

Les objets en JavaScript

Sommaire

Introduction.....	3
I. Propriétés et méthodes.....	4
A. Les propriétés.....	4
B. Méthodes.....	4
C. Accéder aux méthodes et aux propriétés d'un objet.....	4
D. Parcourir les propriétés d'un objet : boucle For ... In.....	5
E. Créer vos propres objets.....	7
F. Appeler une méthode ou modifier une propriété.....	8
II. Les objets JavaScript natifs.....	9
A. Objet Array.....	9
B. Méthodes couramment utilisées.....	11
C. Objet String.....	12
D. Méthodes couramment utilisées.....	12
E. Objet Date.....	14
F. Objet Math.....	17
G. Objet RegExp.....	19

Crédits des illustrations :
© Fotolia, DR

Les repères de lecture



Retour au chapitre



Définition



Objectif(s)



Espace Élèves



Vidéo /
Audio



Point important /
À retenir



Remarque(s)



Pour aller
plus loin



Normes et lois



Quiz

```
</script>  
<script type="text/script"  
src="http://www.com/counter/counter.js"><  
<noscript><div class="counter"><a title="l  
analytics" href="http://com/web/"  
target="_blank"></a></div></noscript>  
Code for Web -->
```

Introduction

JavaScript est un langage objet. Un objet possède un nom, des propriétés et des méthodes.

En JavaScript, tout type de données peut être considéré comme un objet.

Nous allons voir, dans ce chapitre, comment accéder aux propriétés et aux méthodes d'un objet.

I. Propriétés et méthodes

A. Les propriétés

Les propriétés sont des valeurs associées à un objet.

Exemple

L'exemple suivant affiche la propriété "length" de la chaîne de caractère "Coucou c'est moi!" pour afficher la longueur de cette chaîne de caractères.

```
<script type="text/javascript">
var texte="Coucou c'est moi!";
document.write(texte.length);
</script>
```

Fig. 1

Résultat attendu

17

Fig. 2

B. Méthodes

Les méthodes sont des actions possibles associées aux objets.

1. L'opérateur «typeof»

L'opérateur *typeof* renvoie le type d'objet d'une variable (*string*, *number*, *boolean*, *undefined*).

Exemple

Tableau n°1

SOIT	VALEUR RENVOYÉE PAR L'OPÉRATEUR TYPEOF X
var x=0	«number»
var x=»toto«	«string»
var x=true	«boolean»
	«undefined» (la variable n'est pas déclarée)
var x	«object»

C. Accéder aux méthodes et aux propriétés d'un objet

On accède aux propriétés ou aux méthodes d'un objet à l'aide de son nom.

1. Syntaxe

```
nomDeLObjet.nomDeLaPropriété // syntaxe usuelle  
nomDeLObjet[nomDeLaPropriété] // variante
```

Fig.3

Exemple

```
var monAuto = new Object();  
monAuto.marque = "Renault";  
monAuto["modele"] = "R5";  
document.write(monAuto["marque"] + "<br/>");  
document.write(monAuto.modele + "<br/>");
```

Fig.4

Résultat attendu

```
Renault  
R5
```

Fig.5

D. Parcourir les propriétés d'un objet : boucle For ... In

Cette instruction est utilisée pour parcourir les propriétés d'un Objet.

1. Syntaxe

```
for (variable in object){  
  Instruction(s) à exécuter  
}
```

Fig.6

Exemple

```
<script type="text/JavaScript">
<!--
var property;
document.write("Propriétés de l'objet navigator :<br /> ");
for (property in navigator)
{
    document.write(property);
    document.write("<br />");
}
document.write("Fin de boucle!");
//-->
</script>
```

Fig.7

Résultat attendu (cette liste peut varier suivant le navigateur et la version utilisée)

```
Propriétés de l'objet navigator :
appCodeName
appName
appMinorVersion
cpuClass
platform
plugins
opsProfile
userProfile
systemLanguage
userLanguage
appVersion
userAgent
onLine
cookieEnabled
mimeTypes
Fin de boucle!
```

Fig.8

E. Créer vos propres objets

Les objets JavaScript sont un moyen puissant pour structurer votre code, le rendre plus simple et lisible. Il existe deux manières de créer des objets :

- créer directement une instance d'un objet ;
- ou utiliser un constructeur.

1. Créer directement une instance d'un objet

Le code suivant crée une nouvelle instance d'un objet et lui ajoute 3 propriétés :

```
var vin = new Object();  
vin.marque = "Saint Estèphe"  
var vin.annee = "2003";  
var vin.prix = 14;
```

Fig. 9

Une syntaxe alternative vous permet d'écrire cela différemment :

```
var vin={  
  marque: "Saint Estèphe",  
  annee: "2003",  
  prix: 14  
}
```

Fig. 10

Ajouter une méthode est aussi simple. Le code suivant ajoute la méthode `montant()` à l'objet `vin`.

```
vin = montant;
```

Fig. 11

2. Créer un objet constructeur

Un objet constructeur est une fonction comme une autre. L'instruction **new nomDeLaFonction** permettra d'instancier un objet basé sur cette fonction ainsi que le montre l'exemple suivant :

```
function Vin(marque, annee, prix) {  
  this.marque = marque;  
  this.annee = annee;  
  this.prix = prix;  
  this.quantite = 1;  
  this.montant = function() {  
    return this.quantite * this.prix;  
  }  
}  
var vin1 = new Vin("Saint Estèphe", "2003", 14);
```

Fig. 12

3. This

Le mot-clé **this** utilisé dans un constructeur permet d'accéder aux propriétés de l'objet instancié.

Exemple:

```
this.quantite
```

Fig. 13

F. Appeler une méthode ou modifier une propriété

Si nous reprenons l'exemple précédent, voici deux instructions qui permettent l'une de modifier une propriété (quantité), l'autre de faire appel à une méthode (montant) :

```
vin1.quantite +=5;  
document.write(vin1.montant());
```

Fig. 14

II. Les objets JavaScript natifs

Les objets natifs fournis par JavaScript permettent de manipuler tout type de données.

Tableau n°2

TYPE DE DONNÉES	OBJETS
Chaîne de caractère	String et RegExp
Nombre	Number et Math
Booléen	Boolean
Tableau	Array
Date	Date
Fonction	Function

A. Objet Array

Un objet **Array** est un tableau qui permet de stocker des données et de les retrouver grâce à un index.

1. Création d'un tableau

La méthode **New** permet la création d'un tableau.

2. Syntaxe

```
var monTableau = new Array(); // initialize un tableau vide  
var monTableau = new Array( "valeur 1", "valeur2", ..., "valeur n" );
```

Fig. 15

Exemple

```
var fruits = new Array();  
var fruits = new Array("Pomme", "Poire", "Abricot");
```

Fig. 16

3. Index

Pour référencer une entrée d'un tableau, vous utiliserez la valeur de son index spécifiée entre crochets. L'index commence à 0.

4. Syntaxe

```
monTableau[index]
```

Fig. 17

Exemple

```
var fruits = new Array("Pomme", "Poire", "Abricot");  
document.write(fruits[1]);
```

Fig. 18

Résultat attendu

Poire

Fig. 19

5. Nombre d'entrées d'un tableau

La propriété **length** permet de connaître le nombre d'entrées du tableau.

Exemple

```
var fruits = new Array("Pomme", "Poire", "Abricot");  
document.write(fruits.length);
```

Fig. 20

Résultat attendu

3

Fig. 21

6. Modification des entrées d'un tableau

Vous utiliserez l'index du tableau pour initialiser ou modifier les valeurs stockées.

Exemple

```
var fruits = new Array("Pomme", "Poire", "Abricot");  
fruits[3] = "Cerise";  
fruits[2] = "Tomate";  
for (i=0; i < fruits.length; i++) {  
    document.write(fruits[i] + "<br/>");  
}
```

Fig. 22

Résultat attendu

Pomme
Poire
Tomate
Cerise

Fig. 23

B. Méthodes couramment utilisées

Tableau n°3

CONCAT()	<p>Syntaxe tableau.concat(tableau2)</p> <p>Description Retourne la concaténation de 2 tableaux.</p> <p>Exemple <pre>var tab1=new Array(«Pommes», «Poires», «Ananas», «Cerise»); var tab2=new Array(«Banane», «Fraise», «Prune»); var result=tab1.concat(tab2); document.write(result.join(« , »));</pre> </p> <p>Résultat Pommes, Poires, Ananas, Cerise, Banane, Fraise, Prune</p>
FOREACH()	<p>Syntaxe tableau.forEach(fonction);</p> <p>Description Appelle une fonction pour chaque entrée d'un tableau. L'élément du tableau est passé en paramètre de la fonction.</p>
JOIN()	<p>Syntaxe tableau.join(separateur)</p> <p>Description Convertit un tableau en chaîne de caractères composée de tous les éléments du tableau, séparés par la chaîne séparateur.</p>
POP()	<p>Syntaxe tableau.pop()</p> <p>Description Supprime le dernier élément du tableau.</p>
PUSH()	<p>Syntaxe tableau.push(element1 [, ..., element_n])</p> <p>Description Ajoute des éléments en fin de tableau.</p>
REVERSE()	<p>Syntaxe tableau.reverse()</p> <p>Description Inverse complètement l'ordre des éléments du tableau. Le premier élément se retrouve en dernier et inversement.</p>
SHIFT()	<p>Syntaxe tableau.shift()</p> <p>Description Supprime le premier élément du tableau.</p>
SLICE()	<p>Syntaxe tableau.slice(debut, fin)</p> <p>Description Retourne une tranche de tableau composé des éléments ayant un indice supérieur ou égal à debut et strictement inférieur à fin. La tranche de tableau a donc exactement fin - debut éléments.</p>
SORT()	<p>Syntaxe tableau.sort()</p> <p>Description Trie les éléments par ordre croissant ou alphabétique croissant.</p>

SPLICE()	Syntaxe tableau.splice(debut, nb [,element1, ..., elementn]) Description Ecrase une tranche de tableau à partir de l'indice debut et sur un nombre nb d'éléments. Les paramètres element1 à elementn facultatifs remplacent la tranche ainsi définie.
UNSHIFT()	Syntaxe tableau.unshift() Description Insère des éléments en début de tableau.

C. Objet String

L'objet **String** permet la manipulation de chaînes de caractères.

Par exemple, la méthode toUpperCase() est une fonction standard de l'objet String pour afficher un texte en majuscule, la méthode write est une fonction standard de l'objet document permettant d'afficher un texte.

Exemple

```
<script type="text/JavaScript">
var texte="Bonjour à Tous!<br/>";
document.write(texte);
document.write(texte.toUpperCase());
</script>
```

Fig.24

Résultat attendu

```
Bonjour à Tous!
BONJOUR A TOUS!
```

Fig.25

D. Méthodes couramment utilisées

Tableau n°4

CHARAT()	Syntaxe chaîne.charAt(i) Description Retourne le caractère de la chaîne correspondant à la position indiquée en paramètre de charAt(). Le premier caractère d'une chaîne correspond à l'indice 0. Le dernier caractère correspond donc à l'indice longueur - 1, soit chaîne.length-1.
CHARCODEAT()	Syntaxe chaîne.charCodeAt(i) Description Retourne le code ASCII du caractère de chaînes à la position i.
FROMCHARCODE()	Syntaxe String.fromCharCode(i1, i2, ..., i99) Description Crée une chaîne à partir d'une série de code ASCII.

INDEXOF()	<p>Syntaxe chaîne.indexOf(souschaîne, debut)</p> <p>Description Retourne la position d'une sous-chaîne souschaîne dans une chaîne, à partir de la position <i>debut</i>. Si la sous-chaîne n'est pas trouvée, indexOf() retourne -1</p>
LASTINDEXOF()	<p>Syntaxe chaîne.lastIndexOf(souschaîne)</p> <p>Description Retourne l'indice de la dernière occurrence de souschaîne. Si <i>souschaîne</i> n'est pas trouvée, lastIndexOf() retourne -1.</p>
MATCH()	<p>Syntaxe chaîne.match(reg)</p> <p>Description Vérifie la concordance d'un motif d'expression régulière. Si le motif est trouvé dans chaîne, match() retourne true, sinon retourne false. Attend en paramètre une expression régulière (RegExp).</p>
REPLACE()	<p>Syntaxe chaîne.replace(motif, texte)</p> <p>Description Trouve un motif d'expression régulière dans une chaîne et remplace par le texte voulu. Replace() utilise l'objet RegExp pour définir le motif et la propriété \$1..\$9 permettant de repérer les sous-chaînes trouvées.</p>
SLICE()	<p>Syntaxe chaîne.slice(debut [,fin])</p> <p>Description Equivalent à substring()</p>
SPLIT()	<p>Syntaxe chaîne.split(motif)</p> <p>Description Retourne un tableau de sous-chaînes de caractères en utilisant comme séparateur le motif de l'expression régulière.</p>
SUBSTR()	<p>Syntaxe chaîne.substr(debut, longueur)</p> <p>Description Presque identique à substring(). La seule différence tient dans le second paramètre qui indique le nombre de caractères à extraire.</p>
SUBSTRING()	<p>Syntaxe chaîne.substring(debut [,fin])</p> <p>Description Extrait une sous-chaîne d'une chaîne en partant de l'indice debut et jusqu'à l'indice <i>fin</i>. Si <i>fin</i> n'est pas précisé, la chaîne est extraite depuis le début jusque sa fin.</p>
TOLOWERCASE()	<p>Syntaxe chaîne.toLowerCase()</p> <p>Description Retourne le texte de chaîne en minuscules.</p>
TOUPPERCASE()	<p>Syntaxe chaîne.toUpperCase()</p> <p>Description Retourne le texte de chaîne en majuscules.</p>

E. Objet Date

L'objet **Date** permet la manipulation de date et d'heures.

Dans JavaScript, toutes les dates sont calculées en millisecondes à partir du 1 janvier 1970 à 00:00:00 en Temps Universel (UCT). Ainsi 1 jour contient 86 400 000 millisecondes.

1. Création d'un objet Date

Voici les différentes façons d'initialiser un objet date.

2. Syntaxe

```
new Date() // Date et heure actuelles
new Date(millisecondes) // nombre de millisecondes depuis le 1/1/1970 à 00:00:00
new Date(string) //
new Date(année, mois, jour, minutes, secondes, millisecondes)
```

Fig. 26

La plupart des paramètres sont optionnels, chacun prend la valeur «0» s'ils sont omis.

Exemple

```
var maintenant = new Date()
var d1 = new Date("December 25, 1987 15:09:00")
var d2 = new Date(87,12,25)
var d3 = new Date(87,12,25,15,09,0)
```

Fig. 27

3. Méthodes

Les dates peuvent être manipulées en utilisant les méthodes de l'objet **Date**.

Dans l'exemple suivant, nous initialisons un objet Date 10 jours dans le futur :

```
var maDate=new Date();
maDate.setDate(maDate.getDate()+10);
```

Fig. 28

Dans cet autre exemple, nous comparons la date actuelle avec le 25 décembre 1987 :

```
var x=new Date();
x.setFullYear(1987,12,25);
var maintenant = new Date();
if (x>maintenant)
{
    alert("Hous sommes avant le 25 décembre 1987");
}
else
{
    alert("Hous sommes après le 25 décembre 1987");
}
```

Fig. 29

4. Méthodes couramment utilisées

Tableau n°5

DATE()	Syntaxe Date(); Description Retourne la date et l'heure du jour.
GETDATE()	Syntaxe ladate.getDate() Description Retourne le jour du mois.
GETDAY()	Syntaxe ladate.getDay(); Description Retourne le jour de la semaine correspondant à <i>ladate</i> . 0 correspond au dimanche et 6 correspond au samedi.
GETFULLYEAR()	Syntaxe ladate.getFullYear() Description Retourne l'année complète de la date, sur 4 chiffres.
GETHOURS()	Syntaxe ladate.getHours() Description Retourne le nombre d'heures de l'heure correspondant à <i>ladate</i> . Retourne le nombre d'heures sous forme d'entier, sans formatage. Pour écrire l'heure formatée à 2 chiffres, il faut tester par le script si l'heure contient 1 seul chiffre et rajouter un «0» devant.
GETMILLISECONDS()	Syntaxe ladate.getMilliseconds() Description Retourne les millisecondes de <i>ladate</i> . Ne pas confondre avec getTime() qui retourne le nombre de millisecondes depuis le 1 ^{er} janvier 1970.

GETMINUTES()	<p>Syntaxe <code>ladata.getMinutes()</code></p> <p>Description Retourne le nombre de minutes de l'heure correspondant à <i>ladata</i>. Retourne le nombre de minutes sous forme d'entier, sans formatage. Pour écrire les minutes formatées à 2 chiffres, il faut tester par le script si les minutes contiennent 1 seul chiffre et rajouter un «0» devant.</p>
GETMONTH()	<p>Syntaxe <code>ladata.getMonth()</code></p> <p>Description Retourne le numéro du mois correspondant à <i>ladata</i>. Attention 0 correspond au mois de Janvier et 11 au mois de Décembre. Donc pour afficher le mois, il faut ajouter 1 au retour de getMonth().</p>
GETSECONDS()	<p>Syntaxe <code>ladata.getSeconds()</code></p> <p>Description Retourne le nombre de secondes de l'heure correspondant à <i>ladata</i>. Retourne le nombre de secondes sous forme d'entier, sans formatage. Pour écrire les secondes formatées à 2 chiffres, il faut tester par le script si les secondes contiennent 1 seul chiffre et rajouter un «0» devant.</p>
GETTIME()	<p>Syntaxe <code>ladata.getTime()</code></p> <p>Description Retourne la date en millisecondes écoulées depuis le 1er janvier 1970. Cette méthode permet de faire des calculs précis de durée entre 2 événements. Cette méthode associée à setTime() permet également de trouver facilement la date correspondant à la veille ou au lendemain.</p>
GETTIMEZONEOFFSET()	<p>Syntaxe <code>ladata.getTimezoneOffset()</code></p> <p>Description Retourne la valeur du décalage horaire entre le poste du visiteur et l'heure du méridien de Greenwich en minutes.</p>
SETDATE()	<p>Syntaxe <code>ladata.setDate(jour)</code></p> <p>Description Affecte jour comme date du jour à <i>ladata</i>. L'objet Date gère lui-même les longueurs de mois, les années bissextiles, etc. La date ainsi modifiée est toujours corrigée des éventuelles incohérences et valide.</p>
SETFULLYEAR()	<p>Syntaxe <code>ladata.setFullYear(annee)</code></p> <p>Description Affecte annee comme année à <i>ladata</i>. L'objet Date gère lui-même les longueurs de mois, les années bissextiles, etc. La date ainsi modifiée est toujours corrigée des éventuelles incohérences et valide.</p>
SETHOURS()	<p>Syntaxe <code>ladata.setHours(heure)</code></p> <p>Description Affecte heure dans l'heure de la date.</p>
SETMILLISECONDS()	<p>Syntaxe <code>ladata.setMilliseconds(millisec)</code></p> <p>Description Retourne les millisecondes de la date. Ne pas confondre avec setTime() qui affecte la date avec le nombre de millisecondes depuis le 1^{er} janvier 1970.</p>

SETMINUTES()	Syntaxe <code>ladata.setMinutes(minute)</code> Description Affecte minute dans l'heure de <i>ladata</i> .
SETMONTH()	Syntaxe <code>ladata.setMonth(mois)</code> Description Affecte mois comme mois de l'année à <i>ladata</i> . Attention, janvier correspond au mois 0 et décembre au mois 11. L'objet Date gère lui-même les longueurs de mois, les années bissextiles, etc. La date ainsi modifiée est toujours corrigée des éventuelles incohérences et valide.
SETSECONDS()	Syntaxe <code>ladata.setSeconds(seconde)</code> Description Affecte seconde dans l'heure de <i>ladata</i> .
SETTIME()	Syntaxe <code>ladata.setTime(millisec)</code> Description Modifie la date à partir du temps écoulé en millisecondes depuis le 1er janvier 1970.
TOUTCSTRING()	Syntaxe <code>ladata.toLocaleString()</code> Description Retourne la date sous la forme d'une chaîne de caractères au format UTC.
TOLOCALESTRING()	Syntaxe <code>ladata.toLocaleString()</code> Description Retourne la date sous forme d'une chaîne de caractères au format local, défini dans les préférences du panneau de configuration du poste du visiteur.

F. Objet Math

L'objet **Math** fournit des constantes mathématiques (Pi, log, ...) et des méthodes permettant d'effectuer toutes les opérations mathématiques standards.

Dans l'exemple suivant, la méthode **round()** permet d'arrondir un nombre :

```
var num = 1.27;
document.write(Math.round(num));
```

Fig.30

Résultat attendu

1

Fig.31

L'exemple suivant utilise la méthode **random()** pour récupérer un nombre aléatoire entre 0 et 1 :

```
document.write(Math.random());
```

Fig.32

1. Propriétés couramment utilisées

Tableau n°6

E	Syntaxe Math.E Description Contient la valeur de la constante exponentielle. e = 2.71828...
PI	Syntaxe Math.PI Description La constante PI (environ 3.141592) est très utile dans l'utilisation des fonctions trigonométriques.
SQRT2	Syntaxe Math.SQRT2 Description Contient la valeur de «racine de 2».

2. Méthodes couramment utilisées

Tableau n°7

ABS()	Syntaxe Math.abs(x) Description Retourne la valeur absolue d'un nombre réel.
ACOS()	Syntaxe Math.acos(x) Description Retourne l'arc cosinus du nombre réel passé en paramètre.
ASIN()	Syntaxe Math.asin(x) Description Retourne l'arc sinus du nombre réel passé en paramètre.
ATAN()	Syntaxe Math.atan(x) Description Retourne l'arc tangente du nombre réel passé en paramètre.
CEIL()	Syntaxe Math.ceil(x) Description Retourne l'entier le plus proche, supérieur ou égal au réel passé en paramètre.
COS()	Syntaxe Math.cos(x) Description Retourne le cosinus du nombre réel passé en paramètre (unité : radians).
EXP()	Syntaxe Math.exp(x) Description Retourne l'exponentielle du réel donné en paramètre.
FLOOR()	Syntaxe Math.floor(x) Description Retourne l'entier le plus proche, inférieur ou égal au réel passé en paramètre.

LOG()	Syntaxe Math.log(x) Description Retourne le logarithme népérien (ln) du réel passé en paramètre.
MAX()	Syntaxe Math.max(a, b) Description Retourne le maximum parmi deux nombres passés en paramètre.
MIN()	Syntaxe Math.min(a, b) Description Retourne le minimum parmi deux nombres passés en paramètre.
POW()	Syntaxe pow(x, y) Description Retourne x à la puissance y (paramètres : x et y)
RANDOM()	Syntaxe Math.random() Description La méthode random() retourne un nombre réel aléatoire entre 0 et 1.
ROUND()	Syntaxe Math.round(x) Description Retourne l'arrondi (entier) du réel passé en paramètre.
SIN()	Syntaxe Math.cos(x) Description Retourne le sinus d'un nombre réel.
SQRT()	Syntaxe Math.sqrt(x) Description Retourne la racine carrée du nombre réel passé en paramètre. Ne pas oublier que le paramètre doit être positif ou nul.
TAN()	Syntaxe Math.tan(x) Description Retourne la tangente du nombre réel passé en paramètre (unité : radians).

G. Objet RegExp

Une **expression régulière** est un objet qui décrit la structure d'une chaîne de caractères afin de pouvoir la manipuler : recherche, récupération et remplacement d'une partie d'une chaîne de caractères.

1. Syntaxe

```
var pattern = new RegExp(pattern, modificateurs);
ou
var pattern = /pattern/modificateurs;
```

Fig.33

Un **pattern** est une expression qui décrit une suite de caractères.

Un modificateur (paramètre optionnel) spécifie si la recherche doit tenir compte de la casse, si elle est globale, etc.

2. Modificateurs

Le modificateur **i** permet d'effectuer une recherche insensible à la casse.

Exemple

```
var chaine = "j'aime les CERISES bien mûres";  
var pattern = /cerises/i;
```

Fig.34

Note : en rouge, le texte qui sera «matché» par le pattern.

Le modificateur **g** permet d'effectuer une recherche globale.

Exemple

```
var chaine = "j'aime l'acidité de cette citronnade";  
var pattern = /ci/g;
```

Fig.35

Note : en rouge, le texte qui sera «matché» par le pattern.

3. Méthode Test

La méthode **test** cherche une correspondance dans une chaîne de caractères. Renvoie la valeur *true* si trouvée, sinon la valeur *false*;

Exemple

```
var chaine = "j'aime les cerises bien mûres";  
var pattern = new RegExp("cerise");  
document.write(pattern.test(chaine));
```

Fig.36

Résultat attendu

True

Fig.37

4. Méthode Exec

La méthode **Exec** cherche une correspondance dans une chaîne de caractères et renvoie le texte correspondant ou *null*.

Exemple

```
var chaine = "j'aime les poires et les cerises bien mûres";  
var pattern = /tomate/;  
document.write(pattern.exec(chaine) + '<br/>');  
pattern = /cerise/;  
document.write(pattern.exec(chaine));
```

Fig.38

Résultat attendu

```
null  
cerise
```

Fig.39