Introduction au Javascript

Manipuler les variables en Javascript

`parseInt()`.

À quoi ça sert ?

Cette fonction retourne en résultat un **nombre** en analysant une **chaîne de caractères**;
Seul le **premier** nombre trouvé dans la chaîne de caractère est retourné;
Si le premier caractère de la chaîne ne peut pas être converti en un nombre, la fonction retourne en résultat `NaN` (Not A Number);

Exemple :

```
var texte1 = "Je suis une chaîne de caractère";
var texte2 = "9 février 2016";
parseInt(texte1); // Retourne NaN
parseInt(texte2); // Retourne 9
```

L'intérêt principal est donc de pouvoir convertir un nombre écrit dans une chaîne de caractère de type **string** `"10"` en un vrai nombre de type **number** `10` sur lequel il est possible d'effectuer des opérations.

C'est le cas quand vous récupérez une information de l'utilisateur avec `prompt()`.

Remarque :

Si vous essayez de convertir du texte qui ne peut pas être converti en type `number`, alors Javascript renverra la valeur `NaN` pour "Not a Number" :

```
var test1 = parseInt("1");
var test2 = parseInt("Je suis une chaîne de caractères");
```

Ici `test1` sera bien de type `number` et contiendra `1`. La conversion est réussie.

Et `test2` sera aussi de type `number` mais contiendra `NaN` pour indiquer qu'il y a eu une erreur car la chaîne ne peut pas être convertie en nombre.

Une division par zéro donnera aussi `NaN`.

- -

`parseInt()` existe, c'est `parseFloat`.

À quoi ça sert ?

Cette fonction retourne en résultat un **nombre** qui peut être **décimal** en analysant une **chaîne de caractères** là où `parseInt()` renvoie un **nombre entier** uniquement;
Seul le **premier** nombre trouvé dans la chaîne de caractère est retourné;
Si le premier caractère de la chaîne ne peut pas être converti en un nombre, la fonction retourne en résultat `NaN` (Not A Number);

Exemple :

```
var texte = "3.14";
parseInt(texte); // Retourne 3
parseFloat(texte); // Retourne 3.14
```

- -

`substr()` récupère une partie d'une chaîne de caractères;
la sélection du texte à extraire utilise la **position du premier caractère** à extraire puis la **longueur** de la sous-chaîne que vous souhaitez extraire;
la position de début est **obligatoire**, la longueur est **optionnelle**;
si la longueur n'est pas indiquée, tous les caractères jusqu'à la fin de la chaîne seront récupérés à partir de la position de début;

Exemple :

```
var chaine = "Voici du texte";
var resultat = chaine.substr(9,5); // Affiche "texte"
```

- -

`toLowerCase()` permet de mettre le texte en minuscule;
`toUpperCase()` permet de mettre le texte en majuscule;

Exemple :

```
var chaine = "Voici Du Texte";
var resultat1 = chaine.toLowerCase(); // Affiche "voici du texte"
var resultat2 = chaine.toUpperCase(); // Affiche "VOICI DU TEXTE"
```

- -

`toString()` ; quand vous utilisez une variable de type **number** et que vous souhaitez la convertir en type **string**.

Exemple :

```
var nombre = 2;
var resultat1 = nombre + ''; // Affiche "2"
var resultat2 = nombre.toString(); // Affiche "2"
```

- -

.

En Javascript, il est possible de faire des opérations plus complexes de façon automatisée grâce à `Math.`

```
Math.PI; // Ici PI est une propriété de Math qui récupère le nombre Pi

var result = Math.PI; // 3.141592653589793
```

- -

fonction `round()`:

Exemple :

```
var nombre = 10.23;
var resultat = Math.round(nombre); // Affiche 10
```

Remarque :

Il existe plusieurs fonctions pour `Math` qui permettent de faire d'autres opérations automatiquement (calcul d'angle ,etc.)

- -

`Math.floor()` Idem Math.round() mais arrondi à l'entier inférieur, donc : 10.99 = 10

- -

developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

`Math.random()`. Cette fonction génère un nombre décimal entre 0 (inclus) et 1 (non-inclus). Si on veux des numbre entre

Exemple :

```
var resultat = Math.random(); // 0.985461321876
```

```
var resultat1 = Math.random() * 10; // 9.85461321876
```

---

récupérer le nombre le plus grand, il faut utiliser la fonction `Math.max()`. Idem pour `Math.min()`

Exemple :

```
var grandNombre = Math.max(2,4,6,8); // Récupère 8

var var1 = 42
var var2 = 12
var var3 = 57
var result = Math.max(var1,var2, var3); // 57
```

Pop-Ups:

`prompt()`. Javascript propose nativement des **fonctions** qui permettent de faire des choses bien précises. Vous verrez plus tard comment construire vos propres fonctions. `alert()` est aussi une fonction native de Javascript. aussi il y a `confirm()`

```
var prenom = prompt("Quel est votre prénom ?");
alert(prenom);
```

Les Structures de contrôle en JavaScript

Leçon 1 les condition en Javascript

Jusqu'à maintenant vous avez appris à créer et manipuler des variables de plusieurs types mais les possibilités de vos scripts restent limitées.

Dans cette partie du cours, vous allez voir les **conditions** qui vont permettre d'introduire une certaine intelligence à vos scripts qui vont pouvoir se comporter de façon différente au regard de situations spécifiques que vous aurez prévues et anticipées.

Les conditions utilisent **trois concepts** dont un que vous connaissez déjà :

les **booléens**;
les opérateurs de **comparaison**;
les opérateurs **logiques**;

Une condition est donc un **test** qui permet de vérifier si une **expression** est vérifiée (`true`) ou pas (`false`). Vous connaissez déjà les booléens, dans cet exercice vous allez découvrir les **opérateurs de comparaison** et notamment les opérateurs **d'égalité** :

```
var test1 = 1 == "1";
var test2 = 1 === "1";
```

Dans le code ci-dessus, `test1` vaut `true` et `test2` vaut `false`.

Dans le premier cas `==` permet de vérifier si la valeur est la même, ce qui est vrai.

Dans le second cas `===` permet de vérifier si la valeur **et** le type sont identiques, ce qui est faux car il y a un nombre et une chaîne de caractère.

Le second opérateur est donc plus restrictif.

Remarque :

Dans la plupart des cas, utiliser l'égalité stricte `===` est plus sûre et aussi plus rapide pour Javascript. Avec `==` Javascript va d'abord essayer de convertir le type des variables avant de comparer leur valeur, comme avec `1 == "1"`.

2. Les opérateurs de comparaison d'inégalité

2/16

Dans cet exercice vous allez découvrir les opérateurs permettant de comparer des **inégalités**.

Exemple :

```
var test1 = 1 != 2; // true
var test2 = 1 !== "1"; // false
```

Dans le code ci-dessus, `test1` vaut `true` car 1 est bien différent de 2. Et `test2` vaut `true` car même si le contenu est identique, le **type** de variable est différent.

Dans le premier cas, l'opérateur compare juste si la valeur est différente, dans le second cas l'opérateur compare si la valeur **ou** le type est différent (et le type est bien différent car on compare un nombre avec du texte).

3. Les opérateurs de comparaison supérieur et inférieur

3/16

Dans cet exercice vous allez découvrir les opérateurs permettant de comparer des **supériorités** et **infériorités**.

Exemple :

```
var test1 = 1 > 2; // False
var test2 = 1 >= 1; // True
var test3 = 1 < 2; // True
var test4 = 1 <= 1; // True
```

4. Opérateur logique ET pour toutes les expressions

4/16

Outre les opérateurs de comparaison, les conditions utilisent aussi les **opérateurs logiques**.

Ils sont au nombre de trois :

Opérateurs	Sens/EN	Exemple
&&	ET/and	var1 && var2
||	OU/or	var1 || var2
!	NON	!var1

Dans cet exercice vous allez voir l'opérateur **&&**. Cet opérateur permet de vérifier que **toutes** les expressions sont vraies (`true`) et pas seulement l'une d'entre elles.

Exemple :

```
var expression1 = 1 < 2; // True
var expression2 = 1 > 2; // False
var test =  expression1 && expression2;
```

Ici, `test` vaut `false` car toutes les expressions testées ne sont pas vraies (`true`).

5. Opérateur logique OU pour au moins une expression

.

Exemple :

```
var expression1 = 1 < 2; // True
var expression2 = 1 > 2; // False
var test =  expression1 || expression2;
```

Ici, `test` vaut `true` car une des expressions testées est vraie (`true`).

Remarque :

L'opérateur `||` (OU) sert aussi à retourner **le contenu** d'une variable évaluée à `true`.

6. Les opérateurs logiques NON pour inversé la valeur

6/16

Dans cet exercice vous allez voir l'opérateur **!**.

Attention, cet opérateur ne s'utilise qu'avec une seule expression et pas deux comme les opérateurs logiques précédents.

L'opérateur `!` est une **négation** car il **inverse** la valeur ou expression qui lui est donnée.

Exemple :

```
var expression = true; // True

var test = !expression; // False
```

Ici, `test` vaut `false` car l'opérateur à inversé la valeur d'origine.

7. Qu'est-ce qui est vrai ou faux ? Falsos y verdaderos

7/16

Avant d'aborder concrètement les conditions. Il est nécessaire de faire le point sur ce qui est considéré comme vrai et comme faux en Javascript. Il y a quelques subtilités à connaître.

Ce qui est considéré comme faux :

le booléen `false`;
une chaîne vide `""`;
le nombre zéro `0`
`undefined`;

Ce qui est considéré comme vrai :

le booleéen `true`;
une chaîne contenant zéro `"0"`;
une chaîne contenant false `"false"`;

Il existe d'autres subtilités similaires mais celles-ci sont les plus évidentes.

8. La structure if

Leçon 8/16

Voyons maintenant concrètement les **structures conditionnelles**, qu'on appellera simplement "conditions". Il existe trois types de conditions :

la stucture `if` `else`;
les switchs;
les ternaires;

Commençons par la première qui est aussi la plus utilisée.

Dans les exercices précédents vous avez vu comment récupérer un booléen (`true`, false`) en testant des variables avec les opérateurs de comparaison et les opérateurs logiques.

Avec les conditions; le résultat (booléen) d'un test permettra de modifier le flux d'exécution de votre code et donc de donner une certaine intelligence à votre script.

La structure if (si)

Elle est composée :

du mot-clé `if`;
suivi de parenthèses `()` contenant l'expression à tester, et donc le booléen qui est retourné en résultat;
des accolades `&#123;}` contenant le code à exécuter si la condition entre les parenthèses est vérifiée, résultat `true`.

Exemple :

```
if (1 == 1)
&#123;
    // La condition est vérifiée (true), on exécute le code qui est ici.
}

if (1 === "1")
&#123;
    // La condition n'est pas vérifiée (false) car les variables n'ont pas le même type, donc tout le code ici ne sera pas exécuté.
}
```

9. La structure else

La structure **if** permet de vérifier si une condition est vérifiée et d'exécuter du code seulement si c'est le cas. Il serait bien de pouvoir, en plus, exécuter du code dans le cas où cette même condition n'est pas vérifiée.

Il est possible d'utiliser plusieurs **if** :

```
var var1 = true;
if (var1)
&#123;
    // Code exécuté si var1 est vrai;
}
if (!var1)
&#123;
    // Code exécuté si var1 est faux;
    // En effet, ici on teste l'inverse de var1 avec !,
donc on teste si c'est faux;}
```

Mais il y a plus simple avec l'utilisation de la structure **else**.

La structure else (sinon)

Si un `if` peut s'utiliser seul, un `else` va de paire avec un `if` et ne peut pas être utilisé seul, ce qui donne :

```
if ()
&#123;
}
else
&#123;
}
```

L'autre différence est qu'il n'est pas nécessaire d'indiquer quelle expression il faut vérifier entre parenthèses. En fait on le sait déjà puisque c'est l'inverse de l'expression testée dans le `if` qui précède.

Ainsi l'exemple ci-dessus devient :

```
var var1 = true;
if (var1) // Si var1 est vrai
&#123;
    // Code exécuté si var1 est vrai;
}
else // Sinon, var1 est forcément faux
&#123;
    // Code exécuté si var1 est faux;
```

```
    // En effet, ici on teste l'inverse de var1 avec !,
donc on teste si c'est faux;
}
```

10. La structure elseif

10/16

Vous savez utiliser une structure `if` et `else` pour tester si une condition est vérifiée ou pas. Il est aussi possible de tester **plusieurs conditions** les unes après les autres en procédant ainsi :

la première condition est testée avec `if`;
une deuxième condition est testée **si la précédente n'est pas vérifiée** avec `else if`;
d'autres conditions peuvent être testées en ajoutant autant de `else if`;
enfin, si aucune des conditions précédentes n'est vérifiée, la structure `else` exécute le code souhaité;

La structure elseif (sinon si)

Avec `else if` il est donc possible de tester plusieurs conditions à la fois:

```
if (condition1)
&#123;
    // Code exécuté si "condition1" est vérifié
}
else if (condition2)
&#123;
    // Code exécuté si "condition1" n'est pas vérifié...
    // ... et si "condition2" est vérifiée
}
else if (condition3)
&#123;
    // Code exécuté si "condition2" n'est pas vérifée...
    // ... et si "condition3" est vérifiée
}
... // Ainsi de suite
else
&#123;
```

```
    // Si aucune des conditions n'est vérifiée alors le code ici est exécuté
}
```

11. La structure switch

Vous venez de voir la première structure conditionnelle `if elseif else`.

Dans l'exercice précédent vous avez testé si `number` était positif, négatif ou égale à zéro. Imaginez une situation dans laquelle vous devez tester toutes les possibilités pour cette même variable `number`.

Il va falloir vérifier si `number` est égale à -10, -9, -8 ... jusqu'à 10. Avec une structure de type `if elseif else`, le code devient vite lourd et pas forcément lisible.

C'est là qu'intervient la structure `switch` :

```
switch (maVariable)
&#123;
    case valeur1:instruction1;
    break;
    case valeur2:instruction2;
    break;
    case valeur3:instruction3;
    break;
    default:instruction4;
}
```

Comment ça marche ?

il faut le mot-clé `switch` suivi de la variable à tester entre parenthèses `()`;
ensuite tout se passe entre une seule paire d'accolades `&#123;}`;
chaque possibilité est testée avec le mot-clé `case` suivi de la valeur à laquelle doit être comparée `maVariable`;
attention, `case` vérifie uniquement si `maVariable` est **égale** à la valeur spécifiée (ici `valeur1, valeur2, valeur3`) et rien d'autre;

si l'égalité est vérifiée, alors l'instruction qui suit le double-point `:` est exécutée;
pour chaque `case` il faut un `break` qui permet d'arrêter le switch si l'égalité est vérifiée car pas besoin d'aller plus loin;
si aucun des `case` n'est vérifié, alors on exécute l'instruction par défaut avec le mot-clé `default`;

Remarque :

Avec `case` c'est une égalité stricte `===` qui est testée, c'est-à-dire que le contenu et le type de `maVariable` doivent être identique soit à `valeur1` ou `valeur2` ou `valeur3`.

12. La structure ternaire

Vous venez de voir :

la structure conditionnelle `if elseif else`;
le switch;

Le dernier type de structure conditionnelle sont les **ternaires**. Leur avantage est d'être simple à écrire mais c'est au détriment de la lisibilité.

Une structure ternaire est comme une structure `if else` mais écrite sur une seule ligne.

Exemple :

```
var permis = confirm("Avez-vous le permis de conduire ?");
var resultat;
if (permis) // confirm() renvoi un booléen, donc on peut directement tester "permis", pas besoin de mettre "permis == true"
&#123;
    resultat = "Vous pouvez conduire";
}
else
&#123;
    resultat = "Vous ne pouvez pas conduire";
}
```

Equivalent ternaire :

```
var permis = confirm("Avez-vous le permis de conduire ?");
```

```
var resultat = permis ? "Vous pouvez conduire" : "Vous ne pouvez pas conduire";
```

Comment ça marche ?

la variable `resultat` récupère le résultat de la ternaire;
la variable `permis` est testée par la ternaire;
suivie par `?`;
suivi par une première valeur puis `:` et une seconde valeur;

Si `permis` est vérifiée (évaluée à `true`) alors la première valeur sera retournée.

Si `permis` n'est pas vérifiée (évaluée à `false`) alors la seconde valeur sera retournée.

13. Astuce pour tester une variable

Vous savez qu'il est possible de tester ce que contient une variable avec `typeof`. Vous avez vu aussi que, quelque soit le contenu d'une variable, il peut être converti en booléen `true` (un nombre différent de zéro, une chaîne avec du texte) ou `false` (le nombre zéro, une chaîne vide), etc.
Du coup, en utilisant une condition `if` il est tout à fait possible de tester si une variable est `true` ou `false` de façon très simple :

Méthode standard :

```
var test = "Je suis une chaîne de caractères";
if (test == true)
&#123;
    alert("La variable test est vraie");
}
```

Méthode simple :

```
var test = "Je suis une chaîne de caractères";
if (test)
&#123;
    alert("La variable test est vraie");
}
```

14. Astuce avec l'opérateur OU

14/16

Cet exercice est un rappel sur l'opérateur logique **OU** qui possède une fonctionnalité particulière (retournez voir la description de l'exercice en question si nécessaire).
Exemple :

```
var var1 = "";
var var2 = 1;
var var3 = "Je suis une chaîne de caractères";
var resultat = var1 || var2 || var3;
```

Dans cet exemple, `resultat` contient la valeur de la première variable évaluée à `true`.

15. Exercice sur les conditions

Exercice 15/16

Soit les variables `var1`, `var2` et `var3` déjà déclarées et qui contiennent du texte.

Dans le fichier `"script.js"` :

Déclarez la variable `texteComplet`;
Concaténez les 3 variables dans leur ordre (de var1 à var3);
Récupérez l'ensemble du contenu concaténé dans `texteComplet`';
Créez une structure pour vérifier si la longueur de `texteComplet` est strictement supérieure à 100;
Si c'est vérifié, affichez "C'est une grande phrase" avec `alert()`;
Si ce n'est pas vérifié, affichez "C'est une petite phrase" avec `alert()`;
cliquez sur "**Soumettre ma réponse**".

LEÇON 2 Les boucles en JavaScript

1. La boucle for

Les **boucles** sont aussi des structures de contrôles qui permettent de contrôler le flux d'exécution de votre script en fonction de critères ou conditions que vous aurez définis. Cela ajoute un degré d'intelligence à votre code.

À quoi ça sert ?

Une boucle permet d'exécuter une portion de code un certain nombre de fois.

La structure de base d'une boucle est très similaire à un `if` à la différence que le mot-clé n'est pas le même (ce n'est pas `if` mais un autre mot).

Chaque fois que vous utiliserez une boucle, il faudra s'arranger pour sortir de la boucle à un moment donné : si votre boucle tourne à l'infini, ça fera planter votre navigateur.

Dans cet exercice vous allez voir le premier type de boucle : **for**.

Exemple théorique :

```
for (declaration1;declaration2;declaration3)
&#123;
    // Code à exécuter
}
```

`declaration1` est exécuté avant que la boucle ne commence;
`declaration2` est la condition pour exécuter la boucle;
`declaration3` est exécuté à chaque itération de la boucle;

Exemple pratique :

Ci-dessous un exemple concrêt d'une boucle qui s'exécute tant que `number` est inférieur ou égal à 5.

```
for (var i = 0; i <= 5; i++)
&#123;
    alert("La boucle est à l'itération "+i);
}
```

Ici, la boucle sera exécutée 6 fois, on dit qu'il y a 6 **itérations**. Pour chaque itération la boucle affichera dans une pop-up le numéro de l'itération.

Comment ça marche :

avant la première exécution de la boucle, on déclare `i = 0`;
tant que `i <= 5` on exécute ce qu'il y a dans la boucle;
à chaque fin d'itération (donc une fois que le code est exécuté), on incrémente `i++`;

2. La boucle while avec itérateur

2/6

Vous venez de voir la boucle **for** qui exécute une portion de code **un certain nombre de fois** selon les déclarations indiquées en entrée de la boucle.

Dans cet exercice vous allez voir la boucle **while** qui exécute une portion de code **tant que la condition en entrée est vérifiée** (égale à `true`).

La différence entre `for` et `while` est subtile :

`for` utilise un itérateur qui est incrémenté (`++`) pour permettre de sortir de la boucle à un moment donné;
`while` vérifie si une condition est vérifiée et, si à un moment donné elle ne l'est plus, alors on sortira de la boucle;

Exemple théorique :

```
while (condition)
&#123;
    // Code exécuter tant que condition est vérifiée (true)
}
```

3. La boucle while sans itérateur

3/6

Dans l'exercice précédent vous avez utilisé la boucle **while** de la même manière qu'une boucle **for**, c'est-à-dire avec un itérateur.
MAIS il est tout à fait possible de sortir d'une boucle `while` sans utiliser d'itérateur. Il suffit de s'arranger pour que la condition en entrée de boucle ne soit plus vérifiée (`false`). Ce n'est pas le cas de la boucle `for` qui utilise forcément un itérateur.

Pour sortir de la boucle, il suffit de cliquer sur **annuler** au moment où le `prompt()` demande de rentrer du texte.

4. Sortir d'une boucle avec break

4/6

Dans l'exercice précédent vous avez vu comment sortir d'une boucle en faisant en sorte que la condition ne soit plus vérifiée.

Il existe une méthode plus simple pour sortir d'une boucle, avec l'utilisation de **break**. Vous l'avez déjà vu en abordant les **switch**.

5. Sortir d'une itération avec continue

5/6

Soit trois variables : `i` qui vaut `0` , `j` qui contient un nombre et `limit` qui contient un nombre.

Dans le fichier `"script.js"` :

créez une boucle `while` qui doit s'exécuter tant que `i` est strictement inférieure à `limit`;
dans la boucle, incrémentez `i` pour chaque itération;
ajoutez une condition `if` qui exécutera `continue` si `i` est strictement inférieur à `limit` divisé par 2;
après la condition, incrémentez `j`;
cliquez sur "**Soumettre ma réponse**".

6. La boucle do while

6/6

La structure de boucle `do while` est similaire à `while` à la différence qu'ici la boucle sera **toujours** exécutée **au moins une fois**.
Avec ce type de structure, la boucle est exécutée une première fois, puis la condition est vérifiée. Il y a donc toujours au moins une itération.
Exemple :

```
do
&#123;
    // Votre code exécuté au moins une fois
}
while (condition);
```

Remarque :
Notez la présence du `;` à la fin du `while`.

LEÇON 3 Révision des conditions et des boucles

1. Exercice script vérification valeur

1/6

Exercice de rappel dans lequel il faut écrire un script qui vérifie quel est le plus grand nombre entre 2 nombres entiers ou si ils sont égaux.

2. Exercice écrire un script avec demande utilisateur

2/6

Exercice de rappel dans lequel il faut écrire un script qui demande à l'utilisateur un nombre entre 0 et 10 tant qu'il n'a pas donné un nombre correct.

Astuce:
Pour vérifier que `result` est correct, il faut vérifier si c'est bien entre 0 et 10 inclus. Et que c'est bien un nombre avec `isNaN(result)`.
`isNaN()` vérifie si un nombre est incorrect. En effet `prompt()` renvoie une chaîne de caractères, `parseInt();` permet de convertir la chaîne en nombre ("11" > 11), si la chaîne est vide alors `parseInt();` renverra `NaN`.

```
function milliseconds(x) {
  if (isNaN(x)) {
    return 'Not a Number!';
```

```
  }
  return x * 1000;
}

console.log(milliseconds('100F'));
// expected output: "Not a Number!"

console.log(milliseconds('0.0314E+2'));
// expected output: 3140
```

3. Exercice écrire un script qui conditionne affichage

3/6

Exercice de rappel dans lequel il faut écrire un script qui affiche les nombres pairs entre 0 et 10.

Exemple de rendu :

0 2 4 ...

4. Exercice script qui donne affichage et information

4/6

Exercice de rappel dans lequel il faut écrire un script qui affiche les nombres entre 0 et 10 et indiquer si le nombre est pair ou impair.
Astuce:
Pour savoir si c'est pair ou impair, un bon moyen est d'utiliser le **modulo**, l'opérateur `%` qui représente le reste entier d'une division.

Sachant que :

```
1 % 2 = 1;
2 % 2 = 0;
3 % 2 = 1;
```

On constate que un chiffre impair renvoi 1, un chiffre pair renvoi 0. En Javascript 1 est l'équivalent de true et 0 l'équivalent de false. Il est donc possible de vérifier si c'es pair ou impair.
Exemple de rendu :
0 Pair 1 Impair 2 Pair ...

5. Exercice reconstitution de chaïne de caractères

5/6

Exercice de rappel dans lequel il faut écrire un script qui reconstitue une chaîne de caractères.

astuce:
Par exemple, pour récupérer le deuxième caractère d'une chaîne il faut utiliser `maChaine.charAt(1)` car le premier caractère est à la position 0.

6. Exercice comparaison de 2 chaînes de caractères

6/6

Exercice de rappel dans lequel il faut écrire un script qui compare deux chaînes lettre par lettre et récupère la position à partir de laquelle les chaînes diffèrent.

astuce :
Pour comparer les deux caractères en cours, il suffit de vérifier leur égalité ainsi :

```
chaine1.charAt(i) == chaine2.charAt(i);
```

Les tableaux, les objets, les fonctions

LEÇON 1 Les tableaux et les objets en Javascript

1. Déclarer un tableau

1/31

Depuis le début du cours vous avez vu différents types de variables permettant de stocker différents types de valeurs.

Maintenant une question. Comment feriez-vous pour stocker une liste de courses dans une variable ?

Une liste de courses c'est du texte alors utilisons une variable "string" :

```
var liste = "fruits,légumes,eau,lait";
```

C'est une liste assez courte mais il apparaît déjà peu pratique de pouvoir accéder à un élément de la liste en particulier, de le remplacer, d'en supprimer, d'en ajouter, etc.

C'est là qu'interviennent **les tableaux**. Un tableau (en anglais "array") permet de stocker une liste de valeurs. Ces valeurs peuvent être du texte, un nombre, un booléen, même un tableau.

Comment déclarer un tableau ?

```
var monTableau = [];
```

Il suffit en fait de créer une variable et de lui affecter des crochets `[]`. Le tableau est déclaré mais il est vide. Ci-dessous un exemple de tableau qui contient des valeurs :

```
var monTableau = ["fruits","légumes",5,[]];
```

Ici `monTableau` contient du texte, un nombre et un tableau. Chaque valeur doit être séparée par une `,`.

2. Tableau et objet

Aborder les tableaux en Javascript nécessite de faire un point sur la notion **d'objet**.

Un objet c'est quoi :

En Javascript, une chaîne de caractères, un nombre, un booléen sont en fait des objets.

```
var texte = "Je suis une chaîne de caractères.";
```

Ci-dessus, on créé la variable `texte` qui contient un objet qui représente une chaîne de caractères. Plus qu'une variable, on a donc créé un objet.

La structure d'un objet :

Un objet contient toujours trois éléments :

un constructeur;
des propriétés;
des méthodes;

Vous avez déjà utilisé des **propriétés** et des **méthodes** natives que Javascript met à disposition pour les chaînes, les nombres, etc. Si on reprend le code précédent :

```
var texte = "Je suis une chaîne de caractères.";
var longueur = texte.length; // Propriété
var maj = texte.toUpperCase(); // Méthode
```

Dès que l'on créé une chaîne de caractères, pour Javascript ce sera un objet et il propose par défaut certaines propriétés et méthodes qui permettent de manipuler l'objet stocker dans la variable `texte`.

Le caractère important ici est le point `.`, il permet de dire que l'on souhaite accéder à la propriété `length` de `texte` qui contient l'objet en question, autrement dit la chaîne de caractères.

Pourquoi parler des objets :

Aborder cette notion maintenant est important. Soit le code suivant :

```
var txt = "blablabla";
var nb = 10;
alert(typeof txt); // affiche "string"
alert(typeof nb); // affiche "number"
```

Avec `typeof` on peut connaître le type d'une variable selon ce qu'elle contient. A votre avis, que donne `typeof` avec un tableau `[]` ?? Faites l'exercice pour avoir la réponse.

3. Tableau et indice

3/31

Dans l'exercice précédent, vous avez constaté que utiliser `typeof` sur une variable qui contient un tableau `[]` renvoi l'information `object`.

C'est parce que, comme expliqué dans l'exercice précédent, les variables contiennent en réalité des **objets** qui représentent soit du texte, un nombre, un tableau, etc.

A la question "pourquoi `typeof` renvoi `object` et non par exemple `array` (tableau en anglais) ?" nous y répondrons par la suite.

Donc, pour résumé, en Javascript il y a **5** types de variables (ou d'objets pouvant être représentés dans une variable pour être précis) que peut renvoyer `typeof` :

> string
> number
> boolean
> undefined

> object

Dans cet exercice, vous allez apprendre comment accéder et récupérer un élément grâce aux indices.

Qu'est-ce qu'un indice ?

L'indice est un nombre unique qui correspond à un élément du tableau. Soit le tableau `fruits` suivant :

```
var fruits = [Banane,Fraise,Pomme,Poire,Kiwi];
```

Pour Javascript, voilà à quoi il ressemble :

0 | 1 | 2 | 3 | 4 Banane | Fraise | Pomme | Poire | Kiwi

Chaque valeur possède un indice. Et donc pour récupérer le premier fruit il suffit de faire ainsi :

```
var banane = fruits[0];
```

Il suffit d'indiquer l'indice correspondant à la valeur souhaitée entre crochet après le nom de la variable.

Remarque :

Les indices commencent à zéro. Donc pour récupérer la première valeur d'un tableau il faut utiliser `[0]`.

7. Ajouter un élément à la fin d'un tableau

Dans cet exercice, vous allez ajouter un élément à la fin d'un tableau en utilisant la méthode `push()`.

Exemple :

```
var tab = ['banane','fraise','pomme'];
tab.push('kiwi'); // Ajoute "kiwi" à la fin du tableau
```

8. Ajouter un élément au début d'un tableau

8/31

Dans cet exercice, vous allez ajouter un élément au début d'un tableau en utilisant la méthode `unshift()`.

Exemple :

```
var tab = ['banane','fraise','pomme'];
tab.unshift('kiwi'); // Ajoute "kiwi" au début du tableau
```

9. Supprimer le dernier élément d'un tableau

9/31

Dans cet exercice, vous allez supprimer le dernier élément d'un tableau en utilisant la méthode `pop()`.

Exemple :

```
var tab = ['banane','fraise','pomme'];
tab.pop(); // Supprime "pomme"
```

10. Supprimer le premier élément d'un tableau

10/31

Dans cet exercice, vous allez supprimer le premier élément d'un tableau en utilisant la méthode `shift()`.

Exemple :

```
var tab = ['banane','fraise','pomme'];
tab.shift(); // Supprime "banane"
```

11. Convertir une chaîne en tableau

11/31

Dans cet exercice, vous allez convertir une chaîne de caractères en tableau avec `split()`.

Exemple :

```
var texte = 'Voici du texte';
var tableau = texte.split(' ');
```

Ici le critère de découpage de `texte` est l'espace indiqué avec `split(' ')`, on obtient donc :

```
tableau = ['Voici','du','texte'];
```

12. Convertir un tableau en chaîne

12/31

Dans cet exercice, vous allez convertir un tableau en chaîne de caractères avec `join()`.

Exemple :

```
var tableau = ['banane','fraise','pomme'];
var texte = tableau.join(' '); //
```

Ici les éléments de `tableau` seront espacés dans `texte` car on a précisé un espace avec `join(' ')`, on obtient donc :

```
texte = 'banane fraise pomme';
```

Si vous utilisez simplement `join()` (sans espace), les éléments du tableau seront collés les uns aux autres.

13. Parcourir un tableau

13/31

Dans cet exercice, vous allez parcourir un tableau en utilisant une boucle.

Quelle type de boucle utiliser ?

Les boucles `while` et `for` permettent de parcourir un tableau. Mais la boucle `for` semble plus logique à utiliser.

La boucle `for` utilise forcément un **itérateur** (le `i++`) et c'est avec cet itérateur que l'on va parcourir le tableau.

Exemple :

```
var tableau = ['banane','fraise','pomme'];
var i;
var longueur = tableau.length; // On récupère la longueur une fois pour
toute ici car la longueur ne changera pas.
for (i = 0; i < longueur; i++)
&#123;
```

```
    alert(tableau[i]);
}
```

Dans l'exemple, la boucle affiche l'élément du tableau dont l'index (et donc la position dans le tableau) correspond à `i`, et ce tant que `i` est inférieur à la longueur du tableau (et donc au nombre d'éléments contenus dedans).

Ici le tableau a une longueur de 3 (car trois éléments), comme `i` par de zéro pour correspondre à l'index du premier élément, il suffit d'indiquer **strictement inférieur** à la longueur du tableau.

La boucle fera donc trois itérations avec i = 0, i= 1, i = 2.

14. Déclarer un tableau associatif

Jusque là vous avez vu comment utiliser un tableau **ordonné**. On dit ordonné car les éléments sont associés à un **indice**.

Il existe aussi les tableaux **associatifs** dans lesquels les éléments sont accessibles via un **identifiant**.

Pour déclarer un tableau associatif on utilise des accolades et non des crochets.

```
var voiture = &#123;};
```

Si les indices n'ont pas besoin d'être précisés lors de la déclaration d'un tableau ordonné, les identifiants d'un tableau associatif doivent l'être.

On associe donc un identifiant à une valeur.

Exemple :

```
var voiture = &#123;
    marque : 'Bugatti',
    modele : 'Chiron',
    couleur : 'bleue',
    annee : 2016
};
```

Voici un tableau associatif avec des identifiants (marque, modele, couleur, annee) et des valeurs associées.

Remarque :

On pourrait tout mettre sur une même ligne avec le `;` à la fin comme d'habitude. Mais pour plus de lisibilité, c'est ainsi qu'on présente un tableau associatif.

Le double-point `:` séparant identifiant et valeur sont obligatoires ainsi que la virgule `,` qui sépare chaque couple identifiant/valeur **sauf** pour le dernier.

15. Accéder aux éléments d'un tableau associatif

15/31

Avec les tableaux ordonnés, vous savez comment accéder à un élément en utilisant l'indice :

```
var fruits = [Banane,Fraise,Pomme,Poire,Kiwi];
var banane = fruits[0];
```

Et pour les tableaux associatifs ?

```
var voiture = &#123;
    marque : 'Bugatti',
    modele : 'Chiron',
    couleur : 'bleue',
    annee : 2016
};
```

Il suffit de faire comme ceci :

```
var quelleCouleur = voiture.couleur;
```

On indique la variable qui contient le tableau associatif puis le point `.` puis l'identifiant souhaité.

Que remarquez-vous ?

La syntaxe est la même que ceci :

```
var texte = "Voici du texte";
var longueur = texte.length; // Pareil que voiture.couleur
```

Rappelez-vous, les variables qui contiennent du texte, un nombre, etc. contiennent en fait un objet qui représente du texte, un nombre, etc.

Et les objets possèdent des propriétés. `.length` est une propriété fournie par défaut par Javascript quand vous créé une chaîne de caractères, comme pour la variable `texte` ci-dessus.

Un tableau associatif est structuré comme un objet, tel que Javascript le conçoit. Et **l'identifiant** est en fait une **propriété**.

`voiture` contient donc un objet avec des propriétés qui ont des valeurs.

Remarque :

Un tableau associatif ne possède pas de propriétés ou méthodes natives puisque c'est vous qui allez définir les propriétés.

16. Ajouter un élément dans un tableau associatif

16/31

Etant donné qu'un tableau associatif ne possède pas de méthodes par défaut il n'est pas possible d'utiliser `push()` comme dans un tableau ordonné.

En fait il suffit de spécifier une nouvelle propriété (ainsi que sa valeur) comme ceci :

```
voiture.prix = "2.4 millions d'euros"; // Oui c'est le prix de la Bugatti !
```

17. Parcourir un tableau associatif

17/31

Pour parcourir un tableau ordonné, la boucle `for` est adaptée car elle utilise l'indice correspondant à chaque élément du tableau.

Un tableau associatif ne possède pas d'indice numérique donc la boucle `for` n'est pas adaptée. Il faut utiliser un nouveau type de boucle spécifique aux tableaux associatifs : `for in`.

Exemple :

```
var voiture = &#123;
    marque : 'Bugatti',
    modele : 'Chiron',
    couleur : 'bleue',
```

```
    annee : 2016
};
// Ci-dessous la boucle
for (var id in voiture)
&#123;
    alert(voiture[id]);
}
```

Il y a trois mots clés indispensables dans la boucle : for, var, in. Ici, on déclare une nouvelle variable `id` qui va parcourir chaque propriété du tableau `voiture`.

Pour chaque propriété, on l'affiche avec `alert(voiture[id])`.

Remarque :

Utiliser le nom `id` n'est pas obligatoire, vous pouvez donner le nom que vous voulez, mais c'est conseillé de faire commme ceci.

18. Exercice récupérer la longueur d'une propriété

Maintenant, une série d'exercices de rappel sur les tableaux, mais aussi le reste (conditions, boucles, variables, etc.). Les instructions seront volontairement minimales.

Dans cet exercice, l'objectif est de récupérer la longueur d'une propriété d'un tableau associatif.

Rappel :

Il n'est pas possible de récupérer la longueur d'un tableau associatif, mais le faire sur une des propriétés du tableau oui. Par exemple, récupérer la longueur du texte associé à la propriété "marque" du tableau "voiture".

LEÇON 2 Les fonctions en JavaScript

1. Déclarer une fonction

1/12

Dans cet exercice vous allez découvrir ce qu'est une **fonction**, à quoi ça sert et comment déclarer une fonction.

À quoi ça sert ?

Quand vous allez commencer à écrire des scripts, vous allez vous rendre compte que certains morceaux de code font la même chose et sont répétés à plusieurs endroits dans votre script.

Ceci alourdit le code et n'est pas pratique à maintenir à jour. Si vous souhaitez modifier ce code qui est réutilisé un peu partout dans le script, il faudra le faire pour toutes ses occurrences. Pas pratique. C'est la que les fonctions interviennent.

Une **fonction** est une sorte de "boîte noire" dans laquelle est écrit une portion de code qui fait quelque chose. Une fonction porte **un nom** que vous aurez choisi (comme une variable). Il suffira d'appeler ce nom dans le script et le code correspondant sera exécuté.

La syntaxe d'une fonction

```
function maFonction () &#123;
    // Code a exécuter
}
```

Pour déclarer une fonction, il faut le mot-clé **function** suivi du nom que vous voulez donner à cette fonction. Puis un couple de parenthèses `()` dans lequel vous pourrez donner des paramètres / arguments qui seront utilisés dans la fonction (mais ils ne sont pas obligatoires). Entre les accolades `&#123;}` se trouve la portion de code à exécuter. Il n'y a pas de `;` à la fin : c'est une **structure** (comme les boucles et les conditions) et non une **instruction**.

Ici `maFonction` n'est pas exécutée, juste déclarée. Pour exécuter la fonction il faut l'appeler :

```
maFonction();
```

Fonction ou méthode ?

Sans le savoir, vous avez déjà utilisé des fonctions proposées nativement par Javascript : `alert()`, `prompt()`, `confirm()` ...

Il y a aussi `toUpperCase()` qui permet de mettre du texte en majuscule. C'est une fonction utilisée avec un objet de type chaîne de caractères (`string`).

Lors de l'introduction sur les **objets** on avait vu qu'un objet possédait des **propriétés** et des **méthodes**. Une méthode est en fait une fonction native d'un objet Javascript, comme `toUpperCase()` et plein d'autres.

Mais c'est exactement la même chose.

2. Notion de variable globale

2/12

Une fonction est une sorte de boîte noire dans lequel du code est exécuté. Donc ce code est en quelque sorte "isolé" du reste du script.

Si on déclare une variable dans le script, elle est accessible dans l'ensemble du script, c'est donc une variable **globale**.

Exemple :

```
// on déclare une variable globale
var maVariable = "Variable globale";
// on déclare une fonction
function test() &#123;
    alert(maVariable);
}
// on exécute la fonction
test();
```

Ici, la fonction `test()` va bien afficher "Variable globale" car la variable déclarée dans le script global est accessible dans la fonction.

3. Notion de variable locale

3/12

Si une fonction a accès à une variable **globale**, cela fonctionne-t-il dans le sens contraire ? Et bien non.

Une variable déclarée au sein même d'une fonction est une variable **locale** et elle n'est pas accessible dans le reste du script. Et ce pour la bonne raison que une fois la fonction exécutée, la variable locale est détruite.

Exemple :

```
// on déclare une fonction
function test() &#123;
    var localVar = "Variable locale";
    alert(localVar);
}
// on exécute la fonction
test();
// on affiche directement localVar
alert(localVar);
```

Ici, la fonction `test()` va bien afficher "Variable locale" car la fonction a accès à la variable. Mais le `alert()` qui se trouve dans le script global va renvoyer `undefined` car il n'a pas accès à la variable déclarée dans la fonction.

Il faut donc faire attention à la **portée** d'une variable.

Globale ou locale ?

Faut-il donc déclarer toutes les variables globalement ?

Non, car si une variable n'est utilisée que dans une fonction, alors il suffit jsute de la déclarer localement dans la fonction. Elle ne sera pas utile dans le reste du script.

4. Fonction et arguments

Si une fonction est en partie indépendante du reste du code, il serait quand même pratique de lui donner des informations dont elle pourrait avoir besoin pour exécuter son propre code.

Les arguments

Argument, ou paramètre, ou encore valeur, autant de mots pour désigner la même chose : une information passée en entrée à la fonction pour qu'elle s'en serve dans son code.

`alert()` est une fonction qui prend en paramètre ce qu'on souhaite qu'elle affiche.

La syntaxe d'une fonction avec un argument

```
function maFonction (arg) &#123;
    // Code a exécuter
}
```

Ici on a déclaré une fonction avec un paramètre en plus `arg`. Il est possible d'en indiquer plusieurs :

```
function maFonction (arg1,arg2,arg3) &#123;
    // Code a exécuter
}
```

Exemple :

```
// on déclare la fonction
function maFonction (prenom) &#123;
    alert('Bonjour '+prenom);
}
// on exécute la fonction
maFonction("Jean");
```

On indique à la fonction une chaîne de caractère en paramètre d'entrée.

La fonction va comprendre que pour elle `prenom` = `Jean`. Elle va donc se servir de ce paramètre comme une variable interne.

5. Fonction et valeur de retour

Une fonction peut prendre des arguments en entrée mais une fonction sert aussi à renvoyer un résultat (une valeur) en retour.

Cela peut servir si on a besoin de récupérer une valeur qui doit être utilisée dans le reste du script. Pour renvoyer un résultat, une fonction utilise le mot-clé `return`.

Comme une fonction est comme une "boîte noire" et qu'on ne peut pas accéder à ses variables locales. Le fait de pouvoir renvoyer une valeur qui peut être utilisée dans le reste du script est bien pratique.

Exemple :

```
// on déclare la fonction
function maFonction (texte) &#123;
    if (typeof texte == 'string')
    &#123;
        return texte;
    }
    else
    &#123;
        return false;
    }
}
// on exécute la fonction
var recup = maFonction("Jean");
```

Ici, la fonction demande un argument :

si l'argument donné est bien du texte alors la fonction retournera "Jean";
si ce n'est pas du texte la fonction retourne false;
selon le cas, la variable globale `recup` contient soit false soit "Jean".

Attention :

Dès que la fonction rencontre le mot-clé `return` elle s'interrompt et retourne la valeur. Si il y a du code après il ne sera pas exécuté.

Manipuler le DOM

LEÇON 1 Trouver un élément HTML

1. Le concept du DOM

1/8

L'un des avantages de Javascript est que c'est un langage exécutée du côté client, donc dans votre navigateur. De ce fait, Javascript a accès à de nombreuses informations sur l'environnement d'exécution d'une page web et donc la fenêtre du navigateur.

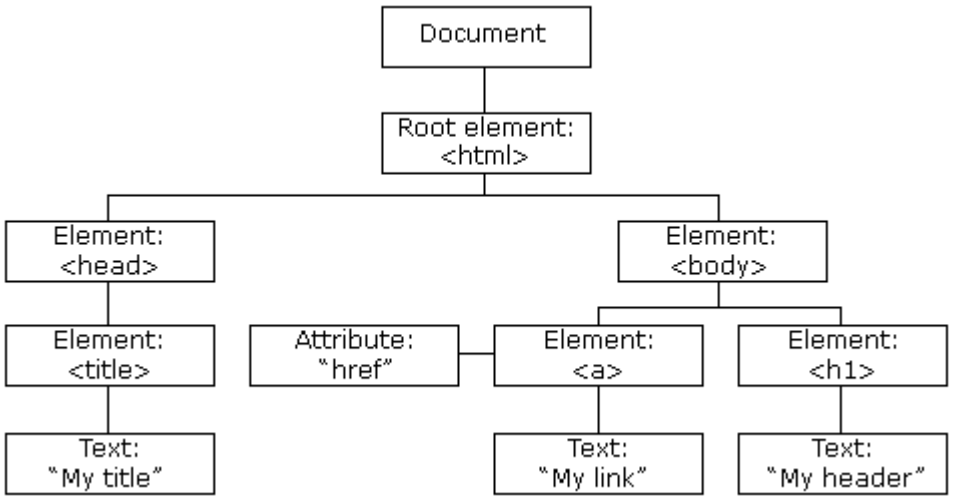
Pour Javascript, la fenêtre du navigateur est représentée par l'objet `window` et `alert()` est en fait une fonction (ou méthode) de `window` et l'on pourrait tout aussi bien écrire `window.alert()`. Idem pour `prompt()`, `confirm()`.

De même `window` possède des propriétés permettant de connaître la largeur et hauteur en pixels de la fenêtre du navigateur.

Ce qui nous intéresse ici, c'est l'objet `document` qui représente la structure HTML de la page web et tout ce qui est contenu dans `<html></html>`.

`document` possède aussi des fonctions permettant de manipuler la structure HTML (le DOM) d'une page web.

La structure du DOM



Le DOM est constitué par des noeuds (nodes en anglais) :

le noeud racine est `<html>`;
chaque noeud possède un parent (sauf le noeud racine);
chaque noeud peut avoir plusieurs enfants;
chaque noeud peut avoir des noeuds frères / soeurs.

Un noeud peut être une balise HTML mais aussi du texte contenu dans cette balise.

2. Récupérer l'élément par son nom

2/8

Dans cet exercice vous allez voir comment trouver un élément HTML par son nom de balise avec la fonction `.getElementsByTagName()`.

DOM :

```
<div>
    <p>Paragraphe</p>
</div>
```

Javascript :

```
var div = document.getElementsByTagName('div');
```

Ici la variable `div` contiendra un objet qui représente l'élément HTML div. Il faut préciser entre les parenthèses le nom de la balise HTML souhaitée.

3. Compter les éléments récupérés

3/8

Avec `getElementsByTagName(h1)` vous avez en fait récupéré une collection des éléments HTML présents dans le DOM.

Une collection est en fait une liste des éléments HTML stockés sous forme de tableau.

Dans cet exercice, vous allez récupérer en plus la longueur de la collection (ou du tableau c'est pareil).

4. Parcourir les éléments récupérés selon leur balise

4/8

Dans le fichier `script.js` :

Soit les variables `result` et `longueur` déja déclarées.

utilisez `document.getElementsByTagName()` pour récupérer les paragraphes **p** et affectez le résultat à `result`;
récupérez dans `longueur` la longueur du tableau;
créez une boucle `for` avec l'itérateur `i`;
dans la boucle affichez chaque élément avec `alert()`;;
cliquez sur **Soumettre ma réponse**

5. Parcourir les éléments récupérés selon leur class

5/8

Dans le fichier `script.js` :

déclarez les variables `result` et `longueur`;
utilisez `document.getElementsByClassName()` pour récupérer les éléments HTML qui ont la class "par" et affectez le résultat à `result`;
récupérez dans `longueur` la longueur du tableau;
créez une boucle `for` avec l'itérateur `i`;

dans la boucle affichez chaque élément avec `alert()`;
cliquez sur **Soumettre ma réponse**.

6. Récupérer les éléments selon leur id

6/8

Il est aussi possible de récupérer les éléments HTML selon leur id avec la fonction suivante :

```
.getElementById()
```

Elle fonctionne de la même manière que la fonction précédente. La différence est que **seul un élément** est retourné. En effet, un `id` en css est censé être unique. Si plusieurs éléments possèdent le même `id` alors cette fonction retournera que **le premier** élément trouvé dans le DOM.

Donc ici, pas besoin de boucle.

7. Récupérer un élément avec un sélecteur css

7/8

Il existe un moyen beaucoup plus flexible que les méthodes précédentes pour sélectionner un élément HTML dans le DOM :
`.querySelector()`

Cette fonction sélectionne un élément en utilisant les sélecteurs css. De ce fait les possibilités sont nombreuses. Il suffit de préciser en paramètres de la fonction un sélecteur css comme vous l'écririez dans un fichier .css
Exemple :

```
document.querySelector("div > p");
```

Ici, on sélectionne tous les `p` qui sont des enfants directs d'un `div`. Attention, cette fonction ne récupère que le premier élément HTML correspondant. Pas besoin de boucle ici pour parcourir le résultat.

8. Récupérer une liste d'éléments avec un sélecteur css

8/8

Pour récupérer une liste de plusieurs éléments HTML toujours en utilisant les sélecteurs css, il faut utiliser la fonction `.querySelectorAll()`.

Cette fonction récupère aussi une liste (ou collection) d'éléments (comme `getElementsByTagName()` et `getElementsByClassName()`). Il faut donc une boucle pour parcourir les résultats.

LEÇON 2 Modifier un élément HTML

1. Modifier le HTML d'un élément

1/7

Il est possible de récupérer ou de modifier le contenu d'un élément HTML avec la propriété `innerHTML`.

Récupérer le contenu HTML

```
var contenu = element.innerHTML;
```

Modifier le contenu HTML

```
element.innerHTML = "Nouveau contenu HTML (texte + balises)";
```

2. Modifier la valeur d'un attribut

2/7

Il est possible de modifier un attribut d'un élément HTML.

Modifier l'attribut d'un élément HTML

```
element.attribute = "Nouvelle valeur";
```

Ici `attribute` est à remplacer par le nom de l'attribut en question.

Exemple :

```
element.src = "monimage.png"
```

Ici on récupère un élément HTML (une image) et on change le lien vers le fichier en question.

3. Ajouter un attribut

3/7

Il est aussi possible d'ajouter un attribut à un élément HTML avec la fonction `.setAttribute()`.

Exemple :

```
element.setAttribute("class", "democlass");
```

Ici on récupère un élément HTML et on ajoute une class css appelée `democlass`.

4. Modifier une propriété css

4/7

Il est possible de modifier l'apparence d'un élément HTML.

Exemple :

```
element.style.fontSize = "1em";
```

Ici on récupère un élément HTML et on modifie la taille de son texte.

5. Sélecteur de photos 1/3

5/7 - 6/7 - 7/7

Une interface de sélecteur de photos a été mise en place graphiquement, nous souhaiterions cependant avoir une interaction.

Nous souhaiterions détecter le clique sur une photo pour rendre l'apparence comme sélectionnée de celle ci et de modifier la valeur numérique du nombre de photos sélectionnées dans le texte "Vous avez sélectionné N photo(s).

LEÇON 3 Ajouter et supprimer un élément HTML

1. Ajouter un élément HTML

1/3

Il est possible d'ajouter un nouvel élément HTML avec la fonction `.appendChild()`. Cette fonction ne s'utilise pas seule, elle est souvent associée à :

`.createElement()` pour créer une nouvelle balise non présente dans le DOM;
`.createTextNode()` pour ajouter du texte dans une balise;
une des fonctions vues pour **trouver** un élément dans le DOM;

Exemple :

```
// on créé une balise p
var element = document.createElement("p");
// on créé du texte
var texte = document.createTextNode("Voici du texte");
// on ajoute le texte dans l'élément p
var balise = element.appendChild(texte);
// on ajoute la balise p et son texte dans un div présent dans le DOM
document.querySelector("div").appendChild(balise);
```

Astuce :
getElementsByTagName() contient une collection (ou tableau) des éléments HTML trouvés. Pour savoir comment récupérer un élément au sein de la collection, revoyez les exercices de la catégorie "Trouver un élément HTML".

2. Supprimer un élément HTML

2/3

Si il est possible d'ajouter un élément enfant avec `.appendChild()`, il est aussi possible de supprimer un élément enfant avec `.removeChild()`. Pour utiliser cette fonction il faudra aussi :

une des fonctions vues pour **trouver** un élément dans le DOM auquel supprimer un enfant;
`childNodes`, cette propriété récupère une collection (ou tableau) de tous les éléments enfants d'un élément donné. On peut récupérer la longueur du tableau avec `length` et utiliser une boucle pour supprimer chaque élément;

Exemple :

```
// on sélectionne une liste "ul"
var liste = document.querySelector("ul");
// on supprime le premier élément enfant
liste.removeChild(liste.childNodes[0]);
// on cherche tous les enfants
var lis = liste.querySelectorAll("li");
// On supprime le 2ème enfants
liste.removeChild(lis[1]);
```

Dans l'exemple on a supprimé uniquement le premier enfant, si on veut tous les supprimer (surtout si il y en a beaucoup), il faut faire une boucle pour parcourir le tableau avec un itérateur.

3. Remplacer un élément HTML

3/3

Si il est possible d'ajouter et supprimer un élément enfant avec `.appendChild()` et `.removeChild()`. Il est aussi possible de remplacer un élément (balise HTML, texte) avec `replaceChild()`. Comme les deux autres, cette fonction ne s'utilise pas seule, il faut aussi :

une des fonctions vues pour **trouver** un élément dans le DOM auquel on souhaite remplacer l'élément enfant (balise ou texte);
`.childNodes()` pour récupérer l'élément enfant que l'on souhaite remplacer;
`.createElement()` ou `.createTextNode()`, pour créer le nouvel élément (balise ou texte) qui remplacera l'ancien;

Exemple :

```
// on créé un nouvel élément de type texte
var nouveau = document.createTextNode("Nouveau texte");
// on récupère le premier "li" de "ul" (1ère méthode au choix)
var li = document.querySelector("ul li");
// on récupère le premier "li" de "ul" (2ème méthode au choix)
var li = document.querySelectorAll("ul li").childNodes[0];
// on récupère le texte actuel (celui qu'il faut remplacer)
```

```
var ancien = li.childNodes[0];
// on remplace le texte du premier "li" par le nouveau texte
li.replaceChild(nouveau, ancien);
```

Bravo !! Fin