JXC

ESIR3

WEB ENGINEERING : FRONTEND

Décembre 2021 – Janvier 2022

P L A N

Généralités et rappel

Développement Frontend

HTML et CSS

JavaScript

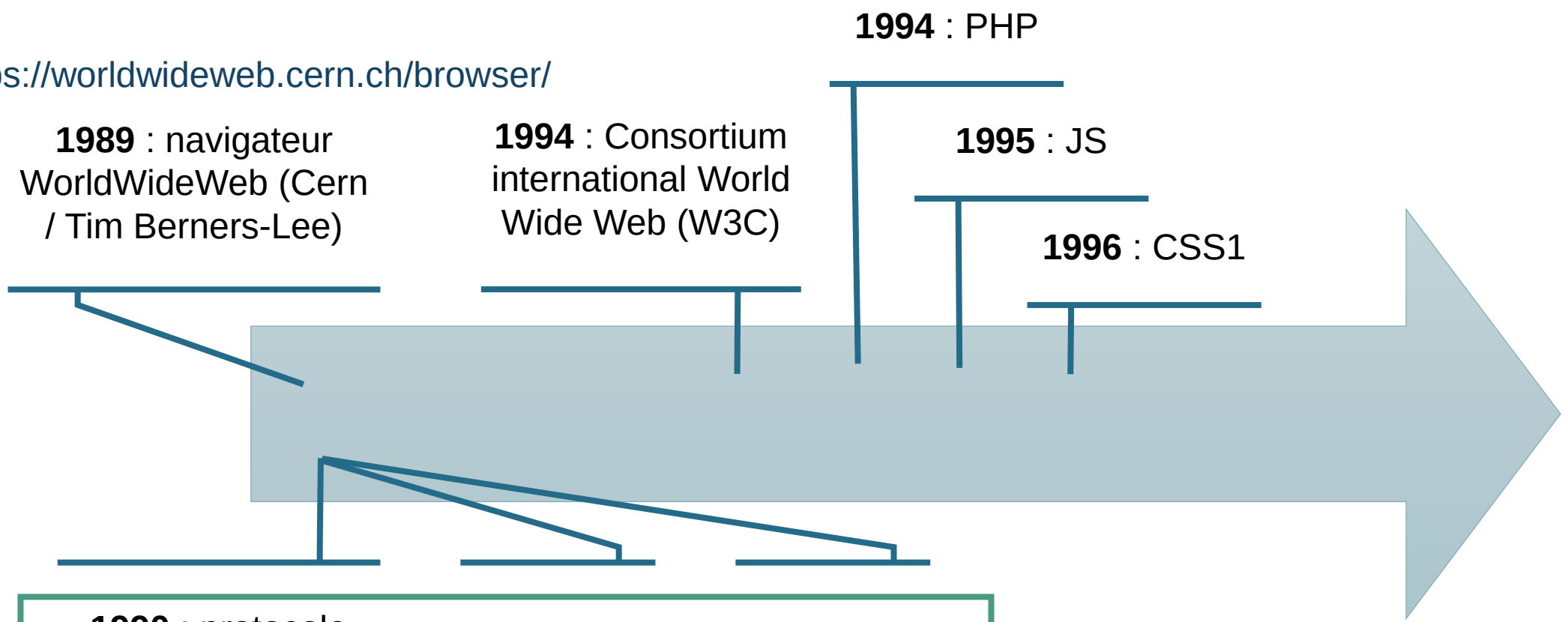
Utilisation de Framework : Angular

Sécurité

Conclusion

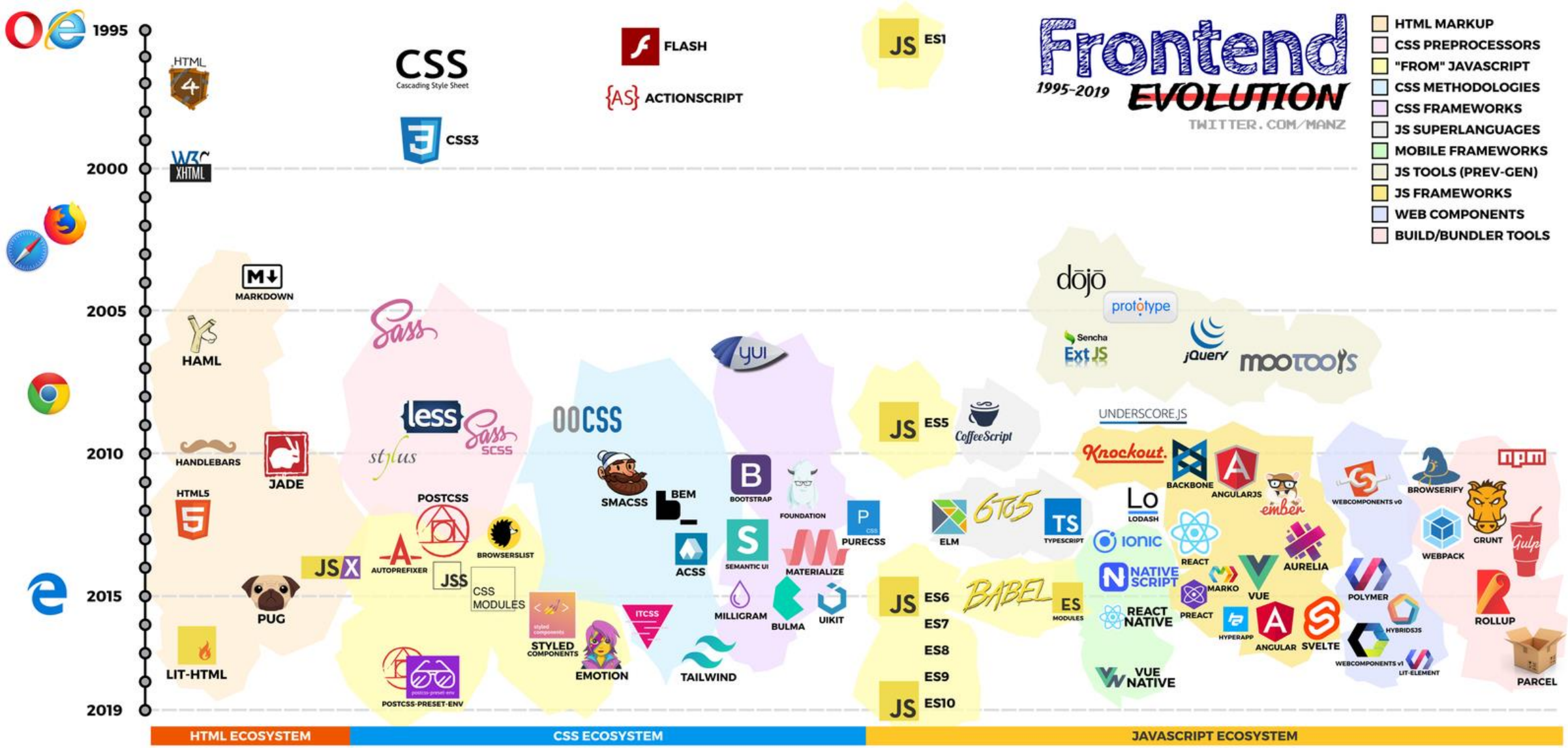
GÉNÉRALITÉS

https://worldwideweb.cern.ch/browser/



Piliers du Web

GÉNÉRALITÉS



G É N É R A L I T É S

Evolution du Web

→ Séparation mise en forme (CSS)

→ Pages statiques vers interactions dynamiques

→ Cloud computing, Web sémentique, Web embarqué, etc.

→ HTML5 (Conteneur d’applications complexes)

GÉNÉRALITÉS

Evolution du Web

→ **Web 2.0** (web collaboratif, wiki, blog, etc.)

Web 3.0 avec le web sémantique et les objets connectés

→ **Moyens techniques**

Langage de scripts

Côté serveur (PHP, ASP, C#, etc.)

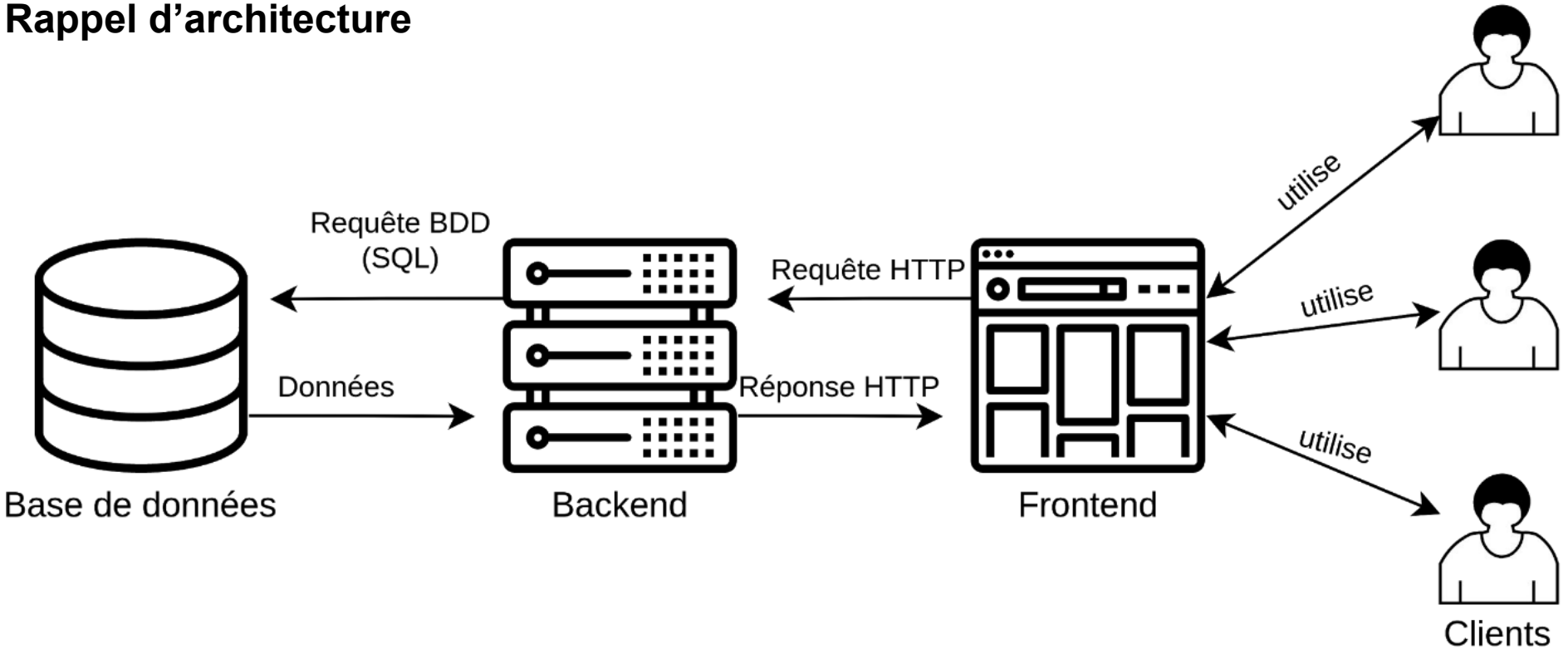
Côté client (Javascript, flash, etc.)

→ **Services Web**

Échange de données entre applications (HTML / XML, JSON)

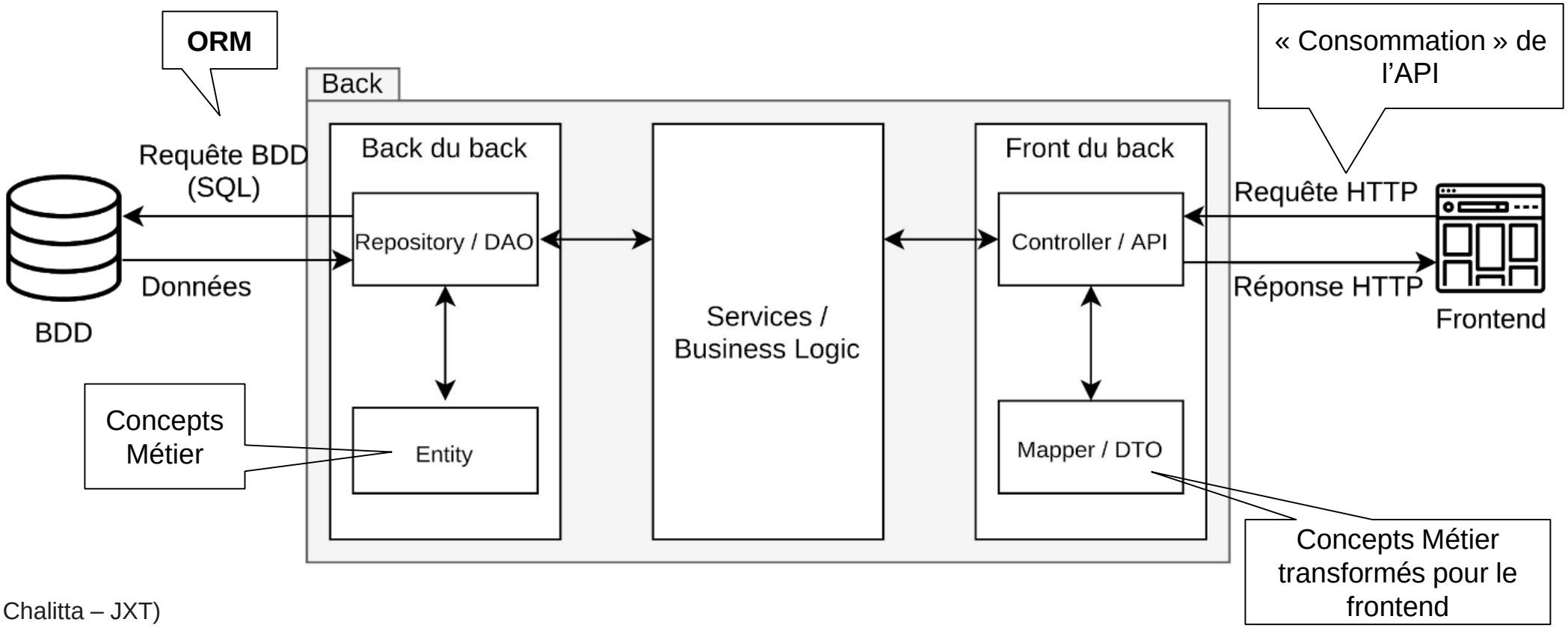
GÉNÉRALITÉS

Rappel d'architecture



G É N É R A L I T É S

Rappel backend



GÉNÉRALITÉS

Pourquoi utiliser des frameworks ?

→ Simplifier la vie du développeur et réduire le coup

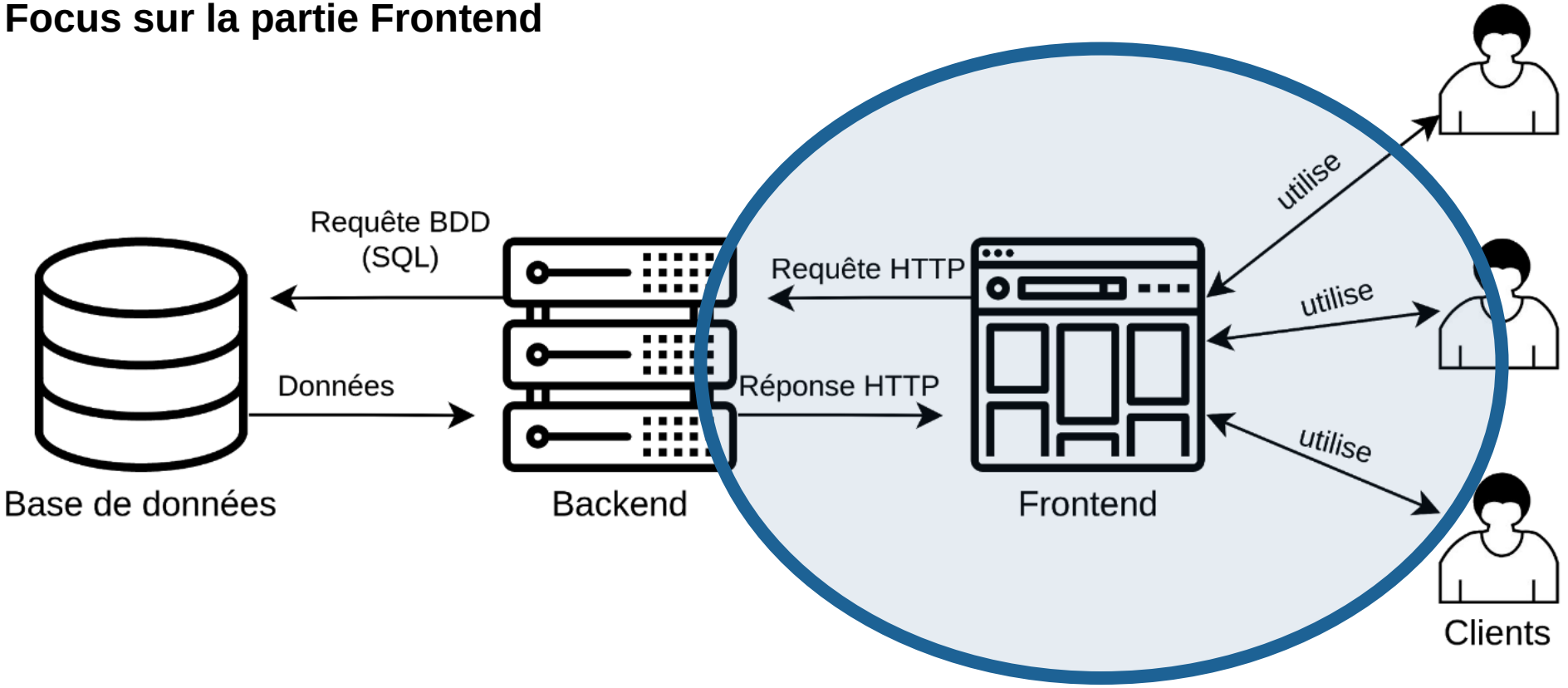
Différents types d'architectures (push vs pull based architecture) :

→ Basé sur des actions (Django, Ruby on Rails, Symfony)

→ Basé sur des composants (Vue, Angular, React)

GÉNÉRALITÉS

Focus sur la partie Frontend



OBJECTIFS

→ Fondamentaux sur le frontend web (Single Page Application)

→ Rappel sur les technologies de base du web (HTML/CSS/JS)

→ Rappel TypeScript

→ Panorama des frameworks JavaScript

ORGANISATION

→ Suite cours backend (JXT – ESIR2)

→ Slide disponible sur moodle

→ Cours : 16h

→ TP : 32h

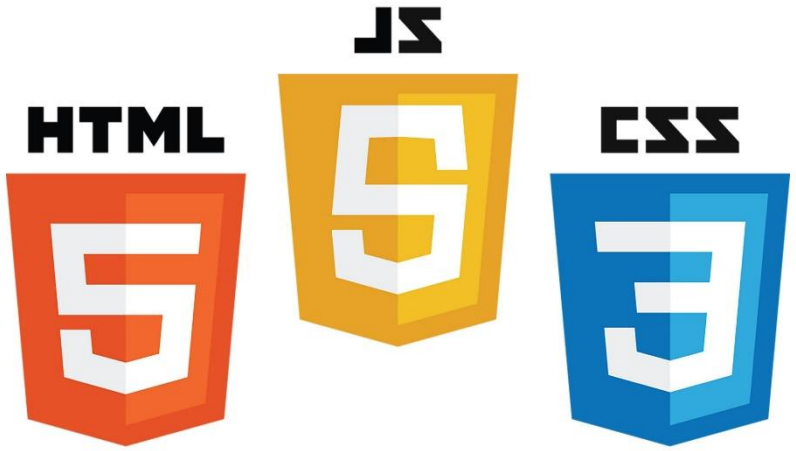
DÉVELOPPEMENT FRONTEND

Les piliers du web

Architecture front-end : MPA - SPA

DÉVELOPPEMENT FRONTEND

→ **HTML / CSS / JavaScript**



→ De nombreuses variations à JavaScript existent.

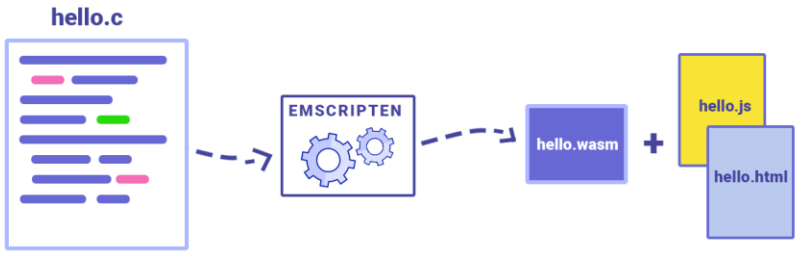
DÉVELOPPEMENT FRONTEND

Création du nouveau standard du web en 2019 : WebAssembly (wasm)

→ Ne remplace pas JavaScript mais devient complémentaire.

→ Bas niveau (de style assembleur) avec un format binaire (Code C++ de base transformé en binaire *.wasm* via *Emscripten*)

→ Permet un gain de performance (format plus léger qu'un fichier JS, déjà compilé en amont, etc.)

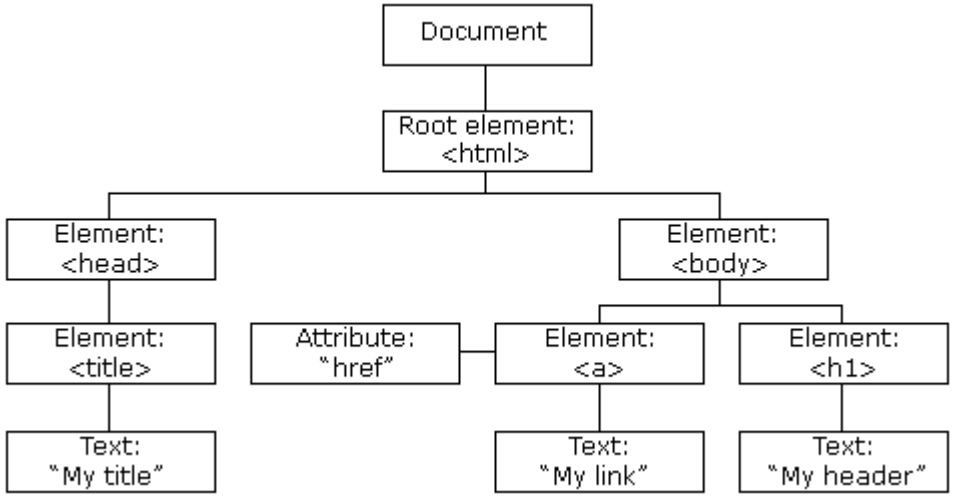


DÉVELOPPEMENT FRONTEND

DOM (Document Object Model)

→ Représentation objet des données qui composent la structure et le contenu d'un document sur le web.

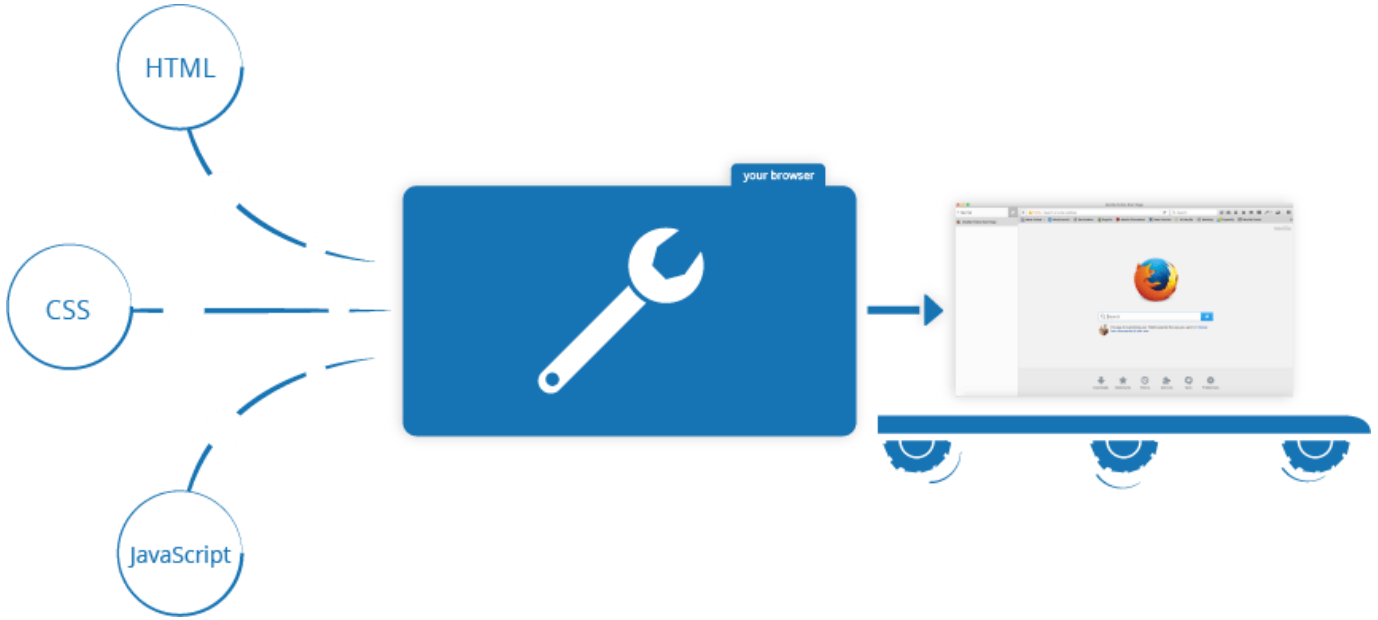
→ Il peut être manipulé à l'aide d'un langage script comme JavaScript.



DÉVELOPPEMENT FRONTEND

→ Lorsque la page web se charge dans un navigateur, les codes HTML, CSS et JavaScript s'exécutent dans un environnement.

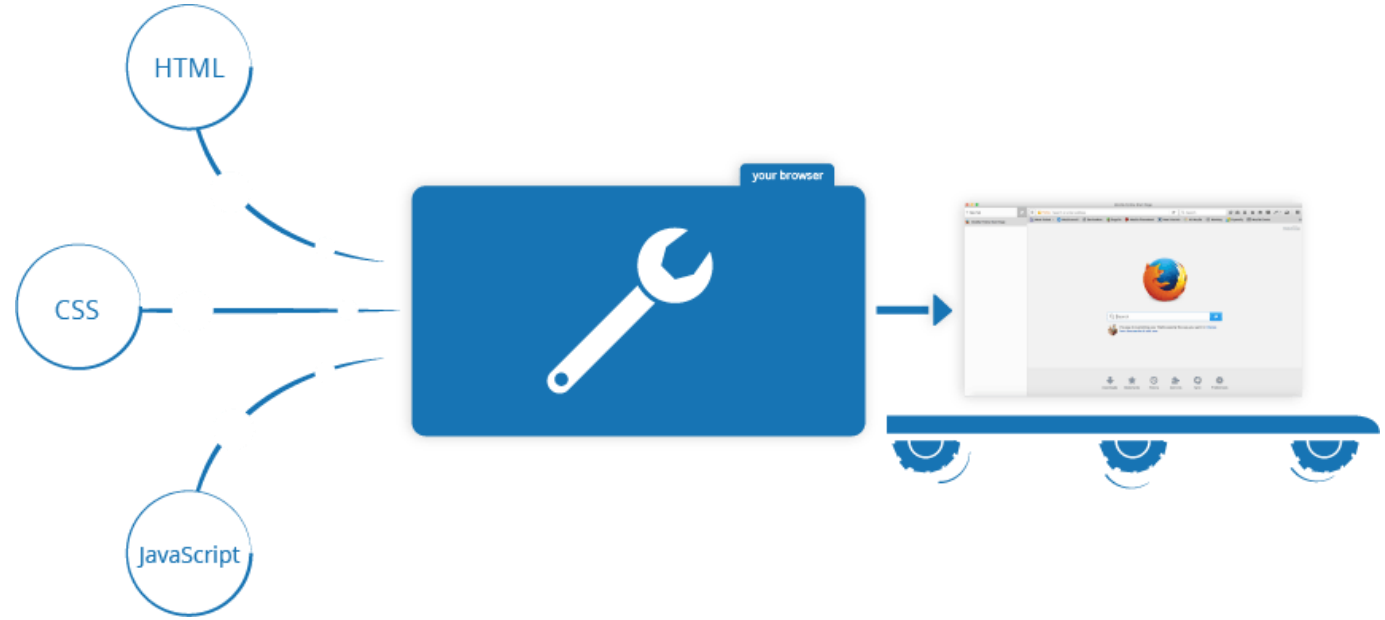
→ Le JavaScript est exécuté par le moteur JavaScript du navigateur, après que le HTML et le CSS ont été assemblés et combinés en une page web.



DÉVELOPPEMENT FRONTEND

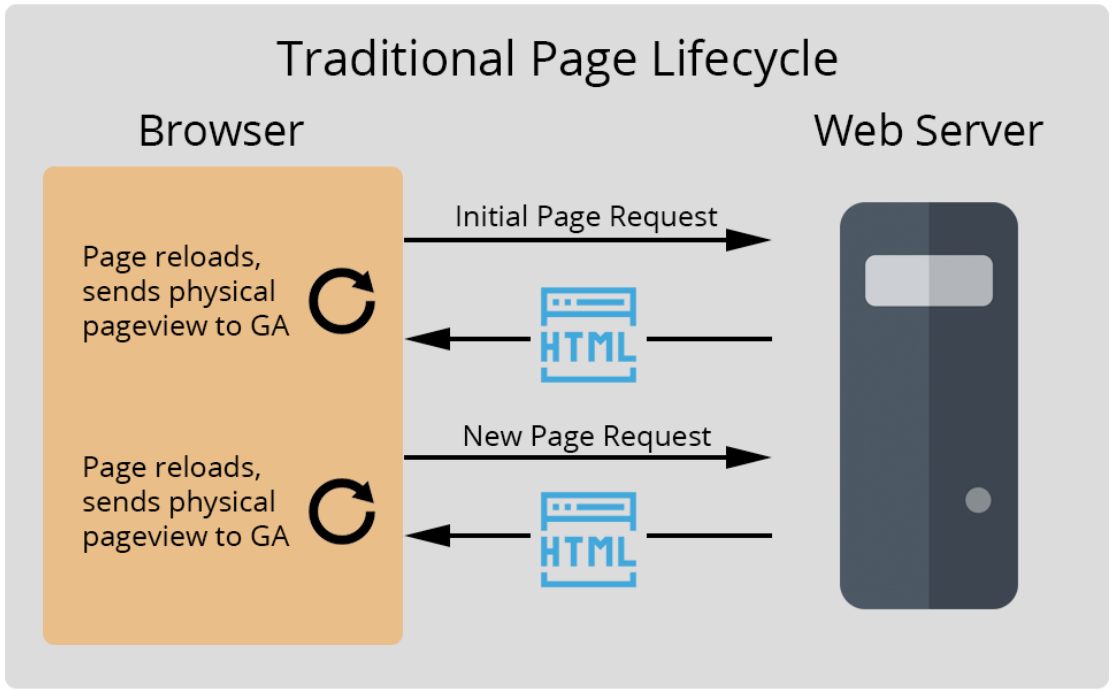
→ Possibilité d'utiliser différentes technologies pour créer des applications web

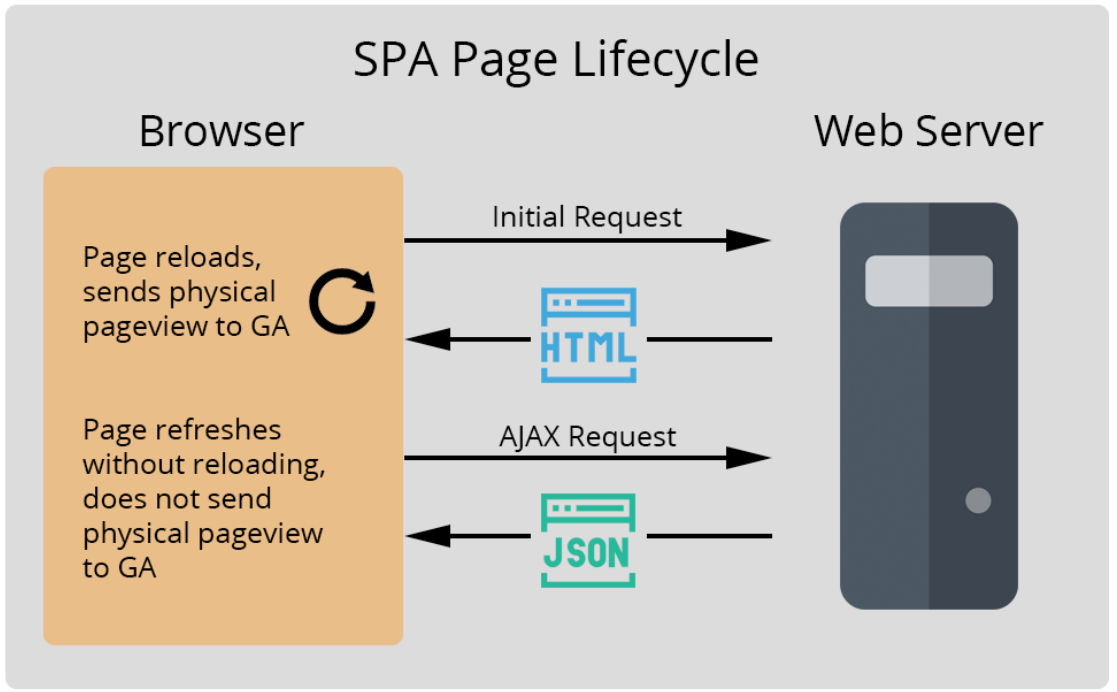
- VanillaJS
- Langages exitants (GWT)
- Nouveaux langages (DART)
- Extension de JS (TypeScript, Coffee)
- Frameworks JS (Angular, Vue, React,)



MULTIPAGES APPLICATION VS SINGLE PAGE APPLICATION

Différentes possibilités d'architecture pour le front





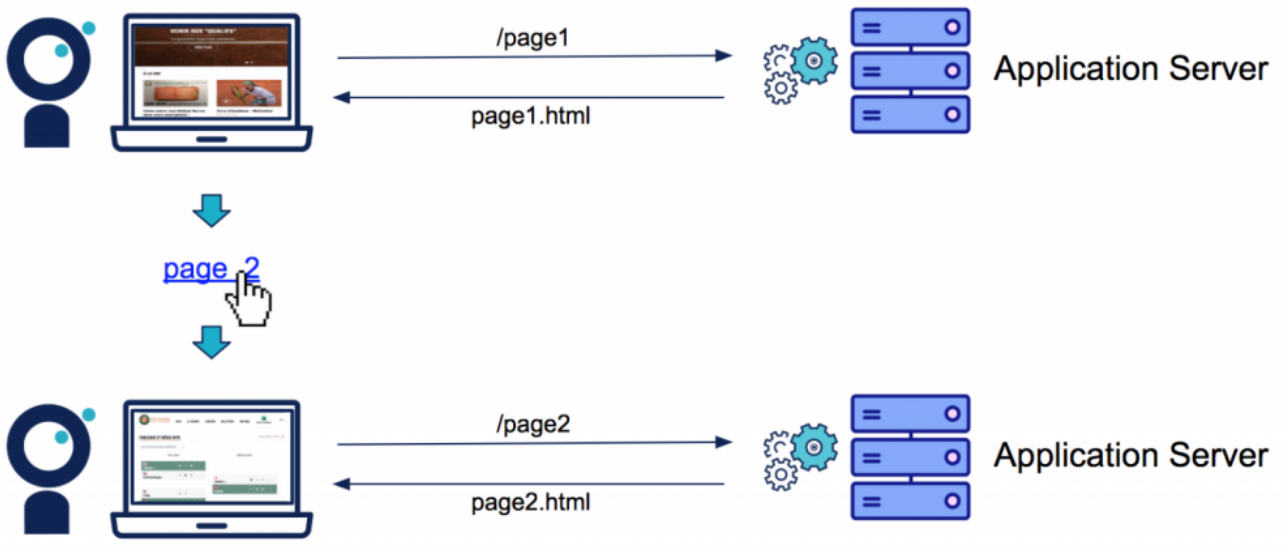
(Stéphanie Chalitta – JXT)

MULTIPAGES APPLICATION VS SINGLE PAGE APPLICATION

Multiple page application (MPA)

→ Chaque action de l'utilisateur déclenche une requête HTTP vers le serveur.

→ Rechargement complet de la page même si une partie du contenu reste inchangée.



MULTIPAGES APPLICATION VS SINGLE PAGE APPLICATION

Single page application (SPA)

→ Type de pattern apparu avec l'arrivée de nouvelle technologie (AJAX en 2004, jQuery en 2006, Google Chrome avec un nouveau moteur JavaScript V8, etc.).

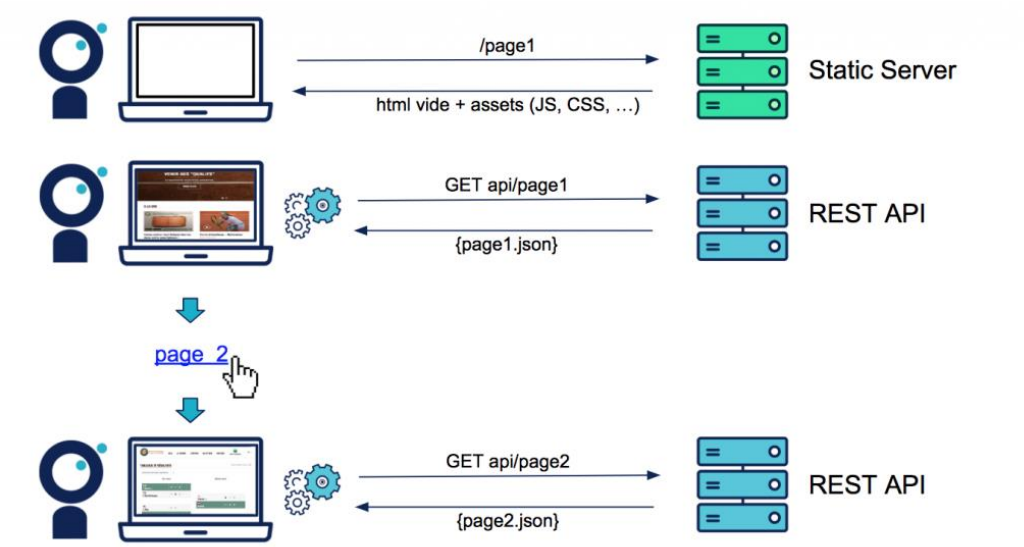
→ Utilisé par les frameworks AngularJS et Backbone.js (2010), Ember (2011), React (2013), Vue (2014) ou encore Angular (2016).

MULTIPAGES APPLICATION VS SINGLE PAGE APPLICATION

Single page application (SPA)

→ L'ensemble des éléments de l'application est chargé sur le navigateur. Tous les templates du site sont donc pré-chargés.

→ Navigation côté client uniquement (émulation d'une navigation).



H T M L

Langage de balises

Anatomie d’un document HTML

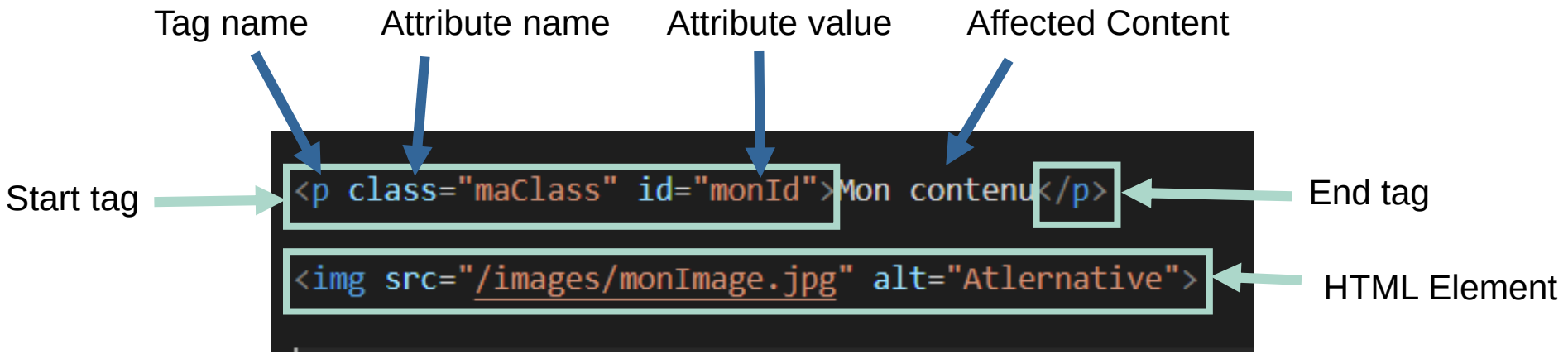
Rappel formulaire

Notion de template

HTML 5

→ **H**yper**T**ext **M**arkup **L**anguage (HTML) : code utilisé pour structurer une page web et son contenu.

→ **Langage de balises**



https://developer.mozilla.org/fr/docs/Web/HTML/Element

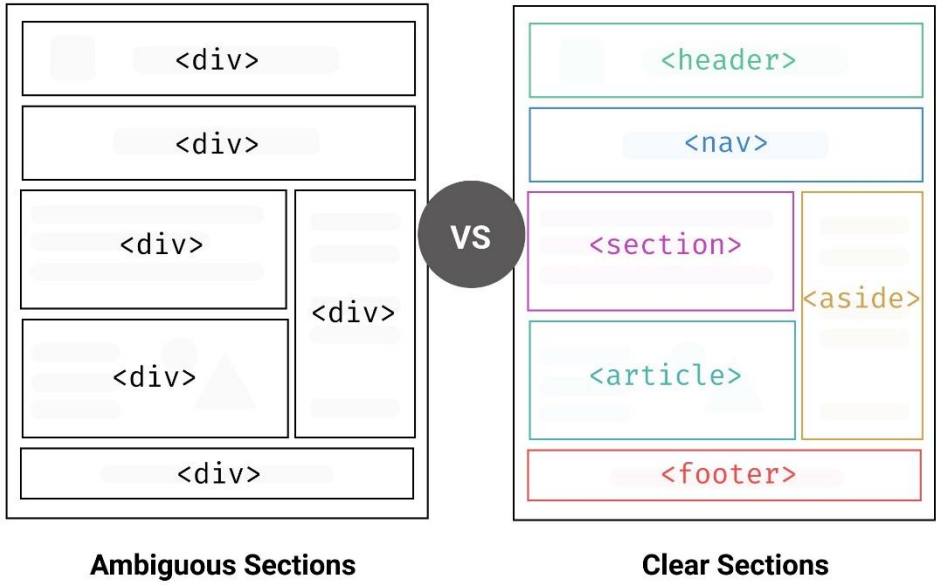
HTML5

→ Structure simple d'une page HTML5 :

```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <!-- Metadata -->
    <meta charset="utf-8">
  </head>
  <body>
    <!-- Contenu -->
  </body>
</html>
```

HTML5

→ Structure d’une page HTML : balises header, nav, section, article, aside et footer



HTML5

→ Formulaire : balise **<form>**


```
<fieldset>
    <legend>Fruit juice size</legend>
    <p> <input type="radio" name="size" id="size_1" value="small"> <label for="size_1">Small</label> </p>
    <p> <input type="radio" name="size" id="size_2" value="medium"> <label for="size_2">Medium</label> </p>
    <p> <input type="radio" name="size" id="size_3" value="large"> <label for="size_3">Large</label> </p>
</fieldset>
```

```
<label for="taste_1">J'aime les cerises</label>
<input type="checkbox" id="taste_1" name="taste_cherry" value="1">
```

```
<label for="username">Nom :<abbr title="required">*</abbr></label>
<input id="username" type="text" name="username" required>
```

```
<select id="card" name="usercard">
    <option value="visa">Visa</option>
    <option value="mc">Mastercard</option>
    <option value="amex">American Express</option>
</select>
```

```
<button type="submit">Valider</button>
```

https://developer.mozilla.org/fr/docs/Learn/Forms

HTML 5

Formulaire : validation des données côté HTML

Champ required

```
<label for="choose">Login ?</label>
<input id="choose" name="i_like" required>
```

Expression régulière

```
<label for="choose">Mot de passe ? </label>
<input id="choose" name="mdp" required pattern="[A-F][0-9]{5}">
```

Champ input

```
<label for="t2">Adresse électronique ?</label>
<input type="email" id="t2" name="email">
```

→ API de validation des contraintes disponible en JavaScript

H T M L 5

Formulaire : envoi du formulaire

→ A travers l’attribut action du formulaire.

→ En utilisant des requêtes AJAX (e.g. via un objet FormData avec XmlHttpRequest).

H T M L 5

Balise template :

→ Mécanisme utilisé pour stocké côté client du contenu HTML qui ne doit pas être affiché lors du chargement de la page.

→ Permet de stocker des modèles de code HTML pouvant être clonés et collées dans un document à l'aide de scripts JS (structure répétitive).


```
<template id="monParagraphe">
    <h2>JXC</h2>
</template>
```

```
let template = document.getElementById('monParagraphe');
let templateContent = template.content.cloneNode(true);
document.body.appendChild(templateContent);
```

DESIGN

CSS

Responsive design

Pré et Post-processeurs CSS (SASS, LESS)

Framework CSS (Bootstrap)

DESIGN

→ Le webdesign doit être réfléchi : UX et UI designer (e.g. expérience utilisateur à travers l'ergonomie et la navigation du site, réalisation de maquette pour définir la chartre graphique).

→ A éviter : un design complexe ou chargé, des polices d'écriture trop petites, des couleurs trop marquées, une navigation peu ergonomique, des pop-ups ou bannières trop agressives.

CSS

→ **C**ascading **S**tyle **S**heets : code utilisé pour mettre en forme une page web.

→ Insertion d'une feuille de style dans un fichier html :

```
<link rel="stylesheet" href="css/monCSS.css" />
```


→ Code CSS inline à éviter :

```
<h1 style="color: blue;background-color: yellow;border: 1px solid black;">Hello World!</h1>
```

→ Possibilité d'appliqué un style à tous les éléments du même type, plusieurs éléments, un élément particulier.

```
p {
  color: red;
  width: 500px;
  border: 1px solid black;
}
```

```
p,li,h1 {
  color: red;
}
```

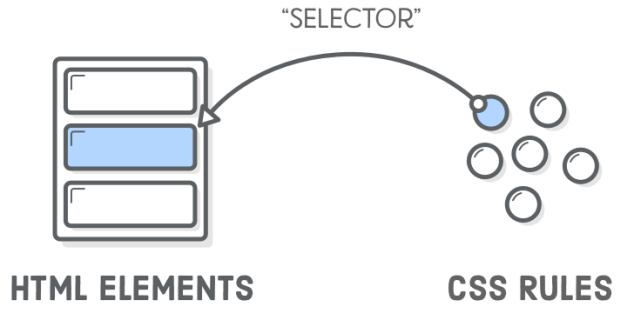
CSS : STRUCTURE

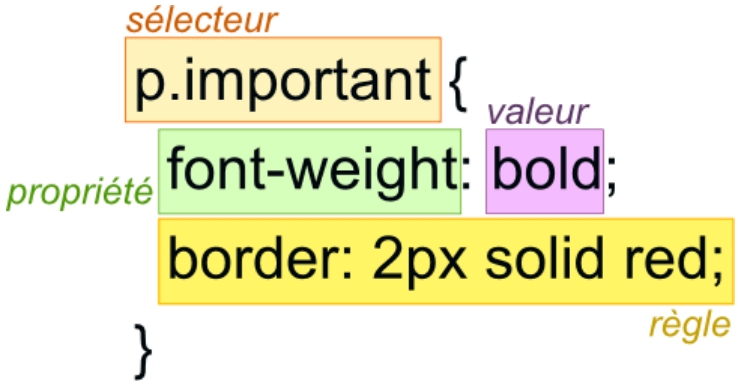
→ Sélecteur (élément HTML)

→ Déclaration (règle simple qui détermine les propriétés de l'élément mis en forme)

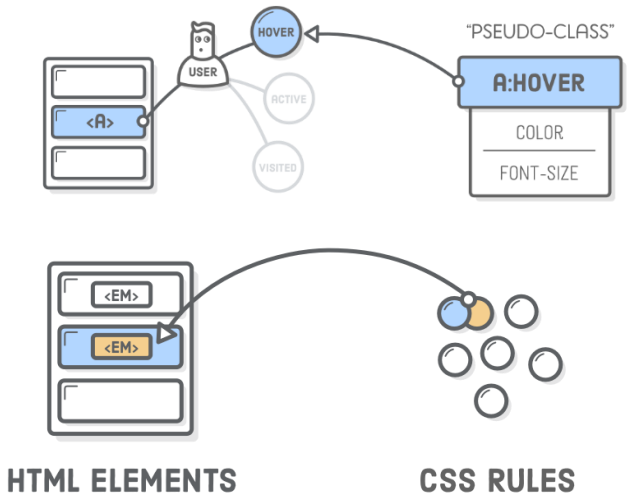
→ Propriétés

→ Valeur de la propriété





CSS : SÉLECTEUR




```
#maDiv {
  background-color: #15DEA5;
}
.button {
  background-color: #DB464B;
}
p .maClass {
  background-color: steelblue;
}
a:hover{
  color: cornflowerblue;
}
```

Selector	Example
Type selector	`h1 { }`
Universal selector	`* { }`
Class selector	`.box { }`
id selector	`#unique { }`
Attribute selector	`a[title] { }`
Pseudo-class selectors	`p:first-child { }`
Pseudo-element selectors	`p::first-line { }`
Descendant combinator	`article p`
Child combinator	`article > p`
Adjacent sibling combinator	`h1 + p`
General sibling combinator	`h1 ~ p`

RESPONSIVE DESIGN

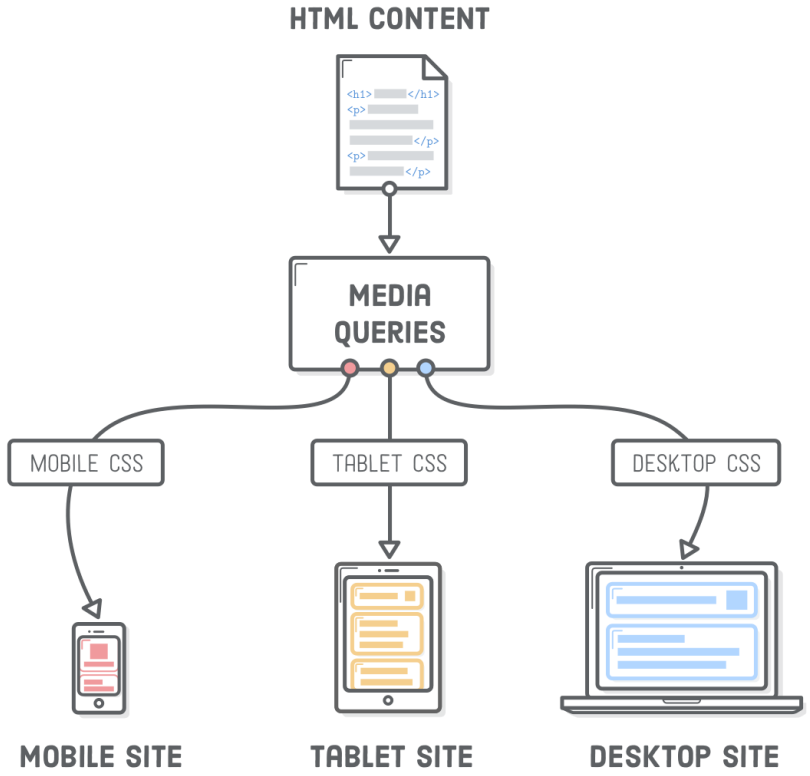
Comment adapter le design aux différents types de supports et résolution d'écran ?

→ Une solution : les **media queries**

Moyen d'appliquer conditionnellement des règles CSS.

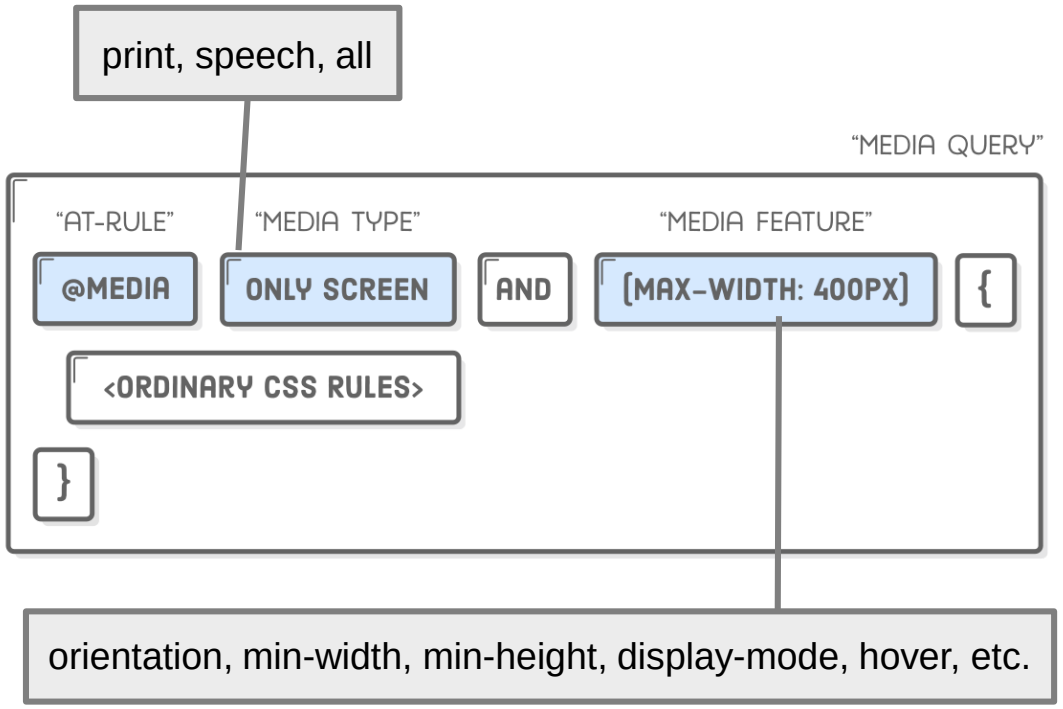
Utilisation des mêmes éléments HTML

→ Généralement débuter par la mise en page mobile.



RESPONSIVE DESIGN

→ **Media query**




```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

/* Mobile Styles */
@media only screen and (max-width: 400px) {
  body {
    background-color: #F09A9D; /* Red */
  }
}

/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px) {
  body {
    background-color: #F5CF8E; /* Yellow */
  }
}

/* Desktop Styles */
@media only screen and (min-width: 961px) {
  body {
    background-color: #B2D6FF; /* Blue */
  }
}
```

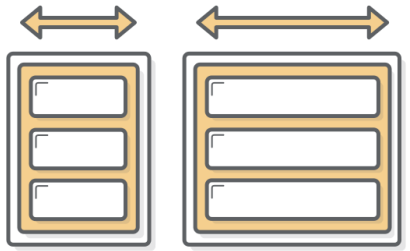
RESPONSIVE DESIGN

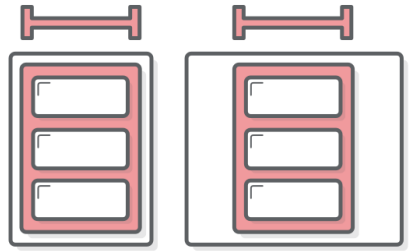
→ Zoom automatique sur les navigateurs mobiles pour ajuster la page dans la largeur.

→ Possibilité de le désactiver si le site est responsive :

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```


→ Attention aux unités de longueur pour un design adaptable (pour la taille du texte par exemple), au type de positionnement des éléments et leur conteneur (Grid, FlexBox, etc.)





DESIGN

CSS

Responsive design

Pré et Post-processeurs CSS (SASS, LESS)

Framework CSS (Bootstrap)

PRE-PROCESSEUR : SASS

→ SASS (**S**yntactically **A**wesome **S**tyle**s**heets) permet d'ajouter à CSS diverse fonctionnalités permettant d'organiser de manière plus maintenable les feuilles de style.

→ Commande pour compiler les fichiers sass en css

```
sass --watch input.scss output.css
```

Outils à disposition pour organiser et gérer les styles :

→ Variables

→ Imbrication des sélecteurs

→ Fonctions

→ Directives @import

PRE-PROCESSEUR : SASS

→ Hiérarchie entre les sélecteurs

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

```
nav {
    ul {
      margin: 0;
      padding: 0;
      list-style: none;
    }

    li { display: inline-block; }

    a {
      display: block;
      padding: 6px 12px;
      text-decoration: none;
    }
  }
```

PRE-PROCESSEUR : SASS

→ Variables

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

PRE-PROCESSEUR : SASS

→ Utilisation de modules

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}

.inverse {
  background-color: #333;
  color: white;
}
```

```scss
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
font: 100% $font-stack;
color: $primary-color;
}
```

```scss
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

PRE-PROCESSEUR : SASS

→ Utilisation des mixins

```
.info {
  background: DarkGray;
  box-shadow: 0 0 1px rgba(169, 169, 169, 0.25);
  color: #fff;
}

.alert {
  background: DarkRed;
  box-shadow: 0 0 1px rgba(139, 0, 0, 0.25);
  color: #fff;
}

.success {
  background: DarkGreen;
  box-shadow: 0 0 1px rgba(0, 100, 0, 0.25);
  color: #fff;
}
```

```
@mixin theme($theme: DarkGray) {
    background: $theme;
    box-shadow: 0 0 1px rgba($theme, .25);
    color: #fff;
  }

  .info {
    @include theme;
  }
  .alert {
    @include theme($theme: DarkRed);
  }
  .success {
    @include theme($theme: DarkGreen);
  }
```

LESS

→ LESS (**Le**aner **S**tyle **S**heets) est un préprocesseur CSS, influencé par SASS.

→ Permet également une plus grande facilité de gestion des feuilles de styles et supprime les répétitions de code.

→ Concept similaires (imbrication, variable, héritages, mixins, etc.)

→ Repose sur une syntaxe plus naturelle (ressemblant à CSS) par rapport à SASS qui utilise des mots clés (@use, @include, etc.).

BOOTSTRAP

→ Framework open source utilisant Sass, développé au début par une équipe de Twitter.

→ Compatible avec les différents navigateurs.

→ Prend en charge la conception réactive et propose des modèles de conception prédéfinis.

→ Pour inclure Bootstrap : CSS + JS (Bootstrap Bundle avec Popper)

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
```

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
```

→ Développé au début pour mobile (responsive design)

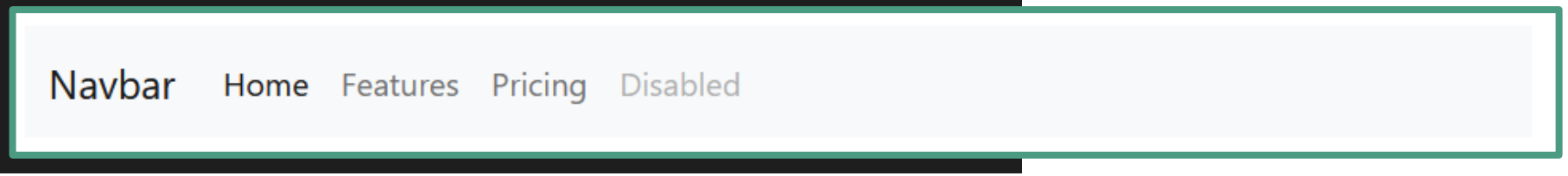
```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

BOOTSTRAP : EXEMPLE

```
<div class="btn-group" role="group" aria-label="Basic mixed styles example">
  <button type="button" class="btn btn-danger">Left</button>
  <button type="button" class="btn btn-warning">Middle</button>
  <button type="button" class="btn btn-success">Right</button>
</div>
```

BOOTSTRAP : EXEMPLE

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Features</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Pricing</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled">Disabled</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

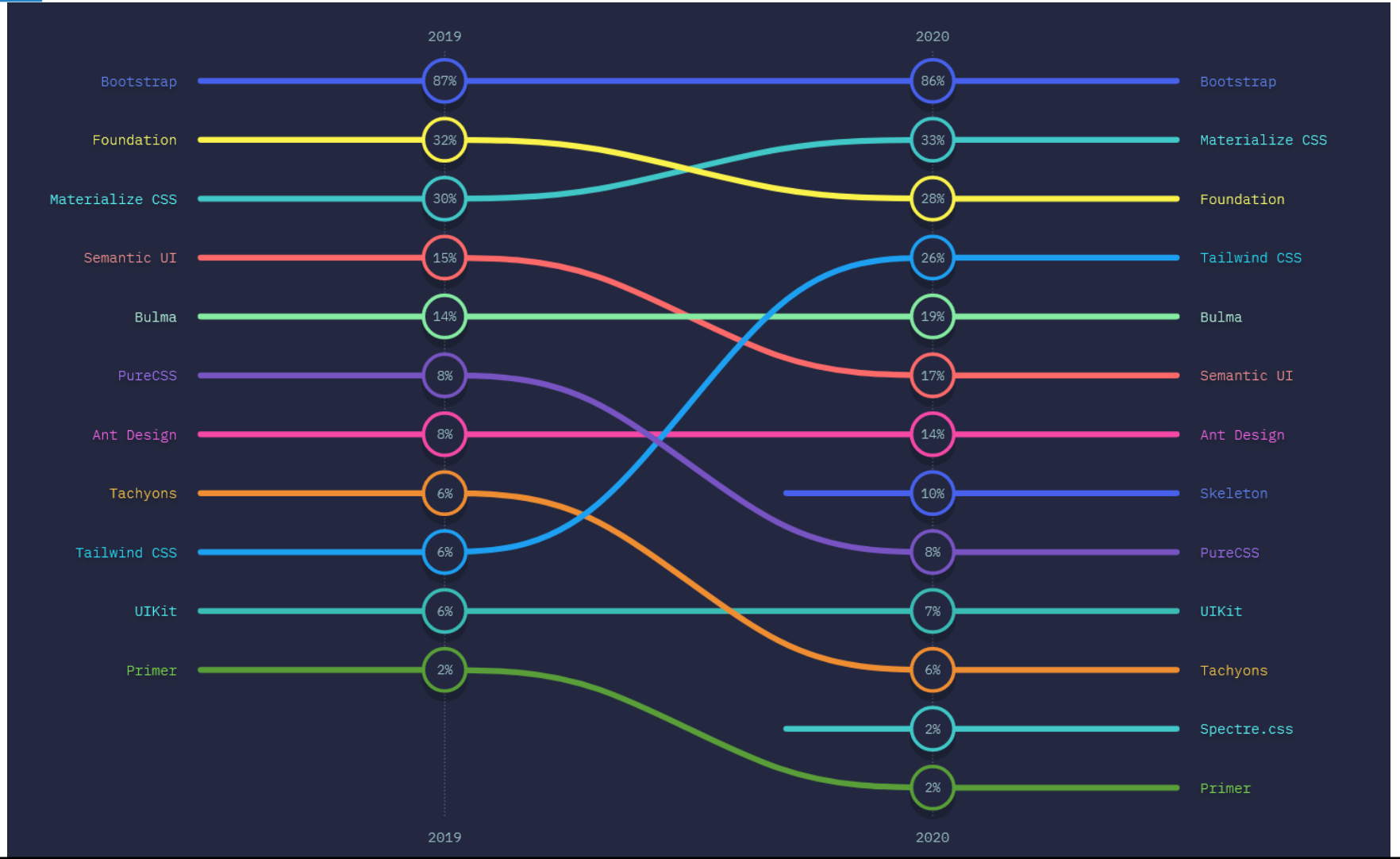


PRE-POSTPROCESSEUR CSS

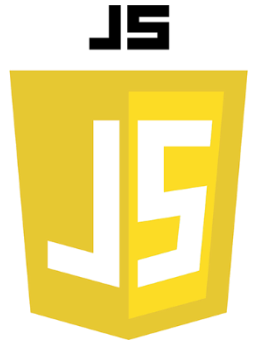
→ Ratio d’utilisation

FRAMEWORK CSS

→ Ratio d'utilisation



JAVASCRIPT



Définition

Syntaxe

Objets

Evènements

Accès à un éléments quelconque d’une page

JAVASCRIPT

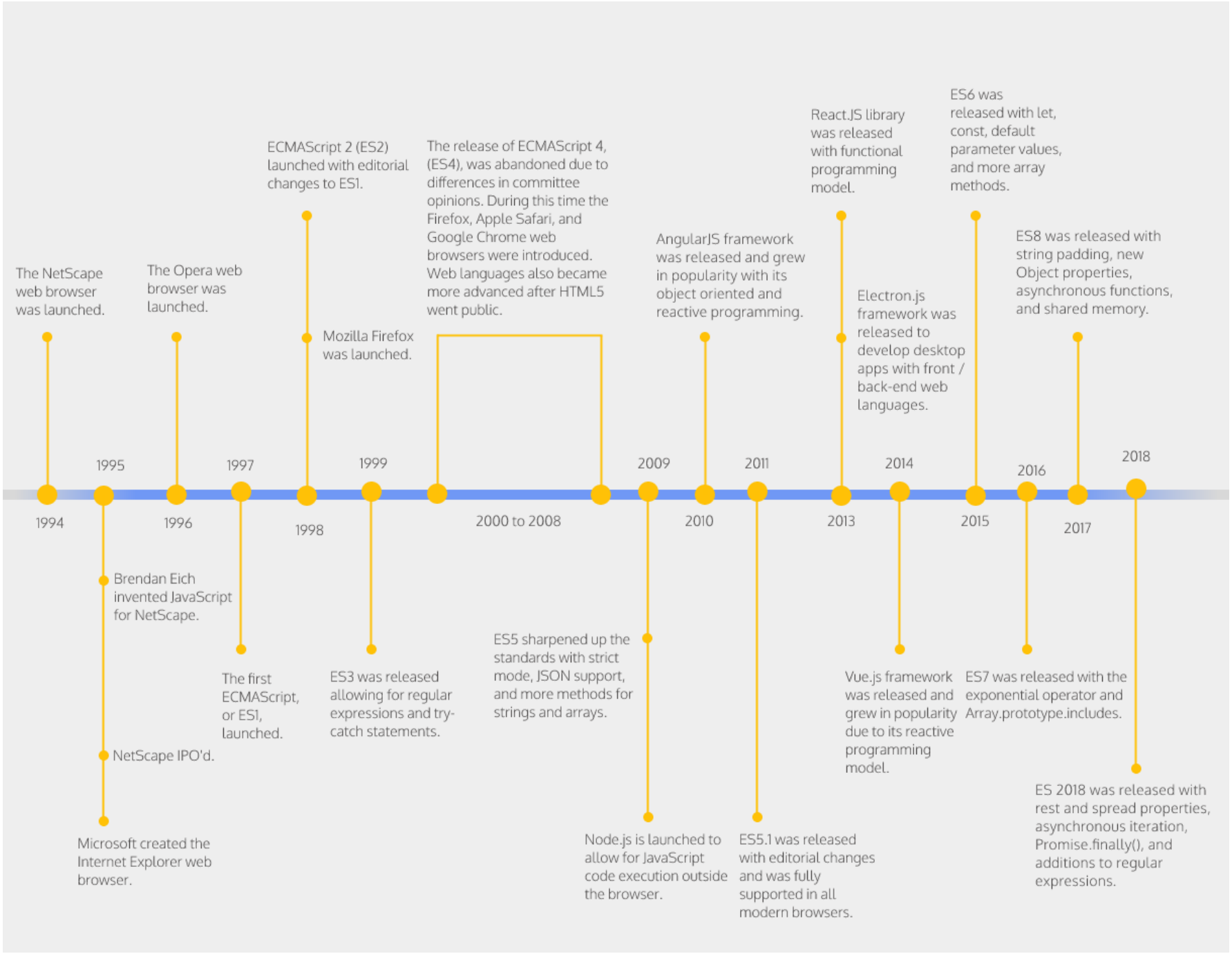
→ Développé par Netscape et Sun dans les années 95 (principal développeur : Brendan Eich, co-fondateur du projet Mozilla, de la Mozilla Foundation et Mozilla Corporation)

→ Précédentes versions : Mocha (en interne), LiveScript, puis JavaScript

→ Conçu à l'origine comme un langage de script complémentaire à Java

→ Proposé à la standardisation à ECMA (*European Computer Manufacturers Association*) en 1996 (ECMA-262) :

- **ES1** en 1997
- **ES6** en 2015
- **ES2021** en juin 2021


1994
The NetScape web browser was launched.
1995
Brendan Eich invented JavaScript for NetScape.
NetScape IPO'd.
Microsoft created the Internet Explorer web browser.
1996
The Opera web browser was launched.
1997
The first ECMAScript, or ES1, launched.
1998
ECMAScript 2 (ES2) launched with editorial changes to ES1.
Mozilla Firefox was launched.
1999
ES3 was released allowing for regular expressions and try-catch statements.
2000 to 2008
The release of ECMAScript 4, (ES4), was abandoned due to differences in committee opinions. During this time the Firefox, Apple Safari, and Google Chrome web browsers were introduced. Web languages also became more advanced after HTML5 went public.
2009
ES5 sharpened up the standards with strict mode, JSON support, and more methods for strings and arrays.
Node.js is launched to allow for JavaScript code execution outside the browser.
2010
AngularJS framework was released and grew in popularity with its object oriented and reactive programming.
2011
ES5.1 was released with editorial changes and was fully supported in all modern browsers.
2013
React.JS library was released with functional programming model.
2014
Electron.js framework was released to develop desktop apps with front / back-end web languages.
Vue.js framework was released and grew in popularity due to its reactive programming model.
2015
ES6 was released with let, const, default parameter values, and more array methods.
2016
ES7 was released with the exponential operator and Array.prototype.includes.
2017
ES8 was released with string padding, new Object properties, asynchronous functions, and shared memory.
2018
ES 2018 was released with rest and spread properties, asynchronous iteration, Promise.finally(), and additions to regular expressions.

JAVASCRIPT

Langage populaire :

→ Disponible dans tous les navigateurs (compatibilité)

→ La plupart des navigateurs modernes supporte le standard ES6 (98% à 100%)

→ Modules tiers disponibles grâce à NPM et GitHub : il existe des modules JavaScript pour quasiment chaque besoin.

Utilisation de JavaScript :

→ Historiquement exécuté sur le navigateur web

→ Node.js et NPM à la base du nouveau succès de JavaScript (serveur, local)

JAVASCRIPT

JavaScript permet :

→ De modifier l'apparence de la page

→ De communiquer avec le serveur

→ D'enregistrer les actions de l'utilisateur,

→ De réagir aux évènements utilisateur

→ De sauvegarder des données

JAVASCRIPT

→ Langage orienté **prototype** et non orienté objet :

- Déclaration d'un objet générique (modèle), puis héritage
- Notion de classe depuis ECMAScript 6

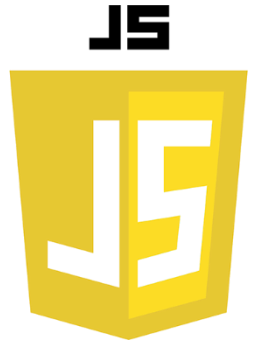
→ Dynamique :

- typage (faiblement typé),
- fonctions,

→ Evènementiel :

- paradigme de programmation, attente puis réaction aux actions utilisateur

JAVASCRIPT



Définition

Syntaxe

Objets

Evènements

Accès à un éléments quelconque d’une page

JAVASCRIPT : SYNTAXE

Intégration dans la page :

→ En **interne** :

```
<script>alert("Hello World");</script>
```

→ En **externe** :

```
<script src="js/fichierJS.js"></script>
```

→ Attention au **positionnement des balises** dans le fichier html.

Quand le navigateur rencontre un bloc JavaScript, il l'exécute dans l'ordre. Le code est **interprété**, le résultat du code exécuté est envoyé directement.

JAVASCRIPT : SYNTAXE

Depuis **ES6** : mot-clé **async** et **defer**

```
<script src="js/monScript.js" defer></script>
```

→ **async** : le navigateur continu de traiter l'HTML et le fichier JS est téléchargé en parallèle. Une fois chargé, le contenu est exécuté.

→ **defer** : diffère l'exécution du JavaScript jusqu'à ce que la page soit complètement chargée.

JAVASCRIPT : SYNTAXE

Bonne pratique :

→ fichier *.js* externe

→ lecture par évènement ***onload***,

→ *separation of concerns* (Dijkstra):

- pour le script principale

- puis utilisation de fonction

JAVASCRIPT : SYNTAXE

→ **Syntaxe et opérateur semblable à C / C# / Java**

Operateurs (+, *, !+, etc.)

Variables

Chaînes de caractères et Array

Structures conditionnelles

Structures itératives

Fonctions

JAVASCRIPT : SYNTAXE

Portées des variables

→ **var :** permet de déclarer une variable dont la portée est le **contexte d'exécution courant** (**la fonction** qui contient la déclaration ou **le contexte global** si la variable est déclarée en dehors de toute fonction).

→ **let :** permet de déclarer une variable dont la portée est **le bloc courant**. *let* crée une variable globale alors que *var* ajoute une propriété à l'objet global au niveau le plus haut.

→ **const :** variable accessible qu'en lecture.

JAVASCRIPT : SYNTAXE

```
function varTest() {
    var x = 31;
    if (true) {
      var x = 71;
      console.log(x);
    }
    console.log(x);
  }

function letTest() {
    let x = 31;
    if (true) {
        let x = 71;
        console.log(x);
    }
    console.log(x);
}
```

```
var x = 'global';
let y = 'global2';
console.log(this.x);
console.log(this.y);
console.log(y);
```

```
var a = 5;
var b = 10;

if (a === 5) {
  let a = 4;
  var b = 1;

  console.log(a);
  console.log(b);
}

console.log(a);
console.log(b);
```

JAVASCRIPT : SYNTAXE

Types de données :

→ type **booléen** (true et false)

→ Type **nul** (null)

→ Type **indéfinie** (undefined)

```
let x;          //undefined
x = 5;          //number
x = "Toto";     //string
let myArray = [1, 3.3, "toto"];
let myHashTable = {a: 5, c:9.5, c:"toto"};
```

→ Type pour les **nombres** entiers ou réels (number)

→ Type pour les **chaînes de caractères** (string)

→ Type pour les **symboles** (depuis ES6 : type pour des données immuables et uniques)

→ Type pour les **objets** (Object, avec par exemple Array)

JAVASCRIPT : SYNTAXE

Objet :

→ Chaque variable est considéré comme un objet.

→ Méthodes existantes pour les objets (string et array par exemple)

String :

```
let x = "toto tata titi tutu tyty tete";
console.log(x[3]);
console.log(x.charAt(8));
console.log("test".substring(1, 3));
console.log("test".toUpperCase());
```

```
let y = 16 + 4 + "Volvo";
let z = "Volvo" + 16 + 4;
```

```
parseInt("234");
parseInt("2.99abc");
```

```
console.log("toto"=="tata");
```

JAVASCRIPT : SYNTAXE

Liste

Pour appliquer un traitement à chaque élément d'un liste :

→ *forEach(<fonction de callback>)*

→ *map(<fonction de callback>)*

```
let arr = [1, 2, 3, 4, 5];
console.log(arr.length);
arr.push(6);
arr.splice(3, 0, 7);
console.log(arr.indexOf(7));
arr.sort();
arr.forEach(element => {
    console.log(element);
});
console.log(arr.map(element => element*2 ));
```

JAVASCRIPT : SYNTAXE

Structures conditionnelles

→ structure **if … else ….**

```
var a = 0;
var b = true;
if (typeof(a)=="undefined" || typeof(b)=="undefined") {
  document.write("Variable a or b is undefined.");
}
else if (!a && b) {
  document.write("a==0; b==true;");
} else {
  document.write("a==" + a + "; b==" + b + ";");
}
```

```
if (x > 50){
    // faire quelque chose
} else if (x > 5) {
    // faire autre chose
} else {
    // faire encore autre chose
}

/*
Greater than: >
Less than: <
Greater than or equal to: >=
Less than or equal to: <=
Equal: ==
Not equal: !=
*/
```

JAVASCRIPT : SYNTAXE

Structures conditionnelles

→ structure **switch … case …**

```
switch (variable) {
    case 1:
      // do something
      break;
    case 'a':
      // do something else
      break;
    case 3.14:
      // another code
      break;
    default:
      // something completely different
}
```

```
var toto = 1;
var output = 'Résultat : ';
switch (toto) {
    case 0:
        output += 'Donc ';
    case 1:
        output += 'quel ';
        output += 'est ';
    case 2:
        output += 'votre ';
    case 3:
        output += 'nom ';
    case 4:
        output += '?';
        console.log(output);
        break;
    case 5:
        output += '!';
        console.log(output);
        break;
    default:
        console.log('Veuillez choisir un nombre entre 0 et 5 !');
}
```

JAVASCRIPT : SYNTAXE

Structures itératives

→ boucles classiques

```
for(let counter=0 ; counter < 5 ; counter++){
    console.log(counter);
}
var i=0;
while(i<5){
    console.log(++i);
}
do{
    --i;
} while(i>0)
console.log(i);
```

JAVASCRIPT : SYNTAXE

Structures itératives

→ boucles **for … in …** et **for … of …**

```
let phones = {"type":"phone", "brand":"tomato", "name":"tomatoPhone"};
for(let key in phones){
    console.log(key);
    console.log(phones[key]);
}
let brandPhone = ["tomato", "pear"];
for(let i in brandPhone){
    console.log(i);
}
for(let elem of brandPhone){
    console.log(elem);
}
```

```
type
phone
brand
tomato
name
tomatoPhone
0
1
tomato
pear
```

JAVASCRIPT : SYNTAXE

Fonctions

Définition d'une fonction :

```
function nomFonction(arg1, arg2)
```


→ Contrairement au langage C, on ne donne pas le type des arguments ni celui de la valeur de retour éventuelle.

```
function average(a, b, c) {
    return ( a + b + c ) /3;
}
```

→ Possibilité d'écrire une fonction **anonyme :**

```
var result = function() { /* instructions */ }
```

JAVASCRIPT : SYNTAXE

Il n'est pas obligatoire :

→ de définir une valeur de retour

→ de spécifier tous les arguments lors de l'appel d'une fonction

Les fonctions ont accès à tous les paramètres d'entrée via le tableau **arguments** :

```
function sum() {
    var sum = 0;
    for(let i = 0 ; i<arguments.length ; i++)
        sum += parseInt(arguments[i]);
    return sum;
}
alert(sum(1, 2, 4));
```

JAVASCRIPT : SYNTAXE

→ **Imbrication et fermeture de fonction**

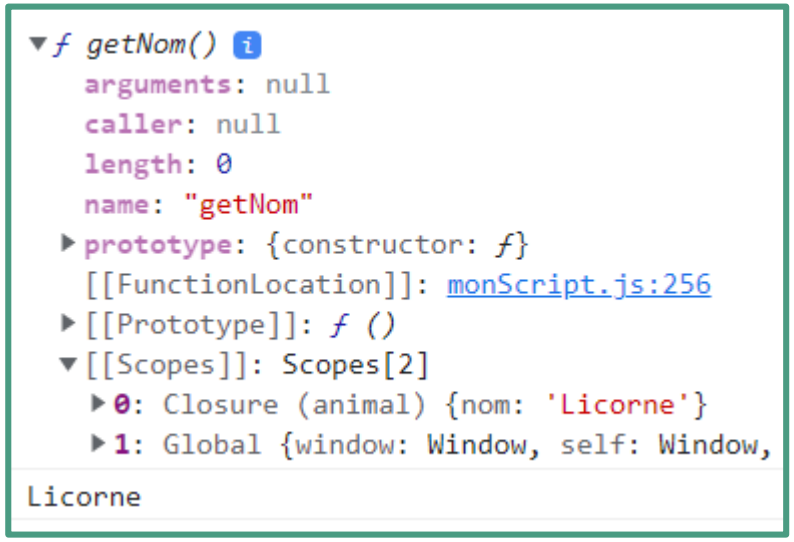
La fonction interne à accès aux variables et paramètres de la fonction parente (mais pas l'inverse).

Création d'une fermeture lorsque la fonction interne est disponible en dehors de la fonction parente.

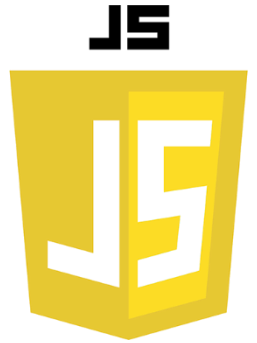
Fermeture de fonction

Renvoie la fonction interne pour la rendre disponible en dehors de la portée de la fonction parente.

```
var animal = function(nom) {
    var getNom = function () {
      return nom;
    }
    return getNom;
}

monAnimal = animal("Licorne");
console.dir(monAnimal);
console.log(monAnimal());
```



JAVASCRIPT



Définition

Syntaxe

Objets

Evènements

Accès à un éléments quelconque d’une page

JAVASCRIPT : OBJET

→ Notion d'objet est important en JavaScript : quasiment **tout est objet**.

Avant la normalisation ES6, il n'existait pas de notion de **class**.

Il est possible d'interagir à deux niveaux :

→ Au niveau du navigateur internet

→ Au niveau de la page affichée dans le navigateur

Tous les éléments HTML du DOM peuvent être manipulés en tant qu'objet.

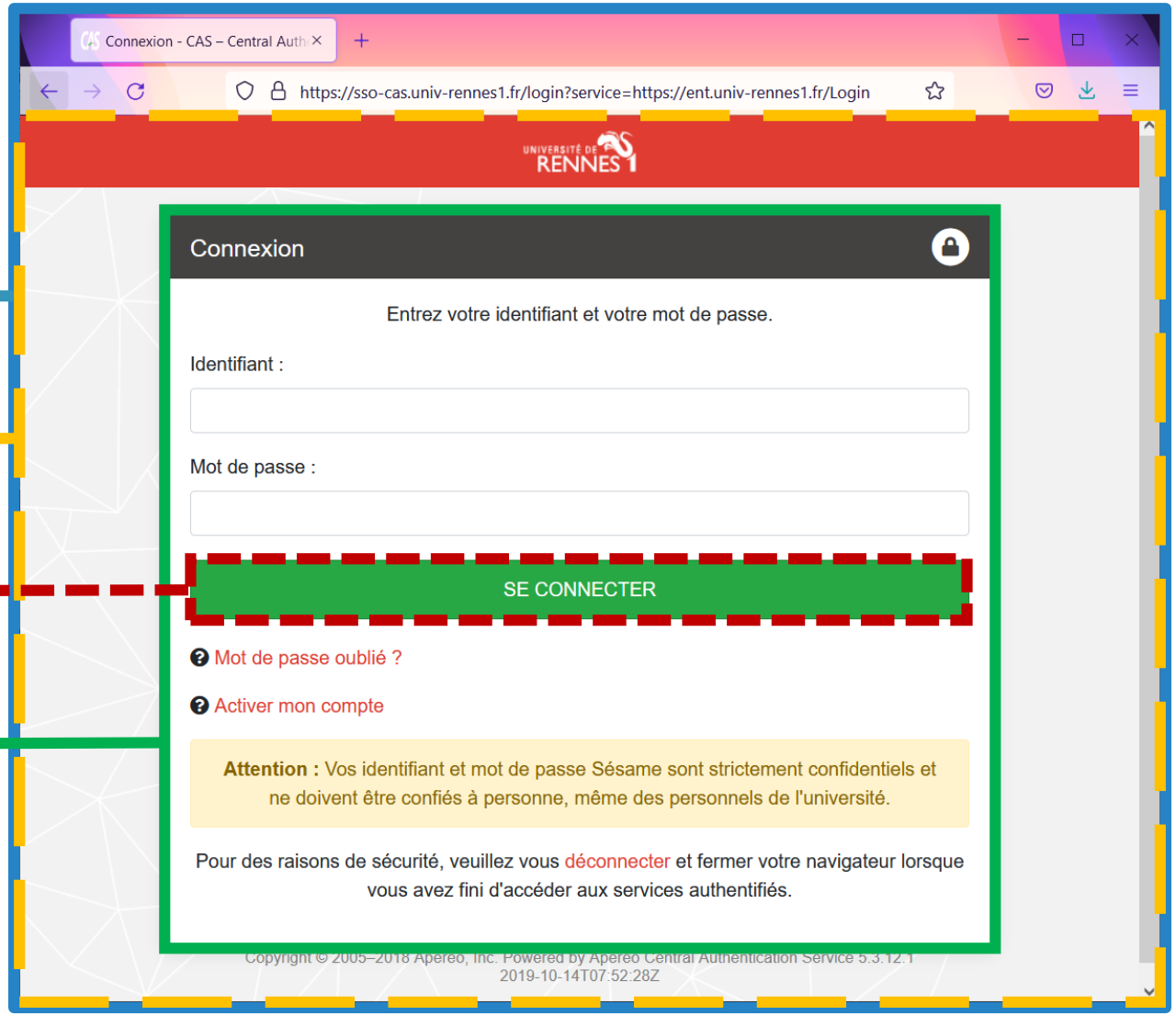
JAVASCRIPT : OBJET

Objet fenêtre

Objet document

Objet bouton

Objet formulaire



JAVASCRIPT : OBJET

→ L'accès se fait de façon hiérarchique.

```
window.document.forms["nomFormulaire"].nomElement
```

→ Par chaque objet il existe des méthodes et des attributs.

```
<form name="monForm">
    <label for="login">Votre login :</label>
    <input type="text" name="login" id="login" />
</form>
```

→ Par exemple, pour obtenir la valeur du champ login du formulaire :

```
let loginUser = window.document.forms["monForm"].login.value;
```

JAVASCRIPT : OBJET

Possibilité de créer ses propres objets

→ **Objets littéraux (JSON)**

```
let myPhone = { "type": "phone", "brand": "Tomato", "name": "myPhone"}
```


Une propriété d'un objet peut avoir n'importe quelle valeur :

→ Une valeur booléenne,

→ Une valeur scalaire,

→ Une liste,

→ Un objet,

→ Un code implémentant une fonction

JAVASCRIPT : OBJET

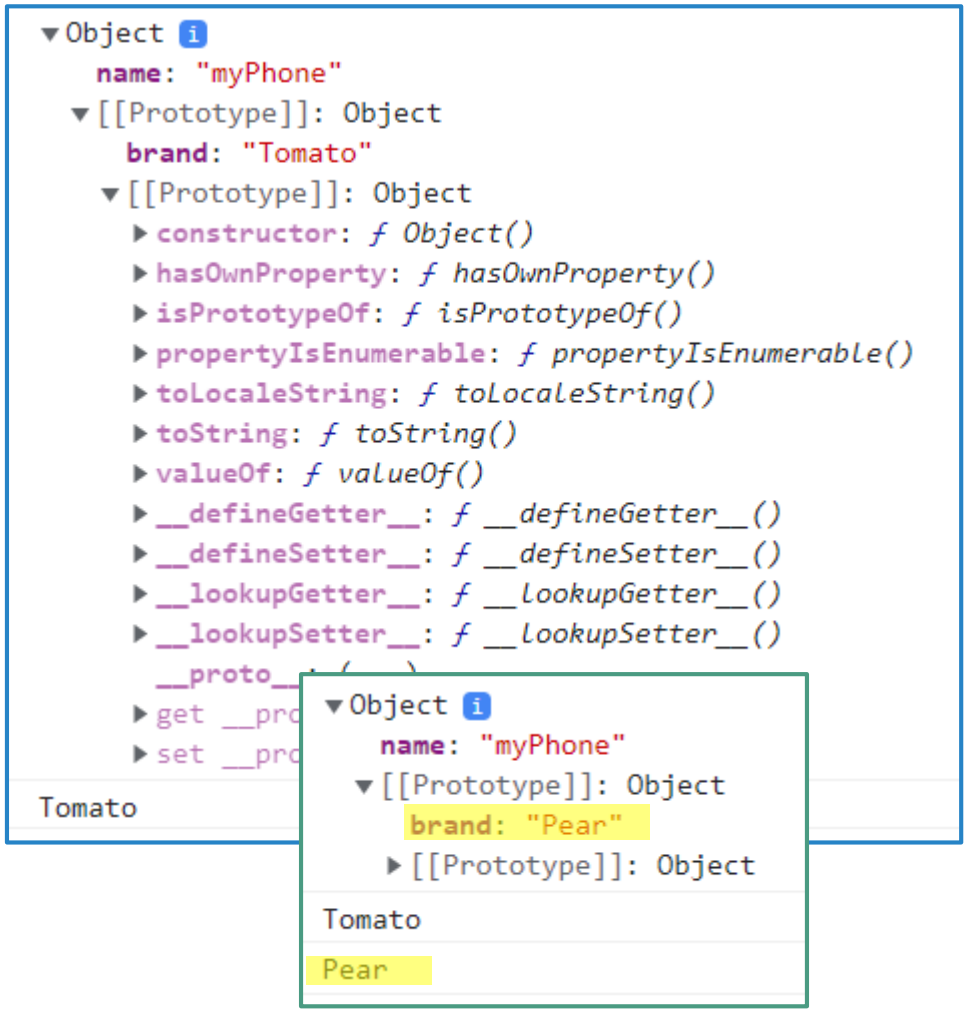
→ Héritage par chaînage de **prototype**

Création d'un nouveau objet héritant des propriétés d'un autre objet grâce à la **propriété __proto__** :

Pointeur sur objet hérité (*tomatoPhone*)

```
let tomatoPhone = {brand: "Tomato"};

let myPhone = {
    name: "myPhone",
    __proto__ : tomatoPhone
};

console.dir(myPhone);
console.log(myPhone.brand);
```

```
tomatoPhone.brand = "Pear";
console.log(myPhone.brand);
```



JAVASCRIPT : OBJET

Attribut prototype :

→ La valeur de l'attribut **prototype** est un objet rassemblant les attributs et les méthodes que l'on souhaite appliquer aux objets tout au long de la chaine de prototypage.

__proto__ ≠ **prototype**

JAVASCRIPT : OBJET

→ Création d'un objet par une **fonction constructrice**

```
function Phone(name, brand) {
    this.name = name;
    this.brand = brand;
}
Phone.prototype.whoAmI = function() {
    console.log("Je suis le " + this.name + "de marque " + this.brand);
}
var tomatoPhone = new Phone("tomatoPhone", "Tomato");
var myPhone = new Phone("myPhone", "Pear");
tomatoPhone.whoAmI();
myPhone.whoAmI();
```

```
Je suis le tomatoPhone de marque Tomato
Je suis le myPhone de marque Pear
```

```
console.dir(Phone)
▼ ƒ Phone(name, brand) i
    arguments: null
    caller: null
    length: 2
    name: "Phone"
  ▼ prototype:
    ▶ whoAmI: ƒ ()
    ▶ constructor: ƒ Phone(name, brand)
    ▶ [[Prototype]]: Object
      [[FunctionLocation]]: monScript.js:1
  ▶ [[Prototype]]: ƒ ()
```

```
console.dir(tomatoPhone)
▼ Phone i
    brand: "Tomato"
    name: "tomatoPhone"
  ▼ [[Prototype]]: Object
    ▶ whoAmI: ƒ ()
    ▶ constructor: ƒ Phone(name, brand)
    ▶ [[Prototype]]: Object
```

JAVASCRIPT : OBJET

→ L'héritage est implémenté par le **chaînage de prototypes**.

```
function Phone(name) {
    this.name = name;
}
Phone.prototype.whoAmI = function() {
    console.log("Je suis le " + this.name + " de marque " + this.brand);
}

function TomatoPhone(name) {
    Phone.call(this, name);
    this.brand = "Tomato";
}
TomatoPhone.prototype = Phone.prototype;

var tomatoPhone = new TomatoPhone("tomatoPhone");
tomatoPhone.whoAmI();
```

```
Je suis le tomatoPhone de marque Tomato
```

JAVASCRIPT : OBJET

→ Création de **classe** (ES6)

```
class Phone {
    constructor(name, brand){
        this.name = name;
        this.brand = brand;
    }
    whoAmI() {
        console.log("Je suis le " + this.name + " de marque " + this.brand);
    }
}
let tomatoPhone = new Phone("tomatoPhone", "Tomato");
tomatoPhone.whoAmI();
```

```
Je suis le tomatoPhone de marque Tomato
```

JAVASCRIPT : OBJET

→ **Héritage** entre classe (ES6)

Utilisation du mot clé **extends**

Appel du constructeur de la classe mère avec **super()**

```
Je suis le tomatoPhone de marque Tomato
Je suis le super pearPhone de marque Pear
```

```
class Phone {
    constructor(name, brand){
        this.name = name;
        this.brand = brand;
    }
    whoAmI() {
        console.log("Je suis le " + this.name + " de marque " + this.brand);
    }
}
class SuperPhone extends Phone{
    constructor(name, brand){
        super(name, brand);
    }
    whoAmI() {
        console.log(`Je suis le super ${this.name} de marque ${this.brand}`);
    }
}
let tomatoPhone = new Phone("tomatoPhone", "Tomato");
tomatoPhone.whoAmI();
let smartPhone = new SuperPhone("pearPhone", "Pear");
smartPhone.whoAmI();
```

JAVASCRIPT : OBJET

→ Attention à la compatibilité des navigateurs avec les fonctionnalités récentes proposé par ECMASript.

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Deno	Node.js
`classes`	49 ▼	13	45	No	36 ▼	9	49 ▼	49 ▼	45	36 ▼	9	5.0 ▼	1.0	6.0.0 ▼
`constructor`	49 ▼	13	45	No	36 ▼	9	49 ▼	49 ▼	45	36 ▼	9	5.0 ▼	1.0	6.0.0 ▼
`extends`	49 ▼	13	45	No	36 ▼	9	49 ▼	49 ▼	45	36 ▼	9	5.0 ▼	1.0	6.0.0 ▼
Private class fields	74	79	90	No	62	14.1	74	74	90	53	14.5	11.0	1.0	12.0.0
Private class fields 'in'	91	91	90	No	77	No	91	91	90	64	No	16.0	1.9	No
Private class methods	84	84	90	No	70	15	84	84	90	60	15	14.0	1.0	14.0.0
Public class fields	72	79	69	No	60	14.1 ▼	72	72	79	51	14.5 ▼	11.0	1.0	12.0.0
`static`	49 ▼	13	45	No	36 ▼	14.1	49 ▼	49 ▼	45	36 ▼	14.5	5.0 ▼	1.0	6.0.0 ▼
Static class fields	72	79	75	No	60	14.1	72	72	79	51	14.5	11.0	1.0	12.0.0
Class static initialization blocks	94	94	93	No	80	No	94	94	93	No	No	No	1.14	No

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes

JAVASCRIPT : OBJET

Mot clé this

Il se comporte légèrement différemment des autres langage de programmation.

→ **Dans le contexte global** : ***this*** fait référence à l'objet global.


```
console.log(this === window); // true

this.a = 37;
console.log(window.a); // 37

this.b = "MDN";
console.log(window.b); // "MDN"
console.log(b);        // "MDN"
```

JAVASCRIPT : OBJET

Mot clé this

Il se comporte légèrement différemment des autres langage de programmation.

→ **Dans le contexte d'une fonction** : la valeur de ***this*** dépend de la façon dont la fonction est appelée.

Dans un appel basic à une fonction : le ***this*** correspond à l'objet global.

```
function f1(){
    return this;
}
console.log(f1() === window); // true (objet global)
```

JAVASCRIPT : OBJET

Mot clé this

Quand une fonction est appelée comme **méthode d'un objet**, ***this*** correspond à l'objet possédant la méthode qu'on appelle.

```
var o = {
    prop: 37,
    f: function() {
        return this.prop;
    }
};

console.log(o.f()); // 37
```

Quand une fonction est utilisée comme un **constructeur** (avec le mot clef new), ***this*** sera lié au nouvel objet.

```
function Phone(){
    this.brand = "tomato";
}
var myPhone = new Phone();
console.log(myPhone.brand); // tomato
```

JAVASCRIPT : OBJET

Mot clé this

En utilisant les **fonctions fléchées**, ***this*** correspond à la valeur de this utilisé dans le contexte englobant.


```
var objet = {
    i: 10,
    b: () => console.log(this.i, this),
    c: function() {
      console.log(this.i, this);
    }
  }

  objet.b();
  // affiche undefined, Window (ou l'objet global de l'environnement)

  objet.c();
  // affiche 10, Object {...}
```

JAVASCRIPT : OBJET

Mot clé this

Pour passer ***this*** d'un **contexte à un autre**, les fonctions suivantes peuvent être utilisées : **call()**, **apply()** et **bind()**.


```
function fonction(name) {this.name = name;}
fonction.prototype.methode = function(callback) {
    console.log(this);  //Affiche : Object {name : "object1"}
    callback();
};
var objet = new fonction('objet1');
objet.methode(function(){
    console.log(this);  //Affiche : window
});
```

JAVASCRIPT : OBJET

Mot clé this

Méthode call() :

Réalise un appel à une fonction avec une valeur ***this*** donnée et des possibles arguments.


```
fonction.prototype.methode2 = function(callback) {
    console.log(this);
    callback.call(this, "un param");
};
var objet2 = new fonction('objet2');
objet2.methode2(function(param) {
    console.log("callback call "+ param + " et " + this.name);
});
```

JAVASCRIPT : OBJET

Mot clé this

Méthode apply() :

Appelle une fonction en lui passant une valeur ***this*** et des arguments sous forme d'un tableau ou d'une liste.


```
fonction.prototype.methode3 = function(callback) {
    console.log(this);
    callback.apply(this, ["un autre param"]);
};
var objet3 = new fonction('objet3');
objet3.methode3(function(param) {
    console.log("callback apply" + param + " et " + this.name);
});
```

JAVASCRIPT : OBJET

Mot clé this

Méthode bind() :

Création d'une nouvelle fonction qui possède le même corps et la même portée mais où le **this** sera lié au premier argument passé à **bind** (de façon permanente).

```
fonction.prototype.methode4 = function(callback) {
    console.log(this);
    let newFonction = callback.bind(this, "mon dernier param");
    newFonction();
};
var objet4 = new fonction('objet4');
objet4.methode4(function(param) {
    console.log("callback bind " + param + " et " + this.name);
});
```

JAVASCRIPT : OBJET

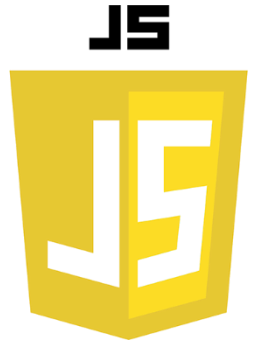
Mot clé this

Méthode bind() :

Création d'une nouvelle fonction qui possède le même corps et la même portée mais où le **this** sera lié au premier argument passé à **bind** (de façon permanente).

```
this.x = 9; // en dehors de tout contexte,
            // pour un navigateur, this est
            // l'objet window
var module = {
  x: 81,
  getX: function() { return this.x; }
};

module.getX();

var getX = module.getX;
getX();



var boundGetX = getX.bind(module);
boundGetX();
```

JAVASCRIPT



Définition

Syntaxe

Objets

Evènements

Accès à un éléments quelconque d’une page

JAVASCRIPT : EVENEMENT

L’action sur un élément de la page HTML se fait lors d’un évènement particulier :

→ clic sur un bouton,

→ champ input d’un formulaire qui change,

→ Redimensionnement de la fenêtre,

→ Formulaire en cours de soumission,

→ fin de chargement de la page HTML, etc.

Les **gestionnaires d’évènements** peuvent être utilisés pour gérer et vérifier les entrées utilisateur, les actions utilisateur et les actions du navigateur.

JAVASCRIPT : EVENEMENT

Exemple d'évènement possible pour une page Web :

→ click (onClick)

→ load (onLoad)

→ unload (onUnload)

→ mouseOver (onMouseOver)

→ mouseOut (onMouseOut)

→ focus (onFocus)

→ change (onChange)

→ submit (onSubmit)

https://developer.mozilla.org/fr/docs/Web/Events

JAVASCRIPT : EVENEMENT

→ Exemple d'utilisation de **onChange** avec un champ **input** d'un formulaire :

Fichier HTML

```
<body>
  <form name="monForm">
      <label for="login">Votre login :</label>
      <input type="text" name="login" id="login" onchange="afficherLogin();" />
      <input type="text" name="loginBis" id="loginBis" />
  </form>
</body>
```

Fichier .js

```
function afficherLogin(){
    window.document.forms["monForm"].loginBis.value =
        window.document.forms["monForm"].login.value;
}
```

Résultat

JAVASCRIPT : EVENEMENT

→ Exemple d'utilisation de **onChange** avec un champ **input** d'un formulaire :

Fichier HTML

```
<body>
  <form name="monForm">
      <label for="login">Votre login :</label>
      <input type="text" name="login" id="login" />
      <input type="text" name="loginBis" id="loginBis" />
  </form>
</body>
```

Fichier .js

```
var loginEvent = document.forms["monForm"].login.onchange = function() {
    window.document.forms["monForm"].loginBis.value =
        window.document.forms["monForm"].login.value;
}
```

Résultat

Méthode à privilégier

JAVASCRIPT : EVENEMENT

→ Utilisation de la méthode **addEventListener()** :

Enregistre un écouteur d'évènement sur un élément DOM.

Permet également d'enregistrer plusieurs gestionnaires pour le même écouteur.

Possibilité de supprimer un écouteur ajouté précédemment (*removeEventListener()*).

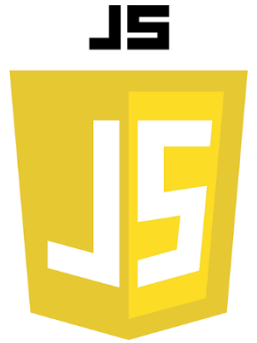

```html
<body>
    <form name="monForm">
        <label for="login">Votre login :</label>
        <input type="text" name="login" id="login" />
        <input type="text" name="loginBis" id="loginBis" />
    </form>
</body>
```

Fichier HTML

```js
var inputLogin = document.forms["monForm"].login;
inputLogin.addEventListener('change', function(){
    window.document.forms["monForm"].loginBis.value =
        window.document.forms["monForm"].login.value;
});
```

Fichier .js

JAVASCRIPT



Définition

Syntaxe

Objets

Evènements

Accès à un éléments quelconque d’une page

JAVASCRIPT : ACCES ELEMENT

La manière la plus simple pour avoir **accès à un élément d'une page** est d'utiliser son **identifiant** (Id).

→ Méthode **getElementById('id de l'élément')**


```
var inputLogin = document.forms["monForm"].login;
var inputLogin = document.getElementById("login");
```

Il est possible ensuite d'avoir accès à des informations et d'agir sur l'élément.

→ **innerHTML** : récupération/modification du contenu HTML de l'élément.

→ **textContent** : récupération/modification du contenu de l'élément.

→ **nodeName** : récupération du nom de l'élément.

JAVASCRIPT : EXEMPLE

Fichier HTML

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
    <link rel="stylesheet" href="css/monCSS.css" />
    <script src="js/mesFonctions.js" defer></script>
  </head>
  <body>
    <div id="maDiv"></div>
    <button type="button" id="btnvert">Vert</button>
    <button type="button" id="btnrouge">Rouge</button>
  </body>
</html>
```

JAVASCRIPT : EXEMPLE

Fichier CSS

```css
#maDiv{
    background-color: #635ead;
    width:200px;
    height:200px;
}
```

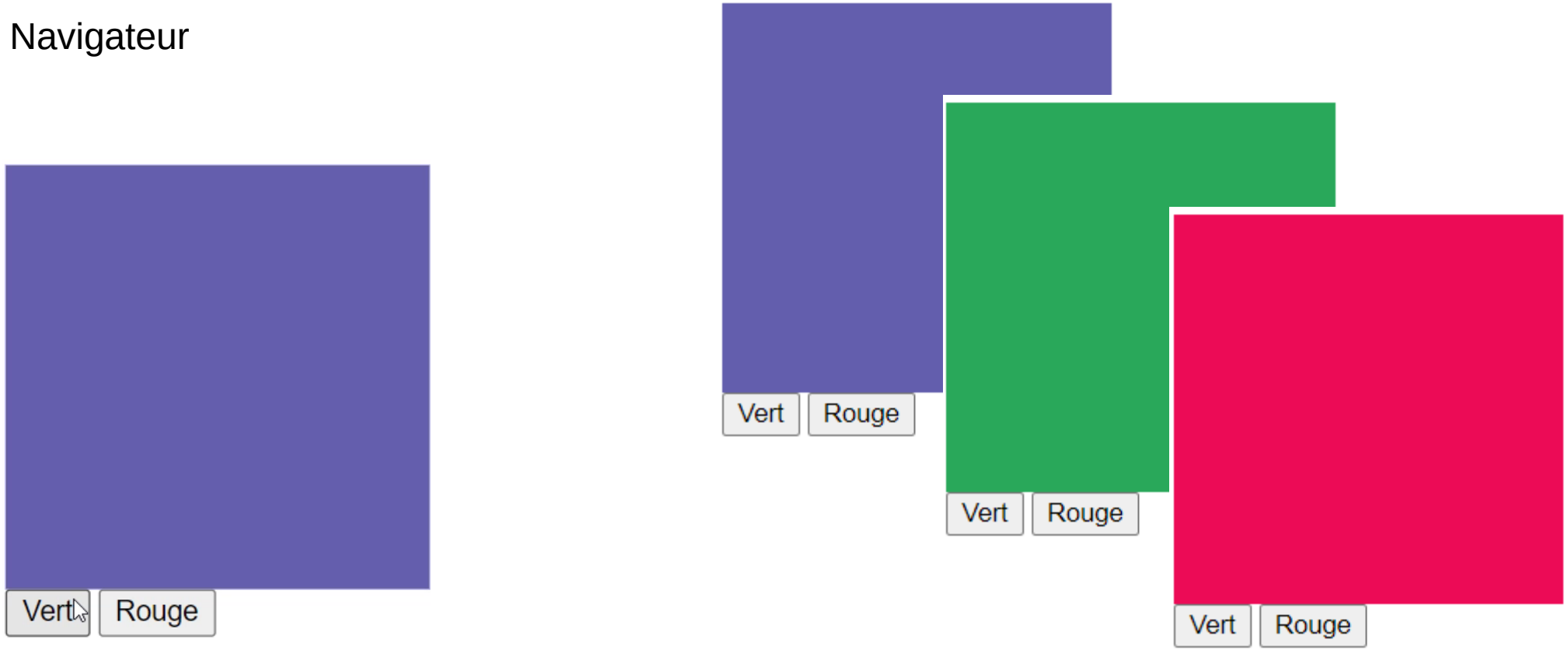
JAVASCRIPT : EXEMPLE

Fichier JS

```
function vert(){
    document.getElementById('maDiv').style.backgroundColor='#29a85a';
}
function rouge(){
    document.getElementById('maDiv').style.backgroundColor='#ec0b56';
}

document.getElementById('btnvert').addEventListener('click', vert);
document.getElementById('btnrouge').addEventListener('click', rouge);
```

JAVASCRIPT : EXEMPLE

Navigateur



JAVASCRIPT : ECMASCRIPT

→ **Lien vers les nouvelles spécifications ECMAScript :**

https://www.ecma-international.org/technical-committees/tc39/?tab=published-standards

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

→ **Table de compatibilité ECMAScript :**

http://kangax.github.io/compat-table/es6/

http://kangax.github.io/compat-table/es2016plus/

JAVASCRIPT : ECMASCRIPT

Exemple de nouvelles fonctionnalités a travers les nouvelles spécifications de ECMAScript

→ **ES6 :** Déclaration de variable avec let et const, la structure for … of, gabarit de chaîne de caractères, valeur par défaut des paramtres, le paramètre de reste, la décomposition, les symboles, les classes, fonction fléchée, etc. (et bien d'autres !)

→ **ES2016 :** Opérateur d'exponentiation, méthode include pour les tableaux

→ **ES2017 :** Programmation asynchrone (async, await)

→ **ES2018 :** Modification des les expressions régulières, Opérateurs rest/spread, asynchronous iteration

→ **ES2019 :** Ajout de fonctionnalité au Array et String ajout de la méthode fromEntries, modification de la méthode toString, du type Symbol

→ **ES2020 :** BigInt, nouvel operateur nullish coalescing, import dynamic, optional chaining operator

→ **ES201 :** Promise.any, combinaison d'opérateur logique, méthode replaceAll pour les String, etc.

TYPESCRIPT

Généralités

Rappel et exemple

TYPESCRIPT : GÉNÉRALITÉS

→ Open source

→ Surcouche ajouté au langage JavaScript

→ Créé par Anders Hejlsberg (C#) et supporté par Microsoft (2012)

→ Pourquoi TypeScript a été créé ?

TYPESCRIPT : GÉNÉRALITÉS

Reproche à Vanilla JavaScript :

→ Initialement créé comme un langage de scripting utilisé pour apporter un peu de dynamisme aux sites web statiques.

→ Mais il est utilisé maintenant pour créer de grandes applications.

→ Langage intéprété, le code écrit est exécuté sans phase de compilation préalable.

→ Langage dynamiquement typé, un élément peut changer de type en cours d'exécution.

→ Les propriétés propres à l'objet (et celles héritées) peuvent être supprimées, modifiées et ajoutées à tout moment.

TYPESCRIPT : GÉNÉRALITÉS

TypeScript :

→ Langage compilé et fortement typé

→ Facile d'apprentissage pour les développeurs OO : permet de créer des classes et les instancier, notion d'interface, héritage simplifié, gestion de l'accès aux données d'une classe.

→ Prend en compte les librairies JS existante (definitivelyTyped).

→ Support dans les IDE (particulièrement VSCode).

Transpiler :

→ Le code TypeScript est transformé en code JavaScript via un transpiler (*tsc*).

TYPESCRIPT : RAPPEL

```
class Point {
    private x: number
    y: number
    readonly ptType: string = "point2D";
    constructor (x: number, y: number, otherType?: string){
        this.x = x;
        this.y= y;
        if(otherType !== undefined){
            this.ptType = otherType;
        }
    }
    scale(n: number): void {
        this.x *= n;
        this.y *= n;
      }
}
```

```
class Point3D extends Point{
    z: number;
    constructor(x: number, y:number, z: number){
        super(x, y, "Point3D");
        this.z = z;
    }
    scale(n: number) {
        super.scale(n);
        this.z *= n;
    }
}
const pt1 = new Point(10, 20);
pt1.scale(2);
console.log(pt1);
const pt2 = new Point3D(30, 40, 6);
pt2.scale(10);
console.log(pt2);
```

```
▶ Point {ptType: 'point2D', x: 20, y: 40}
▶ Point3D {ptType: 'Point3D', x: 300, y: 400, z: 60}
```

TYPESCRIPT : RAPPEL

```typescript
interface ITest {
    id: number;
    name?: string;
}

type TestType = {
    id: number,
    name?: string,
}

function myTest(args: ITest): string {
    if (args.name) {
        return `Hello ${args.name}`
    }
    return "Hello Word"
}

myTest({ id: 1, name: "Toto" })
```

```typescript
// method asynchrone retourne toujours une Promesse
async asyncMethod(): Promise<number> {
    return 1;
}
async method(): Promise<number> {
    // première façon pour récupérer une valeur asynchrone
    const asyncValue: number = await this.asyncMethod();
    // deuxième façon pour récupérer une valeur asynchrone
    this.asyncMethod()
        .then(value => console.log(value))
        .catch(error => console.log('error'));
    return asyncValue;
}
```

TYPESCRIPT : EXEMPLE FETCH API

```typescript
const nbPokemon: number = 10;
interface IPokemon {
    id: number;
    name: string;
    image: string;
    type: string;
}
```


```typescript
async function getPokemon(id: number): Promise<IPokemon>{
    return fetch(`https://pokeapi.co/api/v2/pokemon/${id}`)
        .then(result => result.json())
        .then(result => {
            const type = result.types
                .map((poke: any) => poke.type.name)
                .join(", ");
            const transformedPokemon = {
                id: result.id,
                name: result.name,
                image: `${result.sprites.front_default}`,
                type: type
            }
            return transformedPokemon as IPokemon
        })
}
function getPokemons() {
    for (let i = 1; i <= nbPokemon; i++) {
        getPokemon(i).then(pokemon => {
            console.log(pokemon);
            displayCard(pokemon);
        })
    }
}
```