JavaScript

Jean-Baptiste Yunès
Université Paris 7 – Denis Diderot
France

On trouvera à l'adresse suivante des exemples ainsi qu'un cours :

`http ://www.liafa.jussieu.fr/~yunes/internet/javascript/`

ECMAScript

Dernier standard (ISO) : ECMA 262 (2^{e}édition - Août 1998) accessible sur
`http ://www.ecma.ch/stand/ecma-262.htm`.

Langage de programmation basé sur JavaScript (Netscape Communications) et JScript (Microsoft Corporation).

Les deux principaux navigateurs (Netscape Communicator et Internet Explorer) supportent tout les deux ce langage qui comme complément à HTML permet de rendre les documents dynamiques.

On peut trouver la documentation en ligne de Netscape à l'adresse : `http ://devedge.netscape.com/central/javascript/`

Conformance

Un navigateur implémentant un support conforme à ECMAScript doît fournir les les objets permettant d’accéder aux différents composants (fenêtres, menus, cadres, etc.).

De plus il doît fournir des mécanismes permettant d’attacher du code ECMAScript aux différents événements provenant de l’utilisateur.

Caractéristiques principales du langage

1. comme tout langage de programmation il offre des types de base,
2. une syntaxe proche des langages de programmation impératifs comme le C,
3. implémente un modèle objet rudimentaire (et donc quelques objets de base),
4. pas de système de type,
5. pas d’entrées/sorties.

ECMAScript est un langage basé sur les objets.

Un objet est une collection de propriétés possédant des attributs.

Les propriétés sont des boîtes contenant soit des valeurs primitives soit des méthodes soit des objets.

Les valeurs primitives peuvent être prises dans les types primitifs suivant : `Undefined`, `Null`, `Boolean`, `Number` ou `String`.

Une méthode est simplement une fonction associée à un objet.

L’encapsulation de code JavaScript

On peut :

– soit utiliser la balise < SCRIPT>,

– soit utiliser un fichier externe < SCRIPT SRC="*fichier*.js">,

– soit dans un attribut de balise,

– soit en gestion d’événements.

L’interprétation du code JavaScript se fait au fur et à mesure de sa lecture dans la page ou lors du déclenchement d’événements par l’utilisateur.

Voici un exemple :

```
<html>
<head>
<title>Glups</title>
</head>

<body>
<h1>Glups</h1>
Avec un bon navigateur
<script language="JavaScript1.2">
document.writeln("(comme "+navigator.appName+")")
</script>
on peut réaliser des documents personnalisés et même dynamiques.
</body>
</html>
```

Le langage de script par défaut peut-être indiqué par l’utilisation de la balise < META http-equiv="Content-Script-Type" type="*type*">

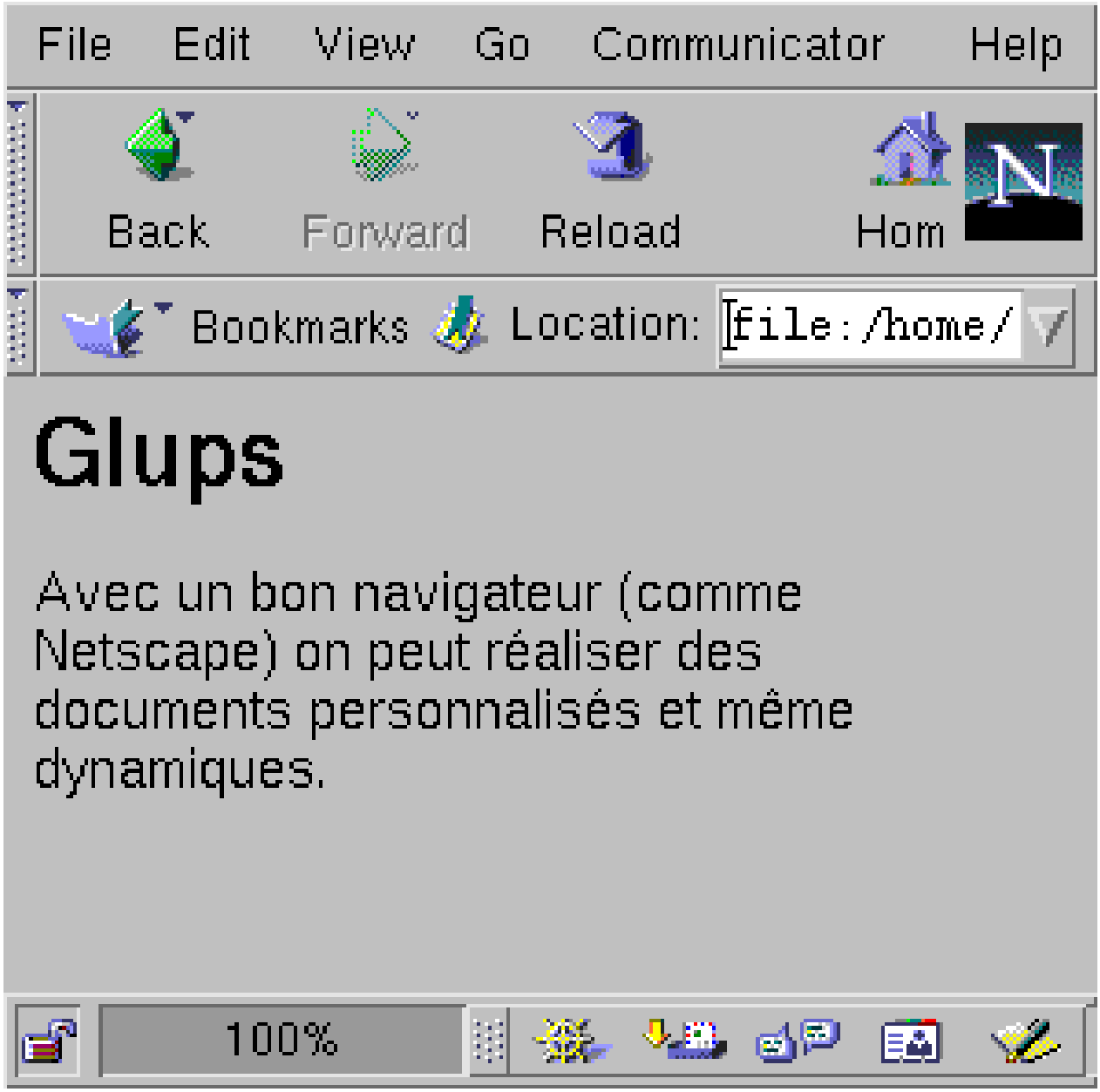

FIG. 1 – avec Netscape

FIG. 2 – avec Internet Explorer

Les objets

ECMAScript ne supporte pas la notion de classe habituelle des langages orientés objet. Les objets sont simplement créés par l'intermédiaire de `constructeurs`.

Toutes les fonctions sont des objets.

Chaque constructeur possède une propriété appelée `Prototype` utilisée pour les notions d'héritage basé sur les prototypes et les propriétés partagées.

Valeurs, Noms et Littéraux

JavaScript reconnaît les types primitifs suivants :

– les nombres : `42` ou `3.1415`

– les booléens : `true` ou `false`

– les chaînes : `"Bonjour !"`

– `null`

Il n’y a pas de distinction entre entiers et réels.

Les identificateurs doivent commencer par une lettre ou un _ suivi par des lettres, des chiffres ou des _.

Les conversions

JavaScript étant faiblement typé les variables ne sont pas associées à un type particulier. Exemple :

```
var i = 42
i = "Bonsoir"
```

Dans une expression faisant intervenir des nombres et des chaînes les nombres sont convertis en chaîne. Exemple :

```
p = "J’ai commande " + 1 + "aspirateur et " + 3 + "assiettes."
```

Portée des variables

Globale : la variable est visible depuis n'importe quel endroit.

Locale : la variable n'est visible que dans la fonction courante. Pour déclarer une variable locale à une fonction il faut utiliser le mot-clé `var`.

Les littéraux

Les entiers

Les entiers peuvent être exprimés en base 8, 10 ou 16.

S'il commence par `0` sa base est 8.

S'il commence par `0x` ou `0X` sa bse est 16.

Sinon sa base est 10.

Les réels

Utilisent `.` pour la virgule et `e` ou `E` pour la partie exposant.

Les chaînes

On peut utiliser `"` ou `'` pour délimiter une chaîne. Quelques caractères spéciaux peuvent être utilisés à l'intérieur :

– `\b` pour un espacement arrière,
– `\f` pour un saut de page,
– `\n` pour un saut de ligne,
– `\r` pour un retour au début de ligne,
– `\t` pour une tabulation.

Opérateurs

Opérateurs d'affectation

Une affectation permet de donner la valeur d'une expression à une variable. L'opérateur de base est =. Exemple : `x = y + 3`. Les autres sont :

`a+=b` équivalent à `a=a+b`

`a-=b` équivalent à `a=a-b`

`a*=b` équivalent à `a=a*b`

`a/=b` équivalent à `a=a/b`

`a%=b` équivalent à `a=a%b`

`a<<=b` équivalent à `a=a<< b`

`a>>=b` équivalent à `a=a>> b`

`a>>>=b` équivalent à `a=a>>> b`

`a&=b` équivalent à `a=a&b`

`aˆ=b` équivalent à `a=aˆb`

`a|=b` équivalent à `a=a|b`

Opérateurs arithmétiques

On trouve les opérateurs classiques `+`, `-`, `/` et `*`.

L’opérateur de modulo calcule le reste de la division entière de deux nombres il se note %.

Les incrémentations ++ et - -

La négation unaire -

Opérateurs bit-à-bit

Ceux-ci traitent les opérandes comme des champs de bits.
Ils fonctionnent ainsi :

– les opérandes sont converties en mots de 32 bits,
– chaque bit de la première opérande est apparié avec le bit correspondant de la seconde,
– l’opérateur est appliqué à chaque paire de bit et donne le bit correspondant du résultat.

Les opérateurs sont les suivants :

& calcule la fonction ET,

| calcule la fonction OU,

ˆ calcule la fonction OU-EXCLUSIF.

<< décalage vers la gauche.

>> décalage vers la droite avec préservation du signe.

>>> décalage vers la droite sans préservation du signe.

Les opérateurs logiques

Ceux-ci sont “lazy”.

&& c’est le ET logique

|| le OU logique

! le NON logique

Les opérateurs de comparaison

== teste l’égalité

!= teste l’inégalité

> plus grand que

< plus petit que

> = plus grand ou égal

< = plus petit ou égal

Les opérateurs sur les chaînes

+ concaténation

+= concaténation et affectation.

Les opérateurs spéciaux

`? :` (expressions conditionnelles). La forme est `condition? val1 : val2`. Lorsque la condition est vraie la valeur de cette expression est `val1`, sinon `val2`. Exemple : `mm=(age>=18)?"majeur" :"mineur"`

`,` permet le séquençage d'expressions. La valeur renvoyée est celle de la seconde. Sa forme est *expr1*`,` *expr2*.

`new` permet de créer un objet.

`typeof` permet de déterminer le type de l'opérande.

Syntaxe : `typeof` *expression* ou `typeof (`*expression*`)`.

Par exemple : `typeof("coucou")` vaut `"string"` ou `typeof(true)` vaut `"boolean"`

`void` permet de spécifier une expression à évaluer sans qu'elle ne retourne de valeur. Syntaxe : `javascript :void(`*expression*`)`.

Les instructions

Les commentaires

Deux formes possibles :

```
// commentaire jusqu'en fin de ligne
/* commentaire jusqu'à */
```

La déclaration de variables

Syntaxe : `var` *nom1 [= valeur1] [..., nomN [= valeurN]]*. Par exemple :

```
var i = 0, j = 2
```

La déclaration de fonctions

Existe sous deux formes syntaxiques possibles :

`function` *nom*`()` { *instructions* }

`function` *nom*`(`*param1*`,...,`*paramN*`)` { *instructions* }

Par exemple :

```
function f(a,b,c) {
  var i = 1
}
```

L'instruction `return`

Permet de sortir d'une fonction en cours en retournant une valeur à l'appelant.

Deux formes syntaxiques :

– `return`

– `return` *expression*

La boucle `do ... while`

Permet de réaliser un bloc d'instructions tant qu'une certaine condition reste vraie.

La forme est : `do` *instruction-ou-bloc* `while(`*condition*`).`

Exemple :

```
var s = 0, i = 0
do {
   s += i++
} while ( i <= 10 )
```

La boucle `for`

Réalise l'itération d'un bloc d'instructions.

La syntaxe est :

`for (`*initialisation*`;`*condition*`;`*incrémentation*`)` *instruction-ou-bloc*.

Exemple :

```
for (var s=0,i=0; i<=10; i++) s += i
```

La boucle `for ... in`

Itère un bloc d'instruction sur l'ensemble des propriétés d'un objet.

Syntaxe : `for (`*variable* `in` *objet*`)` *instruction-ou-bloc*

Exemple :

```
document.writeln("<H1>Les propriétés de mon navigateur</P>");
for (var i in navigator)
  document.writeln("<P>"+i+"="+navigator[i]+"</P>")
```

qui produit le résultat :

Les propriétés de mon navigateur

```
userAgent=Mozilla/4.76 [en] (X11; U; SunOS 5.6 sun4u)
appCodeName=Mozilla
appVersion=4.76 [en] (X11; U; SunOS 5.6 sun4u)
appName=Netscape
language=en
platform=SunOS5.5.1
securityPolicy=US & CA domestic policy
plugins=[object PluginArray]
mimeTypes=[object MimeTypeArray]
```

Le branchement multiple `switch`

Permet d'exécuter des instructions sélectionnés par la valeur d'une expression.

Par exemple :

```
switch(i) {
case 1:
  // qqe chose
  break
case 2:
  // autre chose
  break
default:
  // encore autre chose
  break
}
```

La boucle `while`

Tant qu'une certaine condition est vérifiée une instruction ou un bloc est exécuté.

Syntaxe : `while` (*condition*) *instruction-ou-bloc*

Les étiquettes

Permettent de nommer des blocs d'instructions.

Syntaxe : *étiquette* `:` *bloc*

L'instruction `break`

Permet d'interrompre la boucle `for` ou `while` ou le `switch` en cours ou le bloc d'instruction nommé en transférant le contrôle à l'instruction qui suit immédiatement la boucle ou le bloc.

Les deux formes syntaxiques sont : `break` et `break <étiquette>`.

L'instruction `continue`

Permet de recommencer la boucle `for` ou `while` en cours ou le bloc d'instruction nommé en transfèrant le contrôle à la première instruction de la boucle ou du bloc.

Les deux formes syntaxiques sont : `continue` et `continue < étiquette>`.

L'instruction `delete`

Permet de supprimer une propriété d'un objet, ou un élément dans un tableau.

Trois formes sont possibles :

– `delete` *objet*`.`*propriété*

– `delete` *objet*`[`*indice*`]`

– `delete` *propriété* forme disponible avec `with`

Par exemple :

```
t = new Array(1,2,3)
for (var i=0; i<t.length; i++) {
  document.writeln(t[i]);
}
delete t[1];
document.writeln("<P>");
for (var i=0; i<t.length; i++) {
  document.writeln(t[i]);
}
```

Résultat :

```
1 2 3
1 undefined 3
```

Le test `if ... else`

Réalise une instruction conditionnelle.

Deux formes syntaxiques possibles :

– `if (`*condition*`)` *instruction-ou-bloc*

– `if (`*condition*`)` *instruction-ou-bloc1* `else` *instruction-ou-bloc2*

L'instruction `export`

Permet de rendre visible certains objets, fonctions ou propriétés dans un autre script.

L'instruction `import`

Permet de "voir" des propriétés, objets ou fonctions d'un autre script.

Le nommage par défaut `with`

Permet de spécifier que les identificateurs se réfèrent d'abord à ceux d'un objet particulier.

Syntaxe : `with (`*objet*`) {` *instruction-ou-bloc* `}`

Le modèle objet

Objets et propriétés

Chaque objet possède un ensemble de propriétés. La notation pour accéder à l'une d'entre elle est : `objet.propriété`.

Pour définir une propriété il suffit de lui associer une valeur :

```
personne = new Object();
personne.nom    = "Dupont"
personne.prenom = "Jean"
```

Il est aussi possible d'utiliser la notation "tableau" :

```
personne = new Object();
personne["nom"]    = "Dupont"
personne["prenom"] = "Jean"
```

En fait il s'agit d'un tableau d'association (chaque indice est associé à une chaîne) :

```
function toutvoir(o,s) {
  var result = ""

  for (var i in o)
    result += "<P>" + s + "." + i + " = " + o[i]
  return result
}
```

l'appel à `toutvoir(personne,"p")` produit le résultat :

```
p.nom = Dupont
p.prenom = Jean
```

Fonctions

Une fonction définit une entité de calcul. La syntaxe est :

```
function <nom> ( <liste de paramètres> ) {
  <suite d'instructions>
}
```

Exemple :

```
function somme(n) {
  var r = 0

  for (var i = 1; i <= n; i++)
    r += i
  return r
}
document.writeln("<P>" + somme(10));
```

produit : `55`.

Les arguments d'une fonction sont accessibles à l'intérieur de celle-ci en utilisant la propriété `arguments` qui est un tableau. Le premier argument est d'indice `0` et il y en a `arguments.length` :

```
function voirArguments() {
  var r = ""

  for (var i=0; i < voirArguments.arguments.length; i++)
    r += "Arg[" + i + "]=" + voirArguments.arguments[i] + " (de type " +
         typeof(voirArguments.arguments[i]) + ")<BR>"
  return r
}
document.writeln("<P>" + voirArguments("un","deux","trois",4));
```

produit :

```
Arg[0]=un (de type string)
Arg[1]=deux (de type string)
Arg[2]=trois (de type string)
Arg[3]=4 (de type number)
```

Méthodes

Une méthode est une fonction associée à un objet.

Pour réaliser cette association il suffit d'utiliser la syntaxe suivante : `< objet>.< méthode> = < fonction>`.

Le mot-clé `this` permet de faire référence à l'objet courant.

Exemple :

```
function observe() {
  var result = ""

  for (var i in this)
    result += "<P>" + i + " = " + this[i]
  return result
}
p.maFonction = observe;
document.writeln(p.maFonction());
```

dont le résultat est :

```
p.nom = Dupont
p.prenom = Jean
p.maFonction = function observe(s) { var result = ""; for (var i in this)
{ result += "<P>" + s + "." + i + " = " + this[i]; } return result; }
```

La création d’objets

Bien que de nombreux objets soient définis par l’environnement d’exécution, il est évidemment possible de créer de nouveaux objets. Pour cela il faut :

– définir un type d’objets en créant une fonction correspondante,
– créer une instance de ce type en utilisant new

```
function personne(nom,prenom) {
  this.nom = nom;
  this.prenom = prenom;
}
p = new personne("Dupont","Jean")
```

Il est évidemment possible de créer autant d’objets que l’on désire.

Une propriété d’un objet peut être un autre objet. Exemple :

```
function bureau(numero,proprietaire) {
  this.numero       = numero
  this.proprietaire = proprietaire
}

p = new personne("Dupont","Jean");
b = new bureau("117",p);
document.writeln("<P>"+toutvoir(p,"p"));
document.writeln("<P>"+toutvoir(b,"b"));
```

produit :

```
p.nom = Dupont
p.prenom = Jean
```

```
b.numero = 117
b.proprietaire = [object Object]
```

D’autre part on peut aussi rajouter dynamiquement des propriétés aux objets à tout instant :

```
p = new personne("Dupont","Jean");
b = new bureau("117",p);
document.writeln("<P>"+toutvoir(p,"p"));
document.writeln("<P>"+toutvoir(b,"b"));
p2 = new personne("Durand","Jeanne");
p2.sexe = "F";
```

produit :

```
p.nom = Dupont
p.prenom = Jean

b.numero = 117
b.proprietaire = [object Object]

p2.nom = Durand
p2.prenom = Jeanne
p2.sexe = F
```

Attention : cela ne permet que d’ajouter des propriétés à une instance particulière... Pour ajouter une propriété à tous les objets (existants ou futurs) d’une classe il faut ajouter une propriété à la propriété `prototype`.

Définition de méthode

On peut maintenant définir des méthodes pour l'objet :

```
function voirBureau() {
  return "Le bureau numéro " + this.numero +
         " est affecté à " + this.proprietaire.affiche()
}
function voirPersonne() {
  return this.prenom + " " + this.nom
}

b = new bureau(117,p);
bureau.prototype.affiche = voirBureau
personne.prototype.affiche = voirPersonne
document.writeln("<P>"+b.affiche());
```

produit :

```
Le bureau numéro 117 est affecté à Jean Dupont
```

Héritage en JavaScript

L'héritage est construit par chaînage des prototypes. Par exemple :

```
function A(n) {
  this.a = n
}
function B(m) {
  this.b = m
}
B.prototype = new A(1)
a = new A("a")
b = new B("b")
document.writeln(toutvoir(a,"a"))
document.writeln(toutvoir(b,"b"))
A.prototype.c = "c"
document.writeln(toutvoir(a,"a"))
document.writeln(toutvoir(b,"b"))
```

produit :

```
a.a = a

b.b = b
b.a = 1

a.a = a
a.c = c

b.b = b
b.a = 1
b.c = c
```

Objets prédéfinis et fonctions

On trouve dans le cœur de JavaScript les objets prédéfinis suivants :

```
Array      Boolean   Date
Function   Math      Number
Objet      String    RegExp
```

Les objets `Array`

Leurs constructeurs :

```
Array(n)
Array(élément0, ... ,élémentN-1)
```

Leurs propriétés :

```
index       input   length
prototype
```

Leurs méthodes :

```
concat    join       pop       push
reverse   shift      slice     splice
sort      toString   unshift
```

Exemple :

```
tableau = new Array("Vive","les","vacances");
document.writeln("<P>"+tableau.join());
document.writeln("<P>"+tableau.join(" "));
document.writeln("<P>"+tableau.slice(1,2));
tableau.splice(0,1,"A","bas");
document.writeln("<P>"+tableau.join(" "));
```

produit :

```
Vive,les,vacances
Vive les vacances
les
A bas les vacances
```

Les objets `Boolean`

Leur constructeur : `Boolean(valeur)`.

Leur propriété : `prototype`.

Leur méthode : `toString`.

Les objets `Date`

Leurs constructeurs :

`Date()`

`Date("mois jour, année heure :minutes :secondes")`

`Date(année,mois,jour)`

`Date(année,mois,jour,heure,minutes,secondes)`

Leur propriété : `prototype`.

Leurs méthodes :

getDate	getDay	getHours
getMinutes	getMonth	getSeconds
getTime	getTimezoneOffset	getYear
parse	setDate	setHours
setMinutes	setMonth	setSeconds
setTime	setYear	toGMTString
toLocaleString	UTC	

Exemple :

```
date = new Date()
document.writeln("<P>"+date)
document.writeln("<P>"+date.toGMTString())
function toFrench() {
  var date = new Date();
  var s = ""
  var mois = new Array("Janvier","Février","Mars","Avril","Mai","Juin",
                       "Juillet","Août","Septembre","Octobre","Novembre",
                       "Décembre")
  var jours = new Array("Dimanche","Lundi","Mardi","Mercredi","Jeudi",
                        "Vendredi","Samedi")
  return jours[date.getDay()] + " " + date.getDate() + " " + mois[date.getMonth()]
}
document.writeln("<P>"+toFrench());
```

Résultat :

```
Fri Nov 26 18:38:00 GMT+0100 (MET) 1999
Fri, 26 Nov 1999 17:38:00 GMT
Vendredi 26 Novembre
```

Les objets `Function`

Les fonctions ainsi créées ne possèdent pas de nom. Elles doivent donc être référencées par une variable.

Leur constructeur :

`Function(arg1, ... ,argN,corps)`.

Leurs propriétés :

`arguments` `arity` `caller` `prototype`

Leur méthode : `toString`.

Exemple :

```
var plus = new Function("a","b","var i = a+b; return i")
document.writeln("<P>"+plus(3,4))
document.writeln("<P>"+plus.toString())
```

produit :

```
7
function anonymous(a, b) { var i = a + b; return i; }
```

L'objet `Math`

C'est un objet prédéfini de JavaScript. Ce n'est pas une classe et ne peut donc être instanciée. On peut considérer cet objet comme une bibliothèque ou collection de fonctions et constantes.

Ses propriétés (constantes) :

```
E       LN10   LN2       LOG10E
LOG2E   PI     SQRT1_2   SQRT2
```

Ses méthodes (fonctions utilitaires) :

```
abs     acos     asin    atan
atan2   ceil     cos     exp
floor   log      max     min
pow     random   round   sin
sqrt    tan
```

```
for (i=0; i<10; i++)
  document.write(Math.floor(Math.random()*100) + " ")
```

produit :

```
3 95 89 69 37 43 66 47 79 81
```

Les objets `Number`

Leur constructeur : `Number(valeur)`.

Leurs propriétés :

```
MAX_VALUE            MIN_VALUE            NaN
NEGATIVE_INFINITY    POSITIVE_INFINITY    prototype
```

Leur méthode : `toString`.

Exemple :

```
var n = new Number(111e3);
n += 444
document.writeln("<P>MIN="+Number.MIN_VALUE+" MAX="+Number.MAX_VALUE+" n="+n);
```

produit :

```
MIN=5e-324 MAX=1.7976931348623157e+308 n=111444
```

Les objets `Object`

Leur constructeur : `Object()`

Leurs propriétés :

`constructor` `prototype`

Leurs méthodes :

`eval` `toString` `unwatch`

`valueOf` `watch`

Exemple :

```
function oeil(p,o,n) {
  document.writeln("<P>" + p + " qui valait " + o + " vaut maintenant " + n +
     " qui est de type " + typeof(n) );
  return n
}
o = new Object()
o.prop = 3
o.watch("prop",oeil)
o.prop = 4
o.prop = "cuicui"
```

produit :

```
prop qui valait 3 vaut maintenant 4 qui est de type number
prop qui valait 4 vaut maintenant cuicui qui est de type string
```

Autre exemple :

```
function somme(n) {
  this.n = n
}
function doIt() {
  for (var s=0,i=0; i<=this.n; i++) s += i
  return "(somme de 1 à "+this.n+"="+s+")"
}
n = new somme(10)
somme.prototype.valueOf = doIt
a = n
document.writeln("a="+a+" n.n="+n.n+" n="+n)
```

produit :

```
a=(somme de 1 à 10=55) n.n=10 n=(somme de 1 à 10=55)
```

Les objets `String`

Il existe un objet prédéfini de nom `String`.

Ils possèdent les propriétés suivantes :

`length` `prototype`

et les méthodes suivantes :

`anchor`	`big`	`blink`
`bold`	`charAt`	`charCodeAt`
`concat`	`fixed`	`fontcolor`
`fontsize`	`fromCharCode`	`indexOf`
`italics`	`lastIndexOf`	`link`
`match`	`replace`	`search`
`slice`	`small`	`split`
`strike`	`sub`	`substr`
`substring`	`sup`	`toLowerCase`
`toUpperCase`		

Exemple :

```
texte = "Youpi!!!"
hrefTexte = "http://www.youpi.com/"
document.writeln("Cliquez pour aller sur " + texte.link(hrefTexte))
```

produit la chaîne de caractères suivante :

```
Cliquez pour aller sur < A HREF="http ://www.youpi.com/"> Youpi ! ! !< /A>
```

Les objets `RegExp`

Leur constructeur : `RegExp("motif","options")`.

Leurs propriétés :

`$1, ... ,$9`	`$_`	`$*`
`$&`	`$+`	`$``
`$´`	`global`	`ignoreCase`
`input`	`lastIndex`	`lastMatch`
`lastParen`	`leftContext`	`multiline`
`rightContext`	`source`	

Leurs méthodes :

`compile`	`exec`	`test`

Exemple :

```
var s = "une CHAîNE de caractères toute neuve."
var re = new RegExp("ne\\b","gi");
while (true) {
  res = re.exec(s);
  if ( res == null ) break;
  document.writeln("<P>J'ai trouvé \""+res[0]+"\" en position "+
                   res.index+" dans \""+res.input+"\". ");
  document.writeln("Je l'ai trouvé entre ["+RegExp.leftContext+"] et ["+
```

```
                    RegExp.rightContext+"]");
}
```

produit :

```
J’ai trouvé "ne" en position 1 dans "une CHAîNE de caractères toute
neuve!". Je l’ai trouvé entre [u] et [ CHAîNE de caractères toute neuve!]

J’ai trouvé "NE" en position 8 dans "une CHAîNE de caractères toute
neuve!". Je l’ai trouvé entre [une CHAî] et [ de caractères toute neuve!]
```

Autre exemple :

```
re = /(\w+)\s(\w+)/;
str = "Jean Dupont";
newstr=str.replace(re, "Nom: $2 Prénom: $1");
document.writeln("<P>"+newstr)
```

produit :

```
Nom: Dupont Prénom: Jean
```

Le document

L'objet `document`

L'objet `document` est créé indirectement par la balise HTML `< BODY>`.

Ses propriétés sont :

`alinkColor`	`anchors`	`applets`
`bgColor`	`cookie`	`domain`
`embeds`	`fgColor`	*formulaire*
`forms`	`images`	`lastModified`
`layers`	`linkColor`	`links`
`plugins`	`referrer`	`title`
`URL`	`vlinkColor`	

Ses méthodes :

`captureEvents`	`close`	`getSelection`
`handleEvent`	`open`	`releaseEvents`
`routeEvent`	`write`	`writeln`

Ses gestionnaires d'événements :

onClick onDblClick onKeyDown

onKeyPress onKeyUp onMouseDown

onMouseUp

Exemple :

```
<a href="ftp://ftp.lip6.fr/pub/">FTP LIP6</a>
<a href="#toto">voir plus bas</a>
<a name="toto">ici</a>
<script>
for (var i=0; i<document.anchors.length; i++)
  document.writeln("<P>" + toutvoir(document.anchors[i],"ancre"))
for (var i=0; i<document.links.length; i++)
  document.writeln("<P>" + toutvoir(document.links[i],"lien"))
</script>
```

produit :

```
ancre.text = ici
ancre.name = toto
ancre.x = 160
ancre.y = 8
lien.href = ftp://ftp.lip6.fr/pub/
lien.protocol = ftp:
lien.host = ftp.lip6.fr
lien.hostname = ftp.lip6.fr
lien.port =
lien.pathname = /pub/
lien.hash =
lien.search =
lien.target = null
lien.text = FTP LIP6
lien.x = 8
lien.y = 8
lien.href = file:essai.html#toto
lien.protocol = file:
lien.host =
lien.hostname =
lien.port =
lien.pathname = essai.html
```

```
lien.hash = #toto
lien.search =
lien.target = null
lien.text = voir plus bas
lien.x = 73
lien.y = 8
```

Ou encore :

```
document.writeln("Vous avez obtenu ce document en cliquant sur un lien contenu dans le document "+document.referrer+". Vous pouvez y retourner en cliquant "+"ici".link(document.referrer))
```

L’objet `Link` et `Area`

Ses propriétés :

`hash`	`host`	`hostname`
`href`	`pathname`	`port`
`protocol`	`search`	`target`
`text`		

Sa méthode : `handleEvent`.

Ses gestionnaires d’événements :

`onClick`	`onDblClick`	`onKeyDown`
`onKeyPress`	`onKeyUp`	`onMouseDown`
`onMouseOut`	`onMouseUp`	`onMouseOver`

Pour les objets `Area` :

`onClick`	`onDblClick`	`onKeyDown`
`onKeyPress`	`onKeyUp`	`onMouseDown`
`onMouseOut`	`onMouseUp`	`onMouseOver`

L’objet `Anchor`

Ne possède ni propriétés ni méthodes.

L’objet `Image`

Son constructeur : `Image(`*largeur*`,`*hauteur*`)`.

Attention les images ”visibles” ne peuvent être créées qu’en utilisant la balise `< IMG>`, ce qui implique par ailleurs que leur position ne peut être modifiée. Les images créées par construction sont simplement chargées et décodées afin de pouvoir être placés dans un récipient créé par la balise `< IMG>`.

Ses propriétés :

`border`	`complete`	`height`
`hspace`	`lowsrc`	`name`
`prototype`	`src`	`vspace`
`with`		

Son unique méthode : `handleEvent`.

Ses gestionnaires d’événements :

`onAbort`	`onError`	`onKeyDown`
`onKeyPress`	`onKeyUp`	`onLoad`

Exemple :

```
<html>
<head>
<script>
var mesImages = new Array(5);
for (var i=0; i<mesImages.length; i++) {
  mesImages[i] = new Image();
  mesImages[i].src = "img"+i+".gif"
}
function charge(m) {
  document.images["laVraie"].src = mesImages[m.value].src;
}
</script>
</head>
<body>
<img src="img.gif" name="laVraie">
<form>
<input type="button" name="b0" value="0" onClick="charge(this)">
<input type="button" name="b1" value="1" onClick="charge(this)">
<input type="button" name="b2" value="2" onClick="charge(this)">
<input type="button" name="b3" value="3" onClick="charge(this)">
<input type="button" name="b4" value="4" onClick="charge(this)">
</form>
</body>
</html>
```

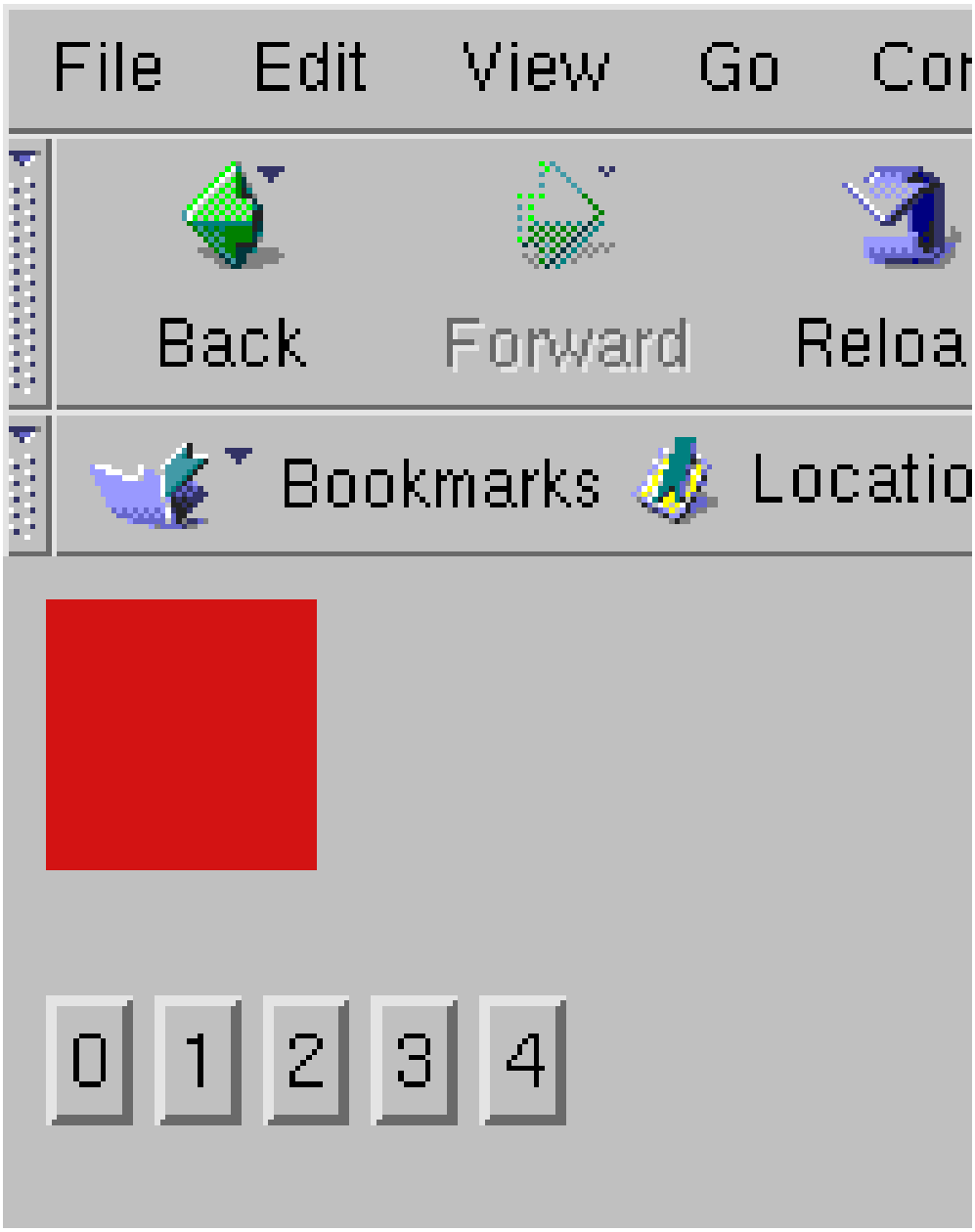

FIG. 3 – après le chargement

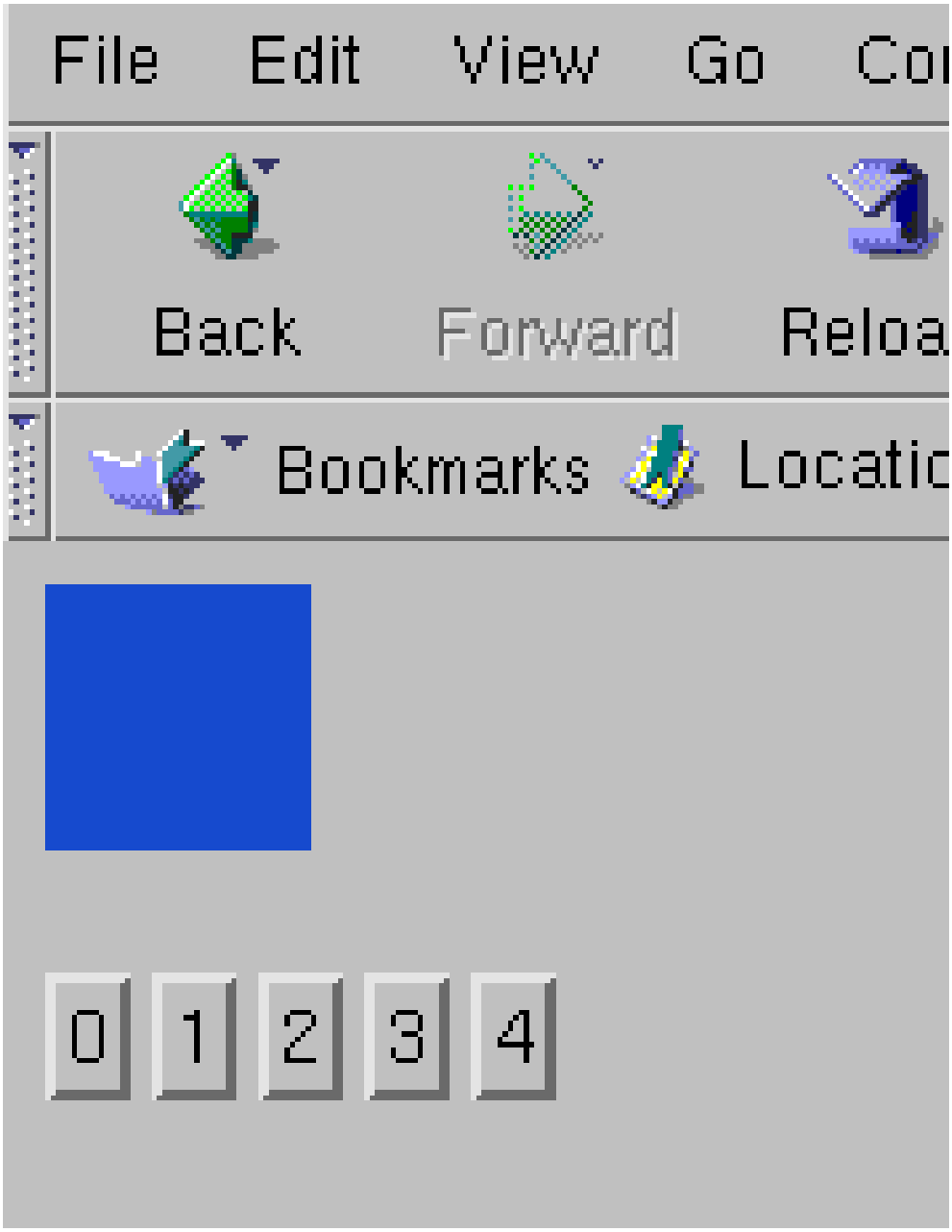

FIG. 4 – après la sélection 0

L’objet `Applet`

Les propriétés et les méthodes `public` de l’applet sont les propriétés et méthodes de l’objet de type `Applet`.

Exemple (JavaScript) :

```
<head>
<script>
function something() {
  alert(document.applets["App"].f())
}
</script>
</head>
<body>
<applet name="App" code="App.class" width="100" height="100"></applet>
<form>
<input type="button" name="b5" value="ok" onClick="something()">
</form>
</body>
```

avec l’exemple d’une Applet Java :

```
import java.applet.*;

public class App extends Applet {
    public String f() {
return "coucou";
    }
}
```

L’objet `Layer`

Ses propriétés :

above	background	bgColor
below	clip.bottom	clip.height
clip.left	clip.right	clip.top
clip.width	document	left
name	pageX	pageY
parentLayer	siblingAbove	siblingBelow
src	top	visibility
zIndex		

Ses méthodes :

captureEvents	handleEvent	load
moveAbove	moveBelow	moveBy
moveTo	moveToAbsolute	releaseEvents
resizeBy	resizeTo	routeEvent

Ses gestionnaires d’événements :

onMouseOver	onMouseOut	onLoad
onFocus	onBlur	

La fenêtre

L’objet `Window` et `Frame`

Ses propriétés :

`closed`	`defaultStatus`	`document`
`frames`	`history`	`innerHeight`
`innerWidth`	`length`	`location`
`locationBar`	`menuBar`	`name`
`opener`	`outerHeight`	`outerWidth`
`pageXOffset`	`pageYOffset`	`parent`
`personnalbar`	`scrollbars`	`self`
`status`	`statusbar`	`toolbar`
`top`	`window`	

Ses méthodes :

```
alert              back               blur
captureEvents      clearInterval      clearTimeout
close              confirm
disableExternalCapture
enableExternalCapture
find               focus              forward
handleEvent        home               moveBy
moveTo             open               print
prompt             releaseEvents      resizeBy
resizeTo           routeEvent         scroll
scrollBy           scrollTo           setInterval
setTimeout         stop
```

Ses gestionnaires d’événements :

```
onBlur     onDragDrop   onError
onFocus    onLoad       onMove
onResize   onUnload
```

Un exemple :

```
<html><head><script>
var cto, ctof = false

function popUp(e,num) {
  if (ctof) window.clearTimeout(cto)
  var current = document.layers["lay"+num];
  current.moveTo(e.pageX-10,e.pageY-10)
  current.visibility = "show"
  cto = window.setTimeout(popDown,5000,e,current)
  ctof = true
  return true;
}

function popDown(e,l) {
  l.visibility = "hide"
  if (ctof) {
    window.clearTimeout(cto)
    ctof = false
  }
  return true
}

function init(e,l) {
  l.visibility = "hide"
  l.bgColor = "FFFFFF"
}
</script>
<title>Glups</title>
</head><body>
<layer id=lay2 onLoad="init(event,this);"
 visibility="hide" src="essai2.html"
 onMouseOut="return popDown(event,this);">
</layer>
<layer id=lay3 onLoad="init(event,this);"
 visibility="hide" src="essai3.html"
 onMouseOut="return popDown(event,this);">
</layer>
<A NAME="toto" HREF="javascript:void(0)"
 onMouseOver="return popUp(event,2);">Essai</A><BR>
<A NAME="titi" HREF="javascript:void(0)"
 onMouseOver="return popUp(event,3);">Essai2</A>
</body></html>
```

```
<head><script>
function arriere() {
  ret = window.confirm("Êtes-vous sûr de vouloir revenir en arrière ?")
  if (ret) window.back()
}
</script></head>
<body>
<input type="button" name="b6" value="arrière" onClick="arriere()">
</body>
```

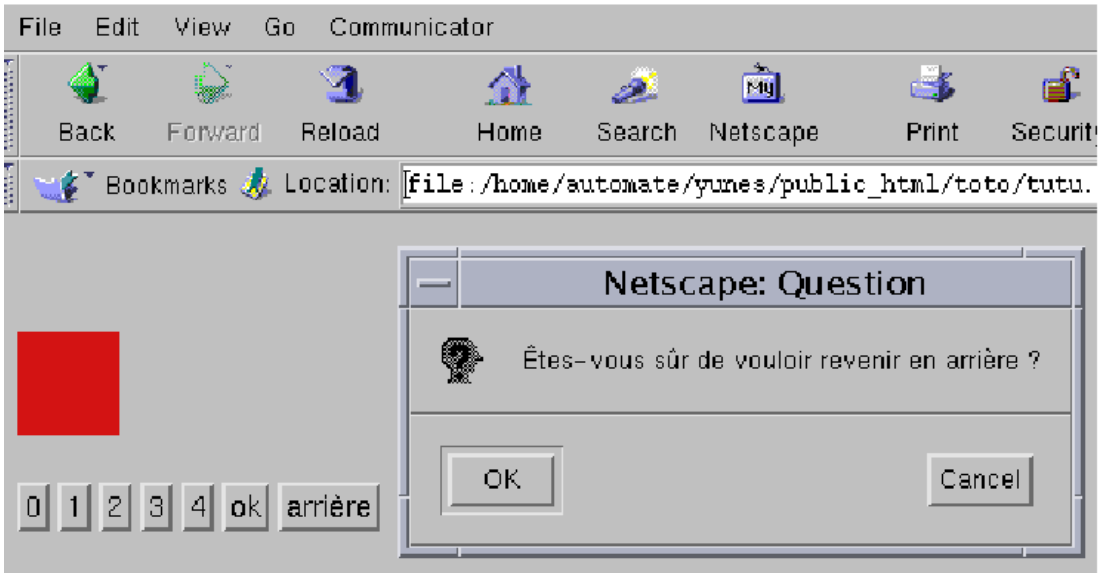

FIG. 5 – après clic sur “arrière”

L’objet `Location`

Ses propriétés :

`hash`	`host`	`hostname`
`href`	`pathname`	`port`
`protocol`	`search`	

Ses méthodes :

```
reload   replace
```

Les différents types d'URL :

Protocole	Example
`javascript :`	`javascript :history.go(-1)`
`view-source :`	`view-source :`*URL*
`about :`	`about :cache`
`http :`	`http ://www.w3.org/`
`file :/`	`file :///home/toto/truc.html`
`ftp :`	`ftp ://ftp.lip6.fr/pub/`
`mailto :`	`mailto :toto@aol.fr`
`news :`	`news ://`*serveur*`/`*forum*
`gopher :`	`gopher ://`*URL*

L’objet `History`

Ses propriétés :

`current` `length` `next`

`previous`

Ses méthodes :

`back` `forward` `go`

L’objet `screen`

Ses propriétés :

`availHeight` `availWidth` `colorDepth`

`height` `pixelDepth` `width`

Les formulaires

L'objet `Form`

Ses propriétés :

`action` `elements` `encoding` `length`

`method` `name` `target`

Ses méthodes :

`handleEvent` `reset` `submit`

Ses gestionnaires d'événements :

`onReset` `onSubmit`

L'objet `Hidden`

Ses propriétés :

`form` `name` `type` `value`

L'objet `Text`

Ses propriétés :

`defaultValue` `form` `name`

`type` `value`

Ses méthodes :

```
blur   focus   handleEvent   select
```

Ses gestionnaires d'événements :

```
onBlur   onChange   onFocus   onSelect
```

L'objet `Textarea`

Ses propriétés :

```
defaultValue   form    name
type           value
```

Ses méthodes :

```
blur   focus   handleEvent   select
```

Ses gestionnaires d'événements :

```
onBlur      onChange     onFocus
onKeyDown   onKeyPress   onKeyUp
onSelect
```

L'objet `Password`

Ses propriétés :

```
defaultValue   form    name
type           value
```

Ses méthodes :

`blur` `focus` `handleEvent` `select`

Ses gestionnaires d'événements :

`onBlur` `onFocus`

L'objet `FileUpload`

Ses propriétés :

`form` `name` `type` `value`

Ses méthodes :

`blur` `focus` `handleEvent` `select`

Ses gestionnaires d'événements :

`onBlur` `onChange` `onFocus`

L'objet `Button`

Ses propriétés :

`form` `name` `type` `value`

Ses méthodes :

`blur` `click` `focus` `handleEvent`

Ses gestionnaires d'événements :

```
onBlur        onClick     onFocus
onMouseDown   onMouseUp
```

L’objet `Submit`

Ses propriétés :

```
form   name   type   value
```

Ses méthodes :

```
blur   click   focus   handleEvent
```

Ses gestionnaires d’événements :

```
onBlur   onClick   onFocus
```

L’objet `Reset`

Ses propriétés :

```
form   name   type   value
```

Ses méthodes :

```
blur   click   focus   handleEvent
```

Ses gestionnaires d’événements :

```
onBlur   onClick   onFocus
```

L’objet `Radio`

Ses propriétés :

```
checked   defaultChecked   form
name      type             value
```

Ses méthodes :

```
blur          click   focus
handleEvent
```

Ses gestionnaires d’événements :

```
onBlur   onClick   onFocus
```

L’objet `Checkbox`

Ses propriétés :

```
checked   defaultChecked   form
name      type             value
```

Ses méthodes :

```
blur          click   focus
handleEvent
```

Ses gestionnaires d’événements :

`onBlur` `onClick` `onFocus`

L'objet `Select`

Ses propriétés :

`form` `length` `name`

`options` `selectedIndex` `type`

Ses méthodes :

`blur` `focus`

`handleEvent`

Ses gestionnaires d'événements :

`onBlur` `onChange` `onFocus`

L'objet `Option`

Son constructeur : `Option(`*texte*`,`*valeur*`,`*état-par-défaut*`,`*état*`)`.

Ses propriétés :

`defaultSelected` `selected` `text`

`value`

Un exemple :

```
<html>
<head>
<title>PIZZA Express</title>
<script language="JavaScript1.2">
var max = 10

function bonneQuantite(o) {
  var qte = parseInt(o.value)
  return qte >0 && qte <= max
}

function verifie(o) {
  if (bonneQuantite(o)) return true
  alert("Vous êtes trop gourmand!")
  return false
}

function verifieEtCommande(o) {
  if ( bonneQuantite(o.np) ) {
    return confirm("Voulez-vous vraiment commander "+o.np.value+" pizzas ?")
  }
  return false
}
</script>
</head>
<body>
<h1>Bienvenue sur PIZZA EXPRESS</h1>
<form name="bebop" action="http://www.pizzaexpress.fr/commandeenligne.cgi"
 method="post" onSubmit="return verifieEtCommande(this)">
Combien de pizzas ?
<input type="text" name="np" value="2" onChange="return verifie(this)">
<br>
<input type="submit" value="Commander">
</form>
</body>
</html>
```

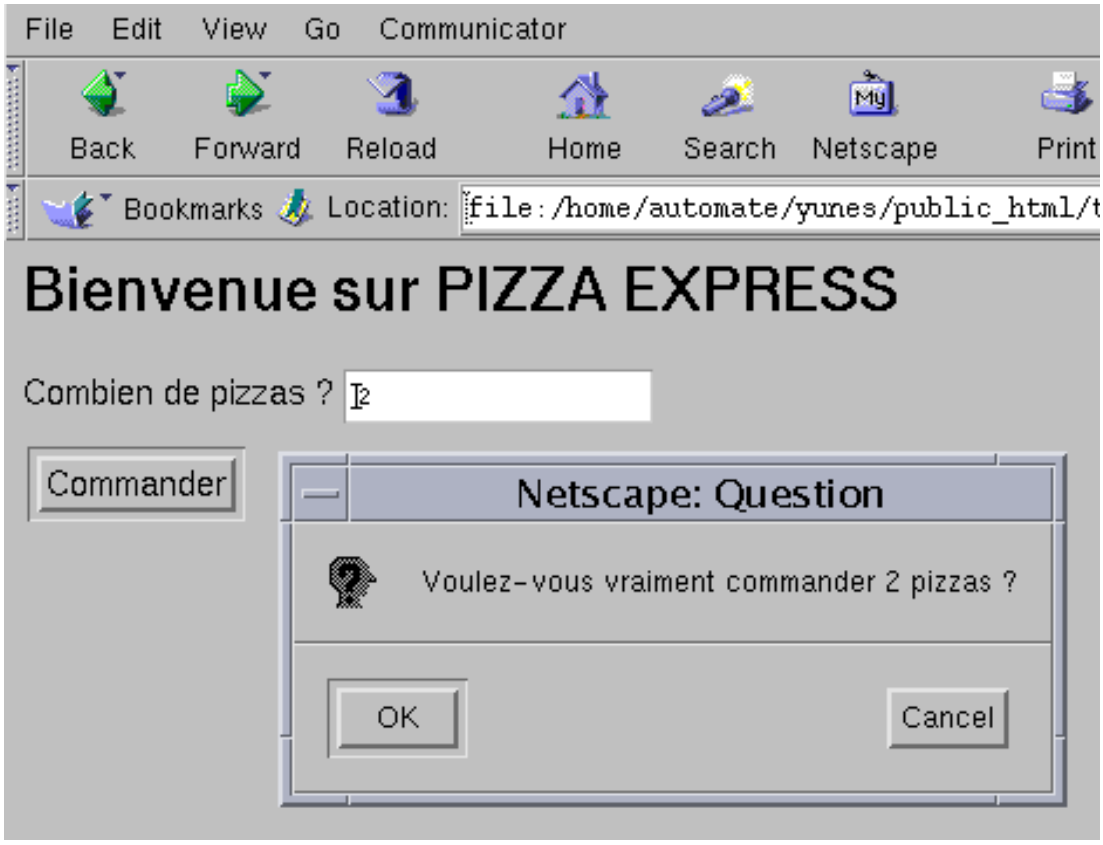

FIG. 6 – après clic sur “Commander”

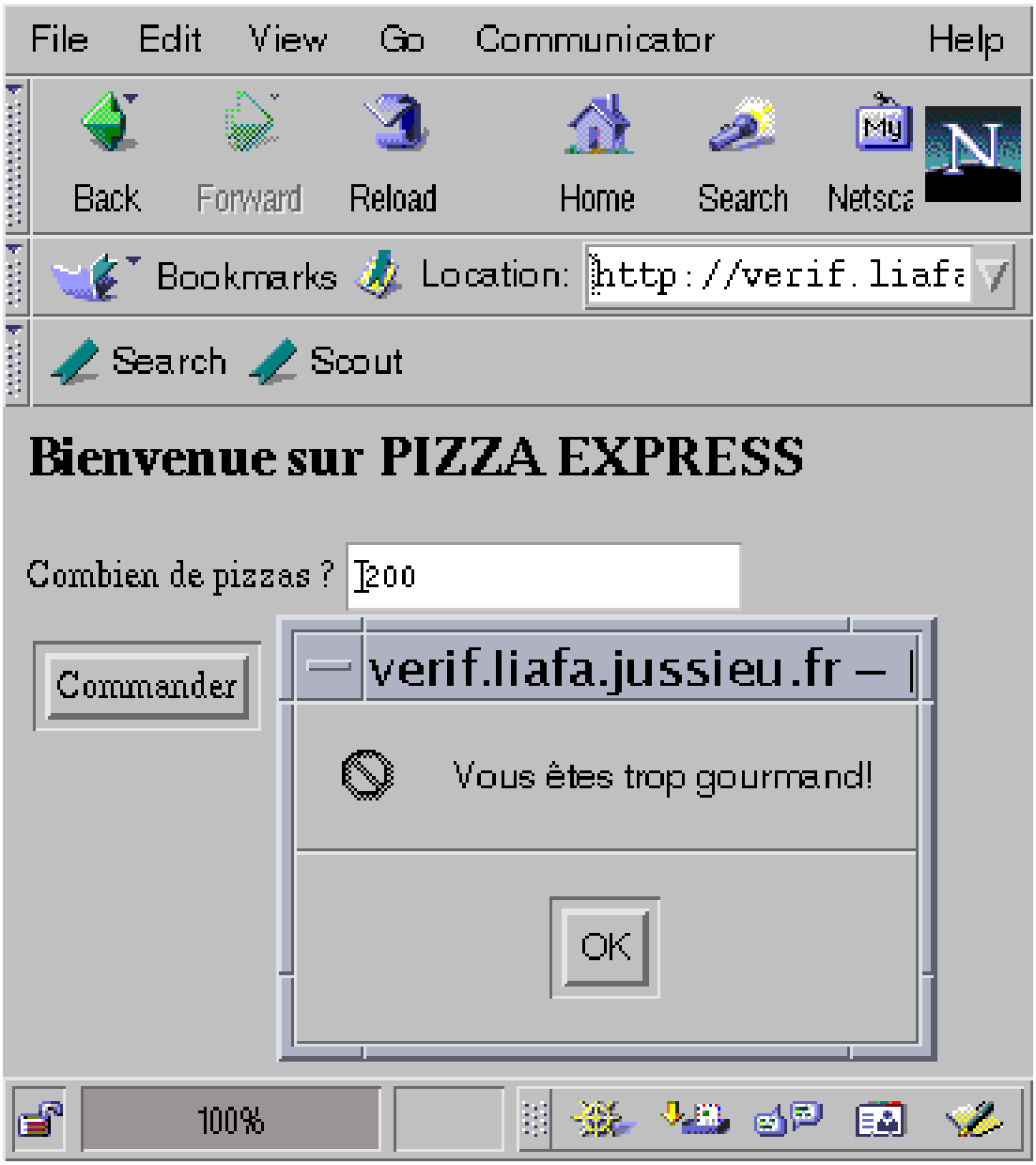

FIG. 7 – après saisie de 200 et clic sur “Commander”

Le navigateur

L'objet `navigator`

Ses propriétés :

`appCodeName` `appName` `appVersion`
`language` `mimeTypes` `platform`
`plugins` `userAgent`

Ses méthodes :

`javaEnabled` `plugins.refresh` `preference`
`taintEnabled`

L'objet `MimeType`

Ses propriétés :

`description` `enabledPlugin` `suffixes`
`type`

L'objet `Plugin`

Ses propriétés :

`description` `filename` `length`
`name`

Les événements

L'objet `event`

Ses propriétés :

`target`	`type`	`data`
`height`	`layerX`	`layerY`
`modifiers`	`pageX`	`pageY`
`screenX`	`screenY`	`which`
`width`		

Les gestionnaires d'événements

`onAbort`	`onBlur`	`onChange`
`onClick`	`onDblClick`	`onDragDrop`
`onError`	`onFocus`	`onKeyDown`
`onKeyPress`	`onKeyUp`	`onLoad`
`onMouseDown`	`onMouseMove`	`onMouseOut`
`onMouseOver`	`onMouseUp`	`onMove`
`onReset`	`onResize`	`onSelect`
`onSubmit`	`onUnload`	

Les fonctions prédéfinies

escape	eval	isNaN
Number	parseFloat	parseInt
String	taint	unescape
untaint		

Exemple :

```
function cal(exp) {
  exp.form.res.value = eval(exp.value)
}
<form>
<input type="text" name="exp" value="" onChange="return cal(this)">
<input type="text" name="res" value="" readonly>
</form>
```