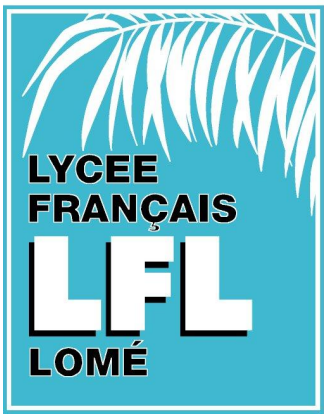
Le langage

Javascript

Partie 2



Sommaire :

I. Les fonctions du noyau :

JavaScript fait partie d'une normalisation par l'ECMA intitulé ECMA-262.
Cette norme spécifie toutes les règles de syntaxes et commandes de bases du langage *(vue dans la première partie de la formation).*
Elle définit également les objets de base existant dans toutes implémentations **JavaScript**. On retrouvera l'ensemble de ces objets dans tous les langages utilisant cette spécification comme base du langage. Parmi ceux-ci :

- **JavaScript** pour les navigateurs ;
- **JavaScript** pour Adobe Acrobat ;
- **ActionScript** pour Flash.

Donc, en apprenant **JavaScript**, vous apprenez également les règles de base pour comprendre le langage ActionScript de Flash. Il restera à connaître les fonctions à utiliser dans le cas d'une manipulation d'animation.

Dans ce chapitre, nous allons voir les différents objets présent dans le noyau de **JavaScript**, ainsi que leurs propriétés et leurs méthodes.

Cette partie de la formation va lister la plupart des méthodes des objets du noyau. Ces méthodes renvoient parfois des valeurs, mais surtout prennent des arguments d'un certain type. Ainsi, lors de la présentation la fonction `substr()` de l'objet `String`, on notera :

String `str.substr(`*number* `num1,`*number* `num2)`

Les informations écrites en italique indiquent que les arguments `num1` et `num2` sont de types *Number* et que cette fonction renvoie une valeur qui est de type `String`

A. Global :

Dans la construction du langage **JavaScript**, tout est question d'objet. On a vu qu'un objet peut contenir des sous-objets.
En fait, il existe un objet contenant tous les autres : c'est l'objet global. Cet objet n'a pas besoin d'être cité explicitement puisqu'il est considéré comme le "*tout*" de tout ce que contient le langage.
Dans le corps même du script, il est accessible via le mot-clef `this`.

Pour l'implémentation **JavaScript** dans les navigateurs, cet objet a également pour nom `window`.

Dans le cas du noyau **JavaScript** *(norme ECMA-262)*, voici les propriétés et les méthodes de l'objet global :

Propriété :

number `NaN`

objet de type `Number` signifiant “Not-A-Number”

number `Infinity`

objet de type `Number` représentant la valeur $+\infty$

`undefined`

valeur repésentant une valeur non définie. Par exemple, dans le cas d'une variable déclarée mais non-affectée

Etant des propriétés de l'objet global, les deux instructions suivantes sont équivalentes :

```
var a=NaN;     var a=window.NaN;
```

Méthode :

`eval(`*string* `str)`

str est une chaîne de caractères contenant du code **JavaScript**. La méthode `eval` va évaluer le code contenu dans la chaîne `str`.
Le résultat retourné par cette méthode sera celui de la dernière instructions de `str` exécuté.

int `parseInt(`*string* `str,` *int* `base)`

La méthode `parseInt()` va convertir en nombre la chaîne de caractère `str`. L'argument `base` est optionnel, si il est présent cette méthode essayera d'interprêter la chaîne `str` comme une représentation d'un nombre en base `base`.

number `parseFloat(`*string* `str)`

`str` est une chaîne de caractères contenant la représentation d'un nombre décimal. Cette méthode renverra le nombre décimal contenu dans cette chaîne.

boolean `isNan(`*number* `nbr)`

renverra `true` si `nbr` a la valeur `NaN`, et renverra `false` dans le cas contraire.
Renvoie la valeur `NaN` en cas d'impossibilité de conversion de la chaîne en nombre.

boolean `isFinite(`*number* `nbr)`

renverra `true` si `nbr` a une valeur finie et renverra `false` dans le cas contraire.

Renvoie la valeur `NaN` en cas d'impossibilité de conversion de la chaîne en nombre.

string `encodeURI(`*string* `str)`

Cette méthode certains caractères de `str` par leurs séquences d'échappement afin de la rendre compatible avec les chaînes de caractères utilisées comme adressse :

`encodeURI("2 3")` renverra `"2%203"`

Le caractère espace n'est pas accepté dans une adresse et sera remplacé par la séquence d'échappement `%20`.

string `decodeURI(`*string* `str)`

Cette méthode remplace toutes les séquences d'échappement présent dans `str` par les caractères correspondants, pour la rendre humainement lisible.

B. Object :

En programmation orientée objet, on peut créer de nouveaux types d'objets en se servant d'objets existant. Le nouvele objet possédera les propriétés et méthodes de l'objet ayant servi de modèle plus les nouvelles propriétés et méthodes le caractérisant.

L'objet `Object` fait partie du noyau de **JavaScript**, et tout objet de **JavaScript** est crée à partir de celui-ci. Il contient donc le minimum de propriété et de méthode permettant de définir un objet, et tout autre objet possèdera au minimum celle-ci : tout objet aura pour patron `Object`.

La plupart des propriétés et méthodes présentées ici n'ont pas d'intérêt pour le programmateur débutant et moyen.

`obj` représente un objet de type *object*.

Propriété :

function `obj.constructor`

Cette méthode représente le constructeur qui a permis de créer l'objet `obj`

Méthode :

string `obj.toString()`

Cette méthode convertit `object` en chaîne de caractères *(voir paragraphe sur la conversion de types)*.

Tout objet **JavaScript** possède cette méthode héritée de `Object`, mais celle-ci peut être inefficace tel que fournit initialement, il sera nécessaire alors de la réécrire dans

le cas d'objet créé hors du noyau.

string `obj.toLocaleString()`

Cette méthode est identique à la méthode précédente, mais prend en compte les paramètres locaux définis sur l'ordinateur du client lors de la conversion :

- pour les nombre décimaux, le séparateur entre la partie entière et décimale sera affiché avec un point en Amérique, et avec une virgule en France
- il en est de même pour les dates.

`obj.valueOf()`

Cette méthode renvoie une valeur primitive *(nombre, chaîne, booléen)* censée représentée `obj`.
Cette méthode est hériée par tous les objets **JavaScript**.
Les objets du noyau redéfinissent cette méthode à partir du prototype afin de renvoyer une valeur significative.

boolean `obj.hasOwnProperty(`*string* `prop)`

Cette méthode vérifie si `prop` est une propriété de l'objet `object` : elle renverra `true` dans ce cas, `false` sinon. Elle ne vérifiera pas la présence de la propriété dans le prototype de l'objet : ainsi, `obj.hasOwnProperty("toString")` renverra `false`.

boolean `Obj.prototype.isPrototypeOf(obj)`

`Obj` représente l'objet-constructeur d'un type d'objet.
Cette méthode érifie si `Obj` est inclus dans le prototype de `obj` : c'est à dire si l'objet `obj` a été construit à un moment donné à partir du modèle de l'objet `Obj`.

```
1  <script>
2  var a= new Boolean(true);
3  chaine=Boolean.prototype.isPrototypeOf(a);
4  chaine+=" - ";
5  chaine+=Object.prototype.isPrototypeOf(a);
6  chaine+=" - ";
7  chaine+=Function.prototype.isPrototypeOf(a);
8
9  document.write(chaine);
10 </script>
```

boolean `obj.propertyIsEnumerable(`*string* `prop)`

Cette méthode vérifie si `prop` est une propriété énumérable *(n'ayant pas l'attribut dontEnum)* de l'objet `obj`.
Elle renvoie `true` dans ce cas et `false` dans le cas contraire.

C. Function :

Toute fonction en **JavaScript** découle de l'objet `Function`. Généralement on n'utilise pas le constructeur de cet objet pour créer, on utilise plutôt la première séquence ci-dessous :

```
<script>
 function multiplie1(arg1,arg2){
  return (arg1*arg2);
 }

 //Cette fonction aurait pu être aussi définie de la manière suivante
 var multiplie2 = new Function("arg1","arg2","return (arg1*arg2);");
</script>
```

Les fonctions `multiplie1` et `multiplie2` sont parfaitement équivalentes : la seconde est déclarée à l'aide du constructeur de l'objet `Function`.

Lors de l'appel à une fonction, **JavaScript** crée l'objet `arguments` qui sera local au corps d'exécution de la fonction et permettant de manipuler l'ensemble des arguments passés lors de l'appel à la fonction *(même ceux qui n'ont pas été prévus lors de la déclaration de la fonction).*

`fct` représente un objet de type *function*.

Propriété :

int `fct.length`

Cette propriété représente le nombre d'arguments demandés lors de la déclaration de la fonction.

Méthode :

Considéré comme hérité de `Object`, tout objet de **JavaScript** hérite des méthodes `toSource()` et `toString()`

`fct.apply(`*object* `thisValue,`*array* `arg)`

Cette méthode permet de lancer un appel à la fonction `fct` de la part de l'objet `thisValue` : dans le corps de déclaration de la fonction, `this` aura pour valeur `thisValue` ; si `thisValue` a pour valeur `undefined` ou `null`, **JavaScript** lancera l'appel de la part de l'objet global.
On peut fournir l'argument `arg` qui est optionnel : ce tableau fournira l'ensemble des arguments lors de l'appel à cette fonction.
Cette méthode renverra la valeur de retour de la fonction si celle-ci existe.

`fct.call(`*object* `thisValue, arg1, arg2,...)`

Cette méthode est identique à `apply()` sauf que les arguments passés *(si ils existent)* sont passés les uns après les autres dans l'appel de la méthode `call`.

Ces deux dernières méthodes servent pour hériter une méthode d'un objet sur un autre afin d'éviter de la réécrire : autant dire qu'elle ne servira qu'au programmeur chevronné.

D. Array :

Les tableaux sont des objets existant dans tous les langages de programmation.
Ce sont des conteneurs : leur seule utilité est de stocker un ou plusieurs objet en leur sein ; objets pas nécessaire du même type.
On distingue habituellement deux types de tableaux :

- les tableaux à index numérique : chaque élément est repéré par son ordre d'insertion dans le tableau. On parlera du rang de l'élément dans le tableau.
  Le premier élément a pour rang 0, le second élément a pour rang 1...
- les tableaux associatifs : chaque élément est identifié dans le tableau par une chaîne de caractères. On parle alors de paire "*clé/valeur*" pour parler de identifiant/élément.

En **JavaScript**, les tableaux sont de type *Array* et sont crées

- explicitement par l'objet `Array`.
  ```
  var a=new Array("Bonjour","Hello","Buenos dias");
  ```
  Ce tableau possède trois éléments.
- implicitement en utilisant l'écriture littérale d'une tableau :
  ```
  var b=["Bonjour","Hello","Buenos dias"]
  ```
  Les variables `a` et `b` ainsi crées sont totalement équivalentes.

Voici la syntaxe pour récupérer les éléments du tableau :

- L'élément de rang 0 du tableau `tab` est accessible par :
  ```
  tab[0]
  ```
- L'élément de rang 1 du tableau `tab` est accessible par :
  ```
  tab[1]
  ```

En **JavaScript**, une propriété `prop` d'un objet `obj` peut être accessible via les deux instructions suivantes :

```
obj.prop
obj["prop"]
```

On peut utiliser cet astuce pour créer des tableaux associatifs en **JavaScript**: en rajoutant de nouvelles propriétés à un tableau vide, il est alors possible d'associer une valeur à un identifiant *(chaîne de caractères)*.
Le problème, alors, en **JavaScript** et qu'on ne sait pas réelement quelles sont les propriétés de l'objet qui doivent être considérées comme une valeur du tableau.

```
1 <script>
2 var a=new Array();
3 a["fr"]="Bonjour";
4 a["en"]="Hello";
5 a["es"]="Buenos dias";
6 document.write(a.length);
7 </script>
```

Le code suivant indique que la variable `c` a une longueur nulle.
Seul une boucle `for(x in c)...` permettra d'identifier toutes les clefs installés dans le tableau.

Les instructions suivantes :

```
1 <script>
2 var a=new Array(1,2,3);
3 a[90]=91;
4 document.write(a.length);
5 </script>
```

montrent que la variable `a` ainsi créée a une longueur de 91, même si en réalité ce tableau ne comprend que 4 éléments définits *(valeur distincte de* `undefined`*)*.

Constructeur :

Nous allons étudier le constructeur `Array()` de tableaux. Plus particulièrement, le type et le nombre d'argument qu'utilise ce constructeur.

- *aucun argument :* `var a=new Array();`
  Un tableau vide *(de longueur 0)* est affecté à la variable `a`.

- *un seul argument numérique :* `var b=new Array(5);`
  Un tableau de longueur 5 dont tous les éléments ont pour valeur `undefined` sera affecté à `b`.

- *un seul argument non-numérique ou plusieurs arguemnts de type quelconque :* `var c=new Array(1,"Bonjour","Au revoir");`
  Un tableau de trois éléments est créé.

Voici quelques exemples de création de tableau :

```
1  //Crée un tableau vide, puis affecte les valeurs
2  var tab1 = new Array();
3  tab1[0] = "premier";
4  tab1[1] = "second";
5
6  //Crée un tableau à deux éléments mais vide
7  var tab2 = new Array(2);
8  tab2[0] = "premier";
9  tab2[1] = "second";
10
11 //Crée le tableau avec ses éléments incorporés
12 var tab3 = new Array("premier","second");
13
14 //Crée un tableau à l'aide
15 //de la notation litérale
16 var tab4 = { 0 : "premier", 1 : "second" };
17
18 //Crée un tableau associatif
19 var tab5 = new Array();
20 tab5["a"] = "premier";
21 tab5["b"] = "second";
22
23 //Crée un tableau associatif
24 //à l'aide de la notation litérale
25 var tab6 = { "a" : "premier", "b" : "second" };
```

L'instruction suivante :

```
var tab=new Array(); tab[5]="bonjour";
```

créera un tableau contenant 6 éléments dont les cinq premiers ont pour valeur `undefined`.

Propriété :

`tab` représentera un objet de type `Array` :

number `tab.length`

La valeur de cette propriété représente le nombre d'éléments contenus dans le tableau.

Méthode :

array `tab.concat(`*array* `tab1,`*array* `tab2,...)`

On passe un ou plusieurs tableaux en arguments.
Cette méthode concaténe le tableau `tab` avec l'ensemble des tableaux passés en arguments : on obtient un tableau obtenu en mettant bout à bout tous ces tableaux. Le tableau ainsi obtenu sera retourné par la fonction.

string `tab.join(`*string* `str)`

Cette méthode convertit `tab` en chaîne de caractères en se servant de `str` comme séparateur des éléments.
Le résultat retourné est la chaîne de caractère ainsi formée.
`str` est optionnel, si une valeur n'est pas fourni, la virgule est utilisée

object `tab.pop()`

Après application de cette méthode, `tab` se voit retirer son dernier élément. Celui-ci est la valeur retournée.

number `tab.push(`*object* `obj1,`*object* `obj2,...)`

Cette méthode un ou plusieurs objets de nature quelconque en arguments.
Après exécution de cette méthode, `tab` se voit rajouter en fin de tableau les éléments passés en arguments.
La valeur retournée est le nombre d'éléments constituant `tab` après application de la méthode.

array `tab.reverse()`

Après exécution de cette méthode, `tab` verra l'ordre de tous ses éléments inversés *(le premier élément sera le dernier, ...)*.
Le résultat retourné est la nouvelle valeur de `tab`.

array `tab.shift()`

Après exécution, `tab` se voit retirer son premier élément.
Le résultat retourné sera l'élément retiré.

array `tab.slice(`*int* `num1,`*int* `num2)`

Cette méthode extrait une partie du tableau `tab`. Voici l'interprêtation des arguments :

- Si les deux paramètres sont positifs : La valeur retournée est le sous tableau constitué des éléments de `tab` d'ordre compris entre `num1`$^{\text{ième}}$ *(compris)* et `num2`$^{\text{ième}}$ *(non compris)*.
- Un argument négatif représentera l'élément en commençant par compter par la fin : si *num1* est négatif, il désignera l'élément de rang `tab.length+num1)`
- L'argument `num2` est optionnel *(a pour valeur* `undefined`*)* : en cas d'abscence, la méthode utilisera la longueur du tableau.
  Ainsi, `tab.slice(5)` retournera le sous tableau de l'élément de rang 5 jusqu'à la fin du tableau.

array `tab.sort(`*function* `compareFonction)`

Cette méthode permet d'ordonner les éléments d'un tableau.

- L'argument `compareFonction` est optionnel, de fait si sa valeur est `undefined`, ses éléments sont transformés en chaîne de caractères pour subir une comparaison *(comparaison naturelle des chaînes de caractères : elle porte successivement sur les premiers caractères en fonction de la table ASCII)*
- La fonction `compareFonction` est une fonction prennant deux arguments, de la forme `compareFonction(a,b)` et retourne :

- un nombre négatif pour indiquer que $a < b$
- zéro si $a == b$.
- un nombre positif dans le cas où $a > b$

Cette fonction crée, par les valeurs qu'elle retourne, l'ordre de comparaison.

Après application de cette méthode, la valeur de `tab` est modifiée et celle-ci est retournée par cette méthode.

array `tab.splice(`*int* `num1,`*int* `num2,`*obj* `item1,`*obj* `item2...)`

Seul les deux premiers éléments sont obligatoires.
Cette méthode aura pour effet d'ajouter et/ou d'enlever des éléments à partir de l'élément de rang `num1` *(compris)*.

- Si `num2` vaut 0, aucun élément ne sera enlevé.
  Si `num2` a une valeur positive, cette méthode supprimera `num2` à partir de l'élément de rang `num1`.
- `item1` et `item2` sont des arguments optionnels et seront rajoutés au tableau : `item1` au pour rang `num1`.

Le tableau `tab` est directement affecté par cette méthode : la valeur renvoyée est un tableau constitué par les éléments retirés du tableau *(dans le cas où* `num2` *est positif)*

number `tab.unshift(`*object* `obj1,`*object* `obj2,...)`

Cette méthode rajoute en début du tableau `tab`, les différents objets passés en paramètre.
Cette méthode retourne le nombre d'éléments composant ce tableau.

E. String :

L'objet `String` appartenant au noyau de **JavaScript**, sert de modèle à toutes les variables de type "*chaîne de caractères*".

De même que pour les tableaux, le premier élément d'une chaîne de caractère est le caractère de rang 0.

Constructeur :

On déclare une variable de type "*chaîne de caractères*" de la manière suivante :

```
var str = new String();
```

La variable `str` contiend une chaîne vide.

On peut également affecter en même temps une valeur à la variable :

```
var str = new String("Bonjour");
```

Si le paramètre passé au constructeur n'est pas une chaîne de caractère, **JavaScript** se charge de la conversion :

```
var str = new String(true);
```

Propriété :

number `str.length`

Cette propriété contient le nombre de caractères qui compose la chaîne `str`.

Méthode :

string `str.charAt(`*int* `num)`

Cette méthode renvoie le caractère de rang `num` dans la chaîne `str`.

Number `str.charCodeAt(`*int* `num)`

Cette méthode renverra le numéro correspondant au caractère de rang `num` *(relative-*

ment à la table ASCII et UTF-8).
Si aucun caractère n'est présent à cette position, cette méthode renverra la valeur `NaN`.

string `str.concat(`*string* `str1,` *string* `str2,` *string* `str3...)`

Cette méthode prend au minimun un paramètre ; les autres sont optionnels.
Cette méthode renvoit la nouvelle chaîne obtenue en mettant `str` et les chaînes passées en arguments. La valeur de `str` ne sera pas modifiée. On obtient le même résultat à partir de l'opérateur "+" :

```
str+=str1;
```

number `str.indexOf(`*string* `str1,` *int* `pos)`

Cette méthode cherche la première occurence de la chaîne `str1` dans la chaîne `str`.
L'argument `pos` est optionnel, si il est fourni, la recherche s'effectuera à partir du du rang `pos`.
Cette méthode renvoit la position de la première de ces occurences ou -1 si celle-ci n'est pas trouvée.

number `str.lastIndexOf(`*string* `str1,` *int* `pos)`

Cette méthode recherche la dernière occurence `str1` présente dans la chaîne `str`.
L'argument `pos` est optionnel, si sa valeur est différente de `undefined` la recherche de cette occurence s'effectuera sur la sous-chaîne de `str` formée par les caractères de rang 0 au rang `pos` *(compris)*.
Cette méthode renverra la position de la dernière occurence trouvée ou -1 en cas d'échec.

number `str.localeCompare(`*string* `str1)`

Cette méthode compare la chaîne de caractères `str` avec `str1`.
Cette méthode renvoit :

- -1 si `str` est plus petit que codestr1 ;
- 0 si `str` est identique à `str` ;
- -1 si `str` est plus grand que `str1`.

Cette méthode prend en compte l'ordre des caract'eres défini dans le système d'exploitation du client.

string `str.slice(`*int* `num1,`*int* `num2)`

La méthode `slice()` retourne la partie de `str` comprise du caractère de rang `num1`

jusqu'au caractère de rang `num2` *(ce dernier caractère non-compris)*. Voici quelques précisions quant à ses arguments :

- Si `num1` ou `num2` sont positifs, les caractères désignés seront les caractères de rang correspondant.
- Si `num1` ou `num2` sont négatifs, la désigniation des caractères se fera à partir de la fin de la chaîne : -1 désigne le dernier caractère, -2 l'avant dernier...
- Si l'argument `num2` n'est pas fourni ou a une valeur `undefined`, la valeur par défaut sera la longueur de la chaîne *(allant ainsi jusqu'au bout de la chaîne)*.

array `str.split(`*expreg* `exp, `*int* `num)`

Cette méthode utilise l'expression rationnelle `exp`, comme séparateur, pour découper la chaîne `str` en plusieurs parties.
Elle renverra ces différentes parties dans un tableau.
L'argument `num` est optionnel et permet de limiter le nombre de parties retournées dans le tableau de sortie.

Une chaîne de caractère peut être considérée comme une expression rationnelle: plus simplement `str.split(",")` retourne dans un tableau la chaîne `str` découpée à chaque emplacement d'une virgule.

string `str.substr(`*int* `num1, `*int* `num2)`

Cette méthode renvoit la sous-chaîne de `str` commençant au caractère de rang `num1` et ayant une longueur de `num2`.
Si `num1` est négatif, le caractère de départ est identifié à partir de la fin de la chaîne *(-1 représente le dernier caractère, -2 l'avant dernier...)*.
Une valeur nulle ou négative de `num2` renverra la chaîne vide.

string `str.substring(`*int* `num1,`*int* `num2)`

Cette méthode retourne la sous-chaîne de `str` comprise entre le caractère de rang `num1` *(compris)* et de rang `num2` *(non-compris)*.

Voici quelques remarques quant aux valeurs des arguments :

- Si `num1` a pour valeur `undefined`, il sera remplacé par 0.
  Si `num2` a pour valeur `undefined`, il sera remplacé par la longueur de la chaîne.
- Si un des arguments est négatif ou a une valeur de `NaN`, il sera remplacé par 0.
- Si `num1`>`num2`, alors ses deux arguments seront intervertis.

string `str.toLowerCase()`

Cette méthode retourne la chaîne `str` où tous ces caractères ont été transformés en minuscule.

string `str.toUpperCase()`

Cette méthode retournera la chaîne `str` où tous ces caractères ont été transformés en majuscule.

string `str.toLocaleUpperCase(`*string* `str)`

Cette méthode est identique à `toUpperCase`, elle permet d'adapter son fonctionnement au particularité régionale du client : si les règles sont différentes des règles de casses définies par Unicode *(langue turque par exemple).*

string `str.toLocaleLowerCase(`*string* `str)`

Cette méthode est identique à `toLowerCase`, elle permet d'adapter son fonctionnement au particularité régionale du client.

Les méthodes suivantes utilisent les expressions régulières. Un chapitre est prévu à cet effet ; reportez vous à celui-ci pour de plus ample information. Seul sera énoncé ici les méthodes des objets de type *String* utilisant en argument des expressions rationnelles :

array `str.match(`*expreg* `exp)`

La méthode `match()` recherche l'expression régulière `exp` dans la chaîne `str`. Elle retournera, dans un tableau, la ou les occurences *(dans le cas d'une expression globale)* de l'expression régulière présentes dans `str`.
Si aucune occurence n'est pas trouvée, cette méthode renvoit la valeur `null`

string `str.replace(`*expreg* `exp,`*string* `str2)`

Cette méthode cherche dans `str` la première ou toutes les parties de `str` *(dépendant si* `exp` *est définie avec le drapeau "g")* validant l'expression `exp` et retournera une nouvelle chaîne où toutes ces occurences auront été remplacées par `str2`.

Il est possible de contrôler la valeur de `str2` en fonction de l'occurence de `exp` trouvée dans `str` à l'aide des séquences de remplacement suivante :

Séquence	Chaîne de remplacement
$$	$
$&	la partie validant l'expression `exp`
$´	La partie de `str` suivant la partie validant `exp`
$`	La partie de `str` suivant la partie validant `exp`
$n	La $n^{\text{ième}}$ partie validant `exp` si elle existe *(sinon représente la chaîne vide)* où n est un entier à un chiffre.
$nn	La $nn^{\text{ième}}$ partie validant `exp` si elle existe *(sinon représente la chaîne vide)* où n est un entier à deux chiffres.

Une autre forme de cette méthode est également possible :

`str.replace(`*expreg* `exp,`*function* `fct)`

La fonction `fct()` recevra un argument qui est l'occurence de `str` vérifiant `exp` et retournera pour valeur la chaîne de caractères de remplacement.

number `str.search(`*expreg* `exp)`

Cette méthode cherche la première sous-chaîne de `str` satisfaisant l'expression régulière `exp` et renvoit le rang du premier caractère de cette sous-chaîne.
Si aucune occurence de `exp` n'est trouvée, cette méthode renverra -1.

F. Boolean :

L'algèbre de Boole *(mathématicien britannique du XIX^e siècle)* est l'utilisation de techniques algébriques sur un ensemble à deux valeurs : ces deux valeurs représentées souvent par 0 et 1 représentent le status "*vrai*" et "*faux*".

JavaScript propose l'objet `Boolean` pour représenter cet ensemble. Cet objet peut prendre deux valeurs `false` et `true`. Cet objet ne propose :

- pour propriété : que sa valeur qui le caractérise.

- pour méthode : que celles héritées de l'objet `Object`, c'est à dire :

  `toString()` `valueOf()`

On manipulera cet objet principalement à l'aide des opérateurs logiques lors de l'utilisation de structure de contôles.

G. Number :

L'objet `Number` représente le modèle de tous les nombres en **JavaScript**.

Propriété :

Les propriétés suivantes ne font pas parties du prototype de l'objet `Number` : autrement dit, elles n'appartiennent qu'à l'objet `Number` lui-même et pas à ses instances.

`Number.MAX_VALUE`

Cette propriété a pour valeur $1{,}7976931348623157 \times 10^{308}$. C'est la plus grande valeur acceptée par **JavaScript**.

`Number.MIN_VALUE`

La valeur de cette propriété est 5×10^{-324} : la plus petite valeur numérique de **JavaScript**.

`Number.NaN`

Le résultat retourné est `NaN`

`Number.NEGATIVE_INFINITY`

Le résultat retourné est `-Infinity` représentant la valeur $-\infty$.

`Number.POSITIVE_INFINITY`

Le résultat retourné est `+Infinity` représentant la valeur $+\infty$.

Méthode :

`num` représente un objet de type *number*

string `num.toString(`*int* `num2)`

`num2` est un entier compris entre 2 et 36.
Cette méthode renvoie une chaîne de caractères représentant la valeur `num` dans la base `num2`.
L'argument `num2` est optionnel, s'il n'est pas précisé la valeur 10 sera utilisée *(base décimale)*.

number `num.toFixed(`*int* `num2)`

`num2` est un nombre entier compris entre 0 et 20.

Cette méthode retourne une chaîne de caractères représentant `num` sans utiliser la notation exponentielle avec exactement `num2` chiffres après la virgule.
Dans le cas, où `num` est plus grand que 2×10^{21} *(ne pouvant pas alors représenter ce nombre sans utiliser la notation exponentielle)*, cette méthode appelle directement `toString()`.

string `num.toExponential(`*int* `num2)`

`num2` est optionnel et représente un entier compris entre 0 et 20.
Cette méthode renvoie l'écriture scientifique du nombre `num` avec `num2` chiffres après la virgules.

string `num.toPrecision(`*int* `num2)`

`num2` est optionnel et représente un entier compris entre 1 et 21.
Cette méthode retourne une chaîne de caractère représentant la valeur `num` avec `num2` chiffres significatifs ; la méthode passe en écriture scientifique si nécessaire.

H. Date :

Beaucoup de systèmes informatiques, dont **JavaScript**, mesure le temps en millisecondes en prenant comme date de référence le 1er janvier 1970 en heure UTC. Ainsi,

- Le 1er janvier 1970 est repéré par le nombre 0.
- Le 2 Janvier 1970 est associé à :
  $24\times3600\times1000 = 86\,400\,000$
- Le 19 Mars 2008 à minuit a une date informatique de :
  1 211 097 600 000

L'heure UTC *(temps universel coordonné)* est une échelle de temps utilisé par un grand nombre de pays prenant en compte les variations de la rotation de la terre sur elle-même alors que l'heure atomique ne subit aucune influence. Elle remplace l'heure GMT.

Dans les méthodes présentées ci-dessous, pour la plupart, il existe une version prenant en compte les changements locals de l'heure *(heure d'été...)* et une autre donnant l'heure UTC sans prise en compte du système d'exploitation du client.

La plupart des méthodes présentées dans ce chapitre utilisent le même type d'arguments. On utilisera pour une identification plus rapide les notations suivantes :

- `a` représente l'année ;
- `m` représente le mois et peut prendre une valeur de 0 à 11 ;
- `j` représente le jours et peut prendre une valeur de 1 à 31 ;

- `h` représente l'heure et peut prendre une valeur de 0 à 23 ;
- `m` représente les minutes et peut prendre une valeur de 0 à 59 ;
- `s` représente les secondes et peut prendre une valeur de 0 à 59 ;
- `ms` représente les milli-secondes et peut prendre une valeur de 0 à 999 ;

Si une valeur en dehors de l'intervalle précisé est fournie, alors les valeurs adjacentes seront influencées :

- Si `h` a une valeur de 26, alors `j` sera incrémenté d'un et `h` aura la valeur considérée de 2.
- Si `h` a une valeur de -1, alors `j` représentera le jour précédent et `h` aura la valeur 23.

Constructeur :

date `new Date(`*int* `a,`*int* `m,`*int* `j,`*int* `h,`*int* `m,`*int* `s,`*int* `m)`

L'objet retourné est la date représentée par les arguments passés au constructeur *(relativement au système local)*.
Seul les trois premiers arguments sont obligatoires, les autres seront remplacés par 0 en cas d'abscence. Par exemple, le code :

```
var a=new Date(2008,08,22)
```

déclare et affecte à la variable `a`, la date du 22 Septembre 2008 à minuit précise.

date `new Date(`*int* `num)`

L'objet retourné par le constructeur sera la date obtenue `num` milli-secondes après le 1er Janvier 1970.
Si `num` n'est pas fourni alors la date renvoyée est la date actuelle du système du client.

date `new Date(`*string* `str)`

Le constructeur renverra la date correspondant à la chaîne `str`.
La chaîne doit pouvoir être interprêtée par la méthode `Date.parse()`.

Méthode :

Les trois méthodes ci-dessous ne font pas partie des objets instanciés.
Elles n'appartiennent qu'à l'objet `Date`.

number `Date.parse(`*string* `str)`

`str` représente, pour **JavaScript**, une date ; en voici deux exemples :

- `"Wed Mar 19 2008"`
- `"1 January 2005 00:00:01 UTC"`

Cette méthode transforme la chaîne de caractère passée en argument en nombre entier représentant cette date *(le nombre de milli-secondes écoulées depuis le* 1^{er} *septembre 1970)*.
La valeur `NaN` sera renvoyée si `str` ne pourra pas être interprêté comme une date.

number `Date.UTC(`*int* `a,`*int* `m,`*int* `j,`*int* `h,`*int* `m,`*int* `s)`

La méthode `Date.UTC()` retourne le nombre de milli-secondes caractérisant la date fournie en arguments dans le système UTC universel de temps.
Tous les arguments sont optionnels sauf `a` et `m`, les arguments non-fournis sont remplacés par 0.

La suite des méthodes présentées appartiennent aux instances de l'objet `Date`.

string `date.toString()`

Cette méthode renvoie une chaîne de caractères, lisible humainement, représentant la date de la variable `date`.

La méthode `date.toLocaleString()` renvoie une chaîne de caractères en prenant compte du système d'exploitation du client *(notamment la langue, l'écriture d'une date...)*

La méthode `date.toUTCString()` renvoie une chaîne de caractères représentant `date` dans l'échelle de temps UTC.

string `date.toDateString()`

Identique à `toString()`, mais la chaîne retournée ne contient que la date *(jour, mois, année)* représentant `date`.

A noter la m{ethode `toLocaleDateString()` renvoyant ces informations en tenant compte du système du client.

string `date.toTimeString()`

Identique à `toString()`, mais la chaîne retournée ne contient que l'heure exprimée par `date`.

La méthode `toLocaleTimeString()` prend en compte les spécifications locales du système d'exploitation du client.

int `date.valueOf()`

Cette méthode renvoie le nombre entier caractérisant `date`.

Rappelons que ce nombre représente le nombre de milli-secondes écoulés depuis le 1^{er} janvier 1970.

int `date.getTime()`

Identique à `valueOf()` mais renvoie une erreur dans le cas où `date` n'est pas de type `Date`.

int `date.getFullYear()`

Cette méthode renvoie l'année représentée par `date` sur quatre chiffres.

La méthode `date.getUTCFullYear()` renvoie l'année représentée par `date` dans l'échelle de temps UTC

int `date.getMonth()`

Cette méthode renvoie le mois, de 0 à 11, représentée par `date`.

La méthode `date.getUTCMonth()` utilisant l'échelle de temps UTC

int `date.getDate()`

Cette méthode renvoie le jour du mois *(de 1 à 31)* contenu dans la date `date` en heure locale.
`date.getUTCDate()` est équivalente mais utilisera l'échelle de temps UTC.

int `date.getDay()`

Cette méthode renvoie le jour de la semaine exprimée par `date` sous la forme d'un nombre entier de 0 à 6 où 0 représente le dimande.

Pour utiliser l'échelle UTC, on utilise la méthode `date.getUTCDay()`.

int `date.getHours()`

Cette méthode retourne l'heure exprimée par `date` : un entier de 0 à 23.

La méthode `getUTCHours()` utilise l'échelle de temps UTC.

int `date.getMinutes()`

Cette méthode renvoie les minutes exprimées dans la date `date` : un entier compris entre 0 et 59.

Pour l'échelle de temps UTC, utiliser la méthode `date.getUTCMinutes()`.

int `date.getSeconds()`

Cette méthode renvoie les secondes exprimées dans `date` : entier compris entre 0 et 59.

Pour l'échelle de temps UTC, utiliser `date.getUTCSeconds()`

int `date.getMilliseconds()`

Cette méthode renvoie les millisecondes exprimées dans `date` : entier compris entre 0 et 999.

Pour l'échelle de temps UTC, utiliser `date.getUTCMilliseconds()`.

int `date.getTimezoneOffset()`

Le résultat retourné est la différence, exprimée en minutes, entre l'heure locale et l'heure UTC.

number `date.setTime(`*date* `date2)`

Cette méthode affecte à la variable `date` la valeur `date2` et renvoie l'entier caractérisant la nouvelle valeur de `date` *(c'est à dire* `date2.valueOf()`*)*.

Les méthode ci-dessous permettent de modifier une partie d'une date : son nombre de minutes, de secondes, le jour du mois...
Attention de prendre en compte le dépassement de l'intervalle de valeur pour chacune des valeurs d'une date : ceci aura pour effet de modifier les autres valeurs adjacentes de la date *(voir note en début de paragraphe.)*

date `date.setMilliseconds(`*int* `ms)`

`ms` est un entier.
Cette méthode fixe le nombre de millisecondes exprimées dans `date` à `ms` et renvoie l'entier représentant la nouvelle valeur de `date`.

La méthode `setUTCMillisecond()` permet d'utiliser l'échelle de temps UTC.

date `date.setSeconds(`*int* `s,`*int* `ms)`

Seul `s` est obligatoire.
Cette méthode modife le nombre de secondes exprimées par `date`. Le résultat retourné sera l'entier représentant la nouvelle valeur de `date`.

`date.setUTCSeconds()` permet d'utiliser l'échelle de temps UTC.

date `date.setMinutes(`*int* `min,`*int* `s,`*int* `ms)`

Seul `min` est obligatoire.
Cette méthode modifie le nombre de minutes exprimées par `date` et retourne l'entier représentant cette nouvelle valeur.

La méthode `setUTCMinutes()` utilise l'échelle de temps UTC.

date `date.setHours(`*int* `h,`*int* `min,`*int* `s,`*int* `ms)`

Seul `h` est obligatoire.
Cette méthode modifie le nombre d'heures exprimées par `date` et elle retourne l'entier caractérisant cette nouvelle valeur.

La méthode `date.setUTCHours()` utilise l'échelle de temps UTC.

date `date.setDate(`*int* `j)`

Cette méthode modifie le jour du mois de `date`, mais ne modifiera pas les heures, minutes....
Elle renvoie l'entier caractérisant la nouvelle valeur de `date`

La méthode `date.setUTCDate()` utilise, quant à elle, l'échelle de temps UTC.

date `date.setMonth(`*int* `m,`*int* `j)`

Seul `m` est obligatoire.
Cette méthode modifie le mois exprimé par `date` et retourne l'entier caractérisant la nouvelle valeur de `date`.

La méthode `date.setUTCMonth()` utilise l'échelle de temps UTC.

date `date.setFullYear(`*int* `a,`*int* `m,`*int* `j)`

Seul `annee` est obligatoire.
Cette méthode modifie l'année de `date` et renvoit l'entier représentant `date`.

`date.setUTCFullYear()` utilise l'échelle de temps UTC.

I. RegExp :

Les expressions régulières *(également appelées expressions rationnelles)* sont des outils présents, actuellement, dans tous les langages de programmation. Ils permettent d'explorer une chaîne de caractères à la recherche d'un motif et non pas à la recherche d'une sous-chaîne précise :

- rechercher dix chiffres consécutifs dans une chaîne de caractère permet d'extraire tous les

numéros de téléphone de celle-ci ;

- un mail suit des règles syntaxiques : les expressions régulières permettent de vérifier si une chaîne de caractères représente une adresse électronique.

⚠ Nous avons vu au chapitre sur les chaînes de caractères, les méthodes `str.match()`, `str.replace()`, `str.search()` qui prennent en arguments des expressions réguliGlign ères. L'expression régulière peut se réduire à une chaîne de caractères au quel cas cette chaîne sera considérer comme une expression régulière ayant les caractéristiques suivantes :

- Cette expression représente exactement la chaîne de caractères ;
- Elle est sensible à la casse ;
- La recherche sera non-globale *(voir plus loin)*

Une expression régulière est composée de deux parties :

- Le motif représentant "*la forme*" de la sous-chaîne recherchée.
- Le drapeau exprime le comportement de l'expression régulière et du type de recherche associée : sensible à la casse ou non, recherchant la première occurence du motif ou toutes les occurences...

Les motifs :

Pour exprimer un motif, certains caractères *(^, $, *, ...)* sont utilisés pour représenter une position spéciale dans la chaîne de caractère. Des caractères d'échappement seront utilisés pour les intégrer au motif.

Marquer le début et fin de chaîne :

- Le caractère ^ permet d'indiquer le début de la chaîne de caractères.
- Le signe dollar $ indique la fin de la chaîne de caractère

`abc`	Ce motif valide les chaînes de caractères contenant la sous-chaîne `abc`
`^abc`	Valide les chaînes començant par `a` suivi de `bc`
`abc$`	Valide les chaînes de caractères finissant par `c` juste précédée par `ab`
`^abc$`	Valide les chaînes de caractères commençant et finissant par `abc`

Indiquer la répétition de caractères :

Les expressions régulières nous permettent de préciser combien de fois un caractère *(nous verrons plus tard comment indiquer un groupe de caractères)* doit se répéter dans une chaîne

de caractères.

`abc*`	Valide les chaînes contenant la sous-chaîne `ab` suivie de zéro ou plusieurs fois le caractère `c`.
`abc?`	Valide une chaîne contenant la sous-chaîne `ab` suivie de zéro ou d'une seule fois le caractère `c`.
`abc+`	Valide une chaîne contenant la sous-chaîne `ab` suivie d'une ou plusieurs fois le caractère `c`.
`abc{2}`	Valide une chaîne contenant la sous-chaîne `ab` suivie d'exactement deux caractères `c` : c'est à dire contenant `abcc`.
`abc{2,}`	Valide une chaîne contenant la sous-chaîne `ab` suivie de deux ou plusieurs caractères `c`.
`abc{2,5}`	Valide une chaîne contenant la sous-chaîne `ab` suivie de deux à cinq caractères `c`.

Le choix à l'alternative :

Le caractère | représente l'opérateur logique OU :

`(ab|cd)`	Désigne les chaîne de caractères contenant la séquence `ab` ou `cd`

Représenter une séquence de caractères :

Les parenthèses permettent de regrouper un ou plusieurs caractères :

`ab(cd){3}`	Désigne la chaîne de caractères qui contient la séquence `ab` suivi directement de trois fois la séquence `cd`

Capturer des parties du motifs :

La méthode `exec()` permet de récupérer les différentes parties d'une chaîne de caractères vérifiant le motif ou une partie du motif. Pour cela, lors de la définition du motif, on entoure les parties qu'on souhaite capturer à l'aide de parenthèses.

Si on souhaite représenter dans le motif une séquence de caractères, mais que celle-ci ne doit pas être capturée par la méthode `exec`, il faut respecter la séquence suivante :

(?:xy)	Ceci représente la séquence `xy` mais ne sera pas capturé par la méthode `exec()`

La présence conditionnelle :

Les parenthèses peuvent également servir à exprimer la condition qu'une séquence est suivie ou non par une autre séquence :

x(?=y)	Valide toute chaîne contenant `x` si celui-ci est directement suivi de `y`
x(?!=y)	Valide toute chaîne contenant `x` si celui-ci n'est pas suivi de `y`

Le point

Le point " . " représente tous les caractères sauf ceux représentant les sauts de lignes : \n, \r, \u2028, \u2029.

`ab.$`	Valide les chaînes de caractères se terminant par `ab` suivi d'un troisième caracère quelconque *(autre qu'un saut de ligne)*.

On utilisera [*backslash*`s`\S] pour représenter tous les caractères.

Les listes de caractères :

Les crochets permettent de représenter un caractère de la chaîne ayant pour valeur une "plage" ou un ensemble de caractères.

`[-bc?]`	Représente dans une expression régulière au moins un des caractères suivants : "`-`", "`b`" ,"`c`" ou "`?`".
`[a-z]`	Représente toutes lettres en minuscules.
`[a-g]`	Représente toutes les lettres en minuscules comprises entre `a` et `g` compris.
`[0-9]`	Représente n'importe quel chiffre.
`[a-zA-Z0-9]`	Représente n'importe quel caractère alpha-numérique.

"`-`" a une valeur différente s'il se trouve en début ou fin des crochets ou entre deux caractères : il définit alors un intervalle de lettres.

Une liste commeçant par ^ indique une négation des lettres acceptés :

`[^0-9]`	représente tous les caractères non-numériques
`[^a-z]`	représente tous caractères sauf les lettres de l'aphabet en minuscules.

Les caractères spéciaux :

\b	Représente un caractère d'espacement tel qu'un espace, une tabulation, un saut de ligne... Ainsi, l'expression régulière \bch validera les chaînes "charme", "bonjour\rcharles" mais pas "beach".
\B	A l'inverse, ce caractère spécial représente tous les caractères autres que ceux d'espacements.
\d	Désigne un caractère numérique décimal. \d{5} est équivalent à [0-9]{5} Cette expression représente un code postal.
\	Désigne tous les caractères qui ne sont pas des chiffres décimaux.
\f	Désigne un saut de page.
\n	Désigne un saut de ligne.
\r	Désigne un retour de chariot
\s	Désigne tous les caractères blancs : espace, tabulation...
\S	A l'opposé désigne tous les caractères non-blancs.
\t	Désigne une tabulation horizontale
\v	Désigne une tabulation verticale
\w	Désigne tous les caractères alphanumériques et le caractère _. \w est équivalent à [a-zA-Z0-9_]
\W	Désigne tous les caractères non-alphanumériques. \W est équivalent à [^a-zA-Z0-9_]
\oXXX	représente le nombre XXX écrit en base octale.
\xXXX	Représente le nombre XXX écrit en base héxadécimale.

En utilisant les caractères spéciaux dans une chaîne, penser à doubler vos barres contre-oblique : elles sont utilisés dans les chaînes pour les caractères d'échappement.

Les drapeaux :

Par défaut, une expression a certains comportements :

- Elle différencie les caractères majuscules des miniscules.
- Avec la méthode exec(), l'expression régulière ne gardera pas en mémoire les dernières recherches : seul la première occurence sera accessible.

On modifie ces comportements en modifiant les drapeaux :

`g`	Permet en lançant successivement la méthode `exec()` d'obtenir toutes les parties de la chaîne représentant le motif de l'expression régulière.
`i`	Effectue une recherche sans que soit tenu compte de la différence majuscule/minuscule des caractères de la chaîne *(insensible à la casse)*.
`m`	L'expression régulière effectuera une recherche "multiligne". Le motif "`^a`" cherche le caractère `a` en début de ligne : la chaîne `"b\na"` sera alors validée par cette expression régulière seulement si le drapeau `m` est présent.

Les drapeaux présentés ci-dessus peuvent être ordonnées, sans souci d'ordre :

`gi` donnera une recherche globale et insensible à la casse.

Constructeur :

Voici deux méthodes pour déclarer et définir une variable `expr` dont le type est une expression régulière : une instance de la classe `RegExp`

- A l'aide de la notation simplifiée :

```
var expr=/motif/drapeau;
```

- En utilisant le constructeur de la classe d'objet `expreg` :

```
var expr=new RegExp("motif","drapeau");
```

Propriété :

Toutes les propriétés suivantes, sauf la dernière, ont l'attribut `ReadOnly` : ainsi, une fois créé, il est impossible de modifier le comportement d'une expression régulière. Il faut alors la recréer.

string `expReg.source`

La valeur retournée est une chaîne de caractère représentant le motif de l'expression régulière `expReg`.

boolean `expReg.global`

Cette propriété vaut `true` si le drapeau de `expReg` contient la lettre "`g`" et vaut `false` dans le cas contraire.

boolean `expReg.ignoreCase`

Cette propriété vaut `true` si le drapeau de `expReg` contient la lettre "`i`" et vaut `false` dans le cas contraire.

boolean `expReg.multiline`

Cette propriété vaut `true` si le drapeau de `expReg` contient la lettre "`m`" et vaut `false` dans le cas contraire.

number `expReg.lastIndex`

Dans le cas d'une recherche globale, cette propriété contiendra la position à laquelle commencera la prochaine recherche *(avec la méthode* `exec()` *)*.

Dans le cas d'une recherche non-globale, cette propriété vaut toujours 0.

Cette propriété peut être modifiée après création de `expReg` permettant ainsi de modifier le début de la recherche dans une chaîne.

Méthode :

array `expReg.exec(`*string* `chaine)`

Cette méthode renvoie `null` si une partie de `chaine` n'est pas validé par `expReg`. Dans le cas contraire, un tableau est renvoyé. Voici sa composition :

- L'élément de rang 0 est la sous-chaîne vérifiant `expReg`.
- L'élément de rang 1 est la partie de `chaine` vérifiant les premières parenthèses capturantes du motif de `expReg`.
- Lélément de rang 2 est la partie correspondante aux secondes parenthèses capturantes de `expReg`.

Ainsi, l'expression régulière `/(\d{2})-(\d{3})/` utilisée sur la chaîne `"23-841"` renverra le tableau :

```
Array("23-841","23","841")
```

Lors de sa première exécution, cette méthode s'arrêtera sur la première partie de la chaîne vérifiant le motif. Il faut que le drapeau de cette expression possède la propriété globale et exécuter plusieurs fois cette méthode afin d'explorer toutes les parties de `chaine` vérifiant l'expression *(jusqu'à ce que la valeur* `null` *apparaîsse)*.

boolean `expReg.test(`*string* `chaine)`

Cette méthode renvoie `true` si `chaine` vérifie l'expression régulière `expReg` et renverra `false` dans le cas contraire.

L'objet `RegExp` possèdait des propriétés qui ont été dépréciées dans les dernières version de **JavaScript**:

- `global`, `ignoreCase`, `lastIndex`, `multiline`, `source` sont devenus des propriétés des instances de cet objet.

- `input`, `lastMatch`, `lastParen`, `leftContext`, `rightContext` sont des propriétés de l'objet `RegExp` et ne sont plus documentés dans les dernières documentations.

  La méthode `compile()` a, elle aussi, été abandonnée

J. Error :

JavaScript permet la gestion des erreurs : plus précisemment, on peut anticiper la survenue d'erreur dans le code.

Nous ne traiterons pas cette partie dans cette formation car elle est peut utilisée. Elle devient utile lorsque le client peut entrer des valeurs utilisables par le code ou lorsqu'un script est utilisé dans différentes pages.

Voici les différents types d'erreurs que nous pouvons intercepter :

- Toutes les erreurs : classe "error"
- Les erreurs de la fonction primitive eval() : classe "EvalError"
- Les erreurs de dépassement des capacités numériques : classe "RangeError"
- Les erreurs de référencement invalide : classe "ReferenceError"
- Les erreurs de syntaxe d'écriture : classe "SyntaxError"
- Les erreurs de type d'argument fourni aux méthode et fonctions : classe "TypeError"

Voici un exemple de code permettant la gestion d'erreurs. Ce code essaye de parcourir l'élément `document.images` contenant toutes les images de la page.

Si le code compris dans `try{...}` s'exécute avec une erreur, alors la partie contenue dans `catch{...}` s'exécutera pour gérer l'erreur.

```
1  <script>
2   try{
3    for(i=0;i<document.images.length + 1;i++){
4     alert('Largeur de l\'image numéro '+i+' : '+document.↩
         images[i].width+' pixels');
5    }
6    document.writeln("Tout s'est déroulé normalement.")
7   }
8
9   catch (err){
10   document.writeln("Une exception a eu lieu !<br>");
```

```
11    document.writeln("Nom de l'exception : " + err.name+"<br>");
12    document.writeln("Message d'erreur reçu : " + err.message+"<br>");
13  }
14
15  finally{
16    document.writeln("<p>Cette partie s'exécute toujours</p>");
17  }
18
19  document.writeln("Fin du script");
20 </script>
```

K. Math :

L'objet `Math` a pour propriétés la plupart des constantes importantes mathématiques et ses méthodes sont les fonctions mathématiques usuelles.

Cet objet ne possède pas de constructeur : il n'est pas possible de créer une copie de l'objet `Math` *(une instance de* `Math`*)*. Ainsi, on utilise directement l'objet `Math`.

Propriété :

number `Math.E`

Le résultat retourné est le nombre e qui est la base du l'exponenetiel népérien

LN10 `Math.LN10`

Le résultat retourné est la valeur du logarithme népérien de 10.

number `Math.LN2`

Le résultat retourné est la valeur du logarithme népérien de 2

number `Math.LOG2E`

Le résultat retourné est la valeur de l'image de e par le logarithme de base 2.

number `Math.LOG10E`

Le résultat retourné est la valeur de l'image de e par le logarithme de base 10

number `Math.PI`

Le résultat retourné est la valeur de π.

number `Math.SQRT1_2`

Le résultat retourné est la racine carré de $\frac{1}{2}$.

number `Math.SQRT2`

Le résultat retourné est la racine carré de 2.

Méthode :

number `Math.abs(`*number* `x)`

Le résultat retourné est la valeur absolue du nombre `x`.

number `Math.acos(`*number* `x)`

Le résultat retourné est la valeur du cosinus inverse de `x`. Il sera retourné un nombre compris entre 0 et π

number `Math.asin(`*number* `x)`

Le résultat retourné est la valeur du sinus invers de `x`. Il sera retourné un nombre compris entre $-\frac{\pi}{2}$ et $\frac{\pi}{2}$

number `Math.atan(`*number* `x)`

Le résultat retourné est la valeur de la tangente inverse de `x`. Il sera retouné un nombre compris entre $-\frac{\pi}{2}$ et $\frac{\pi}{2}$

number `Math.atan2(`*number* `y,`*number* `x)`

Le résultat retourné est sera la tangente inverse du quotient $\frac{y}{x}$ et sera compris entre $-\pi$ et π. Le signe du résultat dépendra du quadran où se trouve le point de coordonné $(\,x\,;\,y\,)$

number `Math.ceil(`*number* `x)`

Le résultat retourné sera l'entier supérieur et le plus proche de `x` *(c'est à dire le plus petit des entiers supérieur à x).*

number `Math.cos(`*number* `x)`

Le résultat retourné est le cosinus du nombre `x`

number `Math.exp(`*number* `x)`

Le résultat retourné est l'image de `x` par la fonction exponentielle.

number `Math.floor(`*number* `x)`

Le résultat retourné est l'entier inférieur et le plus proche de `x` *(c'est à dire le plus grand des entiers inférieurs à x)*

number `Math.log(`*number* `x)`

Le résultat retourné est le logarithme népérien de `x`.

number `Math.max(`*number* `x1,`*number* `x2,...)`

Le résultat retourné est le plus grand des nombres `x1`, `x2`, ...
Si aucun argument n'est fourni, la méthode retournera `-Infinity`

number `Math.min(`*number* `x1,`*number* `x2,...)`

Le résultat retourné est le plus petit des nombres `x1`,`x2`,....
Si aucun argument n'est fourni, la méthode retournera `Infinity`

number `Math.pow(`*number* `x,`*number* `y)`

Le résultat retourné sera la valeur de x à la puissance d'exposant y : le résultat de x^y

number `Math.random()`

Le résultat retourné est un nombre plus grand ou égal à 0 mais strictement plus petit que 1.

number `Math.round(`*number* `x)`

Le résultat retourné est le nombre entier le plus proche de x.
Pour 1,5, cette méthode retournera 2 et avec l'argument $-1{,}5$ renverra -1.

number `Math.sin(`*number* `x)`

Le résultat retourné est le sinus de `x`.

number `Math.sqrt(`*number* `x)`

Le résultat retourné est la racine carré de `x`.

number `Math.tan(`*number* `x)`

Le résultat retourné est la tangente de `x`.

Index