📝 Les bases du langage

Objectifs du cours

Objectifs du cours de JavaScript

Dans les cours précédents, vous avez appris à créer une page web, ainsi qu'à la décorer. Cependant, celle-ci ne présentait aucune interaction réellement possible. L'utilisation de JavaScript permet à un site d'ajouter d'innombrables fonctionnalités.

Dans ce cours, vous allez apprendre à :

- Utiliser les bases du langage JavaScript
- Manipuler des variables et des données
- Créer des fonctions

Présentation du langage

La version actuelle de JavaScript est une implémentation d'un langage de script appelé EcmaScript. EcmaScript est une standardisation d'une ancienne version de JavaScript créée par Netscape dans les années 90.

À l'origine, ce langage a été développé dans le but de rajouter un système d'interactions aux pages Web, qui étaient complètement statiques.

Il était ainsi devenu possible de :

- Définir une action au clic
- Vérifier des données entrées dans un formulaire
- Détecter les changements de page et lancer une alerte

Ce qui est affiché par le navigateur se résume à l'interaction entre le HTML, le CSS et le JavaScript.

Chacune de ses technologies a un rôle très différent des autres. La rigueur du développeur est de mise ici, car il faut retenir que :

- Tout ce qui sert à structurer l'information sur la page se retrouve dans un fichier .html
- Tout ce qui sert à habiller et décorer cette page (couleurs, tailles de texte, etc.) se retrouve dans le .css
- Toute la logique, les calculs et les interactions utilisateur se retrouvent dans le JavaScript.

Une des règles de base est de ne pas mélanger ces concepts afin de faciliter le développement.

- Pas de règles CSS dans le JavaScript
- Pas de balises HTML dans le JavaScript (Même si des frameworks comme Angular ou React de nos jours sont des exceptions à la règle)

Un script .js (JavaScript) est un fichier contenant plusieurs instructions, placées les unes à la suite des autres. Chacune de ces instructions peut être vue comme un ordre que vous donnez au moteur JavaScript, une action que vous souhaitez réaliser, à la manière d'une recette de cuisine :

Faire fondre le beurre

Mélanger le beurre et la farine

Verser le chocolat petit à petit

Les instructions se suivent et sont effectuées les unes à la suite des autres, du haut vers le bas.

Syntaxe de base

Chaque instruction doit se terminer par un point-virgule ; afin d'indiquer au moteur JavaScript que cette instruction est terminée.

L'indentation, qui consiste en l'ajout d'espaces ou tabulations en début de ligne pour lire plus facilement le code, n'est pas prise en compte par le navigateur mais est primordiale pour le développeur.

Elle permet d'analyser beaucoup plus facilement son code ou celui d'un autre et permet d'aider à visualiser le chemin que parcourt l'information.

De plus, une erreur de structure comme un oubli de parenthèse est plus facilement identifiable.

Les commentaires, qu'ils soient en

```
// ligne
```

ou bien

```
/*
En bloc
*/
```

aident à la compréhension et sont à utiliser à volonté pour expliquer à quoi sert un ligne ou un bloc de code.

```
// Cette fonction sert à calculer le nombre d'années qui nous séparent
du prochain millénaire

/*
J'ai mis tel code pour telle raison
Se référer à la documentation sur tel site pour plus d'explications
*/
```

Un code bien indenté et bien commenté permet de s'y replonger facilement, et aide le développement en équipe: reprendre le code mal indenté d'un autre est comme relire un texte bourré de fautes et sans espace... c'est possible, mais désagréable !

En JavaScript, les erreurs ne pardonnent pas. Si l'interpréteur voit une variable qu'il ne connaît pas, une opération sur un mauvais type de donnée ou autre erreur, il s'arrête immédiatement et le code qui se trouve après n'est pas exécuté. Si un script ne fonctionne pas comme il devrait, il faut aller vérifier la console de l'inspecteur en premier. Toutes les erreurs sont affichées dedans.

La console

Chaque navigateur dispose d'un outil qui permet d'inspecter les différents constituants de la page (HTML, CSS, Javascript).

Dans cet inspecteur, on peut trouver une console qui dispose d'un interpréteur et qui peut être utilisée pour exécuter du code JavaScript.

On peut y écrire une ou plusieurs instructions, que l'on peut exécuter en appuyant sur entrée.

Par exemple :

```
4+5; // Affiche 9
```

Une fonctionnalité extrêmement importante de la console est console.log() qui permet d'y afficher une information de son choix.

Dans le code qu'on écrit dans le fichier JavaScript, on peut ainsi écrire:

```
ma_variable = 8;
console.log(ma_variable); // La console du navigateur affiche 8
```

Les variables

Une variable est un contenant, un réceptacle, permettant d'y stocker des valeurs.

Il est possible de créer une variable en lui donnant un nom, de lui assigner une valeur, de modifier celle-ci ou de la supprimer.

Par exemple :

Si une personne marche vers un magasin, le nombre de mètres entre le magasin et la personne diminue avec le temps. On pourrait alors stocker cette distance dans une variable, nommée par exemple d.

À un certain moment, la personne se trouve à une *distance* de 10 mètres du magasin. Notre variable d a alors comme valeur 10.

Plus tard, la personne a avancé, et se trouve à une distance de 9 mètres. Notre variable vaut maintenant 9.

En JavaScript, si je veux assigner un montant de *10* à une variable *d*

d = 10

Maintenant que la personne a avancé vers le magasin, je peux modifier la variable *d*

d = 9

Une variable peut être globale ou avoir une portée limitée (scope).

Quand une variable est définie sans le mot-clé var, elle est crée dans l'objet window, qui a une portée globale.

Une variable globale est accessible depuis n'importe quelle partie du code , même depuis une librairie tierce chargée pour d'autres fonctionnalités.

Elle est ainsi facilement écrasable si le nom est trop commun.

Ainsi, il faut toujours contrôler la portée de la variable (savoir quelle partie du code peut lire et réécrire celle-ci).

Déclarer une variable avec var limite sa portée à la fonction actuelle et tous ses descendants.

```
function maFonction(){
   var locale = "Je suis une variable locale";
   globale = "Je suis globale";
   console.log(locale); // Affiche "Je suis une variable locale"
}

maFonction();
console.log(globale); // Affiche "Je suis globale"
```

```
console.log(locale); // Affiche une erreur, locale n'est pas accessible
ici
```

Les types de données

Tout langage de programmation possède plusieurs types de données.

En français, "bonjour, je m'appelle Bob" correspond à une phrase et 23 à un nombre.

En Javascript, nous avons, entre autres :

- string (chaîne de caractères) correspond à notre concept de phrase et sert à représenter toute donnée textuelle : salutation = "Bonjour, comment allez-vous?"
- number (nombre) est utilisé pour les nombres qu'ils soient entiers ou flottants.

Un entier est un nombre sans décimale, par exemple *25*

Un flottant est un nombre avec une décimale (séparée par un point et non une virgule), par exemple *36.1452*

- boolean est un type spécial très utilisé en logique. Ne peut contenir que deux valeurs: true (vrai) et false (faux).
- array et object sont utilisés pour contenir des listes.

Un array, ou tableau en français, est utilisé pour des listes simples : liste = [12, 15, "pomme"];

Un object est une liste contenant des associations.

Si je veux lister le contenu de mon frigo et le trier par type d'aliments :

```
frigo = {
   "fruits" : ["pomme", "poire", "banane"],
   "legumes" : ["tomate", "poireau"],
   "laitages" : ["yaourt", "lait"],
};
```

Opérations et incrémentation

Les opérations sur les nombres s'apparentent à ceux sur calculatrice :

- + pour l'addition
- - pour la soustraction

- * pour la multiplication
- / pour la division

```
operation1 = 1+2; // operation1 contient maintenant 3
operation2 = 8-4; // operation2 contient maintenant 4
operation3 = 6*3; // operation3 contient maintenant 18
operation4 = 10/2; // operation4 contient maintenant 5
```

Le + peut aussi être utilisé sur des chaînes de caractères, ce qui aura pour effet de coller les différents textes les uns à la suite des autres. Cette opération s'appelle une concaténation :

```
ma_chaine = "je m'appelle" + " bob"; // ma_chaine contient maintenant
"je m'appelle bob"
```

% modulo sert à récupérer le reste d'une division

```
operation1 = 10%2 // operation1 contient 0, car il n'y a pas de reste
operation2 = 17%3 // operation2 contient 2, car 17 / 3 = 5 (plus un
reste de 2)
```

Dans le cas des *tableaux* (ou arrays), je peux récupérer une valeur si je connais son index (emplacement dans le tableau).

J'utiliserai l'expression tableau[index]

L'index commence toujours à 0, ainsi le deuxième élément de la liste est à l'index 1.

```
fruits = ["pomme", "poire", "banane"];
un_fruit = fruits[1]; // un_fruit contient "poire"
```

Je peux faire de même avec un *object* si je connais la clé avec l'expression objet["cle"] ou objet.cle

```
frigo = {
   "fruits" : ["pomme", "poire", "banane"],
   "legumes" : ["tomate", "poireau"],
   "laitages" : ["yaourt", "lait"]
};
```

```
mes_fruits = frigo.fruits; // mes_fruits contient ["pomme", "poire",
"banane"]
mes_legumes = frigo["legumes"]; // mes_legumes contient ["tomate",
"poireau"]
```

Les opérateurs mathématiques fonctionnent de la même manière que sur une calculatrice.

```
2 * 3 = 6
12 - 2 * 3 = 6
30 / 3 / 2 + 2 * 3 = 11
```

L'incrémentation et la décrémentation sont un raccourci pour ajouter ou soustraire une valeur à une variable.

- ++ signifie rajouter 1 à la variable
- -- signifie soustraire 1 à la variable
- += permet d'ajouter un nombre à une variable
- -=, *=, /=, %= sont des déclinaisons de cet opérateur

```
heure = 5;
heure++; // heure devient 6
heure += 2; // heure devient 8
heure /=2 // heure devient 4
heure %= 3 // heure devient 1
```

x++ et ++x ont le même résultat à la différence près que x++ retourne la valeur de x AVANT d'effectuer l'opération et ++x retourne la valeur de x APRÈS avoir effectué l'opération.

```
x = 5;
console.log(x++); // Donne 5 à console.log mais x est maintenant égal à
6
console.log(++x); // x est maintenant égal à 7 et donne 7 à console.log
```

Opérateurs logiques et comparaisons

Opérateurs logiques

Les opérateurs logiques sont des opérateurs qui établissent une liaison entre deux valeurs, ou effectuent une opération sur une simple variable.

Si l'on me demande de ramener des pommes et des poires

```
pommes && poires
```

et que je ne ramène que des pommes ou des poires

```
pommes || poires
```

cela ne suffira pas.

Je devrai ramener des pommes et des poires pour que l'expression se vérifie et devienne vraie.

En calcul booléen, les valeurs n'ont que deux états différents : true et false. Ils sont utilisées pour des opérations de logique.

L'énoncé 7 est plus grand que 8 est false, 10 est plus petit que 11 et 2000 plus grand que 1995 donne true.

```
7 > 8; // Retourne false
10 < 11 && 2000 > 1995; // Retourne true
```

Opérateurs de comparaison

```
// Toutes ces instructions retournent true
6 > 5; // Plus grand que
8 < 9; // Plus petit que
6 >= 6; // Plus grand ou égal à
10 <= 11; // Plus petit ou égal à
"bonjour" =="bonjour";  // Égal à
"bonjour" != "au revoir" // Différent de
45 === 45 // Strictement égal à (compare la valeur et aussi le type de donnée)
"bonjour" !== 2 // Strictement différent de

5 > 6 && 8 < 9 // 5 plus grand que 6 ET 8 inférieur à 9
```

```
texte = "bonjour";
texte == "au revoir" || texte == "bonjour"; // texte est égal à "au
revoir" OU texte est égal à "bonjour"

// ! est un opérateur de négation qui inverse la valeur
!false == true
!(6 < 4) // 6 NON PLUS PETIT QUE 4
```

La comparaison if else

Les instructions if et else servent à introduire des conditions pour permettre de n'exécuter du code que si celles-ci sont satisfaites.

Elles correspondent aux termes français *si* et *sinon*.

```
if(porte_verrouillee){
    deverrouiller_porte();
}
```

Peut se traduire par :

Si la porte est verrouillée alors déverrouiller la porte.

```
if(melon){
    manger_melon();
}
else{
    manger_chips();
}
```

Si j'ai du melon alors je le mange, sinon je mange des chips.

```
var frigo = {
    melon: 2
    pasteque: 1,
    parme: 0
};

if(frigo.melon && frigo.parme){ // SI j'ai du melon ET du jambon de
parme, 0 correspond à false en booléen
    manger(frigo.melon, frigo.parme);
}
```

```
else if(frigo.fromage){ // SINON SI j'ai du fromage, retourne undefined
qui correspond à false
    manger(frigo.fromage);
}
else{ // SINON
    if(frigo.pasteque){ // SI j'ai de la pastèque
        manger(frigo.pasteque);
    }
    else{
        manger("chips");
    }
}
```

Les fonctions

Une fonction est un bloc constitué de plusieurs instructions qu'on peut exécuter à sa guise.

Le mot-clé function doit être suivi de parenthèses () qui contiendront les paramètres, puis d'accolades {} qui contiendront les instructions.

Pour garder une référence à la fonction, on peut:

Mettre un nom après le mot-clé function

```
function bonjour(){
    console.log("Bonjour!");
}
```

Ou assigner la fonction à une variable

```
bonjour = function(){
    console.log("Bonjour!");
}
```

bonjour contient une référence à la fonction. On pourrait dire qu'elle pointe du doigt la fonction elle-même.

Pour exécuter une fonction, on prend la variable bonjour qui lui fait référence et on lui accole des parenthèses.

```
bonjour(); // La console affiche “Bonjour!”
```

Une fonction peut aussi envoyer une valeur qui sera ainsi récupérée à l'endroit où la fonction a été lancée.

```
function bonjour(){
   return “Bonjour!”;
}

salutations = bonjour();
console.log(salutations); // Affiche “Bonjour!”
```

Tableaux et objets

```
var mon_tableau = [10, 20, 30, 40, 50];
var frigo = {
   fruits: ["banane", "mangue", "cerise"],
   legumes: ["courgette", "tomate", "poireau"]
};
console.log(mon_tableau[2]); // Affiche 30
console.log(frigo.fruits[0]); // Affiche "banane"

var type = "legumes";
console.log(frigo[type][1]); // Affiche tomate, si on utilise la
notation par crochets [], on peut passer une variable à la place d'une
clé

.push() permet de rajouter un élément à un tableau.
mon_tableau.push(100);
console.log(mon_tableau); // Affiche [10, 20, 30, 40, 50, 100]
.pop() renvoie la dernière valeur du tableau tout en l'enlevant de
celui-ci
mon_tableau.push("banane"); // mon_tableau est maintenant égal à [10,
20, 30, 40, 50, 100, "banane"]
console.log(mon_tableau.pop()); // Affiche "banane", mon_tableau est
maintenant égal à [10, 20, 30, 40, 50, 100]
```

Les boucles

Les instructions for et while servent à répéter une ou plusieurs lignes de code.

for prend 3 paramètres encadrés dans des parenthèses, suivis des instructions à exécuter. Les paramètres sont :

- Initialisation : Ici on déclare la variable qui servira de compteur
- Condition : Tant que cette condition est vraie, la boucle s'exécutera
- Expression finale : Expression qui est exécutée à chaque fin de boucle

```
for(var i = 0; i < 5; i++){
console.log(i);
}
```

Dans l'ordre :

- Initialisation d'une variable i égale à 0
- Tant que i est inférieure à 5, on exécute la boucle
- A chaque fin de boucle, on incrémente i de 1

Ici, la boucle sera ainsi exécutée 5 fois , quand la variable i aura comme valeurs 0, 1, 2, 3 et 4.

À 5, la condition sera fausse et la boucle ne sera pas exécutée.

```
tableau = [5,4,3,2,1]; // tableau.length permettra de récupérer le
nombre d'éléments que le tableau contient

for(var i = 0; i < tableau.length; i++){ // i aura successivement les
valeurs 0,1,2,3,4. Il s'arrêtera à 5 vu que la condition `5 <
tableau.length` ne sera pas vérifiée
   console.log(tableau[i]); // Afficher la valeur du tableau
correspondante
}
```

Ici, je parcours un tableau et affiche toutes ses valeurs.

while (tant que) définit une liste d'instructions à exécuter tant qu'une condition est vraie.

```
var x = 0;

while(x < 20){
   x += 5;
}

console.log(x);
```

while ne prend qu'un seul paramètre : la condition à vérifier. Tant qu'elle est fausse, la boucle est réexécutée.

Ici, la boucle sera exécutée 5 fois et x aura comme valeur finale 20;

for...in permet de parcourir un objet.

```
frigo = {
   fruits : ["pommes", "poires"],
   legumes : ["tomates", "carottes"]
};

for(type in frigo){ // Parcourt chaque clé de l'objet. type aura "fruits" et "legumes" comme valeurs
   console.log(frigo[type]);
}
```