JavaScript

UN PEU D'HISTOIRE

"JavaScript a été créé en 1995 par Brendan Eich et intégré au navigateur web Netscape Navigator 2.0. L'implémentation concurrente de JavaScript par Microsoft dans Internet Explorer jusqu'à sa version 9 se nommait JScript, tandis que celle d'Adobe Systems se nommait ActionScript. JavaScript a été standardisé sous le nom d'ECMAScript en juin 1997 par Ecma International dans le standard ECMA-262. La version en vigueur de ce standard depuis juin 2022 est la 13e édition." (*wikipédia*)

En 2015, la version ECMAScript Edition 6 (ES6) a apporté une grande quantité de nouveautés au langage (venant le moderniser), notamment les classes.

UTILITÉE DE JAVASCRIPT

JavaScript est exécuté par le navigateur de l'utilisateur. On dit qu'il est exécuté côté **client**.

Ce langage sert généralement à contrôler les données d'un formulaire HTML ou à modifier le document HTML, le plus souvent en réaction à des évènements utilisateur. JavaScript permet donc de faire de la **programmation évènementielle**.

La programmation évènementielle permet d'exécuter une fonction précise si un **évènement** se produit sur la page. Un **évènement** peut être : un déplacement de la souris, un clic, une saisie clavier…

OÙ INCLURE LE CODE JavaScript ?

On peut inclure le code JavaScript de deux manières différentes :

1. directement dans la page HTML, le code sera alors contenu entre deux balises ***<script>…</script>***
2. dans un fichier dédié (portant l'extension **.js**) qui sera lui-même inclus dans la page HTML de cette façon : ***<script src="fichier.js"></script>.***

Bien qu'en pratique on peut inclure le code javascript partout dans la page, il est recommandé dans l'inclure à la toute fin de la balise ***<body>***.

LES CLASSES

Depuis la version ES6 (sortie en 2015), JavaScript permet de créer des classes et des objets, un peu comme en Java.

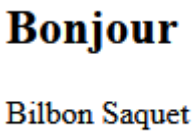
Attention : il est nécessaire de déclarer la classe avant de l'instancier.

Voici un exemple simpliste :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Tests des classes</title>
</head>
<body>
    <h2>Bonjour</h2>
    <div id="etudiant"></div>

    <script>
        class Etudiant {
            constructor(nom, prenom) {
                this.nom = nom;
                this.prenom = prenom;
            }

            get nomComplet() {
```

```
                return this.prenom + ' ' + this.nom;
            }
        }

        const etu = new Etudiant('Saquet', 'Bilbon');

        document.getElementById('etudiant').innerHTML = etu.nomComplet;
    </script>
</body>
</html>
```

Et son résultat dans le navigateur :



Bilbon Saquet

LE FORMAT JSON

“***JavaScript Object Notation*** (**JSON**) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l’information structurée comme le permet le format XML, par exemple. Créé par Douglas Crockford entre 2002 et 2005, la première norme du JSON est ECMA-404 qui a été publiée en octobre 2003.

La dernière version des spécifications du format date de décembre 2017.

Bien qu’utilisant une notation JavaScript, JSON est indépendant du langage de programmation (des dizaines de langages de programmation ont intégré JSON, de base). JSON sert à faire communiquer des applications dans un environnement hétérogène. Il est notamment utilisé comme langage de transport de données par AJAX et les services Web.” (*wikipedia*)

Voici un exemple d’objet JavaScript représenté au format JSON :

```
{
    "nom" : "Saquet",
    "prenom" : "Bilbon"
}
```

Il s’agit tout simplement d’un “tableau” de clés associées à une valeur.

Évidemment, le langage JavaScript nous permet de passer de l’un à l’autre. C'est-à-dire de passer d’un objet à sa version JSON (et réciproquement).

Voici un exemple :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Tests des classes</title>
</head>
<body>
    <h2>Bonjour</h2>
    <div id="etudiant"></div>

    <script>
        class Etudiant {
            constructor(nom, prenom) {
                this.nom = nom;
```

```
                this.prenom = prenom;
            }

            get nomComplet() {
                return this.prenom + ' ' + this.nom;
            }
        }

        const etu = new Etudiant('Saquet', 'Bilbon');
        const versionJson = JSON.stringify(etu);

        const donneesJson = '{"nom" : "Saquet", "prenom" : "Frodon"}';
        const donneesTableau = JSON.parse(donneesJson);
        const etu2 = Object.assign(new Etudiant(), donneesTableau);

         document.getElementById('etudiant').innerHTML =
            etu.nomComplet + ' et ' + etu2.nomComplet;
      </script>
</body>
</html>
```

et dont voici le résultat à l'écran :

Bilbon Saquet et Frodon Saquet

LES WEB SERVICES

Un *Web Service* est une application qui permet d'échanger des données avec d'autres applications web, et ce, indépendamment des langages de programmation utilisés. Actuellement, l'architecture la plus connue et celle à privilégier est l'architecture REST (representational state transfer).

Le principe des services REST est d'échanger via le protocole HTTP des données représentées, le plus souvent, au format JSON.

Exemple de fonctionnement simpliste :
Une application **serveur** Java connectée à une BDD, requêtable sur le net (via **HTTP**), met à disposition la liste des étudiants du DUT Info. Ce serveur donne ces informations dans le format JSON.
Une application **client** JavaScript, utilise des requêtes **HTTP** pour récupérer ces données et les afficher dans une page HTML.

Dans l'architecture REST, en plus de lire des données depuis un serveur, il est également possible d'en ajouter, d'en modifier ou d'en supprimer.

Voici les 4 principales actions possibles en REST :

Méthode HTTP	Description
GET	Permet de lire une ressource.
POST	Permet de créer une ressource.
PUT	Permet de modifier une ressource.
DELETE	Permet de supprimer une ressource.

LES REQUÊTES AJAX

“Ajax est une méthode utilisant différentes technologies ajoutées aux navigateurs web entre 1995 et 2005, et dont la particularité est de permettre d'effectuer des requêtes au serveur web et de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète. Cette architecture informatique permet de construire des applications Web et des sites web dynamiques interactifs. Ajax est l'acronyme d'***asynchronous JavaScript and XML*** : *JavaScript et XML asynchrones*.

Ajax combine JavaScript et DOM, qui permettent de modifier l'information présentée dans le navigateur en respectant sa structure, les API Fetch et XMLHttpRequest, qui servent au dialogue asynchrone avec le serveur Web ; ainsi qu'un format de données (XML ou JSON), afin d'améliorer maniabilité et confort d'utilisation des applications. XML, cité dans l'acronyme, était historiquement le moyen privilégié pour structurer les informations transmises entre serveur Web et navigateur, de nos jours le JSON tend à le remplacer pour cet usage.” (*wikipedia*)

De nos jours, **XMLHttpRequest** est à **proscrire**. Il a été remplacé par **Fetch** depuis la version ECMAScript 6 (en 2015).

LES PROMESSES

L'objet **Promise** (pour « promesse ») est utilisé pour réaliser des traitements de façon **asynchrone**. Une promesse représente une valeur qui peut être disponible maintenant, dans le futur, voire jamais.

Pour gérer la réussite d’une Promesse, il faut utiliser sa fonction **then()**.
Pour gérer l’échec d’une Promesse, il faut utiliser sa fonction **catch()**;

Voir l’exemple avec l’utilisation de **Fetch**.

L’API FETCH

Désormais, pour lancer une requête AJAX, il faut utiliser la fonction **fetch()**.
Cette fonction retourne une Promesse.

Voici un exemple simple :

```
fetch(URL)
    .then(function (reponse) {
        return reponse.json();
    })
    .then(function (json) {
        console.log(json);
    })
    .catch(function (erreur) {
        alert('erreur : ' + erreur);
    })
    .finally(function () {
        // cette fonction est exécutée dans tous les cas
    })
```

Ou sous cette forme équivalente.

```
fetch(URL)
    .then(reponse => {
        return reponse.json();
    })
    .then(json => {
        console.log(json);
    })
    .catch(erreur => {
        alert('erreur : ' + erreur);
    })
```

Ou encore…

```
fetch(URL)
    .then(reponse => reponse.json())
    .then(json => console.log(json))
    .catch(erreur => alert('erreur : ' + erreur))
```

LES *FRAMEWORKS* JavaScript

Comme pour tout langage de programmation, il existe aussi des ***frameworks*** pour JavaScript.

Qu'est-ce qu'un *framework* ?
"Un *framework* est un ensemble d'outils et de composants logiciels organisés conformément à un plan d'architecture et des *patterns*, l'ensemble formant un « squelette » de programme, un canevas.
Un *framework* est conçu en vue d'aider les programmeurs dans leur travail. L'organisation du *framework* vise la productivité maximale du programmeur qui va l'utiliser. Ceci aura pour conséquence de baisser les coûts de construction et de maintenance du programme." (wikipedia)

Les 3 frameworks les plus connus sont :

- React
- Vue JS
- Angular.

Mais il en existe beaucoup d'autres…

LA SÉCURITÉ DANS LE WEB

La sécurité intervient à différents niveaux.
En vrac :

- serveur HTTPS
- hashage de mot de passe
- validation des formulaires
- préparation des requêtes SQL (côté serveur)
- authentification
- gestion des droits utilisateur
- …

Un peu de vocabulaire.

1.
Un mot de passe se ***hash***.
Il n'existe pas de fonction inverse pour retrouver le mot de passe.

2.
On peut **chiffrer** des données, par exemple avec l'algorithme **RSA**, pour ensuite les **déchiffrer** plus tard par un lecteur possédant la clé.
RSA est un chiffrement **asymétrique** et utilise une **clé de chiffrement**.

3.
Le mot **décrypter** signifie : tenter de retrouver un message en clair, **sans** posséder la **clé de chiffrement**.
Ainsi, le mot **crypter** n'existe **pas**. Il est **impossible** de crypter un message !