

<WA1/>
<AW1/>
2021

JavaScript (basics)

"The" language of the Web

Fulvio Corno
Luigi De Russis
Enrico Masala



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

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

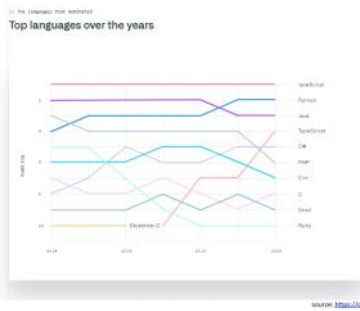
Also some ES7 constructs but we don't push too forward to avoid compatibility problems

Outline

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

JavaScript – The language of the Web

WHAT IS JAVASCRIPT?



Most popular language in Github world

Typescript=Typed version of js

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!

Deno (inverse of node) -> lighter version with less functionality

JavaScript – The language of the Web

HISTORY AND VERSIONS

JAVASCRIPT VERSIONS

European Computer Manufacturers Association

Standardization committee
Official name=ECMAScript
(but everybody calls it JS)

JAVASCRIPT (December 4th 1995) Netscape and Sun press release

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

ES1 (June 1997) Object-based, Scripting, Relaxed syntax, Prototypes

ES2 (June 1998) Editorial changes for ISO 16262

ES3 (December 1999) Regexp, Try/Catch, Do-While, String methods

ES5 (December 2009) Strict mode, JSON, .bind, Object mts, Array mts

ES5.1 (June 2011) Editorial changes for ISO 16262:2011

ES6 (June 2015) Classes, Modules, Arrow Fs, Generators, Const/Let, Destructuring, Template Literals, Promise, Proxy, Symbol, Reflect

ES7 (June 2016) Exponentiation operator (**) and Array Includes

ES8 (June 2017) Async Fs, Shared Memory & Atomics

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/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**

Top level companies contributed development of js (they add the functions they need to the language)

Official ECMA standard (formal and unreadable)

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/>

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 discontinued
 - it was used in Edge
- JavaScriptCore by Apple
 - used in Safari

Standard vs. Implementation (in browsers)

MDN tells us for every language features how they are implemented in desktop browsers, mobile browsers and node (compatibility matrix)
Different vendors implements the feature at different version number.
Using this we know if we can use certain features or not (how many people are using older versions?)
Some are safe to use everywhere, others not!

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?
Mechanism inside language: keyword strict mode (at least ES5)
Mechanism inside browser: allow browser to interpret versions of the language even newer than itself
2 ways:
Transpiling: (compiles from a language to the same language but of an older version) when you wrote js using latest features, before shipping the code, js gets translated into a simpler version (retranslate code using language constructs compatible with various, older versions of the browser) using a transpiler like Babel, for each minimum version that we want to support (avoid syntax errors "I don't know that keyword")
Polyfilling: new library function/old function with new parameters -> fill the holes in the old engine (Not easy to do but possible)

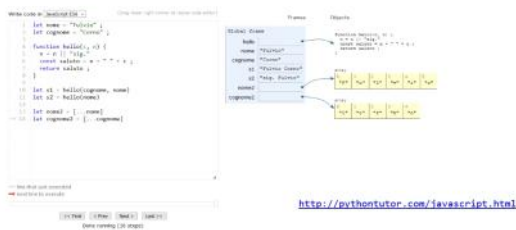
Transparent parts of the process of building an application (call transpiler, include polyfill library)

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

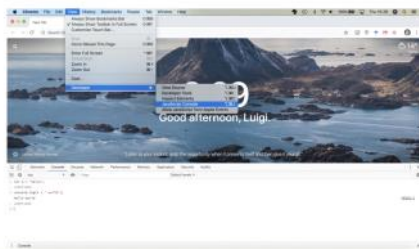
JS Execution Environments

Where does it run?





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

External environment (not from inside the programs!)

FIRST PARSED ENTIRELY, THEN EXECUTED ENTIRELY

N.B. Since we have 2 (3) different execution environments, we have 3 different standard libraries, both large, with parts in common (js standard library) but 1 has Browser library, the other has Node library

Lexical structure

```
> let bob = 'appalled'
> bob
'appalled'
```

- 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

```
> let x = '👉';
// undefined
> console.log(x);
👉
If parser doesn't understand instruction, it tries to insert a ; and see if it works!
It could make errors! Better to avoid all possible errors! Insert them!
```

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>

ApplicationsWeb1 - Web Applications I - 2020/2021

21

Strict Mode

```
// first line of file
"use strict";
// ...
```

First line of the file (program) -> Interpreter switches to strict mode

When importing module it is enabled by default

- Directive introduced in ES5: "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)
 - ...

ApplicationsWeb1 - Web Applications I - 2020/2021

22

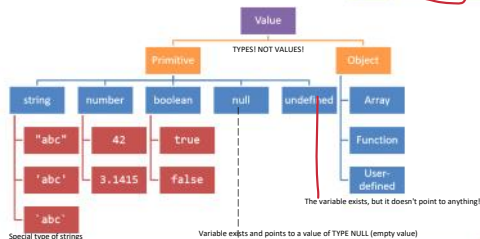


JavaScript: The Definitive Guide, 7th Edition
Chapter 2. Types, Values, and Variables

JavaScript – The language of the Web

TYPES AND VARIABLES

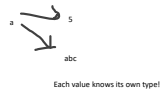
Values and Types



ApplicationsWeb1 - Web Applications I - 2020/2021

23

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!)



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'
 - 5, false, {}, {} (empty array), {} (empty object)

```
> Boolean(3)
true
> Boolean('')
false
> Boolean(' ')
true
```

NaN=special number (es. Result of division by zero)

This is why people hate js! Strange behaviour for historical reasons, without real motivations!

- Booleans and Comparisons
 - a == b // convert types and compare results
 - a === b // inhibit automatic type conversion and compare results

ApplicationsWeb1 - Web Applications I - 2020/2021

24

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'

Applications/Week 1 - Web Applications I - 2020/2021

26

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

Applications/Week 1 - Web Applications I - 2020/2021

27

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

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

- Declaring a variable:

– `let`
 X – `const`
 – `var`

```
> 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;`
`a="abc";`
`let b=a;`



Applications/Week 1 - Web Applications I - 2020/2021

28

Variable declarations

Declarator	Can reassign?	Can re-declare?	Scope	Hoisting *	Note
let	Yes	No	Enclosing block {...}	No	Preferred
const	No ¹	No	Enclosing block {...}	No	Preferred
var	Yes	Yes a... Var a...	Enclosing function, or global <small>Escapes function! Also BEFORE DECLARATION!</small>	Yes, to beginning of function or file <small>Legacy, beware its quirks, try not to use</small>	
None (implicit)	Yes	N/A	Global	Yes	Forbidden in strict mode

Assignment to constant generates error!
 The reference to the object is bound to the assigned value (address), while the object itself can be modified freely!
 Some says to always use const to avoid possible errors, except when let is really needed!

Strange behaviour!

¹ 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)

Applications/Week 1 - Web Applications I - 2020/2021

29

Scope

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

Applications/Week 1 - Web Applications I - 2020/2021

30

Scope

≠

```
"use strict";
let a = 1;
const b = 2;
let c = true;
{ // creating a new scope...
  let a = 5;
  console.log(a);
}
console.log(a);
```

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

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

They exist only in their defined and inner scopes.

ApplicationsWeb1 - Web Applications I - 2022/2023

34

Scope and Hoisting

```
"use strict";
function example(x) {
  let a = 1;
  console.log(a); // 1
  console.log(b); // ReferenceError: b is not defined
  console.log(c); // undefined
  if (x > 1) {
    let 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);
```

var c ; // hoisted

b is not defined

Variable c exists but doesn't have a value yet

ApplicationsWeb1 - Web Applications I - 2022/2023

35



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

ApplicationsWeb1 - Web Applications I - 2022/2023

36

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_PrecedenceTable

ApplicationsWeb1 - Web Applications I - 2022/2023

37

Assignment

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

Short	Short-hand operator	Meaning
Assignment	<code>x = y</code>	<code>x = y</code>
Arithmetic assignment	<code>x += y</code>	<code>x = x + y</code>
Subtraction assignment	<code>x -= y</code>	<code>x = x - y</code>
Multiplication assignment	<code>x *= y</code>	<code>x = x * y</code>
Division assignment	<code>x /= y</code>	<code>x = x / y</code>
Remainder assignment	<code>x %= y</code>	<code>x = x % y</code>
Exponentiation assignment	<code>x **= y</code>	<code>x = x ** y</code>
Left shift assignment	<code>x <<= y</code>	<code>x = x << y</code>
Right shift assignment	<code>x >>= y</code>	<code>x = x >> y</code>
Unsigned right shift assignment	<code>x >>>= y</code>	<code>x = x >>> y</code>
Bitwise AND assignment	<code>x &= y</code>	<code>x = x & y</code>
Bitwise XOR assignment	<code>x ^= y</code>	<code>x = x ^ y</code>
Bitwise OR assignment	<code>x = y</code>	<code>x = x y</code>

ApplicationsWeb1 - Web Applications I - 2022/2023

38

Comparison operators

Operator	Description	Examples returning true
Equal (==)	Returns true if the operands are equal.	5 == 5 7.0 == 7 0 == 0
Not equal (!=)	Returns true if the operands are not equal.	5 != 4 7.0 != 7 0 != 0
StrictEqual (===)	Returns true if the operands are equal and of the same type. See also (0==0.0) and (0==0.0).	5 === 5 7.0 === 7.0 0 === 0
StrictEqual (===)	Returns true if the operands are of the same type but not equal, or one of different type.	5 === 7.0 7.0 === 5 0 === 0
Greater than (>)	Returns true if the left operand is greater than the right operand.	5 > 4 7.0 > 7 0 > 0
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	5 >= 4 7.0 >= 7 0 >= 0
Less than (<)	Returns true if the left operand is less than the right operand.	5 < 4 7.0 < 7 0 < 0
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	5 <= 4 7.0 <= 7 0 <= 0

Same value?

Same problems as PHP! "5" > 2 > 32

Same type and same value

Comparing Objects

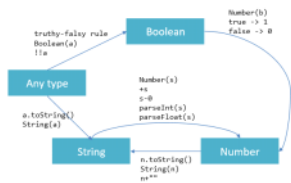
When comparing the internal structure of objects, it compares the reference. We need an "equal" method!

- 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
```

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



+ operator can only apply to numbers! JS tries to first convert it to a number!

<https://github.com/jeremyhu/Don't-know-it-but-it-does-exists/blob/master/184.md>

Logical operators

Used a lot in shortcut computation:

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

(a&&b)->if(a is true) return b
Else return a

(a||b)->if(a is true) return a
Else return b

Common operators

Or string concatenation	Addition (+) Decrement (--) Division (/) Exponentiation (**) Increment (++) Multiplication (*) Remainder (%) Subtraction (-) Unary negation (-) Unary plus (+)	Logical AND (&&) Logical OR () Logical NOT (!) Nullish coalescing operator (??) Conditional operator (?:) typeof	Useful idiom: <code>a b</code> if a then a else b (a, with default b)
-------------------------	---	---	---

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()

Applications Web 1 - Web Applications I - 2020/2021

81



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

Applications Web 1 - Web Applications I - 2020/2021

82

Conditional statements

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

If truthy (beware!)

```
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

Applications Web 1 - Web Applications I - 2020/2021

83

Loop statements

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

Usually declares loop variable

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

May use break; or continue;

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

Applications Web 1 - Web Applications I - 2020/2021

84

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

```
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
i
String>letter by letter

Composite object (array, string)

Iterated

Python in = Java := js of

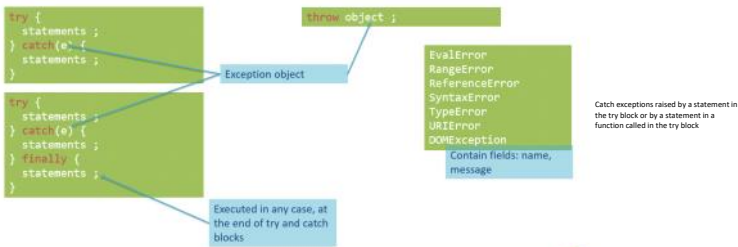
Applications Web 1 - Web Applications I - 2020/2021

85

Other iteration methods

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

Exception handling



JavaScript: The Definitive Guide, 7th Edition
Chapter 6. Arrays

Mozilla Developer Network
JavaScript Guide » Indexed Collections

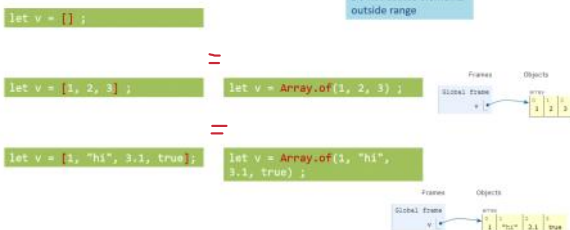
JavaScript – The language of the Web

ARRAYS

Arrays

- Rich of functionalities
- Elements do not need to be of the same type
- Simplest syntax: `[]`
NameOfArray.length (no "l")
- Property `.length`
- Distinguish between methods that:
 - Modify the array (in-place)
 - Return a new array

Creating an array



Adding elements

.length adjusts automatically

```
let v = [];
v[0] = "a";
v[1] = 8;
v.length // 2
```

Frames: Global, v, v[0], v[1]
Objects: v[0] (a), v[1] (8)

let v = [];
v.push("a");
v.push(8);
v.length // 2

Java add
Python append
js push

.push() adds at the end of the array

.unshift() adds at the beginning of the array

31

Adding and Removing from arrays (in-place)



Can be used as static array or variable list:
Easy LIFO stack using Push and Pop
Easy FIFO queue using Push and Shift

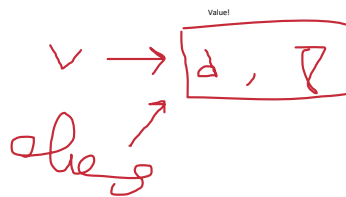
32

Copying arrays

```
let v = [];
v[0] = "a";
v[1] = 8;
let alias = v;
alias[1] = 6;
```

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

console.log(v);
["a", 6]
undeclared
console.log(alias);
["a", 6]
undeclared



33

Copying arrays

```
let v = [];
v[0] = "a";
v[1] = 8;
let alias = v;
let copy = Array.from(v);
```

Array.from creates a shallow copy

Creates an array from any iterable object

Frames: Global, v, alias, copy
Objects: v[0] (a), v[1] (8), alias[0] (a), alias[1] (8), copy[0] (a), copy[1] (8)

Real copy!

Always think if you are modifying a variable or a value!

34

Iterating over Arrays

- 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

Functional style - later

35

Main array methods

- .concat()**
 - joins two or more arrays and returns a new array.
 - .join(delimiter = ',')**
 - joins all elements of an array into a (new) string.
 - .slice(start, upto_index)**
 - extracts a section of an array and returns a new array.
 - .splice(index, count to remove, addElement1, addElement2, ...)**
 - MODIFY 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
 - .reverse()**
 - MODIFY EXISTING ARRAY
 - transposes the elements of an array, in place
 - .sort()**
 - sorts the elements of an array in place
 - .indexOf(searchElement[, fromIndex])**
 - searches the array for searchElement and returns the index of the first match
 - .lastIndexOf(searchElement[, fromIndex])**
 - like indexOf, but starts at the end
 - .includes(valueToFind[, fromIndex])**
 - search for a certain value among its entries.
- Search useful only for basic types!
- Faster, just find if there is or not.

- EXAMPLE: RETURN OF AN ARRAY AND RETURNING A NEW ARRAY.
- `.splice(index, count to remove, addElement1, addElement2, ...)`
MODIFY 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

- returns the **INDEX** of the first match
- `.lastIndexOf(searchElement[, fromIndex])`
– like `indexOf` but starts at the end
- `.includes(valueToFind[, fromIndex])`
– search for a certain value among its entries, returning true or false

Faster, just find if there is or not.

Applications/Week 1 - Web Applications I - 2020/2021

56

Destructuring assignment

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

```
let [x,y] = [1,2];
[x,y] = [y,x]; // swap

var foo = ['one', 'two', 'three'];
var [one, two, three] = foo;
```

Array-like syntax to initialize/assign multiple variables in a single step
`let first = numbers[0];`
`let last = numbers[numbers.length-1];`
OR
`let [first,last] = [numbers[0], numbers[numbers.length-1]]`
Useful for destructuring array, object's properties, or swap 2 elements without 3rd variable
`[a[0], a[1]] = [a[1], a[0]]`
- Useful especially with passing and returning values from functions

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

Applications/Week 1 - Web Applications I - 2020/2021

57

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 x=1, y=[2,3,4] (insert it whole)
```

```
const parts = ['shoulders', 'knees'];
const lyrics = ['head', ...parts, 'and', 'toes']; // ["head", "shoulders", "knees", "and", "toes"]
```

Expands the array, doesn't insert it whole!
- Works on the left- and right-hand side of the assignment

Applications/Week 1 - Web Applications I - 2020/2021

58

Curiosity

`C=[...a,...b] -> concat 2 arrays`
`Copy=[...a]=Array.from(a)=Array.of(...a) -> shallow copy (REAL COPY!)`
(REAL COPY!)

Array+Functions+Object+JavaScript

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

Applications/Week 1 - Web Applications I - 2020/2021

59



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

Mozilla Developer Network
JavaScript Guide » Text Formatting

JavaScript – The language of the Web

STRINGS

Applications/Week 1 - Web Applications I - 2020/2021

60

Strings in JS

NO CHANGING STRINGS IN PLACE!

- 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/Week 1 - Web Applications I - 2020/2021

61

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()`

String methods

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

<https://stackoverflow.com/questions/2243824/what-is-the-difference-between-string-slice-and-string-substring>

Template literals

Can span multiple lines
Can embed expressions

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

## 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. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or 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/>

