Dièse Dev #1 - Introduction à JavaScript

Introduction

Cette première formation est dédiée à l'apprentissage du langage JavaScript.

Javascript, c'est quoi ? JavaScript est au coeur du web. Normalement utilisé côté client, c'est un langage de programmation de script principalement employé dans les pages web interactives. Avec l'arrivé de Node.js, il est maintenant utilisé de plus en plus côté serveur, ce qui fait de lui un langage complet et essentiel pour le web.

Pourquoi apprendre Javascript ? React Native est un framework JavaScript. C'est donc en JavaScript que nous développerons nos applications mobiles.

> C'est un langage orienté objet.
>
> -- *fr.wikipedia.org/wiki/JavaScript*

Pour commencer

Pour lancer un script JavaScript, on utilisera un navigateur classique - Firefox, Chrome, Edge, Opéra, Safari - juste pour cette formation.

1. Récupérer `formation_1.html` et `formation_1.js`
2. Ouvrir `formation_1.html` dans un navigateur et `formation_1.js` dans un éditeur de code.

Afficher dans la console

```
// console.log("Ma chaîne de caractères")
console.log('👋 Hello World!')
>> '👋 Hello World!'
```

Pour voir la console dans le navigateur, il faut ouvrir le panneau de développement, souvent en appuyant sur `F12`, puis en sélectionnant un onglet "Console" ou semblable.

Déclarer une variable ou une constante

> Le mot-clé **const** permet de déclarer une constante accessible uniquement en lecture.

On ne peux pas modifier une constante ! 😅

```
const a = 10
a = 12 // Erreur
const b = a + 2
// b = 12
```

> Pour déclarer une variable, deux mots clés existent : **var** et **let**.

Ces deux mot-clés sont semblables, mais il existe quelques subtilités.

```
let a = 5
a = 10
console.log(a)
>> 10

var b = 10
console.log(b)
>> 10

b = 42
console.log(b)
>> 42
```

Les premières subtilités entre **var** et **let**.

```
// ✅ Fonctionne ✅

/* On peut utiliser une variable avant de la déclarer ou même
 * sans la déclarer. Cependant, il est préférable de prendre
 * la bonne habitude de la déclarer avant de l'utiliser.
 */
prenom = "Jack"
var prenom

var age = 16
var age = 21 // On peut redéclarer une variable avec var

// ❌ Ne fonctionne pas ❌

// Il faut déclarer la variable avant de l'utiliser !
prenom = "Jack" // ❌
let prenom

let age = 16
let age = 21 // ❌ On ne redéclare pas les variables avec let
```

À l'origine, on n'avait que `var` pour déclarer une variable. Puis, la spécification **ES6** a introduit `let` et `const` pour gagner en finesse. Outre ce qui prècède, `let` et `const` n'existent que dans le bloc où ils sont définit, au contraire de `var` qui est global.

```
var i = 3

for (var i = 0; i < 5; i++) {
    //
```

```
}
// Ici, i = 5. On n'a plus i = 3.


if (i === 5) {
    let j = "Hello"
    const h = " world!"
    console.log(j + h) // Fonctionne ✅
}
console.log(j + h) // Ne fonctionne pas ❌
```

Les conditions

> JavaScript possède des opérateurs de comparaisons stricts et des opérateurs de comparaisons qui effectuent des conversions. Une comparaison strict (ex. : ===) ne sera vraie que si les deux opérandes sont du même type. La comparaison d'égalité faible (==) convertira les deux opérandes en un même type avant d'effectuer la comparaison. Pour les comparaisons relationnelles (ex. : <=), les opérandes sont tout d'abord converties en valeurs, puis en valeurs du même type, enfin la comparaison est effectuée.
>
> *https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Op%C3%A9rateurs_de_comparaison*

Égalité simple (==)

```
1 == 1     ✅
1 == '1'   ✅
0 == false ✅
1 == 2     ❌
```

Attention à la manière dont Javascript s'occupe de convertir les types !

Inégalité simple (!=)

```
1 != 2;      ✅
1 != "1";    ❌
1 != true;   ❌
```

Attention à la manière dont Javascript s'occupe de convertir les types !

Égalité stricte (===)

```
1 === 1   ✅
1 === '1' ❌
```

Inégalité stricte (!==)

```
3 !== '3' ✅
4 !== 3   ✅
```

If..else

> L'instruction if exécute une instruction si une condition donnée est vraie ou équivalente à vrai. Si la condition n'est pas vérifiée, il est possible d'utiliser une autre instruction.
>
> *https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/if...else*

```
if (condition) {
    ...
} else {
    ...
}
```

Attention à ne pas confondre les valeurs booléennes « primitives » true et false avec les valeurs true et false d'un objet Boolean. Toute valeur qui n'est pas false, undefined, null, 0, -0, NaN ou la chaîne vide (""), et tout objet, y compris un objet Boolean dont la valeur est false, seront évalués à true lors d'une instruction conditionnelle if. Ainsi :

```
var b = new Boolean(false);
if (b) // la condition sera évaluée à true car b est définie mais il est
différent de false

if(a) //la condition sera évaluée à false car a est undefined
```

Exemple :

```
var nombreSoda = 5
if(nombreSoda){
    console.log("la variable nombreSoda est définie")
}

if(nombreSoda > 0){
    console.log('Il reste du 🍷 ')
}else{
    console.log("Il n'y a plus de 🍷 ")
}

>>"la variable nombreSoda est définie"
>>"Il reste du 🍷 "
```

> Pour concaténer des chaines de caractères entre elles ou avec des variables, on utilise l'opérateur **+**

```
const prixSoda = 0.99

console.log('🥤 : $'+prixSoda)
>>'🥤 : $0.99'
```

Les fonctions

On peut déclarer des fonctions de différentes manières en JavaScript.

```
// Méthode classique
function carré(a){
    return a * a
}

// Via une fonction anonyme
const carré = function (a){
    return a * a
};

// Via une fonction fléchée introduite par ES6
const carré = (a) => {
    return a * a
}
```

> À noter : on ne peut pas accéder à des variables définies dans une fonction en dehors de la fonction.

```
// Les variables suivantes sont globales
var num1 = 20,
    num2 = 3,
    nom = "Licorne";

// Cette fonction est définie dans la portée globale
function multiplier() {
  return num1 * num2;
}

multiplier(); // Renvoie 60

// Un exemple de fonction imbriquée
function getScore () {
    var a = 12
    var num1 = 2,
        num2 = 3;

    function ajoute() {
        return nom + " a marquée " + (num1 + num2);
```

```
    }

    return ajoute();
}

getScore(); // Renvoie "Licorne a marquée 5"

console.log(a) // ❌ Erreur : on ne pas avoir accès à une variable en
dehors de la fonction où elle est déclaré.
```

Les objets

On peut créer un objet simplement de la manière suivante :

```
var menu = {
    burger: 'Bacon CheeseBurger',
    accompagnement: 'Frites',
    boisson: 'Coca Vanille Cerise'
}

console.log('Le client a commandé un '+menu.burger+' accompagné de
'+menu.accompagnement+' et il sirotera un '+ menu.boisson)

>> "Le client a commandé un Bacon CheeseBurger accompagné de Frites et il
sirotera un Coca Vanille Cerise"
```

On peut également modifier les objets :

```
menu.boisson = "Sprite"
console.log(menu)
>> {
    burger:'Bacon CheeseBurger',
    accompagnement: 'Frites',
    boisson:'Sprite'
}
```

Un objet peut aussi avoir des méthodes permettant de manipuler ses données.

```
var menu = {
    burger:'Bacon CheeseBurger',
    accompagnement: 'Frites',
    boisson:'Coca Vanille Cerise',
    afficher_burger :function (){
        console.log('Burger: '+this.burger)
    },
    modifier_burger : function(newBurger){
      this.burger = newBurger
```

```
      console.log('Burger modifié')
    }
}

menu.afficher_burger()
menu.modifier_burger('Big Mac')
menu.afficher_burger()

>> Burger: Bacon CheeseBurger
>> Burger modifié
>> Burger: Big Mac
```

On peut faire la concaténation de deux objets

```
menu2={
  burger:'Big Mac',
  accompagnement:'Oignons rings',
  boisson:'Sprite'
}

var concat = {...menu, ...menu2}

console.log(concat)
>>{
burger:"Big Mac",
accompagnement:"Oignons rings",
boisson:"Sprite",
afficher_burger:f afficher_burger {...},
modifier_burger:f modifier_burger {...}
}
```

Cette méthode permet de concaténé les objets, s'ils ont des attributs en commun, seul celui du premier objet est gardé.

```
const a = {
    param1 : 1
    param2 : 2
}

const b = {
    param3 : 3,
    param4: 4
}

console.log({...a,...b})
>>{
param1:1,
param2:2,
param3:3,
```

```
param4:4
}
```

Les listes (Array)

Déclaration d'une liste

```
let L = [1,2,3]
```

Comme en Python, le première élément d'une liste est indexé par 0.

```
console.log(L[0])
>> 1

console.log(L[L.length - 1]) // Dernier élément de la liste
>> 3
```

Boucler sur une liste avec `forEach`.

```
L.forEach((item, index) => {
    console.log(index + ":" + item)
})

>> 0:1
   1:2
   2:3
```

Méthode map

```
const noms = ['Arthur','Bernard','Michelle']

const bonjourNoms = noms.map(elem => 'Bonjour '+elem)

console.log(bonjourNoms)
>> [
"Bonjour Arthur",
"Bonjour Bernard",
"Bonjour Michelle"
]

const prix = [1,9.99,19.99]

//On veut baisser tous les prix de 10%

const solde = prix.map(prix => Number((prix-prix*0.1).toFixed(2))) // On
```

```
arrondue au centième

console.log(solde)
>> [
0.9,
8.99,
17.99
]
```

Plein d'autres méthodes intéressantes sur les tableaux :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array

Les promesses (promise)

> Une promesse est un objet (Promise) qui représente la complétion ou l'échec d'une opération asynchrone. La plupart du temps, on « consomme » des promesses et c'est donc ce que nous verrons dans la première partie de ce guide pour ensuite expliquer comment les créer.
>
> En résumé, une promesse est un objet qui est renvoyé et auquel on attache des callbacks plutôt que de passer des callbacks à une fonction. Ainsi, au lieu d'avoir une fonction qui prend deux callbacks en arguments
>
> --https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses

Un exemple pour illustrer :

- Imagine toi enfant si ta mère t'avait promis de t'acheter un nouveau téléphone la semaine prochaine
- Tu ne sais pas encore si maman allait t'acheter ce téléphone avant la semaine prochaine : elle peut te l'acheter si tu es sage mais elle peut ausse refuser si elle n'est pas contente...
- C'est une promesse et en JavaScript, une promesse a 3 états :
  - `pending` : Tu ne sais pas si tu vas avoir ce téléphone
  - `FullFilled` : Maman est contente, elle vas t'acheter un nouveau téléphone
  - `Rejected` : Maman n'est pas contente, elle ne t'achetera pas ce téléphone

```
var isMomHappy = false;

// Promise
var willIGetNewPhone = new Promise(
    function (resolve, reject) {
        if (isMomHappy) {
            var phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone); // fulfilled
        } else {
            var reason = new Error('mom is not happy');
            reject(reason); // reject
        }
```

```
    }
);
```

Maintenant, voyons comment appeller cette promesse.

```
var askMom = function () {
    willIGetNewPhone
        .then(function (fulfilled) {
            // yay, you got a new phone
            console.log(fulfilled);
             // output: { brand: 'Samsung', color: 'black' }
        })
        .catch(function (error) {
            // oops, mom don't buy it
            console.log(error.message);
             // output: 'mom is not happy'
        });
};

askMom();
```

Exemple tiré de https://www.digitalocean.com/community/tutorials/javascript-promises-for-dummies

Ci dessous un autre example montrant l'interet des Promesses.

```
const random_in_a_b = (a, b) => {
    return Math.floor(Math.random() * (b - a + 1) + a);
}

const generate_2 = () => {
    return new Promise(callback => {
        var result = random_in_a_b(1, 1000000)
        while (result != 2) {
            result = random_in_a_b(1, 1000000)
        }
        callback("succés")
    })
}

const promise = generate_2().then(r => {
    console.log(r)
})

const a = 2
const b = 4

console.log('a+b = '+ (a+b))
```

```
>>'a+b = 6'
>>'succés'
```

Node et Expo

La suite de ce document sera un petit tutoriel pour installer React Native sur votre ordinateur. Ce sera nécessaire pour la prochaine séance ! 🙂

Node (et npm)

Tout d'abord, nous allons installer Node.js et npm (Node Package Manager), car cela va nous permettre de tester notre application.

- Node.js est, actuellement, la technologie à utiliser pour profiter de tout le potentiel de Javascript. C'est avec ça que tourne Javascript côté serveur.
- npm est le principal gestionnaire de paquets pour Node.js : c'est lui qui s'occupe d'installer les dépendances d'un projet.

Ubuntu

Lancez dans le terminal :

```
sudo apt-get update
sudo apt-get install nodejs npm
```

Cela installera Node et npm. On souhaite ensuite installer expo, qui est un package qui va nous permettre d'observer en temps réel notre application sur notre téléphone.

Enfin on installe Expo et React-Native qui sera utile au développement d'application mobile

Expo

```
npm install --global expo-cli
```

React-Native

```
npm install -g react-native-cli
```

Cela va installer expo et React-Native globalement sur votre ordinateur et le rendre accessible en ligne de commande.

Pour fini, vous devez installer l'application expo sur votre téléphone via l'App Store ou le Play Store.

Windows

Vous pouvez trouver Node.js ici : https://nodejs.org/en/download/

Cliquez sur la version de votre ordinateur. Dans l'installeur Windows, il y aura une option pour installer npm, donc faites attention à ne pas la décocher.

Pour tester si votre installation marche, vous pouvez ouvrir un terminal en faisant Windows + R, et en écrivant cmd ou powershell. Une fois le terminal ouvert, vous pouvez lancer :

```
node -v
npm -v
```

Ces deux commandes vont vous donner les versions de node et npm installés. S'il n'y a pas d'erreurs, c'est tout bon ! 🙂

MacOS

Sur MacOs, vous devez dans un premier temps installer Xcode qui est disponible dans l'appStore.

Ensuite installez homebrew en entrant cette commande dans le terminal

```
ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Une fois ceci fait, on installe Node grâce à HomeBrew depuis le terminal encore une fois

```
brew install node
```

Pour voir si l'installation a bien fonctionné, vous pouvez regarder les versions de node et npm.

```
node -v
npm -v
```

Pour mettre à jour Node si besoin

```
brew upgrade node
```

pour désinstaller Node :

```
brew uninstall node
```

Enfin on installe Expo et React-Native qui sera utile au développement d'application mobile

Expo

```
npm install --global expo-cli
```

React-Native

```
npm install -g react-native-cli
```