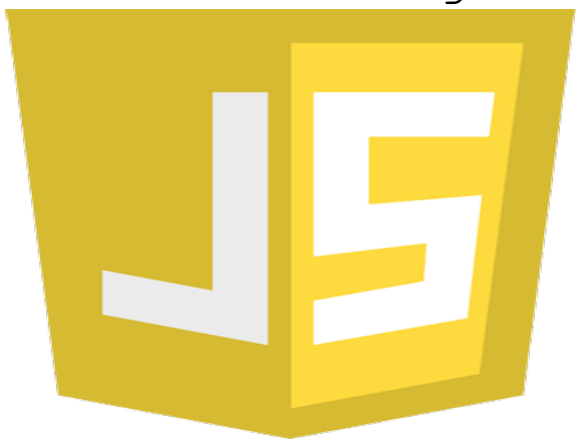
# **JavaScript** : fondamentaux

**Achref El Mouelhi**

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



# Plan

# Plan

# JavaScript

## Une variable ?

- Un pointeur vers une zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

# JavaScript

## Une variable ?

- Un pointeur vers une zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

## **JavaScript** est un langage de programmation faiblement typé

- Pas de type à préciser pour les variables
- Une variable peut changer de type et de valeur

# JavaScript

## Caractéristiques d'une variable

- Son nom dans le programme
- Son type

# JavaScript

## Conventions de nommage des variables en **JavaScript**

- Choisir des noms courts et significatifs
- Utiliser **camelCase**
- Les noms de variables peuvent contenir des lettres, des chiffres, des soulignements (_) et des signes de dollar (`$`)
- Le nom de la variable ne peut pas commencer par un chiffre
- Les noms de variables sont sensibles à la casse : `maVariable` et `MaVariable` sont deux variables distinctes.
- Éviter les conflits avec les mots-clés **JavaScript** tels que `if`, `for`, `class`...
- Utiliser des noms explicites pour les booléens comme `estMajeur`...
- Bien que les navigateurs modernes puissent souvent gérer les caractères spéciaux dans les noms de variables, il est préférable de rester sur des caractères **ASCII standard** et d'éviter d'utiliser les caractères spéciaux.

# JavaScript

## Déclaration d'une variable avec le mot clé `var`

```
var x;
```

# JavaScript

## Déclaration d'une variable avec le mot clé `var`

```
var x;
```

## Initialisation

```
x = 0;
```

# JavaScript

## Déclaration d'une variable avec le mot clé `var`

```
var x;
```

## Initialisation

```
x = 0;
```

## Déclaration + initialisation

```
var y = 5;
```

# JavaScript

**Déclaration d'une variable avec le mot clé `var`**

```
var x;
```

**Initialisation**

```
x = 0;
```

**Déclaration + initialisation**

```
var y = 5;
```

**Il est possible de déclarer simultanément plusieurs variables**

```
var x = 0, y = 5;
```

# JavaScript

**Initialisation d'une variable non-déclarée $\Rightarrow$ déclaration**

```
z = 5;
```

# JavaScript

**Initialisation d'une variable non-déclarée $\Rightarrow$ déclaration**

```
z = 5;
```

**Affectation d'une variable non-déclarée**

```
t = x + y;
```

# JavaScript

**Utiliser une variable non-déclarée et non-initialisée $\Rightarrow$ ReferenceError**

```
alert (v);
```

# JavaScript

**Utiliser une variable non-déclarée et non-initialisée $\Rightarrow$ ReferenceError**

```
alert (v);
```

**Une variable déclarée mais non-initialisée a par défaut la valeur** `undefined`

```
var w;
alert (w);
```

# JavaScript

## Quelques types primitifs utilisés en **JavaScript**

- number
- string
- boolean
- undefined
- null
- ...

# JavaScript

## `undefined` vs `null`

- `undefined` : une variable déclarée mais non initialisée aura automatiquement la valeur `undefined`
- `null` :
  - généralement utilisé de manière intentionnelle pour indiquer l'absence de valeur ou une valeur nulle
  - souvent assigné manuellement par un développeur pour indiquer qu'une variable ne contient aucune valeur significative

# JavaScript

## Quelques types complexes utilisés en **JavaScript**

- object
- array
- function
- map
- set
- date
- ...

# JavaScript

**Pour récupérer le type de valeur affectée à une variable**

```
console.log(typeof 5);
// affiche number

console.log(typeof 5.2);
// affiche number

console.log(typeof true);
// affiche boolean

console.log(typeof "bonjour");
// affiche string

console.log(typeof 'c');
// affiche string

var x;
console.log(typeof x);
// affiche undefined
```

# JavaScript

## Une variable déclarée avec le mot-clé `var` a une visibilité globale

```
{
  var x = 1;
}
console.log(x);
// affiche 1
```

# JavaScript

**Une variable déclarée avec le mot-clé `var` a une visibilité globale**

```
{
  var x = 1;
}
console.log(x);
// affiche 1
```



## Remarque

Une variable déclarée dans un bloc `{ ... }` autre que fonction (`if`, `for`...) a une portée globale.

# JavaScript

## Quelques opérateurs arithmétiques

- $+$ : addition
- $*$ : multiplication
- $-$ : soustraction
- $/$ : division
- $\%$ : reste de la division
- $**$ : exposant

# JavaScript

## Exemple avec la multiplication

```
var x = 4;
var n = x * 2;
console.log(n);
// affiche 8
```

# JavaScript

## Exemple avec la multiplication

```
var x = 4;
var n = x * 2;
console.log(n);
// affiche 8
```



## Ceci retourne `NaN` : Not-a-Number

```
var x = "bonjour";
n = x * 2;
```

# JavaScript

**Pour convertir un `string` en `number`**

```
var x = "1.5"

console.log(parseInt(x));
// affiche 1

console.log(Number.parseInt(x));
// affiche 1

console.log(parseFloat(x));
// affiche 1.5

console.log(Number.parseFloat(x));
// affiche 1.5

console.log(Number(x));
// affiche 1.5
```

# JavaScript

## `Number.parseX()` VS `parseX` VS `Number`

- `parseInt` et `parseFloat` : deux fonctions globales permettant de convertir une chaîne en un entier
- `Number`, `Number.parseInt` et `Number.parseFloat` : trois méthodes statiques introduites **ES6**
  - `Number` : permet de convertir n'importe quelle valeur en un nombre (entier ou réel)
  - `Number.parseInt` : permet de convertir des chaînes en entiers
  - `Number.parseFloat` : permet de convertir des chaînes en réels

# JavaScript

**Quelques cas particuliers avec les fonctions de conversion**

```
console.log(Number(true));
// affiche 1

console.log(Number.parseInt(true));
// affiche NaN

console.log(Number.parseFloat(true));
// affiche NaN

console.log(Number("123.5a"));
// affiche NaN

console.log(Number.parseInt("123.5a"));
// affiche 123

console.log(Number.parseFloat("123.5a"));
// affiche 123.5

console.log(Number("a123.5"));
// affiche NaN

console.log(Number.parseInt("a123.5"));
// affiche NaN

console.log(Number.parseFloat("a123.5"));
// affiche NaN
```

# JavaScript

## Addition (qui peut se transformer en concaténation) et conversion

```
var x = 1;
var y = 3;
var z = '8';
var t = "2";
var u = "bonjour";
var v = "3bonjour";
var m;
var n = 2.5;
console.log(x + y); // 4
console.log(x + z); // 18
console.log(x + parseInt(z)); // 9
console.log(x + t); // 12
console.log(x + y + z); // 48
console.log(z + y + x); // 831
console.log(x + u); // 1bonjour
console.log(x + parseInt(u)); // NaN
console.log(x + v); // 13bonjour
console.log(x + parseInt(v)); // 4
console.log(u + v); // bonjour3bonjour
console.log(x + m); // NaN
console.log(x + n); // 3.5
```

# JavaScript

**Pour vérifier si une chaîne de caractère contient un nombre**

```
var x = "1"
console.log(isNaN(x));
// affiche false

x = "a"
console.log(isNaN(x));
// affiche true
```

# JavaScript

**Pour vérifier si une chaîne de caractère contient un nombre**

```
var x = "1"
console.log(isNaN(x));
// affiche false

x = "a"
console.log(isNaN(x));
// affiche true
```



## Autres méthodes `isX`

- `isFinite` : vérifie si une valeur est finie (c'est-à-dire, ni NaN ni infinie)
- `isArray` : vérifie si une valeur est un tableau
- ...

# JavaScript

**Attention, l'opérateur + pour les chaînes de caractères est différent de tous les autres opérateurs arithmétiques**

```
var a = "2";
var b = '3';
var resultat = a * b;
console.log(resultat);
// affiche 6
```

# JavaScript

**Attention, l'opérateur + pour les chaînes de caractères est différent de tous les autres opérateurs arithmétiques**

```
var a = "2";
var b = '3';
var resultat = a * b;
console.log(resultat);
// affiche 6
```

**Cependant ce code génère un `NaN`**

```
var a = "bon";
var b = 'jour';
var resultat = a * b;
console.log(resultat);
// affiche NaN
console.log(b + parseInt(a));
// affiche jourNaN
```

# JavaScript

**En JavaScript, une division par 0 ne génère pas d'erreur**

```
a = 2;
b = 0;
console.log(a / b);
```

# JavaScript

**En JavaScript, une division par 0 ne génère pas d'erreur**

```
a = 2;
b = 0;
console.log(a / b);
```

**Pour convertir une chaîne en entier**

```
var a = '4';
b = 2;
console.log(b + parseInt(a));
```

# JavaScript

## Quelques raccourcis

- `i++;` ≡ `i = i + 1;`
- `i-;` ≡ `i = i - 1;`
- `i += 2;` ≡ `i = i + 2;`
- `i -= 3;` ≡ `i = i - 3;`
- `i *= 2;` ≡ `i = i * 2;`
- `i /= 3;` ≡ `i = i / 3;`
- `i %= 5;` ≡ `i = i % 5;`

# JavaScript

## Exemple de post-incrémentation

```
var i = 2;
var j = i++;
console.log(i); // affiche 3
console.log(j); // affiche 2
```

# JavaScript

## Exemple de post-incrémentation

```
var i = 2;
var j = i++;
console.log(i); // affiche 3
console.log(j); // affiche 2
```

## Exemple de pre-incrémentation

```
var i = 2;
var j = ++i;
console.log(i); // affiche 3
console.log(j); // affiche 3
```

# JavaScript

**Pour permuter le contenu de deux variables, on peut utiliser la décomposition**

```
a = 2;
b = 0;
[a, b] = [b, a];
```

# JavaScript

**Pour permuter le contenu de deux variables, on peut utiliser la décomposition**

```
a = 2;
b = 0;
[a, b] = [b, a];
```

**Pour évaluer une expression arithmétique exprimée sous forme de chaîne de caractères (EvalError si l'expression est mal formulée)**

```
var str = "2 + 5 * 3";
console.log(eval(str));
// affiche 17
```

# JavaScript

**L'opérateur unaire + peut être utilisé pour convertir en nombre**

```
console.log(typeof (+"3"));
// affiche number

console.log(+true);
// affiche 1
```

# JavaScript

**L'opérateur unaire + peut être utilisé pour convertir en nombre**

```
console.log(typeof (+"3"));
// affiche number

console.log(+true);
// affiche 1
```

**L'opérateur unaire - peut être utilisé pour la négation ou la conversion en nombre**

```
console.log(typeof (-"-3"));
// affiche number

console.log(-"-3");
// affiche 3

console.log(-true);
// affiche -1
```

# JavaScript

## Méthodes utiles pour les chaînes de caractères

- `length` : la longueur de la chaîne
- `toUpperCase()` : pour convertir une chaîne de caractères en majuscule
- `toLowerCase()` : pour convertir une chaîne de caractères en minuscule
- `trim()` : pour supprimer les espaces au début et à la fin
- `substr()` : pour extraire une sous-chaîne de caractères
- `indexOf()` : pour retourner la position d'une sous-chaîne dans une chaîne, -1 sinon.
- ...

# JavaScript

## Pour connaître la longueur d'une chaîne

```
var str = "bonjour";
console.log(str.length);
// affiche 7
```

# JavaScript

## Pour connaître la longueur d'une chaîne

```
var str = "bonjour";
console.log(str.length);
// affiche 7
```

## Pour supprimer les espaces au début et à la fin de la chaîne

```
var str = "  bon jour  ";
console.log(str.length);
// affiche 12

var sansEspace = str.trim();
console.log(sansEspace.length);
// affiche 8
```

# JavaScript

**Pour extraire une sous-chaîne à partir de l'indice 3 jusqu'à la fin**

```
var str = "bonjour";
console.log(str.substr(3));
// affiche jour
```

# JavaScript

**Pour extraire une sous-chaîne à partir de l'indice 3 jusqu'à la fin**

```
var str = "bonjour";
console.log(str.substr(3));
// affiche jour
```

**On peut aussi préciser le nombre de caractère à extraire**

```
var str = "bonjour";
console.log(str.substr(3, 2));
// affiche jo
```

# JavaScript

**Pour extraire une sous-chaîne à partir de l'indice 3 jusqu'à la fin**

```
var str = "bonjour";
console.log(str.substr(3));
// affiche jour
```

**On peut aussi préciser le nombre de caractère à extraire**

```
var str = "bonjour";
console.log(str.substr(3, 2));
// affiche jo
```

**Pour extraire les trois derniers caractères, on utilise une valeur négative**

```
var str = "bonjour";
console.log(str.substr(-3)); //eq substr(4) avec 4 = length - 3
// affiche our
```

# JavaScript

**Ne pas confondre `substr()` avec `substring()` qui elle prend comme paramètre l'indice de début et l'indice de fin (non-inclus)**

```
var str = "bonjour";
console.log(str.substring(1, 3));
// affiche on
```

# JavaScript

**Pour déterminer l'indice d'une sous chaîne dans une chaîne de caractère**

```
var str = "bonjour";
console.log(str.indexOf("jour"));
// affiche 3
```

# JavaScript

**Pour déterminer l'indice d'une sous chaîne dans une chaîne de caractère**

```
var str = "bonjour";
console.log(str.indexOf("jour"));
// affiche 3
```

**S'il n'y a aucune occurrence, elle retourne -1**

```
var str = "bonjour";
console.log(str.indexOf("soir"));
// affiche -1
```

# JavaScript

**Pour accéder à un caractère d'indice `i` dans une chaîne de caractères**

```
// soit directement via l'indice
console.log(str[i]);

// soit en faisant l'extraction d'une sous chaine de
    caractère
console.log(str.substr(i, 1));

// soit avec la méthode d'extraction de caractère
console.log(str.charAt(i));
```

# JavaScript

## Remarque

Appeler une de ces méthodes à partir d'une variable ayant la valeur `undefined` génère une erreur nommée **TypeError**.

# JavaScript

## Exécuter si une condition est vraie

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

# JavaScript

## Exécuter si une condition est vraie

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

## Exemple

```
var x = 3;
if (x > 0)
{
   console.log(x + " est strictement positif");
}
```

# JavaScript

## Opérateurs de comparaison

- == : pour tester l'égalité des valeurs
- != : pour tester l'inégalité des valeurs
- === : pour tester l'égalité des valeurs et des types
- !== : pour tester l'inégalité des valeurs ou des types
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

# JavaScript

## Exercice

Écrire un code **JavaScript** qui demande à l'utilisateur de saisir un entier positif et qui affiche ensuite sa parité (sans `else`).

# JavaScript

**Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`**

```
if (condition1) {
   ...
}
else {
   ...
}
```

# JavaScript

**Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`**

```
if (condition1) {
   ...
}
else {
   ...
}
```

**Exemple**

```
var x = 3;
if (x > 0)
{
   console.log(x + " est strictement positif");
}
else
{
  console.log(x + " est négatif ou nul");
}
```

# JavaScript

**On peut enchaîner les conditions avec `else if` (et avoir un troisième bloc voire ... un nième)**

```
if (condition1)
{
    ...
}
else if (condition2)
{
    ...
}
...
else
{
    ...
}
```

# JavaScript

## Exemple

```
var x = -3;
if (x > 0)
{
  console.log(x + " est strictement positif");
}
else if (x < 0)
{
  console.log(x + " est strictement négatif");
}
else
{
  console.log(x + " est nul");
}
```

# JavaScript

## Exercice

Écrire un code **JavaScript** qui

- demande à l'utilisateur de saisir trois entiers `a`, `b` et `c`
- affiche le résultat de l'équation $\texttt{a}x^2 \texttt{ + b}x \texttt{ + c = 0}$.

# JavaScript

## Opérateurs logiques

- `&&` : ET
- `||` : OU
- `!` : NON
- `^` : XOR

# JavaScript

## Opérateurs logiques

- && : ET
- || : OU
- ! : NON
- ^ : XOR



**Tester plusieurs conditions (en utilisant des opérateurs logiques)**

```
if (condition1 && !condition2 || condition3) {
...
}
```

# JavaScript

## Exercice 1

Écrire un code **JavaScript** qui

- demande à l'utilisateur de saisir une année (un entier),
- affiche si l'année saisie est bissextile (voir `https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`).

# JavaScript

## Exercice 2

Écrire un code **JavaScript** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b` différents de zéro,
- affiche le signe du résultat de la multiplication sans calculer le produit.

# JavaScript

## Exercice 3

Écrire un code **JavaScript** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b`,
- détermine et affiche si le résultat de l'addition (sans calculer la somme) est pair ou impair.

# JavaScript

## Structure conditionnelle `switch` : syntaxe

```
switch (nomVariable) {
  case constante-1:
    groupe-instructions-1;
    break;
  case constante-2:
    groupe-instructions-2;
    break;
  ...
  case constante-N:
    groupe-instructions-N;
    break;
  default:
    groupe-instructions-par-défaut;
}
```

# JavaScript

## Remarques

- Le `switch` permet **seulement** de tester l'égalité
- Le `break` permet de quitter le `switch` une fois le bloc de case est vérifier
- Il est possible de regrouper plusieurs `case`
- Le bloc `default` est facultatif, il sera exécuter si la valeur de la variable ne correspond à aucune constante de `case`

# JavaScript

## Structure conditionnelle avec `switch`

```
var x = 5;
switch (x) {
  case 1:
    alert('un');
    break;
  case 2:
    alert('deux');
    break;
  case 3:
    alert('trois');
    break;
  default:
    alert("autre");
}
```

# JavaScript

## Un multi-case pour un seul traitement

```
var x = 5;
switch (x) {
  case 1:
  case 2:
    alert('un ou deux');
    break;
  case 3:
    alert('trois');
    break;
  case 4:
  case 5:
    alert('quatre ou cinq');
    break;
  default:
    alert("autre");
}
```

# JavaScript

## La variable dans `switch` peut être

- un nombre
- un caractère
- une chaîne de caractère (contrairement aux C++, C...)

# JavaScript

## Exercice

Écrire un programme qui demande à l'utilisateur de saisir l'indice d'un mois (entier compris entre 1 et 12) et qui retourne le nombre de jours de ce mois

- si l'entier est égal à 1, 3, 5, 7, 8, 10 ou 12 le programme affiche 31
- sinon si l'entier est égal à 4, 6, 9 ou 11 le programme affiche 30
- sinon si l'entier est égal à 2, le programme demande à l'utilisateur de saisir l'année et lui retourne 29 si l'année est bissextile, 28 sinon (voir `https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`)
- pour toute autre valeur, le programme affiche une erreur

## Considérons l'exemple suivant

```
var nbr = prompt('Saisir un nombre');
if (nbr % 2 == 0) {
  alert ("pair);
}
else {
  alert ("imapir");
}
```

## Considérons l'exemple suivant

```
var nbr = prompt('Saisir un nombre');
if (nbr % 2 == 0) {
  alert ("pair);
}
else {
  alert ("imapir");
}
```

## Simplifions l'écriture avec l'expression ternaire

```
var nbr = prompt('Saisir un nombre');
nbr % 2 == 0 ? alert("pair") : alert('impair');
```

## Considérons l'exemple suivant

```
var nbr = prompt('Saisir un nombre');
if (nbr % 2 == 0) {
  alert ("pair);
}
else {
  alert ("imapir");
}
```

## Simplifions l'écriture avec l'expression ternaire

```
var nbr = prompt('Saisir un nombre');
nbr % 2 == 0 ? alert("pair") : alert('impair');
```

## Ou

```
alert(nbr % 2 == 0 ? "pair" : 'impair');
```

# JavaScript

## Remarques

- En **JavaScript**, les valeurs suivantes sont considérées comme `falsy` lorsqu'elles sont évaluées dans un contexte booléen :
  - `false` : la valeur booléenne.
  - `null` ou `undefined` : la valeur nulle.
  - `0` ou `-0` : la valeur entière zéro.
  - `0.0` : la valeur flottante zéro.
  - `''` ou `""` : la chaîne de caractères vide.
  - `NaN`
- Toutes les autres valeurs sont considérées comme `truthy` lorsqu'elles sont évaluées dans un contexte booléen, ce qui signifie qu'elles sont considérées comme vraies.

# JavaScript

**Les valeurs `falsy` ne sont pas toutes égales même si elles sont toutes converties en `false` dans un contexte booléen**

```
console.log(false == 0);
// affiche true

console.log(false == "");
// affiche true

console.log(false == null);
// affiche false

console.log(false == undefined);
// affiche false

console.log(null == undefined);
// affiche true

console.log(0 == "");
// affiche true

console.log(0 == null);
// affiche false

console.log(0 == undefined);
// affiche false

console.log(null == "");
// affiche false

console.log(undefined == "");
// affiche false
```

# JavaScript

**L'opérateur `!!` est une double négation logique qui sert à convertir une valeur en un booléen (`true` ou `false`)**

```
console.log(!!"hello");
// affiche true (car "hello" est une valeur truthy)

console.log(!!0);
// affiche false (car 0 est une valeur falsy)

console.log(!!null);
// affiche false (car null est une valeur falsy)

console.log(!!undefined);
// affiche false (car undefined est une valeur falsy)

console.log(!!123);
// affiche true (car un nombre différent de 0 est truthy)

console.log(!!"");
// affiche false (car une chaîne vide est falsy)
```

# JavaScript

## Explication

- Le premier `!` (négation logique) transforme une valeur en son opposé booléen :
  - Une valeur `truthy` devient `false`.
  - Une valeur `falsy` devient `true`.
- Le second `!` annule l'effet du premier, revenant ainsi à la valeur booléenne d'origine (`true` ou `false`).

# JavaScript

**Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements**

```
while (condition[s]) {
   ...
}
```

# JavaScript

**Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements**

```
while (condition[s]) {
   ...
}
```



Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

# JavaScript

## Exemple

```
var i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

# JavaScript

## Exemple

```
var i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

## Le résultat est

```
0
1
2
3
4
```

# JavaScript

## Exercice 1

Écrire un script **JavaScript** qui permet d'afficher les nombres pairs inférieurs à 10.

# JavaScript

## Exercice 2

Écrire un script **JavaScript** qui

- demande à l'utilisateur de saisir une chaîne de caractères
- compte puis affiche le nombre de voyelles définies dans la chaîne saisie

# JavaScript

**La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition**

```
do {
   ...
}
while (condition[s]);
```

# JavaScript

**La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition**

```
do {
   ...
}
while (condition[s]);
```



Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

# JavaScript

## Exemple

```
var i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
```

# JavaScript

## Exemple

```
var i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
```



## Le résultat est

```
0
1
2
3
4
```

# JavaScript

## Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```

# JavaScript

## Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```



Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

# JavaScript

## Exemple

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
```

# JavaScript

## Exemple

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
```

## Le résultat est

```
0
1
2
3
4
```

# JavaScript

## Exercice 1

Écrire un code **JavaScript** qui permet d'afficher les nombres pairs compris entre 0 et 10.

# JavaScript

## Exercice 1

Écrire un code **JavaScript** qui permet d'afficher les nombres pairs compris entre 0 et 10.

### Première solution

```
for (var i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    console.log(i);
  }
}
```

# JavaScript

## Exercice 1

Écrire un code **JavaScript** qui permet d'afficher les nombres pairs compris entre 0 et 10.

### Première solution

```
for (var i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    console.log(i);
  }
}
```

### Deuxième solution

```
for (var i = 0; i < 10; i += 2) {
  console.log(i);
}
```

# Python

## Exercice 2

Écrire un code script **JavaScript** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b` (avec `a < b`),
- afficher les nombres pairs compris entre `a` et `b`.

# Python

## Exercice 3

Écrire un programme qui demande à l'utilisateur de saisir un entier positif `n` et qui calcule puis affiche la somme de tous les entiers positifs inférieurs à `n`.

# Python

**Étant donnée la chaîne de caractères suivante**

```
maChaine = "Bonjour tout le monde. On est dans le
   sud. Et il fait bon.";
```

# Python

**Étant donnée la chaîne de caractères suivante**

```
maChaine = "Bonjour tout le monde. On est dans le
   sud. Et il fait bon.";
```

## Exercice 4

Écrire un script **JavaScript** qui permet de compter le nombre de phrases et celui de mots de la chaine de caractères `maChaine`.

# JavaScript

## Exemple avec `break`

```
var j = 5;
do {
    console.log(j);
    if (j == 3) {
        break;
    }
    j--;
} while (j > 0);
```

# JavaScript

## Exemple avec `break`

```
var j = 5;
do {
    console.log(j);
    if (j == 3) {
        break;
    }
    j--;
} while (j > 0);
```



## Résultat

```
5
4
3
```

# JavaScript

**Le `break` avec un label permet de sortir de plusieurs niveaux de structure de contrôle imbriqués, pas seulement du `switch`**

```
outer: for (let i = 0; i < 3; i++) {
    for (let j = 0; j < 3; j++) {
        if (j === 1) break outer; // sort des deux boucles
        console.log(i, j);
    }
}
```

# JavaScript

**Le `break` avec un label permet de sortir de plusieurs niveaux de structure de contrôle imbriqués, pas seulement du `switch`**

```
outer: for (let i = 0; i < 3; i++) {
    for (let j = 0; j < 3; j++) {
        if (j === 1) break outer; // sort des deux boucles
        console.log(i, j);
    }
}
```



## Explication

- Utilisation de `break outer;` pour quitter les deux boucles.
- Nécessite de nommer le bloc à quitter.
- Uniquement avec `break`, pas `continue`.

# JavaScript

**Exemple avec** `continue`

```
var j = 5;
while (j > 0) {
    if (j == 3) {
        j--;
        continue;
    }
    j--;
    console.log(j);
}
```

# JavaScript

**Exemple avec** `continue`

```
var j = 5;
while (j > 0) {
    if (j == 3) {
        j--;
        continue;
    }
    j--;
    console.log(j);
}
```



**Résultat**

```
4
3
1
0
```

# JavaScript

## Remarque

Il est préférable d'utiliser `break` et `continue` judicieusement, en tenant compte de leur impact sur la lisibilité et la maintenabilité du code.

# JavaScript

## Une constante

un élément qui ne peut changer de valeur

# JavaScript

## Une constante

un élément qui ne peut changer de valeur

## Pour déclarer une constante

il faut utiliser le mot-clé `const`

# JavaScript

## Une constante

un élément qui ne peut changer de valeur

## Pour déclarer une constante

il faut utiliser le mot-clé `const`

### Déclaration d'une constante

```
const PI = 3.1415;
```

# JavaScript

## Une constante

un élément qui ne peut changer de valeur

## Pour déclarer une constante

il faut utiliser le mot-clé `const`

**Déclaration d'une constante**

```
const PI = 3.1415;
```

**L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)**

```
PI = 5;
```

# JavaScript

**Considérons l'objet suivant déclaré avec le mot-clé `const`**

```
const obj = {
    nom: "wick",
    prenom: "john"
};
```

# JavaScript

**Considérons l'objet suivant déclaré avec le mot-clé `const`**

```
const obj = {
    nom: "wick",
    prenom: "john"
};
```

**L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)**

```
obj = {};
```

# JavaScript

**Considérons l'objet suivant déclaré avec le mot-clé `const`**

```
const obj = {
    nom: "wick",
    prenom: "john"
};
```

**L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)**

```
obj = {};
```

**Cependant, l'instruction suivante est correcte et ne lève donc pas d'exception**

```
obj.nom ="travolta";
obj['prenom'] = "denzel";
obj.age = 50;

console.log(obj);
// affiche {nom: "travolta", prenom: "denzel", age: 50}
```

# JavaScript

## Idem pour les tableaux

```
const tableau = [2, 3, 8];
```

# JavaScript

**Idem pour les tableaux**

```
const tableau = [2, 3, 8];
```

**L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)**

```
tableau = [];
```

# JavaScript

**Idem pour les tableaux**

```
const tableau = [2, 3, 8];
```

**L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)**

```
tableau = [];
```

**Cependant, l'instruction suivante est correcte et ne lève donc pas d'exception**

```
tableau[2] = 1;
tableau[0] = 9;

console.log(tableau);
// affiche [9, 3, 1]
```