Javascript 101

Introduction

JavaScript est un puissant langage de programmation conçu pour ajouter de l'interactivité aux pages web, améliorant ainsi l'expérience utilisateur. Il est présent dans tous les navigateurs web. De nos jours, il est également de plus en plus utilisé sur les serveurs en raison de ses nombreux avantages.

Grâce à sa simplicité, c'est un bon outil pour apprendre l'algorithmique. Les stagiaires auront également l'avantage de connaître un langage de programmation largement répandu à la fin de ce cours.

Il ne faut pas confondre Java et JavaScript. Ce sont des langages de programmation complètement différents. Jetez un œil ici si vous voulez comprendre pourquoi ils ont des noms si similaires.

silentTeacher

computeIt

codeWars

Bases

Afficher des données

```
console.log("hello") ;
```

`console.log()` est souvent utilisé pour déboguer votre code ou pour afficher un petit message. Utilisez-le et abusez-en dans tous les exercices pour comprendre ce qui se passe dans vos programmes.

Variables

```
let age = 25 ;
```

Une variable est… eh bien, c'est un objet qui contient des choses. Le mot-clé `let` nous permet de définir une nouvelle variable. Ensuite, nous pouvons lui attribuer ce que nous voulons à l'aide de l'opérateur `=`.

```
let age = 25 ; // j'ai 25 ans
age = 36 ; // Je vieillis
age = 18 ; // Par miracle, je rajeunis.
```

Types de variables

Les nombres :

```
let age = 25 ;
```

Chaîne de caractères :

```
let phrase = "Coding is fun !";
```

Booléens :

```
let isItTrue = true ;
let isItFalse = false ;
```

Nombres

```
let age = 23 ;

let sisterAge = age + 11 ;

age += 12 ; // age = age + 12 ;
```

```
console.log(age) ;
```

Opérateurs arithmétiques

- Addition : `+`
- Soustraction : `-`
- Multiplication : `*`
- Division : `/`
- Modulo : `%`

Chaînes de caractères

```
let name = "Jean" ;
let nameSister = "Emma" ;
console.log("Mon nom est " + nom) ;
console.log("Le nom de ma soeur est " + nom) ;
```

`new Number()`

```
let htmlBadges = new Number('5') ;

let cssBadges = new Number('7') ;

let totalBadges = htmlBadges + cssBadges ;

console.log('Woaw, vous avez ' + totalBadges + " !") ;
```

`new Number()` convertit une chaîne de caractères en un nombre.

Attention aux types de variables !

```
console.log('10' + '5') ; // '105' => type chaîne de caractères

console.log(10 + 5) ; // 15 => type nombre
```

Exercices

Exercice 1.0 - Mode interactif

Ouvrez la console et utilisez-la pour faire une addition. Puis répétez la même étape jusqu'à ce que vous ayez testé les cinq opérateurs arithmétiques que nous avons présentés précédemment.

Merveilleux ! A partir de maintenant, tu n'auras plus besoin d'une calculatrice.

Exercice 1.1

Créez un nouveau fichier javascript.

Définissez une variable et affichez-la dans la console.

Exercice 1.2

Définissez trois variables : `name`, `firstName` et `city`. Affichez-les comme ceci : "Vous vous appelez Gérard Lambert et vous habitez à Paris".

Ressources

- let
- Opérateurs arithmétiques
- Nombre()

Structures de contrôle

Les structures de contrôle sont des éléments du langage qui affectent la façon dont le programme se comporte.

Théorie

Opérateurs de comparaison

Quelques nouveaux opérateurs qui sont beaucoup utilisés avec les structures de contrôle :

- `===` : retourne `true` si les deux opérandes ont exactement la même valeur. Sinon, il renvoie `false`.
- `!==` : renvoie `true` si les deux opérandes n'ont pas la même valeur.
- `>` : renvoie `true` si le premier opérande est strictement plus grand que le second.
- `<` : renvoie `true` si le premier opérande est strictement inférieur au second.
- `>=` : renvoie `true` si le premier opérande est supérieur ou égal au second.
- `<=` : renvoie `true` si le premier opérande est inférieur ou égal au second.

Opérateurs logiques

- `&&` : renvoie `true` si les deux opérandes sont vrais.
- `||` : renvoie `true` si l'un des opérandes est vrai.
- `!` : ne prend qu'un seul opérande. Inverse la valeur de son opérande. Si elle était `true`, elle devient `false`. S'il était `false`, il devient `true`.

Combiner des opérateurs

Si vous combinez plusieurs opérateurs, JavaScript appliquera certaines règles prioritaires. Mais celles-ci sont difficiles à apprendre et à retenir, il est donc conseillé d'utiliser fréquemment les parenthèses pour les combiner. Exemples :

- `(a > 2) && (b < 4)` : renvoie `true` seulement si `a` est supérieur à 2 **et** `b` est inférieur à 4.
- `(x >= y) || (y === z)` : renvoie `true` uniquement si `x` est supérieur ou égal à `y` **ou** `y` est égal à `z`.
- `!(a < 2)` : renvoie `true` uniquement si `a` n'est *pas* inférieur à 2. Cela revient à écrire `a >= 2`.
- `( !(a < b) && (x > z)) || !(i === j)` : huu… essayez de deviner !

if

```
let size = 185 ;
let weight = 80 ;

if ((size >= 150) || (weight >= 45)) {
  console.log("Vous êtes probablement un adulte") ;
}
```

Le `if` permet de tester une condition et d'exécuter du code uniquement si elle est vraie.

else

```
let size = 185 ;
let weight = 80 ;

if ((size >= 150) || (weight >= 45)) {
  console.log("Vous êtes probablement un adulte") ;
} else {
  console.log("Vous êtes probablement un enfant") ;
}
```

`else` est complémentaire de `if` et permet d'exécuter un autre code si la condition est fausse.

else if

```
let size = 185 ;
let weight = 80 ;

if ((size >= 150) || (weight >= 45)) {
  console.log("Vous êtes probablement un adulte") ;
} else if ((size >= 50) || (weight >= 10)) { {
  console.log("Vous êtes probablement un enfant") ;
} else {
  console.log("Vous êtes probablement un bébé") ;
}
```

Une autre structure à utiliser avec `if`. Elle permet d'effectuer un deuxième test uniquement si le précédent a déjà échoué. Notez que vous pouvez utiliser autant de `else if` que vous le souhaitez.

while

```
let i = 1 ;

while (i <= 100) {
  console.log(i) ;
  i += 1 ;
}
// Ce code affiche tous les nombres de 1 à 100
```

`while` répète le code tant que la condition reste vraie.

Attention aux boucles infinies ! Si la condition à l'intérieur d'un `while` reste toujours vraie, cela signifie que votre programme ne se terminera jamais.

for

```
for (let i = 1 ; i <= 100 ; i += 1) {
  console.log(i) ;
}
// Ce code affiche également tous les nombres de 1 à 100
```

`for` est comme une version spécialisée de `while`. Tout ce qu'il peut faire, `while` peut aussi le faire. Mais sa syntaxe est plus courte pour certains cas d'utilisation courants et il est donc assez souvent utilisé.

A l'intérieur des parenthèses, il y a trois arguments séparés par le caractère `;` :

- Le premier est une commande qui sera exécutée une seule fois, après la première exécution de la boucle.
- Le deuxième est un test qui sera exécuté après chaque exécution de la boucle pour savoir si nous devons continuer ou non (exactement comme dans `while`).
- Le troisième est une commande qui sera exécutée après chaque exécution de la boucle.

switch

```
let consomable = "dattes";
```

```
switch (consomable) {
  case "courgette":
  case "pomme de terre":
  case "carotte":
    console.log("Ceci est un légume");
    break;

  case "banane":
  case "dattes":
    console.log("Ceci est un fruit");
    break;

  default:
    console.log("Ceci n'est ni un fruit ni un légume");
}
```

`switch` commence par évaluer l'expression fournie (cette évaluation ne se produit qu'une fois). Si une correspondance est trouvée, le programme exécutera les instructions associées. Si plusieurs cas de figure correspondent, le premier sera sélectionné (même si les cas sont différents les uns des autres).

Le programme recherche tout d'abord une clause case dont l'expression est évaluée avec la même valeur que l'expression d'entrée (au sens de l'égalité stricte. Si une telle clause est trouvée, les instructions associées sont exécutées. Si aucune clause case n'est trouvée, le programme recherche la clause optionnelle default.

Exercices

Exercice 2.1

Demandez à l'utilisateur d'entrer son âge. Si son âge est d'au moins 18 ans affichez "Vous êtes un adulte". Si ce n'est pas le cas affichez "Vous n'êtes pas encore un adulte".

astuce : Vous pouvez utiliser la fonction prompt pour demander à l'utilisateur de saisir des données dans le navigateur.

Exercice 2.2

Demandez à l'utilisateur de saisir trois nombres : `min`, `max` et `current`. Affichez si `current` est entre `min` et `max`.

Bonus : si `min` est supérieur à `max`, affichez un message d'erreur pour expliquer à l'utilisateur qu'il n'a rien compris puis quittez le programme. (Il ne doit pas poser d'autre question).

Bonus 2 : soyez poli dans les messages d'erreur. (facultatif)

Exercice 2.3

Affichez tous les nombres **pairs** entre 1 et 100.

Bonus : faites deux versions différentes. La première doit utiliser `while` et faire une boucle qui s'exécutera 100 fois. La seconde doit utiliser `for` et créer une boucle qui s'exécutera 50 fois.

Exercice 2.4

Comptez de 1 à 100. Pour chaque nombre pair, affichez le résultat de leur division par 2. Pour chaque nombre impair, affichez le résultat de leur multiplication par 3.

Exercice 2.5

Faites un programme qui demande le numéro préféré de l'utilisateur. Si ce nombre est autre que 42, il affiche "Are you sure ?" et demande à nouveau. (Ce programme ne devrait jamais se terminer tant que l'utilisateur n'a pas choisi 42).

Exercice 2.6

Faites un programme qui demande à l'utilisateur d'entrer un nombre entre 1 et 7. En fonction du nombre, afficher le jour de la semaine qui correspond. (1 => lundi, 2 => mardi, etc…)

Exercice 2.7

Réalisez un programme qui demande à l'utilisateur de saisir un nombre nommé `n`. Puis lui demande `n` fois d'entrer un nouveau nombre. A la fin, affichez la somme de tous les nombres collectés de cette façon.

Exemple : Si l'utilisateur entre `3` pour `n` puis `1`, `2` et `3`, le programme doit afficher `6`.

Ressources

- if…else
- while
- for
- switch

Tableaux

Les tableaux sont l'une des structures de données les plus simples en programmation. Ils permettent de stocker une collection d'éléments dans un ordre précis.

Théorie

Déclaration

```
let emptyArray = [] ; // un tableau vide

let numbersArray = [1, 2, 3] ; // un tableau de 3 nombres

let stringsArray = ["Apple", "Pear", "Banana", "Cherry"] ; // un tableau de 4 chaînes de caractères.
```

Ce n'est pas beaucoup plus compliqué.

Accéder à un élément spécifique

```
let arr = ["Gérard", "Lambert"] ;

console.log("Bonjour " + arr[Ø] + " " + arr[1]) ;
```

La syntaxe `[]` permet d'accéder à un élément spécifique du tableau en utilisant son index. Notez que, en JavaScript, on considère toujours que les index commencent à `0`. Ainsi, le premier élément est à l'indice `0`, le second à l'indice `1`, etc…

Vous pouvez également modifier un élément spécifique de cette façon :

```
let arr = [1, 2, 3] ;

arr[2] = 4 ;

console.log(arr) ; // [1, 2, 4]
```

Obtenir la taille d'un tableau

```
let arr = [1, 2, 3] ;

console.log("La taille est " + arr.length) ; // affiche "La taille est 3".
```

`<array>.length` donne le nombre d'éléments d'un tableau.

Ajouter et supprimer des éléments d'un tableau

```
let arr = ["Apple", "Pear", "Banana"] ;

console.log(arr) ; // ["Apple", "Pear", "Banana"]
```

```
arr.push("Cerise") ;

console.log (arr) ; // ["Pomme", "Poire", "Banane", "Cerise"]

arr.pop() ;

console.log(arr) ; // ["Pomme", "Poire", "Banane"]
```

`push()` et `pop()` respectivement. Notez qu'elles ne fonctionnent qu'à l'extrémité du tableau.

Ainsi, voici les principales fonctionnalités :

```
let fruits = ['pomme', 'banane', 'poire', 'fraise']
```

- `fruits.length`: retourne le nombre d'éléments dans le tableau (ici retourne 4)
- `fruits[0]`: sélectionne le premier élément
- `fruits[length - 1]`: sélectionne le dernier élément
- `fruits.push('pamplemousse')`: ajoute un élément à la fin du tableau
- `fruits.unshift('pamplemousse')`: ajoute un élément au début du tableau
- `fruits.pop()`: supprime le dernier élément
- `fruits.shift()`: supprime le premier élément
- `fruits.indexOf('banane')`: retourne l'index d'un élément
- `fruits.join()`: concatène les éléments dans une chaîne de caractères avec virgules, mais il est possible de spécifier un autre séparateur dans les paranthèses
- `fruits.slice()`: crée une copie du tableau (à associer à une autre variable donc)
- `fruits.splice([début], [nbASupprimer], [élément(s)])`: retire, remplace ou ajoute des éléments.
- Début : l'index à partir duquel commencer le changement, si négatif, part de la fin du tableau
- nbASupprimer : un entier indiquant le nombre d'éléments à retirer ou remplacer
- Element(s) : les éléments à ajouter à partir du début précisé. Si aucun élément n'est spécifié, alors n'en ajoutera pas.

Itérer sur un tableau

Très souvent, nous avons besoin d'itérer sur un tableau pour effectuer une opération sur tous ses éléments. Vous verrez différentes techniques.

La vieille et moche mais qui fonctionne :

```
let arr = ["Apple", "Pear", "Banana"] ;

for (let i = 0 ; i < arr.length ; i += 1) {
  console.log("Voulez-vous manger un " + arr[i] + " ?") ;
}
```

Et les cool kids :

```
let arr = ["Apple", "Pear", "Banana"] ;

for (let elem of arr) {
  console.log("Voulez-vous manger un " + elem + " ?") ;
}
```

Le `for...of` est une syntaxe spécifique pour itérer sur des tableaux. Essayez de l'utiliser pour plus de lisibilité.

Exercices

Exercice 3.0

Créez un tableau, affichez chaque éléments dans la console en utilisant une boucle `for ... of ....`

Exercice 3.1

Ecrivez un programme qui ajoutera tous les éléments d'un tableau.

Testez-le avec les tableaux suivants :

- `[1, 2, 3, 4, 5]`
- `[100, 101, 102]`

Exercice 3.2

Ecrivez un programme qui calculera la valeur moyenne d'un tableau donné.

Testez-le avec les tableaux suivants :

- `[1, 2, 3, 4, 5]`
- `[100, 101, 102]`

Exercice 3.3

Ecrivez un programme qui va créer un double d'un tableau donné.

Bonus : faites une première version qui ne le fera qu'en utilisant `push()`. Faites une deuxième version qui utilise un seul appel de méthode pour effectuer la copie. (Vous devrez faire une recherche sur le MDN ou sur Google pour celle-ci).

Exercice 3.4

Écrivez un programme qui affiche les éléments minimum et maximum d'un tableau donné.

Ressources

- Tableau

Fonctions

Les fonctions sont des morceaux de code qui peuvent être réutilisés à plusieurs endroits différents dans le même programme. Elles sont essentielles pour la réutilisation du code, la modularité et la séparation des préoccupations.

Théorie

Définition d'une fonction simple

```
function hello(nom) {
  console.log("Bonjour " + nom) ;
}

hello() ;
```

```
function add(a, b) {
  return a + b ;
}

console.log("La somme de 2 et 3 est " + add(2, 3)) ;
```

C’est la manière de base de déclarer une fonction. Les paramètres supplémentaires que nous passons lorsque nous appelons une fonction sont appelés ses *arguments*. Une fonction peut aussi retourner quelque chose en utilisant `return`. (Mais elle n’est pas obligée de le faire).

Fonctions fléchées

```
let myFunction = (a, b) => {
  return a + b ;
}

console.log(myFunction(2, 3)) ; // 5

// ou encore plus court
```

```
let myFunction = (a, b) => a + b
```

Les fonctions fléchées (“arrow” est simplement lié à l’aspect du symbole `=>`) est une syntaxe plus courte pour définir des fonctions. Attention, il existe quelques différences, et quelques pièges lors de l’utilisation de cette syntaxe (vous pouvez en apprendre plus à ce sujet ici).

Même si elles peuvent ne pas sembler utiles actuellement, elles ont beaucoup de spécificités qui font qu’elles sont très utilisées en JavaScript. Vous devez donc les connaître.

Récursivité

```
function count(i) {
  if (i <= 100) {
    console.log(i) ;
    count(i + 1) ;
  }
}

count(i) ;
// affiche tous les nombres de 1 à 100
```

Cela n’a rien à voir avec un élément de langage en JavaScript. La *Recursivité* est une technique de programmation que vous pouvez utiliser dans tous les langages de programmation. Elle est utilisée lorsqu’une fonction s’appelle elle-même.

Exercices

Exercice 4.1

Créez une fonction nommée `calcSurface` qui prend la longueur et la largeur d'un rectangle et retourne sa surface. Créez ensuite un programme qui demande la longueur et la largeur d'un rectangle à l'utilisateur puis affiche la surface de ce rectangle. Ce programme doit utiliser la fonction que vous avez créée.

Écrivez une documentation pour la fonction `calcSurface`.

Exercice 4.2

Créez une fonction nommée `rand10()` qui retourne un nombre entier aléatoire entre 1 et 10. Créez un programme qui affichera le résultat de cette fonction à chaque fois qu'elle sera exécutée.

Rédigez une documentation pour la fonction `rand10()`.

Pour cet exercice, vous devrez chercher sur Google comment générer des nombres aléatoires en JavaScript.

Exercice 4.3

En réutilisant la fonction `rand10()` créée dans l'exercice 2, écrivez une fonction nommée `multiRand(n)` qui retourne un tableau de `n` nombres entre 1 et 10. Vous ne devez rien modifier dans `rand10()` pour y parvenir.

Ecrivez une documentation pour la fonction `multiRand(n)`.

Utilisez cette fonction pour créer un programme qui demandera le nombre de nombres aléatoires à générer puis affichera ce même nombre de nombres aléatoires.

Exercice 4.4

Créez une fonction `pickLearner(inputAr, n)` qui prend 2 paramètres.

- `inputAr` : Un tableau d'apprenants (celui que vous avez créé dans exercice 3.0 par exemple)
- `n` : Un nombre, qui doit être supérieur à 0 et inférieur à la longueur de `inputAr`.

La fonction devrait retourner un tableau d'apprenants choisis au hasard.

Exercice 4.5

Créez une fonction nommée `calcDistance` qui prend les coordonnées de deux points différents `A` et `B` dans un espace 2D. Cette fonction doit retourner la distance entre ces deux points.

Exemples de résultats :

- Point A = [1, 1], point B = [2, 2] => 1.41
- Point A = [1, 1], point B = [3, 1] => 2
- Point A = [4, 1], point B = [1, 1] => 3
- Point A = [-2, 2], point B = [2, -2] => 5.65

Créez un programme pour utiliser cette fonction.

Ecrivez une documentation pour la fonction `calcDistance`.

Note : Vous aurez probablement besoin de faire quelques recherches sur Google pour apprendre la formule mathématique pour faire cela. Vous aurez aussi probablement besoin de chercher des fonctions utiles en JavaScript pour vous aider à faire des formules mathématiques complexes…

Exercice 4.6

Créez une fonction `factorial(a)` qui retourne la factorielle d'un nombre.

Cette fonction doit être récursive.

Ressources

- Fonction
- Fonctions fléchées

Objets

Les objets sont un autre type de structure de données en JavaScript.

Théorie

Objets simples

A la base, ils sont une structure assez simple comme les tableaux. Au lieu d'assigner un index à chaque élément, ils permettent de les assigner à une chaîne de caractères. Cette chaîne sera appelée la *clé* et la valeur qui lui est associée sera la… valeur.

Exemple 1:

```
let myObject = {
  // nous pouvons utiliser la notation "" pour la clé
  "prenom" : "Gérard",
  // ... ou simplement le spécifier de cette façon si on le souhaite.
  nom : "Lambert",
  // on peut utiliser n'importe quel type pour la valeur
  âge : 42
} ;

// nous pouvons utiliser la notation [] pour accéder aux valeurs
console.log(myObject["age"]) ; // 25

// ... ou la notation .
console.log(myObject.prenom) ; // Gérard

// nous pouvons attribuer une nouvelle valeur
myObject.age = 54 ;

// ... et il existe un mot-clé spécial pour supprimer des clés
delete myObject["name"] ;

console.log(myObject) ; // {prénom : "Gerard", âge : 54}
```

Exemple 2:

```
let mark = {
  prenom: "Mark", // propiété de l'object
  nom: "Zukerberg",
  email: "mark@facebook.com",

  // A l'ancienne

  sePresenter: function () {
    console.log("Bonjour ! je m'appelle " + this.prenom); // on utilise la
propriété this pour faire référence à l'object.
  },
};

console.log(mark.prenom);
mark.sePresenter();
```

les object littéraux ne sont pas utilisé dans ce sens mais plutôt pour transmettre des informations…

```
function recevoirLesCoordonnees() {
  return { latitude: 40, longitude: 123 };
}

let coordonnées = recevoirLesCoordonnees();
console.log(coordonnées.latitude);
```

Objets et tableaux comme outils pour les structures de données

En combinant les objets, les tableaux et quelques types simples, nous pouvons représenter des structures de données complexes adaptées à la plupart des types de données.

```
{
  prénom : "Gérard",
  nom de famille : "Lambert",
  âge : 42,
  skills : [
    {
      skillName : "JavaScript",
      niveau : "avancé"
    },
    {
      skillName : "HTML",
      niveau : "avancé"
    }
  ],
  adresse : {
    rue : "rue des Campanules",
    numéro : "10",
    ville : "Paris",
    code postal : 1000
```

```
  }
}
```

Ceci constitue également la base de JSON, un format de données inspiré de JavaScript et utilisé aujourd'hui dans presque tous les langages de programmation.

JSON peut représenter des nombres, des booléens, des chaînes, la valeur null, des tableaux (séquences de valeurs ordonnées) et des objets constitués de ces valeurs (ou d'autres tableaux et objets). JSON ne représente pas nativement des types de données plus complexes tels que des fonctions, des expressions régulières, des dates, etc.

Il est construit par rapport à deux structures :

- Une collection de paires nom / valeur. Dans les différentes langages, ce type de structure peut s'appeler objet, enregistrement, dictionnaire, table de hachage, liste à clé ou tableau associatif.
- Une liste ordonnée de valeurs. Dans la plupart des langages, c'est ce qu'on va appeler tableau, liste, vecteur ou séquence.

En JavaScript, on possède ainsi un objet JSON. L'objet global JavaScript JSON possède deux méthodes pour interpréter du JSON et convertir des valeurs en JSON. : les méthodes parse() et stringify().

- La méthode `parse()` analyse une chaîne de caractères JSON et construit la valeur JavaScript ou l'objet décrit par cette chaîne. On peut lui passer une option en deuxième argument qui va prendre la forme d'une fonction permettant transformer la valeur analysée avant de la transformer.
- La méthode `stringify()` convertit une valeur JavaScript en chaîne JSON. On peut lui passer une fonction qui modifie le processus de transformation ou un tableau de chaînes de caractères et de nombres qui sont utilisés comme liste blanche pour sélectionner/filtrer les propriétés de l'objet à inclure dans la chaîne JSON en deuxième argument facultatif.

Exercices

Exercice 5.0

Créer un tableau

Exercice 5.1

Créez une fonction nommée `askTvSerie()` qui demandera à l'utilisateur les données suivantes sur sa série TV préférée :

- Nom
- Année de production
- Noms des acteurs (il peut y en avoir autant que l'utilisateur le souhaite)

Cette fonction doit rassembler toutes les données dans un seul objet et le retourner. La structure des données doit être élégante.

Créez un programme qui utilise cette fonction pour générer un objet série TV et l'afficher à l'utilisateur au format JSON.

Exercice 5.2

Créez une fonction nommée `randomizeCast(tvSerie)` qui prendra comme argument la structure de données que vous avez définie dans l'exercice précédent et retournera une liste de la même distribution mais dans un ordre aléatoire.

Créez un programme qui utilisera `randomizeCast(tvSerie)` et `askTvSerie()` pour demander à l'utilisateur une série télévisée puis affichera une liste aléatoire des acteurs précédents qui formeront le nouveau casting de votre prochaine série.

Ressources

- Initialisateur d'objet

API JavaScript : Application Programming Interface

Une API (Application Programming Interface ou Interface de Programmation Applicative en français) est une interface, c'est-à-dire un ensemble de codes grâce à laquelle un logiciel fournit des services à des clients.

Le principe et l'intérêt principal d'une API est de permettre à des personnes externes de pouvoir réaliser des opérations complexes simplement.

Une API peut être comparée à une commande de voiture (pédale d'accélération, essuie-glace, etc.) : lorsqu'on accélère ou qu'on utilise nos essuies glace, on ne va

pas se préoccuper de comment la voiture fait pour effectivement avancer ou comment les essuies glace fonctionnent.

Les API JavaScript vont pouvoir être classées dans deux grandes catégories :

- Les API intégrées aux navigateurs web et qu'on va donc pouvoir utiliser immédiatement pour du développement web comme l'API DOM (Document Object Model) qui va nous permettre de manipuler le HTML et le CSS d'une page, l'API Geolocation qui va nous permettre de définir des données de géolocalisation ou encore l'API Canvas qui permet de dessiner et de manipuler des graphiques dans une page.
- Les API externes, proposées par certains logiciels ou sites comme la suite d'API Google Map qui permettent d'intégrer et de manipuler des cartes dans nos pages web ou encore l'API Twitter qui permet d'afficher une liste de tweets sur un site par exemple ou bien l'API YouTube qui permet d'intégrer des vidéos sur un site.

BOM : Browser Object Model

Le BOM est une sorte de « super API » elle-même composée de plusieurs API dont certaines sont elles mêmes composées de plusieurs API et etc.

A la base du BOM, nous avons l'interface Window qui représente une fenêtre de navigateur contenant une page ou un document.

L'objet Window implémente l'interface Window. Cet objet est supporté par tous les navigateurs et tous les objets globaux, variables globales et fonctions globales appartiennent automatiquement à cet objet.

Cet objet Window est un objet dit « implicite » : nous n'aurons généralement pas besoin de le mentionner de manière explicite pour utiliser les méthodes (ou fonctions globales) et propriétés (ou variables globales) lui appartenant.

Le BOM est composé de différentes interfaces qu'on va pouvoir utiliser via des objets.

- Object Navigator : permet de récupérer à par exemple la géolocalisation, la langue de l'utilisateur. Cela nous permet de récupérer et de modifier dynamiquement notre page.
- Object History permet d'avoir une action sur l'historique de l'utilisateur (rediriger vers la page précédente, redirection) pas lire l'historique mais rediriger avant après.

- Object location : permet de connaître l'url, le dossier sur le serveur web => redirection.
- Object Screen :palette de couleur, taille écran, résolution.
- Object Document : l'objet que l'on va le plus utilisé, il permet d'avoir accès à la page web et à modifier le html de l'utilisateur, modifier des éléments, supprimer des éléments, écouter des éléments, lorsque l'utilisateur tape sur le clavier, clique sur un élément, descend la scrollbar, envoie un formulaire etc.

Ressources

DOM : Document Object Model

Le Document Object Model (DOM) relie les pages web aux scripts ou aux langages de programmation en représentant la structure d'un document.

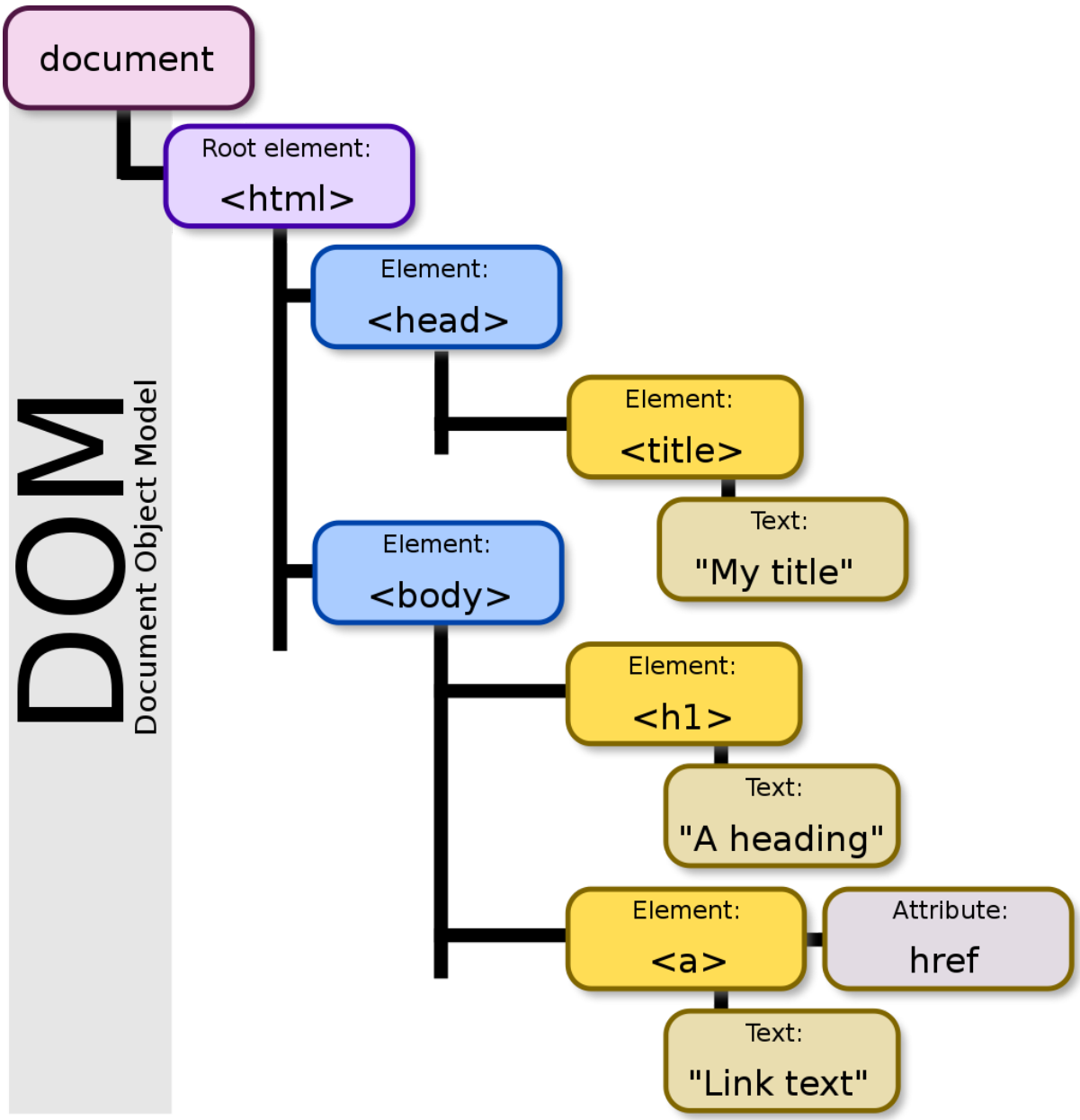
Dans cette série de défis, nous allons explorer différents aspects de cette technologie. Sauf indication contraire explicite, vous ne devez JAMAIS éditer les fichiers HTML fournis, tout doit être fait en utilisant Javascript.

Introduction

Les concepts

L'arbre DOM

Le Document Object Model nous permet de visualiser tout document HTML comme un "arbre". L'arbre est composé de `nœuds`, qui sont les différents éléments HTML de la page. Chaque document est constitué d'un noeud racine, qui contient tous les noeuds enfants suivants.



Un noeud contenant un ou plusieurs noeuds sera appelé un `noeud parent` tandis que les noeuds contenus seront appelés `noeuds enfants`.

Nœuds de texte

A la fin de la ligne, nous trouvons les `noeuds de texte`. Ce sont des noeuds spéciaux qui ne peuvent pas avoir d'enfant, ni d'attributs. Par exemple :

```
<p>Bonjour</p>
```

Ici nous avons le noeud `p` contenant un noeud texte `Hello`.

Autres noeuds

Les attributs peuvent également être manipulés comme des nœuds. Les commentaires HTML aussi.

Accéder au DOM en utilisant Javascript

Il y a quelques objets Javascript prédéfinis dans votre navigateur. Le DOM est l'un d'entre eux. Si vous ouvrez une console et tapez `document.` vous verrez une liste d'accesseurs à l'arbre DOM.

Votre exploration commence ici…

Ce qu'on appelle « DOM » est en fait un ensemble d'interfaces qui vont pouvoir fonctionner ensemble et nous permettre notamment d'accéder à et de manipuler divers éléments de nos documents en JavaScript.

Parmi les interfaces du DOM, quelques-unes vont particulièrement nous intéresser :

- L'interface `Window` qu'on a déjà étudié et qui est liée au DOM
- L'interface `Event`qui représente tout événement qui a lieu dans le DOM (nous allons définir précisément ce qu'est un évènement dans la suite de cette partie)
- L'interface `EventTarget`qui est une interface que vont implémenter les objets qui peuvent recevoir des évènements
- L'interface `Node`qui est l'interface de base pour une grande partie des objets du DOM
- L'interface `Document`qui représente le document actuel et qui va être l'interface la plus utilisée
- L'interface `Element`qui est l'interface de base pour tous les objets d'un document

En plus de ces interfaces incontournables, on pourra également citer les interfaces (mixin) `ParentNode`, `ChildNode`, `NonDocumentTypeChildNode`, `HTMLElement` et `NonElementParentNode` qui vont également nous fournir des propriétés et méthodes intéressantes.

Pour bien vous situer dans la hiérarchie du DOM et entre ces interfaces, vous pouvez retenir que :

- L'interface `EventTarget` est l'interface parent de Node et donc Node hérite (des propriétés et méthodes) de l'interface `EventTarget`.

- L'interface `Node` est le parent des interfaces `Document` et `Element` qui héritent donc de Node (et donc par extension également de EventTarget). De plus, Document et Element implémentent les mixin `ParentNode` et`ChildNode`.
- L'interface `Element` implémente également le mixin `NonDocumentTypeChildNode`.
- L'interface `Document` implémente également le mixin `NonElementParentNode`.
- L'interface `HTMLElement` hérite de l'interface `Element`.

Sélecteurs css DOM

Vous pouvez accéder à tous les éléments du DOM en utilisant les méthodes `querySelector` et `querySelectorAll`.

- `querySelector` renvoie le premier élément qui correspond au sélecteur
- `querySelectorAll` renvoie une liste d'éléments qui correspondent au sélecteur

Exemples

```
<p>
   Ceci est un <a href="homepage.html"> lien </a>, ceci est <a
href="contact.html" id="bold"> un autre lien </a>.
</p>
```

```
const qS = document.querySelector ('a')
console.log (qS.href) // "page d'accueil.html"
console.log (qS.innerText) // "lien"

const qSA = document.querySelectorAll ('a')
console.log (qSA) // [...]
console.log (qSA [0] .innerText) // "lien"
console.log (qSA [1] .id) // "gras"
```

Les sélecteurs de requête Javascript fonctionnent comme des sélecteurs CSS, si vous souhaitez sélectionner une balise `<a>` avec un identifiant `bold`, utilisez simplement :

```
const qS = document.querySelector ('a # gras')
console.log (qS.innerText) // "un autre lien"
```

Accéder à un élément en fonction de la valeur de son attribut id

La méthode getElementById() est un méthode du mixin NonElementParentNode et qu'on va implémenter à partir d'un objet Document.

Cette méthode renvoie un objet Element qui représente l'élément dont la valeur de l'attribut id correspond à la valeur spécifiée en argument.

La méthode getElementById() est un moyen simple d'accéder à un élément en particulier (si celui-ci possède un id) puisque les id sont uniques dans un document.

```
document.getElementById('p1').style.color = 'blue';
```

Accéder à un élément en fonction de la valeur de son attribut class

Les interfaces Element et Document vont toutes deux définir une méthode getElementsByClassName() qui va renvoyer une liste des éléments possédant un attribut class avec la valeur spécifiée en argument. La liste renvoyée est un objet de l'interface HTMLCollection qu'on va pouvoir traiter quasiment comme un tableau.

```
//Sélectionne les éléments avec une class = 'bleu'
let bleu = document.getElementsByClassName('bleu');

//"bleu" est un objet de HTMLCollection qu'on va manipuler comme un tableau
for(valeur of bleu){
    valeur.style.color = 'blue';
}
```

Accéder à un élément en fonction de son identité

La méthode getElementsByTagName() permet de sélectionner des éléments en fonction de leur nom et renvoie un objet HTMLCollection qui consiste en une liste d'éléments correspondant au nom de balise passé en argument.

```
//Sélectionne tous les éléments p du document
let paras = document.getElementsByTagName('p');

//"paras" est un objet de HTMLCollection qu'on va manipuler comme un tableau
for(valeur of paras){
    valeur.style.color = 'blue';
}
```

Accéder directement à des éléments particuliers avec les propriétés de Document

Finalement, l'interface Document fournit également des propriétés qui vont nous permettre d'accéder directement à certains éléments ou qui vont retourner des objets contenant des listes d'éléments.

Les propriétés qui vont le plus nous intéresser ici sont les suivantes :

- La propriété body qui retourne le nœud représentant l'élément body
- La propriété head qui retourne le nœud représentant l'élément head
- La propriété links qui retourne une liste de tous les éléments a ou area possédant un href avec une valeur
- La propriété title qui retourne le titre (le contenu de l'élément title) du document ou permet de le redéfinir
- La propriété url qui renvoie l'URL du document sous forme de chaine de caractères
- La propriété scripts qui retourne une liste de tous les éléments script du document

- La propriété cookie qui retourne la liste de tous les cookies associés au document sous forme de chaine de caractères ou qui permet de définir un nouveau cookie.

```
//Sélectionne l'élément body et applique une couleur bleu
document.body.style.color = 'blue'

//Modifie le texte de l'élément title
document.title= 'Le Document Object Model'
```

Accéder au contenu des éléments et le modifier

Pour récupérer le contenu d'un élément ou le modifier, nous allons pouvoir utiliser l'une des propriétés suivantes :

- La propriété innerHTML de l'interface Element permet de récupérer ou de redéfinir la syntaxe HTML interne à un élément
- La propriété outerHTML de l'interface Element permet de récupérer ou de redéfinir l'ensemble de la syntaxe HTML interne d'un élément et de l'élément en soi
- La propriété textContent de l'interface Node représente le contenu textuel d'un nœud et de ses descendants. On utilisera cette propriété à partir d'un objet Element
- La propriété innerText de l'interface Node représente le contenu textuel visible sur le document final d'un nœud et de ses descendants. On utilisera cette propriété à partir d'un objet Element.

```
//Accède au contenu HTML interne du div et le modifie
document.querySelector('div').innerHTML +=
    '<ul><li>Elément n°1</li><li>Elément n°2</li></ul>';

//Accède au HTML du 1er paragraphe du document et le modifie
document.querySelector('p').outerHTML = '<h2>Je suis un titre h2</h2>';

/*Accède au contenu textuel de l'élément avec un id='texte' et le modifie.
 *Les balises HTML vont ici être considérées comme du texte*/
document.getElementById('texte').textContent = '<span>Texte modifié</span>';

//Accède au texte visible de l'élément avec l'id = 'p2'
let texteVisible = document.getElementById('p2').innerText;
//Accède au texte (visible ou non) de l'élément avec l'id = 'p2'
let texteEntier = document.getElementById('p2').textContent;

//Affiche les résultats du dessus dans l'élément avec l'id = 'p3'
document.getElementById('p3').innerHTML =
    'Texte visible : ' + texteVisible + '<br>Texte complet : ' + texteEntier;
```

En resumé

```
document.getElementsByTagName => permet de sélectionner des balises
document.getElementById => permet de sélectionner des id
document.getElementsByClassName => permet de sélectionner des classes
document.querySelector => permet de sélectionner balise, id ou classe. (h1,
#h1, .h1)
document.querySelectorAll => permet de récupérer tous les éléments (p)
textContent est une propriété qui nous permet de modifier l'intérieur de
notre balise.
innerHTML permet de modifier l'intérieur de modifier l'intérieur de notre
html.
```

Element

A noter qu' avec getElements les Éléments sont tous sélectionnés et stockés dans un tableau pour les utiliser il faudra utiliser l'index du tableau.

Pour ajouter des éléments au DOM il y a trois méthodes.

```
Simplement écrire du texte =>     document.write('test'); ajoute du texte (chaine de caractère à la suite du contenu.)
Ajouter un élément brut =>     document.body.append('test')
```

Grâce à cette méthode on peut sélectionner la ou l'on veut ajouter notre contenu.

3ème méthode pour ajouter des objets.

Créer un élément

```
Let helloWorld = document.createElement('div')
```

L'ajouter

```
document.body.append(helloWorld)
document.body.appendChild(helloWorld)
document.body.insertBefore(helloWorld)
document.querySelector('.container').prepend(helloWorld)
```

Détection des comportements des utilisateurs.

Accéder aux éléments

```
getElementsByTagName() - Sélectionne tous les éléments avec la balise entre parenthèses
getElementById() - Sélectionne un seul élément : le premier ayant l'ID entre parenthèses
getElementsByClassName() - Sélectionne tous les éléments avec la classe entre parenthèses
querySelector() - Sélectionne un seul élément : celui avec le sélecteur entre parenthèses
querySelectorAll() - Sélectionne tous les éléments avec le sélecteur entre parenthèses
```

Modifier les éléments

```
textContent - Modifie le texte d'un élément
innerHTML - Modifie l'HTML d'un élément
```

Ajouter et supprimer des éléments

```
createElement() - Crée un élément
prepend() - Ajoute l'élément entre parenthèses devant l'élément cible
```

```
append() - Ajouter l'élément entre parenthèses derrière l'élément cible (peut contenir du texte)
appendChild() - Ajouter l'élément entre parenthèses derrière l'élément cible (ne peut pas contenir du texte)
insertBefore() - Insère un élément avant l'élément cible
```

Modifier le style d’un élément

```
style.propriété - Modifie la propriété CSS spécifiée, par exemple :
style.color = "orange"
className - Modifie les classes d'un élément
```

Les écouteurs “on” et les propriétés JavaScript // à ne pas utiliser

```
onfocus - Quand l'utilisateur sélectionne l'élément
onchange - Quand l'utilisateur change la valeur de l'élément
onclick - Quand l'utilisateur clique sur l'élément
ondblclick - Quand l'utilisateur double-clique sur l'élément
onkeypress - Quand l'utilisateur appuie sur une touche du clavier dans l'élément
```

Les évènements avec addEventListener

```
click - Quand l'utilisateur clique sur l'élément
mouseover - Quand l'utilisateur passe avec sa souris au-dessus d'un élément
mouseout - Quand l'utilisateur sort avec sa souris d'un élément
copy - Quand l'utilisateur copie un élément
cut - Quand l'utilisateur coupe un élément
paste - Quand l'utilisateur colle un élément
```