ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Développement Web:

JavaScript

Professeur:

Nouhaila MOUSSAMMI

n.moussammi@emsi.ma

2024/2025

Syntaxe et Structure

Syntaxe et Structure

JS JavaScript

Présentation des variables JavaScript : Déclaration

-> Une variable est un conteneur servant à stocker des informations de manière temporaire, comme une chaine de caractères (un texte) ou un nombre par exemple. Lorsqu'on stocke une valeur dans une variable, on dit également qu'on assigne une valeur à une variable.

-> Pour déclarer une variable en JavaScript, il faut utiliser le mot clef var ou le mot clef let.

-> Il faut respecter des règles de nomenclature :

- Le nom d'une variable doit obligatoirement commencer par une lettre ou un underscore (_) et ne doit pas commencer par un chiffre ;
- Le nom d'une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux ;
- Le nom d'une variable ne doit pas contenir d'espace.

-> Les noms des variables sont sensibles à la casse en JavaScript. (texte, TEXTE et tEXTe)

-> En JavaScript, certains noms sont réservés et ne peuvent pas être utilisés pour les variables, car ils sont déjà employés pour désigner des éléments du langage.

-> La convention **lower camel** case est généralement utilisée pour nommer les variables en JavaScript, où chaque mot après le premier commence par une majuscule (ex: monAge).

Syntaxe et Structure

Présentation des variables JavaScript : Initialisation

Initialiser une variable consiste à lui attribuer une valeur pour la première fois, soit au moment de sa déclaration, soit après.

-> Il est plus efficace d'initialiser une variable lors de sa déclaration, en utilisant **l'opérateur =** qui sert à assigner une valeur.

-> Le propre d'une variable et l'intérêt principal de celles-ci est de pouvoir stocker différentes valeurs.

D: > cours-javascript > JS cours.js > ...

```
1  // On déclare une variable et initialise la variable en même temps
2  let prenom = "Ali";
3
4
5  //On déclare une variable puis on l'initialise ensuite
6  let monAge;
7  monAge = 11;
8
```

D: > cours-javascript > JS cours.js > ...

```
1  // On déclare une variable et initialise la variable en même temps
2  let prenom = "Ali";
3
4
5  /* On modifie la valeur stockée dans prénom
6   * Notre variable stocke désormais la valeur "Mathilde"
7   */
8  prenom = "Mathilde";
9
```

Syntaxe et Structure

Présentation des variables JavaScript : Initialisation

JS JavaScript

La différence entre les mots clefs let et var

-> Lors de la création de JavaScript, et pendant de nombreuses années, le seul mot-clé disponible pour déclarer des variables était **var**.

-> Les créateurs de JavaScript ont introduit le mot-clé **let** pour remplacer **var**, jugé source de confusion. Ils en ont également profité pour résoudre certains problèmes associés à **var**.

Il existe trois principales différences entre les variables déclarées avec var et celles déclarées avec let, que nous allons illustrer :

- **La remontée ou « hoisting » des variables**
- **La redéclaration de variables**
- **La portée des variables**

Syntaxe et Structure

JS JavaScript

Présentation des variables JavaScript : Initialisation

- **La remontée ou « hoisting » des variables :**

Le "hoisting" (remontée) permettait aux variables déclarées avec var d'être utilisées avant leur déclaration dans le code, car JavaScript traitait ces déclarations en premier. Ce comportement, jugé inadapté, a été corrigé avec let, où les variables doivent être déclarées avant d'être utilisées. Cette modification vise à encourager des scripts plus structurés et clairs.

D: > cours-javascript > JS cours.js > ...

```
1   //Ceci fonctionne
2   prenom = "Ali";
3   var prenom;
4
5
6   //Ceci ne fonctionne pas et renvoie une erreur
7   monAge = 22;
8   let monAge;
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "d:\cours-javascript\cours.js"
d:\cours-javascript\cours.js:7
monAge = 22;
       ^

ReferenceError: Cannot access 'monAge' before initialization
    at Object.<anonymous> (d:\cours-javascript\cours.js:7:8)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:75:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.281 seconds
```

Syntaxe et Structure

JS JavaScript

Présentation des variables JavaScript : Initialisation

- **La redéclaration de variables**

Avec **var**, il était possible de redéclarer plusieurs fois la même variable, ce qui modifiait sa valeur. Avec **let**, cela n'est plus autorisé : il suffit d'affecter une nouvelle valeur à la variable sans la redéclarer. Ce changement a été fait pour améliorer la clarté et la pertinence du code, évitant ainsi des redéfinitions inutiles.

D: > cours-javascript > JS cours.js > ...

```
1   //Ceci fonctionne
2   var prenom = "Ali";
3   var prenom = "Mathilde";
4
5
6   //Ceci ne fonctionne pas et renvoie une erreur
7   let monAge = 22;
8   let monAge = 14;
9
```

```
[Running] node "d:\cours-javascript\cours.js"
d:\cours-javascript\cours.js:8
let monAge = 14;
    ^

SyntaxError: Identifier 'monAge' has already been declared
    at wrapSafe (internal/modules/cjs/loader.js:1001:16)
    at Module._compile (internal/modules/cjs/loader.js:1049:27)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:75:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.233 seconds
```

Syntaxe et Structure

Les types de données en JavaScript

JS JavaScript

-> Les variables en JavaScript peuvent stocker différents types de valeurs, comme du texte ou des nombres.

-> Il n'est pas nécessaire de préciser à l'avance le type de valeur qu'une variable va stocker, contrairement à d'autres langages de programmation.

-> JavaScript détecte automatiquement le type de la valeur stockée dans une variable.

-> On peut ensuite effectuer des opérations adaptées au type de la variable, ce qui est très pratique.

-> Une même variable peut contenir des valeurs de différents types au fil du temps, sans se préoccuper de la compatibilité.

-> Par exemple, une variable peut stocker une chaîne de caractères à un moment donné, puis un nombre à un autre moment.

D: > cours-javascript > JS cours.js > ...

```
1   //Ceci fonctionne
2   let variable = 25;
3
4   variable = "Ali";
5
6   variable = [1,2,3];
7
```

Syntaxe et Structure

Les types de données en JavaScript

JS JavaScript

En JavaScript, il existe 7 types de valeurs différents. Chaque valeur qu'on va pouvoir créer et manipuler en JavaScript va obligatoirement appartenir à l'un de ces types. Ces types sont les suivants :

- **String** ou « chaine de caractères » en français ;
- **Number** ou « nombre » en français ;
- **Boolean** ou « booléen » en français ;
- **Null** ou « nul / vide » en français;
- **Undefined** ou « indéfini » en français ;
- **Symbol** ou « symbole » en français ;
- **Object** ou « objet » en français ;

Il est crucial de connaître le type des données, car elles sont manipulées différemment selon leur type, ce qui permet de créer des scripts fonctionnels.

Syntaxe et Structure

Les types de données en JavaScript

JS JavaScript

❖ Le type chaîne de caractères ou string:

Le premier type de données qu'une variable va pouvoir stocker est le type String ou chaîne de caractères. Une chaine de caractères est une séquence de caractères, ou ce qu'on appelle communément un texte.

Toute valeur stockée dans une variable en utilisant des guillemets ou des apostrophes sera considérée comme une chaine de caractères, et ceci même dans le cas où nos caractères sont à priori des chiffres comme "29" par exemple.

D: > cours-javascript > JS cours.js > ...

```
1   let prenom = "Je m'appelle Pierre";
2   let age = 29;
3   let age2 = '29';
4
5   console.log('Type de prenom : ' + typeof prenom);
6   console.log('Type de age : ' + typeof age);
7   console.log('Type de age2 : ' + typeof age2);
8
```

```
[Running] node "d:\cours-javascript\cours.js"
Type de prenom : string
Type de age : number
Type de age2 : string

[Done] exited with code=0 in 0.179 seconds
```

Syntaxe et Structure

Les types de données en JavaScript

❖ Le type nombre ou number

Les variables en JavaScript vont également pouvoir stocker des nombres. En JavaScript, et contrairement à la majorité des langages, il n'existe qu'un type prédéfini qui va regrouper tous les nombres qu'ils soient positifs, négatifs, entiers ou décimaux (à virgule) et qui est le type **Number**.

D: > cours-javascript > JS cours.js > ...

```
1   let x = 12;
2   let y = 3.14;
3   let z = -2;
4
5
6   console.log("Le type de x est : " + typeof x);
7   console.log("Le type de y est : " + typeof y);
8   console.log("Le type de z est : " + typeof z);
9
```

```
[Running] node "d:\cours-javascript\cours.js"
Le type de x est : number
Le type de y est : number
Le type de z est : number

[Done] exited with code=0 in 0.168 seconds
```

Attention ici : lorsque nous codons nous utilisons toujours des notations anglo-saxonnes, ce qui signifie qu'il faut remplacer nos virgules par des points pour les nombres décimaux.

Syntaxe et Structure

Les types de données en JavaScript

❖ Le type de données booléen (boolean)

Une variable en JavaScript peut encore stocker une valeur de type Boolean, c'est-à-dire un booléen.

Un booléen, en algèbre, est une valeur binaire (soit 0, soit 1). En informatique, le type booléen est un type qui ne contient que deux valeurs : les valeurs **true** (vrai) et **false** (faux).

D: > cours-javascript > JS cours.js > ...

```
1   let vrai = true;
2   let faux = false;
3
4   let resultat = 8>4;
5
6   console.log("Le type de vrai est : " + typeof vrai);
7   console.log("Le type de faux est : " + typeof faux);
8   console.log("La valeur affecté à résultat est : " + resultat);
9
```

```
[Running] node "d:\cours-javascript\cours.js"
Le type de vrai est : boolean
Le type de faux est : boolean
La valeur affecté à résultat est : true

[Done] exited with code=0 in 0.267 seconds
```

Syntaxe et Structure

Les types de données en JavaScript

JS JavaScript

❖ Les types de valeurs null et undefined

Les types **Null** et **Undefined** sont particuliers, car chacun ne contient qu'une seule valeur : null et undefined.

null indique explicitement l'absence de valeur connue et doit être assigné de manière explicite, tandis que **undefined** désigne une variable à laquelle aucune valeur n'a été affectée.

D: > cours-javascript > JS cours.js > ...

```
1   let nul = null;
2   let ind;
3
4   console.log('Type de nul : ' + typeof nul);
5   console.log('Type de ind : ' + typeof ind);
6
```

```
[Running] node "d:\cours-javascript\cours.js"
Type de nul : object
Type de ind : undefined

[Done] exited with code=0 in 0.177 seconds
```

Syntaxe et Structure

Les opérateurs en Javascript

JS JavaScript

Qu'est-ce qu'un opérateur ?

Un opérateur est un symbole qui va être utilisé pour effectuer certaines actions notamment sur les variables et leurs valeurs.

Par exemple, l'opérateur * va nous permettre de multiplier les valeurs de deux variables, tandis que l'opérateur = va nous permettre d'affecter une valeur à une variable.

Il existe différents types d'opérateurs qui vont nous servir à réaliser des opérations de types différents. Les plus fréquemment utilisés sont :

- Les opérateurs arithmétiques ;
- Les opérateurs d'assignation ;
- Les opérateurs de comparaison ;
- Les opérateurs d'incrémentation et décrémentation ;
- Les opérateurs logiques ;
- L'opérateur ternaire ;
- L'opérateur de concaténation :

Syntaxe et Structure

Les opérateurs en Javascript

JS JavaScript

Les opérateurs arithmétiques :

Les opérateurs arithmétiques vont nous permettre d'effectuer toutes sortes d'opérations mathématiques entre les valeurs de type Number contenues dans différentes variables.

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
+	opérateur d'addition	Ajoute deux valeurs	x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	x*3	21
/	plus: opérateur de division	Divise deux valeurs	x/3	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	x=3	Met la valeur 3 dans la variable x

Syntaxe et Structure

Les opérateurs en Javascript

JS JavaScript

Les opérateurs d'affectation :

Les opérateurs d'affectation vont nous permettre, comme leur nom l'indique, d'affecter une certaine valeur à une variable.

Opérateur	Effet
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
*=	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le résultat dans la variable

Syntaxe et Structure

JS JavaScript

Les opérateurs arithmétiques et d'affectation

Les opérateurs de comparaison ;

Les opérateurs de comparaison nous permettent, comme leur nom l'indique, de comparer deux valeurs afin de déterminer si elles sont égales, différentes ou si l'une est supérieure ou inférieure à l'autre.

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
== A ne pas confondre avec le signe d'affectation (=)!!	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	Retourne 1 si X est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	Retourne 1 si X est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	Retourne 1 si X est inférieur à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x>3	Retourne 1 si X est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	Retourne 1 si X est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x!=3	Retourne 1 si X est différent de 3, sinon 0

Syntaxe et Structure

JS JavaScript

Les opérateurs arithmétiques et d'affectation

Utilisation des opérateurs de comparaison ;

D: > cours-javascript > JS cours.js > ...

```
1   let x = 4; //On stocke le chiffre 4 dans x
2
3   /*Les comparaisons sont effectuées avant l'affectation. Le JavaScript va donc
4    *commencer par comparer et renvoyer true ou false et nous allons stocker ce
5    *résultat dans nos variables test*/
6   let test1 = x == 4;
7   let test2 = x === 4;
8   let test3 = x == '4';
9   let test4 = x === '4';
10  let test5 = x != '4';
11  let test6 = x !== '4';
12  let test7 = x > 4;
13  let test8 = x >= 4;
14  let test9 = x < 4;
15
16  console.log('Valeur dans x égale à 4 (en valeur) ? : ' + test1 +
17          '\nValeur dans x égale à 4 (valeur & type) ? : ' + test2 +
18          '\nValeur dans x égale à "4" (en valeur) ? : ' + test3 +
19          '\nValeur dans x égale à "4" (valeur & type) ? : ' + test4 +
20          '\nValeur dans x différente de "4" (en valeur) ? : ' + test5 +
21          '\nValeur dans x différente de "4" (valeur & type) ? : ' + test6 +
22          '\nValeur dans x strictement supérieure à 4 ? : ' + test7 +
23          '\nValeur dans x supérieure ou égale à 4 ? : ' + test8 +
24          '\nValeur dans x strictement inférieure à 4 ? : ' + test9);
25
```

```
[Running] node "d:\cours-javascript\cours.js"
Valeur dans x égale à 4 (en valeur) ? : true
Valeur dans x égale à 4 (valeur & type) ? : true
Valeur dans x égale à "4" (en valeur) ? : true
Valeur dans x égale à "4" (valeur & type) ? : false
Valeur dans x différente de "4" (en valeur) ? : false
Valeur dans x différente de "4" (valeur & type) ? : true
Valeur dans x strictement supérieure à 4 ? : false
Valeur dans x supérieure ou égale à 4 ? : true
Valeur dans x strictement inférieure à 4 ? : false

[Done] exited with code=0 in 0.297 seconds
```

Syntaxe et Structure

JS JavaScript

Les opérateurs en Javascript

Les opérateurs d'incrémentation et décrémentation ;

Les opérateurs d'incrémentation et de décrémentation permettent d'augmenter ou de diminuer la valeur d'une variable de manière simple et rapide.

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable.

Un opérateur de type x++ permet de remplacer des notations lourdes telles que x=x+1 ou bien x+=1

Opérateur	Dénomination	Effet	Syntaxe	Résultat (avec x valant 7)
++	Incrémentation	Augmente d'une unité la variable	x++	8
--	Décrémentation	Diminue d'une unité la variable	x--	6

Syntaxe et Structure

Les opérateurs en Javascript

JS JavaScript

Les opérateurs logiques :

Ils sont utilisés pour combiner ou inverser des expressions booléennes, facilitant ainsi la prise de décisions dans le code.

Ils permettent de vérifier si plusieurs conditions sont vraies:

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

Syntaxe et Structure

JS JavaScript

Les opérateurs en Javascript

L'opérateur ternaire ;

L'opérateur ternaire est une forme concise de structure conditionnelle qui permet d'évaluer une expression et de retourner une valeur basée sur un test booléen.

Il se compose de trois parties : **une condition**, **une valeur** si la condition est **vraie**, et **une valeur** si la condition est **fausse**. Sa syntaxe est la suivante :

condition ? valeurSiVraie : valeurSiFausse ;

Cela permet d'écrire des conditions simples de manière plus compacte, rendant le code plus lisible.

D: > cours-javascript > JS cours.js > ...

```
1   const nom = 'Julien'
2
3   // Condition ternaire ou condition if else en une ligne :
4   const color = (nom === 'Julien')? 'blue' : 'green'
5   // Si la variable name est égale à "Julien", la couleur est "blue" sinon, elle est "green"
6
7   console.log(color)
8   // Sortie : "blue"
9
```

Syntaxe et Structure

Les opérateurs en Javascript

L'opérateur de concaténation :

Concaténer signifie littéralement « mettre bout à bout ». l'opérateur de concaténation est le signe **+**. Faites bien attention ici : lorsque le signe **+** est utilisé avec deux nombres, il sert à les additionner. Lorsqu'il est utilisé avec autre chose que deux nombres, il sert d'opérateur de concaténation.

D: > cours-javascript > JS cours.js > ...

```
1   let x = 28 + 1; //Le signe "+" est ici un opérateur arithmétique
2   let y = 'Bonjour';
3   let z = x + ' ans'; //Le signe "+" est ici un opérateur de concaténation
4
5
6   console.log(y + ', je m\'appelle Pierre, j\'ai ' + z);
7
```

```
[Running] node "d:\cours-javascript\cours.js"
Bonjour, je m'appelle Pierre, j'ai 29 ans

[Done] exited with code=0 in 0.161 seconds
```

Syntaxe et Structure

Les constantes

Une constante est similaire à une variable au sens où c'est également un conteneur pour une valeur. Cependant, à la différence des variables, on ne va pas pouvoir modifier la valeur d'une constante.

D: > cours-javascript > JS cours.js > ...

```
1   const prenom = "Pierre";
2   const age = 27;
3
4   //Ceci déclence une erreur
5   age = 24;
6
```

```
[Running] node "d:\cours-javascript\cours.js"
d:\cours-javascript\cours.js:4
age = 24;
    ^

TypeError: Assignment to constant variable.
    at Object.<anonymous> (d:\cours-javascript\cours.js:4:5)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:75:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.177 seconds
```

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

JS JavaScript

Définition :

Les tableaux sont des éléments qui vont pouvoir contenir plusieurs valeurs. En JavaScript, comme les tableaux sont avant tout des objets, il peut paraitre évident qu'un tableau va pouvoir contenir plusieurs valeurs comme n'importe quel objet.

-> Le principe des tableaux est relativement simple : un indice ou clef va être associé à chaque valeur du tableau. Pour récupérer une valeur dans le tableau, on va utiliser les indices qui sont chacun unique dans un tableau.

-> Les tableaux ne sont pas des valeurs primitives. Cependant, nous ne sommes pas obligés d'utiliser le constructeur Array() avec le mot clef new pour créer un tableau en JavaScript.

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

JS JavaScript

Création d'un tableau en JavaScript :

JS cours.js > ...

```
1   // Création d'un tableau contenant des nombres
2   const nombres = [1, 2, 3, 4, 5];
3
4   // Création d'un tableau contenant des chaînes de caractères
5   const fruits = ['Pomme', 'Banane', 'Cerise', 'Orange'];
6
7   // Création d'un tableau contenant un autre tableau
8   const tableau2D = [
9       [1, 2, 3],
10      [4, 5, 6],
11      [7, 8, 9]
12    ];
13
14  // Création d'un tableau contenant des valeurs de types variés
15  const diverseArray = [42, 'Hello', true, null, [1, 2]];
16
```

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

Accéder à une valeur dans un tableau

Lorsqu'on crée un tableau, un indice est automatiquement associé à chaque valeur du tableau. Chaque indice dans un tableau est toujours unique et permet d'identifier et d'accéder à la valeur qui lui est associée. Pour chaque tableau, l'indice 0 est automatiquement associé à la première valeur, l'indice 1 à la deuxième et etc.

JS cours.js > ...

```
1   let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
2   let ages = [29, 27, 29, 30];
3   let produits = ['Livre', 20, 'Ordinateur', 5, ['Magnets', 100]];
4
5
6
7   // Utilisation de console.log au lieu de innerHTML
8   console.log(prenoms[0] + ' possède 1 ' + produits[2]);
9   console.log(prenoms[1] + ' a ' + ages[1] + ' ans');
10  console.log(produits[4][1] + ' ' + produits[4][0]);
11
```

```
[Running] node "d:\cours-javascript\cours.js"
Pierre possède 1 Ordinateur
Mathilde a 27 ans
100 Magnets

[Done] exited with code=0 in 0.144 seconds
```

Syntaxe et Structure

JS JavaScript

Les tableaux en JavaScript et l'objet global Array

Les propriétés et les méthodes du Array() :

Le constructeur Array() en JavaScript possède deux propriétés, length pour le nombre d'éléments d'un tableau et prototype, ainsi qu'une trentaine de méthodes utiles, dont certaines sont particulièrement puissantes.

❖ **La propriété length :** Elle retourne le nombre d'éléments d'un tableau

```
JS cours.js > ...
1   let ages = [29, 27, 29, 30];
2   console.log(ages.length);
3
```

```
[Running] node "d:\cours-javascript\cours.js"
4

[Done] exited with code=0 in 0.173 seconds
```

❖ **La méthode push()** va nous permettre d'ajouter des éléments en fin de tableau et va retourner la nouvelle taille du tableau.

```
JS cours.js > ...
1   let prenoms = ['Pierre', 'Mathilde'];
2   let taille = prenoms. push('Florian', 'Camille');
3
4   console.log("La nouvelle taille de prenoms est : " + taille);
5
```

```
[Running] node "d:\cours-javascript\cours.js"
La nouvelle taille de prenoms est : 4

[Done] exited with code=0 in 0.15 seconds
```

Syntaxe et Structure

JS JavaScript

Les tableaux en JavaScript et l'objet global Array

Les propriétés et les méthodes du Array() :

- **La méthode pop()** va elle nous permettre de supprimer le dernier élément d'un tableau et va retourner l'élément supprimé.

```
JS cours.js > ...
1   let ages = [29, 27, 32];
2   let age = ages.pop();
3
4   console.log("La valeur supprimée est : " + age);
```

```
[Running] node "d:\cours-javascript\cours.js"
La valeur supprimée est : 32

[Done] exited with code=0 in 0.177 seconds
```

- **La méthode join()** retourne une chaine de caractères créée en concaténant les différentes valeurs d'un tableau. Le séparateur utilisé par défaut sera la virgule mais nous allons également pouvoir passer le séparateur de notre choix en argument de join().

```
JS cours.js > ...
1   let ages = [29, 27, 32];
2   let resultat1 = ages.join();
3   let resultat2 = ages.join(' - ')
4
5   console.log("Le premier résultat de join : " + resultat1);
6   console.log("Le deuxième résultat de join : " + resultat2);
```

```
[Running] node "d:\cours-javascript\cours.js"
Le premier résultat de join : 29,27,32
Le deuxième résultat de join : 29 - 27 - 32

[Done] exited with code=0 in 0.155 seconds
```

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

Les propriétés et les méthodes du Array() :

- **La méthode forEach() :** Elle est utilisée pour exécuter une fonction fournie une fois pour chaque élément d'un tableau. Elle est particulièrement utile pour effectuer des opérations sur chaque élément du tableau.

```
JS cours.js > ...
1   const fruits = ['Pomme', 'Banane', 'Cerise', 'Fraise'];
2
3   // Utilisation de forEach pour afficher chaque fruit
4   fruits.forEach(function(fruit, index) {
5       console.log(index + ': ' + fruit);
6   });
7
```

```
[Running] node "d:\cours-javascript\cours.js"
0: Pomme
1: Banane
2: Cerise
3: Fraise

[Done] exited with code=0 in 0.152 seconds
```

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

JS JavaScript

D'autres méthodes du Array() :

❖ **shift() :** Supprime le premier élément d'un tableau et retourne cet élément.

```
let fruits = ["pomme", "banane", "cerise"];
let premier = fruits.shift();
console.log(premier); // Affiche: "pomme"
console.log(fruits); // Affiche: ["banane", "cerise"]
```

❖ **unshift() :** Ajoute un ou plusieurs éléments au début d'un tableau et retourne la nouvelle longueur du tableau.

```
let fruits = ["banane", "cerise"];
let longueur = fruits.unshift("pomme", "orange");
console.log(longueur); // Affiche: 4
console.log(fruits); // Affiche: ["pomme", "orange", "banane", "cerise"]
```

Syntaxe et Structure

JS JavaScript

Les tableaux en JavaScript et l'objet global Array

❖ **slice() :** Renvoie une copie superficielle d'une portion d'un tableau dans un nouveau tableau, sans modifier le tableau d'origine.

```
let fruits = ["pomme", "banane", "cerise", "orange"];
let copie = fruits.slice(1, 3);
console.log(copie); // Affiche: ["banane", "cerise"]
console.log(fruits); // Affiche: ["pomme", "banane", "cerise", "orange"]
```

❖ **map() :** Crée un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau d'origine.

```
let nombres = [1, 2, 3, 4];
let doubles = nombres.map(function (nombre) {
  return nombre * 2;
});
console.log(doubles); // Affiche: [2, 4, 6, 8]
console.log(nombres); // Affiche: [1, 2, 3, 4]
```

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

JS JavaScript

❖ **some() :** Teste si au moins un élément du tableau passe le test implémenté par la fonction fournie. Elle retourne un booléen.

```
let nombres = [1, 3, 5, 7];
let aDesPairs = nombres.some(function (nombre) {
  return nombre % 2 === 0; // Vérifie s'il y a des nombres pairs
});
console.log(aDesPairs); // Affiche: false
```

❖ **every() :** Teste si tous les éléments du tableau passent le test implémenté par la fonction fournie. Elle retourne également un booléen.

```
let nombres = [2, 4, 6, 8];
let tousPairs = nombres.every(function (nombre) {
  return nombre % 2 === 0; // Vérifie si tous les nombres sont pairs
});
console.log(tousPairs); // Affiche: true
```

Syntaxe et Structure

Les tableaux en JavaScript et l'objet global Array

❖ **filter() :** Crée un nouveau tableau contenant tous les éléments du tableau d'origine qui passent un test fourni par une fonction.

```
let nombres = [1, 2, 3, 4, 5];
let pairs = nombres.filter(function (nombre) {
  return nombre % 2 === 0;
});
console.log(pairs); // Affiche: [2, 4]
console.log(nombres); // Affiche: [1, 2, 3, 4, 5]
```

❖ **find() :** Renvoie la première valeur trouvée dans le tableau qui satisfait une fonction de test fournie. Si aucune valeur n'est trouvée, elle retourne undefined.

```
let nombres = [1, 2, 3, 4, 5];
let premierPair = nombres.find(function (nombre) {
  return nombre % 2 === 0; // Cherche le premier nombre pair
});
console.log(premierPair); // Affiche: 2

```

Syntaxe et Structure

JS JavaScript

Les tableaux en JavaScript et l'objet global Array

Autres Méthodes:

- **findIndex()** : retourne l'index du premier élément qui satisfait une condition donnée, ou -1 si aucun élément ne la satisfait.
- **sort()**: trie les éléments d'un tableau en place et retourne le tableau trié. Par défaut, il trie les éléments comme des chaînes de caractères.
- **concat()** : fusionne deux ou plusieurs tableaux et retourne un nouveau tableau.
- **splice() :** Change le contenu d'un tableau en supprimant ou en remplaçant des éléments existants et/ou en ajoutant de nouveaux éléments à la place. Elle retourne les éléments supprimés.

Syntaxe et Structure

Structures de contrôle

JS JavaScript

On appelle « structure de contrôle » un ensemble d’instructions qui permet de contrôler l’exécution du code.

Il existe deux principaux types de structures de contrôle que l'on retrouve dans la plupart des langages de programmation, y compris JavaScript :

1. **Structures de contrôle conditionnelles (ou simplement « conditions ») :** Elles permettent d'exécuter un ensemble d'instructions uniquement si une certaine condition est remplie.
2. **Structures de contrôle de boucles (ou simplement « boucles ») :** Elles permettent d'exécuter un bloc de code de manière répétée tant qu'une condition donnée reste vraie.

Syntaxe et Structure

Structures de contrôle

JS JavaScript

Les conditions en JavaScript :

Les structures de contrôle conditionnelles (ou plus simplement conditions) vont nous permettre d'exécuter une série d'instructions si une condition donnée est vérifiée ou (éventuellement) une autre série d'instructions si elle ne l'est pas.

En JavaScript, nous disposons des structures conditionnelles suivantes :

- **La condition if (si) :** Elle permet d'exécuter un bloc de code uniquement si une condition est vraie.
- **La condition if... else (si... sinon) :** Elle permet d'exécuter un bloc de code si la condition est vraie et un autre bloc si elle est fausse.
- **La condition if... else if... else (si... sinon si... sinon) :** Elle permet de tester plusieurs conditions successives, en exécutant le bloc de code correspondant à la première condition vraie.
- **L'instruction switch**

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les conditions en JavaScript :

-> La condition if

D: > cours-javascript > JS cours.js > ...

```
1   let x = 4;
2   let y = 0;
3
4   if(x > 1){
5       console.log('x contient une valeur strictement supérieure à 1');
6   }
7
8   if(x == y){
9       console.log('x et y contiennent la même valeur');
10  }
11
12  if(y){
13      console.log('La valeur de y est évaluée à true');
14  }
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "d:\cours-javascript\cours.js"
x contient une valeur strictement supérieure à 1

[Done] exited with code=0 in 0.155 seconds
```

Syntaxe et Structure

Structures de contrôle

JS JavaScript

Les conditions en JavaScript :

-> La condition if…..else

D: > cours-javascript > JS cours.js > ...

```
1   let x = 0.5;
2
3   if(x > 1){
4       console.log('x contient une valeur strictement supérieure à 1');
5   }else{
6       console.log('x contient une valeur inférieure ou égale à 1');
7   }
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "d:\cours-javascript\cours.js"
x contient une valeur inférieure ou égale à 1

[Done] exited with code=0 in 0.173 seconds
```

Syntaxe et Structure

Structures de contrôle

JS JavaScript

Les conditions en JavaScript :

-> La condition if…else if…else

D: > cours-javascript > JS cours.js > ...

```
1   let x = 0.5;
2
3   if(x > 1){
4       console.log('x contient une valeur strictement supérieure à 1');
5   }else if(x == 1){
6       console.log('x contient la valeur 1');
7   }else{
8       console.log('x contient une valeur strictement inférieure à 1');
9   }
10
```

```
[Running] node "d:\cours-javascript\cours.js"
x contient une valeur strictement inférieure à 1

[Done] exited with code=0 in 0.159 seconds
```

D: > cours-javascript > JS cours.js > ...

```
1   let x = 1;
2
3   if(x > 1){
4       console.log('x contient une valeur strictement supérieure à 1');
5   }else if(x == 1){
6       console.log('x contient la valeur 1');
7   }else{
8       console.log('x contient une valeur strictement inférieure à 1');
9   }
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "d:\cours-javascript\cours.js"
x contient la valeur 1

[Done] exited with code=0 in 0.152 seconds
```

Syntaxe et Structure

Structures de contrôle

JS JavaScript

Les conditions en JavaScript :

-> L'instruction switch

L'instruction **switch** va nous permettre d'exécuter un code en fonction de la valeur d'une variable. On va pouvoir gérer autant de situations ou de « cas » que l'on souhaite.

En cela, l'instruction switch représente une alternative à l'utilisation d'un if…else if…else.

```
[Running] node "d:\cours-javascript\cours.js"
x stocke la valeur 15

[Done] exited with code=0 in 0.33 seconds
```

D: > cours-javascript > JS cours.js > ...

```
let x = 15;

switch(x){
    case 2:
        console.log('x stocke la valeur 2');
        break;
    case 5:
        console.log('x stocke la valeur 5');
        break;
    case 10:
        console.log('x stocke la valeur 10');
        break;
    case 15:
        console.log('x stocke la valeur 15');
        break;
    case 20:
        console.log('x stocke la valeur 20');
        break;
    default:
        console.log('x ne stocke ni 2, ni 5, ni 10, ni 15 ni 20');
}

```

Syntaxe et Structure

Structures de contrôle

JS JavaScript

Les boucles en JavaScript :

Les boucles permettent d'exécuter plusieurs fois un bloc de code, c'est-à-dire de faire tourner un code « en boucle » tant qu'une condition donnée est vérifiée.

Cela nous fait gagner un temps précieux lors de l'écriture des scripts. L'utilisation d'une boucle permet de ne rédiger le code à exécuter qu'une seule fois.

En JavaScript, nous avons trois types de boucles :

- **La boucle while (« tant que ») :** Elle exécute un bloc de code tant qu'une condition est vraie.
- **La boucle do... while (« faire... tant que ») :** Elle exécute un bloc de code au moins une fois, puis continue tant qu'une condition est vraie.
- **La boucle for (« pour ») :** Elle permet d'exécuter un bloc de code un nombre précis de fois, en utilisant un compteur.
- **La boucle for...of :** utilisée pour itérer sur les éléments d'un tableau et tout type itérable

Syntaxe et Structure

Structures de contrôle

JS JavaScript

Les boucles en JavaScript :

Pour éviter de rester bloqué à l'infini dans une boucle, il faut que la condition donnée soit fausse à un moment donné (pour pouvoir sortir de la boucle).

Les boucles vont donc être essentiellement composées de trois choses :

- **Une valeur :** de départ qui va nous servir à initialiser notre boucle et nous servir de compteur ;
- **Un test ou une condition de sortie :** qui précise le critère de sortie de la boucle ;
- **Un itérateur :** qui va modifier la valeur de départ de la boucle à chaque nouveau passage jusqu'au moment où la condition de sortie est vérifiée. Bien souvent, on incrémentera la valeur de départ.

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les boucles en JavaScript :

-> La boucle JavaScript while

D: > cours-javascript > JS cours.js > ...

```
1   //On initialise une variable let x
2   let x = 0
3
4   //Tant que...
5   while(x < 10){
6       //...exécute ce code
7       console.log('x stocke la valeur ' + x + ' lors du passage n°'+ (x + 1) + ' dans la boucle');
8
9       //faire incrémenter le valeur de x
10      x++;
11  }
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "d:\cours-javascript\cours.js"
x stocke la valeur 0 lors du passage n°1 dans la boucle
x stocke la valeur 1 lors du passage n°2 dans la boucle
x stocke la valeur 2 lors du passage n°3 dans la boucle
x stocke la valeur 3 lors du passage n°4 dans la boucle
x stocke la valeur 4 lors du passage n°5 dans la boucle
x stocke la valeur 5 lors du passage n°6 dans la boucle
x stocke la valeur 6 lors du passage n°7 dans la boucle
x stocke la valeur 7 lors du passage n°8 dans la boucle
x stocke la valeur 8 lors du passage n°9 dans la boucle
x stocke la valeur 9 lors du passage n°10 dans la boucle

[Done] exited with code=0 in 0.172 seconds
```

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les boucles en JavaScript :

-> La boucle JavaScript do... while

D: > cours-javascript > JS cours.js > ...

```
1   // Initialisation d'une variable pour compter les itérations
2   let compteur = 0;
3
4   // La boucle do... while commence ici
5   do {
6       // Affiche le compteur actuel dans la console
7       console.log("Compteur actuel : " + compteur);
8
9       // Incrémente le compteur de 1
10      compteur++;
11
12      // La condition est vérifiée ici. La boucle continuera tant que le compteur est inférieur à 5
13  } while (compteur < 5);
14
15  // Lorsque le compteur atteint 5, la boucle s'arrête et le programme continue
16  console.log("La boucle est terminée.");
17
```

```
[Running] node "d:\cours-javascript\cours.js"
Compteur actuel : 0
Compteur actuel : 1
Compteur actuel : 2
Compteur actuel : 3
Compteur actuel : 4
La boucle est terminée.

[Done] exited with code=0 in 0.267 seconds
```

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les boucles en JavaScript :

-> La boucle JavaScript for

D: > cours-javascript > JS cours.js > ...

```
1   // La boucle for commence ici
2   for (let i = 0; i < 5; i++) {
3       // Affiche la valeur actuelle de i dans la console
4       console.log("Valeur de i : " + i);
5   }
6
7   // Après la boucle, on affiche un message indiquant que la boucle est terminée
8   console.log("La boucle for est terminée.");
9
```

```
[Running] node "d:\cours-javascript\cours.js"
Valeur de i : 0
Valeur de i : 1
Valeur de i : 2
Valeur de i : 3
Valeur de i : 4
La boucle for est terminée.

[Done] exited with code=0 in 0.176 seconds
```

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les boucles en JavaScript :

-> La boucle JavaScript for... of

```
JS cours.js > ...
1   const couleurs = ['Rouge', 'Vert', 'Bleu', 'Jaune'];
2
3   // Utilisation de for...of pour afficher chaque couleur
4   for (const couleur of couleurs) {
5       console.log(couleur);
6   }
7
```

```
[Running] node "d:\cours-javascript\cours.js"
Rouge
Vert
Bleu
Jaune

[Done] exited with code=0 in 0.162 seconds
```

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les boucles en JavaScript :

-> Intégration avec les instructions continue et break

- **Continue :** Pour sauter une itération de boucle et passer directement à la suivante, on peut utiliser une instruction continue. Cette instruction va nous permettre de sauter l'itération actuelle et de passer directement à l'itération suivante.

D: > cours-javascript > JS cours.js > ...

```
1   // Boucle for qui itère de 0 à 5
2   for (let i = 0; i < 6; i++) {
3       // Si i est égal à 3, on utilise continue pour sauter l'affichage
4       if (i === 3) {
5           continue; // Passe à l'itération suivante de la boucle
6       }
7
8       // Affiche la valeur actuelle de i dans la console
9       console.log("Valeur de i : " + i);
10  }
11
12  // Après la boucle, on affiche un message indiquant que la boucle est terminée
13  console.log("La boucle avec continue est terminée.");
14
```

```
[Running] node "d:\cours-javascript\cours.js"
Valeur de i : 0
Valeur de i : 1
Valeur de i : 2
Valeur de i : 4
Valeur de i : 5
La boucle avec continue est terminée.

[Done] exited with code=0 in 0.181 seconds
```

Syntaxe et Structure

JS JavaScript

Structures de contrôle

Les boucles en JavaScript :

-> Intégration avec les instructions continue et break

- **break :** utilisée pour quitter une boucle (comme une boucle for, while ou do...while) ou une instruction switch avant qu'elle ne soit terminée. Elle permet de sortir immédiatement de la structure de contrôle en cours, ce qui peut être utile pour interrompre des itérations sous certaines conditions.

```
JS cours.js > ...
1   for (let i = 0; i < 10; i++) {
2       if (i === 5) {
3           break; // Quitte la boucle lorsque i est égal à 5
4       }
5       console.log(i);
6   }
7
```

```
[Running] node "d:\cours-javascript\cours.js"
0
1
2
3
4

[Done] exited with code=0 in 0.163 seconds
```

Exercices

JS JavaScript

1 Objectifs

1. **Comprendre et appliquer les concepts de contrôle de flux et de conditions en JavaScript.**
2. **Travailler avec HTML et JavaScript pour créer des interactions utilisateur.**

2 Instructions Générales

- Créez un fichier HTML pour chaque exercice.
- Incluez un script JavaScript dans le fichier HTML pour exécuter vos solutions.
- Utilisez les éléments HTML pour interagir avec l'utilisateur (comme des formulaires et des boutons).

Exercices

JS JavaScript

Exercice 1 : Compteur de 1 à 10

Créez un bouton qui, lorsqu'il est cliqué, affiche les nombres de 1 à 10 dans un élément .

Voici la lise des nombres de 1 à 10 Afficher

1
2
3
4
5
6
7
8
9
10

Solution Exercice 1


```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <title>Compteur de 1 à 10</title>
  </head>
  <body>
    Afficher les nombres de 1 à 10
    <button onclick="afficherCompteur()">Afficher</button>
    <div id="resultat"></div>

    <script>
      function afficherCompteur() {
        let affichage = "";
        for (let i = 1; i <= 10; i++) {
          affichage += i + "<br>";
        }
        document.getElementById("resultat").innerHTML = affichage;
      }
    </script>
  </body>
</html>
```

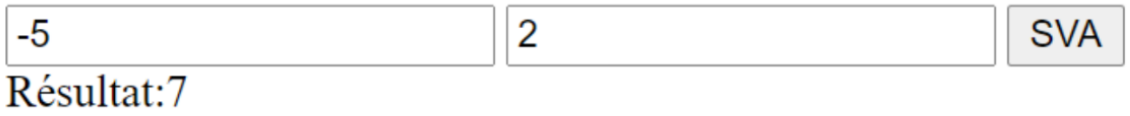
Exercice 2

JS JavaScript

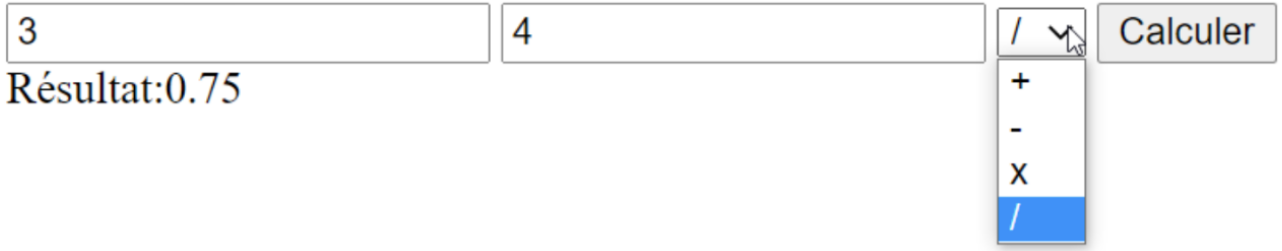
Exercice 2 : Calculatrice Simple

1. Ecrire un programme Javascript qui permet de calculer la valeur absolue d'un nombre saisi dans un champ.

2. Ecrire un programme Javascript qui permet de calculer la somme des valeurs absolues de deux nombres.



3. Ecrire un programme Javascript qui permet de réaliser le calcul selon l'opération choisie.



Solution Exercice 2.1

JS JavaScript

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Exercice1-Q1</title>
    <script src="test.js"></script>
</head>
    <body>
        <input id="nombre" type="nombre" placeholder="Nombre">
        <button onclick="calculer()">Valeur absolue</button><br>
        Valeur absolue:<label id="valeur_absolue"></label><br>
    </body>
</html>
```

```javascript
function calculer()
{
    const nombre = document.querySelector("#nombre").value
    if (nombre<0)
        {   document.querySelector("#valeur_absolue").innerHTML=-nombre }
    else
        {   document.querySelector("#valeur_absolue").innerHTML=nombre  }
}
```

Notez que: QuerySelector: Permet de sélectionner le premier élément qui correspond à un sélecteur CSS donné. Peut utiliser n'importe quel sélecteur CSS (ID, classe, type d'élément, etc.).

Solution Exercice 2.2

JS JavaScript

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Exercice1-Q2</title>
    <script src="test.js"></script>
</head>
    <body>
        <input id="nombre1" type="text" placeholder="Nombre1">
        <input id="nombre2" type="text" placeholder="Nombre2">
        <button onclick="calculer()">SVA</button><br>
        Résultat:<label id="SVA"></label><br>
    </body>
</html>
```

```javascript
function calculer()
{
    let nombre1 = parseFloat(document.querySelector("#nombre1").value)
    let nombre2 = parseFloat(document.querySelector("#nombre2").value)
    if (nombre1<0) nombre1=-nombre1
    if (nombre2<0) nombre2=-nombre2
    document.querySelector("#SVA").innerHTML =nombre1+nombre2
}
```

Solution Exercice 2.3

JS JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Exercice1-Q3</title>
    <script src="test.js"></script>
</head>
<body>
    <input id="nombre1" type="text" placeholder="Nombre1">
    <input id="nombre2" type="text" placeholder="Nombre2">
    <select id="operation">
        <option value="+">+</option>
        <option value="-">-</option>
        <option value="x">x</option>
        <option value="/">/</option>
    </select>
    <button onclick="calculer()">Calculer</button><br>
    Résultat:<label id="resultat"></label><br>
</body>
</html>
```

Solution Exercice 2.3

JS JavaScript

```
function calculer()
{
    let nombre1 = parseFloat(document.querySelector("#nombre1").value)
    let nombre2 = parseFloat(document.querySelector("#nombre2").value)
    const operation = document.querySelector("#operation").value
    let resultat
    switch (operation){
        case '+':resultat = nombre1+nombre2
        break
        case '-':resultat = nombre1-nombre2
        break
        case 'x':resultat = nombre1*nombre2
        break
        case '/':resultat = nombre1/nombre2
        break
    }
    document.querySelector("#resultat").innerHTML=resultat
}
```

Fonctions

JS JavaScript

Définition

- Une fonction est définie comme un bloc de code organisé et réutilisable, créé pour effectuer une action unique ;
- Le rôle des fonctions est d'offrir une meilleure modularité au code et un haut degré de réutilisation ;
- Une fonction JavaScript est un ensemble d'instructions qui peut prendre des entrées de l'utilisateur, effectuer un traitement et renvoyer le résultat à l'utilisateur ;
- Une fonction JavaScript est exécutée au moment de son appel.

En JavaScript, il existe 2 types de fonctions :

Les fonctions propres à JavaScript : appelées "méthodes". Elles sont associées à un objet bien particulier comme la méthode **alert**() avec l'objet **window**.

Les fonctions écrites par le développeur : déclarations placées dans l'en-tête de la page.

Syntaxe et Structure

JS JavaScript

Les fonctions :

-> Les fonctions prédéfinies :

Définition : Une fonction correspond à un bloc de code nommé et réutilisable et dont le but est d'effectuer une tâche précise.

Le langage JavaScript dispose de nombreuses fonctions que nous pouvons utiliser pour effectuer différentes tâches. Les fonctions définies dans le langage sont appelées fonctions prédéfinies ou fonctions prêtes à l'emploi car il nous suffit de les appeler pour nous en servir.

Exemple:

D: > cours-javascript > JS cours.js > ...

```
1   // Générer un nombre aléatoire entre 0 (inclus) et 1 (exclus)
2   const nombreAleatoire = Math.random();
3
4   // Afficher le nombre aléatoire dans la console
5   console.log("Nombre aléatoire généré :", nombreAleatoire);
6
```

```
[Running] node "d:\cours-javascript\cours.js"
Nombre aléatoire généré : 0.23213463978735605

[Done] exited with code=0 in 0.162 seconds
```

Autres exemples :

parseInt() : Convertit une chaîne de caractères en un entier.

alert() : Affiche une boîte de dialogue d'alerte avec un message.

setTimeout() : Exécute une fonction après un certain délai.

Fonctions

JS JavaScript

Syntaxe des fonctions en JavaScript

- En JavaScript, une fonction est définie avec le mot clé **function**, suivi du nom de la fonction , et des parenthèses () ;
- Le nom de la fonction doit respecter les mêmes règles que les noms des variables ;
- Les parenthèses peuvent inclure des noms de paramètres séparés par des virgules. Les arguments de la fonction correspondent aux valeurs reçues par la fonction lorsqu'elle est invoquée ;
- Le code à exécuter, par la fonction, est placé entre accolades : {}

```
function nomFonction(paramètre1, paramètre2, paramètre3, …)
{
    // Code de la function
   return … ;
}
```

Retour de la fonction

- L'instruction **return** est utilisée pour renvoyer une valeur (souvent calculée) au programme appelant.
- Lorsque l'instruction **return** est atteinte, la fonction arrête son exécution.

Syntaxe et Structure

JS JavaScript

Les fonctions :

-> Les fonctions personnalisées :

-> On crée **une fonction personnalisée** grâce au mot clef **function** ;

-> Si une fonction a besoin qu'on lui passe des valeurs pour fonctionner, alors on définira des paramètres lors de sa définition.

D: > cours-javascript > JS cours.js > ...

```
1   // Déclaration d'une fonction sans argument
2   function maFonction() {
3       // Corps de la fonction (peut rester vide)
4   }
5
```

D: > cours-javascript > JS cours.js > ...

```
1   // Déclaration d'une fonction avec deux arguments
2   function maFonction2(arg1,arg2) {
3
4   }
5
```

-> Pour exécuter le code d'une fonction, il faut l'appeler. Pour cela, il suffit d'écrire son nom suivi d'un couple de parenthèses en passant éventuellement des arguments dans les parenthèses ;

D: > cours-javascript > JS cours.js

```
1   maFontion();
2   maFonction2("Ali",28);
3
```

Fonctions

JS JavaScript

Appel de fonction

Le code de la fonction est exécuté quand la fonction est appelée selon les cas suivants :

- Lorsqu'un événement se produit (clic sur un bouton par exemple) ;
- Lorsqu'il est invoqué (appelé) à partir d'un autre code JavaScript ;
- Automatiquement (auto-invoqué).

La fonction est appelée en utilisant son nom et, si elle prend des paramètres, en indiquant la liste de ses arguments :

```
nomFonction(p1, p2, p3, …)
```

Exemple : fonction avec retour

```
let x = myFunction(4, 3);   // La function est appelée, la valeur retournée est affectée à x
function myFunction(a, b) {
  return a * b;             // La function retourne le produit de a et b
}
```

Syntaxe et Structure

JS JavaScript

Les fonctions :

-> La notion de portée des variables : Définition

Il est essentiel de saisir la notion de « portée » des variables lorsque l'on travaille avec des fonctions en JavaScript.

Définition de la portée : La portée d'une variable fait référence à la zone du script dans laquelle elle est accessible. En effet, toutes les variables ne sont pas disponibles à chaque endroit d'un script, ce qui limite leur utilisation.

Il existe deux espaces de portée principaux :

- **Espace global :** Il englobe l'ensemble du script, à l'exception du code à l'intérieur des fonctions. Les variables déclarées dans cet espace sont accessibles partout dans le script.
- **Espace local :** À l'opposé, l'espace local est celui d'une fonction spécifique. Les variables déclarées à l'intérieur d'une fonction ne peuvent être utilisées que dans le cadre de cette fonction.

Cette distinction entre portée globale et locale est cruciale pour éviter les conflits de noms de variables et pour gérer correctement l'accès aux données dans votre code.

Fonctions

JS JavaScript

Variables locales à la fonction

- A l'intérieur de la fonction, les arguments (les paramètres) sont considérés comme des variables locales.
- Les variables locales ne sont accessibles qu'à partir de la fonction.

```
// Le code ici ne peut pas utiliser la variable "nom"

function myFunction() {
    let nom = "Hassan";
    // Le code ici peut utiliser la variable "nom"
  }

// Le code ici ne peut pas utiliser la variable "nom"
```

Remarques

- Étant donné que les variables locales ne sont reconnues qu'à l'intérieur de leurs fonctions, les variables portant le même nom peuvent être utilisées dans différentes fonctions.
- Les variables locales sont créées lorsqu'une fonction démarre et supprimées lorsque la fonction est terminée.

Syntaxe et Structure

JS JavaScript

Les fonctions :

-> La notion de portée des variables : Exemple pratique

: > cours-javascript > JS cours.js > ...

```
1   // On déclare deux variables globales
2   let x = 5;
3   var y = 10;
4
5   // On définit une première fonction qui utilise les variables globales
6   function portee1() {
7       console.log('Depuis portee1() :');
8       console.log('x = ' + x);
9       console.log('y = ' + y);
10  }
11
12  // On définit une deuxième fonction qui définit des variables locales
13  function portee2() {
14      let a = 1;
15      var b = 2;
16      console.log('Depuis portee2() :');
17      console.log('a = ' + a);
18      console.log('b = ' + b);
19  }
20
21  // On définit une troisième fonction qui définit également des variables locales
22  function portee3() {
23      let x = 20;
24      var y = 40;
25      console.log('Depuis portee3() :');
26      console.log('x = ' + x);
27      console.log('y = ' + y);
28  }
29
```

Appels:

```
30  // On pense bien à exécuter nos fonctions !
31  portee1();
32  portee2();
33  portee3();
34
```

Résultat:

```
[Running] node "d:\cours-javascript\cours.js"
Depuis portee1() :
x = 5
y = 10
Depuis portee2() :
a = 1
b = 2
Depuis portee3() :
x = 20
y = 40

[Done] exited with code=0 in 0.175 seconds
```

Syntaxe et Structure

JS JavaScript

Les fonctions :

-> Les valeurs de retour des fonctions

Une valeur de retour est le résultat qu'une fonction renvoie à l'endroit où elle a été appelée. Cela permet d'utiliser le résultat de cette fonction dans d'autres parties du code.

Pour qu'une fonction retourne une valeur, on utilise l'instruction return, suivie de la valeur ou de l'expression à renvoyer.

D: > cours-javascript > JS cours.js > ...

```
1   function addition(a, b) {
2       return a + b; // La fonction retourne la somme de a et b
3   }
4
5   let resultat = addition(3, 4); // resultat vaudra 7
6
```

Il est important de noter qu'il n'est pas nécessaire qu'une fonction renvoie une valeur. Si aucune instruction return n'est utilisée, la fonction renverra undefined par défaut.

Lorsqu'on utilise des fonctions prédéfinies en JavaScript, il est essentiel de vérifier leur valeur de retour, car cela peut affecter le comportement de votre code.

Fonctions

JS JavaScript

Les expressions lambdas

- Les fonctions fléchées (arrow functions) sont des fonctions qui ont une syntaxe compacte. Par conséquent, elles sont plus rapide à écrire que les fonctions traditionnelles ;
- Les fonctions fléchées sont limitées et ne peuvent pas être utilisées dans toutes les situations ;
- Principe: à la place du mot clé function, on utilise le signe (**=>**) plus une parenthèse carrée fermante (>) après la parenthèse fermante de la fonction :

Exemple 1 :

```
const variable = () => {
    return "ma_variable"
  }
  console.log(variable()) // "ma_variable"
```

Exemple 2 :

```
const variable = (a,b) => {
    return a*b;
  }
  console.log(variable(2,3)) // 6
```

Fonctions

JS JavaScript

Les expressions lambdas

Exemple 3 :

```
const langages = [
    'Java',
    'Python',
    'PHP',
    'Scala'
  ];

  console.log(langages.map(L => L.length));
```

La fonction **map** parcourt les éléments de la liste "langages" et applique l'expression lambda définie par la fonction: (L)=>L.lenght.

Exemple 4 :

```
const langages = [
    'Java',
    'Python',
    'PHP',
    'Scala'
  ];
 langages.forEach(L=>{
     if (L.length>4){
        console.log(L);
     }
 })
```

Notez que:

- **map()** : Cette méthode crée un nouveau tableau contenant les résultats de l'appel d'une fonction fournie sur chaque élément du tableau d'origine.
- L est un paramètre qui représente chaque élément du tableau lors de l'itération.
- L.length renvoie la longueur de la chaîne de caractères L.
- **forEach()** : C'est une méthode qui exécute une fonction fournie une fois pour chaque élément du tableau, sans créer un nouveau tableau.

Les Fonctions Récursives

Les fonctions récursives en JavaScript sont des fonctions qui s'appellent elles-mêmes jusqu'à ce qu'une condition de sortie soit remplie. Elles sont particulièrement utiles pour résoudre des problèmes qui peuvent être décomposés en sous-problèmes plus petits,

Syntaxe:

```
function recurse() {
    // Code de la fonction
    if (condition) {
        // Condition de sortie
        return;
    } else {
        // Appel récursif
        recurse();
    }
}
// Appel initial de la fonction récursive
recurse();
```

Fonctions

JS JavaScript

- Exemple

```
function somme(n) {
    if (n <= 1) {
        return n;
    } else {
        return n + somme(n - 1);
    }
}
let resultat = somme(10);
document.write("La somme des nombres de 1 à 10 est : " +
resultat);
```

Fonctions: Exercices

Exercice 1 : Calcul de la somme

Écrivez une fonction somme qui prend deux nombres en paramètres et retourne leur somme.

Exercice 2 : Vérification de la parité

Écrivez une fonction est Pair qui prend un nombre en paramètre et retourne true si le nombre est pair, et false sinon

Exercice 3 : Calcul de la factorielle

Écrivez une fonction factorielle qui prend un nombre en paramètre et retourne sa factorielle.

Exercice 4 : Filtrage des nombres pairs

Écrivez une fonction filtrerPairs qui prend un tableau de nombres en paramètre et retourne un nouveau tableau contenant uniquement les nombres pairs. Utilisez cette fonction pour filtrer les nombres pairs du tableau [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

Fonctions: Correction

JS JavaScript

- Exercice 1

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

let resultat = somme(5, 7);
document.write("La somme de 5 et 7 est : " + resultat);
```

Fonctions: Correction

JS JavaScript

- Exercice 2

```
function estPair(nombre) {
    return nombre % 2 === 0;
}

document.write("10 est pair : " + estPair(10) + "<br>");
document.write("15 est pair : " + estPair(15) + "<br>");
```

Fonctions: Correction

JS JavaScript

- Exercice 3

```
function factorielle(n) {
    if (n === 0) {
        return 1;
    } else {
        return n * factorielle(n - 1);
    }
}
let resultat = factorielle(5);
document.write("La factorielle de 5 est : " + resultat);
```

Fonctions: Correction

JS JavaScript

- Exercice 4

```
function estPair(nombre) {
    return nombre % 2 === 0;
}
function filtrerPairs(tableau) {
    return tableau.filter(estPair);
}
let nombres = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let pairs = filtrerPairs(nombres);
document.write("Les nombres pairs sont : " + pairs.join(", "));
```

Notez que: La méthode filter() en JavaScript est utilisée pour créer un nouveau tableau contenant tous les éléments du tableau d'origine qui passent un test (fourni par une fonction)