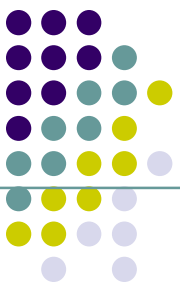


Le langage JavaScript

Références



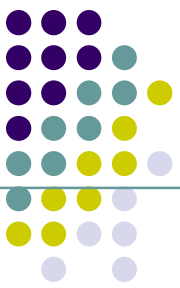
1. World Wide Web Consortium (W3C) : w3.org
2. World Wide Web School : w3schools.com
3. www.developpez.com
4. Youtube.com
5. Faire des recherches sur internet



Sommaire

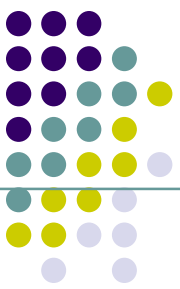
- | | |
|--|---|
| <ul style="list-style-type: none">• Introduction• Utilisation• Méthodes et propriété d’affichage• La syntaxe de JavaScript• Les variables• Les opérateurs• Les types de données• Les fonctions• Les objets | <ul style="list-style-type: none">• Les objets du navigateur• Structures conditionnelles et de contrôle• Nommage des objets• Manipulation des objets• Les évènements• Quelques exemples• Les expressions régulières• Gestion des erreurs• Le débogage en JavaScript |
|--|---|

Introduction



Historique

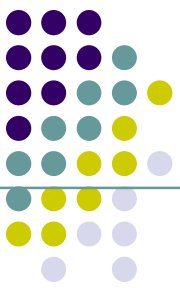
- JavaScript et Java sont des langages de programmation complètement différentes, à la fois dans le concept et le design.
- Java (inventé en 1995 par Sun) est un langage de programmation plus complexe dans la même catégorie que C
- JavaScript a été inventé par Brendan Eich en 1995. Il est apparu dans Netscape (un navigateur qui n'existe plus) et est devenu un standard de l'ECMA en 1997.
- ECMA-262 est son nom officiel. ECMAScript 5 (JavaScript 1.8.5 - Juillet 2010) est le dernier standard.



Introduction

C'est quoi JavaScript?

- JavaScript est un des langages de programmation les plus populaires au monde. C'est un langage pour HTML et le Web, pour les serveurs, les PC, les ordinateurs portables, les tablettes, les Smartphones, etc.
- JavaScript est un langage de script c.à.d. un langage de programmation léger.
- JavaScript est l'un des **trois langages** que tout développeur Web **doit** apprendre:
 1. **HTML** pour définir le contenu des pages web
 2. **CSS** pour spécifier la mise en forme des pages web
 3. **JavaScript** pour programmer le comportement des pages Web



Utilisation

La balise `<script>`

Le code JavaScript doit être inséré dans le HTML entre les balises `<script>` et `</ script>`.

Il peut être mis dans le body et dans la section `<head>` de la page et / ou dans les deux.

Exemple:

```
<script>  
    alert("Ma première page avec JavaScript");  
</script>
```

Vous pourrez trouver l'attribut **type = "text/javascript"** dans la balise `<script>`. Ce n'est plus nécessaire car JavaScript est le langage de script par défaut dans tous les navigateurs modernes et pour HTML5.



Utilisation

JavaScript dans un document HTML

- Dans le `<body>`

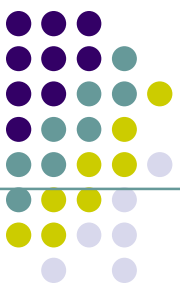
Exemple 1:

```
<!DOCTYPE html>
<html>
<body>
...
<script>
document.write("<h1>Ceci est un entête</h1>");
document.write("<p>Ceci est un
paragraphe</p>");
</script>
...
</body>
</html>
```

- Dans le `<head>`

Exemple 2:

```
<!DOCTYPE html>
<html><head>
<script>
    function myFct(){
        alert("Bonjour");
    }
</script>
</head>
<body>
    <h1>Ma page Web</h1>
    <button onclick="myFct()">Essaie
</button>
</body>
</html>
```



Utilisation

Du code JavaScript externe

Le code JavaScript peut également être placé dans des fichiers externes qui sont souvent utilisés par plusieurs pages HTML différentes.

Un fichier JavaScript externe est d'extension .js.

Pour utiliser un script extérieur, on pointe au fichier .js dans l'attribut "src" de la balise `<script>` :

Exemple 3 :

```
<!DOCTYPE html>
<html><head>
  <script src="monscript.js"></script>
</head>
<body>
  <h1>Ma page Web</h1>
  <button onclick="myFct()">Essaie </button>
</body>
</html>
```

monscript.js

```
function myFct(){
    alert("Bonjour");
}
```




Utilisation

Du code JavaScript externe

- On peut placer le script dans la section `<head>` ou dans `<body>` comme vous le souhaitez.
- Le script va se comporter comme s'il était placé exactement dans la balise `<script>` du document.
- Les scripts externes ne peuvent pas contenir de balises `<script>`.

Avantages : Cela permet de:

- séparer le code JavaScript du code HTML.
- rendre le code HTML et JavaScript plus facile à lire et à mettre à jour
- d'accélérer le chargement de la page avec les fichiers JavaScript en cache.



Utilisation

Fonctions et événements JavaScript

- Les instructions JavaScript dans l'exemple 1 ci-dessus sont exécutées au chargement de la page.
- Le plus souvent, on veut exécuter du code lorsqu'un **événement** se produit, comme lorsque l'utilisateur clique sur un bouton (Exemple 2 et 3).
- Si on met le code JavaScript à l'intérieur d'une **fonction**, on pourra appeler cette fonction lorsqu'un événement se produit.
- Les fonctions et les événements JavaScript seront traités dans la suite.



Méthodes et propriété d'affichage

JavaScript ne possède pas de méthodes d'affichage intégrés.

Il peut afficher les données de différentes manières:

- L'écriture dans une boîte d'alerte, en utilisant **window.alert ()** .
- L'écriture dans la page HTML en utilisant **document.write ()** .
- L'écriture dans un élément HTML, en utilisant **innerHTML** .
- L'écriture dans la console du navigateur, en utilisant **console.log ()** .

La méthode **window.alert()**:

```
<!DOCTYPE html>
<html><body>
<h1>Ma première page web</h1>
<script>
    window.alert(5 + 6);
</script>
</body>
</html>
```

La méthode **document.write()**:

```
<!DOCTYPE html>
<html><body>
<h1>Ma première page web</h1>
<script>
    document.write(5 + 6);
</script>
</body>
</html>
```



Méthodes et propriété d'affichage

La méthode `document.write()`

- La méthode `document.write()` supprime tout le contenu HTML existant si elle est utilisée après le chargement du document HTML.

Exemple :

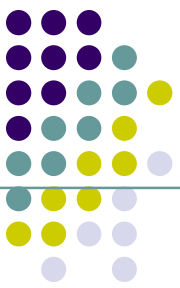
```
<button onclick="myFunction">Click</button>
```

```
<script>
```

```
function myFunction() {  
    document.write(5 + 6);  
}
```

```
</script>
```

NB: La méthode `document.write ()` doit être utilisée que pour des tests.



Méthodes et propriété d'affichage

La propriété innerHTML

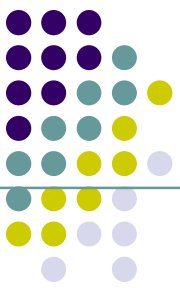
- Pour accéder à un élément HTML, JavaScript peut utiliser la **méthode** `document.getElementById (id)` .
- L'attribut **id** définit l'élément HTML. La **propriété innerHTML** définit le contenu HTML:

Exemple:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```



Méthodes et propriété d'affichage

La méthode `console.log()`

- Pour le débogage, on peut utiliser la **méthode `console.log()`** pour afficher les données.
- On apprendra plus sur le débogage dans la suite:

Exemple:

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```



La syntaxe de JavaScript

- La **syntaxe** JavaScript est l'ensemble de règles qui régissent un programme JavaScript.
- Un programme JavaScript est une liste d'instructions à exécuter par un navigateur web. Ces instructions sont séparées par un **point-virgule (;)**.

Exemple:

```
var x, y, z;  
x = 5;  
y = 6;  
z = x + y;
```

- Une instruction JavaScript est composée de **valeurs**, d'**opérateurs**, d'**expressions**, de **mots clés** et de **commentaires**.



La syntaxe de JavaScript

Les valeurs

- Une valeur est soit fixe soit variable.
- Une valeur fixe est appelée **littérale**.

Exemple: 10.5 1001 (nombre)

"Ugb", 'ndar' (Chaines de caractères)

(Un nombre à virgule est séparé par un point (.) et non par une virgule (,))

- Une valeur variable est appelée **variable**.

Les variables

- Les **variables** sont **utilisées** pour **stocker** des valeurs de données.
- JavaScript utilise le mot-clé **var** pour **déclarer les** variables.
- Un **signe égal** est utilisé pour attribuer **des valeurs** aux variables.

Exemple :

```
var x;
```

```
x = 6;
```




La syntaxe de JavaScript

Les opérateurs

- JavaScript utilise les **opérateurs arithmétiques** (+ - * /) pour **calculer** les valeurs:

Exemple : (5 + 6) * 10

- JavaScript utilise l'**opérateur d'affectation** (=) pour attribuer des valeurs aux variables:

Expressions JavaScript

- Une expression est une combinaison de valeurs, de variables et d'opérateurs, qui calcule une valeur.
- Le calcul s'appelle une évaluation.

Exemples : 5 * 10

x * 10

"Salif" + " " + "Ndour" est évaluée à "Salif Ndour"



La syntaxe de JavaScript

Les mots-clés

- Les **mots clés** sont utilisés pour identifier les actions à effectuer.
- Par exemple le mot-clé **var** indique au navigateur de créer des variables.

Exemple:

```
var x;  x = 5 + 6;
```

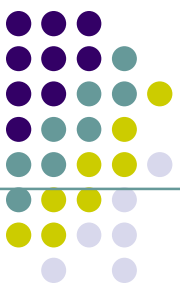
Les commentaires

- Toutes les instructions JavaScript ne sont pas "exécutées".
- Tout code placé après // (double slash) ou entre /* et */ est traité comme un **commentaire** .
- Les commentaires sont ignorés et ne seront pas exécutés:

Exemple :

```
var x = 5;  // Il sera exécuté
```

```
// var x = 6;  Il ne sera pas exécuté
```



La syntaxe de JavaScript

Identificateurs JavaScript

- Les identificateurs sont des noms utilisés pour nommer des variables (des mots clés, des fonctions et des labels).
- Les règles pour les noms légaux sont pratiquement les mêmes dans la plupart des langages de programmation.
- En JavaScript, le premier caractère doit être une lettre, un trait de soulignement (_) ou un signe dollar (\$).
- Les caractères suivants peuvent être des lettres, des chiffres, des caractères de soulignement ou des signes de dollar.
- Les caractères spéciaux et accentués sont interdits (é, à, ç, ï, etc..)
- Les mots réservés (comme JavaScript) ne sont pas autorisés
- Les nombres ne sont pas autorisés comme premier caractère. Ce qui permet à JavaScript de distinguer facilement les identificateurs des nombres.



La syntaxe de JavaScript

Sensibilité à la casse

- Tous les identificateurs JavaScript sont **sensibles à la casse**.

Exemple : Les variables **lastName** et **lastname** sont différentes.

La nomenclature des identificateurs

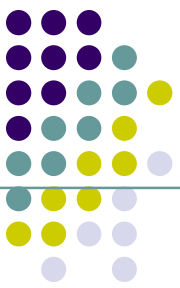
Trois manières sont historiquement utilisées:

- Avec trait d'union: **Exemple :** nom-famille, carte-mere.
Elle n'est pas autorisée en JavaScript. Elle est réservée à la soustraction.
- Avec trait de soulignement: **Exemple:** nom_famille.
- Camel Case: **Exemple :** NomDeFamille
Elle est la plus utilisée en JavaScript avec le premier mot commençant par une lettre minuscule:

Exemple : nomDeFamille.



Jeu de caractères : JavaScript utilise **Unicode** .



Les variables

- **Déclaration et affectation**

- La lecture d'une variable non déclarée provoque une erreur
- Une variable correctement déclarée mais dont aucune valeur n'est affectée, est indéfinie (undefined).

- **La portée**

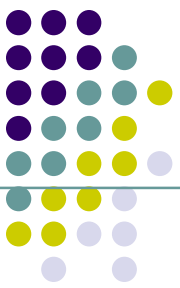
- les variables peuvent être globales ou locales.
- Une variable globale est déclarée en début de script et est accessible à n'importe quel endroit du programme.
- Une variable locale est déclarée à l'intérieur d'une fonction et n'est utilisable que dans la fonction elle-même.



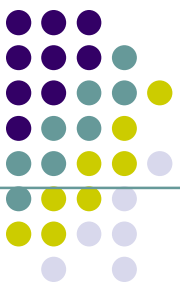
Les variables

- Le type d'une variable dépend de la valeur stockée dans cette variable.
Pas de déclaration de type.
 - Exemple
`var maVariable = 'Cheikh' maVariable = 10;`
- trois principaux types de valeurs
 - String
 - Number : $10^{-308} > \text{nombre} < 10^{308}$
 - » Les nombres entiers
 - » les nombres décimaux en virgule flottant
 - 3 valeurs spéciales :
 - » Positive Infinity ou +Infinity (valeur infini positive)
 - » Negative Infinity ou -Infinity (valeur infinie négative)
 - » Nan (Not a Number) habituellement générée comme résultat d'une opération mathématique incohérente
- Boolean : deux valeurs littérales : true (vrai) et false (faux).

Les variables



- JavaScript inclut aussi deux types de données spéciaux :
 - **Null** : possède une seule valeur, **null**, qui signifie l'absence de données dans une variable
 - **Undefined** : possède une seule valeur, **undefined**. Une variable dont le contenu n'est pas clair car elle n'a jamais stocké de valeur, pas même **null** est dite non définie (undefined).



Les opérateurs

- Les opérateurs arithmétiques**

Ils permettent d'effectuer les opérations sur les nombres.

- Les opérateurs d'affectation**

Ils permettent d'affecter des valeurs aux variables.

Opérateur	Exemple	Equivalent à
=	<code>x = y</code>	<code>x = y</code>
+=	<code>x += y</code>	<code>x = x + y</code>
-=	<code>x -= y</code>	<code>x = x - y</code>
*=	<code>x *= y</code>	<code>x = x * y</code>
/=	<code>x /= y</code>	<code>x = x / y</code>
%=	<code>x %= y</code>	<code>x = x % y</code>
op=	<code>X op= y</code>	<code>x = x op y</code>

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
++	Incrément
--	Décrément

où op est un opérateur



Les opérateurs

Quelques opérateurs sur les chaînes de caractères

- La concaténation
 - `Var chaine = " bonjour " + " FI3/FCD1 ";`
- Déterminer la longueur d'une chaîne
 - `Var ch1 = " bonjour ";`
 - `Var longueur = ch1.length;`
- Identifier le nième caractère d'une chaîne
 - `Var ch1 =" Rebonjour ! ";`
 - `Var carac = ch1.charAt(2);`
- Extraction d'une partie de la chaîne
 - `Var dateDuJour = " 04/04/03 "`
 - `Var mois = datteDuJour.substring(3, 5);`
 - » 3: est l'indice du premier caractère de la sou-chaîne à extraire
 - » 5 : indice du dernier caractère à prendre en considération ; ce caractère ne fera pas partie de la sous-chaîne à extraire

Additionner un nombre avec une chaîne donne une chaîne!



Les opérateurs

- Les opérateurs de comparaison

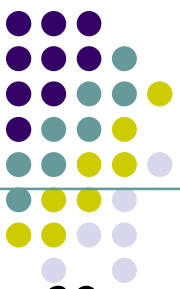
Opérateur	Description
==	Égale à
===	Égalité de valeur et de type
!=	Non égale
!==	Non égalité de valeur et non égalité de type
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale
<=	Inférieur ou égale
?	Opérateur ternaire ex: max = (a>b)?a:b

- Les opérateurs logiques

Opérateur	Description
&&	Et logique
	Ou logique
!	Non logique

- Les opérateurs de type

Opérateur	Description
typeof	Renvoie le type d'une variable
instanceof	Renvoie true si un objet est une instance d'un type d'objet

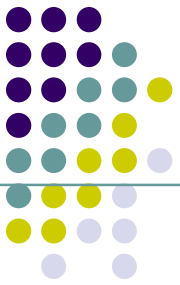


Les opérateurs

• Les opérateurs binaires

Les opérateurs binaires traitent leurs opérandes comme des séquences de 32 bits. Chaque opérande numérique est converti en un nombre de 32 bits. Le résultat est converti en nombre. **(20).toString(2)); // Affiche 10100**

Opérateur	Description	Exemple	équivalent	Résultat	Décimal
&	AND	5 & 1	0101 & 0001	0001	1
	OR	5 1	0101 0001	0101	5
~	NOT ~x = -(x+1) Ex: -9=~8	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Décalage à gauche	5 << 1	0101 << 1	1010	10
>>	Décalage à droite (avec propagation du signe)	5 >> 1	0101 >> 1	0010	2
>>>	Décalage à droite non signé (Le bit de signe devient 0)	5 >>> 1	0101 >>> 1	0010	2



Les opérateurs

- **Les opérateurs binaires (Exemples)**

9 >> 2 donne 2 :

9 (base 10) : 000000000000000000000000000000001001 (base 2)

9 >> 2 (base 10) : 0000000000000000000000000000000010 (base 2) = 2 (base 10)

-9 >> 2 donne -3, parce que le signe est préservé

-9 (base 10) : 111111111111111111111111111111110111 (base 2)

-9 >> 2 (base 10) : 1111111111111111111111111111111101 (base 2) = -3 (base 10)

9 >>> 2 donne 2

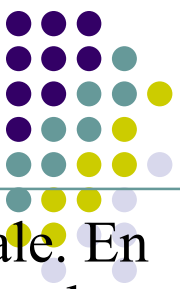
9 (base 10) : 000000000000000000000000000000001001 (base 2)

9 >>> 2 (base 10) : 0000000000000000000000000000000010 (base 2) = 2 (base 10)

-9 >>> 2 donne 1073741821, ce qui est différent de -9 >> 2 (qui donne -3)

-9 (base 10) : 111111111111111111111111111111110111 (base 2)

-9 >>> 2 (base 10) : 0011111111111111111111111111111101 (base 2) = 1073741821 (base 10)



Les types de données

- En programmation, les types de données sont d'une importance capitale. En effet, sans les types de données, un ordinateur ne pourrait pas par exemple résoudre le problème suivant en toute sécurité: `var x = 16 + "Volvo";`
- Les variables JavaScript peuvent être de types de données divers: nombres, chaînes, objets et plus:

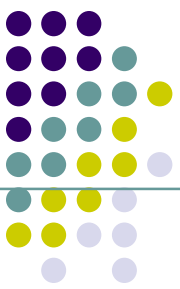
- **Exemple:**

```
var longueur = 16;           // Nombre
var prenom = "Johnson";     // Chaîne
var x = {prenom:"Modou", nom:"Faye"}; // Objet
```

- JavaScript est de **type dynamique**. Cela signifie que la même variable peut être successivement de différents types de données:

- **Exemple:**

```
var x;           // x is indéfini
var x = 5;       // Maintenant x est un nombre
var x = "John";  // Maintenant x est une chaîne
```



Les types de données

- **Le type Chaîne de caractères (String)**

Une chaîne (ou une chaîne de caractères) est une suite de caractères comme "Senegal". Elle est délimitée par des guillemets simples ou doubles.

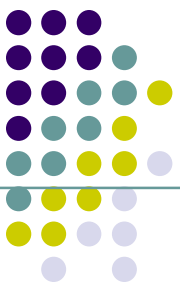
Exemple:

```
var nom = "Sall"; // Avec des guillemets doubles  
var prenom= 'Ngom'; // Avec des guillemets simples
```

Des guillemets peuvent être utilisés à l'intérieur d'une chaîne pourvu qu'ils ne correspondent pas aux guillemets entourant la chaîne:

Exemple:

```
var nom = "Senegal 'pays de la Teraanga";  
nom = 'Senegal "pays de la Teraanga";
```



Les types de données

- **Le type nombre**

JavaScript n'a qu'un seul type nombre. Les nombres peuvent être écrits avec ou sans décimales:

Exemple:

```
var x1 = 34.00;    // Avec decimal
var x2 = 34;       // Sans decimal
```

Une notation scientifique (exponentielle) peut être utilisée pour les nombres très grands ou très petits:

Exemple:

```
var y = 123e5;     // 12300000
var z = 123e-5;    // 0.00123
```

- **Le type boolean**

Le type booléen n'a que deux valeurs: true (vrai) et false (faux).

Exemple:

```
var x = true;
var y = false;
```



Les types de données

- **Le type Array (Tableau)**

Un élément de type tableau peut plusieurs contenir des valeurs de types divers
Les tableaux séparés par des virgules. Ces valeurs sont écrits entre crochets.

Exemple:

```
var voiture= ["Saab", "Volvo", "BMW"];
```

L'indice d'un tableau commence par zéro. Ce qui signifie que le premier élément s'obtient par voiture[0], le second par voiture[1], et ainsi de suite.

- **Le type Object (Objet)**

Les objets sont écrits avec des accolades. Les propriétés des objets sont écrites sous la forme de paires **nom: valeur**, séparées par des virgules.

Exemple:

```
var person = {prenom:"Jean", nom:"Séne", age:50, couleurYeux:"bleue"};
```

L'objet (personne) dans l'exemple ci-dessus a 4 propriétés: prenom, nom, age et couleurYeux.



Les types de données

- **L'opérateur typeof**

L'opérateur **typeof** renvoie le type d'une variable ou d'une expression:

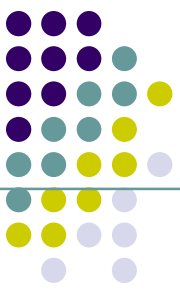
Exemple:

```
typeof (3 + 4)           // Returne "number"  
typeof "Jean"           // Returne "string"
```

- **Données primitives**

Une valeur de données primitive est une simple valeur de données sans propriétés et méthodes supplémentaires. L'opérateur **typeof** peut renvoyer un de les types primitifs:

- string
- number
- boolean
- null
- undefined



Les types de données

- **Données complexes**

L'opérateur `typeof` peut renvoyer l'un des deux types complexes: **function** **object**.

Exemple:

```
typeof [1,2,3,4] // Retourne "object" (non "array" car en JavaScript un tableau est un objet)
```

```
typeof {nom:'Jean', age:34} // Retourne "object"
```

```
typeof function myFunc(){ } // Retourne "function"
```

- **Type Undefined**

En JavaScript, une variable sans valeur a la valeur `undefined` et son type est `undefined`.

Exemple:

```
var personne; // Sa valeur est undefined, et son type est undefined
```

```
var personne = undefined; // Sa valeur est undefined, et son type est undefined
```



Les types de données

- **Valeur vide**

Une valeur vide est une chaîne de longueur égale à zero. Elle n'a rien à voir avec une valeur indéfinie. Une chaîne vide a à la fois une valeur et un type.

Exemple: `var car = "";` // La valeur est "", le typeof est "string"

- **Valeur null**

null signifie quelque chose qui n'existe pas. Null est de type object.

On peut rendre un objet vide en lui affectant la valeur null:

```
var person = null;      // Sa valeur est null, mais il est de type object
```

On peut également rendre un objet vide en lui affectant la valeur undefined:

```
var person = undefined; // Sa valeur est undefined et son type est undefined
```

- **Difference entre Undefined et Null**

```
typeof undefined      // undefined
```

```
typeof null           // object
```

```
null === undefined    // false
```

```
null == undefined     // true
```



Les fonctions

- Une fonction est un bloc de code conçu pour exécuter une tâche particulière.
- Elle est exécutée lorsque qu'elle est appelée depuis le code source ou suite à un évènement (clic, survol de la souris, etc.).
- En JavaScript comme en C, une fonction peut prendre *zéro*, *un* ou *plusieurs* arguments.
- Elle peut renvoyer une valeur en sortie ou ne rien renvoyer
- On déclare une fonction par le mot clé *function* suivie du *nom* de la fonction

Syntaxe:

```
function nom (parametre1, parametre2, parametre3) {  
    code à exécuter  
}
```



Les fonctions

```
function salut(){  
    alert('bonjour le monde');  
}
```

Sans argument et sans valeur de retour

```
function salut(var x){  
    alert ('bonjour' +x);  
}
```

Avec argument et sans valeur de retour

```
function calcul(){  
    var x = 10, y=5, s = x+y;  
    return 'la somme est :' +s;  
}
```

Sans argument et avec valeur de retour

```
function calcul(var x, y){  
    var s=x+y;  
    return s;  
}
```

Avec argument et valeur de retour

Attention : L'appel d'une fonction sans () renverra la définition de la fonction:



Les fonctions

Les fonctions prédéfinies

- **eval** : Elle exécute un code JavaScript à partir d'une chaîne de caractères.

...

```
<script>
```

```
function evaluation() {  
    document.formulaire.calcul.value=eval(document.formulaire.saisie.value);  
}
```

```
<script>
```

...

```
<form name="formulaire">
```

Saisissez une expression mathématique :

```
<input type="text" name="saisi" maxlength="40" size="40">
```

```
<input type="button" value="evaluation" onclick="evaluation()>
```

```
<input type="text" name="calcul" maxlength="40" size="40">
```

```
</form>
```

...



Les fonctions

Les fonctions prédéfinies

- **isFinite**

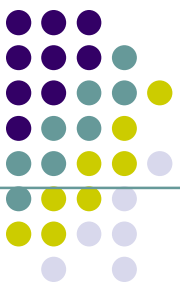
Détermine si le paramètre est un nombre fini. Renvoie *false* si ce n'est pas un nombre ou l'infini positif ou infini négatif.

```
isFinite(240) //retourne true  
isFinite("Un nombre") //retourne false
```

- **isNaN**

Détermine si le paramètre n'est pas un nombre (NaN : Not a Number).

```
isNaN("un nombre") //retourne true  
isNaN(20) //retourne false
```



Les fonctions

Les fonctions prédéfinies

- **parseFloat**
 - analyse une chaîne de caractères et retourne un nombre décimal.
 - Si l'argument évalué n'est pas un nombre, renvoie *NaN* (Not a Number).
- **parseInt**
 - analyse une chaîne de caractères et retourne un nombre entier de la base spécifiée.
 - La base peut prendre les valeurs *16* (hexadécimal) *10* (décimal), *8* (octal), *2* (binaire).

```
var numero="125";  
var nombre=parseFloat(numero); //retourne le nombre 125
```


```
var prix=30.75;  
var arrondi = parseInt(prix, 10); //retourne 30
```




Les objets

Les objets dans la vie réelle

- Dans la vie réelle, une voiture est un objet .
- Une voiture a des propriétés comme le poids et la couleur, et des méthodes comme le démarrage et l'arrêt:

Objet	Propriétés	Méthodes
	<code>voiture.nom = Fiat</code> <code>voiture.modele = 500</code> <code>voiture.poids = 850kg</code> <code>voiture.couleur = white</code>	<code>voiture.demarrer()</code> <code>voiture.conduire()</code> <code>voiture.tourner()</code> <code>voiture.arreter()</code>

- Toutes les voitures ont les mêmes propriétés , mais les valeurs de propriété diffèrent d'une voiture à une autre.
- Toutes les voitures ont les mêmes méthodes, mais elles sont appelées à des moments différents.



Les objets

Affectation de valeurs à un objet

- On a déjà vu comment affecter une valeur à une variable simple:

Exemple : `var car = "Fiat";`

- Les objets sont aussi des variables mais pouvant contenir de nombreuses valeurs.

Exemple : `var car = { type:"Fiat", model:"500", color:"white" };`

- Les valeurs sont écrites sous forme de paires **nom: valeur** (nom et valeur séparés par deux points).

Accès aux propriétés des objets

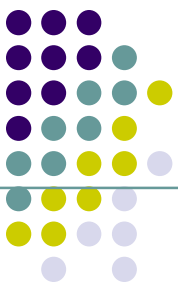
- On accède aux propriétés d'objet de deux façons:

nomObjet.nomPropriété ou *nomObjet["nomPropriété"]*

- **Exemple :**

`personne.nom;`

`personne["nom"];`



Les objets

Accès aux méthodes des objets

- On accède à une méthode objet par la syntaxe suivante: *nomObjet.nomMethode()*

Exemple :

```
nom = personne.prenom();
```

- Accéder à la **méthode** prenom sans () renverra sa **définition**.

Une méthode est en fait une définition de fonction stockée comme une valeur de propriété.

Déclarer un objet avec le mot-clé new

- Une variable JavaScript déclarée avec le mot-clé "**new**" est créée en tant qu'objet:

Exemple :

```
var x = new String();    // Declare x comme un objet String
var y = new Number();    // Declare y comme un objet Number
var z = new Boolean();    // Declare z comme un objet Boolean
```

- Évitez les objets String, Number et Boolean. Ils compliquent votre code et ralentissent la vitesse d'exécution.



Les objets

Réaction aux évènements:

- Les objets JavaScript peuvent réagir à des "Evénements".

Exemples d'objets liés à JavaScript

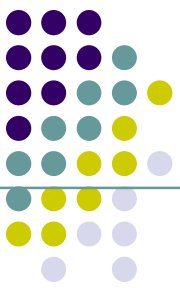
- Le navigateur est un objet qui s'appelle "**navigator**".
- La fenêtre du navigateur est un objet qui se nomme "**window**".
- La page HTML est un autre objet, que l'on appelle "**document**".
- Un formulaire à l'intérieur d'un "**document**", est aussi un objet.
- Un lien hypertexte dans une page HTML, est encore un autre objet. Il s'appelle "**link**". etc...
- Tous les navigateurs ne supportent pas les mêmes objets



Les objets

L'objet String

- Propriété :
 - *length* : retourne la longueur de la chaîne de caractères;
- Méthodes :
 - *anchor()* : formate la chaîne avec la balise <A> nommée;
 - *b()* : formate la chaîne avec la balise ;
 - *big()* : formate la chaîne avec la balise <BIG>;
 - *charAt()* : renvoie le caractère se trouvant à une certaine position;
 - *charCodeAt()* : renvoie le code du caractère se trouvant à une certaine position;
 - *concat()* : permet de concaténer 2 chaînes de caractères;
 - *fromCharCode()* : renvoie le caractère associé au code;
 - *indexOf()* : permet de trouver l'indice d'occurrence d'un caractère dans une chaîne;

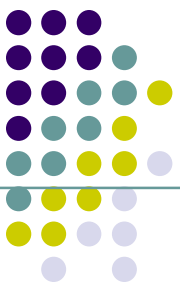


Les objets

L'objet String

– Méthodes :

- *italics()* : formate la chaîne avec la balise <I>;
- *lastIndexOf()* : permet de trouver le dernier indice d'occurrence d'un caractère;
- *link()* : formate la chaîne avec la balise <A> pour permettre de faire un lien;
- *indexOf()* : permet de trouver l'indice d'occurrence d'un caractère dans une chaîne;
- *slice()* : retourne une portion de la chaîne;
- *substr()* : retourne une portion de la chaîne;
- *substring()* : retourne une portion de la chaîne;
- *toLowerCase()* : permet de passer toute la chaîne en minuscule;
- *toUpperCase()* : permet de passer toute la chaîne en majuscules;



Les objets

L'objet Array

- Propriété :
 - *length* : retourne le nombre d'éléments du tableau;
- Méthodes :
 - *concat()* : permet de concaténer 2 tableaux;
 - *join()* : converti un tableau en chaîne de caractères;
 - *reverse()* : inverse le classement des éléments du tableau;
 - *slice()* : retourne une section du tableau;
 - *sort()* : permet le classement des éléments du tableau;



Les objets

L'objet Math

— Propriétés :

- *E* : renvoie la valeur de la constante d'Euler (~ 2.718);
- *LN2* : renvoie le logarithme népérien de 2 (~ 0.693);
- *LN10* : renvoie le logarithme népérien de 10 (~ 2.302);
- *LOG2E* : renvoie le logarithme en base 2 de e (~ 1.442);
- *LOG10E* : renvoie le logarithme en base 10 de e (~ 0.434);
- *PI* : renvoie la valeur du nombre pi (~ 3.14159);
- *SQRT1_2* : renvoie 1 sur racine carrée de 2 (~ 0.707);
- *SQRT2* : renvoie la racine carrée de 2 (~ 1.414);



Les objets

L'objet Math

– Méthodes :

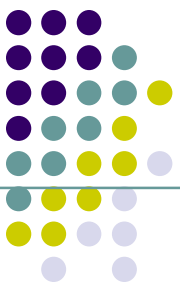
- *abs()*, *exp()*, *log()*, *sin()*, *cos()*, *tan()*, *asin()*, *acos()*, *atan()*, *max()*, *min()*, *sqrt()* sont les opérations mathématiques habituelles;
- *atan2()* : retourne la valeur radian de l'angle entre l'axe des abscisses et un point;
- *ceil()* : retourne le plus petit entier supérieur à un nombre;
- *floor()* : retourne le plus grand entier inférieur à un nombre;
- *pow()* : retourne le résultat d'un nombre mis à une certaine puissance;
- *random()* : retourne un nombre aléatoire entre 0 et 1;
- *round()* : arrondi un nombre à l'entier le plus proche.



Les objets

L'objet Date

- Propriété : aucune;
- Méthodes :
 - *getFullYear()*, *getYear()*, *getMonth()*, *getDay()*, *getDate()*, *getHours()*, *getMinutes()*, *getSeconds()*, *getMilliseconds()*: retournent respectivement l'année complète, l'année (2chiffres), le mois, le jour de la semaine, le jour du mois, l'heure, les minutes, les secondes et les millisecondes stockés dans l'objet *Date*;
 - *getUTCFullYear()*, *getUTCYear()*, ... retournent respectivement l'année complète, l'année (2chiffres), ... stockés dans l'objet *Date* en temps universel;
 - *setFullYear()*, *setYear()*, ... remplacent respectivement l'année complète, l'année (2 chiffres), ... dans l'objet *Date*;



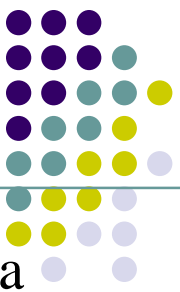
Les objets

L'objet Date

- *setUTCFullYear()*, *setUTCYear()*, ... remplacent l'année complète, l'année (2chiffres), ... dans l'objet *Date* en temps universel;
- *getTime()* : retourne le temps stocké dans l'objet *Date*;
- *getTimezoneOffset()* : retourne la différence entre l'heure du client et le temps universel;
- *toGMTString()*, *toLocaleString()*, *toUTCString()* : convertissent la date en chaîne de caractère selon la convention GMT, selon la convention locale ou en temps universel;

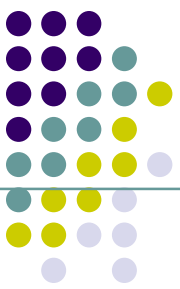
Exemple:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

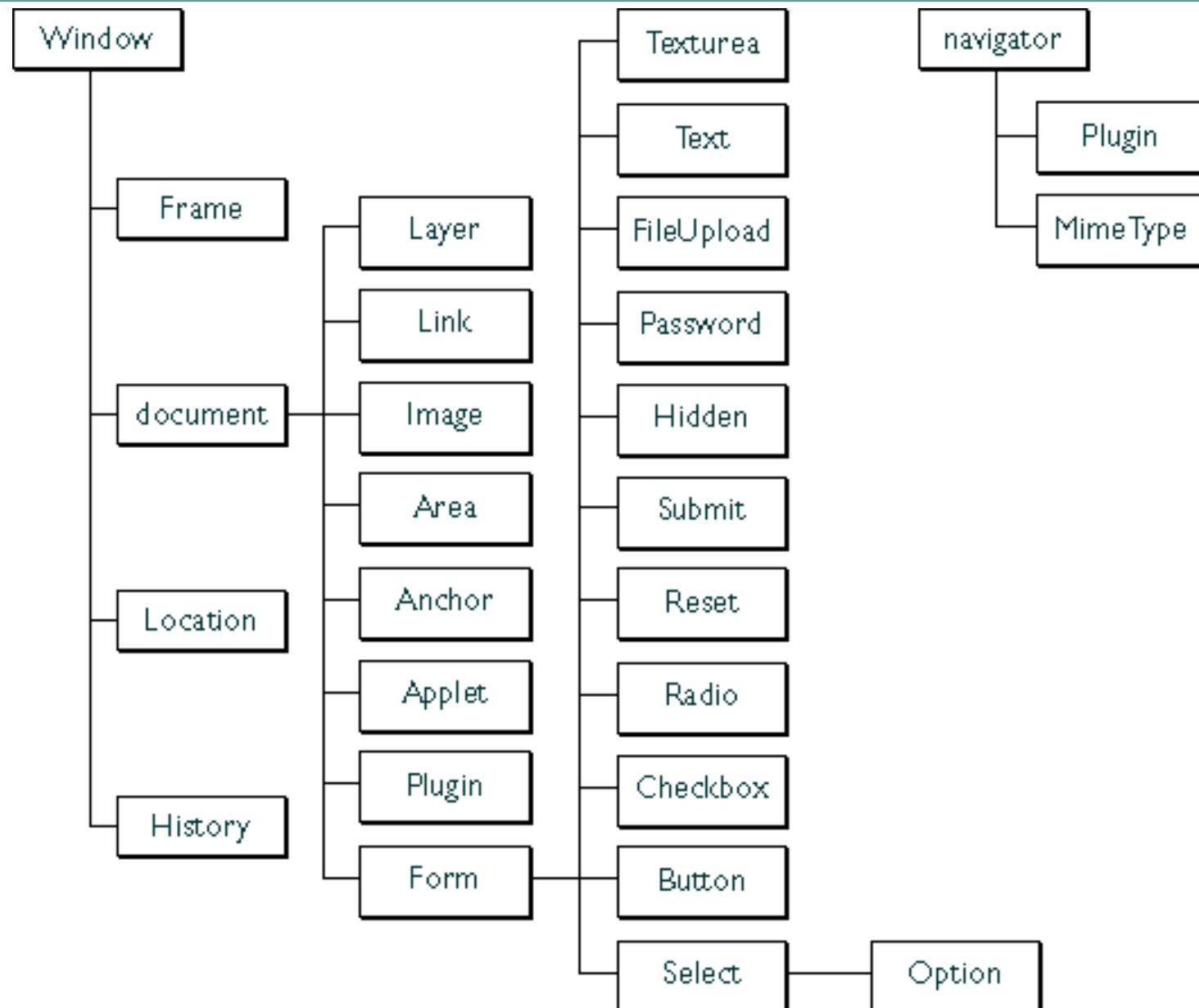


Les objets du navigateur

- L'objet le plus haut dans la hiérarchie est *window* qui correspond à la fenêtre même du navigateur.
- L'objet *document* fait référence au contenu de la fenêtre.
- *document* regroupe au sein de propriétés l'ensemble des éléments HTML présents sur la page. Pour atteindre ces différents éléments, nous utiliserons :
 - *soit des méthodes propres à l'objet document*, comme la méthode *getElementById()*, qui permet de trouver l'élément en fonction de son identifiant (id);
 - *soit des collections d'objets* qui regroupent sous forme de tableaux Javascript tous les éléments de type déterminé.



Les objets du navigateur

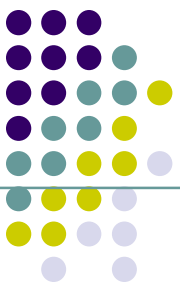




Les objets du navigateur

L'objet window

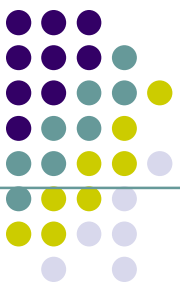
- Propriétés : (accessibles avec IE et N)
 - *closed* : indique que la fenêtre a été fermée;
 - *defaultStatus* : indique le message par défaut dans la barre de status;
 - *document* : retourne l'objet *document* de la fenêtre;
 - *frames* : retourne la collection de cadres dans la fenêtre;
 - *history* : retourne l'historique de la session de navigation;
 - *location* : retourne l'adresse actuellement visitée;
 - *name* : indique le nom de la fenêtre;



Les objets du navigateur

L'objet window

- *navigator* : retourne le navigateur utilisé;
- *opener* : retourne l'objet *window* qui a créé la fenêtre en cours;
- *parent* : retourne l'objet *window* immédiatement supérieur dans la hiérarchie;
- *self* : retourne l'objet *window* correspondant à la fenêtre en cours;
- *status* : indique le message affiché dans la barre de status;
- *top* : retourne l'objet *window* le plus haut dans la hiérarchie.



Les objets du navigateur

L'objet window

- Méthodes :
 - *blur()* : enlève le focus de la fenêtre;
 - *close()* : ferme la fenêtre;
 - *focus()* : place le focus sur la fenêtre;
 - *moveBy()* : déplace d'une distance;
 - *moveTo()* : déplace la fenêtre vers un point spécifié;
 - *open()* : ouvre une nouvelle fenêtre;
 - *print()* : imprime le contenu de la fenêtre;
 - *resizeBy()* : redimensionne d'un certain rapport;
 - *resizeTo()* : redimensionne la fenêtre;
 - *setTimeout()* : évalue une chaîne de caractère après un certain laps de temps.



Les objets du navigateur

L'objet Document

- **Propriétés :**
 - *applets* : retourne la collection d'applets java présente dans le document;
 - *cookie* : permet de stocker un cookie;
 - *domain* : indique le nom de domaine du serveur ayant apporté le document;
 - *forms* : retourne la collection de formulaires présents dans le document;
 - *images* : retourne la collection d'images présentes dans le document;
 - *links* : retourne la collection de liens présents dans le document;



Les objets du navigateur

L'objet Document

- *referrer* : indique l'adresse de la page précédente;
 - *title* : indique le titre du document.
-
- **Méthodes :**
 - *close()* : ferme le document en écriture;
 - *open()* : ouvre le document en écriture;
 - *write()* : écrit dans le document;
 - *writeln()* : écrit dans le document et effectue un retour à la ligne
 - *getElementById()*: récupère un élément du document par son id



Les objets du navigateur

L'objet Navigator

- **Propriétés**
 - *appName* : application (Netscape, Internet Explorer)
 - *appVersion* : numero de version.
 - *platform* : système d'exploitation (Win32)
 - *plugins*
 - *language*
 - *mimeTypes*
 - *JavaEnabled()*



Les structures conditionnelles et de contrôle

- **Structures conditionnelles :**

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

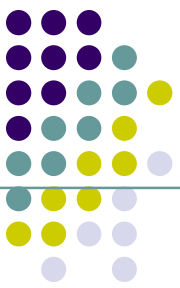
```
switch( ... ) {  
    case ... : { ... } break  
    ...  
    default : { ... }  
}
```

Exemple:

```
if(a>=0 ) {  
    alert("A est positif");  
} else {  
    alert("A est négatif");  
}
```

Exemple:

```
switch(n) {  
    case 1 : { alert("Un"); } break  
    case 2 : { alert("Deux"); }  
                break  
    default : { ... }  
}
```



Les structures conditionnelles et de contrôle

- **Structures de contrôle:**

```
for( ... ; ... ; ... ) {  
    ...  
}
```

Exemple:

```
For(i=1;i<=10;i++)  
    som = som + i;
```

```
while( ... ) {  
    ...  
}
```

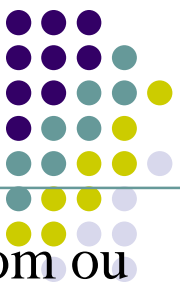
Exemple:

```
Som = 0; i=1;  
While (i<=10){  
    som = som + i;  
}
```

```
do {  
    ...  
} while( ... );
```

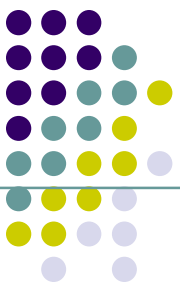
Exemple:

```
Som = 0; i=1;  
do{  
    som = som + i;  
}while(i<=10);
```



Nommage des objets-éléments

- Pour pouvoir manipuler un objet en JavaScript, il doit posséder un nom ou un identifiant
- Pour pouvoir distinguer les différents objets-éléments d'une page web, il suffit de leur donner un nom à travers l'attribut NAME ou l'attribut ID
 - `<Table Name="tableau1">...`
 - `<Table id="tab2">...`
 - `<Form Name = "formulaire1">...`
 - `<Form Name ="formulaire2">...`
 - `<Textarea id="texte1">...`
- Dans le cas où l'objet serait unique alors pas besoin de nom pour désigner cet objet
 - Exemple : le cas de BODY (une seul BODY par document), DOCUMENT (un seul DOCUMENT par fenêtre)



Manipulation des objets

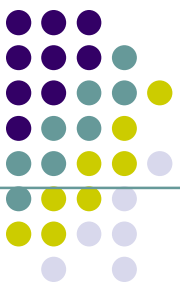
- **En utilisant le « chemin d'accès » dans l'arborescence**
 - Pour adresser un objet, il faut préciser son « chemin d'accès » dans l'arborescence de la structure

```
<body onLoad="window.document.formulaire.zone.value='Bonjour';">  
  <form name="formulaire">  
    <input name="zone" type="text">  
</form></body>
```

- Si le nom de la fenêtre est omis, le navigateur utilisera par défaut la fenêtre courante
- Dans le cas des iframes, on donne le nom de la fenêtre
- Il peut omettre window.document s'il n'y a qu'un seul objet « document » dans la fenêtre
- **En utilisant la méthode getElementById**

Pour adresser un objet par son id, on utilise la fonction getElementById.

```
<body onLoad="document.getElementById('zone').value='Bonjour';">  
  <form name="formulaire">  
    <input id="zone" type="text">  
</form></body>
```



Les événements

- Javascript est dépendant des événements qui
 - se produisent lors d'actions diverses sur les objets d'un document HTML.
 - onLoad;
 - onClick
 - onMouseover
 - onMouseout
 - ...
- Il est possible de baser l'exécution de fonctions sur des événements
- Pour avoir la liste complete des événements possible,
https://www.w3schools.com/jsref/dom_obj_event.asp



Les événements

- **Événement onLoad**

- Se produit lorsque une page web est chargée dans la fenêtre du navigateur
- Toute la page (y compris les images qu'elle contient si leur chargement est prévu) doit avoir été chargée pour qu'il ait lieu
- Cet événement peut être associé à une image seulement ; auquel cas, il se produit une fois son chargement terminé

```
<HTML><BODY onLoad="alert('page chargée');">  
Exemple de l'événement onLoad  
</BODY></HTML>
```

- **Événement onUnload**

Se produit lorsque l'utilisateur quitte la page

```
<HTML><BODY onUnload="alert('page quittée');">  
Exemple de l'événement onUnload  
</BODY></HTML>
```

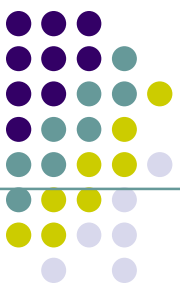


Les événements

- **Événement onClick**

- Se produit lorsque l'utilisateur clique sur un élément spécifique dans une page, comme un lien hypertexte, une image, un bouton, du texte, etc.
- Ces éléments sont capables de répondre séparément à cet événement
- Il peut également être déclenché lorsque l'utilisateur clique n'importe où sur la page s'il a été associé non pas à un élément spécifique, mais à l'élément body tout entier

```
<HTML><BODY>  
<INPUT TYPE="Button" Value="cliquer ici"  
onClick="alert('Clic') ">  
</BODY></HTML>
```



Les événements

- **Événement onmouseover**

- Analogue à onClick sauf qu'il suffit que l'utilisateur place le pointeur de sa souris sur l'un des éléments précités (lien hypertexte, image, bouton, texte, etc.) pour qu'il ait lieu

- **Événement onmouseout**

- A l'inverse de onmouseover, cet événement se produit lorsque le pointeur de la souris quitte la zone de sélection d'un élément.

```
<HTML><BODY>  
<IMG SRC="image.gif" onMouseOver="src='image2.gif';"  
onMouseOut="src='image.gif';">  
</BODY></HTML>
```



Les événements

- **Événement onFocus**

Lorsque un élément de formulaire a le focus c.-à-d. devient la zone d'entrée active

- **Événement onBlur**

Lorsque un élément de formulaire perd le focus c.-à-d. que l'utilisateur clique hors du champs et que la zone d'entrée n'est plus active.

- **Événement onChange**

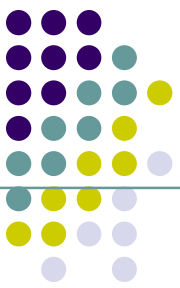
Lorsque la valeur d'un champ de formulaire est modifiée

- **Événement onSelect**

Lorsque l'utilisateur sélectionne un champ dans un élément de formulaire.

- **Événement onSubmit**

Lorsque l'utilisateur clique sur le bouton Submit pour envoyer un formulaire.



Quelques exemples

Exemple 1: Ouverture de fenêtre

```
<FORM>  
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre"  
onClick="open('test.html', 'new',  
'width=300,height=150,toolbar=no,location=no,  
directories=no,status=no,menubar=no,scrollbars=no,resizable=no'  
) ">
```

(sans espaces ni passage à la ligne)

```
</FORM>
```

où **open()** ouvre test.html dans une nouvelle fenêtre.



Quelques exemples

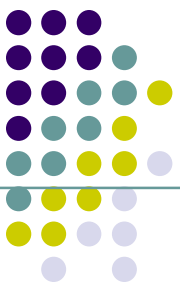
Exemple 2: Fermeture de fenêtre

```
<HTML>
<BODY>
<H1>Ceci est un test<H1>
<FORM>

<INPUT TYPE="button" value= "Continuer"
onClick="self.close()">

</FORM>
</BODY>
</HTML>
```

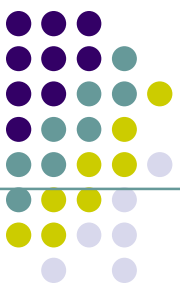
où **self.close()** fermera la fenêtre courante, c.-à-d. la nouvelle fenêtre.



Quelques exemples

Exemple 3: Calcul du carré d'un nombre

```
<body>
<script>
function carre() {
    var nb=document.getElementById("nbre").value;
    nb = parseInt(nb,10);
    elt = document.getElementById("resultat");
    elt.innerHTML=nb+"*"+nb+" = "+nb*nb;
}
</script>
<p>Calcul du carré d'un nombre :</p>
<form>
    <input type="text" id="nbre">
    <input type="button" onclick="carre()" value="Calcul">
</form>
<p id="resultat"></p>
</body>
```



Les expressions régulières

Définition

- Une expression régulière est une séquence de caractères qui forme un **modèle** de recherche.
- Lorsqu'on recherche des données dans un texte, on peut utiliser ce modèle de recherche pour décrire ce qu'on recherche.
- Une expression régulière peut être un caractère unique ou un modèle plus complexe.
- Les expressions régulières peuvent être utilisées pour effectuer tous les types d'opérations de recherche de **texte** et de remplacement de **texte** .

Syntaxe: */pattern/option* ;

Exemple: `var patt = /ec2lt/i;`

`/ ec2lt / i` est une expression régulière.

`ec2lt` est un modèle (à utiliser dans une **recherche**).

`i` est un modificateur (modifie la recherche pour qu'il soit insensible à la casse)⁷².



Les expressions régulières

Utilisation des méthodes de chaîne

En JavaScript, les expressions régulières sont souvent utilisées avec les deux **méthodes de l'objet string** : `search ()` et `replace ()`.

- **La méthode `search ()`** utilise une expression pour rechercher une correspondance et renvoie la position de la chaîne correspondance.
- **La méthode `replace ()`** renvoie une chaîne modifiée où le pattern est remplacé.

Utilisation de `search ()` avec une expression régulière

- **Exemple :** Cet exemple utilise une expression régulière pour effectuer une recherche de "ec2lt" sans tenir compte de la casse:

```
var str = "Visitez Ec2lt";  
var n = str.search(/ec2lt/i);
```

Résultat : `n = 6`



Les expressions régulières

Utilisation de search () avec une chaîne

- Elle accepte également une chaîne comme argument de recherche.
L'argument de chaîne sera converti en une expression régulière:

Exemple

```
var str = "Visitez l'Ec2lt!";  
var n = str.search("Ec2lt");
```

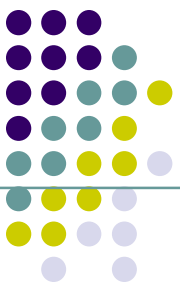
Utilisation de replace() avec une expression régulière

Exemple

```
var str = "Visitez l'UFR SEFS!";  
var res = str.replace(/sefs/i, "SAT");
```

Résultat:

Visitez l'UFR SAT!



Les expressions régulières

Utilisation de `replace()` avec une chaîne

- Elle accepte également une chaîne comme argument de recherche.

Exemple

```
var str = "Visitez l'Ec2lt!";  
var n = str.replace("Ec2lt","Estm");
```

NB: Les expressions régulières peuvent rendre la recherche beaucoup plus puissante (avec la sensibilité à casse par exemple).



Les expressions régulières

Les modificateurs de recherche

Des modificateurs peuvent être utilisés pour effectuer des recherches plus globales sans distinction de cas:

- `i` : Pour ne pas tenir compte de la sensibilité
- `g` : Pour une recherche globale (ne pas s'arrêter au premier trouvé)
- `m` : Pour effectuer une correspondance multiligne

Modèles d'expression régulière

Les crochets sont utilisés pour trouver une gamme de caractères:

- `[abc]` : Trouver les caractères entre parenthèses
- `[0-9]` : Trouver les chiffres entre parenthèses
- `(x|y)` : Trouver l'un des caractères séparées par `|`



Les expressions régulières

Les métacaractères :

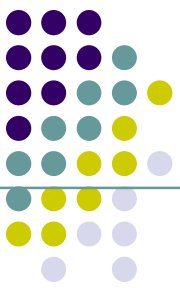
Ce sont des caractères ayant une signification particulière:

- `\d` : Trouver un chiffre
- `\s` : Trouver un caractère en blanc
- `\b` : Trouver une correspondance au début ou à la fin d'un mot
- `\uxxxx` : Trouver le caractère Unicode spécifié par le nombre hexadécimal
xxxx

Les quantificateurs :

Ils définissent des quantités:

- `n+` : n'importe quelle chaîne contenant au moins un n
- `n*` : n'importe quelle chaîne contenant au moins zéro occurrences de n
- `n` : n'importe quelle chaîne contenant zéro ou une occurrence de n



Les expressions régulières

Méthodes d'une expression régulière

Une expression régulière est un objet avec des propriétés et des méthodes prédéfinies.

- **La méthode test ()**

Elle permet de chercher un motif dans une chaîne et renvoie true ou false, selon le résultat.

Exemple: recherche du caractère "e" dans une chaîne :

```
var patt = /e/;  
patt.test("Les meilleurs choses de la vie sont gratuits!");
```

Ces deux lignes sont équivalentes à :

```
/e/.test("Les meilleurs choses de la vie sont gratuits!");  
:
```



Les expressions régulières

Méthodes d'une expression régulière

- La méthode `exec()`

Elle cherche dans une chaîne un motif spécifié et renvoie le texte trouvé.

Si aucune correspondance n'est trouvée, elle renvoie *null*.

Exemple : Recherche du caractère "e" dans une chaîne :

```
/e/.exec("Les meilleurs choses de la vie sont gratuits!");
```



Gestion des erreurs

Les instructions Throw, Try et Catch et Finally

- **try** permet de tester les erreurs dans un bloc de code.
- **catch** permet de gérer les erreurs.
- **throw** permet de créer des messages erreurs personnalisés.
- **finally** permet d'exécuter du code, après try et catch, quel que soit le résultat.

Exemple : Un erreur est délibérément produite avec adddlert :

```
<p id="demo"></p>
<script>
try {
    adddlert("Bienvenue!");
}
catch(err) {
    document.getElementById("demo").innerHTML = err.message;
}
</script>
```




Gestion des erreurs

L'instruction **throw**

- L'instruction **throw** permet de créer un message erreur personnalisé.
- Techniquement, on peut **lancer une exception (lancer une erreur)** .
- L'exception peut être une chaîne JavaScript, un nombre, un booléen ou un objet:

Exemple :

```
throw "Too big";    // throw a text  
throw 500;          // throw a number
```

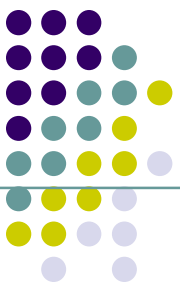
- En utilisant **throw** avec avec **try** et **catch**, on peut contrôler le flux du programme et générer des messages d'erreur personnalisés.



Gestion des erreurs

Exemple sur throw

```
function myFunction() {  
  var x, message = document.getElementById("message");  
  message.innerHTML = "";  
  x = document.getElementById("demo").value;  
  try {  
    if(x == "") throw "vide";  
    if(isNaN(x)) throw "Pas un nombre";  
    x = Number(x);  
    if(x < 5) throw "Trop bas";  
    if(x > 10) throw "Trop haut";  
  }  
  catch(err) {  
    message.innerHTML = "L'entrée est " + err;  
  }  
}
```



Gestion des erreurs

L'instruction finally

L'instruction finally vous permet d'exécuter du code, après try et catch, quel que soit le résultat:

```
try {  
    Bloc de code pour essayer  
}  
catch(err) {  
    Bloc de code pour gérer les erreurs  
}  
finally {  
    Bloc de code à exécuter indépendamment du résultat du bloc try / catch  
}
```



Gestion des erreurs

Exemple sur l'instruction finally

```
function myFunction() {  
    var x, message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {  
        if(x == "") throw "vide";  
        if(isNaN(x)) throw "Pas un nombre";  
        x = Number(x);  
        if(x < 5) throw "Trop bas";  
        if(x > 10) throw "Trop haut";  
    }  
    catch(err) {message.innerHTML = "Erreur: " + err + ".";}  
    finally {  
        document.getElementById("demo").value = "";  
    }  
}
```



Gestion des erreurs

L'objet Error

- JavaScript possède un objet Error intégré qui fournit des informations d'erreur lorsqu'une erreur se produit.
- L'objet Error fournit deux propriétés utiles: **name** et **message** représentant respectivement le nom de l'erreur et un message d'erreur

Les valeurs de name

Six valeurs différentes peuvent être renvoyées par la propriété name:

Valeur de name	Description
EvalError	Une erreur s'est produite dans la fonction eval ()
RangeError	Une erreur de nombre hors plage s'est produite
ReferenceError	Une référence illégale s'est produite
SyntaxError	Une erreur de syntaxe s'est produite
TypeError	Une erreur de type s'est produite
URIError	Une erreur s'est produite dans encodeURIComponent ()



Gestion des erreurs

Les valeurs de name

- **EvalError (Erreur d'évaluation)**

Elle indique une erreur de la fonction eval (). Les versions plus récentes de JavaScript ne lance plus cette erreur. SyntaxError est utilisée à la place.

- **RangeError (Erreur de plage)**

Elle est déclenchée si un nombre est en dehors de la plage des valeurs légales.

Exemple: On ne peut pas définir à 500 le nombre de chiffres significatifs d'un nombre.

```
var num = 1;
try {
    num.toPrecision(500); // Un nbre ne peut pas avoir 500 chiffres significatifs
}
catch(err) {
    document.getElementById("demo").innerHTML = err.name;
}
```



Gestion des erreurs

Les valeurs de name

- **ReferenceError (Erreur de référence)**

Elle est lancée si on fait référence à une variable non déclarée:

Exemple

```
var x;  
try {  
    x = y + 1; // y ne peut être référencée  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.name;  
}
```



Gestion des erreurs

Les valeurs de name

- **SyntaxError (Erreur de syntaxe)**

Elle est lancée si on essaie d'évaluer le code avec une erreur de syntaxe.

Exemple

```
try {  
    eval("alert('Hello)"); // Oublier ' peut produire une erreur  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.name;  
}
```




Gestion des erreurs

Les valeurs de name

- **TypeError (Erreur de type)**

Elle est **déclenchée** si on utilise une valeur en dehors de la plage des types attendus:

Exemple

```
var num = 1;  
try {  
    num.toUpperCase(); // On ne peut convertir un nombre en majuscule  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.name;  
}
```



Gestion des erreurs

Les valeurs de name

- **URIError (Erreur d'URI)**

Elle est lancée si on utilise des caractères non autorisés dans une fonction URI:

Exemple

```
try {  
    decodeURI("%%%"); // Caractères non autorisés  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.name;  
}
```



Gestion des erreurs

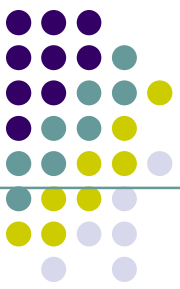
Les valeurs de name

- **Propriétés d'objet erreur non standard**

Mozilla et Microsoft ont définis certaines propriétés d'objet d'erreur non standard:

- FileName (Mozilla)
- LineNumber (Mozilla)
- ColumnNumber (Mozilla)
- stack (Mozilla)
- description (Microsoft)
- number (Microsoft)

NB: Il est conseillé de pas utiliser ces propriétés dans des sites Web publics car elles ne fonctionnent pas dans tous les navigateurs.



Le débogage en JavaScript

Définition

- Le code de programmation peut contenir des erreurs de syntaxe ou des erreurs logiques.
- Beaucoup de ces erreurs sont difficiles à diagnostiquer.
- Souvent, lorsque de telles erreurs se produisent, rien ne se produira, pas de message d'erreur, ni d'indication de recherche de l'erreur.
- Le débogage est le processus de test, de recherche et de réduction des erreurs dans un programme informatique.
- Le premier bug informatique connu était un vrai bogue (un insecte) coincé dans l'électronique.



Le débogage en JavaScript

Les débogueurs intégrés

- Le débogage n'est pas facile. Mais heureusement, tous les navigateurs modernes disposent d'un débogueur JavaScript intégré.
- Les débogueurs intégrés peuvent être activés et désactivés, ce qui oblige à signaler les erreurs à l'utilisateur.
- Avec un débogueur, on peut également définir des points d'arrêt (endroits où l'exécution du code peut être arrêtée) et examiner les variables pendant l'exécution du code.
- Si votre navigateur prend en charge le débogage, vous pouvez utiliser `console.log ()` pour afficher les valeurs JavaScript dans la fenêtre du débogueur:



Le débogage en JavaScript

La méthode console.log ()

Si votre navigateur prend en charge le débogage, vous pouvez utiliser console.log () pour afficher les valeurs JavaScript dans la fenêtre du débogueur:

Exemple

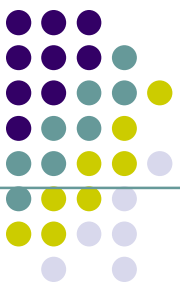
```
<!DOCTYPE html>
<html>
<body>
  <script>
    a = 5;
    b = 6;
    c = a + b;
    console.log(c);
  </script>
</body>
</html>
```



Le débogage en JavaScript

Définition des points d'arrêt

- Dans la fenêtre du débogueur, on peut définir des points d'arrêt dans le code JavaScript.
- À chaque point d'arrêt, JavaScript cesse d'exécuter et permet d'examiner les valeurs JavaScript.
- Après avoir examiné les valeurs, on peut reprendre l'exécution du code (généralement avec un bouton de lecture).



Le débogage en JavaScript

Le mot-clef debugger

- Il arrête l'exécution de JavaScript et appelle (si disponible) la fonction de débogage.
- Cela a la même fonction que de définir un point d'arrêt dans le débogueur.
- Si aucun débogage n'est disponible, l'instruction de débogage n'a aucun effet.
- Avec le débogueur activé, le code suivant cessera de s'exécuter avant l'exécution de la troisième ligne.

Exemple

```
var x = 15 * 5;  
debugger;  
document.getElementById("demo").innerHTML = x;
```