
```

Remark: JavaScript also has the control statements continue and break to be used in the loops.

References:

Johnson, Glenn - *Programmingin HTML5 with JavaScript and CSS3 – Training Guide*, Microsoft Press, 2013

Haverbeke, Marijn - *Eloquent JavaScript* – 3rd Edition, No Starch Press, 2018

Patel, Meher Krishna - *HTML, CSS, Bootstrap, Javascript and jQuery* – PythonDSP, October 2018

http://eloquentjavascript.net/

https://www.w3schools.com/js