GALIMATECH
L'EXPERIENCE DIGITALE

Apprendre le Javascript

***

Plan du cours

- Introduction
- Utilisation du javascript
- Les commentaires
- Les variables
- Les type de données
- Les chaînes de caractères
- Les arrays
- Les functions
- Les fonctions anonymes
- Le rest operator
- Conditions logiques
- Les opérateurs de comparaison
- Les scopes
- Les objets
- Les boucles
- Manipulations du DOM
- Le destructuring assignment
- Import et require
- Liens utiles

Le Javascript est de nos jours l’un des langages de programmation les plus utilisés. Il est à la base des frameworks les plus populaires sur le Web et dans d'autres domaines de la programmation.
C’est un langage très puissant qui permet d’ajouter des animations et des automations sur nos sites web.

Utilisation du Javascript

Il est possible d'intégrer le Javascript dans un document HTML en utilisant deux méthodes.

- Écriture du Javascript dans le HTML. Avec cette méthode nous allons utiliser les balises script> comme suite.

```
<script>
  //INSTRUCTIONS
</script>
```

  Entre les balises script nous mettrons toutes nos instructions Javascript.

- Utilisation d'un fichier externe. Avec cette méthode nous allons utiliser les balises <script> pour importer un fichier:`<script src="chemin/vers/fichier_javascript.js"></script>`
  Les balises sont vides et on utilise un attribut src dont la valeur est le chemin vers le fichier Javascript. Il faudra donc avoir un fichier avec comme extension .js, dans lequel nous mettrons tout le code Javascript.

Les commentaires

Les commentaires en Javascript sont des instructions ayant pour objectif d'expliquer du code. Il existe deux types de commentaires:
nous avons les commentaires écrits sur une ligne avec la syntaxe suivante

```
//un commentaire sur une ligne
```

Nous avons aussi les commentaires sur plusieurs lignes écrites entre les caractères /* et */

```
/*
Text
Du
Commentaire
*/
```

Les commentaires sont très utiles pour expliquer la logique d'un code. Mais il faudra faire attention à ne pas polluer son script avec.

Les variables

Les variables sont des conteneurs qui vont renfermer la valeur d'une information et son type de donnée. Elles sont essentielles en programmation.
Pour initialiser une variable nous allons utiliser deux syntaxes;

```
let variable = "valeur de la variable";
```

cette syntaxe est la plus commune. Elle crée une variable et lui attribue une valeur. Cette méthode est récente elle remplace la méthode utilisant le mot-clé 'var'.

```
let user name = "Moussa DIOP";//On définit le nom d'un utilisateur
let age = 50//on crée une variable qui va contenir l'âge d'un utilisateur
```

```
const NOM_CONSTANTE = 90809;
```

cette méthode est utilisée pour définir une constante. Les constantes ne sont pas des variables. Ce sont des conteneurs qui une fois défini ne peuvent plus être modifiés.

```
const CNI = 1300200045201;//On cree un constante pour le CNI
```

les type de données

Les types de données sont les différents formats que peuvent prendre les informations dans un ordinateur.
Javascript comporte sept (7) types de données:
undefined, **null**, le **boolean**, le **string**, les **symbole**, les **valeurs numériques** et les **objets**.

- Undefined désigne une variable qui n'a pas encore de valeur; qui est indéfinie.
- Null désigne une variable qui est vide.
- Boolean désigne une variable qui est soit égale à true soit false.
- String désigne les chaînes de caractères. C'est-à-dire des valeurs alphanumériques.
- Le symbole désigne une valeur primitive immuable et unique.
- Les valeurs numériques regroupent deux autres type de données:
  - les **integers** pour les entiers
  - les **floats** qui représentent les nombres flottants.
- Les objets sont des ensembles complexes pouvant contenir d'autres types de données d'une manière structurée.

Les chaines de caracteres

Les chaînes de caractères aussi appelés string, sont toujours entre des quotes '' ou des doubles quotes"".

```
let address = "Cité keur Guorgui Villa N122 - Dakar - Senegal"
```

Elles utilisent le \ pour échapper des caractères spéciaux.

```
let message = 'Je vous pris d\'accéder avec ce mot de passe: "passer"';
```

Nous pouvons concaténer un string avec l'opérateur +. La concaténation consiste à coller un autre type de variable à un string.

```
let reponse = "je vous envoie "+5000+" FCFA";
```

La norme ECMAScript 2015 a introduit un nouveau type de string appelé **gabarit de chaîne de caractères** ou **template strings**. Ce sont des caractères entourés de back-ticks ``. Il permet une plus facile concaténation et offre un formatage plus poussé.

```
let text     = `Ce texte utilise un gabarit de chaîne de caractères. Il accepte les retour à la
ligne et les opérations ${2+3}`;
```

Les arrays

Les arrays sont une manière d'enregistrer plusieurs valeurs dans une variable; ce sont des tableaux. La syntaxe de création d'un array est :

```
let tableau_donnees = ['Moussa', 40, 1.88,['1','2','3','4']]
```

Nous pouvons voir dans l'exemple qu'un array peut contenir plusieurs types de données et même un autre array. Un array contenant un array est appelé 'nested array' ou 'multidimensional array'.
On accède au contenu d'un array en utilisant des crochets [index]. La valeur entre crochet est l'index de la valeur cible. Les indexes commence par zéro.

```
let taille = tableau_donnees[2];// Va retourner 1.88
```

```
tableau_donnees[1] = 41;//On change le tableau à l'index 1; 40 devient 41
```

Nous pouvons utiliser des méthodes spéciales pour manipuler un array.
tableau_donnees.**push():** va ajouter une nouvelle valeur à la fin du array
tableau_donnees.**pop():** supprimer la dernière valeur d'un array.
tableau_donnees.**shift():** va supprimer le premier index du array.
tableau_donnees.**unshift():** va ajouter une nouvelle valeur au début du tableau.

Les fonctions

Une fonction est une liste d'instructions réutilisable ayant une tâche bien définie. Les fonctions sont conçues pour regrouper une ou plusieurs instructions dans une entité pour éviter la répétition.
La syntaxe pour créer une fonction est la suivante:

```
function nomFunction([attributes]){
    instructions
  }
```

Dans le second exemple nous utilisons un paramètre dans la définition de la fonction . Ce paramètre sera ensuite utilisé dans les instructions

```
function sayHello(name){
   return "Mes salutations "+name;
  }
```

Pour utiliser une fonction nous pouvons faire:

```
sayHello("Mousssa");
```

Les fonctions anonymes

Les fonctions anonymes sont, comme leur nom l'indique, des fonctions qui ne vont pas posséder de nom. Elle sont utilisées lorsqu'on n'a pas besoin d'appeler notre fonction par son nom c'est-à-dire lorsque le code de notre fonction n'est appelé qu'à un endroit dans notre script.

Nous pouvons utiliser une fonction anonyme de trois façons:

- Utilisation d'une variable: nous pouvons utiliser une variable pour enregistrer la fonction.

```
let function_var = function(){
    //instructions
}  function_var();
```

- Auto-invoquer la fonction anonyme: avec cette méthode la fonction est invoquée au moment de sa définition. `(function(){/*instructions*/})();`
- Utiliser en tant que callback: les callbacks sont des fonctions appelées en paramètre d'une autre fonction. `param.addEvenListener('click',function(){/*instructions*/});`

Rest parameters

Le REST parameter permet de définir un nombre indéfini de paramètres pour une fonction.

Ces paramètres sont accessibles dans la fonction sous forme d'un array.

```
function param_joueur(nom, equipe, ...sposonr) {
        alert(`Hello ${nom}`);
        alert(`Vous jouez a ${equipe}`, "Vous avez les spoonsors suivants");
        for (let i = 0; i < sposonr.length; i++) {
          alert(sposonr[i]);
        }
      }

param_joueur("nom_eleve2", "univesite_name", "option1","option2");
param_joueur("nom_eleve2", "univesite_name", "option1","option2","option3","option4","option5");
```

Les Conditions logiques

Les conditions logiques sont utilisées pour exécuter du code en fonction d'une conditions. Ils sont définis comme suite:

```
if(conditions == true)
{
//Instructions
}
```

On peut enfiler plusieurs conditions ayant des instructions différentes.

```
if(age < 12){
    return "Vous etes jeune";
} else if(age < 28){
    return "Vous etes relativement jeune"
} else if(age > 29){
    return "Vous n'etes plus jeune"
}
```

Et si aucune des conditions n'est vérifiée on peut alors utiliser le mot clé `else`

```
if(age < 12){
    return "Vous êtes jeune";
  } else if(age < 28){
    return "Vous êtes relativement jeune"
  } else if(age > 29){
    return "Vous n'êtes plus jeune"
  } else {
    return "Vous n'êtes pas encore nés"
  }
```

Les opérateurs de comparaison

Les opérateurs de comparaison permettent d'agencer plusieurs conditions pour faire des vérifications précises.

Comme opérateurs nous avons:

- **Les opérateurs d'égalité** qui vont vérifier si deux valeurs sont égales. Ils sont de deux sortes:
  - L'égalité faible avec == qui va vérifier si les valeurs sont identiques mais ne vérifient pas le type de donnée. `12 == "12"; //va retourner true`
  - **L'égalité forte** avec === va vérifier le type de donnée et la valeur. `12 === "12"; //va retourner false`
- On peut vérifier si une valeur est strictement supérieure avec > `12 > 4; //retourne true`
- On peut avoir l'opposé de cet opérateur avec < `12 < 42; //retourne true`
- En ajoutant le signe = a ces deux dernières opérateurs on inclus l'égalité dans la comparaison

`12 >= 12; //retourne true  4 <= 4; //retourne true`

- L'opérateur **&&** qui va permettre de vérifier si deux ou plusieurs conditions sont toutes égales a true

`12 > 4 && 12 == '12' //retourne true`

- L'opérateur || permet de vérifier si au moins une des conditions est égale à true `12 > 42 || 12 == '12'`

Les Scopes

Le scope définit l'accessibilité d'une variable à travers les différents blocs du script. Avant la norme ECMA2015 la définition des variables se faisait avec ou sans le mot-clé var. Déclarer une variable sans le mot-clé var, faisait que cette variable devenait globale c'est-à-dire accessible de partout à travers le code.
Le mot-clé var crée une variable dont le scope pouvait être confus. Depuis ECMA2015 le mot-clé let est préférable au var.

La différence entre le var et le let est que **let** a un scope base sur les blocs alors que **var** a un scope base sur les fonctions.
Cela veut dire qu'une variable créée avec **let** dans un bloc n'est accessible qu'au sein du bloc. Alors que cela n'est vrai pour **var** que dans le cas de variables créées dans une fonction.

Les blocs sont des ensembles tels que les fonctions ou les conditions logiques.

Les objets

Les objets sont des types de données essentiels en Javascript. Ce sont des arrays dont les indexes sont des références. On initialise un objet avec les accolades {}:

```
var utilisateur = {
        nom: "Famara",
        prenom: "Diouf",
        taille: 12.3,
        age: 23,
        options: ["Maths", "Literature", "Politique"]
    };
```

Les noms des références doivent être uniques. Un objet peut contenir tout type de donnée; même d'autre objet ou des fonctions.

Les objets

Nous disposons de deux méthodes pour accéder aux données dans un objet:

Notation avec un point: on va utiliser un point avec les références pour accéder aux données.

```
var utilisateur = {
        nom: "Famara",
        prenom: "Diouf",
      };

utilisateur.name;
utilisateur.nom;//retournera Famara
utilisateur.nom = "Ansou"; //Va changer la valeur de ansou
```

Notation avec crochet: cette notation est pareille à la manière d'accéder aux données d'un array à la différence qu'elle utilise les références à la place des index.

```
var utilisateur = {
        nom: "Famara",
        prenom: "Diouf"
      };
utilisateur["nom"];
```

Nous pouvons noter que les références sont entre quotes `""`

Les Boucles

Les boucles permettent d'exécuter une tâche à plusieurs reprises.

Nous avons différents types de boucles en JavaScript.

- La boucle **for:** elle va s'exécuter tant que la condition est vérifiée. Sa syntaxe est

```
for (let i = 0; i < limit; i++) {
        // instructions
    }
```

La variable `i` est utilisée comme compteur. Elle est incrémentée à chaque itération `i++`. La reference `limit` désigne le nombre d'itérations que la boucle ne doit pas dépasser.

- La boucle do ... while(): cette boucle va d'abord exécuter une action avant de vérifier la condition.

```
do {
        //instructions
} while(conditions)
```

Avec cette boucle l'instruction est exécutée au moins une fois.

Les Boucles

- La boucle while: elle exécute une action tant qu'une condition est vraie.

```
while(conditions){
        //instructions
    }
```

Les boucles peuvent être très utiles mais il faudra faire attention à mettre en place un process qui fera qu'à un moment donné la condition utilisée sera fausse. Une erreur dans les conditions utilisée peut très vite mener à une boucle infinie qui peut planter le navigateur même le système.

Manipulations du DOM

La manipulation du DOM (Document Object Model), consiste à différentes actions qui vont permettre de modifier l'état des balises HTML d'une page Web. Cette manipulation vas se faire avec l'objet document. Cet objet document utilise plusieurs fonctions pour accéder et modifier les éléments de la page:

`document.getElementById('id')`: cette fonction va retourner l'élément HTML ayant l'identifiant 'id'.

`document.getElementsByClassName('class_name')`: va retourner un array des éléments avec la classe 'class_name'.

`document.getElementsByTagName('p')`: cette fonction cible toutes les balises ayant le nom passé en paramètre; ici la balise p.

`document.querySelector('.selecteur_css')`: cette fonction utilise des **queryselectors** pour indexer un élément de la même manière qu'on le ferait en CSS.

Manipulations du DOM

Une fois un élément sélectionné nous pouvons utiliser d'autres méthodes pour les manipuler.

innerHTML: permet d'éditer le texte et les balises HTML dans un élément HTML.

innerText: permet d'éditer le texte d'un élément HTML.

classList: cette méthode permet de manipuler les classes d'un élément. Elle dispose de deux autres méthodes, **add** et **remove**, qui permettent respectivement d'ajouter ou retirer une class.

class_name: cette méthode retourne ou définit les classes d'un élément. Quand elle est utilisée pour définir les classes elle va remplacer le contenu de l'attribut 'class'.

remove(): va supprimer un élément de la page Web.

removeChild: va supprimer les éléments enfants.

createElement: permet de créer un nouvel élément

appendChild: cette méthode permet d'ajouter un élément parmi les enfants d'un autre. Il est souvent utilisé pour ajouter un élément créé avec createElement dans la page.

Affecter par décomposition (destructuring assignment)

L'affectation par décomposition consiste à distribuer des valeurs à des variables depuis un objet.

Elle permet de remplacer le code suivant:

```
let x = objet.valeur1;
let y = objet.valeur2;
let z = objet.valeur3;
```

Par

```
let {x,y,z} = objet;
```

Les variables devant recevoir les valeurs des références de l'objet doivent avoir les mêmes noms que ces références. Toutefois on peut utiliser des noms de variables différents des références comme suite:

```
let {reference:new variable, reference3:new variable2, reference2:new variable3} = objet;
```

IMPORT ET EXPORT

En javascript, nous pouvons inclure le script d'un fichier dans un autre. Cette pratique permet d'organiser les scripts en fonction de leurs tâches.

Les normes ES6 ont introduit l'importation de modules externes.

```
import {nom_fonction,nom_autre_fonction} './chemin/vers/fichier.js';
```

Pour que cette importation soit possible, il faut que les fonctions appelées soient exportées depuis leur source.

```
export function nom_fonction(parametres){/*instructions*/}
export function nom_autre_fonction(){/*instructions*/}
export const foo = "valeur";
```

On peut aussi utiliser une fonction qui est exportée par défaut

```
export default fonction_nom(){/*instructions*/}
```

La manière d'importer cette fonction se fera alors sans les accolades:

```
import NomFonction from './chemin/vers/fichier.js'
```

Liens utiles

GALIMATECH
L'EXPERIENCE DIGITALE

- https://openclassrooms.com/fr/courses/2984401-apprenez-a-coder-avec-javascript
- https://developer.mozilla.org/fr/docs/Web/CSS/display
- https://www.w3.org/Style/Examples/007/units.fr.html
- https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Grid_Layout
- https://flexbox.help/
- https://the-echoplex.net/flexyboxes/
- https://developer.mozilla.org/fr/docs/Web/CSS/Using_CSS_custom_properties