JavaScript Fundamentals

07 July 2020 15:45

Script can be written anywhere in the HTML code, but it is recommended to write a script tag(JS) at the bottom of the HTML code.
Better to write the script in new file or separate file.
https://blog.logrocket.com/how-browser-rendering-works-behind-the-scenes-6782b0e8fb10/

Variable names – camelCase – case matters

Uppercase constants - There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution.
Ex: `const COLOR_RED = "#F00";`

Let vs const vs var:
https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/#:~:text=var%20declarations%20are%20globally%20scoped%20or%20function%20scoped%20while%20let,be%20updated%20nor%20re%2Ddeclar

- Problem with var

```
var greeter = "hey hi";
    var times = 4;

    if (times > 3) {
        var greeter = "say Hello instead";
    }

    console.log(greeter) // "say Hello instead"
```

- const cannot be updated or re-declared. But objects can be updated (`greeting.message = "say Hello instead";`)
- const variables must be initialized at the time of declaration

	Can be redeclared?	updated	hoisting	scope
var	yes	yes	yes and initialized as undefined	global
let	no	yes	yes and but not initialized	block {}
const	no	no	yes and but not initialized	Block {}

```
// Use Object.freeze() to make object immutable
When an object freezed, we cannot add a property or edit any existing property.(refer const obj vs Object.freeze())
```

Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}.

We don’t recommend assigning a variable with undefined. Normally, one uses null to assign an “empty” or “unknown” value to a variable, while undefined is reserved as a default initial value for unassigned things.

The result of typeof null is "object". That’s an officially recognized error in typeof behavior, coming from the early days of JavaScript and kept for compatibility
Definitely, null is not an object. It is a special value with a separate type of its own.

For Binary operator (+), if any of the operands is a string, then the other one is converted to a string too. Other arithmetic operators work only with numbers a
always convert their operands to numbers.
The plus operator + applied to a single value, doesn’t do anything to numbers. But if the operand is not a number, the unary plus converts it into a number. It
actually does the same thing as `Number(...)`, but is shorter.

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

Assignment = returns a value

```
alert( true == 1 ); // true
alert( false == 0 ); // true
alert( '' == false ); // true
Operands of different types are converted to numbers by the equality operator ==. An empty string, just like false,
becomes a zero.

A strict equality operator === checks the equality without type conversion.
alert( null === undefined ); // false
alert( null == undefined ); // true

For math and other comparisons < > <= >=, null/undefined are converted to numbers: null becomes 0 while undefined bec
NaN.
On the other hand, the equality check == for undefined and null is defined such that, without any conversions, they e
each other and don’t equal anything else.
alert( null > 0 );  // (1) false
alert( null == 0 ); // (2) false
alert( null >= 0 ); // (3) true
```

It's not recommended to use the question mark operator for blocks of code. The notation is shorter than the equivaler statement, which appeals to some programmers. But it is less readable.

A chain of OR "||" returns the first truthy value or the last one if no truthy value is found.
A chain of AND "&&" returns the first falsy value or the last value if none were found.

Don't replace if with || or &&

```
(x > 0) && alert( 'Greater than zero!' ); //Not Recommended
if (x > 0) alert( 'Greater than zero!' ); //Recommended
```

A double NOT !! is sometimes used for converting a value to Boolean type

A **polyfill** is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it

important difference is that:

|| returns the first *truthy* value.
?? returns the first *defined* value.
Ex:

```
let a = 0
let b = 1
let c = a || b
console.log(c)
let d = a ?? b
Console.log(d)
```

Due to safety reasons, it's forbidden to use ?? together with && and || operators.

No break/continue to the right side of '?'. Instead of '?' use 'if-else' block.

```
(i > 5) ? alert(i) : continue; // continue isn't allowed here
```

Labels for break/continue

```
labelName: for (...) {
  ...

}
```

The break <labelName> statement in the loop breaks out to the label.
The continue directive can also be used with a label. In this case, code execution jumps to the next iteration of the labeled loop.

Switch:
The value is checked for a strict equality.
If there is no break then the execution continues with the next case without any checks.

TypeConversion:
Numeric conversion happens in mathematical functions and expressions automatically.

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

let age = Number("an arbitrary string instead of a number");
alert(age); // NaN, conversion failed
```

Numeric conversion rules:

Value	Becomes...
undefined	NaN
null	0
True & False	1 & 0
string	Whitespaces from the start and end are removed. If the remaining string is empty, the result is 0. Otherwise, the number is "read" from the string. An error gives NaN.

Number() method calls two different methods, parseInt() and parseFloat() implicitly depending on the value.

```
Number("5.6.7") //NaN
parseInt("5.5.5") //5
ParseFloat("5.5.5") //5.5
```

Boolean Conversion (Boolean(value)):

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

Note: **the string with zero "0" is true.** Some languages (namely PHP) treat "0" as false. But in JavaScript, a non-empty string always true.

The type casting , String() method actually calls the toString() method. The only difference between type casting as string and using toString() is that the type cast can produce a string for a null or undefined value without error as shown above.