SUPPORT DE COURS B1-DEV

JavaScript

date	révision
Septembre 2020	V.123 Reprise de SI4 vers B1-Dev
Novembre 2021	V,126 Ajouts des exceptions (try/catch)
Juillet 2022	V.132 Ajouts écriture doc/console JS, ajout des opérations chaînes et ajout des dictionnaires
Février 2023	V.137 Ajout normalize()
Décembre 2023	V .145 Ajout exemple sur addEventListener + correction tableau événements

TABLE DES MATIÈRES

1 INTRODUCTION

Les origines de JavaScript remontent à mai 1995. Ce langage a été mis au point par Brendan EICH en 10 jours a été baptisé LiveScript dans sa version 1.0. La popularité du langage Java et un partenariat avec Sun Microsystem suggère de remplacer le mot 'live' par 'Java' pour servir l'intérêt des deux sociétés : JavaScript été né ! Il est intégré au navigateur Netscape Navigator 2.0.

Microsoft lui a alors emboîté le pas avec Jscript 1.0 dans Internet Explorer 3.0, mais dès la version 1.1 de Jscript et JavaScript, les deux langages sont devenus incompatibles : c'est le début d'une longue guerre commerciale et technique.

En 1996, Netscape rend public les définitions de son langage et le soumet à l'ECMA (une organisation suisse) pour standardisation et en 1997, le standard devient ECMAScript. Malgré cela, la version de JavaScript 1.2 et Jscript 2.0 deviennent encore moins compatibles. Les concepteurs de pages web doivent effectuer de nombreuses adaptations pour permettre un affichage similaire dans chaque navigateur. L'arrivée de Google puis de son navigateur aura finalement raison de cette guerre avec un moteur JavaScript plus compatible et rapide que les autres.

Aujourd'hui, l'historique d'ECMAScript montre les différentes évolutions, mais surtout, une adoption par la majorité des navigateurs en 2009 (il intègre de nombreuses bibliothèques) :

Si vous êtes observateur, vous devez même voir qu'il n'existe pas de version ES4 : la version était en opposition avec la nature du langage et un consortium a simplement créé sa propre implémentation baptisée ES3.1... au final, ES4 a été abandonné et ES3.1 est devenu ES5.

Ce langage permet d'accéder aux objets dans les applications web notamment. Il est utilisé pour vérifier les champs de saisies dans les formulaires, il permet de faire des animations ou des effets de transitions.

JavaScript s'utilise principalement dans un navigateur (qui intègre un moteur JavaScript).

1.1 FONCTIONNEMENT

JavaScript peut être placé n'importe où dans la page web, dans les balises `<script>` `</script>`. Comme pour CSS, le code JavaScript peut être intégré dans la page HTML ou bien dans un fichier séparé.

1.1.1 Chargement d'un script

HTML permet d'intégrer les scripts JavaScript à n'importe quel emplacement dans la page. Le navigateur lit alors les instructions du script et les exécutent immédiatement : c'est pour cela que la majorité des développeurs placent leurs scripts soit en début de page (à l'intérieur des balises `<head>` `</head>`) soit à la fin de la page (juste avant la balise `</body>`. C'est la taille et la finalité du script qui permettent de prendre la décision.

Dans le premier cas, le script n'aura pas connaissance des balises HTML dans le reste de la page, il sera donc plus difficile d'interagir avec les éléments de la page.

Dans le deuxième cas, il faudra attendre le chargement complet de la page pour charger et exécuter JavaScript.

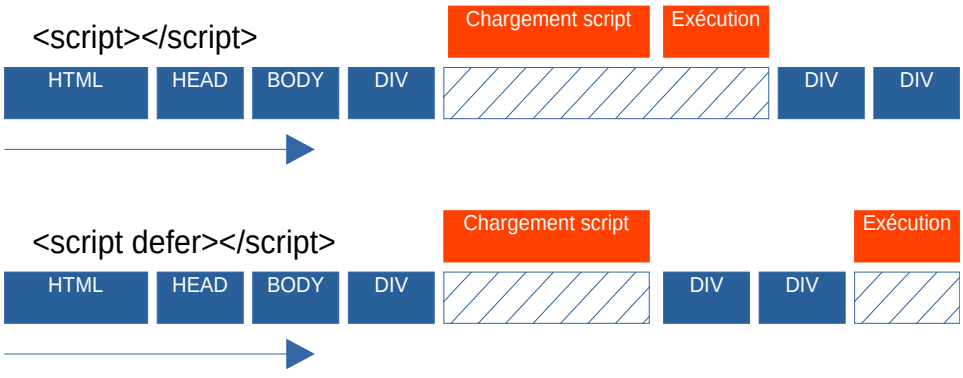
Il faut également préciser qu'il est possible de charger un script en début de page, mais de n'en exécuter les instructions qu'après le chargement complet de la page, grâce à la propriété `defer` de la balise `<script>`.

```
<script defer>
codes JavaScript
</script>
```

ou bien

```
<script src="fichierJavaScript.js" defer></script>
```

Voici un exemple d'exécution sans puis avec defer :



1.1.1.1 Script dans la page HTML

Il est possible de placer les scripts JavaScript dans la section <head> ou dans la section <body>

```
<html>
  <head>
    <title> Ma page web </title>
  </head>
  <body>
    <script language = "javascript">
      alert("Hello World 1.\nCliquez sur OK ")
    </script>
  </body>
</html>
```

Le script est chargé au moment ou il apparaît.
Il s'exécute immédiatement.

1.1.1.2 Script à l'extérieur de la page HTML (fichier séparé)

Il est aussi possible de placer le script dans un fichier à part, pour limiter le mélange des langages :

```
<html>
  <head>
    <title> Ma page web </title>
    <script type="text/javascript"
            src="QuoteOfTheDay.js">
    </script>
  </head>
  <body>

<p>Citation:<script>EcrireQuoteOfDay()</script></p>
  </body>
</html>
```

Le script doit être **chargé** : le navigateur recherche donc le fichier "QuoteOfTheDay.js"

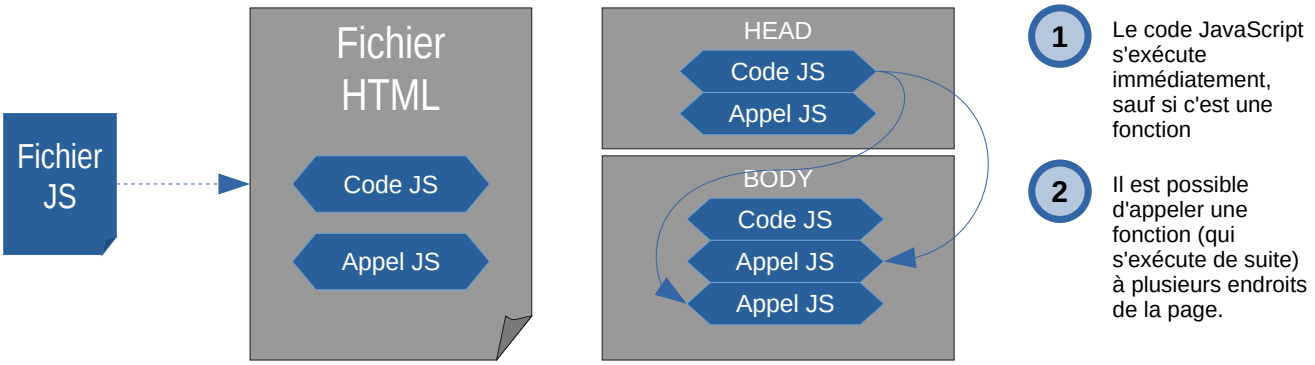
Le script est chargé au moment ou il apparaît, toutefois, si ce sont des fonctions, elles ne s'exécuteront que lorsqu'elles seront appelées.

La fonction *EcrireQuoteOfDay()* est **appelée**, elle s'**exécute**.

Dans tous les cas, le moteur JavaScript exécute le code à l'endroit de la page où il se trouve : si une balise n'existe pas encore, JavaScript ne pourra pas y accéder... c'est pourquoi on recommande de placer les scripts JavaScript juste avant la balise `<body>`, à la fin du document.

Enfin, il est aussi possible de charger le script JavaScript à un emplacement, alors qu'il ne contient aucune instruction exécutable immédiatement (uniquement des fonctions et des déclarations), puis à l'endroit voulu, appeler la fonction souhaitée (qui sera donc exécutée).

Voici un petit schéma pour résumer ces informations :



S'il y a plusieurs scripts dans la page HTML, ils pourront partager des informations (volontairement ou non) et interagir ensemble.

CyberSécurité : C'est pourquoi il est important d'utiliser des noms expressifs pour les variables, les noms de fichiers et les propriétés des balises.

1.1.2 Différence entre code et fonction

JavaScript est un langage interprété, ce qui signifie que lorsque le navigateur lit la page web et rencontre une section <script>, il n'essaye plus d'afficher les informations dans la page mais laisse le moteur JavaScript le faire à sa place. Lorsque le moteur JavaScript ne rencontre plus d'instructions et arrive à la balise de fin de script </script> il redonne la main au langage HTML.

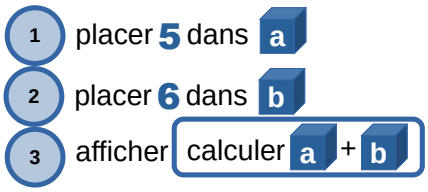
Il est donc possible d'exécuter du code au moment voulu dans une page web : lancer un calcul, demander une action à l'utilisateur, etc.

1.1.2.1 Code

Un code est une suite d'instructions simples qui s'exécutent les unes après les autres : l'ordre des lignes détermine l'ordre d'exécution.

Dans le code suivant, on place le nombre 5 dans une variable a (un espace de stockage une zone de mémoire), puis le nombre 6 dans la variable b et enfin, on demande à JavaScript d'afficher le résultat de l'addition des deux variables.

```
<script>
    a = 5
    b = 6
    document.write(a+b)
</script>
```



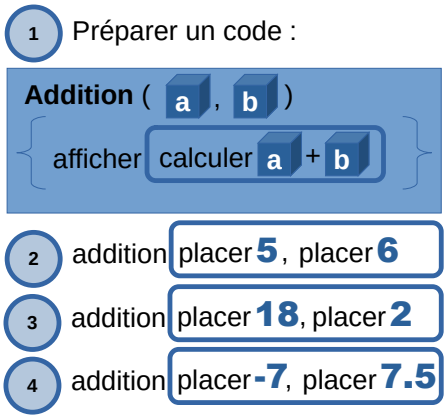
La fin d'une instruction lance l'instruction suivante.

1.1.2.2 Fonction

Une fonction est une suite d'instructions simples dans un bloc nommé. Les instructions ne peuvent se lancer que si le bloc est demandé par quelqu'un. De plus, une fonction peut recevoir des paramètres, qu'elle peut placer dans des variables choisies par le développeur de la fonction.

On peut considérer une fonction comme le nom d'une instruction complexe qui exécute plusieurs instructions simples.

```
<script>
    function addition(a, b) {
        document.write(a+b)
    }

    addition(5, 6)
    addition(18, 2)
    addition(-7, 7.5)
</script>
```



L'intérêt est surtout de pouvoir appeler plusieurs fois le même code, en utilisant le même nom.

1.1.3 Écriture avec JavaScript

1.1.3.1 document.write() et console.log()

JavaScript peut écrire des informations dans la page HTML ou bien dans la console du moteur JavaScript. Nous devons comparer les méthodes, car elles n'ont pas la même utilité :

- l'écriture dans une page HTML se fait avec `document.write(texte)` mais pose un problème, car il peut y avoir des erreurs d'écriture dans les emplacements : cette méthode n'est utilisée que dans les codes d'exemples.
- L'écriture dans la console JavaScript se fait avec `console.log(texte)` mais cette fois, l'utilisateur normal ne voit rien (il faut ouvrir la console de débogage).

1.1.3.2 position.innerHTML()

Plus tard, nous découvrirons deux autres méthodes fiables pour écrire à posteriori (c'est-à-dire, après l'affichage complet de la page HTML). Cela permet de modifier des zones de la page sans tout écraser. Les deux commandes s'appuient sur la recherche d'une balise HTML particulière, que voici :

```
position = document.getElementById(valeur)
```

Une fois la position déterminée, on peut écrire du texte ASCII avec `.innerText` ou du texte HTML (contenant des balises) avec `.innerHTML`

```
position.innerText("<u>ce texte n'est pas souligné et affiche les balises.</u>")
position.innerHTML("<u>ce texte sera affiché en souligné</u>")
```

1.1.3.3 position.insertAdjacentHTML()

Une autre méthode plus efficace encore consiste à créer directement les nœuds du DOM (trop tôt pour expliquer ici) et les insérer dans la page.

```
position.insertAdjacentHTML(emplacement, "Valeur <u>emplacement</u> définit où…")
```

avec `emplacement` pouvant prendre les valeurs suivantes :

`"beforebegin"`	Affiche le texte avant la position (donc avant sa balise)
`"afterbegin"`	Affiche le texte à l'intérieur de la position (au début)
`"beforeend"`	Affiche le texte à l'intérieur de la position (à la fin)
`"afterend"`	Affiche le texte après la position (donc après sa balise)

1.2 EXEMPLE DE CODE

L'exemple est un script qui choisit aléatoirement une citation et l'affiche dans la page.

Sans rentrer dans les détails pour le moment, la fonction QuoteOfTheDay() utilise une table de 5 phrases et en choisit une au hasard et l'affiche dans la page web.

1.2.1 Citation du jour

Le fichier JS contient uniquement le code suivant :

QuoteOfTheDay.js

```
// Créer un tableau et y stocker les phrases, puis en choisir une au hasard.
function QuoteOfTheDay() {
	quotes = []
	quotes[0] = "C'est le rôle essentiel du professeur d'éveiller la joie de travailler et de connaître. (A. Einstein)"
	quotes[1] = "Exige beaucoup de toi-même et attends peu des autres. Ainsi beaucoup d'ennuis te seront épargnés. (Confucius)"
	quotes[2] = "Chuck Norris peut te faire passer un sale quart d'heure en 8 minutes. (Chuck)"
	quotes[3] = "Il est hélas devenu évident aujourd'hui que notre technologie a dépassé notre humanité. (A. Einstein)"
	quotes[4] = "Ils ne savaient pas que c'était impossible, alors ils l'ont fait. (M. Twain)"
	quotes[5] = "Je crois beaucoup en la chance ; et je constate que plus je travaille, plus la chance me sourit. T. Jefferson"

	images = []
	images[0] = "http://david.roumanet.free.fr/images/einstein.jpg"
	images[1] = "http://david.roumanet.free.fr/images/confucius.jpg"
	images[2] = "http://david.roumanet.free.fr/images/chucknorris.jpg"
	images[3] = "http://david.roumanet.free.fr/images/einstein.jpg"
	images[4] = "http://david.roumanet.free.fr/images/twain.jpg"
	images[5] = "http://david.roumanet.free.fr/images/jefferson.jpg"
	// Choisir un nombre aléatoire inférieur à la taille du tableau
	index = Math.floor(Math.random() * quotes.length)
	document.write("<p>\"" + quotes[index] + "\"</p>\n")
	document.write("<p><img src=\"" + images[index] + "\"></p>\n")
}
```

Le \ est un caractère spécial d'échappement. Voir plus bas.

Ici, `\"` permet d'indiquer que le `"` derrière le symbole `\` ne ferme pas la chaîne, mais sera utilisé dans l'affichage (JS pourra afficher ce symbole dans la page HTML).

De même, `\n` permet de créer un retour à la ligne dans le code source affiché dans le débogueur, mais `\n` n'a aucune influence sur l'affichage en HTML (un retour à la ligne se fait avec `<br>` en HTML).

Le fichier HTML contient le code suivant :

QuoteOfTheDay.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Citation</title>
    <script type="text/javascript" src="QuoteOfTheDay.js">
    </script>
  </head>
  <body>
    <h1>Citation du jour</h1>
    <script>QuoteOfTheDay()</script>
  </body>
</html>
```

Chargement du fichier en mémoire

Exécution de la fonction du script

Le caractère d'échappement indique à JavaScript que le caractère qui le suit doit être affiché tel quel, ou bien que le caractère sera une commande :

\"	Affiche "
\'	Affiche '
\\	Affiche \

\n	Passe à la ligne
\t	Affiche une tabulation

Ainsi, dans l'exemple ci-dessus, le code HTML pour afficher une image serait :

```
<p><img src="monImage.jpg"></p>
```

Malheureusement, nous souhaitons écrire une variable JavaScript qui est délimité par des doubles quotes. Le code JavaScript suivant :

```
document.write("<p><img src=" + images[index] + "></p>")
```

Affichera ce code (ou il n'y a plus de double quote) dans le fichier HTML :

```
<p><img src=monImage.jpg></p>
```

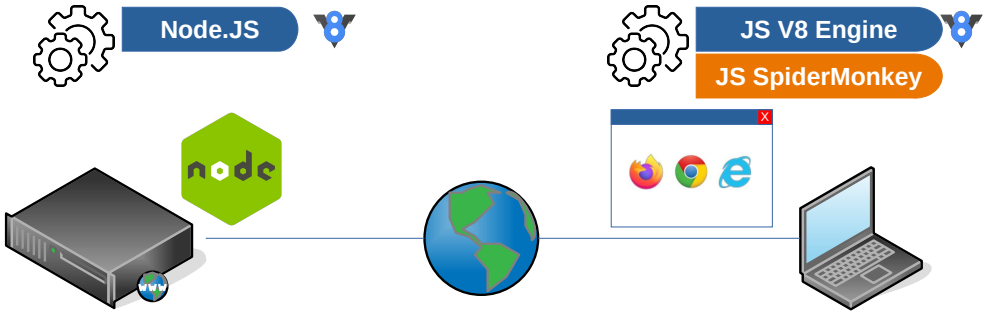
Bien que la plupart des navigateurs soient capables de gérer ce nom de fichier, ce code ne fonctionnera plus si le nom du fichier transmis contient un espace.

fonctionne	Ne fonctionne pas
<img src="monImage.jpg"> <img src="mon Image.jpg"> <img src=monImage.jpg>	<img src=mon Image.jpg>

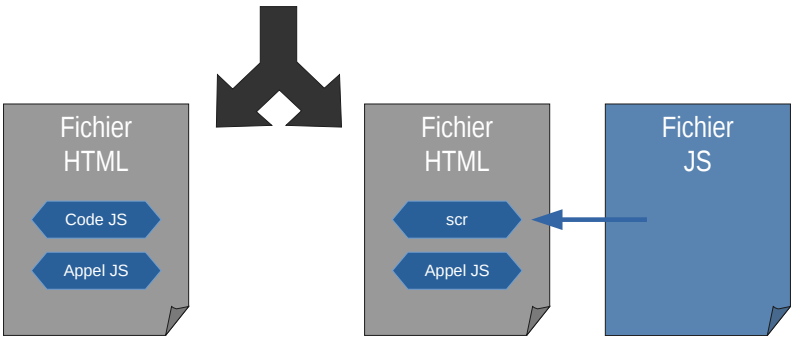
Voilà pourquoi il faut utiliser \" qui indique à JavaScript que le caractère après \ doit être affiché tel quel.

1.3 USAGE JAVASCRIPT, EN RÉSUMÉ

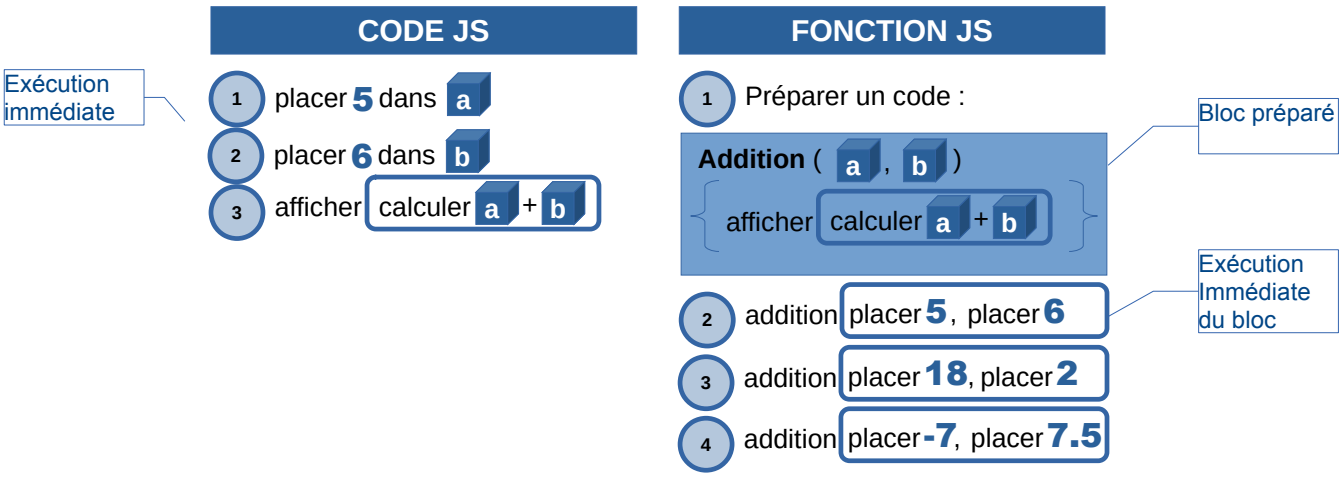
1.3.1 Où utilise-t-on principalement JavaScript ?



1.3.2 intégrer du code JavaScript dans une page HTML ?



1.3.3 différence entre un code et une fonction JavaScript ?



2 JavaScript

Un langage de programmation dispose d'une syntaxe, de mot-clés réservés, et d'une logique d'utilisation.

En tant que développeur, vous devez apprendre ce langage, comme vous apprendriez une nouvelle langue.

2.1 Les commentaires

En JavaScript, on peut utiliser les deux commentaires habituels, à savoir :

- Sur une seule ligne on utilise le double slash `//`
- Pour créer un bloc de commentaire sur plusieurs lignes on commence avec `/*` et on termine avec `*/`

2.2 Intégration en HTML

JavaScript s'exécute sur les navigateurs web : il faut donc pouvoir écrire du code HTML avec JavaScript.

Dans le corps d'une page HTML, on utilise des balises : `<strong>` `</strong>`, `<h1>` `</h1>`, `<p>` `</p>`... et JavaScript peut écrire dans une page web avec l'instruction document.write().

Il devient possible de générer un code HTML en JavaScript :

```
<script language="JavaScript">
  document.write("<h1>Hello World</h1>")
  document.write("<h2>exemple en javascript</h2>")
  document.write("<p>Le texte peut dépasser une ligne ")
  document.write("car il n'y a pas de retour chariot ")
  document.write("mais il faut mettre les balises ")
  document.write("HTML : la page sera correcte.</p>")
</script>
```

Écrit un code HTML à l'emplacement de l'appel du script.

C'est une méthode un peu longue et l'instruction `document.write()` n'est que très rarement employée mais cela permet de comprendre le principe d'une page web !

2.2.1 Délimitation de chaîne

Note : pour ne pas mélanger les instructions du langage et les textes à écrire, on place ces derniers entre des guillemets doubles (double quote) ou des guillemets simples (simple quote).

Ces symboles : 

Il existe aussi une nouvelle notation utilisant les ` (accent grave ou en anglais : backtick) comme début et fin de chaîne, pour intégrer les variables automatiquement à l'intérieur d'une chaîne :

```
let unTexte = "youpi"
let texte = `écrire le contenu de la variable unTexte à partir d'ici : ${unTexte} ! `
```

Ainsi, le code précédent devient :

Il devient possible de générer un code HTML en JavaScript :

```
<script language="JavaScript">
  document.write(`<h1>Hello World</h1>
  <h2>exemple en javascript</h2>
  <p>Le texte peut dépasser une ligne
  car il n'y a pas de retour chariot
  mais il faut mettre les balises
  HTML : la page sera correcte.</p>`)
</script>
```

Pour vous entraîner :

https://tech.io/playgrounds/112198/module-288-utilisation-des-techniques-de-programmation-dans-le-frontend-web/les-variables-et-les-constantes

2.3 LES VARIABLES

2.3.1 Les variables

Les variables ne sont pas **typées** en JavaScript, l'interpréteur reconnaît le type ou l'adapte en fonction du contexte. JavaScript réserve alors la mémoire pour les nombres ou les chaînes de caractères, au moment où il rencontre la déclaration.

2.3.1.1 Noms de variables

Une variable doit avoir un nom (c'est comme une étiquette sur une boite). Il s'agit d'une suite de caractères alphanumériques (a-z, A-Z, 0-9 et quelques symboles comme `-` ou `_`). Généralement, il est interdit de commencer un nom de variable par un chiffre et une variable ne contient jamais d'espace.

2.3.1.2 Types de variables

Les ordinateurs reconnaissent à minima les nombres et les chaînes de caractères. JavaScript reconnaît automatiquement les chaînes, grâce à la présence de guillemets (simples ou doubles) au début et à la fin de la chaîne. Il existe cependant d'autres types utilisés en JavaScript :

Number	String	Array	Object	Function	Undefined/Null
42	"Hello World!"	[1, 2, 3, "ok", 5]	{pr :"Bob", a:43}	function() = {}	let a let b = null

2.3.1.3 Les nombres en JavaScript

Pour JavaScript, tous les nombres sont stockés dans un type 'double', ce qui permet de ne pas rencontrer de problème lors d'opérations arithmétiques.

Déclarer une variable qui contient un nombre se fait très simplement : il suffit d'écrire le nom de la variable, l'opérateur égal et le nombre (en utilisant le point comme séparateur décimal).

```
<script language="JavaScript">
      let rayon = 5
      const pi = 3.14159265
      let Permimetre = rayon * 2 * pi
</script>
```

Déclaration de variables numériques

2.3.1.4 Les textes (chaînes) en JavaScript

Un texte est toujours délimité, par des doubles cotes (symbole `"`) ou simples cotes (symbole `'`), ainsi JavaScript ne peut pas les interpréter comme des noms de variables. On parle également de chaîne de caractères. Récemment, on peut délimiter un texte avec le symbole `` ` `` (beaucoup plus pratique).

```
<script language="JavaScript">
      let nom = "ROUMANET"
      let prenom = 'David'
      let citation = 'David dit "cool" !'
</script>
```

Déclaration de variables textuelles

2.3.1.5 Les opérations

JavaScript permet ensuite d'utiliser des opérations pour modifier les variables. Presque tous les opérateurs mathématiques sont présents mais si les calculs entre variables numériques donnent un résultat mathématique, la présence de chaînes de caractères peut tout compliquer.

2.3.1.5.1 Opérations sur les nombres

```
<script language="JavaScript">
      let a = 5
      let b = 1.25
      let c = a + b * 2
      document.write(c)
      console.log(c)
</script>
```

L'ordre des priorités des opérations est respecté : C = 5 + 1.25 * 2

Ici, c sera égal à `7.5` car JavaScript effectue d'abord la multiplication, puis l'addition.

En remplaçant b par une chaîne contenant uniquement des caractères numériques,

```
<script language="JavaScript">
      let a = 5
      let b = "2.25"
      let c = a + b
      document.write(c)
      console.log(c)
</script>
```

JavaScript regarde les opérandes et décide que dans ce contexte, + signifie concaténer (car b est une chaîne).

Écrit dans la page du navigateur

Écrit dans la console du débogueur [F12]

Le résultat affiché sera `52.25` car JavaScript décide de convertir le nombre a en chaîne de caractère : en effet, l'opérateur + est ici interprétée comme une concaténation de chaîne.

Par contre, en changeant l'opération (remplacer l'addition par une multiplication) :

```
<script language="JavaScript">
      let a = 5
      let b = "2.25"
      let c = a * b
      document.write(c)
</script>
```

JavaScript sait que * n'a qu'une signification : il essaye de convertir la chaîne en nombre.

Le résultat sera cette fois `11.25` car l'opérateur * est considéré comme une multiplication : JavaScript va convertir automatiquement le contenu de la chaîne. Le script ci-dessous renvoie une erreur :

```
<script language="JavaScript">
      let a = 5
      let b = "Jojo"
      let c = a * b
      document.write(c)
</script>
```

JavaScript sait que * n'a qu'une signification : il n'arrive pas à convertir la chaîne en nombre.

C n'est pas un nombre, JavaScript affichera NaN (Not a Number).

2.3.2 Conversion de variables

Même si la plupart du temps, JavaScript effectue les conversions pour vous, il est intéressant de choisir comment effectuer certaines conversions.

- Pour convertir une chaîne en nombre, il suffit d'utiliser les fonctions `parseInt()` et `parseFloat()`.

- Pour transformer un nombre en chaîne de caractères, c'est la fonction `toString()` ;

```
let nombreTexte = "1.24"
let nombre = parseFloat(nombreTexte) + 5
let Texte = "La température est de " + nombre.toString() + "°C"
```

2.3.3 Chaînes et caractères

JavaScript permet de traiter les textes comme des chaînes de caractères : il faut comprendre qu'une chaîne de caractères est un tableau de caractères. Cependant, la première lettre commence à zéro.

0	1	2	3	4	5	6
B	o	n	j	o	u	r

```
let salut = "Bonjour"
let longueur = salut.length   // longueur sera égale à 7
let Lettre3 = salut[2]        // Lettre3 contient 'n'
```

On peut aussi convertir toutes les lettres en majuscules ou minuscules, grâce aux fonctions **toUpperCase()** ou **toLowerCase()** :

```
let salut = "Bonjour"
let salutMaj = salut.toUpperCase()  // salutMaj contient "BONJOUR"
let salutMin = salut.toUpperCase()  // salutMin contient "bonjour"
```

Enfin, il est possible de convertir des textes en mots ou groupes dans des tableaux, si on connaît le

séparateur. Par exemple :

```
let salut = "Bonjour les amis"
let tableau = salut.split(' ')      // utilise l'espace pour trouver les mots
console.log(tableau[2])             // affichera 'amis'
```

2.3.3.1.1 opérations simples sur les chaînes

```
<script language="JavaScript">
      nom = "ROUMANET"
      prenom = 'David'
      full = nom + " " + prenom
      document.write(c)
</script>
```

L'opérateur + a une double fonction :
Pour les nombres, c'est une addition
Pour les chaînes, c'est une concaténation

Ici, la variable `full` sera égale à "`ROUMANET David`" car JavaScript effectue la concaténation de "`ROUMANET`" avec "` `" et "`David`".

2.3.3.1.2 Opérations complexes sur les chaînes

JavaScript permet d'effectuer des opérations de traitements de chaînes très intéressantes. Le tableau suivant énumère les opérations principales.

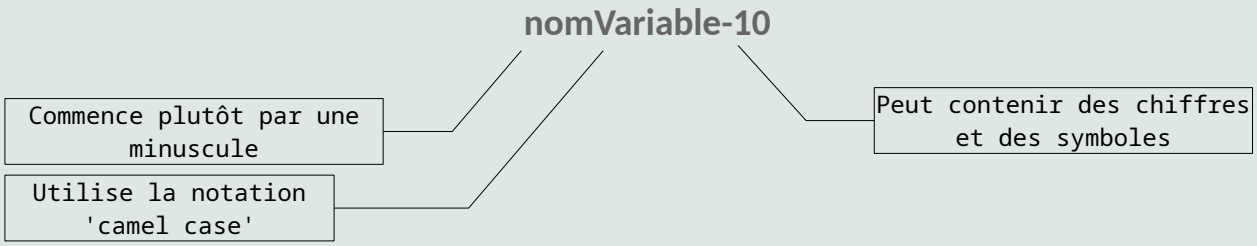
Si la variable `txt = "Étudiants"`, voici les opérations applicables :

`txt.length`	Renvoie la longueur de la chaîne (nombre caract.)	`9`
`txt.slice(3,6)`	Renvoie le morceau de la chaîne entre début :3 et fin :6	`dia`
`txt.substr(5,4)`	Renvoie la sous-chaîne début:5, longueur:4	`ants`
`txt[2]` ou `txt.charAt(2)`	Renvoie le caractère à la position 2 (*commence à 0*)	`u`
`txt.replace("t","7")`	Remplace le premier caractère `t` trouvé, par `7`	`É7udiants`
`txt.replace(/t/g,"7")`	Remplace tous les caractères `t` par `7` (expression reg)	`É7udian7s`
`txt.toUpperCase()` `txt.toLowerCase()`	Modifie la casse du texte en MAJUSCULES ou minuscules	`ÉTUDIANTS` `étudiants`
`txt.indexOf("t", 0)`	Renvoie la position du premier `t` rencontré à partir de 0	`1`
`txt.search("/DI/i")`	Renvoie la position de l'expression régulière (regex)	`3`
`txt.split("u")`	Créée un tableau en utilisant le séparateur u	`["Ét","diants"]`

2.4 Les variables, en résumé

2.4.1 Nommage

Les variables peuvent prendre presque n'importe quel nom si elle ne commence pas par un chiffre.



Par convention internationale, on utilise des noms de variables en anglais, sans accents ou symboles particuliers qui n'existe pas ailleurs.

2.4.2 Contenu

Les variables JavaScript peuvent contenir des objets, mais pour le moment, nous nous intéressons qu'à deux sortes de contenu :

Les textes (chaînes)	Les nombres
Variable = "bonjour"	Variable = -10.5

Le format des nombres utilise le point comme séparateur décimal.

Le format de texte utilise des quotes simples ' ou doubles " pour encadrer le texte

2.4.3 opérations sur les variables

Il est possible de faire toutes les opérations mathématiques sur les variables numériques.

L'addition est la seule opération qui peut s'appliquer aux variables textes : il s'agit d'une concaténation.

Les textes	Les nombres

`VariableA = "bonjour "`
`VariableB = VariableA + "Chuck Norris"`
bonjour Chuck Norris
	`VariableA = -10.5`
	`VariableB = VariableA * 15`
	-157.5

Dans le cas d'une opération mathématique sur une variable texte, JavaScript essaye de la convertir en nombre.

Dans le cas d'une addition entre une variable texte et une variable numérique, JavaScript convertit la variable numérique en chaîne de caractères et concatène l'ensemble.

2.5 LES TABLEAUX

2.5.1 Tableaux simples

JavaScript sait gérer des tableaux de valeurs, ce qui est extrêmement utilisé en informatique.

2.5.1.1 Création d'un tableau

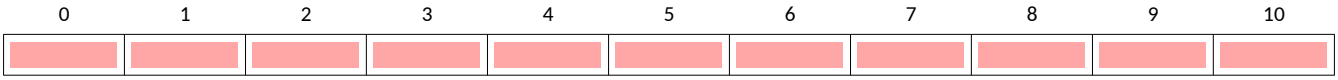
Il est possible de définir un tableau de dimension connu, en spécifiant le nombre de cellule qu'il contient :

```
listeEtudiants = [11] // déclaration d'un tableau de 10 valeurs
```

ou bien sans rien spécifier

```
listeEtudiants = [] // déclaration d'un tableau
```

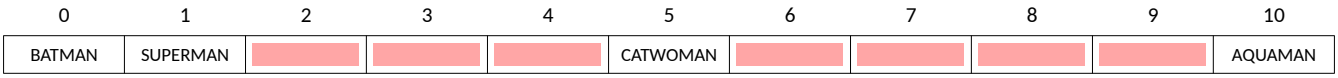
Il faut toutefois être vigilant dans le parcours du tableau : la première cellule commence à l'index zéro !



2.5.1.2 Écrire ou modifier des cellules

Ainsi, placer une valeur dans les premières cellules s'écrira :

```
listeEtudiants[0] = "BATMAN"
listeEtudiants[1] = "SUPERMAN"
listeEtudiants[5] = "CATWOMAN"
```



Dans certains langages, l'instruction suivante serait un plantage pur et simple :

```
listeEtudiants[12] = "PUSHMAN"
```

Mais JavaScript utilise des tableaux dynamiques, il est possible d'augmenter la taille sans s'inquiéter de la taille initiale. JavaScript a une vision assez laxiste, que l'on ne trouve pas en C# ou Java.

Il existe un moyen d'ajouter à la fin du tableau, en utilisant **push()**

```
let fruits = ["Banana", "Orange", "Apple", "Mango"]
fruits.push("Kiwi")
```

2.5.1.3 Lire / parcourir un tableau

Lire un tableau peut se faire en interrogeant directement le numéro de la cellule, ou bien en utilisant une boucle. L'exemple ci-dessous présente les deux boucles **for** utilisables :

2.5.1.3.1 Boucle for classique

```
// initialisation du tableau
let marques = ['BMW', 'Peugeot', 'Tesla', 'Volvo', 'Mini']

// boucle "for" classique
for (let index=0 ; index < marques.length ; index++) {
    document.write(marques[index] + "<br >");
}
```

2.5.1.3.2 Boucle for… of

```

// boucle "for of"
let uneMarque
for (uneMarque of marques) {
    document.write(uneMarque + "<br >");
}
```

Dans ce dernier cas, nous n'avons pas accès à l'indice de l'élément, mais seulement sa valeur.

2.5.1.4 Supprimer une cellule

La gestion des tableaux est tellement fréquente, qu'il existe d'autres instructions pour faciliter cette gestion.

Il est ainsi possible de supprimer la **dernière** cellule du tableau avec pop().

Pour supprimer une cellule précise à l'intérieur d'un tableau, la commande splice(position, nombre).

```
// initialisation du tableau
let marques = ['BMW', 'Peugeot', 'Tesla', 'Volvo', 'Volkswagen']
marques.pop()           // supprime Volkswagen
marques.splice(0,1)           // supprime BMW et Peugeot
for (let index=0 ; index < marques.length ; index++) {
    document.write("[" + marques[index] + "] ");
}
// affichera [Peugeot] [Tesla] [Volvo]
```

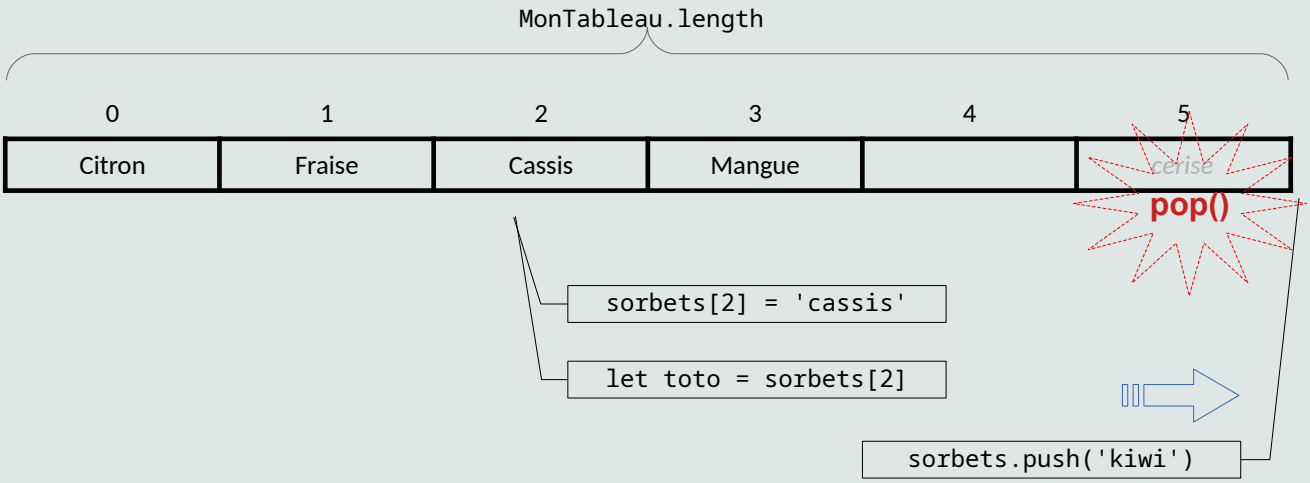
2.6 TABLEAUX SIMPLES, EN RÉSUMÉ

Un tableau contient plusieurs cellules, numérotées à partir de zéro.

2.6.1 Les opérations sur les tableaux

Les méthodes propres aux tableaux sont :

- sorbets = [] : créé un tableau
- sorbets[x] = valeur : écrit 'valeur' dans la cellule numéro 'x' du tableau
- sorbets.**length** : renvoie la taille du tableau (nombre de cellules total)
- sorbets.**push(valeur)** : ajoute 'valeur' à la fin
- sorbets.**pop()** : supprime la dernière cellule
- sorbets.**splice(position, nombre)** : supprime 'nombre' cellules à partir de 'position'
- sorbets.**splice(position, 0, valeur)** : insère 'valeur' dans une nouvelle cellule après 'position'



2.6.2 Parcourir un tableau

Lire un tableau se fait par toutes les boucles (do... while, for, etc.) mais la boucle "for ... of" est la plus pratique (si on n'a pas besoin de l'index mais seulement de la valeur) :

for (variableReceveuse of nomTableau) { ... }

2.6.3 Tableaux multidimensionnels

JavaScript permet de traiter des tableaux ayant plus d'une dimension, cependant, les tableaux multidimensionnels sont un peu plus difficiles à appréhender. Prenons un tableau à deux dimensions, contenant la marque d'une voiture et le nombre de voitures vendues dans le mois : la déclaration du tableau reste identique et ne prend pas en compte les dimensions.

En revanche,

```
vente = []
vente[0] = ["Peugeot",80]
vente[1] = ["Fiat",110]
vente[2] = ["Renault",105]
vente[3] = ["Citroen",180]
```

La cellule numéro 0 contient maintenant un tableau de deux éléments : Peugeot et 80.

On peut aussi déclarer le même tableau sur une seule ligne :

```
vente =[["Peugeot",80], ["Fiat",110], ["Renault",105], ["Citroen",180]]
```

La structure du tableau est en réalité celle d'un tableau de 4 cellules, contenant chacune 2 cellules :

0		1		2		3	
0	1	0	1	0	1	0	1
Peugeot	80	Fiat	110	Renault	105	Citroen	180

Cependant, le fonctionnement est similaire à un tableau à deux dimensions :

	0	1
0	Peugeot	80
1	Fiat	110
2	Renault	105
3	Citroen	180

L'accès aux données se fera avec les indices habituels : le premier paramètre correspond à l'indice de la cellule qui contient le mini-tableau, le deuxième argument est l'indice dans le "sous-tableau".

Exemple pour la troisième vente :

```
venteMarque = vente[2][0]          // renvoie Renault
venteNombre = vente[2][1]          // renvoie 105
```

Il s'agit de tableaux imbriqués et ils sont proches des objets. Ils sont également proches des tableaux associatifs. **Un tableau associatif** est une association d'une clé et d'une valeur (donc 2 dimensions). Ici la clé doit être unique et la paire clé-valeur séparée par une virgule. Le séparateur entre la clé et la valeur est le symbole' : '. Enfin, les valeurs ne sont pas forcément un nombre.

```
CodePostal = {"Paris":75000, "Lyon":69000, "Grenoble":38000}
```

2.6.4 Les variables "objets"

Comme JavaScript est orienté objet, il supporte donc la création de variables objets mais, il ne supporte pas la notion de classe, au sens habituel.

En effet, JavaScript considère les objets comme un tableau associatif. Cela permet une déclaration "originale". Pour cela, JavaScript encadre chaque objet par les accolades { et } (les mêmes que pour un bloc de code). Voici un exemple :

```
let Voiture = {
      // des attributs...
      "marque" : "",
      "modele" : "",
      "annee" : ,
      "puissanceWatt" : 0,

      // une méthode !
      "puissanceCh" : function() {
            return this.puissanceWatt * 1.36
      }
};

let maVoiture = new Voiture("Peugeot", "308", 2011, 97)
console.log(maVoiture.modele)
```

On peut aussi déclarer

```
let maVoiture = {marque : "Peugeot", modele :"308", annee:2011, puissanceWatt:97}
```

Enfin, on peut utiliser la notation objet pour retrouver une propriété. De quelle année est ma voiture ?

```
let maVoiture = {marque : "Peugeot", modele :"308", annee:2011, puissanceWatt:97}
console.log("Année de fabrication : ", maVoiture.annee)
```

2.6.5 Les dictionnaires (clé-valeur)

Il s'agit d'une fonctionnalité très intéressante pour trouver les propriétés correspondantes à une clé.

2.6.5.1 Déclaration et définition d'une Map

L'indice du tableau n'est plus un nombre (commençant à zéro) mais directement la clé. L'exemple suivant montre comment utiliser de tels tableaux.

```
let stockFruits = new Map()
stockFruits.set("orange", 100)
stockFruits.set("banane", 65)
stockFruits.set("pomme", 200)
stockFruits.set("kiwi", 50)

let fruitChoisi = "pomme"
console.log("Quantité de "+fruitChoisi, stockFruits.get(fruitChoisi))
console.log("Quantité de kiwi", stockFruits.get("kiwi"))
```

L'intérêt est que l'on peut accéder immédiatement à la valeur de la clé, sans utiliser de boucles pour lire toutes les valeurs du tableau.

Attention, il est impératif que les clés soient uniques.

2.6.5.2 Lecture de la table (Map)

On peut tout de même lire le tableau à l'aide d'une boucle `forEach` dans laquelle nous passons une fonction qui récupère la clé et la valeur (notez l'ordre inversé) :

```
console.log("Longueur tableau : ", stockFruits.size)
console.log('\n')
console.log("Récapitulatif de notre stock")
stockFruits.forEach( (valeur, cle) => {
  console.log(valeur, cle)
})
```

2.6.5.3 Effacement d'une clé

Enfin, il est possible d'effacer un élément du dictionnaire avec .delete()

```
stockFruits.delete("kiwi")
```

2.7 LES ITÉRATIONS (RÉPÉTITIONS)

JavaScript gère évidemment les itérations : les itérations fixes et les itérations conditionnelles.

2.7.1 Itérations fixées au début : boucle for

Pour lire un tableau, la boucle **for** reste efficace :

```
const tableau = [0,5,10,15,20]
for (let index = 0; index < tableau.length; index++) {
  console.log(tableau[index])
}
```

For (arg1 ; Arg2 ; Arg3) { … }
- Arg1 : initialise la variable au début
- Arg2 : condition de fin de boucle
- Arg3 : choix du pas

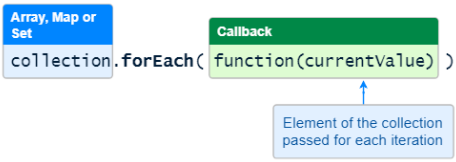
La boucle **for..in** est très élégante :

```
const tableau = [0,5,10,15,20];
for (elm in tableau) {
      document.write('['+elm+']')
      // affichera [0][5][10][15][20]
}
```

Elm contient directement la copie du contenu de la case du tableau en cours...

La boucle **forEach** fait appel aux fonctions anonymes ou fonctions fléchées (programme SIO2).

```
const tableau = [0,5,10,15,20]
tableau.forEach(elm => document.write('['+elm+']'))

// affichera [0][5][10][15][20]
```



Pour en savoir plus, https://www.hugopich.com/fr/javascript-foreach/

2.7.2 Les itérations conditionnées : boucle while

While en anglais se traduit par "tant que" et cela signifie qu'il va y avoir une comparaison.

Il y a cependant deux possibilités pour effectuer cette boucle :

- En testant avant de rentrer dans le bloc de répétition
- En testant à la fin du bloc de répétition

2.7.2.1 Itération avec condition en amont

Il est également possible d'utiliser une boucle **while** :

```
let index = 0;
const tableau = [0,5,10,15,20];
while (index < tableau.length) {
  console.log(tableau[index]);
  index++;
}
```

Tant que la condition est vraie, on exécute l'intérieur du bloc.

Il faut donc faire varier la valeur utilisé dans la condition (sinon on a une boucle infinie)

La boucle précédente peut ne jamais s'exécuter – par exemple, si le tableau est vide – ce qui évitera un plantage du programme ;

2.7.2.2 itération avec condition en aval

En revanche, si l'action est à faire une fois au moins, il faut parcourir une fois le bloc puis tester la condition.

Il existe donc la boucle do… While (la condition est testée après le code à exécuter).

```
do {
  // code a exécuter
} while (condition) ;
```

2.8 LES CONDITIONS

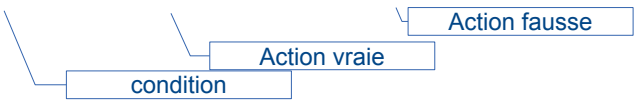
Comme les autres langages, JavaScript permet d'écrire des tests pour valider des conditions.

2.8.1 If … else if et else…

```
if (age > 21) {
    text = "Majeur tout pays"
    autorisation = true
} else if (age > 18) {
    text = "Majeur France"
    autorisation = true
} else {
    text = "Mineur"
    autorisation = false
}
```

Il est également possible d'utiliser une forme abrégée dans le cas de tests ayant une seule action par possibilité :

```
text = (age < 18) ? "Trop Jeune" : "Majeur" ;
```



2.8.2 Comparaisons et opérateurs

Enfin, les opérateurs suivants sont utilisés :

Opérateur	Signification
< *ou* >	Plus petit *ou* plus grand : if (a < b) ou bien if (a > b)
<=	Plus petit ou égal : if (a <= b)
>=	Plus grand ou égal
==	Égal : if (a == b)
!=	Différent : if (a != b)
===	Égal en valeur ET en type : if (nom === surnom)

JavaScript accepte également les opérateurs logiques dans les conditions :

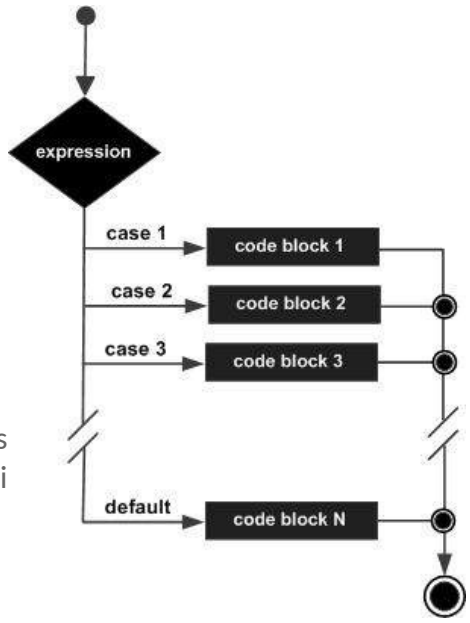
ET	OU	NON
&&	||	!
If (Test1==5 && Test2<0)	If (Test1==5 || Test2<0)	If (!(Test1==5))

2.8.3 switch… case

Et enfin, JavaScript accepte également les conditions sous forme de sélection : switch et case.

Exemple :

```
switch (nbLine) {
    case 0 : Text="pas de lignes"
        break
    case 1 : Text="une seule ligne"
        break
    default : Text=nbLine + "lignes"
}
```



N'oubliez pas l'instruction break à chaque étape.

Une particularité de JavaScript, est d'accepter un test dans les valeurs du case : cela permet l'usage de plage de sélection, ce qui est plus pratique :

Exemple :

```
switch (true) {
    case (x < 5):
        alert("less than five")
        break
    case (x < 9):
        alert("between 5 and 8")
        break
    case (x < 12):
        alert("between 9 and 11")
        break;
    default:
        alert("none")
        break
}
```

Ne pas oublier l'instruction **break**, sinon le programme exécutera toutes les conditions vraies

2.8.4 Tester un 'NaN' (Not a Number)

Il est possible de vérifier si le résultat d'un champ est un nombre ou pas, avec la fonction isNaN(x) :

```
if (isNaN(age)) message = "la saisie n'est pas un nombre"
```

2.8.5 Tester un 'undefined' (non défini)

Il est possible de vérifier si le résultat d'un champ est un nombre ou pas, avec la fonction isNaN(x) :

```
if (typeof variable === "undefined") message = "la saisie est indéfinie"
```

2.9 LES FONCTIONS

2.9.1 Fonction classique

Comme pour PHP et Java, JavaScript est capable d'utiliser les fonctions, avec paramètres d'entrées et paramètre de retour, mais contrairement aux langages typés, il n'est pas nécessaire d'indiquer un type dans la déclaration de la fonction.

Pas de déclaration de type

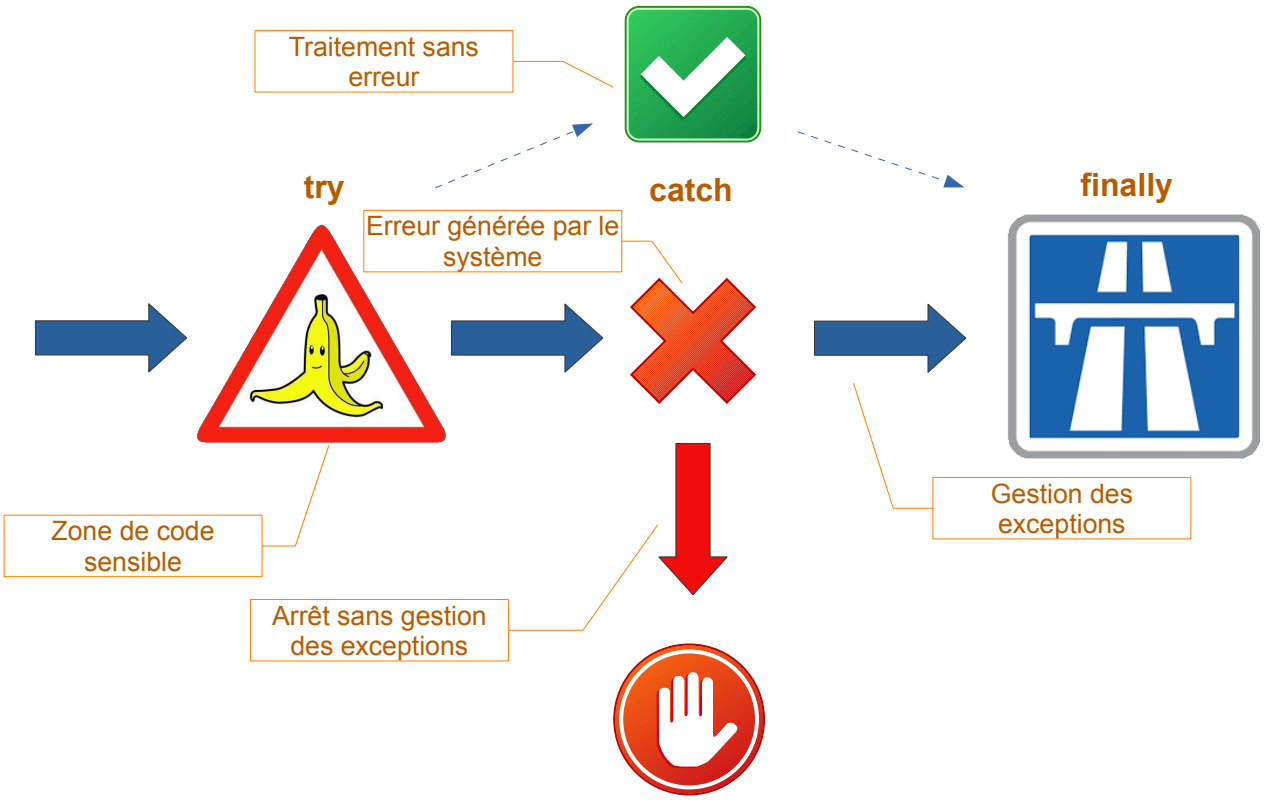
```
function auCarre(nombreA) {
      return nombreA * nombreA ;
}
```

JavaScript étant inspiré de Java, on trouve également l'usage de constructeur et l'instanciation d'objet.

```
<SCRIPT LANGUAGE="JavaScript">
      let maintenant=new Date();
      document.write(maintenant);
</SCRIPT>
```

2.10 LES EXCEPTIONS

Voici le schéma de fonctionnement des exceptions.



Avant la zone de code sensible, on ouvre un bloc try {...}.

À la sortie de la zone on ferme le bloc et on ouvre un bloc catch {...} qui contient ce qu'il faut faire après une erreur.

On termine avec le reste du code normal.

On peut comparer un try-catch à une condition "si tout va bien, continuer ; sinon gérer le problème puis continuer".

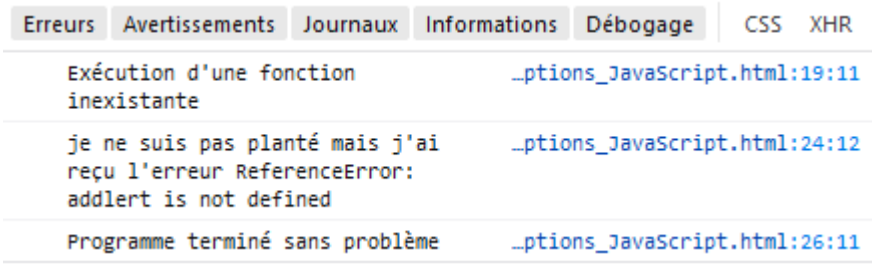
2.10.1 Exemple de gestion

2.10.1.1 Exemple de base

Voici le même code que précédemment, mais en y ajoutant la gestion des exceptions.

```
// Cet exemple ne plante pas, car si la fonction n'existe pas, l'erreur est gérée
console.log("Exécution d'une fonction inexistante")
try {
      console.log(addlert("Quoi ?"))
}
catch(err) {
      console.log("je ne suis pas planté mais j'ai reçu l'erreur "+err)
}
console.log("Programme terminé sans problème")
```

Résultat :



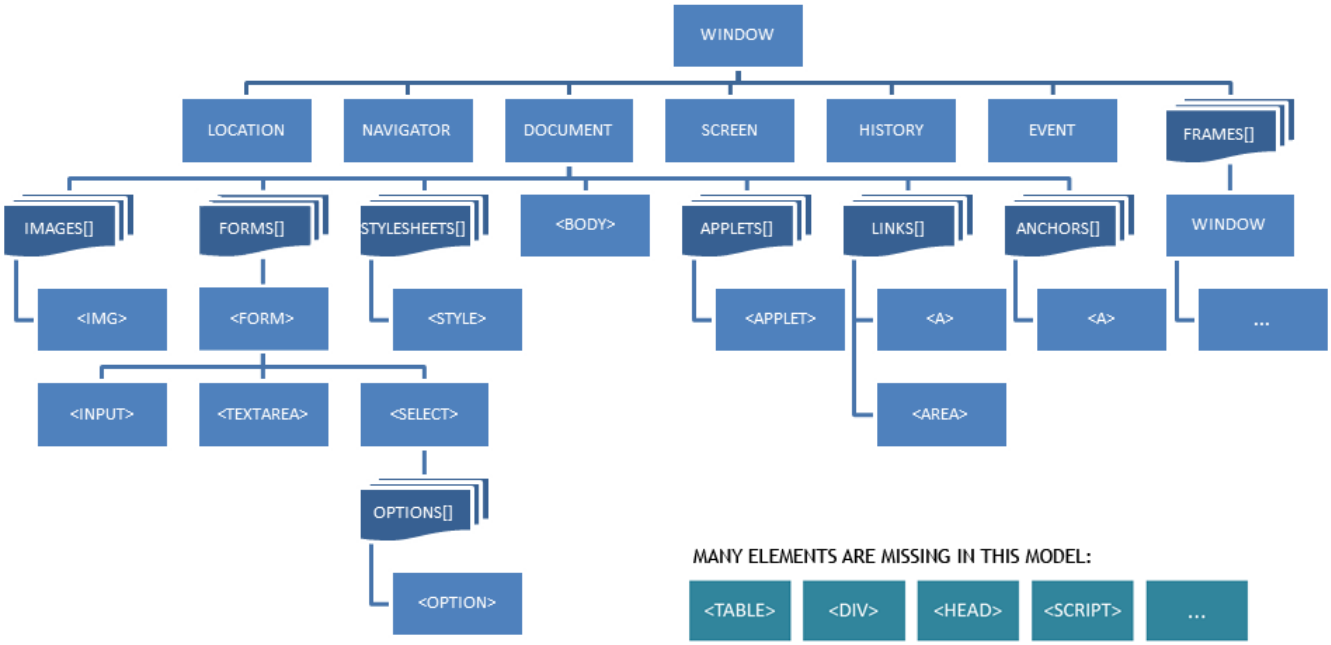
Comme le montre l'exemple, le bloc à l'intérieur du try ne s'est pas exécuté mais celui à l'intérieur du catch s'est bien déclenché.

On dit que le programme a **levé une exception** (en anglais, on dit "**throw an exception**").

2.11 DOM ET ÉVÉNEMENTS

Le DOM est la structure de la page web. Chaque fois qu'une balise est utilisée, un nœud est créé dans le "Document Object Model". Le DOM mémorise les propriétés de chaque nœud (valeur, couleur, etc.)

Voici la figure disponible sur Internet la plus complète du DOM :



La description du DOM utilise un '.' entre chaque nœud. Pour indiquer ou accéder à une propriété de la balise <body> il faudra donc écrire :

```
window.document.body.xxxxxxx
```

Le nœud "document" contiendra les balises de la page affichée sur l'écran de l'utilisateur, mais il existe d'autres nœuds très intéressants.

2.11.1 Propriétés générales

Le DOM contient les propriétés de la page mais aussi du navigateur. Voici quelques propriétés utiles.

Élément DOM	fonction

`window.location.href`
`window.location.hostname`
`window.location.pathname`
`window.location.protocol`
`window.location.port`
	`https://www.w3schools.com/js/js_window_location.asp`
	`www.w3schools.com`
	`/js/js_window_location.asp`
	`https`
	`""`

`navigator.userAgent`	Donne l'agent utilisé pour la navigation. Exemple ► *"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0"*
`navigator.cookieEnabled`	Donne la propriété qui gère l'acceptation des cookies Exemple ► *true*
`navigator.platform`	Donne le type de système d'exploitation Exemple ► *Win64*
`navigator.language`	Donne la langue du navigateur Exemple ► *fr*
`navigator.onLine`	Indique si le navigateur est en ligne (ou en mode hors connexion) Exemple ► *true*
`navigator.javaEnabled()`	Indique si Java est validé dans le navigateur Exemple ► *false*
`window.innerWidth` `window.innerHeight`	Donne la taille de la page écran disponible Exemple ► *1152x983*
`screen.width et screen.height` `screen.colorDepth`	Donne la résolution de l'écran. Méthode dépréciée (mais utile pour les hackers). Exemple ► *1920x1080*
`navigator.appName` *`navigator.appCodeName`* *`navigator.product`* *`navigator.appVersion`*	*Ensemble de propriété sur le navigateur. Souvent erronées : appName donne généralement Netscape, appCodeName donne Mozilla et product donne Gecko, quel que soit le navigateur.*

2.11.2 Actions sur le navigateur

Il s'agit de fonctions utilisant le DOM pour interagir avec le navigateur de l'utilisateur.

Élément DOM	fonction
`window.history.back()` `window.history.forward()`	Accède à la page précédente du navigateur (équivalent au bouton) 
`window.location.assign("http://www.hp.com")`	Naviguer vers une nouvelle page

`window.open()`
`window.close()`
`window.moveTo()`
`window.resizeTo()`
	Ouvre une fenêtre
	Ferme la fenêtre courante
	Déplace la fenêtre courante
	Redimensionne la fenêtre courante

`setTimeout(function, mseconds)`
`clearTimeout(myVar)`
	Exécute une fonction une fois, à la fin du décompte
	`var tm = setTimeout (fonction, 1000) ;`
	`clearTimeout(tm) ;`

`setInterval(function, mseconds)` `clearInterval(myVar)`	Exécute en boucle une fonction selon un intervalle

2.11.3 Liste des événements

Quelques événements générés par l'utilisateur sur le navigateur ou les formulaires de la page web :

	HTML	JavaScript	Description
	onchange	change	Lorsqu'un élément HTML a été changé et perd le focus
	oninput	input	Lorsqu'un élément HTML a été changé
	onclick	click	Lorsque l'utilisateur clique sur un élément
	oncontextmenu	contextmenu	Lorsque l'utilisateur effectue un clic droit pour afficher le menu
	ondblclick	dblclick	Lorsque l'utilisateur double clique sur un élément
	onmouseover	mouseover	Lorsque la souris survole un élément
	onmouseout	mouseout	Lorsque la souris sort de l'élément survolé
	onkeydown	keydown	Lorsque l'utilisateur enfonce une touche du clavier
	onkeypress	keypress	Lorsque l'utilisateur a fini d'enfoncer une touche du clavier
	onkeyup	keyup	Lorsque l'utilisateur relâche une touche du clavier
	onload	load	Lorsque le navigateur a terminé le chargement de la page
	onresize	resize	Lorsque un élément est défilé (vers le bas ou le haut)
	onscroll	scroll	Lorsque un élément est défilé (vers le bas ou le haut)

Tableau 1 : quelques événements courant en JavaScript

Notez qu'il existe encore d'autres nombreux événements :
https://www.w3schools.com/jsref/dom_obj_event.asp

2.11.4 Ajout d'événement en JavaScript (addEventListener)

Pour ajouter un événement à un objet en JavaScript, on utilise `addEventListener(type, fonction)`

Dans ce cas, type prend une valeur du tableau précédent (par exemple `keyup`).

Enfin, dans la fonction, on peut savoir quel élément a été cliqué avec `event.target.id` et obtenir sa valeur avec `event.target.value`.

```
let otxtNom = document.getElementById("txtNom")
otxtNom.addEventListener("input", maFonction)

function maFonction(event) {
    console.log("ID:"+event.target.id, "Value:"+event.target.value)
}
```

2.11.5 Code d'exemple du DOM

Voici un code très simple qui montre comment rendre une page dynamique et adaptable.

```
<!DOCTYPE html>
<html lang="fr">
<head>
	<meta charset="UTF-8">
	<title>Exemple d'utilisation du DOM et des événéments</title>
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<body onresize="afficheInfo()">
	<h1 onclick="alert('clic sur le titre')">Evénements et DOM</h1>
	<p id="positionTaille">Changez la taille de la page...</p>
	<script>
		function afficheInfo() {
			console.log("URL du navigateur : "+window.location.href)
			console.log("Hostname du serveur : "+window.location.hostname)
			console.log("Langue du navigateur : "+navigator.language)
			let defScreen = screen.width + "x" + screen.height
			let defPage = window.innerWidth + "x" + window.innerHeight
			console.log("Votre écran : "+defScreen)
			console.log("Votre page : "+defPage)
			document.getElementById("positionTaille").innerHTML = defPage
		}
	</script>
</body>
</html>
```

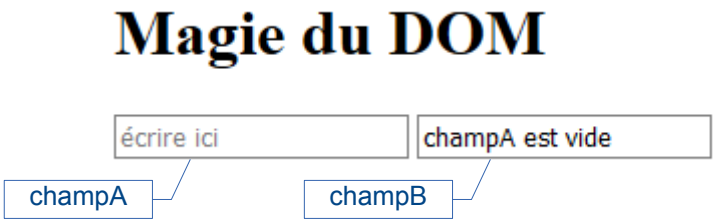
2.11.6 Récupérer / modifier les valeurs d'une propriété du DOM

Parmi les différentes méthodes d'accès au DOM en JavaScript, voici les deux principales méthodes pour récupérer une valeur :

- document.getElementById(*paramètre*)
- document.querySelector(*paramètre*)

Dans les deux cas, les propriétés sont récupérées sous forme de texte (ou d'objet). Il convient d'effectuer une conversion en nombre avec la fonction Number() si nécessaire.

Prenons le cas d'un formulaire :



Le code est le suivant :

```
<h1>Magie du DOM</h1>
<input type="text" id="champA" placeholder="écrire ici" onkeyup="afficheInfo()"/>
<input type="text" id="champB" value="vide"/>
```

L'objet input type='text' dispose des propriétés suivantes (ordre alphabétique) :

autocomplete	Accepte l'auto-complétion (on | off)
autofocus	Donne le focus à cet objet au chargement de la page (true | false)
defaultValue	Donne la valeur initiale (si définie) de l'objet
disabled	Dévalide la possibilité d'utiliser l'objet (true | false)
form	Donne l'ID du formulaire qui contient cet objet (note : lecture seulement)
list	S'utilise avec datalist et permet de proposer la complétion avec des valeurs prédéfinies
maxLength	Spécifie la taille de l'objet en nombre de caractère
name	Donne le nom de l'objet (différent de l'ID). Utilisé pour transmettre un formulaire (PHP)
pattern	Donne le motif REGEX appliqué à l'objet
placeholder	Donne le texte d'aide qui s'affiche lorsque l'objet est vide
readOnly	Verrouille l'objet en lecture seulement (true | false)
required	Valide la nécessité de remplissage du champ lors d'une submission de forme (true | false)
size	Donne la taille de l'objet en nombre de caractère
type	Donne le type de l'objet
value	Donne la valeur contenue dans l'objet actuellement (lecture ou écriture)

2.11.6.1 document.getElementById

C'est la méthode la plus rapide, qui renvoie l'index de l'objet recherché dans le DOM :

```
let oChampA = document.getElementById("champA") // récupère l'emplacement
let leNombre = Number(oChampA.value)            //lecture de la valeur de l'objet
oChampA.value = "15"                            //modification du contenu
```

2.11.6.2 document.querySelector

Cette méthode est moins rapide mais s'applique sur tout le DOM (et pas seulement à partir du nœud "document"). De plus, comme `querySelector()` s'applique à tous les attributs : une classe ou un ID. Pour cela, il faut faire précéder le champ recherché par un `.` pour une classe et un `#` pour un ID.

```
let leNombre = Number(document.querySelector("#champA").value) //lecture
document.querySelector("#champB").value = "15" //modification
```

2.11.6.3 Exemple getElementByID et querySelector

Voici un exemple de code utilisant les deux méthodes.

```
<!DOCTYPE html>
<html lang="fr">
<head>
	<meta charset="UTF-8">
	<title>Exemple d'utilisation du DOM et des événéments</title>
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<link rel="stylesheet" type="text/css" href="styles.css" />
	<script>
		function afficheInfo() {
			let defPage = window.innerWidth + "x" + window.innerHeight
			let oChampA = document.getElementById("champA")
			let oChampB = document.querySelector("#champB")
			if (oChampA.value != "") {
				oChampB.value = defPage
				oChampB.style.backgroundColor = "white"
			} else {
				oChampB.value = "champA est vide"
				oChampB.style.backgroundColor = "red"
			}
		}
	</script>
</head>
<body onresize="afficheInfo()">
	<h1>Magie du DOM</h1>
	<input type="text" id="champA" placeholder="écrire ici"
onkeyup="afficheInfo()"/>
```

```
      <input type="text" id="champB" value="vide"/>
</body>
</html>
```

Il est également possible de modifier le style CSS d'un élément. Par exemple, lors du survol de cet élément (cet exemple permet également de voir deux événements associés au même élément) :

```
<span onmouseover="this.style.color='red'"
onmouseout="this.style.color='black'">Mouse over me!</span>
```

Dans cet exemple, le texte s'affichera en rouge lorsque la souris sera au-dessus, et reprendra une couleur noire à sa sortie.

2.11.6.4 Changer le titre d'une page

Le titre de la page dans un onglet de navigateur peut être modifié dynamiquement. Certains font même défiler les mots (scrolling). Sans aller jusque là, voici la commande :

```
let oldTitle = document.title ;
document.title = "Nouveau titre" ;
```

2.11.6.5 Trouver des éléments multiples (listes/collections)

Les listes sont récupérables avec document.querySelectorAll(). Le résultat est alors une table.

Voici comment trouver toutes les cases cochées d'un document :

```
let CasesChecked = document.querySelectorAll('input[type=checkbox]:checked');

// Lecture de cette liste
for (var i = 0; i < CasesChecked.length; i++) {
  console.log(CasesChecked[i].value);
}
```

2.11.6.6 Validation de formulaire

Le principe est de vérifier avant l'envoi d'un formulaire que l'ensemble des champs sont correctement remplis.

https://openclassrooms.com/courses/tout-sur-le-javascript/td-verification-d-un-formulaire

Au passage, pour aligner les champs en CSS dans les formulaires :

https://openclassrooms.com/courses/aligner-ses-formulaires-sans-tableaux

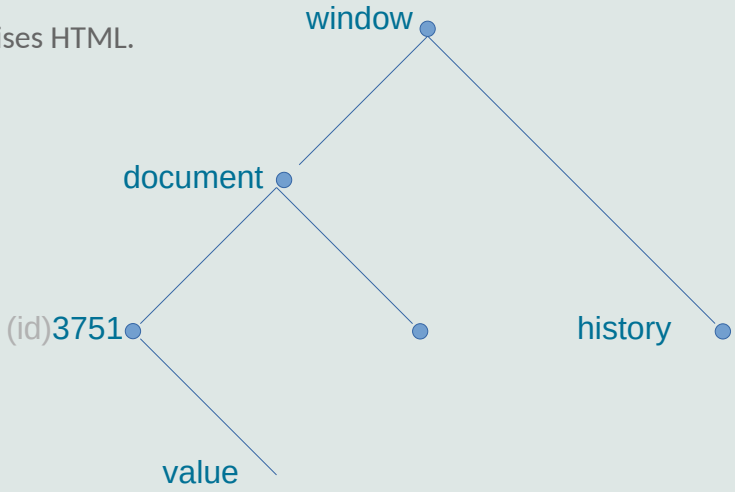
2.12 LE DOM, EN RÉSUMÉ

DOM signifie Document Object Model : c'est une arborescence qui représente la structure (le squelette) d'une page web.

2.12.1 Structure

Chaque nœud du DOM est séparé par un point.

Sous le nœud "document" on trouve les balises HTML.



2.12.2 Propriétés, méthodes, événements

Le DOM contient pour chaque balise ou champ, un ensemble de choses :

- Les **propriétés** (qui sont comme des variables, exemple : navigator.**language**)
- Les **méthodes** (qui sont des fonctions, exemple : document.**querySelector(paramètre)**)
- Les **événements** (qui sont des réactions : survol de souris, touche enfoncé, taille de la page… exemple : **onload**)

2.13 CANEVAS ET SVG

Dans le support de cours HTML, les balises <canvas></canvas> ont été décrites mais pas utilisées. La raison est simple, la surface du canevas ne peut être écrite qu'avec des instructions JavaScript.

2.13.1 canevas

Voici un exemple facile à comprendre : le programme créé une zone canevas puis dessine dessus.

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="UTF-8">
	<title>SI4 - Canvas</title>
</head>
<body>
	<h1>SI6 - Utilisation du canevas</h1>
	<canvas id="maZone" width="500" height="400">Zone graphique</canvas>
	<div id="TXT">Type de canevas : "</div>
	<progress id="PROGBAR" width="400" height="15" value="40" max="100">40%</progress>

	<script>
		let leCanevas = document.getElementById("maZone");
		if (leCanevas.getContext) {
			dessin = leCanevas.getContext("2d");
			document.getElementById("TXT").innerHTML = leCanevas;
			dessin.strokeStyle = "#002277";
			dessin.strokeRect(0,0,499,399);
			dessin.fillRect(50,50,400,300);
			dessin.stroke();
			// Face
			dessin.beginPath();
			dessin.lineWidth = 15;
			dessin.fillStyle = "#AA8800";
			dessin.arc(250,200,120,0,2*Math.PI);
			dessin.fill();
			dessin.stroke();
			// smile and eyes
			dessin.beginPath();
			dessin.lineWidth = 5;
			dessin.arc(250,200,80,Math.PI, 0, true);
			dessin.stroke();

			dessin.beginPath();
			dessin.arc(200,150,30,0,2*Math.PI);
			// oubli volontaire du beginPath()
			dessin.arc(300,150,30,0,2*Math.PI);
			dessin.stroke();
			// Titre du dessin
			dessin.font = '40pt Calibri';
			dessin.fillStyle = 'grey';
```

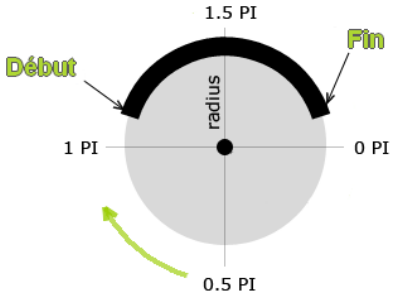
```
                dessin.fillText('Hello World!', 130, 60);
            }
        </script>

</body>
</html>
```

Le dessin est très rudimentaire et est composé de trait, d'arc, de rectangle et de texte. Il est aussi possible d'afficher une image. Des opérations de rotations et de transformations sont également disponibles.

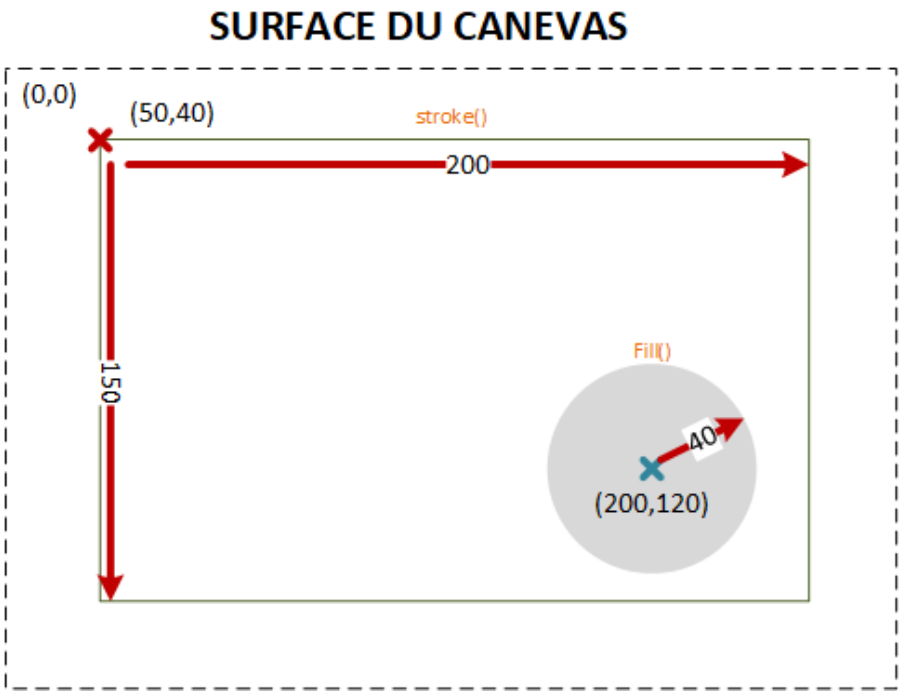
Il faut comprendre que la zone de dessin commence aux coordonnées 0,0 en haut à gauche de la zone.

Pour les cercles et arcs de cercle, le tracé se fait par défaut dans le sens horaire (sauf si le paramètre N°6 de la fonction arc() est mis à ***true***).

Les angles sont exprimés en radians (un tour = 2π).



Ainsi, JavaScript permet de placer les éléments sur le canevas :



La gestion des couleurs est indépendante pour le remplissage (fill) et pour les lignes (stroke).

- fillStyle = "#AA8800";
- strokeStyle = "#002277";

D'autre part, entre chaque figure, il est recommandé d'utiliser la commande **beginPath()** pour éviter qu'un trait ne les relie toutes.

Enfin, pour le même tracé, il est possible de lui appliquer le tracé de ligne ET le remplissage en utilisant de manière consécutive **fill()** et **stroke()**.