

LANGAGE JAVASCRIPT :

LES BASES DU LANGAGES

LA STRUCTURE DE BASE D'UN PROGRAMME EN JAVASCRIPT

Lorsque le code JavaScript se trouve dans un fichier séparé (extension .js) La structure générale d'un programme écrit en JavaScript est relativement simple.

```
/* zone d'en-tête */

/* déclaration des variables globales */
Declaration1;
Declaration2;

/* définition des fonctions (mot clef function) */

/* CORPS DU PROGRAMME */
Instruction1;
Instruction2;

/* FIN */
```

Même s'il est possible de déclarer nos variables à n'importe quels moments, il est conseillé de le faire en début de script pour une meilleure lisibilité du code.

Lorsque le JavaScript est écrit dans une page Web la structure est différente. Dans un premier temps nous allons travailler uniquement sur la partie JavaScript. Cela donne la structure suivante :

```
<!DOCTYPE html>
<html>
  <head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <script>
      //ici les variables globales à la page et les fonctions
    </script>

  </head>
  <body>
    <script>
      //ici les instructions
    </script>
  </body>
</html>
```

Le langage HTML utilise des balises. La balise permettant d'ajouter du JavaScript est `<script> ... </script>`

Cette balise s'ajoute à 2 endroits particulièrement : entre les balises `<head></head>` pour les variables globales et pour les fonctions et entre `<body></body>` pour les instructions.

Quand le code JS se trouve dans un fichier séparée la syntaxe est la suivante

```
<script src="code.js" type="text/javascript"></script>
<script src="code.js"></script>
```

On note ici 2 attributs :

- **src** (pour source): indique le chemin et le nom du fichier source. Ici code.js
- **type** : indique le type de script. Cet attribut est optionnel (cf 2^{ème} exemple)

LES COMMENTAIRES

Les commentaires permettent une relecture de programme pour le développeur lui-même, mais aussi pour ceux qui vont le relire (travail en équipe). Un commentaire d'entête est nécessaire pour indiquer le nom du programmeur, la date, le nom du fichier, descriptif du programme. Les commentaires sont du texte simple et ne sont donc pas interprétés comme étant du code.

2 types de commentaires sont possibles :

- `//` indiquent au compilateur que le reste de la ligne est un commentaire.
- `/* ... */` indiquent au compilateur qu'à partir de `/*` le texte est un commentaire jusqu'au symbole `*/`. Permet des commentaires sur plusieurs lignes.

```
/*
 * File:   HelloWorld.js
 * Author: sgibert
 *
 * Created on 14 septembre 2018, 20:19
 *
 * Description du programme : explique ici ce que fait ton programme
 */
```


Il n'est pas utile d'associer une ligne de code avec un commentaire d'autant plus si ce dernier ne fait que dire ce que fait le code.

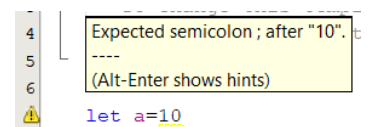
Exemple :

```
// je déclare une variable x = 3 : ce commentaire n'a aucun intérêt
// il vaut mieux expliquer à quoi sert la variable x dans notre code.
let x=3;
```

LES DECLARATIONS ET INSTRUCTIONS

Une instruction ou une déclaration se termine toujours par un point-virgule (;) C'est le cas de nombreux langages de programmation. Mais attention le langage JavaScript n'est pas aussi strict au niveau de sa syntaxe et autorise l'absence de ; de fin d'instruction.

Cependant, comme illustré, l'IDE NetBeans lève un Warning  indiquant qu'un ; est attendu.



Pour faire simple, même si cela n'est pas obligatoire, cela ne coûte rien d'ajouter systématiquement ce marqueur de fin d'instruction, facilitera l'apprentissage d'autres langages **et sera donc la règle pour nous.**

LES DECLARATION DE VARIABLES

Rappel du cours d'algorithmique :

En résumé : **variable = NOM + TYPE + VALEUR**

JavaScript est dit "faiblement typé" contrairement à d'autres langage dit "fortement typé", par exemple le langage C ou JAVA. Avec ses langages il est nécessaire de préciser le type au moment de la déclaration d'une variable.

Exemple de déclaration de variable en JAVA :

```
int a=3;      //a est un entier et restera un entier pendant tous le programme
float b=3.5;  //b est un réel et restera un réel pendant tous le programme
char c='a';   //c est un caractère et restera un caractère pendant tous le programme
```

Avec JavaScript une variable est déclarée avec la commande **var** ou **let**, mais sans indiquer son type. Le type sera fixé à l'utilisation. La notion de type est bien existante mais jamais indiquée de façon explicite dans le code.

Le JavaScript a **7 types dit primitifs** : Number, BigInt, String, Boolean, Null, Undefined, et Symbole, et **1 type Object**.

Pour plus de précision sur le sujet voir le lien suivant :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Structures_de_donn%C3%A9es

Cela donne en Javascript :

```
let a = 3;           // a est un Number
a = "coucou";       // a est maintenant un String
```

Dans ce programme la variable `a` est déclaré comme étant un "Number". Mais au cours du programme elle devient un "String". Bien que cela soit possible on essaiera au maximum ce genre de mélange.

MEMOIRE VIVE

Sur un PC la mémoire vive est une mémoire de travail. Elle stock de façon temporaire des données et des programmes. Lorsque le PC est éteint le contenu de cette mémoire est supprimé. On parle alors de mémoire volatile, contrairement au disque dur d'un ordinateur qui stocke lui aussi des données mais de façon permanente. Cette mémoire est très rapide, et communique directement avec le processeur.

En JavaScript la mémoire est allouée lors de la création des objets puis libérée « automatiquement » lorsque ceux-ci ne sont plus utilisés. Cette libération automatique est appelée *garbage collection* en anglais ou ramasse-miettes.

CHOIX DU NOM

Bien qu'il soit possible de donner des noms de variables avec des caractères accentués en JavaScript on se limitera aux caractères suivants :

- `a..z`, `A..Z`
- `0..9` → interdit au début du nom
- `_` (le caractère souligné ou underscore) → autorisé au début du nom

On ne peut pas utiliser

- Un tiret `-` ou un point `.`
- Utiliser un mot clef du langage comme nom de variable. Exemple de mot clef (`new`, `if`, `typeof`,...)

On utilise l'alternance de minuscule, majuscule, ou `_` (underscore) pour améliorer la lecture du nom.

Rappel 1 : pour la relecture du programme, choisir un nom qui caractérise au mieux la variable.

Rappel 2 : le nom d'une variable est **unique** dans votre programme.

Exemples :

`nbrDeCaisses`, `poste_1`, `poste_123`, etc...

Le langage JavaScript est sensible à la casse. Ainsi une variable `"Poste_1"` n'est pas la même chose que `"poste_1"`

DECLARATION DE VARIABLES ET CONSTANTES

Pour pouvoir être utilisable une variable doit d'abord être déclarés avec **`var`** ou **`let`**. En théorie seulement puisqu'encore une fois le JavaScript est un langage très souple qui autorise de nombreuses choses que d'autres langages refuseraient.

Par exemple le code suivant est accepté :

```
a=10;           // pas de var, pas de let;
alert(a);
```

Pour éviter de prendre de mauvaises habitudes de programmation, il existe un mode "strict" de JS qui oblige la déclaration de variable. Pour plus d'informations sur le sujet : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Strict_mode

```
"use strict";
a=10;           //avec le mode strict : erreur →
alert(a);
```

✖ Uncaught ReferenceError: a is not defined
at `code.js:8`

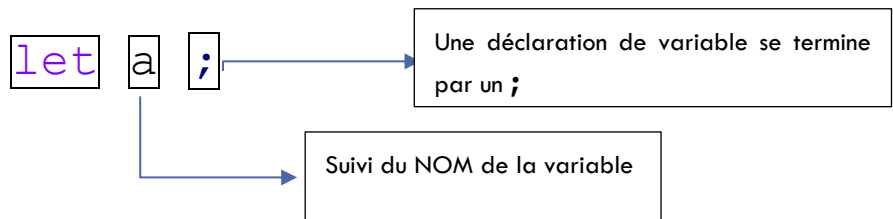
Une variable peut être déclarée n'importe quel moment de 3 façons.

- Avec **var**, quel que soit l'endroit de la déclaration (hormis une fonction) la **variable est globale**, c'est-à-dire accessible à tous point de la page **Web**, ou à tout point du fichier **.JS**. Même déclarée dans un bloc d'instruction la variable est globale
- Avec **let** la **variable est locale**, si déclarée dans un bloc d'instruction entre { }. Locale dans le sens qu'elle n'est accessible ou utilisable uniquement dans ce bloc. Si déclarée en dehors d'un bloc d'instruction la **variable est globale**.

Que choisir entre var et let ? Nous utiliserons **let** car la portée est locale. Nous reviendrons sur ce point trop technique à ce stade des connaissances du langage.

- Enfin il existe un dernier mot clef : **const**. Comme son nom l'indique il permet de déclarer une constante. Ici la constante doit être initialisé au moment de sa création. Son comportement (global ou local) est le même que pour **let**

La syntaxe de base est :



Exemple :

```

<script>
  let a;           // a est ici undefined
  let b=4;         // b est un number vaut 4
  let c,d=3,e;     //3 variables, c et e sont undefined, e est un Number

  const f=3;       //f est une constante de type Number

  var g="Bonjour"; //g est un String
</script>
  
```

LES DIFFERENTS TYPE ET LEUR PLAGE DE LA VALEUR

Tableau récapitulatif :

Type JavaScript	Type Algo	Taille	Plage de valeurs
Number	Entier ou Réel	8 octets	$-(2^{53} - 1)$ et $2^{53} - 1$ pour les Entiers $5e-324$ et $1.7976931348623157e+308$ pour les Réels
BigInt	Entier	?	Permet d'aller au-delà de la limitation de Number.
String	Chaine et Caractère	2 octets à ?	Permet de représenter une chaine de caractère "Bonjour" ou un simple caractère 'c'
Boolean	Booléen	1	2 valeurs possibles : true ou false.
Null	?	?	Une valeur possible : null
undefined	?	?	Une valeur possible : undefined

DETAILS SUR LES DIFFERENTS TYPES

- Number** : représente soit un entier, ou bien un réel (nombre à virgule). Ce type de données peuvent recevoir des valeurs négatives. Dans ce cas-là on précède la valeur d'un (-). Pour une valeur positive il est possible, mais inutile, d'ajouter le signe (+) devant la valeur.

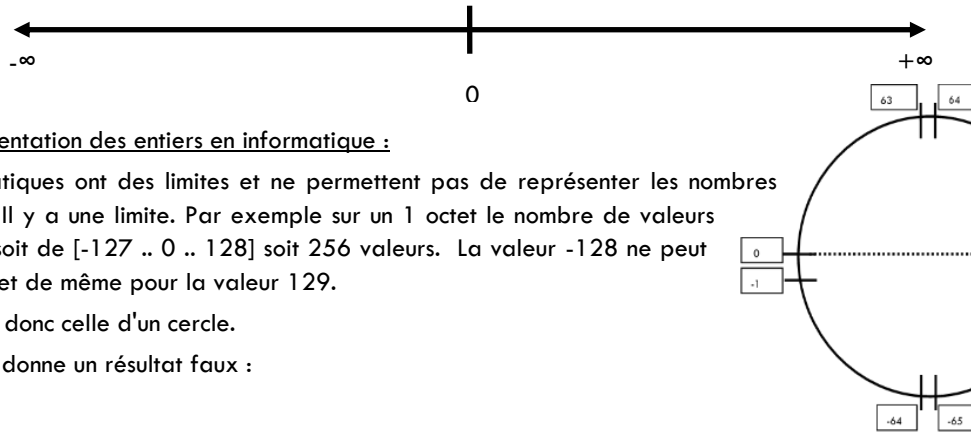
Exemple :

```
let a=+10; //Les 2 déclarations sont équivalentes
let a=10;
let a=Number(10);
```

Si Number est un **entier** sa plage de valeur est comprise entre $-(2^{53} - 1)$ et $2^{53} - 1$, soit entre -9007199254740991 et +9007199254740991. Passé cette plage JavaScript ne garantit plus la cohérence des calculs, et il y aura des incohérences. Pour comprendre ce phénomène il est important de comprendre la différence entre la représentation des nombres en mathématique et en informatique.

Principe de la représentation des entiers en mathématique :

En mathématique existe la notion abstraite de $+\infty$ et $-\infty$. Ainsi tous les nombres peuvent se représenter selon le graphique :



Principe de la représentation des entiers en informatique :

Les machines informatiques ont des limites et ne permettent pas de représenter les nombres entiers de $+\infty$ à $-\infty$. Il y a une limite. Par exemple sur un 1 octet le nombre de valeurs possible est de 2^8 soit de $[-127 .. 0 .. 128]$ soit 256 valeurs. La valeur -128 ne peut pas être représenté, et de même pour la valeur 129.

La représentation est donc celle d'un cercle.

Ainsi le calcul suivant donne un résultat faux :

```
let a=2**53-1; //ici a vaut 9007199254740991 la valeur MAXIMUM pour un Number
a=a+20;        // en théorie a vaut maintenant 9 007 199 254 741 011
                // en réalité cela donne 9007199254741012
```

Avec une erreur de 1 répété beaucoup de fois on peut arriver à de grosses erreurs.

Si Number est un **réel** sa plage de valeurs est comprise entre $5e-324$ et $1.7976931348623157e+308$. Passé cette plage JavaScript propose 2 valeurs pour représenter $+\infty$ (**Infinity**) à $-\infty$ (**-Infinity**). Ces 2 valeurs sont de type "Number" et peuvent être utilisées dans le code

Exemple :

```
let a=(2**2000); // Ici (a) vaut Infinity
let b=-(2**2000); // Ici (b) vaut -Infinity.
let c=Infinity;
console.log(c, typeof(c));
```

- **Le BigInt** : ce type permet de représenter des entiers quand la plage des entiers sur Number est dépassée. Il faut pour cela préfixé chaque valeur d'un petit n.

```
let a=2n**53n-1n; //ici a vaut 9007199254740991
a=a+20n;          // a vaut 9 007 199 254 741 011
```

Pour déclarer ce type de donnée il est possible d'écrire :

```
let a=BigInt(10);
```

- **Le String**: permet la représentation de chaîne de caractères mais aussi de simple caractère. Le langage JavaScript autorise indifféremment l'usage des doubles cotes (") ou simple cote (').

```
let a="coucou";
let b=String('coucou');
```

Chaque caractère formant la chaîne (a) ou (b) est constitué de caractère codé sur 16 bits soit 2 octets. JavaScript utilise la table Unicode. Donc un caractère a une valeur numérique. Ici (a) ou (b) utilise $6 \times 2 = 12$ octets.

Voici une liste de certains caractères spéciaux :

Code	Sortie
\'	Apostrophe
\"	Guillemets
\&	&
\\	Barre oblique inverse
\n	Nouvelle ligne
\t	Tabulation

- **Le boolean** : 2 valeurs sont acceptées : **true** ou **false**.

Exemple :

```
let a=true;           //ici a vaut true
let b=Boolean();      //ici b vaut false la valeur par défaut d'un boolean
console.log(typeof a,typeof b); //a et b sont de type boolean
console.log(a,b);     //affiche true,false
```

EN RESUME

Javascript est faiblement typé mais la notion de type est bien présente. Pour coder de façon la plus propre possible et garder en mémoire que chaque variable a un type voici en résumé comment déclarer les 6 types primitifs indispensables à connaître :

```
let a=Number(10);
let b=BigInt(1000);
let c=String("coucou");
let d=Boolean(true);
let e=undefined; // équivaut à : let e;
let f=null;
```

LES OPÉRATEURS

Il y a 3 grandes catégories d'opérateurs.

LES OPÉRATEURS ARITHMÉTIQUES

Il existe une certaine priorité des opérateurs, comme en mathématique. Pour s'éviter tous problèmes avec ces priorités il suffit de mettre des parenthèses.

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la division entière)
**	Puissance

Le langage JavaScript a la particularité de toujours chercher à retourner quelque chose même quand le calcul n'a pas de logique. Il faut donc faire très attention !!

Quelques exemples :

```
let a=3,b=5,c=a+b;           // c vaut 8 normal
let a=3,b=50.5,c=a+b;       // c vaut 53.5
let a=true,b=50.5,c=a+b;    // c vaut ?
let a=true,b=true,c=a+b;    // c vaut ?
let a=5,b="test",c=a+b;     // c vaut ?
```

Quand une opération n'est vraiment pas possible JavaScript retourne en générale une valeur **NaN** qui signifie **Not A Number**. Comme Infinity, NaN est du type **Number**.

Dans l'exemple qui suit on cherche à diviser un **Number (a)** par un **String (b)** ce qui n'a pas de sens. Le résultat est ici NaN

```
let a=5,b="test",c=a/b;
```

Cas particulier de la division d'entier :

Dans de nombreux cas nous avons besoin de récupérer un résultat de type Entier. Le code suivant ne permet d'obtenir le quotient 2 pour cette opération, mais 2.5. Plusieurs solutions existent pour obtenir le résultat voulu notamment la fonction **parseInt**.

```
let a=5,b=2,c=a/b;           // sans parseInt c vaut 2.5
document.write(parseInt(c)); // avec parseInt c vaut 2
```

Pour obtenir le reste de la division on utilisera l'opérateur %. Avec JavaScript % marche aussi sur les nombres réels. Encore une souplesse qui n'existe pas d'autre langage. A éviter !! On réservera donc % pour la division euclidienne.

```
let a=5,b=2,c=a%b;           // c vaut 1
```

Comme vous l'avez appris en math il ne faut jamais diviser par 0. Dans de nombreux langage ce genre d'opération fait planter le programme par le mécanisme d'exception comme en Java par exemple. Ce n'est pas le cas en JavaScript. Il existe même en JavaScript la valeur -0. Comme nous l'avons déjà dit, JavaScript cherchera toujours à retourner un résultat.

```
let a=5.5,b=0,c=a/b;         // une division par zéro : ici c vaut infinity
let a=5.5,b=-0,c=a/b;       // une division par - zéro : ici c vaut -infinity
```

Les principes énoncés ci-dessus sont les même pour les autres opérateurs.

Attention pour l'opérateur ** (puissance). JavaScript possède également un opérateur binaire ^ (XOR logique, aussi appelé OU Exclusif). ** et ^ sont deux opérateurs bien différents. Par exemple $2 ** 3 === 8$ et $2 ^ 3 === 1$.

LES OPÉRATEURS LOGIQUES

Ce type d'opérateur dans les tests des conditions (if, else if) ou des boucles (for, while, do while). Le résultat d'une opération logique est une valeur booléenne (true ou false)

Opérateur	Signification
!	Non logique
&&	Et logique
	Ou logique
== (double égal) et ===	Égal à, et Egalité strict
!= et !==	Différent de, et Différence Strict
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à

Exemple :

```
let i=4,j=3 ;
let v ;
v=(i<=j) ;           // v vaut : false
v=(i==4 && i > 2) ;   // v vaut ?
```

LES OPÉRATEURS D'AFFECTATION ET D'INCRÉMENTATION

Opérateur	Signification
=	Affectation
+=	Ajout de l'opérande de droite
-=	Suppression de l'opérande de droite
*=	Multiplication de l'opérande de droite
/=	Division de l'opérande de droite
%=	Modulo de l'opérande de droite
**=	Puissance de l'opérande de droite
? :	Affectation conditionnelle
++variable	Pré-incrémentation (avant)
variable++	Post-incrémentation (après)
--variable	Pré-décrémentation
variable--	Post-décrémentation

Quelques exemples simples :

```
let i=3;
i++ ;    // i=i+1 ;
i-- ;    // i=i-1 ;
i+=1 ;   // i=i+1 ;
i*=5 ;   // i=i*5 ;
```

Attention, certains opérateurs sont dits à effet de bord, et s'ils sont mal utilisés (ou mal maîtrisés) peuvent donner des résultats inattendus. Exemple :

```
let a=5,b=1,c=0;
c=++b + a++;    //c vaut ?

let a=5,b=1,c=0;
c=b++ + a++;    //c vaut ?
```

Conseils : Programmez au début en utilisant une façon simple d'incrémenter et d'affecter.

L'affectation conditionnelle :

Syntaxe : **<opérande> = <condition> ?<expression1> :<expression2> ;**

Si condition vaut **true** alors on affecte à opérande expression1, si **false** expression2. C'est l'équivalent du Si...ALORS...SINON

Exemple :

```
let i=3,j=4,k ;
k= i>j ?10 :-10 ;
```


LES FONCTIONS D'ENTRÉE/SORTIE

Les fonctions d'entrée-sortie permettent de "dialoguer" avec un périphérique d'entrée (bien souvent un clavier) et un périphérique de sortie (bien souvent un écran).

Dans un environnement Web, pour commencer simplement nous utiliserons 3 fonctions :

- `Alert` ou plutôt `window.alert(message);`
- `resultat = window.confirm(message);`
 - message** est la chaîne contenant le texte à afficher dans le dialogue.
 - `resultat` est une valeur booléenne indiquant si OK ou Annuler a été sélectionné (`true` signifie OK).
- `resultat = window.prompt(message, default);`
 - message** Une chaîne de caractères qui sera affichée pour l'utilisateur
 - default** : Une chaîne de caractères contenant la valeur par défaut affichée pour la saisie
 - résultat** : La chaîne de caractères qui a été saisie par l'utilisateur ou null