

Technologie du Web - 3I017

Cours 3 - Javascript

Benjamin Piwowarski

1er février 2022

[Version HTML](#)



Machine Learning &
Deep Learning for
Information Access

Introduction

WHY?

! Le Web/HTML apparaît en 1992

- ✚ Traditionnel : client/serveur et pages statiques
- ✚ L'interaction avec l'utilisateur est... lente
- ✚ Sous-utilisation du client pour des tâches **locales**

SOLUTIONS

- Modules externes : Flash, applets JAVA
 - ✚ Code compilé (machine virtuelle)
 - ✚ Pas d'interaction avec le HTML
 - ✚ ☹ En perte de vitesse...
- JavaScript
 - ✚ Code non compilé (ça change: WebAssembly)
 - ✚ Interaction avec le HTML
 - ✚ ☺ Standard actuel

✚ Dans le cours 2, on ne parle **pas** de l'intégration avec le HTML (abordé au cours 7)
...mais l'influence du Web est importante !

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 3

Introduction

BREF HISTORIQUE DE JAVASCRIPT ET DES TECHNOS WEB

🎓 = sera vu durant le cours de TechnoWeb

- 1989 - [WWW](#) Le réseau interconnecté débute (mail et ressources)
- 1992 - [HTML](#) Language (balises)
- 🎓 1996 - [JavaScript](#) (standard [ECMA-262](#))
- 🎓 2006 - [jQuery](#) : Manipulation du HTML
- 🎓 2010 - [Node.js](#) : JavaScript sort du Web (application en JavaScript)
- 🎓 2012 - [TypeScript](#) : Typage pour JavaScript
- 2013 - [Electron](#) : Applications natives (UI) en JavaScript
- 🎓 2013 - [React](#) : Framework front-end (Facebook)
- 2019 - [Web Assembly](#) (spécification en cours)
 - ✚ Machine Virtuelle Javascript
 - ✚ La fin du JavaScript comme langage de programmation Web (C/C++, Rust, .NET, ...) ?
- 2020 - 11ème version du standard ECMA-262

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 4

Introduction

PLAN

- Language (en TD/TP !)
 - Syntaxe
 - Objets et prototypes
- Programmation asynchrone
- Librairies
 - node.js, npm et paquets
 - CommonJS / ESM
- Avancé
 - Typage (typescript, flowtype)
 - Transpilation

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 5

LANGUAGE

Language

EXCEPTIONS

🔗 Mécanisme d'exception

🔗 comme en C++/Java/Python

🔗 ... mais un seul catch (pas de typage, même dynamique)

```
</> try {  
    // Génère une exception (n'importe quelle valeur  
    // fait l'affaire)  
    throw "monException";  
} catch (e) {  
    // On fait ce qu'on veut  
}
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Language

LE LANGUAGE

🔗 Inspiré par Java et C++

🔗 Notion d'objet atypique (prototypes)

Si vous voulez tout savoir...

🔗 [Javascript - Documentation Mozilla](#)

EN ATTENDANT...

🔗 On retrouve les blocs

🔗 On retrouve les mots-clé if, for, while, break, continue, switch

🔗 Le ; est optionnel en fin de ligne

```
</> for(let i = 0; i < 5; ++i) {}  
  
/* Boucle while */  
while (true) {  
    if (keyboardPressed()) break  
    i += 1  
}
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 8

Language

MESSAGES

🔗 Pas de notion de "sortie standard"

JavaScript permet d'afficher des messages via sa console

```
</> console.debug("Voici un message de debug")  
console.log("Voici un message")  
console.warn("Voici un avertissement")  
console.error("Voici une erreur")
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 10

Slide 9

LES VARIABLES

PLUSIEURS TYPES DE DÉCLARATION DE VARIABLES

- Variables constantes avec `const` (portée = bloc)
- Variables avec `let` (portée = bloc)
- Variables avec `var` (portée = fonction)
- (à ne pas utiliser) sans déclaration = comme `var`

</> Exemple

```
var j = 'Technos Web' // Variable globale
const pi = 3.14 // Constante

function f(){
  var k = 'Javascript' // Variable locale à la fonction
  let l = 20 // Variable locale (bloc)
  return pi * l // pi est accessible ici (variable globale)
}
// k et l ne sont plus accessibles ici
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 11

OBJETS FONDAMENTAUX

OBJETS

Un dictionnaire associe un identifiant ou chaîne de caractère à une valeur quelconque

</> ↗

```
var dico = {a: 1, b: 2, c: 3}
console.log(dico.a) // ou dico["a"]
```

🔗 On peut accéder facilement à un sous-ensemble du dictionnaire

</> ↗

```
// Décomposition
var dico = {a: 1, b: 2, c: 3}
let { b, c } = dico
```

```
// Boucles (sur les clefs)
for(let key in dico) {
  console.log(key, dico[key])
}
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 13

TYPES DE DONNÉES

IL Y A PEU DE TYPES DE DONNÉES

boolean

true ou false

number

Tout nombre (entier ou réel)

string

Une chaîne de caractères

function

Une fonction

object

Un objet (cf. plus tard)

undefined

Une variable sans valeur

</>

```
let a
console.log(a) // Affiche "undefined"

console.log(typeof "hello world") // Affiche string
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 12

OBJETS ET JSON

! JSON

W JavaScript Object Notation (JSON)

🔗 Format de donnée très utilisé pour l'échange de données sur le Web

🔗 Il est très proche en notation des objets JavaScript

</> ↗

```
// Renvoie { result: true, count: 42 }
let o = JSON.parse('{"result":true, "count":42}')
```

```
// Renvoie la chaîne '{"result":true,"count":42}'
console.log(JSON.stringify(o))
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 14

OBJETS FONDAMENTAUX

TABLEAUX

```
</> var a = [1, 2, 4]
// Affiche 3 1
console.log(a.length, a[0])

// Décomposition
let [a1, a2, a3] = a
let [a, ...rest] = a // rest == [2, 4]

// Boucle : Affiche 1, puis 2, puis 4
for(let i of a) {
  console.log(i)
}
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 15

LES FONCTIONS

Une fonction est un bloc d'instructions acceptant une liste de paramètres

- Définie par le mot-clé `function`
- Peut posséder un nom ou non
- Retourne éventuellement une valeur (grâce à `return`)

</> Fonctions

```
// Fonction nommée
function nom_fonction(parametre_1, ... , parametre_n) {
  instructions
  return expression
}

// Fonction anonyme
let f = function(parametre_1, ... , parametre_n) {
  instructions
  return expression
}
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 16

LES FONCTIONS (II)

</> Fonctions

```
// Fonction "fléchée"
let plus = (a, b) => a + b
// Avec décomposition
let plus_arg = ({a, b}) => a + b
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 17

LES FONCTIONS - PARAMÈTRES

Un nom de fonction est **unique** (pas d'overloading)

LES PARAMÈTRES PEUVENT NE PAS ÊTRE TOUS DONNÉS (OU ON PEUT EN DONNER PLUS...)

```
</> function f(a, b, c) {
}

f(1, 2) // c == undefined
f(1, 2, 3, 4) // a == 1, b == 2, c == 3 (4 ignoré )
```

NOMBRE DE PARAMÈTRES VARIABLES

```
</> function f(a, ...r) {
}

f(1, 2, 3) // r == [2, 3]
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 18

LES FONCTIONS

UNE FONCTION EST UNE VARIABLE COMME LES AUTRES

Mécanisme très utilisée pour les "callbacks"

```
</> // function plus(a,b) { return a + b }
let plus = (a, b) => a + b

function f(op, a, b) {
  console.log(op(a, b))
}

console.log(f(plus, 2, 3)) // 5
```

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

LA NOTION D'OBJET: HÉRITAGE

Les classes existent depuis ECMAScript 2015

[Les classes javascript](#)

[Voir la documentation sur this](#)

```
</> class X {
  constructor(x) {
    this.x = x
  }
  static f() { /* membre statique */ }
  g() {
    console.log("X =", this.x)
  }
}

X.f() // appel statique
let x = new X(1) // nouvelle instance
x.g() // Affiche X = 1
```

Slide 19

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 20

CLASSES ET HÉRITAGE

On peut créer des sous-classes avec extends

```
</> class Y extends X {
  g() {
    super.g()
    console.log("Y =", this.x)
  }
}

let y = new Y(1) // nouvelle instance
y.g() // Affiche X = 1 \n Y = 1
```

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

SUCRES SYNTAXIQUES

INTERPOLATION POUR LES CHAÎNES DE CARACTÈRES

🔗 Substitution d'une expression

```
</> let world = "world"

// Appelle world.toString()
console.log(`Hello ${world}`)

// On peut aussi avoir des expressions plus complexes
let x = 1
console.log(`Hello ${x + 1}`)
```

Slide 21

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 22

SUCRES SYNTAXIQUES (II)

CHAÎNAGE DE PROPRIÉTÉS (ECMA 2020)

```
</> let o = { a: { b: 1, c: 2 } }

console.log(o.a?.b) // 1
console.log(o.d?.b) // undefined
console.log(o.d.b)  // erreur
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 23 **PROGRAMMATION ASYNCHRONE**

Programmation asynchrone

BOUCLE D'ÉVÉNEMENTS

! Problème

📁 Dans le web, beaucoup d'événement asynchrones

- Actions des utilisateurs
- Accès des API Web

📁 La programmation multi-thread est difficile

? Solution

- 📁 Processus unique (monothread)
- 📁 Programmation asynchrone pour les opérations longues (I/O)

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Programmation asynchrone

BOUCLE D'ÉVÉNEMENTS

! Event Loop (Boucle d'événements)

JavaScript est **mono-processus (monothread)**

📁 Chaque message (tâche = ensemble d'instructions) est exécuté totalement

📁 **pseudo-multitâche I/O asynchrone**

</> Pseudo-algorithme

```
while (queue.attendreMessage()){
  // En vrai, il y a plusieurs files d'attentes
  // (I/O, micro et macro-tâches)
  queue.traiterProchainMessage();
}
```

🔗 En savoir plus...

Slide 26 [Technologie du web](#) / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 27

PROGRAMMATION ASYNCHRONE : CALLBACKS

</> Blocs d'exécution

```
// Quand est-ce que je m'affiche ?
let now = Date.now()
console.log("[start]", Date.now() - now, "ms")
setTimeout(() => { console.log(Date.now() - now, "ms") }, 1000)
setTimeout(() => { console.log(Date.now() - now, "ms") }, 1100)
setTimeout(() => { console.log(Date.now() - now, "ms") }, 1200)
console.log("[end]", Date.now() - now, "ms")
```

[setInterval](#) est une fonction asynchrone : la fonction de callback sera placée dans la pile d'exécution dès l'expiration du délai, et exécuté dès que possible. Il existe une autre fonction lié au temps, [setTimeout](#) (appel à intervalle régulier)

[Technologie du web](#) / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 28

PROGRAMMATION ASYNCHRONE : CALLBACKS

</> Liste des fichiers d'un répertoire: asynchrone

```
reqapi(
  `http://www.monapi.fr/user?name=${name}`,
  (err, response) => {
    if (err) {
      // Aie...
      return
    }

    attenteChoix(reponse, (choix) => {
      if (choix) {
        reqapi(
          `http://www.monapi.fr/message?user_id=${id}`,
          affiche_messages)
        }
      })
    })
  )
```

Comment faire pour avoir un temps limité ?!

[Technologie du web](#) / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 30

BOUCLE D'ÉVÉNEMENTS: EXEMPLE

Appel d'une API : recherche d'un utilisateur

On attend la réponse

Affichage d'un choix pour l'utilisateur pendant un temps limité

On attend que l'utilisateur clique

Appel d'une API : Affichage de la carte une fois le choix effectué

On attend la réponse

Pendant l'attente I/O, d'autres parties du code peuvent s'exécuter

[Technologie du web](#) / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 29

PROGRAMMATION ASYNCHRONE : PROMISE

Pour travailler de manière plus confortable en asynchrone, on utilise les *Promise* (promesse)

</> Utilisation des promesses

```
let promise = maFonctionAsynchrone()

promise.then((value) => {
  console.log("Got", value)
}).catch(() => {
  console.error("Arggggg...")
}).finally(() => {
  console.log("Time to cleanup up")
})
```

En savoir plus [Document Mozilla](#) (Promise)

[Technologie du web](#) / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 31

PROGRAMMATION ASYNCHRONE : PROMISE (ENCHAINEMENT)



- ## Chainage de promesses

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 32

Slide 33

PROGRAMMATION ASYNCHRONE : EXEMPLE

Transformer un callback en promesse

```
function resolveAfter2Seconds(error) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      (error ? reject : resolve)('2s elapsed!')
    }, 2000)
  })
}

resolveAfter2Seconds(true)
  .then((e) => { console.log("[1] Done", e) })
  .catch((e) => console.log("[1] Erreur", e))
resolveAfter2Seconds(false)
  .then((e) => { console.log("[2] Done", e) })
  .catch((e) => console.log("[2] Erreur", e))
```

Slide 34

Slide 35

PROGRAMMATION ASYNCHRONE: AWAIT/ASYNC

👉 Permet de réduire la quantité de code
await peut seulement être appelé depuis une fonction asynchrone (**async**)

```
</>
async function asyncCall() {
  // Une exception est lancée si problème (=> catch)
  const result = await resolveAfter2Seconds()
  console.log("Waited 2 seconds", result)

  // Il faut renvoyer un "Promise" (ou rien)
  return Promise.resolve(result)
}

// exécution asynchrone (r est un Promise)
let s = asyncCall().then(r => console.log("Returned", r))
```

🔗 Documentation des fonctions asynchrones

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 36

Modules

PROBLÈME

ORIGINES DU MAL

Les librairies, c'est pratique...
 mais pas prévu au départ dans JavaScript

🗨️ **Pas de mécanisme d'inclusion**

Répartition du même code à exécuter en plusieurs fichiers

- ❑ Conflit sur les variables
- ❑ Inclusions au niveau global (balise "<script>" en HTML)

ÇA DONNE ÇA

```
</> (presque actuel) Une seule inclusion depuis HTML

...
<script src="librairie1.js"></script>
<script src="librairie2.js"></script>
<script src="moncode.js"></script>
...
```

- ❑ très difficile à utiliser
- ❑ De moins en moins possible quand on augmente le nombre de librairies utilisées

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

MODULES

Modules

SOLUTIONS

C'EST LE B... !

- 2009 - Common JS (CJS) : inclusion dynamique `require (" . . . ")`, porté par Node JS
- 2010 - Asynchronous module definition (AMD) / RequireJS : adaptation pour les navigateurs... mais 📄
- ES Modules (ESM) : inclusion *statique* ou *dynamique* (pas encore totalement standard)

! **Solutions très différentes**

Incompatibilité (mais on peut utiliser un module CJS avec ESM)
 Plus de détails sur les origines du mal
 🔗 Node Modules at War

👉 Ici, on ne présentera que ce qui vous servira :

- CommonJS (utilisé par défaut par Node.JS... mais transition en cours)
- ESM (le nouveau standard, utilisé par les navigateurs modernes et node.js 14+)

👉 Points communs = export/import de symboles

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 40

COMMONJS: EXPORTER DES SYMBOLES

- 🔗 Dynamique avec la fonction `require(id: string)`
- 🔗 Synchrone

🔗 Ce qu'on exporte est défini par `module.exports`

🔗 Exporter de symboles (util.js)

```
module.exports = {
  sum: (x, y) => x + y,
  PI: 3.14
}
```

🔗 ... et le `require` permet de récupérer cette valeur

🔗 Importer des symboles

```
const {sum, PI} = require('util')
// Affiche "5 3.14"
console.log(sum(2, 3), PI)
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

ESM: EXPORTER/IMPORTER DES SYMBOLES

- 🔗 Statique (principalement)
- 🔗 Asynchrone

🔗 Documentation

🔗 Exporter de symboles (util.js)

```
// Il y a plein d'autres façons d'exporter
let sum = (x, y) => x + y

export { sum }
export PI = 3.14
```

🔗 Importer des symboles

```
import {sum, PI} from 'util'
// Affiche "5 3.14"
console.log(sum(2, 3), PI)
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 41

Slide 42

EXPORTER PAR DÉFAUT

🔗 On peut aussi exporter un objet par "défaut"

🔗 Export par défaut

```
// @filename: util.mjs
export default (x, y) => x + y;
```

🔗 Importer

```
// @filename: util.mjs
import yopyop from 'util'
console.log(yopyop(2, 3))
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

IMPORT DYNAMIQUE

CADRE D'UTILISATION

- 🔗 Fonctionne dans la plupart des navigateurs
- 🔗 Pour utiliser ESM avec le Node.JS actuel, utilisez l'extension `.mjs`
- 🔗 (ou bien le transpileur `typescript`)

IMPORTS DYNAMIQUES

! Standard en cours de finalisation (mais le plus probable)

Très utile pour éviter de charger des modules lourds

🔗 Mécanisme de "Promise"

Slide 43

```
import("util").then(module => {
  console.log(module.sum(2, 3))
})
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 44

CRÉER UNE LIBRAIRIE / PROGRAMME

Deux concurrents principalement [npm](#) et [yarn](#)

</> Création du répertoire

```
# Création du répertoire
mkdir node_project
cd node_project

# Initialisation du module (répondre aux questions)
npm init
```

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 45

CRÉER UNE LIBRAIRIE / PROGRAMME (II)

</> Fichier package.json créé dans le répertoire

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Description du projet",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Mon nom",
  "license": "ISC"
}
```

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 46

AJOUT DE DÉPENDANCES

🔗 On peut maintenant ajouter des modules (ajoutés dans le répertoire `node_modules`)

</> Ajout d'un module

```
# Ajoute une dépendance
npm add NOM_DU_MODULE
# ou pour un module de développement (modules)
npm add -d NOM_DU_MODULE
```

🔗 Un répertoire `node_modules` sera créé avec un dossier par module ajouté

🔗 On peut utiliser `require` ou `import`

</> Import de random (CommonJS)

```
const random = require('random')
random.float(0, 1) // uniform float in [ min, max )
```

</> Import de random (ESM)

```
import random from 'random'
random.float(0, 1) // uniform float in [ min, max )
```

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 47

MODULES STANDARDS (NODE.JS)

node.js vient avec [une](#) librairie standard

En particulier, on peut citer :

- Un certain nombre de [variables](#) globales sont prédéfinies
 - `console` (affichage)
 - `require` (import)
 - `setTimeout`, `setInterval` (délai)
- `fs` qui permet d'accéder au système de fichiers
- `path` qui permet de manipuler les chemins (fichiers)

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 48

TYPAGE : TYPESCRIPT

[Documentation](#)

TYPESCRIPT

🔗 Utile pour repérer les erreurs

```
</> type SquareConfig = {
  color?: string;
  width?: number;
}

function f(config: SquareConfig) {
  // Property 'clor' does not exist on type 'SquareConfig'.
  // Did you mean 'color'?
  config.clor

  // The left-hand side of an arithmetic operation must be
  // of type 'any', 'number', 'bigint' or an enum type
  let x = config.color + 3
}
```

TYPAGE

! Applications complexes

JavaScript est interprété = développer des applications complexes est difficile

🔗 *changement du nom d'une variable, mise à jour d'une librairie...*

💡 Ajouter des types

Analyse statique du typage du code

🗨 Inférence de type

On peut inférer les types (comme rust, OCaml, C++...)

```
</> const x = 3
      let y = 2 * x
```

🔗 On sait que 6 est un nombre

PLUSIEURS SURCOUCHES SYNTAXIQUES :

🔗 [TypeScript](#) : typage strict

🔗 [Flow](#) : typage progressif

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

TYPAGE : TYPESCRIPT

PRINCIPES

🔗 Ajout du type des variables

🔗 Inférence

🔗 Définition de types : types, interfaces ou namespaces

QUELQUES EXEMPLES...

```
</> let color: number
function sum(a: number, b: number) : number { ... }

// Type d'un objet
let a : {
  a: number
  b: string
} = { a: 2.3, b: "hello world " } // OK

// Classe
class MaClasse {
  // Variables instance
  reward: number
}
```

TYPAGE : TYPESCRIPT - EXEMPLES

POSSIBILITÉ DE DÉFINIR DES TYPES

```
</> type A = {
  a: number
  b: string
}

// OK
let a: A = { a:1, b: "yep" }

// Property 'b' is missing in type '{ a: number; }'
// but required in type 'A'.ts(2741)
let b: A = { a: 1 }

// Type 'number' is not assignable to type 'string'.ts(2322)
let c: A = { a: 1, b: 1.2 }
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 54

TYPAGE : TYPESCRIPT - EXEMPLES

POSSIBILITÉ DE DÉFINIR DES INTERFACES (PROTO-CLASSES)

```
</> interface Shape {
  color: string;
}

interface PenStroke {
  penWidth: number;
}

interface Square extends Shape, PenStroke {
  sideLength: number;
  scale(factor: number) : Square;
}
```

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 55

C'EST LE B...

! Typage, différentes versions ECMA, extensions...

JavaScript est un langage *interprété* utilisé dans des environnements multiples en C/C++, Java on compile (code natif ou pour machine virtuelle)
Comment faire pour que cela soit portable ?

Comment faire pour s'y retrouver ???

Comment faire pour utiliser TypeScript, Flow, etc. ???

Technologie du web / B. Piowowski / Cours 3 - Javascript / 1er février 2022

Slide 56

MÉTHODE 1: LA DOCUMENTATION

COMMENT S'Y RETROUVER ?

Dans la documentation mozilla, on trouve ce tableau à la fin de chaque page

Browser compatibility

Report problems with this compatibility data on GitHub

	Desktop						Mobile						Node.js
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox Android	Opera Android	iOS Safari	Samsung Internet	Node.js
Optional chaining operator (?.)	80	80	74	No	67	13.1	80	80	79	No	13.4	No	14.0.0

Full support

No support

User must explicitly enable this feature.

Exemple: [chaînage de propriétés](#)

Oui mais... j'aimerais bien programmer en javascript moderne !

MÉTHODE 2 : LA TRANSPIRATION

La transpilation

On compile du code... en code

</> Version typescript, ECMA 2015

```
class A {
  x: number
  constructor(x: number) { this.x = x }
}

let a = new A(1)
```

</> Version Javascript, ES 5 (2009)

```
var A = /** @class */ (function () {
  function A(x) {
    this.x = x;
  }
  return A;
})();
var a = new A(1);
```

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

TRANSPIRATION

IL Y A PLEIN DE VARIANTES

Javascript (ou presque) vers javascript

[Typescript](#) (si vous le souhaitez)

[Babel](#)

EXTENSION DE JAVASCRIPT

[React](#) (cours et TP)

[Svelte](#)

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

MÉTHODE 2 : LA TRANSPIRATION (... MAIS)

! Certaines méthodes ne sont pas présentes dans de vieilles versions

Solution = polyfill

```
</> if (!String.prototype.matchAll) {
  require('./matchAll-polyfill.js')
}
```



Il existe une librairie qui fait ça, [Polyfill.io](#)

Technologie du web / B. Piwowarski / Cours 3 - Javascript / 1er février 2022

Slide 59

EN RÉSUMÉ

Slide 60

RÉSUMÉ

🔗 JavaScript = langage pour le Web, asynchrone et mono-thread

- Très similaire à C++/Java
- I/O asynchrone
- Programmation asynchrone (Promise)

🔗 Gestion des librairies = CommonJS (Node.JS) et ESM (Web)

🔗 Avancé : Inférence de type (TypeScript, Flow) et Transpilation

🟢 Beaucoup plus de standards qu'avant (bon, c'est pas parfait)

🔴 Beaucoup de technologies nouvelles à surveiller constamment

! But du TME 2

Vos premiers pas avec JavaScript / programmation asynchrone

CE QU'ON VERRA PLUS TARD

🔗 Web services

🔗 Intégration avec le Web : API Web, JQuery, React (la semaine prochaine !), Bundlers

[Technologie du web](#) / B. Piwowarski / Cours 3 - Javascript / 1er février 2022