**ES.Next**

**C'est quoi ES ?**

* **ECMAScript** ou ECMA-262
* ECMA: European Computer Manufacturers Association
* JavaScript, ActionScript, V8, ...
* Orienté **prototype**

**Un peu d'histoire**

* 1995: création par Brendan Eich
* juin 1997: première version standardisée ECMA
* 1999: troisième version
* 2009: version suivante... la cinquième
* juin 2015: ES6 heart heart_eyes
* ES.Next: juin X: ES(X-2009)

**ES6**

**(R)évolution**

**et officiellement c'est ECMAScript 2015**

**Arrows**

function(v) {

return v + 1;

}

v => v + 1

(v, i) => v + i

v => {

if (typeof v === 'number') {

numbers.push(v);

}

}

**Classes**

class Car extends Vehicle {

constructor(brand, name, price) {

super(brand, name, price);

...

}

}

**Object literals - 1**

var prenom = "Jean";

var nom = "Valjean";

return {

prenom: prenom,

nom: nom,

}

return {

prenom,

nom,

}

**Object literals - 2**

function action(type, data) {

return {

[type]: data,

};

}

**Template strings**

let myVar = "ma variable"

function myFunction(){

return "ma fonction"

}

let myString = `

Les template strings permettent d'utiliser

des chaînes de caractères multi-lignes.

Aussi, on peut afficher la valeur d'une variable

de cette manière: ${myVar}

On peut de la même façon afficher toute sorte

d'expression, comme des retours de fonctions:

${myFunction()}

`

**Destructuring**

const myArray = ["Clark", "Kent", "azerty"]

[firstname, lastname, password] = myArray

const myObj = {firstname: "Kal",

lastname: "El", password: "012345"}

{firstname, lastname, password} = myObj

function myFunc({firstname, lastname}) {…}

myFunc(myObj)

**Let**

function fn() {

let variableLet = "permier let";

var variableVar = "premier var";

if (true) {

let variableLet;

var variableVar;

variableLet = "second let";

variableVar = "second var";

console.log(variableLet); // second let

console.log(variableVar); // second var

}

console.log(variableLet); // premier let

console.log(variableVar); // second var

}

**Const**

function fn(){

const myConst; // erreur, constante doit être

// initialisée

const myConst = "ma constante"

myConst = "modif" // erreur, la constante

// ne peut être réaffectée

if(myConst === "ma constante"){

let myConst = "modif let"

var myConst = "modif var" // erreur, la variable

// est déjà déclarée

}

}

**Modules**

**test.js**

export default class User{

constructor(name, age){

this.name = name;

this.age = age;

}

}

export function printName(user){

console.log(`Nom d'utilisateur ${user.name}`)

}

export function printAge(user){

console.log(`Age de l'utilisateur ${user.age}`)

}

**Module loaders**

**main.js**

import U, {printName, printAge as pa} from '/test.js'

const user = new U('John', 30);

console.log(user)

printName(user) // log: Nom d'utilisateur John

pa(user) // log: Age de l'utilisateur 30

**Map**

const myMap = new Map([[1, "toto"],[2, "truc"]]);

myMap.size // 2

myMap.has(1) // true

myMap.set(3, "titi")

myMap.get(2) // truc

**Set**

const mySet = new Set([1, "toto", "truc"]);

mySet.size // 3

mySet.has(2) // false

mySet.add("titi")

mySet.delete("toto") // "toto"

const nombres = [2,3,4,4,2,2,2,4,4,5,5,6,6,7,5,32,3,4,5];

console.log([...new Set(nombres)]);

// affichera [2, 3, 4, 5, 6, 7, 32]

**Promises**

var promise1 = new Promise(function(resolve, reject) {

setTimeout(function() {

resolve('foo');

}, 300);

});

promise1.then(function(value) {

console.log(value);

// expected output: "foo"

});

console.log(promise1);

// expected output: [object Promise]

**Mais aussi**

* Iterators & Generators
* Binary & Octal
* Symbol
* Intl
* …

**ES 2016**

**ES 2016**

* Array.includes()
* 7\*\*2 raccourci Math.pow(7, 2)

**ES 2017**

**ES 2017**

* Object.values() && Object.entries()
* String.pad{Start,End}

**Async/Await fire**

// ES 2015

function doTheJob(id) {

getUser(id)

.then(runALongProcess)

.then(result => {

console.log(result);

})

}

// ES 2017

async function doTheJob(id) {

const user = await getUser(id);

const result = await runALongProcess(user);

console.log(result);

}

**ES 2018**

**ES 2018**

* Object rest/spread operator
* Promise.finally
* Boucles asynchrones
* RegExp: groupes de captures nommés

**ES 2019**

**ES 2019**

* Array.prototype.{flat,flatMap}
* Object.fromEntries
* String.prototype.{trimStart, trimEnd}
* Optional Catch Binding
* Function.prototype.toString
* Symbol.prototype.description
* améliorations JSON

**ES.Next ?**

[**TC39 Proposal Stages**](https://tc39.github.io/process-document/)

* 0 (Strawman): Entrée dans le process
* 1 (Proposal): A un "Champion", Identifie un problème et propose une solution, specs en cours d'écriture
* 2 (Draft): Formalisation des specs, syntax et la sémantique
* 3 (Candidate): Spec complète, doit être implémenté
* 4 (Finished): Spec complète et 2 implémentations réelles

**Exemples**

[**Proposals**](https://github.com/tc39/proposals)

[Decimal](https://docs.google.com/presentation/d/1jPsw7EGsS6BW59_BDRu9o0o3UwSXQeUhi38QG55ZoPI/edit?pli=1#slide=id.p) (stage 0)

// Number (binary 64-bit floating point)

0.1 + 0.2

=> 0.30000000000000004

// Decimal (???)

0.1m + 0.2m

=> 0.3m

[Object shortand improvements](https://github.com/rbuckton/proposal-shorthand-improvements)

**Initialisation**

// Before

const myObj = { x: a.x };

// After

const myObj = { a.x };

**Destructuration**

// Before

const { x: a.x } = myObj ;

// After

const { a.x } = myObj;

[Optional Chaining](https://github.com/tc39/proposal-optional-chaining) (stage 1)

// Before

var street = user.address && user.address.street;

// After

var street = user.address?.street

// Before

var fooInput = myForm.querySelector('input[name=foo]')

var fooValue = fooInput ? fooInput.value : undefined

// After

var fooValue = myForm.querySelector('input[name=foo]')?.value

// New

myFunc?.()

[Binary AST](https://github.com/tc39/proposal-optional-chaining) (stage 1)

**Abstract Syntax Tree** : Arbre de la Syntaxe Abstraite

est un arbre dont les nœuds internes sont marqués par des opérateurs et dont les feuilles (ou nœuds externes) représentent les opérandes de ces opérateurs. Autrement dit, généralement, une feuille est une variable ou une constante.

[Temporal](https://github.com/tc39/proposal-temporal) (stage 2)

// Date

// time zone not supported, an offset must be used instead

// Whatever provides the offset needs to know when to provide -05:00 vs -06:00 for Chicago.

let timestampInChicago =

Date.parse("2000-12-31T23:59:00-06:00")

let dateInLocalTimeZone =

new Date(timestampInChicago)

let formatterInSydney =

new Intl.DateTimeFormat('en-US',

{ timeZone: 'Australia/Sydney', year: 'numeric',

month: 'numeric', day: 'numeric', hour: 'numeric',

minute: 'numeric', second: 'numeric' })

let formatterInChicago =

new Intl.DateTimeFormat('en-US',

{ timeZone: 'America/Chicago', year: 'numeric',

month: 'numeric', day: 'numeric', hour: 'numeric',

minute: 'numeric', second: 'numeric' })

dateInLocalTimeZone.toISOString()

// 2001-01-01T05:59:00.000Z

formatterInSydney.format(dateInLocalTimeZone)

// 1/1/2001, 4:59:00 PM

formatterInChicago.format(dateInLocalTimeZone)

// 12/31/2000, 11:59:00 PM

// Temporal

let dateTimeAnywhere =

new CivilDateTime(2000, 12, 31, 23, 59)

let instantInChicago =

dateTimeAnywhere.withZone('America/Chicago');

let instantInSydney =

new ZonedDateTime(

instantInChicago.instant, 'Australia/Sydney'

)

let calendarClockDateTimeFromSydney =

instantInSydney.toCivilDateTime()

dateTimeAnywhere.toString()

// 2000-12-31T23:59:00.000000000

calendarClockDateTimeFromSydney.toString()

// 2001-01-01T16:59:00.000000000

[Dynamic import](https://github.com/tc39/proposal-dynamic-import) (stage 3)

import module from 'myModule';

import(`./myModule.js`)

.then(module => {

module.loadPageInto(main);

})

.catch(err => {

main.textContent = err.message;

});

[Private methods](https://github.com/tc39/proposal-private-methods) (stage 3)

class Test {

#value = 0;

get #x() { return #xValue; }

set #x(value) {

this.#xValue = value;

}}

#clicked() {

this.#x++;

}

}

[Static class features](https://github.com/tc39/proposal-static-class-features/) (stage 3)

class Test {

static #color = "#123456";

static #privMethod(color) {…}

static #otherMethod() {

Test.#privMethod(Test.#color)

}

}

**On peut déjà les utiliser**

* nativement
* via un transpileur -- Babel: preset-stage-X -- Typescript: config: --lib et --target

**Sources**

* ["ES Next Features That'll Make You Dance" by Ben Ilegbodu at Node Summit 2018](https://www.youtube.com/watch?v=9yK4t2CuIHQ)
* [ES6 sur Wikipédia](https://en.wikipedia.org/wiki/ECMAScript)
* [medium.freecodecamp.org/here-are-examples-of-everything-new-in-ecmascript-2016-2017-and-2018](https://medium.freecodecamp.org/here-are-examples-of-everything-new-in-ecmascript-2016-2017-and-2018-d52fa3b5a70e)
* [GitHub TC39](https://github.com/tc39)
* [Tous les détails du process](https://tc39.github.io/process-document/)
* [Repo des proposals](https://github.com/tc39/proposals)
* [can i use](https://caniuse.com/)
* [es6-features](http://es6-features.org/)