




JavaScript

PARTIE 1



A- JavaScript dans le développement informatique

Introduction

- ▶ **JavaScript** est un langage de programmation créé en 1995.
- ▶ **JavaScript** fait partie des **langages web standards** avec le **HTML** et le **CSS**.
- ▶ **ECMA International** est l'organisation qui gère l'évolution de **JavaScript** et se charge de publier ses standards.

Définition du JavaScript

- ▶ Au contraire du langage statique, le JavaScript est un **langage dynamique** qui va nous permettre de :
 - ▶ Faire bouger, apparaître ou disparaître des éléments de la page (un titre, un menu, un paragraphe, une image...).
 - ▶ Mettre à jour des éléments de la page sans recharger la page (changer le texte, recalculer un nombre, ...).
 - ▶ Demander au serveur un nouveau bout de page et l'insérer dans la page en cours, sans la recharger.
 - ▶ Attendre que l'utilisateur fasse quelque chose (cliquer, taper au clavier, bouger la souris...) et réagir (faire une des opérations ci-dessus suite à cette action).

Définition du JavaScript

- ▶ **JavaScript** est un langage principalement utilisé **côté client**, ce qui signifie que le code sera exécuté dans le navigateur des utilisateurs qui demandent la page.
- ▶ **JavaScript** est un **langage interprété** qui s'exécute directement à l'aide des navigateurs, contrairement au langage compilé qu'il faut le transformer en langage machine pour pouvoir l'exécuter.
- ▶ **JavaScript** est un **langage orienté objet**.

Éditeurs de JavaScript

- ▶ Voici une liste d'éditeurs qui permettent de coder en JavaScript.
 - ▶ Komodo Edit
 - ▶ Atom
 - ▶ Notepad++
 - ▶ Brackets
 - ▶ **Visual Studio Code**
 - ▶ codepen.io
 - ▶ jsbin.com



B- Les fondamentaux de JavaScript

Où écrire le code JavaScript ?

- ▶ On peut placer le code JavaScript à trois endroits différents :
 - ▶ Directement dans la balise ouvrante d'un élément HTML ;
 - ▶ Dans un élément script, au sein d'une page HTML ;
 - ▶ Dans un fichier séparé contenant exclusivement du JavaScript et portant l'extension .js

Où écrire le code JavaScript ?

- Placer le code JavaScript dans la balise ouvrante d'un élément HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

  <h1> JavaScript dans la balise ouvrante d'un élément HTML </h1>
  <button onclick="alert('Bonjour !')"> Cliquez ici </button>

</body>
</html>
```

Où écrire le code JavaScript ?

- Placer le code JavaScript dans l'élément **script**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
  </head>
  <body>
    <h1>JavaScript dans un élément script au sein d'une page HTML</h1>
    <button id='b1'>Cliquez ici</button>
    <script>
      let bonjour = document.getElementById('b1');
      bonjour.addEventListener('click', alerte);
      function alerte(){ alert('Bonjour'); }
    </script>
  </body>
</html>
```

Où écrire le code JavaScript ?

- Placer le code JavaScript dans un fichier séparé

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <script src='fichier.js' async></script>
  </head>
  <body>
    <h1>Placer le code JavaScript dans un fichier séparé</h1>
    <button id='b1'>Cliquez ici</button>
  </body>
</html>
```

```
let bonjour = document.getElementById('b1');
bonjour.addEventListener('click', alerte);
function alerte(){
  alert('Bonjour');
}
```

Fichier.js

Les commentaires en JavaScript

- ▶ Les commentaires sont des lignes de texte placées au milieu d'un script et servant à documenter le code ou des parties du code.
- ▶ Les commentaires permettent aux développeurs à se repérer plus facilement dans un script, à le lire et à le comprendre plus vite.

Les commentaires en JavaScript

- ▶ Il existe deux types de commentaires :
 - ▶ Commentaire multi-lignes est encadré par la syntaxe suivante `/* */`
 - ▶ Commentaire mono-ligne est précédé par un double slash `//`

- ▶ Exemples

```
/* Commentaire multi-lignes  
Commentaire multi-lignes */
```

```
/* Commentaire multi-lignes */
```

```
// Commentaire mono-ligne
```

L'indentation en JavaScript

- ▶ L'indentation correspond au fait de décaler certaines lignes de code par rapport à d'autres.
- ▶ Cela est généralement utilisé pour rendre son code plus lisible et plus simple à comprendre.
- ▶ Exemple :

```
let bonjour = document.getElementById('bl');  
bonjour.addEventListener('click', alerte);  
function alerte() {  
  ↕↔ alert('Bonjour');  
}
```


Un premier point sur la syntaxe de base du JavaScript

- ▶ Dans un premier temps, il est obligatoire d'ajouter explicitement des points-virgules à la fin de toutes les instructions afin de garantir que le code sera interprété correctement.
- ▶ Dans un deuxième temps, et après l'acquisition d'une connaissance parfaite du comportement du JavaScript et des règles d'ajout automatiquement des points-virgules, nous pouvons laisser l'ajout de ces derniers au langage.

Qu'est-ce qu'une variable ?

- ▶ Une variable est un **conteneur** servant à stocker des données de manière temporaire. Ces données peuvent changer pendant l'exécution du code.
- ▶ Lorsqu'on stocke une valeur dans une variable, on dit également qu'on **affecte** une valeur à une variable
- ▶ Pour pouvoir utiliser les variables, il faut les **déclarer**.

Déclaration des variables

- ▶ Pour déclarer une variable en JavaScript, nous utilisons le mot clef **var** ou le mot clef **let** suivi du nom qu'on veut donner à notre variable.
- ▶ les noms des variables doivent respecter les règles suivantes :
 - ▶ Le nom d'une variable doit obligatoirement commencer par une lettre ou un sous tiret (_);
 - ▶ Le nom d'une variable ne doit contenir que des lettres, des chiffres et des sous tirets mais pas de caractères spéciaux ;
 - ▶ Le nom d'une variable ne doit pas contenir d'espace.

Déclaration des variables

- ▶ En JavaScript, le nom des variables est **sensible à la casse**. Cela signifie que l'usage de majuscules ou de minuscules avec un même nom va permettre de définir des variables différentes.
- ▶ Exemple :
 - ▶ texte, TEXTE et Texte sont trois variables différentes.

Déclaration des variables

- ▶ Vous ne pouvez pas utiliser des **noms réservés** en JavaScript comme noms pour vos variables.
- ▶ JavaScript utilise déjà les **noms réservés** pour désigner différents éléments intégrés au langage.

Déclaration des variables

► Exemples de déclaration :

```
// Déclarer une variable
```

```
let nom ;
```

```
// Déclarer et initialiser une variable en même temps
```

```
let prenom = 'jamal' ;
```

```
// Déclarer une variable puis l'initialiser ensuite
```

```
let monAge ;
```

```
monAge = 20 ;
```

```
// modifier la valeur stocker dans la variable monAge
```

```
monAge = 19 ;
```


Déclaration des variables

- La différence entre une déclaration avec **let** et une déclaration avec **var** :

Var	Let
Ancienne syntaxe de déclaration	Nouvelle syntaxe de déclaration
On peut utiliser la variable avant sa déclaration	On doit déclarer la variable avant son utilisation
On peut déclarer la même variable plusieurs fois	La déclaration d'une variable ne se fait qu'une seule fois.
Var dans une fonction, la variable sera accessible dans tous les blocs de la fonction.	Let dans une fonction, la variable sera accessible dans le bloc dans lequel elle a été déclarée

Déclaration des Constantes

- ▶ Une constante est un conteneur qui va recevoir une valeur que nous ne pouvons pas changer par la suite.
- ▶ Exemples :

```
const tva = 0.2 ;  
const pi = 3.14 ;
```

Les types de variables

- ▶ En JavaScript, **contrairement à d'autres langages de programmation**, nous n'avons pas besoin de préciser le type de valeur qu'une variable va pouvoir stocker.
- ▶ JavaScript détecte automatiquement le type de la valeur stockée dans une variable.
- ▶ Donc, nous pouvons stocker différents types de valeurs dans une variable au fil du temps sans se préoccuper d'une quelconque compatibilité.

Les types de variables

- ▶ Il existe 7 types de valeurs différents :
 - ▶ String (chaîne de caractères);
 - ▶ Number (nombre);
 - ▶ Boolean (booléen);
 - ▶ Undefined (indéfini);
 - ▶ Null (vide);
 - ▶ Object (objet);
 - ▶ Symbol (symbole);

Les types de variables

► Le type chaîne de caractères

```
//Délimiteurs non trouvés dans la chaîne = rien à échapper  
let a = "Je m'appelle Pierre";  
  
//Délimiteurs non trouvés dans la chaîne (apostrophe non droit) = rien à échapper  
let b = 'Je m'appelle Pierre';  
  
//Délimiteurs trouvés dans la chaîne = on échappe le caractère en question  
let c = 'Je m\'appelle Pierre';  
  
//Délimiteurs non trouvés dans la chaîne = rien à échapper  
let d = "Je m'appelle « Pierre »";  
  
//Délimiteurs trouvés dans la chaîne = on échappe les caractères en question  
let e = "Je m'appelle \"Pierre\"";
```

Les types de variables

- ▶ Le type chaîne de caractères
 - ▶ Exemples :

```
let prenom = "Je m'appelle jamal";  
let age = 20;  
let age2 = '20';
```


Les types de variables

- ▶ Le type chaîne de caractères
 - ▶ la fonction **typeof** nous permet de vérifier le type d'une variable
 - ▶ Exemple :

```
<body>
  <h1>Le type chaîne de caractères</h1>
  <p id='p1' ></p>
  <p id='p2' ></p>
  <p id='p3' ></p>
</body>
```

fichier. html

```
let prenom = "Je m'appelle jamal";
let age = 20;
let age2 = '20';
```

fichier. js

```
document.getElementById('p1').innerHTML = 'Type de prenom : ' + typeof prenom;
document.getElementById('p2').innerHTML = 'Type de age : ' + typeof age;
document.getElementById('p3').innerHTML = 'Type de age2 : ' + typeof age2;
```

Les types de variables

- ▶ Le type nombre

- ▶ Exemples :

```
let x = 10; //x stocke un entier positif  
let y = -2; //y stocke un entier négatif  
let z = 3.14; //z stocke un nombre décimal positif
```

- ▶ il faut remplacer les virgules par des points pour les nombres décimaux.

Les types de variables

► Le type booléen

► Exemples :

```
let vrai = true; //Stocke le booléen true
let faux = false; //Stocke le booléen false

/*On demande au JavaScript d'évaluer la comparaison "8 > 4". Comme 8 est bien
*strictement supérieur à 4, le JavaScript renvoie true en résultat. On
*stocke ensuite ce résultat (le booléen true) dans la variable let resultat*/
let resultat = 8 > 4;
```

Les types de variables

- ▶ Le type vide et le type indéfini
 - ▶ Exemple :
 - ▶ **let a ;**
 - ▶ **Let b;**

Les opérateurs arithmétiques

Opérateur	Nom de l'opération associée
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

Les opérateurs d'affectation

Opérateur	Définition
<code>+=</code>	Additionne puis affecte le résultat
<code>-=</code>	Soustrait puis affecte le résultat
<code>*=</code>	Multiplie puis affecte le résultat
<code>/=</code>	Divise puis affecte le résultat
<code>%=</code>	Calcule le modulo puis affecte le résultat

La concaténation

- ▶ 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 une chaîne de caractères, il sert d'opérateur de concaténation.
- ▶ Exemples

```
let x = 28 + 1; //Le signe "+" est ici un opérateur arithmétique
```

```
let z = x + 'ans'; //Le signe "+" est ici un opérateur de concaténation
```

La concaténation

- ▶ **Les littéraux de gabarits** : est une manière introduite récemment qui permet d'entourer des chaînes de caractères et qui va utiliser des accents graves `
- ▶ Pour que cela fonctionne bien, il faut placer les variables et les expressions entre `\${` et `}`.
- ▶ Exemple :

```
let z = x + 'ans';
```

```
let z = `${x} ans`;
```

Exemple de code

```
let start = 'Bonjour ', nom, end = ' !', result;  
nom = prompt('Quel est votre prénom ?');  
result = start + nom + end;  
alert(result);
```

Exercise



Structures
alternatives

Structures alternatives

- ▶ Les structures alternatives vont nous permettre d'exécuter **une série d'instructions** si une condition donnée est vérifiée **ou une autre série d'instructions** si elle ne l'est pas.
- ▶ donc, nous allons comparer la valeur d'une variable à une certaine autre valeur donnée et **selon le résultat de la comparaison** exécuter un bloc de code ou pas.
- ▶ Pour comparer des valeurs, nous allons utiliser des **opérateurs de comparaison**.

Structures alternatives

► Opérateurs de comparaison:

Opérateur	Définition
==	tester l'égalité sur les valeurs
===	tester l'égalité en termes de valeurs et de types
!=	tester la différence en valeurs
<>	tester la différence en valeurs
!==	tester la différence en valeurs ou en types
<	tester si une valeur est strictement inférieure à une autre
>	tester si une valeur est strictement supérieure à une autre
<=	tester si une valeur est inférieure ou égale à une autre
>=	tester si une valeur est supérieure ou égale à une autre

Structures alternatives

- ▶ Ce que signifie le triple égal **===**
 - ▶ Imaginons que l'on possède une variable **let x** dans laquelle on stocke le **chiffre 4**. On veut ensuite **comparer** la valeur stockée dans notre variable à la **chaîne de caractères '4'**.
 - ▶ Si on utilise le **double signe égal** pour effectuer la comparaison, **l'égalité va être validée** par le JavaScript car celui-ci ne va tester que les valeurs, et **4 est bien égal à '4' en termes de valeur**.
 - ▶ En revanche, si on utilise le **triple signe égal**, alors **l'égalité ne va pas être validée** car nous comparons un nombre à une chaîne de caractères (donc des **types différents de valeurs**).

Structures alternatives

- ▶ Opérateurs de logique:
 - ▶ ce sont des opérateurs qui vont principalement être utilisés avec des valeurs booléennes et au sein de conditions.

Opérateur (nom)	Opérateur (symbole)	Description
ET	&&	Renvoie true si toutes les comparaisons sont évaluées à true ou false sinon
OU		Renvoie true si au moins l'une des comparaisons est évaluée à true ou false sinon
NON	!	Renvoie false si une comparaison est évaluée à true ou renvoie true dans le cas contraire

Structures alternatives

- ▶ Structures alternatives **if**
 - ▶ Cette nous permet d'exécuter un bloc de code **si et seulement si** le résultat d'un test vaut **true**.
 - ▶ Exemple :

```
let x = 4;
let y = 0;
if(x > 1){
    alert('x contient une valeur strictement supérieure à 1');
}
if(x == y){
    alert('x et y contiennent la même valeur');
}
if(y){
    alert('La valeur de y est évaluée à true');
}
```

Structures alternatives

► Structures alternatives **if**

- vous devez savoir que toute valeur évaluée par le JavaScript **dans un contexte booléen** va être évaluée à **true** à l'exception des valeurs suivantes qui vont être évaluées à **false** :
 - Le booléen false ;
 - La valeur 0 ;
 - Une chaîne de caractères vide ;
 - La valeur undefined ;
 - La valeur null ;
 - La valeur NaN (« Not a Number » = « n'est pas un nombre »).

Structures alternatives

- ▶ Structures alternatives **if**
- ▶ Inverser la valeur logique d'un test
 - ▶ le code sera exécuté si le résultat de la comparaison est false.
 - ▶ Dans ce cas il y a deux méthodes :
 - ▶ soit en utilisant l'opérateur logique inverse !
 - ▶ soit en comparant explicitement le résultat de notre comparaison à false.

Structures alternatives

- ▶ Structures alternatives **if**
 - ▶ Exemple :

```
let x = 4;  
if (!(x > 1)) {  
    alert('x contient une valeur inférieure ou égale à 1');  
}
```

```
let x = 4;  
if ((x > 1)==false) {  
    alert('x contient une valeur inférieure ou égale à 1');  
}
```

Structures alternatives

- ▶ Structures alternatives **if...else**
- ▶ Cette structure nous permet d'exécuter le **bloc if du code** si la condition renvoie true ou le **bloc else du code** dans le cas contraire.
 - ▶ Exemple :

```
let x = 4;  
if(x > 1){  
    alert('x contient une valeur strictement supérieure à 1');  
}else{  
    alert('x contient une valeur inférieure ou égale à 1');  
}
```

Structures alternatives

- ▶ Structures alternatives **if...else if...else**
- ▶ Cette structure nous permet de prendre en charge plusieurs que l'on souhaite..
 - ▶ Exemple :

```
let x = 0.5;
if(x > 1){
    alert('x contient une valeur strictement supérieure à 1');
}else if(x == 1){
    alert('x contient la valeur 1');
}else{
    alert('x contient une valeur strictement inférieure à 1');
}
```

Structures alternatives

► Structures ternaires

- Les structures ternaires sont des écritures condensées des structures alternatives et qui se présentent sous la forme suivante :

test ? code à exécuter si true : code à exécuter si false ;

- Exemple :

```
let nbre;  
nbre=prompt("Entrez un nombre entier : ");  
(nbre%2)==0 (?)  
alert(" le nombre "+ nbre + "est pair");  
alert(" le nombre "+ nbre + "est impair");
```

Structures alternatives

► Structures ternaires

- Dans cas où le **code à exécuter si true/si false** se compose de plusieurs instructions, on met le code entre **deux parenthèses** et les instructions seront séparées par des **virgules**.
- La syntaxe sera :

test ? (**instr₁, instr₂, ..., instr_n**) : (**instr₁, instr₂, ..., instr_n**);

Structures alternatives

- ▶ L'instruction **switch**
- ▶ L'instruction switch va nous permettre d'exécuter un code en fonction de la valeur d'une variable.
- ▶ l'instruction switch représente une alternative à l'utilisation d'un if...else if...else.
- ▶ ces deux types d'instructions ne sont pas strictement équivalentes.
 - ▶ Dans un switch chaque cas va être lié à une valeur précise.
 - ▶ L'instruction switch ne supporte pas l'utilisation des opérateurs de **supériorité** ou **d'infériorité**.

Structures alternatives

► L'instruction **switch**

```
let x = prompt("choisis un nombre de 1 à 3 :") ;
switch(x) {
  case "1":
    alert('1\'utilisateur a choisi le nombre 1');
    break;
  case "2":
    alert('1\'utilisateur a choisi le nombre 2');
    break;
  case "3":
    alert('1\'utilisateur a choisi le nombre 3');
    break;
  default:
    alert('1\'utilisateur a choisi un autre nomnre que 1,2,3');
}
```

Exercices



Précédence et
Associativité

Précédence et Associativité

► Précédence des opérateurs :

- La **précédence des opérateurs** détermine l'ordre dans lequel les **différents opérateurs** sont évalués.
- Les opérateurs avec la **plus haute précédence** (ou **priorité**) sont évalués en premier.
- Exemple :
 - * possède une précédence plus haute que +
 - * est évalué en premier puis + en deuxième
 - l'expression : $6 + 4 * 3$ renverra 18 (et pas 30)

Précédence et Associativité

► Associativité

- **L'associativité** détermine l'ordre dans lequel des opérateurs de **même précédence** sont évalués.
- Exemple : $a \text{ op } b \text{ op } c$
- **Une associativité de gauche** (gauche à droite) signifie qu'elle est évaluée comme $(a \text{ OP } b) \text{ OP } c$
- **une associativité de droite** (droite à gauche) signifie qu'elle est interprétée comme $a \text{ OP } (b \text{ OP } c)$
- L'expression $5/2*2$ renverra 5 (et pas 1.25)

Précédence et Associativité

► Tableau de Précédences et d'associativité

Précédence	Opérateur (nom)	Opérateur (symbole)	Associativité
0	Groupement	(...)	Non applicable
1	Post-incrémentation	... ++	Non applicable
1	Post-décrémentation	... —	Non applicable
2	NON (logique)	! ...	Droite
2	Pré-incrémentation	++ ...	Droite
2	Pré-décrémentation	— ...	Droite

Précédence et Associativité

► Tableau de Précédences et d'associativité

3	Exponentiel	$\dots ** \dots$	Droite
3	Multiplication	$\dots * \dots$	Gauche
3	Division	\dots / \dots	Gauche
3	Modulo	$\dots \% \dots$	Gauche
4	Addition	$\dots + \dots$	Gauche
4	Soustraction	$\dots - \dots$	Gauche
5	Inférieur strict	$\dots < \dots$	Gauche
5	Inférieur ou égal	$\dots \leq \dots$	Gauche
5	Supérieur strict	$\dots > \dots$	Gauche
5	Supérieur ou égal	$\dots \geq \dots$	Gauche

Précédence et Associativité

► Tableau de Précédences et d'associativité

6	Égalité (en valeur)	$\dots == \dots$	Gauche
6	Inégalité (en valeur)	$\dots != \dots$	Gauche
6	Egalité (valeur et type)	$\dots === \dots$	Gauche
6	Inégalité (valeur ou type)	$\dots !== \dots$	Gauche
7	ET (logique)	$\&\&$	gauche
8	OU (logique)	$ $	gauche
9	Ternaire	$\dots ? \dots : \dots$	Droite
10	Affectation (simple ou combiné)	$\dots = \dots, \dots += \dots, \dots -= \dots, \text{etc.}$	Droite

Exercices



Structures
itératives

Structures itératives

- ▶ **Structures itératives ou boucles** nous permettent d'exécuter plusieurs fois un bloc de code tant qu'une condition donnée est vérifiée.
 - ▶ En JavaScript, nous disposons de **six boucles différentes** :
 - ▶ La boucle while (tant que) ;
 - ▶ La boucle do... while (faire... tant que) ;
 - ▶ La boucle for (pour) ;
 - ▶ La boucle for... in (pour... dans) ;
 - ▶ La boucle for... of (pour... parmi) ;
 - ▶ La boucle for await... of (pour -en attente-... parmi).
- } **Concernent les objets**

Structures itératives

- ▶ **Structures itératives ou boucles** se composent de trois choses :
 - ▶ **Une valeur de départ** qui nous sert à initialiser le compteur de notre boucle;
 - ▶ **Un compteur** 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, la valeur de départ sera incrémentée ou décrémentée.
 - ▶ **Un test ou une condition** de sortie qui précise le critère de sortie de la boucle ;

Structures itératives

- ▶ **Opérateurs d'incrémentation et de décrémentation**
 - ▶ **Incrémenter une variable** signifie ajouter 1 à cette variable tandis que **décrémenter** signifie enlever 1.
 - ▶ il y a deux façons **d'incrémenter** une variable :
 - ▶ **Pré-incrémentation** (++x) : **incrémenter** la valeur de la variable puis **retourner** (utiliser) la valeur de la variable incrémentée
 - ▶ **Post-incrémentation** (x++) : **retourner** (utiliser) la valeur de la variable avant incrémentation puis ensuite **l'incrémenter**.
 - ▶ **Exemple d'utilisation :**

Structures itératives

► Structures itératives ou boucles

- Exemple d'utilisation :
- La même chose, il y a deux façons **de décrémenter** une variable :
 - Pré-décrémentation
 - Post-décrémentation

```
let x = 1, y = 1;
```

```
let z, t;
```

```
z = ++x * 2;
```

```
t = y++ * 2;
```

Structures itératives

- ▶ **La boucle while** (tant que) : Permet de **répéter** une série d'instructions **tant que la condition de sortie n'est pas vérifiée**.
- ▶ **Exemple :**

```
let x=0;
while(x<3){
    console.log('x = ' + x);
    x++;
}
```

Structures itératives

- ▶ **La boucle do... while** (faire... tant que) : Comme **while** mais elle exécute le code de la boucle au premier passage avant l'évaluation de la condition de sortie.
- ▶ **Exemple :**

```
let x=0;  
do{  
  console.log('x = ' + x);  
  x++;  
}while(x<3)
```

Structures itératives

- ▶ **La boucle for** (pour) : permet d'exécuter le code de la boucle selon un nombre de fois connu à l'avance.
- ▶ **Exemple :**

```
for(let i=0; i<10; i++){  
  console.log('i = ' + i)  
}
```


Structures itératives

- ▶ Instruction **continue** : Permet de sauter l'itération actuelle et de **passer directement à l'itération suivante**.
- ▶ **Exemple** : afficher les nombres pairs compris entre 1 et 9

```
for(let i=1; i!=10; i++){  
  ... if(i%2!=0){  
    ... continue;  
  ... }  
  ... console.log(i + 'est un nombre pair')  
}
```

Structures itératives

- ▶ Instruction **break** : permet de stopper l'exécution d'une boucle et de sortir à un moment donné.
- ▶ **Exemple** : Sortir de la boucle une fois la valeur est trouvée

```
for(let i=0; i<100; i++){  
  ... if(i==20){  
    ... break  
  ... }  
  ... console.log(`i = ${i}`)  
}
```

Exercices



Fonctions

Fonctions

- ▶ Les **fonctions** sont des **blocs de code nommés et réutilisables** et dont le **but est d'effectuer une tâche précise**;
- ▶ Il existe deux grands types de fonctions en JavaScript :
 - ▶ les **fonctions prédéfinies** ou natives (qui sont des **méthodes**) qu'on n'aura qu'à utiliser ;
 - ▶ les **fonctions personnalisées** qu'on va pouvoir créer ;

Fonctions

- ▶ On crée une fonction personnalisée grâce au mot clef **function** ;
- ▶ Si une fonction a besoin qu'on lui passe des **valeurs (arguments)** pour fonctionner, alors on définira des **paramètres** lors de sa définition.
- ▶ 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** qui prendront la place des **paramètres** dans les parenthèses ;

Fonctions

► Exemple 1 : fonction sans paramètres

```
function somme() {  
    let x= parseInt(prompt("Entrer le premier nombre "));  
    let y=parseInt(prompt("Entrer le deuxième nombre "));  
    alert("la somme de deux nombres est : " + (x+y));  
}  
somme();
```

Fonctions

▶ Exemple 2 : fonction avec paramètres

```
function somme(nbre1, nbre2) {  
    return nbre1+nbre2;  
}  
  
let x= parseInt(prompt("Entrer le premier nombre "));  
let y=parseInt(prompt("Entrer le deuxième nombre "));  
let z=somme(x,y) ;  
alert("la somme de deux nombres est :  " + z) ;
```

- ▶ **L'instruction return** permet à la fonction de retourner une valeur qu'on va ensuite pouvoir utiliser.
- ▶ **L'instruction return** met fin à l'exécution d'une fonction.

Fonctions

- ▶ **Une fonction réursive** est une fonction qui va **s'appeler** elle-même au sein de son code.
- ▶ **Exemple :**

```
function quitter() {  
    let x=prompt("voulez-vous continue ? o/n");  
    if(x=="o") {  
        .....  
        quitter()  
    }  
}  
quitter();
```


Fonctions

- ▶ Une fonction anonyme est une fonction qui n'a pas de nom.
- ▶ Exemples :

```
function() {  
    alert("Bonjour tout le monde ");  
}
```

Fonctions

- ▶ Pour exécuter une fonction anonyme, on va notamment pouvoir :
 - ▶ **Enfermer le code** de notre fonction **dans une variable** et utiliser la variable comme une fonction ;
 - ▶ **Auto-invoquer** notre fonction anonyme ;
 - ▶ Utiliser un **évènement pour déclencher** l'exécution de notre fonction.

Fonctions

► Exemple :

```
//Enfermer le code de notre fonction dans une variable
//et utiliser la variable comme une fonction

let x= function(){
    alert("Bonjour tout le monde ");
}
x();

//Auto-invoquer notre fonction anonyme

(function(){
    alert("Bonjour tout le monde ");
})();

//Utiliser un évènement pour déclencher la fonction

let p= document.getElementById('pl');
p.addEventListener('click', function(){alert("Bonjour tout le monde ")});
```

Fonctions

- ▶ **La fonction fléchée** est une fonction qui possède une syntaxe très **compacte**, ce qui la rend **très rapide à écrire**.
- ▶ **Syntaxe générale :**
Let NomVariable = (arg1,arg2...) => { bloc d'instructions}
- ▶ S'il n'y a qu'**un seul argument**, nous pouvons l'écrire **sans parenthèses ()**
- ▶ Si le **bloc d'instructions** ne se compose que **d'une instruction**, nous pouvons l'écrire **sans accolades {}** et **sans l'instruction return**

Fonctions

► Exemple :

```
//fonction classique
```

```
function som1(nb1,nb2) {  
  return nb1+nb2  
}
```

```
// fonction anonyme ou expression de fonction  
let som2=function(nb1,nb2) { return nb1+nb2};
```

```
// fonction fléchée  
let som3=(nb1,nb2)=>nb1+nb2;
```

Fonctions

- ▶ La **portée d'une variable** désigne l'espace du script dans laquelle elle va être accessible. on peut distinguer deux types de portée:
 - ▶ **Portée locale** : la variable est accessible uniquement à l'intérieur du bloc (for, if, fonction) où elle est définie (**variable locale**)
 - ▶ **Portée globale** : la variable est accessible depuis tout le script. (**variable globale**)

Fonctions

► Exemple 1 :

```
let x=40, y=44;
function affiche1(){
    alert("dans affiche1 x= "+ x+ " et y= "+y)
}
function affiche2(){
    let x=1,y=2;
    alert("dans affiche2 x= "+ x+ " et y= "+y)
}
affiche1();    //affiche x=40 et y=44
affiche2();    //affiche x=1 et y=2
alert("dans l'espace globale x= "+ x+ " et y= "+y)
    |      |      |    //affiche x=40 et y=44
```

Fonctions

► Exemple 2 :

```
let x=30 , i=33;  
// x et i sont des variables globales  
for(let i = 0; i < 3; i++){  
  // i est une variable locale de for  
  alert('dans la boucle : i = ' + i + ' et x= '+x);  
  if(i == 2){  
    let i=20;  
    // i est une variable locale de if  
    alert('dans if : i = ' + i + ' et x= '+x);  
  }  
}  
alert('hors la boucle : i = ' + i + ' et x= '+x);
```


Fonctions


- **Exemple 3** : Montre la différence entre **let** et **var** (diapositive 23)

```
function affiche() {  
    let x=1;  
    var y=2;  
    if(true) {  
        let x=10; // variable locale de if  
        var y=20; // la même variable précédente  
        alert("Dans if x= "+ x+ "et y= "+y)  
    }  
    alert("avant if x= "+ x+ "et y= "+y)  
}  
affiche();
```

Fonctions

- ▶ **Remarque :**
- ▶ Il est conseillé d' :
 - ▶ Utiliser le mot clef **let** dans la déclaration que le mot clef **var**.
 - ▶ Utiliser des **noms** de variables **différents** pour faciliter la **compréhension** du script et éviter toute ambiguïté.

Fonctions

- ▶ **Une fonction de rappel** (callback en anglais) est une fonction passée en paramètre d'une autre fonction.
- ▶ Exemple :

- ▶ Attention : ne pas utiliser les parenthèses dans l'appel de la fonction callback.

```
//définition de la fonction de rappel
function afficher(p){
|   alert(p)
}
//définition de la fonction qui reçoit
// la fonction de rappel en paramètre
function calculer (n, m, fonct){
let som = n + m
|   fonct(som)
}
// appeler la fonction calculer avec
// la fonction d'appeler en argument
calculer(6, 7, afficher)
```

Fonctions

► Exercices

Synchrone / asynchrone

- ▶ un code **synchrone** va s'exécuter ligne après ligne en attendant la **fin de l'exécution** de la ligne précédente.
- ▶ un code **asynchrone** va s'exécuter ligne après ligne, mais la ligne suivante **n'attendra pas** que la ligne asynchrone ait fini son exécution.
- ▶ Exemples :
 - ▶ Dans la balise Head, on ajout l'attribut **async** à la balise script pour assurer le chargement du contenu HTML avant l'exécution du JavaScript.
 - ▶ Dans la gestion des événements, on attend l'action de l'utilisateur (clique, survol...) pour effectuer la réaction adéquate)

Gestion du délai d'exécution

- ▶ La méthode **setTimeout** définit une action à exécuter et un délai avant son exécution. Elle retourne un identificateur pour le processus.
 - ▶ Syntaxe: `let id = setTimeout(maFonction, délai en millisecondes)`.
 - ▶ Exemple : `let id = setTimeout(()=>alert("bip"), 5000);`
- ▶ La méthode **clearTimeout** interrompt le délai et l'exécution du code associé à ce délai.
- ▶ Le processus à supprimer est reconnu par l'identificateur retourné par *setTimeout*.
 - ▶ Syntaxe : `clearTimeout(identifieur);`
 - ▶ Exemple : `let id = setTimeout(()=>alert("bip"), 5000);`
`clearTimeout(id);`

Gestion du délai d'exécution

- ▶ La méthode **setInterval**, elle déclenche répétitivement la même action à intervalles réguliers.
 - ▶ Syntaxe : `let id = setInterval(maFonction, délai)`
 - ▶ Exemple : `let id = setInterval(()=>alert("bip"), 5000);`
- ▶ La méthode **clearInterval** Stoppe le processus déclenché par `setInterval`.
- ▶ Exemple:

```
let id=setInterval(()=>document.write("*"), 1000);  
setTimeout(()=>clearInterval(id), 10000);
```

Gestion du délai d'exécution

▶ Exercice