1. Introduction

Définition :

Le JavaScript est un langage de script orienté objet dynamiquement typé.

Langage de script : Nécessitant un interpréteur pour être lu. Dans le cas du JavaScript l’interpréteur est dans le navigateur

Orienté Objet : Les éléments du langage sont des objets. Une chaine de caractère est un ‘objet string’

Pas ou faiblement typé : Les variables contiennent uniquement des objets, de type différent mais toujours des objets. Une variable contenant une chaine de caractère ‘5’ peut être redéfini pour contenir le nombre 5.

Le JavaScript est principalement utilisé du coté client pour dynamiser et ajouter des interactions sur des pages HTML renvoyé par le serveur mais peut aussi être utilisé coté serveur avec node js.

Histoire

En 1995 Brendan Eich de chez netscape développa le langage initialement appelé liveScript destiné à être installé directement dans les navigateurs netscape. Le langage sera ensuite renommé JavaScript en hommage au Java. Microsoft développa l’année qui suit le langage JScript pour rester dans la course des navigateurs.

Toujours au cours de l’année 1996 Netscape envoyèrent une demande pour standardiser le langage European Computer Manufacturers Association (ECMA). Depuis le langage a beaucoup évoluer et la dernière version stable est l’ECMAScript 5 sortie en 2009. L’ECMAScript 6 est sortie en Mai 2015 et le 7 sortira sous peu car il a été développé en parallèle du 6. Bien que l’ECMAScript 6 soit déjà implémenter dans certains navigateurs, il ne marchera pas dans les anciens navigateurs. Son utilisation reste donc très limitée. Des solutions existent pour développer avec l’ECMAScript 6 qui est ensuite compilé en ECMAScript 5.

Pour la suite de ce cours je vais supposer que vous connaissez déjà le HTML et le CSS

Hello World !

Depuis la console de votre navigateur écrivez la ligne suivante :

alert(‘Hello World !) ;

2 Les bases

2.1 Les Variables

Une variable est un conteneur qui peut contenir à peu près tout. Un nombre, une chaine de caractère, un tableau… On se rappellera que les nombre, chaine de caractères… sont des objets.

Il existe deux façons de déclarer une variable en JS. Vous pouvez directement donner un nom à une variable et une valeur ou précéder le nom de la variable par var comme ceci :

maVariable = 5 ;

var maVariable = 5 ;

Même si les deux affectent une valeur de 5 à la variable maVariable, il y a une différence considérable entre les deux. En précédant le nom de la variable par le mot clé var la variable ne sera disponible que dans un ‘scope’ défini. Bien que le langage le permet, c’est une très mauvais pratique de déclarer une variable sans le précéder de var. On va donc toujours déclarer nos variables en les précédant par var. On y reviendra plus tard pour le scope mais pour l’instant retenez juste qu’il y a une différence entre les deux types de déclaration.

2.2 Les opérateurs

2.2.1 Les opérateurs arithmétiques

Addition +

Soustraction -

Multiplication \*

Division /

Modulo %

Les opérateurs sont les mêmes utilisé en maths sauf pour le modulo qui est un peu spéciale. Il retourne le reste d’une division. Par exemple :

10 % 3 = 1

On verra plus tard qu’il y a aussi d’autres façons de les utiliser.

2.2.2 Les opérateurs de comparaisons

== égal à

!= différent de

=== contenu et type égal à

!== contenu ou type différent de

> supérieur à

>= supérieur ou égal à

< inférieur à

<= inférieur ou égal à

Les noms sont suffisamment indicateurs je pense.

Les opérateurs === et !== vérifie le contenu et le type. Le type c’est le type d’objet qui est évaluer, ainsi :

‘5’ == 5 => Vrai

‘5’ === 5 => Faux car un string n’est pas de même type nombre.

2.2.3

Les opérateurs logiques

&& ET

|| OU

! NON

Les opérateurs logiques retournent toujours un booléen.

2.3 Les Conditions

Quand vous développez une application vous avez besoins par moment de savoir si une variable existe ou si elle contient quelque chose ou encore si elle existe et qu’elle contient bien une valeur…. En fait vous aller chercher à tester plusieurs cas possible avant de faire une action.

Par exemple, sur un site de vente de eBooks en ligne, vous ne laisserez l’utilisateur télécharger son eBook que lorsque vous avez vérifié **SI** vous avez les cordonnées de la personne et **SI** le paiement est bien validé **OU SI** le prix est gratuit **ET** que vous n’avez pas besoins des cordonnées de la personne avant de validé votre processus.

C’est là que rentre en jeux les conditions. Ils vont justement nous permettre de dire à notre application si le prix n’est pas gratuit il faut que j’ai reçu le paiement avant de laisser l’utilisateur télécharger son livre.

2.3.1 Le if (**SI**) :

If(condition === vrai) {

// Je fait quelque chose ou plusieurs choses

execution;

}

Vous pouvez aussi combiner plusieurs opérateurs dans le même if comme ceci :

if(condition1 === vrai && condition2 === false || condition3 === vrai) {

// La seulement j’exécute mon processus

}

2.3.2 Le else (**SINON**)

Dans certains cas vous avez besoin de **SI**  ma condition est vrai exécute le processus **SINON** fait autre chose.

if(condition === vrai) {

// J’exécute mon processus

} else {

// Je retourne une alerte

alert(‘la condition est fausse) ;

}

Comme vous l’aurez deviné, vous ne pouvez pas avoir un else qui n’est pas précéder par un if. Sinon l’interpréteur lancera une erreur.

2.3.3 Le else if (**SINON SI**)

Il arrivera aussi des moments où vous devrez faire des choses différentes en fonctions des valeurs contenus dans vos variables. Par exemple, sur un site avec du contenu adulte, vous voudrez vérifier si l’utilisateur est majeur et s’il n’est pas trop vieux non plus. Dépendant de son âge vous l’afficherez un message différent. C’est justement ce que vous permets de faire le else if.

If(age < 18) {

alert(‘Tu est trop jeune’) ;

} else if (age > 31) {

alert(‘Tu est trop vieux’) ;

} else {

alert(‘Amuse toi bien !’) ;

}

Vous pouvez avoir autant de else if que vous voulez dans vos conditions. Mais le problème c’est que ce n’est pas très pratique à écrire et encore moins à lire.

2.3.4 Le switch

Le switch retourne un booléen comme toutes les conditions.

Comme un bon exemple est mieux qu’un long discours…

Vous avez fait une application qui donne des points de 0 à 3. Vous voulez afficher un message différent selon le nombre de points de l’utilisateur. Si l’utilisateur a gagné 0 points vous aller afficher un message pour lui dire qu’il est nul, 1 un message pour lui dire que ça va même si un peu limite, 2 pour lui dire que c’est bien 3 pour lui dire que le jeu était trop facile.

La syntaxe :

var nombreDePoints = 0 ;

switch(nombreDePoints) {

case 0 :

alert('Trop est nul. Merci de ne pas revenir ici') ;

break ;

case 1:

alert('Un peu limite.') ;

break ;

case 2:

alert('Très bien') ;

break ;

case 3:

alert('Tu as eu de la chance de tomber sur un jeu trop facile') ;

break ;

default :

alert('On croirai que j\'ai fait confiance à l\'utilisateur') ;

}

Vous avez surement remarqué les ‘break’ et le ‘default’. Le break dit à votre script de sortir de la condition quand il est rentré dans un ‘case’. Et le default est la valeur par défaut à afficher si la condition n’est pas rentrée dans une case. Par exemple si vous y attribuer une valeur de ‘chaine de caractère’ à nombreDePoints.

Le switch est comme vous pouvez le voir bien plus facile à lire et plus intuitif. Mais il est souvent mal utilisé. Si vous avez des doutes et que vous hésitez entre un if et un switch, il est fort probable qu’un if sera mieux adapté.

2.3.5

Les Ternaires

Les ternaires sont des conditions très rapide à écrire mais qui sont difficile à lire pour ceux qui ne sont pas habituer. De manière générale on évitera de les utiliser même s’ils sont très pratiques parce que votre application évoluera et sera maintenu par d’autres développeurs qui ne sont pas forcément habitué à leur lecture.

var condition = true ;

var process = '';

process = condition ? 'OK !' : 'Failed !' ;

La même chose écrite avec un if/else:

if(condition) {

process = 'OK!';

} else {

process = 'Failed !';

}

2.4 Ou mettre le code JavaScript ?

Le JavaScript peut se trouver directement dans votre fichier HTML ou dans un fichier séparé avec une extension .js

Pour écrire du JavaScript dans votre fichier HTML vous devez le placer dans des balises script comme ceci :

<script></script>

Pour écrire du JavaScript dans un fichier .js pour bien architecturer votre projet et séparé le JavaScript de votre HTML vous devez renseigner dans des balises script vide le chemin depuis votre fichier HTML à votre fichier .js avec l’attribut src comme ceci :

// En supposant que mon fichier JavaScript se trouve dans un dossier js à la racine de mon projet et que mon fichier HTML se trouve aussi à la racine de mon projet

<script src="js/monfichier.js"></script>

TP 1

Créer un répertoire que vous nommerez JavaScript dans lequel vous allez ajouter un nouveau fichier index.html avec le marquage minimum pour être valide. Vous pouvez aussi cloner/télécharger le dépôt suivant <https://github.com/iknsa-formation/JavaScript>.

Dans des balises script de votre fichier index.html vous allez afficher le service approprié selon le numéro que vous aurez en variable. Le message ‘La police’ => 17, ‘Les pompiers’ => 18, ‘Numéro Vert’ => 0800 et ‘Numéro mobile’ => 06.

2.5 Les Boucles

2.5.1 L’incrémentation et la décrémentation

Il existe en JavaScript comme dans beaucoup d’autres langages une syntaxe simple et facile qui permet facilement d’incrémenter ou décrémenter une valeur.

Par exemple :

var maVariable = 5 ;

maVariable = maVariable + 1 // Ma variable vaut maintenant 6

maVariable = maVariable - 1 // Ma variable vaut maintenant 5

La syntaxe raccourcie :

Var maVariable = 5 ;

maVariable++ // Ma variable vaut maintenant 6

maVariable-- // Ma variable vaut maintenant 5

2.5.2 Le while(TANT QUE)

Les boucles nous permettent de répéter un code tant qu’une condition est vraie ou qu’on dise à la boucle de s’arrêter. La boucle arrêtera son exécution aussitôt que la condition n’est plus respectée.

La syntaxe :

while(vrai) {

alert(‘j\’aime le JavaScript’) ;

alert(‘mais j\’aime pas le Java’) ;

}

Exemple :

var index = 0;

while(index < 3) {

index++; // Incrémentation

alert('j\'aime le JavaScript') ;

}

Dans l’exemple précédent, la variable index est déclaré avec une valeur de 0. A chaque passage dans la boucle elle est incrémenter et affiche un alert. Tant que la valeur de index est inférieur à 3 un alert s’affichera à chaque passage dans la boucle. La fonction alert sera donc exécuter 3 fois.

2.5.3 Le do while (FAIT TANT QUE)

Exemple:

var index = 0;

do {

index++; // Incrémentation

alert('j\'aime le JavaScript') ;

} while(index < 3) ;

La différence entre le while et le do while est un peu subtile. Dans la boucle while si vous déclarez la variable index avec une valeur supérieur à 3 la boucle ne sera jamais exécuter alors que avec la boucle do while la boucle sera exécutée au moins une fois étant donné que la condition n’est vérifiée qu’à la fin de la boucle.

2.5.4 La boucle for

Exemple

for(var index=0; index <3; index++) {

alert('J\'aime le JavaScript');

}

Similaire à la boucle while, toute la partie ‘logique’ de la boucle se font à un endroit ce qui rend le code plus facile à lire.

2.6 Les Fonctions

Une fonction c’est un bloc de code qu’on pourra exécuter à la demande. Ce sont un peu comme des variables. Ce sont des conteneurs.

Contrairement aux variables ils pourront contenir plusieurs choses et on pourra même lui dire d’exécuter un code différemment avec un ou plusieurs paramètres.

Passons directement à un exemple.

function afficheHelloWorld()

{

var hello = 'hello world';

alert(hello);

}

Détaillons un peu ce bout de code. Tout d’abord pour créer une fonction il faut le mot clé function, le nom de la fonction, des parenthèses et des accolades.

On verra par la suite que le JavaScript est plus flexible que ça mais pour le moment on s’en tiendra à cette syntaxe.

A l’exécution de ce code dans la console de votre navigateur, vous voyez… rien !

C’est tout à fait normal.

Comme mentionné un peu plus haut, la fonction s’exécute à la demande. Il vous faut donc demander à la fonction de s’exécuter. Pour ce faire rien de plus simple, il vous suffit d’exécuter le nom de la fonction suivi de parenthèses comme ceci :

afficheHelloWorld();

Et maintenant a chaque fois que demander la **fonction**, la variable hello est instancié avec la valeur ‘hello world’ la **fonction** alert est exécuter.

Heuuu… Comment ça ? Quand j’exécute ma fonction, elle exécute une fonction ??

Hé oui !

alert();

Alert est une fonction dite native du JavaScript, ça veut dire que cette fonction existe déjà dans le langage.

Vous vous dites surement que cette fonction exécutera toujours la même chose à chaque fois et qu’il n’y a pas moyen de changer cette fonction et vous avez raison. Mais vous pouvez aussi lui passer des paramètres…. Modifions notre fonction pour illustrer ce fameux paramètre :

function afficheHelloWorld(texte)

{

alert(texte);

}

afficheHelloWorld('Hello world');

afficheHelloWorld('Hello Khalid');

Désormais à chaque appel à la fonction on lui donne le texte à afficher.

Bon vous me direz que notre super fonction avec paramètre ne sers pas à grand-chose et en plus c’est plus long à écrire alors qu’on aurait simplement pu faire des alert directement avec les chaines de caractères…. Mais une fois encore c’était pour l’exemple…

2.7 Les Tableaux

Il existe comme dans beaucoup d’autres langages on a la possibilité en JavaScript d’avoir des tableaux. Un tableau nous permet de ‘ranger’ certaines choses, comme par exemple une liste de prénoms.

La syntaxe pour créer un tableau :

var monTableau = [ ];

Un tableau contenant le prénom de personnes est écrite comme ceci :

var prenoms = ['Khalid', 'Noémie', 'Robert', 'Jean'];

Il existe des fonctions natives du JavaScript qui nous permettent de parcourir ces tableaux pour ajouter, récupérer, changer ou enlever des valeurs ou des clés entre autre. On utilisera souvent les boucles for ou while pour les parcourir.

Par exemple, prenons le tableau prenoms. Si je veux changer la valeur Robert, il faut que je parcoure le tableau et quand je rencontre la valeur Robert, je remplace cette valeur par autre chose comme Dujardin.

Pour notre boucle for ou while, il nous faudra une condition pour sortir de la boucle. Cette condition est la taille du tableau, càd le nombre d’éléments qu’il contient. Dans notre cas on voit bien que notre tableau contient 4 éléments. Mais ce tableau pourrait changer au cours de sa vie et si vous sortez de votre boucle au bout de 4 éléments, votre code ne sera plus bon. Donc pour récupérer la taille du tableau nous utiliserons ceci : prenoms.length

Deuxième problème, comment récupérer une valeur précise d’un tableau et plus encore, comment la changer !!! Essayez-le !

Correction:

prenoms = ['Khalid', 'Noémie', 'Robert', 'Jean'];

for(i=0; i < prenoms.length; i++) {

if(prenoms[i] === 'Robert') {

prenoms[i] = 'Dujardin';

}

alert(prenoms[i]);

}

Pour récupérer une valeur dans un tableau on utilisera le nom du tableau suivi de crochet contenant la position de la valeur que nous voulons récupérer.

Par exemple, prenoms[**2**] retournera la valeur Robert. Et non je ne me suis pas trompé. La valeur du tableau à la position **2** est bien Robert et non Noémie. C’est parce que les index en informatique, commence généralement par 0. Donc notre tableau prenoms contiens les indexes 0, 1, 2 et 3.

Pour affecter une valeur à un indexe dans le tableau comme dans notre exemple on fera tout simplement prenoms[2] = ‘Dujardin’ et la valeur à la position 2 passera désormais à Dujardin.

Tout comme length il existe une multitude de fonctions sont native qui vous permettront d’interagir avec les tableaux.

Une petite note avant de changer de sujet, nous avons vu que les tableaux peuvent contenir des valeurs. Mais ils peuvent aussi contenir des variables, des fonctions et même des tableaux. Mais ça on y reviendra plus tard.

2.8 Les Objets

Vous vous rappelez au début je vous avais dit que le JavaScript est un langage orienté objet. Ce n’est pas tout à fait exact. Ce n’est pas un langage orienté objet comme le Java ou le C++. Le JavaScript est un langage d’objet par prototypage.

Un objet possède une structure spécifique séparé en 3 parties. Le constructeur, les propriétés et les méthodes. Le constructeur est un bout de code qui est exécuté à chaque fois qu’on utilise un nouvel objet. Les propriétés sont les variables au sein d’un objet et les méthodes sont les fonctions contenues dans l’objet qu’on utilisera pour modifier l’objet.

Le langage dispose d’objets natifs qui ont donc déjà leurs propriétés et méthodes. Ces objets nous les avons déjà utilisés.

Exemple :

var chaine = 'Ceci est un objet string';

alert(chaine.length);

alert(chaine.toUpperCase());

Comme vous le voyez j’utilise ici la propriété length de l’objet string et sa méthode toUpperCase() qui transformera votre chaine de caractère en majuscule.

Pour créer un objet on fera :

var personnes = {

nom: 'Khalid',

age: '29',

sex: 'Oui',

};

La syntaxe ressemble un peu à la syntaxe de tableau mais il y a quand même quelques différences.

Pour récupérer la valeur du nom on fera ainsi :

var personnes = {

nom: 'Khalid',

age: '29',

sex: 'Oui',

};

alert(personnes.nom);

Et pour affecter une valeur on fera :

personnes.nom = 'Alfred';

On verra plus tard que les objets sont bien plus puissant que ça notamment la facilité d’ajouter des propriétés, car on n’oublie pas que le JavaScript est un langage d’objet par prototypage. Cela signifie que l’objet personnes que nous avons créé est un prototype.

2.9 Les alertes à toute instant pour développer c’est pénible !

Pour finir ce chapitre d’introduction, nous allons nous intéresser à bien coder et arrêter de faire des alert() constamment.

Pourquoi avons-nous besoin de bien coder ?

Lorsque vous faites une application ou un simple script, le code sera très probablement repris par quelqu’un d’autre.

La personne ne connait pas votre logique car comme il existe trop de façon de faire une même chose chaque développeur fait les choses différemment. Certes s’il est un bon développeur il finira par comprendre ce que vous avez voulu faire mais ça lui prendra des heures voir des semaines dépendant de la complexité de votre code.

Même si vous faite du code personnelle qui ne sera pas utilisé par d’autres développeurs (ça vous ne pouvait en être sure) quand vous reprenez un code plusieurs semaines après l’avoir fait vous ne serez plus dans la même logique et vous n’arriverai plus à comprendre ce que vous avez fait.

Le JavaScript est space insensitive. Ça veut dire qu’on peut écrire tout notre code sur une seule ligne. D’ailleurs lorsque nous allons mettre en production du code JavaScript, on va d’abord le minifier c’est-à-dire, on va enlever les commentaires enlever toutes les espace, tabulations, retour chariot et renommer nos variables par ces noms qui prennent moins de place qui donnera un résultat incompréhensible par l’homme. N’ayez crainte la minification se fait avec les logiciel. Nous verrons ça en fin de cours.

Tout ça pour vous dire qu’avoir un code facilement compréhensible et facile à lire.

Exemple

sI vOuS eSsay

eZ De lIrE Ce

Tte pHraSe VoUs allEZ vOiR qUe c’Est p

aS SuPeR siMplE

Mais c’est sûr que vous seriez arrivé à la lire. 

Voilà l’importance d’être compréhensible et facile à lire. Ce sera exactement la même chose dans votre code.

Il n’existe pas de règles/normes officiel de codage en JavaScript comme le PSR pour le PHP. Chaque développeur peut faire comme il le souhaite mais depuis quelques temps une certaine structure s’est mis en place dans la communauté et est considéré comme une bonne pratique. C’est ce que nous allons voir.

2.9.1 Tout d’abord les commentaires.

Dans un bon code rien n’est plus important que les commentaires. Pour une fonction avec paramètres et valeur de retour, votre commentaire doit inclure, le descriptif de la fonction, le type des paramètres, le paramètre et une description et le type de la valeur de retour et un descriptif de la valeur de retour dans cet ordre. En plus de ça vous devez laisser une espace entre la description et les paramètres, et les paramètres et le retour. Cette convention facilite la lecture.

Exemple de commentaire sur une fonction avec un paramètre et un retour.

/\*\*

\* [commentaire description]

\*

\* @param {string} comment [description]

\*

\* @return {bool} [description]

\*/

function commentaire(comment)

{

// Je fait mes affaires

return true;

}

Ce type de commentaire est assez spécial, il commence par un slash suivi de 2 astérix et reprend un astérix sur chaque ligne pour finir par astérix slash. C’est souvent utilisé pour des annotations. Les annotations sont par exemple @param mais il y en a d’autres comme @TODO. Il existe des librairies qui vont chercher vos annotations pour faire d’autres choses. Par exemple, doctrine qui est un ORM utilise les annotations pour mapper votre model de données à votre base de données.

Vous pouvez installer dans votre sublime text (si vous avez le package control) le plugin docblokr qui vous facilite l’écriture des commentaires.

2.9.2 L’indentation et les accolades.

Pour l’indentation de votre code, on utilisera des espaces et non des tabulations. Vous pouvez configurer votre IDE pour transformer vos tabulations automatiquement en espaces. Le nombre d’espace varie encore pas mal. Certains utilisent 4 espaces et d’autres 2 espaces. Il y a encore quelques années tout étaient en 4 espaces mais depuis comme le code devenait plus complexe on se retrouvait avec des scroll verticale ce qui a donné vie aux 2 espaces. Bien que personnellement j’ai une préférence pour les 4 espaces car je le trouve bien plus lisible, on utilisera 2 espaces qui s’est imposé dans le monde du JavaScript.

Très important, n’indentez pas de 4 espaces au début et changer pour 2 espaces au milieu. Si vous avez commencé par 2 espaces, maintenez les 2 espaces dans toute votre application.

On changera notre indentation à chaque fois qu’on changera de complexité cyclomatique.

Par exemple notre code ressemblera à ça :

var age = 0;

function personnes()

{

var bouche;

function parler(bouche) {

alert(bouche);

}

}

Ici j’utilise 4 espaces d’indentation mais j’aurais tout aussi bien pu utiliser 2 espaces d’indentations.

L’accolade ouvrant de la fonction personnes se trouve à la ligne suivante alors que l’accolade de la fonction parlée lui est séparée d’une espace seulement.

On remarquera aussi que je laisse une ligne blanche entre chaque partie ‘logique’ de mon code. Un espace entre la déclaration des variables et les fonctions.

2.9.3 Console log

Depuis le début nous avons utilisé la fonction alert() pour afficher les valeurs d’une variable ou le retour d’une fonction. Mais il existe une fonction qui est particulièrement faite pour justement débugger notre code. C’est la fonction console.log(). Contrairement à la fonction alert() notre code n’est pas interrompu et on n’est pas obligé de fermer un popup.

Exemple :

var age = 30;

console.log(age);

C’est tout de même plus agréable pour débugger.

De même que dans l’exemple précédent l’exemple suivant marchera tout aussi bien.

var age = 30;

function reponse(age)

{

if(age > 30) {

console.log('t\'es vieux');

} else {

console.log('c\'est bon. Tu peux y accéder:');

}

}

reponse();

2.10 Rappel

Nous avons vu dans ce chapitre les bases du JavaScript, comment déclarer les variables, faire des boucles sur des tableaux ou encore faire des fonctions et les utiliser. Mais le JavaScript nous permets de faire bien plus que de la programmation algorithmique, elle nous permet entre autre de modifier le DOM, et c’est ce que nous allons voir dans le chapitre suivant.

Mais avant un petit TP de validation de question.

Vous avez un tableau contenant des objets de questions et réponses.

var questions = [

{

ques: 'Est-ce que tu aime le JavaScript ?',

ans : 'Oui'

},

{

ques: 'Le JavaScript est une évolution du Java ?',

ans : 'Non'

},

{

ques: 'Le JavaScript est proche du Java ?',

ans : 'Non'

},

{

ques: 'Le JavaScript est un langage d\'objet par prototypage ?',

ans : 'Oui'

},

{

ques: 'Le JavaScript est souvent utilisé coté client ?',

ans : 'Oui'

},

{

ques: 'Le JavaScript peut être utilisé coté client et serveur ?',

ans : 'Oui'

}

];

Vous devez afficher un popup avec la question ou l’utilisateur peut donner la réponse. Si la réponse est bonne, on log un message ‘Bonne réponse pour la question’ et la question dans la console et on passe à la prochaine question sinon on log dans la console ‘mauvaise réponse pour la question’ et la question.

Hint : Vous devriez vous intéresser à la fonction prompt()

Correction : git checkout –b questions\_reponses

Prenez les choses en main : Vous pouvez améliorer notre petite application de question réponses en rajoutant un attribut ‘repondu/answered’ à chaque objet et faire en sorte que les questions passe en désordre. Vous retiendrez chaque réponse en rajoutant un attribut ‘goodAnswer’ ou ‘wrongAnswer’ et vous sortirez de la boucle une fois toutes les questions répondues. A la fin vous compterez les bonnes réponses et vous afficherez les résultats. C’est un peu les concepts des examens en ligne. De cette façon les candidats peuvent difficilement entre eux sur les autres étant donné qu’ils n’ont pas les mêmes questions au même moment.

3.0 La Manipulation du DOM

Comme mentionné précédemment, nous pouvons aussi accéder et modifier le DOM (Document Object Model). Pour ce faire plusieurs méthodes existent au sein même du langage. Toutefois à faire attention, certaines méthodes ne marcheront pas sur tous les navigateurs, notamment les vieux navigateurs, très particulièrement Internet Explorer.

Bien que tous les navigateurs n’interprètent pas le JavaScript (comme le JavaScript est interpréter par le navigateur) de la même manière, il y a assez peu de différences entre les navigateurs. Vous devez donc vérifier si les méthodes que vous utilisé sont compatible avec votre scope navigateur.

3.1 Accéder aux éléments du DOM

3.1.1 Le getElementById()

Pour accéder aux éléments du DOM il existe entre autre la méthode getElementById(‘id’) qui prend en paramètre l’id que vous recherchez. Comme son nom l’indique, il permet de récupérer un ‘id’.

Exemple :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Course - IKNSA</title>

</head>

<div id="test"></div>

<body>

<!-- Page Content -->

<script>

console.log(document.getElementById('test'));

</script>

</body>

</html>

Cette méthode vous retourne comme vous pouvez le voir dans votre console, le div contenant l’id spécifié. Vous remarquerez que j’ai préfixé cette méthode du mot clé ‘document’. Ceci est parce que document représente ici le nœud de premier niveau de votre DOM. C’est la balise HTML, votre page web.

Il existe un autre mot clé très important, c’est le mot clé ‘window’ qui lui représente la fenêtre du navigateur.

3.1.2 getElementsByTagName et getElementsByClassName

Ces deux méthodes vous permettent de récupérer les éléments avec les même balises et classe respectivement. Vous noterez que ici il y a un ‘s’ à ‘Element’. C’est parce que contrairement à la méthode précédente qui vous retourne un objet, ces deux méthode vous retourne un tableau des objets recherché.

Exemple :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Course - IKNSA</title>

</head>

<div class="test"></div>

<div class="test"></div>

<div class="test"></div>

<body>

<!-- Page Content -->

<script>

console.log(document.getElementsByTagName('div'));

console.log(document.getElementsByClassName('test'));

</script>

</body>

</html>

Depuis la console de votre navigateur vous pouvez aussi voir les nombreuses méthodes et propriété auxquelles vous pouvez y accéder. Nous allons en voir quelques-uns pour voir comment ils fonctionnent.

3.1.3 querySelector et querySelectorAll

Ces deux méthodes prennent tous le deux en paramètre une chaine de caractère semblable à votre sélecteur CSS.

Exemple :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Course - IKNSA</title>

</head>

<body>

<!-- Page Content -->

<div id="test">

<p class="paragraph">

<a href="#" class="lien">ceci est un lien</a>

</p>

</div>

<script>

console.log(document.querySelector('#test .paragraph .lien'));

</script>

</body>

</html>

3.2 innerHTML

Nous savons maintenant récupérer un objet du dom mais il serait encore mieux de pouvoir le modifier. C’est ce que nous pouvons faire avec innerHTML.

Exemple :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Course - IKNSA</title>

</head>

<div id="test">

<p>

Lorem ipsum dolor sit amet.

</p>

</div>

<body>

<!-- Page Content -->

<script>

console.log(document.getElementById('test').innerHTML);

document.getElementById('test').innerHTML = '<a href="#"><strong>Modification avec le JavaScript</strong></a>';

console.log(document.getElementById('test').innerHTML);

</script>

</body>

</html>

Analysons l’exemple précédent. Nous avons une div avec l’id test, une balise p et du texte dedans. Notre première ligne de code JavaScript affiche le contenu HTML de notre div dans la console alors que la deuxième ligne affecte une nouvelle valeur HTML à notre div test.

Sachez que nous pouvons aussi accéder aux attributs des éléments du DOM avec getAttributes ou rajouter/enlever des classes sur un élément du DOM.

Exemple :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Course - IKNSA</title>

</head>

<body>

<!-- Page Content -->

<div id="test" class="alfred"></div>

<script>

console.log(document.getElementById('test').classList);

document.getElementById('test').classList.add('nouvelle');

console.log(document.getElementById('test').classList);

document.getElementById('test').classList.remove('alfred');

console.log(document.getElementById('test').classList);

</script>

</body>

</html>

Comme nous pouvons le voir ici, au début nous avons bien une class ‘alfred’ dans notre div. Nous rajoutons ensuite une classe ‘nouvelle’ pour finalement enlever la classe ‘alfred’. Comme tout est loggé en console nous pouvons facilement suivre le processus.