

## N.S.I. – Javascript dans une page HTML

### 1 Présentation de Javascript

Alors que HTML est un langage de description permettant de représenter et structurer une page web et CSS un langage, de description aussi, décrivant la présentation et le style de ces pages<sup>1</sup>, **Javascript** est un **langage de programmation**, permettant d'exécuter des programmes au même titre qu'avec tout autre langage de programmation, exécuté de façon privilégiée « côté client », c'est-à-dire sur l'ordinateur de l'utilisateur et non sur le serveur sur lequel est hébergée la page web.

Les spécificités de **Javascript** qui nous intéresse ici est sa capacité à

- interagir avec l'utilisateur au travers du navigateur et de la page affichée
- et à
- modifier la page affichée, en modifiant le contenu de la page et le style des éléments.

#### 1.1 Inclusion de code Javascript

Javascript est un langage de script, c'est-à-dire que l'on va utiliser Javascript pour exécuter de petits programmes, décrits par un ensemble de lignes de codes (script), composant une suite d'instructions et que l'on va inclure dans les pages web ou des pages annexes de code Javascript.

##### 1.1.a. La balise `script` pour inclure du code dans une page web

Le code Javascript pourra être inclus directement dans la page web entre une paire de balises `<script>...</script>`, soit dans l'entête de la page ( `<head>` ), soit dans le corps de la page ( `<body>` ), le script sera alors exécuté au moment où il est lu par le navigateur, au même titre que le code HTML ou CSS.

Cours_JS_exmp01_window_alert.html	Résultat d'exécution
<pre> &lt;!DOCTYPE html&gt; &lt;html lang= "fr" &gt;   &lt;head&gt;     &lt;title&gt; Cours JS Exemple 01 &lt;/title&gt;     &lt;meta charset= "utf-8" /&gt;   &lt;/head&gt;   &lt;body&gt;     &lt;h3&gt;Premier script JavaScript&lt;/h3&gt;     Affichage dans une fenêtre externe.&lt;br/&gt;     &lt;script&gt;       window.alert("Hello World");     &lt;/script&gt;     &lt;br/&gt;     That's all folks.   &lt;/body&gt; &lt;/html&gt; </pre>	 <p>La fenêtre se ferme lorsque l'on clique sur « OK » :</p> <p><b>Premier script JavaScript</b></p> <p>Affichage dans une fenêtre externe.</p> <p>That's all folks.</p>

Le code Javascript se réduit ici à une seule instruction, qui, lorsqu'elle est exécutée déclenche l'ouverture d'une fenêtre externe dans laquelle s'affiche un message « Hello world » et disposant d'un bouton cliquable, permettant de la fermer.

On peut constater que **le script est réexécuté à chaque réactualisation de la page** du navigateur.

On peut placer autant de scripts que l'on veut dans une page web et à l'endroit que l'on veut.

➤ Exemple : **Cours\_JS\_exmp02\_scripts\_multiples.html**.

C'est le rôle du script qui va déterminer l'endroit où on l'inclut, par exemple, les scripts inclus dans l'entête `<head>` seront de façon privilégiée utilisés pour y définir des fonctions Javascript, appelées par des scripts situés, eux, dans le corps de la page.

<sup>1</sup> une page web est décrite par un code écrit dans un fichier, obtenu auprès d'un serveur suite à une requête, et s'affiche dans une fenêtre grâce au navigateur qui est une application qui lit et interprète ce code pour produire la page attendue.

### 1.1.b. Utilisation de fichiers externes

Le code Javascript peut être inclut dans un (ou plusieurs) fichier(s) externe(s), d'extension **.js**, à l'aide encore de la balise script et avec la syntaxe suivante :

```
<script src="myScript.js"></script>
```

On peut inclure autant de ces balises que l'on veut et le code inclus dans le fichier est lu et exécuté au moment où la balise est lue.

➤ **Exemple : Cours\_JS\_exmp03\_scripts\_externes.html.**

On notera les [avantages](#) suivant en faveur de l'utilisation de scripts externes (qui sont les mêmes que ceux en faveur de l'utilisation de feuilles de style CSS externes) :

- cela permet de séparer HTML et code ;
- facilite la lecture et la maintenance à la fois du code HTML et du code JavaScript ;
- les fichiers JavaScript mis en cache peuvent accélérer le chargement des pages.

## 1.2 Interfaces de sortie

Il existe quatre sortie possibles utilisables en Javascript.

### 1.2.a. Sortie dans une fenêtre externe

On utilise ici une « méthode d'objet HTML », la méthode `alert` d'un [objet HTML de type window](#), ce que nous ne développerons pas ici, mais dont la fonctionnalité est d'ouvrir une fenêtre supplémentaire dans le navigateur avec un message (une chaîne de caractères) destiné à attirer l'attention de l'utilisateur. La syntaxe, déjà vue, pour cette instruction est la suivante :

```
window.alert("ici le message pour l'utilisateur");
```

### 1.2.b. Sortie dans la console du navigateur

Les navigateurs disposent d'une console, destinée aux développeurs<sup>2</sup>, permettant d'afficher par exemple les messages d'erreurs lors de la lecture d'une page HTML et pouvant servir d'interface de sortie pour un script Javascript.

On y accède *via* le menu du navigateur :

- sur Firefox : aller dans le menu, en haut à droite (trois barres horizontales), puis **Développement web > Console web** (raccourci **Ctrl+Maj+K**) > **Console** ;
- sur Chrome : aller dans le menu, en haut à droite (trois points sur une ligne verticale), puis **Plus d'outils > Outils de développement** (raccourci **Ctrl+Maj+I**) > **Console** ;
- sur Edge : aller dans le menu, en haut à droite (trois points sur une ligne horizontale), puis **Plus d'outils > Outils de développement** (raccourci **F12**) > **Console**.

L'instruction Javascript d'affichage est alors :

```
console.log("ici le message, texte ou valeurs", 123);
```

Là encore, même si nous ne développerons pas ce point, il s'agit de l'utilisation d'une *méthode*, `log`, pour un *objet* HTML, [console](#).

Cours_JS_exmp04_console_log.html	Sortie console
<pre>... &lt;body&gt;   ...   &lt;script&gt;     console.log("ici le message, texte ou valeurs", 123);   &lt;/script&gt;   ...   &lt;script&gt;     console.log("message 2");   &lt;/script&gt;   ... &lt;/body&gt; &lt;/html&gt;</pre>	

<sup>2</sup> Cette fonctionnalité est à rapprocher d'autres fonctionnalités comme la possibilité d'[afficher le code source](#) d'une page web affichée par le navigateur.

### 1.2.c. Sortie dans la page web

On utilise ici la page web elle-même comme interface de sortie, soit au niveau du « document » (la page web), dans ce cas l'affichage se fera à l'emplacement de la balise `<script>` contenant le code, soit au niveau d'un élément de la page web (un élément du *document*), ici identifié par un `id`.

La syntaxe est propre à chacune des ces deux méthodes :

- écriture au niveau du **document** : le texte est écrit à l'emplacement de la balise `<script>`

```
document.write("texte dans la page");
```

- écriture au sein d'une balise : le texte remplace le `contenu` de la balise

```
document.getElementById('cible').innerHTML = "modification du \"contenu\""
```

Dans ce dernier exemple, pour l'élément identifié par l'`id` 'cible', c'est le contenu de l'élément (le texte écrit entre la balise ouvrante de l'élément et sa balise fermante) qui est remplacé par la chaîne de caractère spécifiée dans l'instruction.

Dans les deux cas, il s'agit à chaque fois d'une **instruction Javascript** et à ce titre, il faut écrire l'instruction dans une balise de script.

Dans le second cas, l'élément ne sera affecté par le script que si le script est exécuté après le code HTML plaçant l'élément dans la page.

#### Cours\_JS\_exmp05\_sorties\_dans\_la\_page.html

```
<!DOCTYPE html>
<html lang= "fr" >
  <head>
    <title> Cours JS Exemple 05 </title>
    <meta charset= "utf-8" />
    <script>
      document.write("Écriture dans l'entête (malvenu ; inutile ; À ne pas faire !)");
    </script>
  </head>
  <body>
    <h3>Sortie directement dans la page affichée</h3>
    <script>
      document.write("texte dans la page");
    </script>
    <br/>
    <script> <!-- sera sans effet car situé (et donc exécuté) avant
      création de l'élément ciblé-->

      document.getElementById('element_cible').innerHTML = "modification d'un élément
\"identifié\" - essai 1"
    </script>
    <h4 id="element_cible"> titre de niveau 4 </h4>
    <script> <!-- exécuté et produisant un effet, car situé après l'élément ciblé-->
      document.getElementById('element_cible').innerHTML = "modification d'un élément
\"identifié\" - essai 2"
    </script>
    <script> <!-- exécuté mais sans effet, car situé avant l'élément ciblé-->
      document.getElementById('cible2').innerHTML = "sans effet car placé avant l'élément"
    </script>
    <h5 id="cible2"> titre de niveau 5 </h5>
  </body>
</html>
```

#### Résultat

Écriture dans l'entête (malvenu ; inutile ; À ne pas faire !)

##### Sortie directement dans la page affichée

texte dans la page

modification d'un élément "identifié" - essai 2

titre de niveau 5

Ces méthodes sont très puissantes car les éléments de **texte** ajoutés à la page sont **lus** et **interprétés** comme du code HTML et donc il est possible d'ajouter par cette méthode des éléments HTML à la page.

Voici deux exemples où l'on utilise le Javascript pour modifier non pas juste du texte dans la page web, mais pour inclure des éléments nouveaux de code, dont la création de nouveaux éléments à l'aide de balises et l'application de styles CSS.

- Avec la méthode `document.write()`

Cours_JS_exmp06_document_write_html_code.html	
<pre> &lt;!DOCTYPE html&gt; &lt;html lang= "fr" &gt;   &lt;head&gt;     &lt;title&gt; Cours JS Exemple 06 &lt;/title&gt;     &lt;meta charset= "utf-8" /&gt;   &lt;/head&gt;   &lt;body&gt;     &lt;h3&gt;Modification de la page affichée&lt;/h3&gt;     &lt;p&gt;       On exécute un script qui rajoute du code HTML à la page affichée.     &lt;/p&gt;     &lt;p&gt;       Ici un peu de texte:&lt;br/&gt;       &lt;script&gt;         document.write("Hello World 1");       &lt;/script&gt;       &lt;br/&gt;       Ici un titre de niveau 2 :&lt;br/&gt;       &lt;script&gt;         document.write("&lt;h2&gt;Hellow World 2&lt;/h2&gt;");       &lt;/script&gt;       Ici un titre de niveau 2 avec du CSS :&lt;br/&gt;       &lt;br/&gt;       &lt;script&gt;         document.write("&lt;h2 style=\"color:red\"&gt;Hellow World 3&lt;/h2&gt;");       &lt;/script&gt;       &lt;br/&gt;       Ici un titre de niveau 2 avec plus de CSS:&lt;br/&gt;       &lt;br/&gt;       &lt;script&gt;         document.write("&lt;h2 style=\"color:red; " +                         "background-color:yellow\"&gt;" +                         "Hello World 4" +                         "&lt;/h2&gt;");       &lt;/script&gt;     &lt;/p&gt;   &lt;/body&gt; &lt;/html&gt; </pre>	
Résultat	
<p><b>Modification de la page affichée</b></p> <p>On exécute un script qui rajoute du code HTML à la page affichée.</p> <p>Ici un peu de texte: Hello World 1 Ici un titre de niveau 2 :</p> <p><b>Hellow World 2</b></p> <p>Ici un titre de niveau 2 avec du CSS :</p> <p><b>Hellow World 3</b></p> <p>Ici un titre de niveau 2 avec plus de CSS:</p> <p><b>Hellow World 4</b></p>	

- Avec la méthode `document.getElementById()`

Cours_JS_exmp07_getElementById_html_code.html	Résultat
<pre> ... &lt;p id="id1"&gt; paragraphe 1 &lt;/p&gt; &lt;script&gt;document.getElementById('id1').innerHTML =   "&lt;h2&gt; titre 1&lt;/h2&gt;"&lt;/script&gt; <b>Liste numérotée</b> &lt;ol id="id2"&gt;&lt;li&gt;item 1&lt;/li&gt;&lt;li&gt;item 2&lt;/li&gt;&lt;li&gt;item 3&lt;/li&gt;&lt;/ol&gt; &lt;script&gt;   document.getElementById('id2').innerHTML =     "&lt;li&gt;item a&lt;/li&gt;&lt;li&gt;item b&lt;/li&gt;" &lt;/script&gt; ... </pre>	<p><b>titre 1</b></p> <p>Liste numérotée</p> <p>1. item a 2. item b</p> <p>Le contenu de la balise <code>&lt;ol&gt;</code> a été modifié</p>

## 2 Programmation impérative en Javascript

Dans le contexte de ce cours, où l'on utilise le langage de programmation Javascript dans le cadre de l'écriture de pages HTML, c'est le navigateur qui exécute les programmes<sup>3</sup>.

Dans ce paragraphe 2, tous les programmes seront lancés depuis une page web, les sorties étant observées, *via* la console ou directement dans la page à l'aide d'un appel à `document.write()`.

Pour tester les instructions de base (déclaration de variables, types, opérations sur les types, ...), on pourra ouvrir une page vierge d'un navigateur, ouvrir la console interactive, et faire les tests dans la console. **On ajoutera systématiquement un « ; » après chaque instruction.**

Pour tous les exemples, on ouvrira la page **.html** de l'exemple dans un navigateur, ce qui exécutera le script écrit dans le fichier **.js** de l'exemple, et on ouvrira aussi la console du navigateur pour exécuter des lignes de codes et afficher la valeur des variables.

La console est aussi l'endroit où s'affichent les erreurs générées par l'exécution d'un programme.

La **console** est donc un **outil de débogage** des programmes.

### 2.1 Déclaration de variables – affectation

En Javascript, les variables peuvent être définies (« *déclarées* ») de trois façons à l'aide des instructions **var**, **const** et **let**, dans un premier temps, nous n'aborderons que la déclaration à l'aide du mot-clé **var**. On peut, à la différence de Python, déclarer un nom<sup>4</sup> de variable que l'on souhaite utiliser, sans immédiatement lui affecter de valeur :

#### Test dans la console interactive

```
>> var x;
< undefined
>> x;
< undefined
```

La valeur de la variable est alors « undefined », ce qui est à distinguer de l'erreur `ReferenceError`, qui signale qu'un nom n'est pas celui d'une variable déjà déclarée :

#### Test dans la console interactive

```
>> y;
❗ ▶ ReferenceError: y is not defined [En savoir plus] debugger eval code:1:1
```

Le fait de pouvoir déclarer une variable sans la définir pourra être utilisé pour déclarer, au début d'un programme, toutes les variables que l'on y utilisera, même si on ne leur affecte pas de valeur immédiatement.

La déclaration d'une variable avec affectation d'une valeur immédiatement, se fait avec la syntaxe suivante :

#### Test dans la console interactive

```
>> var a = 1;
< undefined
>> a;
< 1
```

Une fois que la variable a été déclarée (avec affectation de valeur ou non), on change sa valeur avec l'instruction d'affectation suivante :

#### Test dans la console interactive

```
>> a = 2;
< 2
```

On retiendra qu'**une variable doit toujours d'abord être déclarée**, avec le mot-clé **var**, et qu'ensuite on peut en changer la valeur avec l'opérateur « = ».

<sup>3</sup> Note : il est possible d'exécuter des programmes Javascript sans passer par un navigateur avec [node.js](https://nodejs.org/)

<sup>4</sup> Note : les noms de variables en Javascript doivent commencer par une lettre ou « \_ » ou « \$ ». Javascript est sensible à la casse (minuscules/majuscules) dans les « identificateurs » (noms de variables).

## 2.2 Types de base pour les valeurs et opérations élémentaires

On peut obtenir le type d'une variable avec l'opérateur `typeof` :

### Test dans la console interactive

```
>> typeof 0;
< "number"
```

### 2.2.a. Le type « number »

Il n'existe qu'un seul type de valeur pour les nombres en Javascript<sup>5</sup>, le type `number`, que l'on doit assimiler au type flottant de Python, ce type accepte la notation scientifique.

Par exemple, les valeurs

```
>> 0; 0.; -1.2; 1e2; -3.14e-2;
```

sont reconnues, avec le type `number`.

Les [opérateurs](#) sur ce type sont les opérateurs usuels associés aux mêmes symboles qu'en Python, seul l'opération de division entière « `//` » n'a pas de correspondant en Javascript, mais le reste dans une division entière existe et se note « `%` ».

On note deux opérateurs n'ayant pas d'équivalent en Python, les opérateurs d'incrément « `++` » et de décrément « `--` » qui, respectivement, ajoute ou retranche 1 à une variable, qui sont indiqués pour gérer des variables servant de « compteurs ».

### Test dans la console interactive

```
>> var a = 0;
< undefined
>> a++; a;
< 1
>> a++; a;
< 2
>> a--; a;
< 1
```

Les priorités, pour les opérateurs utilisés en mathématiques sont les mêmes qu'en mathématiques et sinon décrites [ici](#).

### 2.2.b. Le type « string »

Le type `string` est celui des chaînes de caractères. Pour définir une chaîne, on utilise des guillemets, simples « `'` » (apostrophe) ou doubles « `"` », que l'on utilise comme en Python, de même que certains caractères spéciaux, comme « `\n` ».

### Tests dans la console interactive

```
>> typeof "ok";
< "string"
>> typeof 'ko';
< "string"
>> "jeudi 5\noctobre";
< "jeudi 5
octobre"
>> var text = "l'aurore";
< undefined
>> var text = "l'aurore"; text;
< "l'aurore"
>> 'l\l'aurore';
< "l\l'aurore"
```

On notera que des conversions de type « à la volée » sont possibles, mais il est déconseillé de les solliciter :

```
>> "jour" + 2;
< "jour2"
```

Il existe sur les chaînes des [opérateurs](#), « `+` » et « `+=` » :

### Tests dans la console interactive

```
>> "abra" + "cadabra";
< "abracadabra"
>> var mot = "bara";
< undefined
>> mot += "ka"; mot;
< "baraka"
```

<sup>5</sup> Note : un type `BigInt` existe dans les implémentations les plus récentes du langage.

et propriétés et des « [méthodes](#) », dont nous ne citerons que `.length` (propriété), `.search()`, `.indexOf()` et `.slice()`, et `.substring()` (les quatre dernières sont des méthodes) :

Tests dans la console interactive	
<pre>&gt;&gt; "abracadabra".length; &lt; 11  &gt;&gt; "abracadabra".search('abra'); &lt; 0  &gt;&gt; "abracadabra".indexOf('abra', 0); &lt; 0  &gt;&gt; "abracadabra".indexOf('abra', 1); &lt; 7</pre>	<pre>&gt;&gt; "abracadabra".slice(-3, -1); &lt; "br"  &gt;&gt; "abracadabra".slice(-3, 0); &lt; ""  &gt;&gt; "abracadabra".slice(-3); &lt; "bra"  &gt;&gt; "abracadabra".slice(0, 4); &lt; "abra"  &gt;&gt; "abracadabra".slice(2, 5); &lt; "rac"</pre>

On notera que la numérotation des positions dans une chaîne commence à zéro.

### 2.2.c. Le type « boolean »

Le type boolean est celui des booléens. Ses valeurs sont `true` et `false`.

Tests dans la console interactive	
<pre>&gt;&gt; typeof true; &lt; "boolean"</pre>	<pre>&gt;&gt; typeof false; &lt; "boolean"</pre>

Comme c'est le cas dans tous les langages, les valeurs booléennes sont issues la plupart du temps d'opérations de comparaison et dans un deuxième temps de l'utilisation des opérateurs logiques, en voici le [tableau](#) (avec exemples [ici](#)) :

Opérateurs de comparaison		Opérateurs logiques	
Operator	Description	Operator	Description
<code>==</code>	equal to	<code>&amp;&amp;</code>	logical and
<code>===</code>	equal value and equal type	<code>  </code>	logical or
<code>!=</code>	not equal	<code>!</code>	logical not
<code>!==</code>	not equal value or not equal type	<b>Tests dans la console interactive</b>	
<code>&gt;</code>	greater than		
<code>&lt;</code>	less than		
<code>&gt;=</code>	greater than or equal to		
<code>&lt;=</code>	less than or equal to		
<code>?</code>	ternary operator		

On aura noté que Javascript autorise (ce qui est toujours dangereux) de comparer des valeurs de type différents, comme c'est illustré dans les exemples ci-dessus), pour s'assurer de comparer des valeurs de même type, on pourra utiliser les opérateurs « `===` » et « `!==` ».

L'opérateur ternaire « `?` » est décrit [ici](#), mais on nous ne l'utiliserons pas.

## 2.3 Types construits en Javascript

Les types décrits ici sont des « structures de données », permettant de structurer des ensembles de données.

### 2.3.a. Le type « Object »

Le type [Objet](#) est un type orienté-objet de Javascript. Il permet de définir des entités, nommées, possédant des *propriétés* propres et de *méthodes* (fonctions définies pour l'objet) propres.

Nous n'aborderons pas ici la définition de *méthodes* pour un objet, mais, nous utiliserons des méthodes sur des types d'objets prédéfinis en Javascript (par exemple les tableaux et les dates).



En voici un exemple d'un objet, individu, possédant trois propriétés, `nom`, `prenom` et `age`, l'on accède à la valeur des propriétés d'un objet, en utilisant des crochets, ou un point, comme montré ci-dessous.

### Tests dans la console interactive

```
>> var individu = {nom: 'Bernard', prénom: 'Claude', anneenaiss: 1991};
< undefined
>> individu;
< ▶ Object { nom: "Bernard", "prénom": "Claude", anneenaiss: 1991 }
```

```
>> individu['nom'];
< "Bernard"
>> individu.anneenaiss;
< 1991
```

Il est possible d'ajouter « dynamiquement » des propriétés à un objet :

```
>> var obj = {prop1: 1, prop2: "a"}; obj;
< ▶ Object { prop1: 1, prop2: "a" }
```

```
>> obj['prop3'] = true; obj;
< ▶ Object { prop1: 1, prop2: "a", prop3: true }
```

Il est possible de supprimer « dynamiquement » des propriétés à un objet :

```
>> var obj = {prop1: 1, prop2: "a"}; obj;
< ▶ Object { prop1: 1, prop2: "a" }
```

```
>> delete obj['prop2']; obj;
< ▶ Object { prop1: 1 }
```

De ce point de vue, dans lequel on n'a considéré que les propriétés d'un objet, le type `object` de Javascript est à rapprocher du type « dictionnaire » vu en Python, les propriétés correspondant aux clés. Ainsi, dans cette perspective, le type `object` de Javascript permet pallier l'absence en Javascript d'un type implémentant ce que l'on appelle en informatique des « [tableaux associatifs](#) »,

Il est possible d'itérer sur un objet, mais seulement sur les propriétés de l'objet (contrairement au type `dict` de Python), à l'aide d'un type particulier de boucle `for` :

### Itération sur les propriétés de l'objet

```
>> var obj = {prop1: "a", prop2: 1.2, prop3: 3, prop4: true};
< undefined
```

```
>> for (prop in obj) {console.log(prop, obj[prop]);}
< undefined
prop1 a
prop2 1.2
prop3 3
prop4 true
```

## 2.3.b. Le type « Date »

Le type `Date` est un type d'objet particulier qui pourra être utile ici, en particulier parce que l'utilisation de ce type d'objet permet la comparaison de dates et les opérations sur les dates.

On notera que la fonction prédéfinie `Date()` renvoie la date connue du système au moment où on exécute la fonction.

### Tests dans la console interactive

```
>> Date();
< "Sun May 10 2020 20:14:52 GMT+0200 (heure d'été d'Europe centrale)"
```

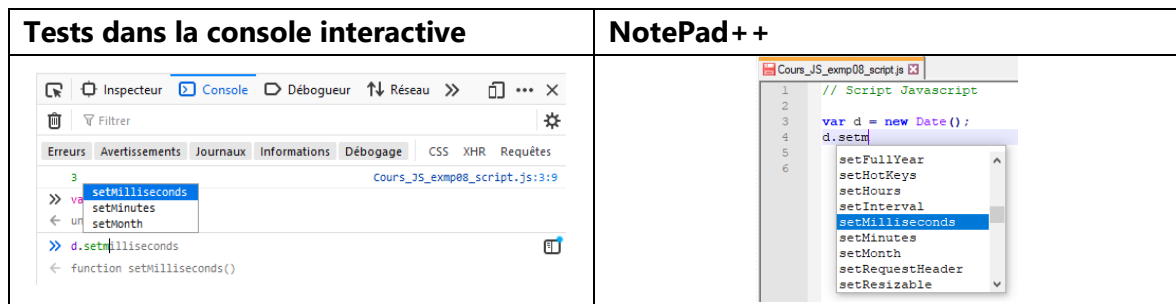
L'appel à la fonction `Date()` est un premier moyen de définir une date, on peut ensuite fixer l'heure, les minutes, *etc.* à l'aide de la fonction *ad hoc*, de la forme `.setHours()`, `.setMinutes()`, *etc.*

### Tests dans la console interactive

```
>> var d = new Date();
< undefined
>> d.setHours(13); d.setMinutes(20); d;
< ▶ Date Sun May 10 2020 13:20:04 GMT+0200 (heure d'été d'Europe centrale)
```

☛ On notera que, dans la console interactive, aussi bien que dans **NotePad++**, si on travaille sur un fichier d'extension **.js**, des propositions de fonctions existantes sont faites directement dans l'éditeur si l'on écrit les premières lettres de la fonction :





On dispose de trois autres façons de [définir une date](#) :

- En spécifiant l'année, puis le mois, le jour, ..., jusqu'au millisecondes si on le souhaite :

```
>> var d = new Date(2020, 4, 11);
< undefined
>> d;
< ▶ Date Mon May 11 2020 00:00:00 GMT+0200 (heure d'été d'Europe centrale)
```

Attention : les mois sont numérotés de 0 (janvier) à 11 (décembre) en Javascript.

- En spécifiant la date sous la forme d'une chaîne de caractères normalisée :

```
>> var d = new Date("2020-05-11"); d;
< ▶ Date Mon May 11 2020 02:00:00 GMT+0200 (heure d'été d'Europe centrale)
>> var d = new Date("2020-05-11T13:20"); d;
< ▶ Date Mon May 11 2020 13:20:00 GMT+0200 (heure d'été d'Europe centrale)
```

- En spécifiant un nombre de millisecondes, qui viennent s'ajouter ) à la date de référence en informatique, à savoir le 1<sup>er</sup> janvier 1970 :

```
>> var d = new Date(0); d;
< ▶ Date Thu Jan 01 1970 01:00:00 GMT+0100 (heure normale d'Europe centrale)
>> var d = new Date(24 * 60 * 60 * 1000); d;
< ▶ Date Fri Jan 02 1970 01:00:00 GMT+0100 (heure normale d'Europe centrale)
```

Note : la date du 1<sup>er</sup> Janvier 1970 est définie pour l'heure d'hiver, d'où le décalage d'une heure.

Enfin, pour afficher une date, ou une heure, dans un format plus léger, adapté à la langue par défaut du navigateur, on pourra utiliser les méthodes `.toLocaleDateString()`, `.toLocaleTimeString()`, `.toLocaleString()` :

Tests dans la console interactive	
<pre>&gt;&gt; var d = new Date(); d; &lt; ▶ Date Mon May 11 2020 07:18:44 GMT+0200 (heure d'été d'Europe centrale) &gt;&gt; d.toLocaleDateString(); &lt; "11/05/2020"</pre>	<pre>&gt;&gt; d.toLocaleTimeString(); &lt; "07:18:44" &gt;&gt; d.toLocaleString(); &lt; "11/05/2020 à 07:18:44"</pre>

On pourra extraire les valeurs de l'année, du mois, etc. avec les méthodes `.getFullYear()`, ..., `.getMilliseconds()` :

Tests dans la console interactive	
<pre>&gt;&gt; var d = new Date(); d; &lt; ▶ Date Mon May 11 2020 07:24:37 GMT+0200 (heure d'été d'Europe centrale) &gt;&gt; d.getFullYear(); &lt; 120</pre>	<pre>&gt;&gt; d.getMonth(); &lt; 4 &gt;&gt; d.getMinutes(); &lt; 24</pre>

Attention : on constate que l'année est ici comptée à partir de l'année 1900 et que les mois sont numérotés, comme déjà indiqué ci-dessus, de 0 à 11.

### 2.3.c. Le type « Array »

Ce type permet de définir des [tableaux](#) (*arrays*), que l'on pourra sommairement assimiler au type `list` Python.

On peut définir un tableau de la façon suivante :

```
>> var tab = [1, "a", true]; tab;
< ▶ Array(3) [ 1, "a", true ]
```

On constate qu'un tableau peut comporter des éléments de différents types, ce que l'on essaiera d'éviter cependant.

Un tableau est un élément de type `Objet`, dont l'une des propriétés est `length` (longueur), qui donne le nombre d'éléments du tableau, on accède aux éléments par leur indice de position dans le tableau (la première position est numérotée zéro). Il n'y a pas de numérotation négative, contrairement à Python.

#### Tests dans la console interactive

```
>> var tab = ["a", "b", "c"]; tab;
< ▶ Array(3) [ "a", "b", "c" ]

>> typeof tab;
< "object"

>> tab;
< (3) [-]
  0: "a"
  1: "b"
  2: "c"
  length: 3
  <prototype>: Array []
```

```
>> tab.length;
< 3

>> tab[0];
< "a"

>> tab[tab.length - 1];
< "c"

>> tab[-1];
< undefined
```

On peut modifier une valeur dans un tableau de la même façon qu'en Python :

```
>> var tab = ["a", "b", "c"]; tab;
< ▶ Array(3) [ "a", "b", "c" ]
```

```
>> tab[0] = "z"; tab;
< ▶ Array(3) [ "z", "b", "c" ]
```

On peut encore supprimer un élément avec l'opérateur `delete`, mais cette façon de faire est déconseillée car elle crée des « trous » dans le tableau :

```
>> var tab = ["a", "b", "c"]; tab;
< ▶ Array(3) [ "a", "b", "c" ]
```

```
>> delete tab[0]; tab;
< ▶ Array(3) [ <1 empty slot>, "b", "c" ]
```

Le tableau vide existe et se note `[]` ; il est aussi possible de définir un tableau de longueur donnée, sans pour autant affecter de valeurs à chaque cellule au moment de la création :

```
>> var tab0 = []; tab0;
< ▶ Array []

>> tab0.length;
< 0
```

```
>> var tab = Array(4); tab;
< ▶ Array(4) [ <4 empty slots> ]

>> tab.length;
< 4
```

- Méthodes pour les tableaux :

On agit sur les tableaux à l'aide de [méthodes spécifiques](#). Nous n'en citerons que quelques-unes.

- Suppression d'éléments : on peut modifier un tableau en retirant un élément, en dernière position (`.pop()`) ou en première position (`.shift()`) ; les deux méthodes renvoient l'élément retiré.

```
>> var tab = ["a", "b", "c"]; tab;
< ▶ Array(3) [ "a", "b", "c" ]

>> tab.pop(); tab;
< ▶ Array [ "a", "b" ]
```

```
>> var tab = ["a", "b", "c"]; tab;
< ▶ Array(3) [ "a", "b", "c" ]

>> tab.shift(); tab;
< ▶ Array [ "b", "c" ]
```

- Ajout d'éléments : on peut modifier un tableau en lui ajoutant un élément, en dernière position (`.push()`) ou en première position (`.unshift()`) ; les deux méthodes renvoient la nouvelle longueur du tableau.

```
>> var tab = []; tab;
< ▶ Array []

>> tab.push(1); tab.push(2); tab.push(3); tab;
< ▶ Array(3) [ 1, 2, 3 ]

>> tab.push(10);
< 4

>> tab;
< ▶ Array(4) [ 1, 2, 3, 10 ]
```

```
>> var tab = []; tab;
< ▶ Array []

>> tab.unshift(1); tab.unshift(2); tab.unshift(3); tab;
< ▶ Array(3) [ 3, 2, 1 ]

>> tab.unshift(10);
< 4

>> tab;
< ▶ Array(4) [ 10, 3, 2, 1 ]
```

- Concaténation de deux tableaux : on utilise la méthode `.concat()`, qui renvoie un nouveau tableau dont les valeurs sont des copies de valeurs dans les deux tableaux, une modification sur l'un des tableaux concaténés n'affecte pas le tableau créé.

<pre>&gt;&gt; var tab1 = [1, 2, 3]; &lt; undefined &gt;&gt; var tab2 = [3, 4, 5]; &lt; undefined &gt;&gt; var tab3 = tab1.concat(tab2); &lt; undefined &gt;&gt; tab3; &lt; Array(6) [ 1, 2, 3, 3, 4, 5 ]</pre>	<pre>&gt;&gt; tab1[0] = 99; &lt; 99 &gt;&gt; tab1; &lt; Array(3) [ 99, 2, 3 ] &gt;&gt; tab3; &lt; Array(6) [ 1, 2, 3, 3, 4, 5 ]</pre>
--	---

- Itérer sur un tableau :

La création de tableaux ne va pas sans possibilité de parcourir ce tableau, pour en traiter les valeurs. Comme en Python, il y a deux possibilités de parcours, par les indices et par les valeurs, implémentés par des formes spécifiques de boucles `for` (qui nécessitent de définir préalablement une variable pour effectuer le parcours).

Parcours par les indices	Parcours par les valeurs
<pre>&gt;&gt; var tab = ["a", "b", "c"]; &lt; undefined &gt;&gt; var i; &lt; undefined &gt;&gt; for (i in tab) {console.log(i, tab[i]);} &lt; undefined</pre> <pre>0 a          debugger eval code:1:25 1 b          debugger eval code:1:25 2 c          debugger eval code:1:25</pre>	<pre>&gt;&gt; var tab = ["a", "b", "c"]; &lt; undefined &gt;&gt; var v; &lt; undefined &gt;&gt; for (v of tab) {console.log(v);} &lt; undefined</pre> <pre>a          debugger eval code:1:25 b          debugger eval code:1:25 c          debugger eval code:1:25</pre>

Il existe des *méthodes* spécifiques pour [itérer sur un tableau](#) que nous ne détaillerons pas ici.

## 2.4 Fonctions

La syntaxe pour définir une fonction en Javascript est très proche de celle de Python, la différence majeure réside dans le fait que le corps de la fonction est délimité par des accolades, `{...}`, alors qu'en Python le corps de la fonction (la séquence d'instructions à exécuter lors d'un appel à la fonction) est délimité par, au début, deux points, « `:` », un passage à la ligne avec une indentation, et, à la fin, un passage à la ligne et une désindentation.

Voici, pour exemple, la définition d'une fonction prenant deux nombres en argument et renvoyant leur somme et leur produit :

Cours_JS_exmp08_fonctions1.html [extrait]	Cours_JS_exmp08_script.js
<pre>... &lt;ul&gt; &lt;script&gt; document.write("&lt;li&gt;valeur de a : ", a,"&lt;/li&gt;") document.write("&lt;li&gt;valeur de b : ", b,"&lt;/li&gt;") &lt;/script&gt; &lt;/ul&gt; ... &lt;script&gt; document.write(somme(a, b), " &lt;br/&gt;") &lt;/script&gt; ...</pre>	<pre>// Script Javascript Exemple 08 function somme(x, y) {     return x + y; } var a = 5; var b = 10;  On notera que :     Javascript ne permet pas l'affectation multiple     (var a, b = 5, 10 est interdit).</pre>

### Résultat d'exécution dans le navigateur

#### Fonction somme

Le script externe définit une fonction `somme`, qui prend en argument deux nombres et renvoie la somme de ces deux nombres. On testera la fonction dans la console du navigateur. Le script externe définit également deux variables `a` et `b`, dont on affiche les valeurs ci-dessous.

- valeur de `a` : 5
- valeur de `b` : 10

Voici enfin ci-dessous le résultat de l'appel `somme(a, b)` :

15

### Test dans la console interactive

```
>> somme(10, 12);
< 22
```

Il est possible qu'une fonction ne prenne pas de paramètres, on laisse alors les parenthèses vides dans sa définition.

Il est possible aussi qu'une fonction n'ait pas de valeur de retour, lorsqu'elle modifie l'environnement (en modifiant des valeurs en mémoire) ou utilise un « périphérique de sortie » (affichage dans la console, ou modification de la page web, par exemple), enfin, noter que la **valeur de retour** est toujours **unique**.

- Exemple de fonction sans valeur de retour, ni paramètre :

Cours_JS_exmp09_fonctions2.html [extrait]	Cours_JS_exmp09_script.js
<pre>... &lt;h2&gt;Voici l'heure actuelle :   &lt;span id="heure" style="color:red"&gt;&lt;/span&gt; &lt;/h2&gt; &lt;script&gt;   aff_heure() &lt;/script&gt; ...</pre>	<pre>// Script Javascript Exemple 09 function aff_heure() {   var heure =     (new Date()).toLocaleTimeString();    document.getElementById('heure').innerHTML     = heure; }</pre>
<b>Résultat d'exécution dans le navigateur</b>	
<b>Fonction sans paramètre et sans valeur de retour</b>	
Le script externe définit une fonction <code>aff_heure</code> qui écrit l'heure dans une balise <code>span</code> identifiée par l'id "heure".	
<b>Voici l'heure actuelle : 09:30:18</b>	

On rappelle que pour l'appel à la fonction ait un effet, la fonction doit être appelée, après la lecture de la définition de l'élément d'id « heure » qu'elle doit modifier.

- Utilisation du type `Object` pour retourner plusieurs valeurs :

Javascript n'autorise pas qu'une fonction renvoie plusieurs valeurs. On pourra contourner cette limitation en utilisant une valeur de retour de type objet, comme dans l'exemple ci-dessous, où l'on définit une fonction, `sp`, prenant en paramètres deux nombres, et qui renvoie un objet pour lequel on a défini deux propriétés, `somme` et `produit` afin de renvoyer la somme et le produit des deux nombres en entrée.

Cours_JS_exmp10_fonctions3.html [extrait]	Cours_JS_exmp10_script.js
<pre>... &lt;ul&gt; &lt;script&gt; document.write("&lt;li&gt;valeur de a : ", a,"&lt;/li&gt;") document.write("&lt;li&gt;valeur de b : ", b,"&lt;/li&gt;") &lt;/script&gt; &lt;/ul&gt; ... &lt;script&gt;   document.write(     "Résultat de l'appel sp(a, b) :&lt;br/&gt;",     sp(a, b).somme, " ", sp(a, b).produit,     "&lt;br/&gt;") &lt;/script&gt; ...</pre>	<pre>// Script Javascript Exemple 10 function sp(x, y) {   var s = x + y;   var p = x * y;   return {somme: s, produit: p}; }  var a = 5; var b = 10;</pre>
<b>Résultat d'exécution dans le navigateur</b>	
<b>Fonction somme et produit</b>	
Le script externe définit une fonction <code>sp</code> , qui prend en argument deux nombres et renvoie la somme et le produit de ces deux nombres, sous la forme d'un objet, possédant deux propriétés, 'somme' et 'produit'. On testera la fonction dans la console du navigateur. Le script externe définit également deux variables <code>a</code> et <code>b</code> , dont on affiche les valeurs ci-dessous.	
<ul style="list-style-type: none"> <li>valeur de <code>a</code> : 5</li> <li>valeur de <code>b</code> : 10</li> </ul>	
Voici enfin ci-dessous le résultat de l'appel <code>sp(a, b)</code> : Résultat de l'appel <code>sp(a, b)</code> : 15, 50	
<b>Test dans la console interactive</b>	
<pre>&gt;&gt; sp(12, 2); &lt; &gt; Object { somme: 14, produit: 24 }</pre>	

On a ici défini des variables au sein de la fonction, `s` et `p`, ces variables ont une **portée locale**, c'est-à-dire qu'elles ne seront pas reconnues dans le programme principal, ce qui n'est pas le cas des variables `a` et `b`, définies en dehors de la fonction.

Test dans la console interactive	
<pre>&gt;&gt; a; &lt; 5 &gt;&gt; b; &lt; 10</pre>	<pre>&gt;&gt; s; ❗ ReferenceError: s is not defined [En savoir plus] &gt;&gt; p; ❗ ReferenceError: p is not defined [En savoir plus]</pre>

- Utilisation du type `Array` pour retourner plusieurs valeurs :

On reprend ici le même exemple que précédemment, mais cette fois en renvoyant la somme et le produit *via* un tableau à deux éléments.

Cours_JS_exmp11_fonctions4.html [extrait]	Cours_JS_exmp11_script.js
<pre>... &lt;ul&gt; &lt;script&gt; document.write("&lt;li&gt;valeur de a : ", a,"&lt;/li&gt;") document.write("&lt;li&gt;valeur de b : ", b,"&lt;/li&gt;") &lt;/script&gt; &lt;/ul&gt; ... &lt;script&gt; document.write(   "Résultat de l'appel sp(a, b) :&lt;br/&gt;",   sp(a, b)[0],",", " ", sp(a, b)[1],   "&lt;br/&gt;") &lt;/script&gt; ...</pre>	<pre>// Script Javascript Exemple 11 function sp(x, y) {   var s = x + y;   var p = x * y;   return [x + y, x * y]; }  var a = 5; var b = 10;</pre>
Résultat d'exécution dans le navigateur	
< même résultat que pour l'exemple 10 >	
Test dans la console interactive	
<pre>&gt;&gt; sp(12, 2); &lt; ▶ Object { somme: 14, produit: 24 }</pre>	

## 2.5 Outils de contrôle de flux

Les exemples pour cette partie sont écrits dans des fichiers **.js**, que l'on pourra exécuter en double-cliquant sur les fichiers **.html**. Les tests pourront alors être effectués en ouvrant la console du navigateur dans la fenêtre qui se sera ouverte.

### 2.5.a. Branchements conditionnels

Les outils disponibles, comme pour la définition des fonctions, sont tout-à-fait semblables à ceux disponibles en Python, `if`, `else if` (au lieu de `elif`), `else`, avec des aménagements de la syntaxe :

- la condition est placée entre des parenthèses ;
- les blocs d'instructions à exécuter sont délimités, non plus, à l'ouverture, par « : » / passage à la ligne / indentation et, à la fermeture, un passage à la ligne / désindentation, mais par des accolades.

Cours_JS_exmp12_script.js	Test dans la console interactive
<pre>// Script Javascript Exemple 12 function affres(moyenne) {   if (moyenne &gt;= 10) {     return "Reçu"   }   else if (moyenne &gt;= 8) {     return "Second tour"   }   else {     return "Refusé"   } };</pre>	<pre>&gt;&gt; affres(14.5); &lt; "Reçu" &gt;&gt; affres(7.5); &lt; "Refusé" &gt;&gt; affres(8); &lt; "Second tour"</pre>

Comme en Python, le `if` peut être utilisé seul, ou apparié avec un `else`, ou associé avec un nombre quelconque de `else if`, avec ou sans `else`.

On dispose d'un outil supplémentaire, le branchement `switch`, permettant de définir un traitement distinct selon les valeurs prises par une expression, avec un traitement par défaut (facultatif) lorsque la valeur ne correspond à aucun des « cas » énumérés :

Cours_JS_exmp13_script.js	Test dans la console interactive
<pre>// Script Javascript Exemple 13 function joursemaine(date) {   switch (date.getDay()) {     case 0:       jour = "Lundi";       break;     case 1:       jour = "Mardi";       break;     case 2:       jour = "Mercredi";       break;     case 3:       jour = "Jeudi";       break;     case 4:       jour = "Vendredi";       break;     default:       jour = "Week-end";   }   return jour; };</pre>	<pre>&gt;&gt; var d = new Date(); d; &lt;&lt; Date Tue May 12 2020 09:54:18 GMT+0200 (heure d'été d'Europe centrale) &gt;&gt; d.getDay(); &lt;&lt; 2 &gt;&gt; joursemaine(d); &lt;&lt; "Mercredi" &gt;&gt; var d = new Date("2020-05-16"); d; &lt;&lt; Date Sat May 16 2020 02:00:00 GMT+0200 (heure d'été d'Europe centrale) &gt;&gt; d.getDay(); &lt;&lt; 6 &gt;&gt; joursemaine(d); &lt;&lt; "Week-end"</pre>

On notera que l'instruction `break` est nécessaire, sauf pour le dernier cas, sinon les blocs de code des cas suivants sont exécutés, même si la valeur ne correspond pas à celle indiquée dans le « case ».

## 2.5.b. Boucles bornées

La [syntaxe pour les boucles for](#) est la suivante :

<pre>for (instruction 1; instruction 2; instruction 3) {   // sequence d'instructions à exécuter (bloc de code) }</pre>
---

où :

- l'instruction 1 est exécutée une seule fois, au lancement de la boucle ;
- l'instruction 2 est un test dont la valeur est évaluée à la fin de chaque itération (on passe à l'itération suivante si sa valeur est `true`, la boucle termine sinon) ;
- l'instruction 3 est exécutée à chaque fois à la fin de chaque itération.

Dans l'utilisation usuelle, l'instruction 1 sert à définir le nom de l'indice de boucle, l'instruction 2 définit la condition d'arrêt, et l'instruction 3 permet d'incrémenter l'indice de boucle (on utilise pour cela l'opérateur `++` qui ajoutera 1 à l'indice de boucle à chaque fin d'itération).

Cours_JS_exmp14_script.js	Test dans la console interactive
<pre>// Script Javascript Exemple 14 function carres(n) {   for (i = 0; i &lt; n; i++) {     console.log(i ** 2)   } }</pre>	<pre>&gt;&gt; carres(5); &lt;&lt; undefined 0 Cours_JS_exmp14_script.js:4:11 1 Cours_JS_exmp14_script.js:4:11 4 Cours_JS_exmp14_script.js:4:11 9 Cours_JS_exmp14_script.js:4:11 16 Cours_JS_exmp14_script.js:4:11</pre>

Les boucles `for` sont adaptées pour parcourir un tableau :


Cours_JS_exmp15_script.js	Test dans la console interactive
<pre>// Script Javascript Exemple 15 var tab = ["Lundi", "Mardi", "Mercredi",   "Jeudi", "Vendredi", "Samedi",   "Dimanche"]; for (i = 0; i &lt; tab.length; i++) {   console.log("le jour n°", i,     " est le ", tab[i]) }</pre>	<pre>le jour n° 0 est le Lundi Cours_JS_exmp15_script.js:6:11 le jour n° 1 est le Mardi Cours_JS_exmp15_script.js:6:11 le jour n° 2 est le Mercredi Cours_JS_exmp15_script.js:6:11 le jour n° 3 est le Jeudi Cours_JS_exmp15_script.js:6:11 le jour n° 4 est le Vendredi Cours_JS_exmp15_script.js:6:11 le jour n° 5 est le Samedi Cours_JS_exmp15_script.js:6:11 le jour n° 6 est le Dimanche Cours_JS_exmp15_script.js:6:11</pre>

### 2.5.c. Boucles non bornées

La syntaxe des boucles `while` est tout-à-fait semblable à celle des boucles Python, avec les mêmes adaptations déjà vues pour délimiter la condition et le bloc de code à exécuter.

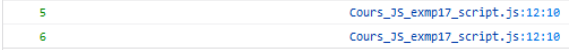
On donne ci-dessous un exemple simulant le lancer d'un dé équilibré à six faces, jusqu'à l'obtention d'un 6.

On y définit une fonction `lancer`, utilisant la fonction `Math.random()`, qui tire un nombre flottant pseudo-aléatoirement dans l'intervalle  $[0, 1[$ , associée la fonction `Math.floor()` qui renvoie la partie entière d'un nombre, pour simuler le lancer d'un dé équilibré à six faces.

Cours_JS_exmp16_script.js	Test dans la console interactive
<pre>// Script Javascript Exemple 16 function lancer() {     var x;     x = 6 * Math.random() // renvoie un     flottant dans [0, 6[     return 1 + Math.floor(x) // renvoie un     entier dans [1, 6] }  var sim = 0;  while (sim &lt; 6) {     sim = lancer();     console.log(sim); };</pre>	

Il existe une variante de boucle `while`, dans laquelle le bloc d'instruction à répéter est exécuté une fois inconditionnellement, et répété ensuite tant que la condition, évaluée cette fois après le bloc de code à répéter, est vraie.

Ce qui, sur le même exemple que précédemment, donne l'implémentation suivante (la définition de la fonction `lancer`, identique, n'a pas été recopiée ici).

Cours_JS_exmp17_script.js	Test dans la console interactive
<pre>// Script Javascript Exemple 17 ...  var sim;  do {     sim = lancer();     console.log(sim); } while (sim &lt; 6);</pre>	

## 3 Interaction entre le Javascript et le code HTML ou CSS

Nous avons vu précédemment comment il est possible de produire du code HTML, sous forme de chaîne de caractères, à l'aide de Javascript et de l'intégrer à une page web à l'aide de `document.write()` ou de `document.getElementById(id de l'élément).innerHTML`.

et renvoyant aux boucles ou (on parle alors, comme on l'a vu en Python, de fonction agissant par « effet de bord »).

Javascript ne permet pas non plus de renvoyer plusieurs valeurs.

En revanche, il existe

le type orienté-objet de Javascript. Il permet de définir des entités, nommées, possédant des *propriétés* propres et de *méthodes* (fonctions définies pour l'objet) propres.

En voici un exemple d'un objet, individu, possédant trois propriétés, `nom`, `prenom` et `age`, l'on accède à la valeur des propriétés d'un objet, en utilisant des crochets, ou un point, comme montré ci-dessous.

### Tests dans la console interactive



<pre>&gt;&gt; var individu = {nom: 'Bernard', prénom: 'Claude', anneeNaiss: 1991}; &lt;&lt; undefined &gt;&gt; individu; &lt;&lt; ▶ Object { nom: "Bernard", "prénom": "Claude", anneeNaiss: 1991 }</pre>	<pre>&gt;&gt; individu['nom']; &lt;&lt; "Bernard" &gt;&gt; individu.anneeNaiss; &lt;&lt; 1991</pre>
---	---

Ses valeurs sont true et false.

Pour définir une chaîne, on utilise des guillemets,

des opérateurs sur les chaînes de caractères

Il n'existe qu'un seul type de valeur pour les nombres en Javascript<sup>6</sup>, le type [number](#), que l'on doit

[https://www.w3schools.com/js/js\\_operators.asp](https://www.w3schools.com/js/js_operators.asp)

On dispose en Javascript des mêmes types de base qu'en Python.

On notera que le type d'une variable n'est pas déclaré et peut changer au gré des affectations de valeurs à la variable, cependant ce n'est pas une bonne pratique d'utiliser des variables dont le type varie au fil de l'exécution d'un programme.

Aussi, on fixera, sur le principe, le type d'une variable, au moment de la déclaration de la variable.

Que la variable ait été préalablement déclarée ou non, l'instruction pour affecter une valeur à une variable est la suivante :

Test dans la console interactive	
<pre>&gt;&gt; y;</pre>	<pre>! ▶ ReferenceError: y is not defined [En savoir plus]  debugger eval code:1:1</pre>

Ce Dans un premier temps,

Cette méthode permet d'afficher du texte ou, de façon plus générale, des valeurs.

On a déjà vu cette possibilité

ce qui sera prêt un programme Javascript se compose d'une suite d'instructions qu'il est possible d'inclure en différent e l'on va inclure dans la

<pre>&lt;!DOCTYPE html&gt; &lt;html lang= "fr" &gt; &lt;head&gt;   &lt;title&gt; Cours JS Exemple 01 &lt;/title&gt;   &lt;meta charset= "utf-8" /&gt; &lt;/head&gt; &lt;body&gt;   &lt;h3&gt;Premier script JavaScript&lt;/h3&gt;   Affichage dans une fenêtre externe.&lt;br/&gt;   &lt;script&gt;&lt;!--début d'une script javascript--&gt;     window.alert("Hello World");   &lt;/script&gt;&lt;!--début d'une script javascript--&gt;   &lt;br/&gt;   That's all folks. &lt;/body&gt; &lt;/html&gt;</pre>
--

`window.alert("Hello World");` permet d'afficher une fenêtre « pop-up ».

C'est première façon d'obtenir un affichage au travers d'une page web, avec javascript.

Utilité : attirer l'attention de l'utilisateur.

<sup>6</sup> Note : un type [BigInt](#) existe dans les implémentations les plus récentes du langage.

### 3.1 Exécution dans le navigateur et sorties

```
<!DOCTYPE html>
<html lang= "fr" >
<head>
  <title> Cours JS Exemple 02 </title>
  <meta charset= "utf-8" />
</head>
<body>
  <h3>Premier script JavaScript</h3>
  Observez le résultat du calcul "1 + 2" dans la console.<br/>
  Pour ouvrir la console attachée à cette page, faire<br/>
  <b><u>Ctrl+Alt+K</u></b> sur Firefox.
  <script>
    console.log(1 + 2);
  </script>
  <br/>
  Cliquer sur le menu "..." en haut à droite de la console,<br/>
  pour ancrer la console à droite de la fenêtre.
  <br/>
  Rafraîchir ensuite la page et observez.
</body>
</html>
```

```
console.log(1 + 2);
```

permet d'afficher du texte, le résultat d'un calcul, une valeur dans la console web du navigateur  
C'est deuxième façon d'obtenir un affichage au travers d'une page web, avec javascript.

Utilité : débogage d'un programme (analogue de print())

#### Exmp03

```
<!DOCTYPE html>
<html lang= "fr" >
<head>
  <title> Cours JS Exemple 01 </title>
  <meta charset= "utf-8" />
</head>
<body>
  <h3>Modification de la page affichée</h3>
  <p>
    On exécute un script qui rajoute du code HTML à la page affichée.
  </p>
  <p>
    Ici un peu de texte:<br/>
    <script>
      document.write("Hello World", "1 + 1 = ", 1 + 1);
    </script>
    <br/>
    Ici un titre de niveau 2 :<br/>
    <script>
      document.write("<h2>Hellow World</h2>");
    </script>
    Ici un titre de niveau 2 avec du CSS :<br/>
    <br/>
    <script>
      document.write("<h2 style=\"color:red\">Hellow World</h2>");
    </script>
    <br/>
    Ici un titre de niveau 2 avec plus de CSS et le résultat d'un calcul 1 +
    1:<br/>
    <br/>
    <script>
      document.write("<h2 style=\"color:red;\" +
```

```

"background-color:yellow\">" +
    "titre de niveau ", 1 + 1, " ajouté"
+
    "</h2>");
    </script>
    </p>
</body>
</html>

```

**document.write**(« ... »)

permet d'écrire dans la page html du code html.

C'est la troisième façon d'obtenir un affichage au travers d'une page web, avec javascript.

Utilité : débogage d'un programme.

#### Ex04

```

<!DOCTYPE html>
<html lang= "fr" >
<head>
    <title> Cours JS Exemple 01 </title>
    <meta charset= "utf-8" />
</head>
<body>
    <h3>Interaction avec la souris et affichage dans la page</h3>
    Cliquer sur le bouton pour afficher la date et l'heure.<br>
    <button type="button"
        onclick="document.getElementById('demo').innerHTML = Date()">
        Cliquer ici
    </button>
    <p id="demo"></p>

</body>
</html>

```

On utilise une balise HTML « button » et son attribut « onclick » qui permet de déclencher, l'exécution d'une fonction, lorsque l'on clique sur le bouton dans la page affichée

```

<button type="button"
    onclick="document.getElementById('demo').innerHTML = Date()">
    Cliquer ici
</button>

```

Si on décompose :

- Date() est une fonction javascript qui renvoie la date
- document.getElementById('demo') est une instruction qui permet de rechercher, dans le document (la page html sur laquelle on est), tous les éléments (il ne peut y en avoir qu'un seul car l'id « demo » ne peut être attribué à plusieurs éléments, car c'est un id)
- document.getElementById('demo').**innerHTML** .innerHTML désigne le code HTML qui est écrit à l'intérieur de la balise identifiée par « demo ».
- document.getElementById('demo').innerHTML = Date() indique de remplacer le contenu HTML de la balise par la valeur renvoyée par la fonction Date()

Sélectionner d'autres éléments

[https://www.w3schools.com/js/js\\_htmldom\\_elements.asp](https://www.w3schools.com/js/js_htmldom_elements.asp)

## 3.2 Petits programmes

Définition de variables

[https://www.w3schools.com/js/js\\_variables.asp](https://www.w3schools.com/js/js_variables.asp)

Manipulation de chaînes de caractères

[https://www.w3schools.com/js/js\\_strings.asp](https://www.w3schools.com/js/js_strings.asp)

Opérations arithmétiques

[https://www.w3schools.com/js/js\\_arithmetic.asp](https://www.w3schools.com/js/js_arithmetic.asp)

Conditions

[https://www.w3schools.com/js/js\\_if\\_else.asp](https://www.w3schools.com/js/js_if_else.asp)

Boucles for

[https://www.w3schools.com/js/js\\_loop\\_for.asp](https://www.w3schools.com/js/js_loop_for.asp)

document.

Document object model

## **4**