

<WA1/>
<AW1/>
2021



JavaScript (basics)

"The" language of the Web

Fulvio Corrao
Luigi De Russis
Enrico Masala

A screenshot of the second page of the JavaScript cheat sheet. It continues the coverage of the language, including more advanced topics like Error Handling, Regular Expressions, and the Fetch API. The layout remains consistent with the first page, featuring a grid of code snippets and explanatory text.

Goal

- Learn **JavaScript as a language**
 - Understand the specific semantics and programming patterns
 - We assume a programming knowledge in other languages
 - Updated to ES6 (2015) language features
 - Supported by server-side (`Node.js`) and client-side (browsers) run-time environments
- Also some ES7 constructs but we don't push too forward to avoid compatibility problems

Weird language and constructs (some very old, some modern)

Application Web I – Web Applications I – 2020/2021

Outline

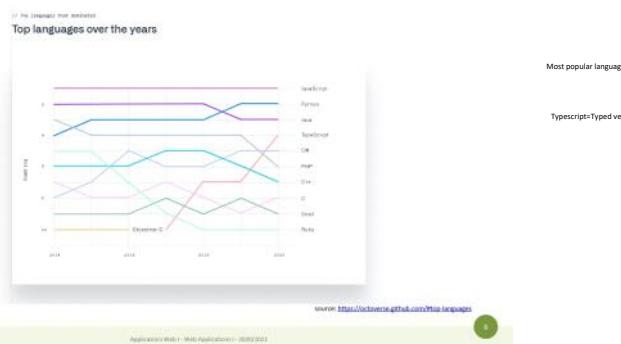
- What is JavaScript?
- History and versions
- Language structure
- Types, variables
- Expressions
- Control structures
- Arrays
- Strings

Application Web I – Web Applications I – 2020/2021

JavaScript – The language of the Web

WHAT IS JAVASCRIPT?

Application Web I – Web Applications I – 2020/2021



JavaScript

- JavaScript (JS) is a programming language
- It is currently the only programming language that a browser can execute natively... server/vm
- ... and it also runs on a computer, like other programming languages (thanks to Node.js) Node interpreter ported javascript engine from chrome to an independent executable stripping browser parts, keeping the interpreter and adding API to the operating system!
- It has **nothing** to do with Java
 - named that way for *marketing reasons*, only
- The first version was written in 10 days (!) Brandon Eich Netscape navigator v2 to add animation!
- several fundamental language decisions were made because of company politics and not technical reasons!

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

JavaScript – The language of the Web

HISTORY AND VERSIONS

JAVASCRIPT VERSIONS

European Computer Manufacturers Association

Standardization committee: ECMA International

Official name: ECMAScript
But everybody calls it JS

10 yrs

Main target

ES9, ES10, ...

Also: ES2015

Also: ES2016

Also: ES2017

Revolution of the language! Necessary for big applications! (compiler's problem)
Start of Modern JavaScript

New standard version of the library -> not immediately implemented in node/browsers (race)

<https://www.ecma-international.org/publications/standards/ECMA-262/>

JavaScript versions

- ECMAScript (also called ES) is the official name of JavaScript (JS) standard
- ES6, ES2015, ES2016 etc. are implementations of the standard
- All browsers used to run ECMAScript 3
- ES5, and ES2015 (=ES6) were huge versions of JavaScript
- Then, yearly release cycles started
 - By the committee behind JS: TC39, backed by Mozilla, Google, Facebook, Apple, Microsoft, Intel, PayPal, SalesForce, etc.
- **ES2015 (=ES6) is covered in the following**

<https://tc39.es/ecma262/>

For compilers developers, not for js programmers!



Official ECMA standard (formal and unreadable)

For compilers developers, not for js programmers!



<https://www.ecma-international.org/ecma-262/>

11

Aplicaciones Web I - Web Applications I - 2020/2021

JavaScript Engines

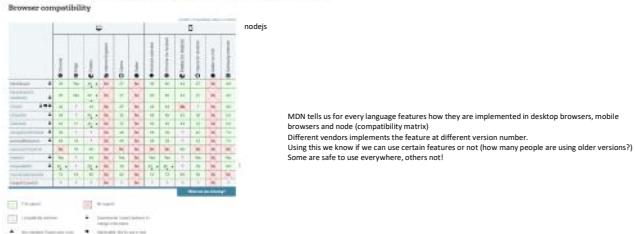
Implementations of the language

- V8 (Chrome V8) by Google
 - used in Chrome/Chromium, Node.js and Microsoft Edge
- SpiderMonkey by Mozilla Foundation
 - Used in Firefox/Gecko
- ChakraCore by Microsoft dismissed
- – it was used in Edge
- JavaScriptCore by Apple
 - used in Safari

12

Aplicaciones Web I - Web Applications I - 2020/2021

Standard vs. Implementation (in browsers)



13

Aplicaciones Web I - Web Applications I - 2020/2021

JS Compatibility

- JS is **backwards-compatible**
 - once something is accepted as valid JS, there will not be a future change to the language that causes that code to become invalid JS
 - TC39 members: "we don't break the web!"
- JS is **not forwards-compatible**
 - new additions to the language will not run in an older JS engine and may crash the program
- **strict mode** was introduced to disable very old (and dangerous) semantics
- Supporting multiple versions is achieved by:
 - **Transpiling** – Babel (<https://babeljs.io>) converts from newer JS syntax to an equivalent older syntax
 - **Polyfilling** – user- or library-defined functions and methods that "fill" the lack of a feature by implementing the newest available one

Old sites are visible by newer js engines!

New html engines can see old sites

Old js engine doesn't see newer js: we want you to update your browser if you want to see newer pages

Old html engines can see newer html pages ignoring newer elements (we don't want to force you to change your browser ->forward compatible)

Older? 3 days? 1 year? Depends on features!
How to deal with compatibility problems?
Mechanisms inside language: keyword strict mode (at least ES6)
2 ways:
transpiling: (converts code from one version of JS to another, but of an older version when you wrote it using latest features, before shipping the code, it gets translated into a simpler version (translating code so language constructs compatible with various, older versions of the browser))
polyfilling: new library function/old function with new parameters > fill the holes in the old engine
(Not easy to do but possible)

Transparent parts of the process of building an application (call transpiler, include polyfill library)

After these we can be sure to deal flexibly with many different users engine

14

Aplicaciones Web I - Web Applications I - 2020/2021

JS Execution Environments

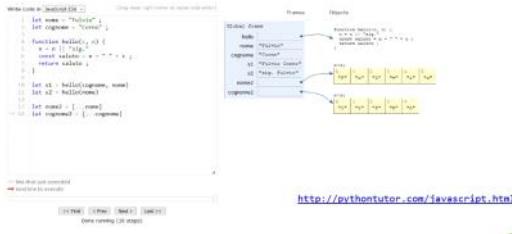


15

Aplicaciones Web I - Web Applications I - 2020/2021

JavaScriptTutor

Graphical representation of the variables!!! (variable representation similar to Python NOT TYPED)



Browser and JS console



JavaScript – The language of the Web

LANGUAGE STRUCTURE

Lexical structure

- One File = One JS program
 - Each file is loaded independently and
 - Different files/programs may communicate through *global state*
 - The “module” mechanism extends that (provides state sharing in a clean way)
 - The file is entirely *parsed*, and then *executed* from top to bottom
 - Relies on a *standard library*
 - and many additional *APIs* provided by the execution environment

Aplicaciones Web I - Web applications I - 2020/2021

Lexical structure

```
> let ooo = 'appalled'  
> ooo
```

- JavaScript is written in Unicode (do not abuse), so it also supports non-latin characters for names and strings
 - even emoji
 - Semicolons (;) are not mandatory (automatically inserted)
 - Case sensitive
 - Comments as in C /*...*/ and //
 - Literals and identifiers (start with letter, \$, _)
 - Some reserved words
 - C-like syntax

Semicolon (;)

- Argument of debate in the JS community
- JS inserts them as needed
 - When next line starts with code that breaks the current one
 - When the next line starts with }
 - When there is return, break, throw, continue on its own line
- Be careful that forgetting semicolon can lead to unexpected behavior
 - A newline does not automatically insert a semicolon: if the next line starts with (or [, it is interpreted as function call or array access)
- We will **loosely** follow the Google style guide, so we will always insert semicolons after each statement
 - <https://google.github.io/styleguide/jsguide.html>

Aplicaciones Web I - Web Applications I - 08/02/2023

// first line of file
"use strict";
if (true) {
 console.log("Hello World");
}

31

Strict Mode

- Directive introduced in ESS: "use strict" ;
 - Compatible with older version (it is just a string)
- **Code is executed in strict mode**
 - This fixes some important language deficiencies and provides stronger error checking and security
 - Examples:
 - fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode
 - eliminates some JavaScript silent errors by changing them to throw errors
 - functions invoked as functions and not as methods of an object have this undefined
 - cannot define 2 or more properties or function parameters with the same name
 - no octal literals (base 8, starting with 0)
 - ...

Aplicaciones Web I - Web Applications I - 08/02/2023

JavaScript: The Definitive Guide, 7th Edition
Chapter 2. Types, Values, and Variables

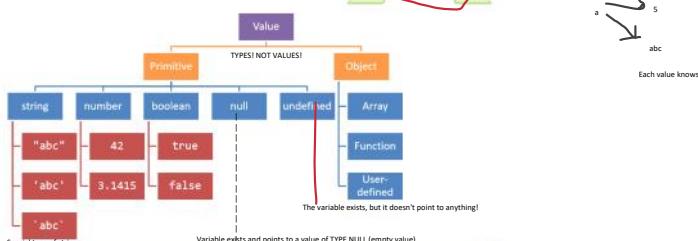
32

JavaScript – The language of the Web

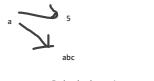
TYPES AND VARIABLES

Aplicaciones Web I - Web Applications I - 08/02/2023

Values and Types



a=5 -> what is the type of THE VALUE represented by this variable?
I can't ask "what is the type of this variable?" Because a variable is only a reference to a value stored somewhere (THE VALUE IS STRONGLY TYPED!)



33

Boolean, true-truthy, false-falsy, comparisons

ONLY 2 REAL BOOLEAN

BUT... ANYTHING (ANY VALUE) CAN BE INTERPRETED AS A BOOLEAN WITH STRANGE RULES!!!

- 'boolean' type with literal values: true, false
- When converting to boolean
 - The following values are 'falsy'
 - 0, -0, NaN, undefined, null, '' (empty string)
 - Every other value is 'truthy'
 - 3, !false, {}, [] (empty array), {} (empty object)
- Booleans and Comparisons
 - a == b // convert types and compare results
 - a === b // inhibit automatic type conversion and compare results

!0 !{} ![] !NaN !undefined !null !''

> Boolean(3)
true

> Boolean('')
false

> Boolean(' ')
true

> Boolean({})
true

> Boolean([])
true

> Boolean({})
true

> Boolean([])
true

> Boolean(NaN)
false

> Boolean(null)
false

> Boolean(undefined)
false

> Boolean(-0)
true

> Boolean(0)
false

> Boolean("")
false

> Boolean(``)
false

Number

- No distinction between integers and reals
- Automatic conversions according to the operation
- There is also a distinct type "BigInt" (*ES11, July 2020*)
 - an arbitrary-precision integer, can represent 2^{53} numbers
 - 123456789n
 - With suffix 'n'

Aplicaciones Web I - Web Applications I - 2020/2021

21

Special values

- **undefined**: variable declared but not initialized
 - Detect with: `typeof variable === 'undefined'`
 - `void x` always returns `undefined`
- **null**: an empty value
- Null and Undefined are called *nullish values*
- **NaN (Not a Number)**
 - It is actually a number
 - Invalid output from arithmetic operation or parse operation

Aplicaciones Web I - Web Applications I - 2020/2021

22

Variables

- Variables are **pure references**: they refer to a *value*
- The same variable may refer to different values (even of different types) at different times
- Declaring a variable:
 - `let`
 - **X** `const`
 - `var`

```
> v = 7 ;  
?  
> v = 'hi' ;  
'hi'
```

```
> let a = 5  
> const b = 6  
> var c = 7  
> a = 8  
8  
> b = 9  
Thrown:  
TypeError: Assignment to  
constant variable.  
> c = 10  
10
```

You must define a variable before assigning it!
You can only define a variable once!
Let a=5;
as="abc";
Let b=4;



Aplicaciones Web I - Web Applications I - 2020/2021

23

Variable declarations

Declarator	Can reassign?	Can re-declare?	Scope	Hoisting *	Note
<code>let</code>	Yes	No	Enclosing block {...}	No	Preferred
<code>const</code>	No §	No	Enclosing block {...}	No	Preferred
<code>var</code>	Yes	Yes a... Var a...	Enclosing function, or global {...}	Yes, to beginning of function or file Escapes function Also BEFORE DECLARATION	Legacy, beware its quirks, try not to use
None (implicit)	Yes	N/A	Global	Yes	Forbidden in strict mode

§ Prevents reassignment (`a=2`), does not prevent changing the value of the referred object (`a,b=2`)

* Hoisting = "lifting up" the definition of a variable (not the initialization!) to the top of the current scope (e.g., the file or the function)

Aplicaciones Web I - Web Applications I - 2020/2021

24

Scope

```
"use strict" ;  
  
let a = 1 ;  
const b = 2 ;  
let c = true ;  
  
let a = 5 ; // SyntaxError: Identifier 'a' has already been declared
```

Aplicaciones Web I - Web Applications I - 2020/2021

25

Scope

Typically, you don't create a new scope in this way!

```
"use strict";
let a = 1;
const b = 2;
let c = true;

{ // creating a new scope...
  let a = 5;
  console.log(a);
}

console.log(a);
```

Each {} is called a **block**. 'let' and 'const' variables are **block-scoped**.

They exist only in their defined and inner scopes.

Aplicaciones Web I - Web Applications I - 2020/2021

11

Scope and Hoisting

```
"use strict";

function example(x) {
  let a = 1;
  var c; // hoisted

  console.log(a); // 1
  console.log(b); // ReferenceError: b is not defined
  console.log(c); // undefined

  if (x1) {
    var a = a+1;
    var c = a*2;
  }

  console.log(a); // 1
  console.log(b); // ReferenceError: b is not defined
  console.log(c); // 2
}

example(2);
```

Aplicaciones Web I - Web Applications I - 2020/2021

12

JavaScript: The Definitive Guide, 7th Edition
Chapter 2. Types, Values, and Variables
Chapter 3. Expressions and Operators

Mozilla Developer Network
JavaScript Guide » Expressions and operators

JavaScript – The language of the Web

EXPRESSIONS

Aplicaciones Web I - Web Applications I - 2020/2021

13

Operators

- Assignment operators
- Comparison operators
- Arithmetic operators
- Bitwise operators
- Logical operators
- String operators
- Conditional (ternary) operator
- Comma operator
- Unary operators
- Relational operators



Full reference and operator precedence:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precendence#Table

Aplicaciones Web I - Web Applications I - 2020/2021

14

Assignment

- let variable = expression ; // declaration with initialization
- variable = expression ; // reassignment

Name	Shorthand operator	Meaning
Assignment	=	a = b
Indirect assignment	!=	a != b
Subtraction assignment	-=	a -= b
Multiplication assignment	*=	a *= b
Division assignment	/=	a /= b
Modulus assignment	%=	a %= b
Exponentiation assignment	**=	a **= b
Left shift assignment	<=	a <= b
Right shift assignment	>=	a >= b
Two-bit right assignment	>>=	a >>= b
Bitwise AND assignment	&=	a &= b
Bitwise OR assignment	=	a = b

Aplicaciones Web I - Web Applications I - 2020/2021

15

Comparison operators

Operator	Description	Example returning true	Same value?
<code>==</code> (<code>==</code>)	Returns <code>true</code> if the operands are equal.	<code>9 == 9</code> <code>9 == '9'</code>	Same problems as PHP! "9" > 2 > 32
<code>!=</code> (<code>!=</code>)	Returns <code>false</code> if the operands are not equal.	<code>9 != 4</code> <code>9 != '9'</code>	
<code>strictEqual (==)</code>	Returns <code>true</code> if the operands are equal and of the same type. See also <code>Object.is</code> and <code>SameValue</code> .	<code>9 === 9</code>	Same type and same value
<code>strictNotEqual (!==)</code>	Returns <code>true</code> if the operands are of the same type but not equal, or of different types.	<code>9 === '9'</code> <code>9 !== 9</code>	
<code>></code> (<code>></code>)	Returns <code>true</code> if the left operand is greater than the right operand.	<code>9 > 4</code> <code>'9' > '4'</code>	
<code><</code> (<code><</code>)	Returns <code>true</code> if the left operand is greater than or equal to the right operand.	<code>9 >= 4</code> <code>'9' >= '4'</code>	
<code>>=</code> (<code>>=</code>)	Returns <code>true</code> if the left operand is less than the right operand.	<code>9 < 4</code> <code>'9' < '4'</code>	
<code><=</code> (<code><=</code>)	Returns <code>true</code> if the left operand is less than or equal to the right operand.	<code>9 <= 4</code> <code>'9' <= '4'</code>	

Application Web 1 - Web Application (v. 2020/2021)

31

Comparing Objects

In order to compare the general structure of objects, it compares the reference! We need an "equal" method!

- Comparison between objects with `==` or `===`
compares the *references* to objects
 - True only if they are *the same object*
 - False if they are *identical objects*
- Comparison with `< > <= >=` first converts the object (into a Number, or more likely a String), and then compares the values
 - It works, but may be unpredictable, depending on the string format

```
> a={x:1}
{ x: 1 }

> b={x:1}
{ x: 1 }

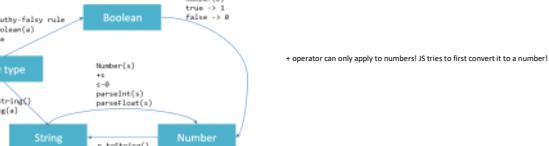
> a==b
false
> a=b
false
```

Application Web 1 - Web Application (v. 2020/2021)

32

Automatic Type Conversions

- JS tries to apply type conversions between primitive types, before applying operators
- Some language constructs may be used to "force" the desired conversions
- Using `==` applies conversions
- Using `===` prevents conversions



Application Web 1 - Web Application (v. 2020/2021)

33

Logical operators

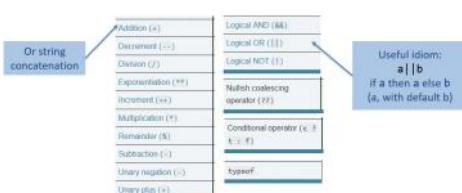
In C && RETURNS the FIRST EXPRESSION if the first is false without computing the second. It then returns the second expression value (TRUE or FALSE) which will be the result of the && operator.
In JS && it is the same if both operands are Booleans.
If both operands are the return value of a function, we can use && to execute the second function (and evaluate its result) only if the first one returned TRUE.
If both operands are the return value of a function, we can use || to execute the second function (and evaluate its result) only if the first one returned FALSE.

Operator	Usage	Description
Logical AND (<code>&&</code>)	<code>expr1 && expr2</code>	Returns <code>expr1</code> if it can be converted to <code>false</code> ; otherwise, returns <code>expr2</code> . Thus, when used with Boolean values, && returns <code>true</code> if both operands are <code>true</code> , otherwise, returns <code>false</code> .
Logical OR (<code> </code>)	<code>expr1 expr2</code>	Returns <code>expr1</code> if it can be converted to <code>true</code> ; otherwise, returns <code>expr2</code> . Thus, when used with Boolean values, returns <code>true</code> if either operand is <code>true</code> ; if both are <code>false</code> , returns <code>false</code> .
Logical NOT (<code>!</code>)	<code>!expr</code>	Returns <code>false</code> if its single operand can be converted to <code>true</code> ; otherwise, returns <code>true</code> .

Application Web 1 - Web Application (v. 2020/2021)

34

Common operators



Application Web 1 - Web Application (v. 2020/2021)

35

Mathematical functions (Math building object)

- Constants: Math.E, Math.LN10, Math.LN2, Math.LOG10E, Math.LOG2E, Math.PI, Math.SQRT1_2, Math.SQRT2
- Functions: Math.abs(), Math.acos(), Math.acosh(), Math.asin(), Math.asinh(), Math.atan(), Math.atan2(), Math.atanh(), Math.cbrt(), Math.ceil(), Math.clz32(), Math.cos(), Math.cosh(), Math.exp(), Math.expm1(), Math.floor(), Math.fround(), Math.hypot(), Math.imul(), Math.log(), Math.log10(), Math.log1p(), Math.log2(), Math.max(), Math.min(), Math.pow(), Math.random(), Math.round(), Math.sign(), Math.sin(), Math.sinh(), Math.sqrt(), Math.tan(), Math.tanh(), Math.trunc()

Application Web I - Web Applications / 2022/2023

11

JavaScript: The Definitive Guide, 7th Edition
Chapter 4. Statements

Mozilla Developer Network
JavaScript Guide » Control Flow and Error Handling
JavaScript Guide » Loops and Iteration

JavaScript – The language of the Web

CONTROL STRUCTURES

Conditional statements

```
if (condition_1) {
    statement_1;
} else {
    statement_2;
}

if (condition_1) {
    statement_1;
} else if (condition_2) {
    statement_2;
} else if (condition_n) {
    statement_n;
} else {
    statement_last;
}
```

```
switch (expression) {
    case label_1:
        statements_1
        [break];
    case label_2:
        statements_2
        [break];
    default:
        statements_def
        [break];
}
```

May also be a string

Loop statements

```
for ([[initialExpression]]; [condition]; [[incrementExpression]]) {
    statement ;
}
```

Usually declares loop variable

```
do {
    statement ;
} while (condition);
```

```
while (condition) {
    statement ;
}
```

May use break; or continue;

Special 'for' statements

```
for (variable in object) {
    statement ;
}
```

- Iterates the variable over all the enumerable properties of an object
- Do not use to traverse an array (use numerical indexes, or for-of)

```
for( let a in {x: 0, y:3}) {
    console.log(a) ;
}
x
y
```

Composed object (array, string)

```
for (variable of iterable) {
    statement ;
}
```

- Iterates the variable over all values of an iterable object (including Array, Map, Set, String, arguments ...)
- Returns the values, not the keys

```
for( let a of [4,7]) {
    console.log(a) ;
}
4
7
Array>element by element
```

```
for( let a of "hi" ) {
    console.log(a) ;
}
h
String>Letter by letter
```

Python in Java : js of

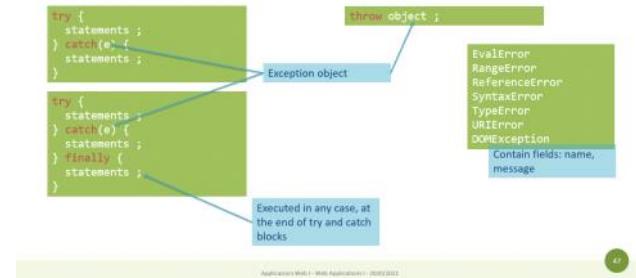
ferred

Other iteration methods

- Functional programming (strongly supported by JS) allows other methods to iterate over a collection (or any iterable object)
Do an operation on every element of an array
 - a.forEach()
 - a.map()
- They will be analyzed later



Exception handling



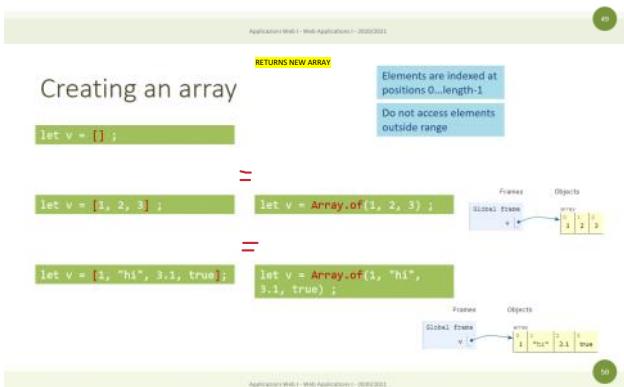
JavaScript – The language of the Web

ARRAYS

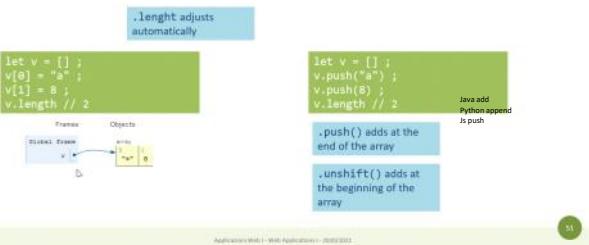


Arrays

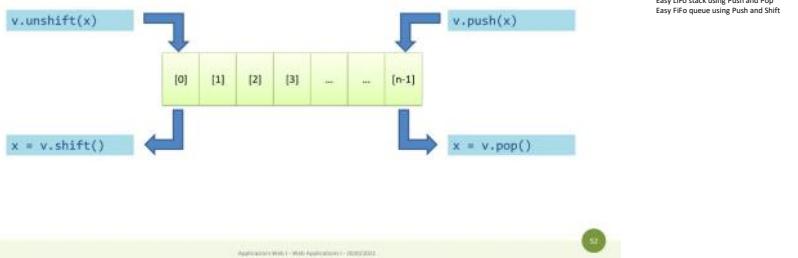
- Rich of functionalities
- Elements do not need to be of the same type
- Simplest syntax: []
- Property .length
- Distinguish between methods that:
 - Modify the array (in-place)
 - Return a new array



Adding elements



Adding and Removing from arrays (in-place)

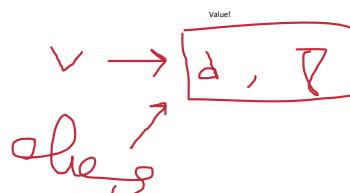


Copying arrays

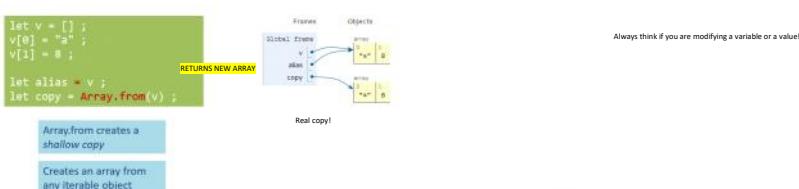
```
let v = [] ;
v[0] = "a" ;
v[1] = 8 ;

let alias = v ;
alias[1] = 5 ;
```

Multiple variables reference to the same value (in this case an array but also an object) (COPIES POINTER, not real copy)



Copying arrays



Iterating over Arrays

- Preferred**
- Iterators: `for ... of`, `for (...; ...; ...)`
 - Iterators: `forEach(f)`
 - f is a function that processes the element
 - Iterators: `every(f)`, `some(f)`
 - f is a function that returns true or false
 - Iterators that return a new array: `map(f)`, `filter(f)`
 - f works on the element of the array passed as parameter
 - Reduce: exec a callback function on all items to progressively compute a result

Main array methods

RETURNS NEW ARRAY	MODIFY EXISTING ARRAY
<code>.concat()</code>	<code>.reverse()</code>
- joins two or more arrays and returns a new array.	- transposes the elements of an array, in place
<code>.join(delimiter = ',')</code>	<code>.sort()</code>
- joins all elements of an array into a (new) string.	- sorts the elements of an array in place
<code>.slice(start_index, upto_index)</code>	<code>.indexOf(searchElement[, fromIndex])</code>
- extracts a section of an array and returns a new array.	- searches the array for searchElement and returns the index of the first match
<code>.splice(index, count_to_remove, addElement1, addElement2, ...)</code>	<code>.lastIndexOf(searchElement[, fromIndex])</code>
- (optionally) replaces them, in place.	- like indexOf, but starts at the end
N.B. You can insert a number of elements different from the number of elements that was removed	<code>.includes(valueToFind[, fromIndex])</code>
	- search for a certain value among its entries.

- example of removing an array and returning a new array.

- `.splice(index, count_to_remove, addElement1, addElement2, ...)`
MODIFIES EXISTING ARRAY AND RETURNS THE REMOVED ELEMENTS
(optionally) replaces them, in place

N.B. You can insert a number of elements different from the number of elements that was removed

Applications Web I - Web Applications I - 2020/2021

57

Destructuring assignment

- Value of the right-hand side of equal sign are extracted and stored in variables on the left

```
let [x,y] = [1,2];
[x,y] = [y,x]; // swap
```
- var foo = ['one', 'two', 'three'];
var [one, two, three] = foo;
 - Useful especially with passing and returning values from functions

```
let [x,y] = toCartesian(r,theta);
```

Applications Web I - Web Applications I - 2020/2021

58

Spread operator (3 dots: ...)

- Expands an iterable object in its parts, when the syntax requires a comma-separated list of elements

```
let [x, ...y] = [1,2,3,4]; // we obtain [1,2,3,4]
```
- const parts = ['shoulders', 'knees'];
const lyrics = ['head', ...parts, 'and', 'toes']; // ["head", "shoulders", "knees", "and", "toes"]
- Works on the left- and right-hand side of the assignment

Applications Web I - Web Applications I - 2020/2021

59

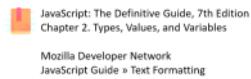
Curiosity

C_a[...a...b] → concat 2 arrays
Copy(...a)=Array.from(a)=Array.of(...a) → shallow copy (REAL COPY!)

- Copy by value:
- const b = Array.from(a)
- Can be emulated by
- const b = Array.of(...a)
- const b = [...a]

Applications Web I - Web Applications I - 2020/2021

60



JavaScript – The language of the Web

STRINGS

Applications Web I - Web Applications I - 2020/2021

61

Strings in JS

- A string is an **immutable** ordered sequence of Unicode characters
- The **length** of a string is the number of characters it contains (not bytes)
- JavaScript's strings use zero-based indexing
 - The empty string is the string of length 0
- JavaScript does not have a special type that represents a single character (use length-1 strings).
- String literals may be defined with 'abc' or "abc"
 - Note: when dealing with JSON parsing, only " " can be correctly parsed

Applications Web I - Web Applications I - 2020/2021

62

String operations

- All operations always return **new** strings
 - Consequence of immutability
- **s[3]**: indexing
- **s1 + s2**: concatenation
- **s.length**: number of characters
 - Note: `.length`, not `.length()`

Application Web I - Web Applications / 2020/2021

String methods

Method	Description
charAt, charCodeAt, codePointAt	Return the character or character code at the specified position in string.
lastIndexOf, indexOf	Return the position of specified substring in the string or last position of specified substring, respectively.
startsWith, endsWith, includes	Returns whether or not the string starts, ends or contains a specified string.
concat	Concatenates the text of two strings and returns a new string.
fromCharCode, fromCodePoint	Constructs a string from the specified sequence of Unicode values. This is a method of the String class, not a String instance.
split	Splits a String object into an array of strings by separating the string into substrings.
slice	Extracts a section of a string and returns a new string.
substring, subString	Return the specified subset of the string, either by specifying the start and end indexes or the start index and a length.
match, search, replace, search	Work with regular expressions.
toLowerCase, toUpperCase	Return the string in all lowercase or all uppercase, respectively.
normalize	Returns the Unicode Normalization Form of the calling string value.
repeat	Returns a string consisting of the elements of the object repeated the given times.
trim	Trim whitespace from the beginning and end of the string.

Application Web I - Web Applications / 2020/2021

Template literals

Can span multiple lines
Interpolate expressions

- Strings included in ``backticks`` can embed expressions delimited by `${}`
- The **value** of the expression is *Interpolated* into the string
 - let name = "Bill";
let greeting = `Hello \${ name }.`;
// greeting == "Hello Bill."
- Very useful and quick for string formatting
- Template literals may also span multiple lines

Application Web I - Web Applications / 2020/2021

License



- These slides are distributed under a Creative Commons license "Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)"
- You are free to:
 - Share — copy and redistribute the material in any medium or format
 - Adapt — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- Under the following terms:
 - Attribution — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#) if you do so in any reasonable manner, but not in any way that suggests the licensor endorses your use.
 - NonCommercial — You may not use the material for [commercial purposes](#).
 - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - No additional restrictions — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Application Web I - Web Applications / 2020/2021