Les bases du langage JavaScript

Sommaire

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

Introduction

Comme tout langage informatique, vous devez apprendre les bases du langage que sont la syntaxe, les variables, les opérateurs, les structures conditionnelles et les boucles, et le traitement des erreurs.

I. Syntaxe, types de données et variables

Pour votre apprentissage, nous vous proposerons de nombreux exercices. Ceux-ci feront largement appel à l'instruction JavaScript **document.write** pour afficher les résultats. En effet, cette instruction permet d'afficher un texte à la fin de la page web courante.

A. Syntaxe

```
document.write(texte_à_afficher);
```

Fig. 1

Le texte est interprété comme du HTML.

Voici un exemple d'utilisation avec saut de ligne en fin de texte.

```
document.write("Bonjour à tous" + "</br>");
```

Fig. 2

B. Syntaxe du JavaScript

1. Délimitation des instructions, utilisation du point-virgule

Un programme JavaScript se présente sous la forme d'une suite d'instructions séparées par un retour à la ligne ou un point-virgule :

```
var a=0; var b=0;
```

Fig. 3

est équivalent à :

```
var a=0
var b=0
```

Fig. 4

Les deux exemples ci-dessus contiennent deux instructions et se comportent semblablement.

Le code suivant génère une erreur :

```
var a=0 var b=0;
```

Fig. 5

En fin de ligne, le point-virgule est optionnel, mais cela est une bonne habitude de les utiliser quelques soient les cas où il se présente.

C. Indentation et formatage du code

Le JavaScript est insensible aux tabulations et aux espaces multiples, cela vous permet d'indenter et de formater votre code pour le rendre plus lisible.

Par convention, on utilise 2 espaces pour indenter une ligne d'instruction imbriquée dans une autre.

D. Sensibilité à la casse

JavaScript est sensible à la casse, ainsi les instructions suivantes : date, Date, DATE seront interprétées différemment.

1. Commentaires

Le texte entre les caractères /* et */ est ignoré du JavaScript. De même pour toutes les lignes débutant par les caractères //.

Dans l'exemple suivant, la première ligne est ignorée du JavaScript :

```
// var a=0 ;
var b=0 ;
```

Fig. 6

Dans l'exemple suivant, seule la dernière ligne sera exécutée :

```
/*
 * Mise en commentaire :
 * var a=0;
 * var b=0;
*/
var c=0;
```

Fig. 7

2. Type de données

JavaScript utilise les types de données standards suivants :
- **numériques** : 123, 123.45 (on utilise le point à la place de la virgule) ;
- **booléennes :** *true* et *false* ;
- **chaînes de caractères** : « Ceci est un texte ». On utilise les guillemets doubles (") ou simples (') pour les spécifier.

En complément, JavaScript fournit les types :
- **null** pour les données non initialisées ;
- et **undefined** : non défini.

Les types *null* et *undefined* sont généralement utilisés pour tester la valeur d'une variable.

E. Les variables

Les variables permettent de stocker des valeurs. Avant d'être utilisées, les variables doivent être déclarées par l'instruction **var** :

```
var a=0.123;
var b; // contient undefined
var bienvenue = "bonjour à tous";
```

Fig.8

1. Nom des variables

Un nom de variable doit démarrer avec une lettre majuscule ou minuscule (a-z et A-Z), un tiret-bas (_) ou un dollar ($). Les caractères spéciaux autres que le tiret (-), le dollar ($) et le tiret-bar (_) ne sont pas tolérés.

Par convention, si le nom utilisé représente un ensemble de mots, chaque mot lié hormis le premier doit être écrit en mettant en majuscule sa première lettre.

Exemples

```
var maVariable;
var maDeuxiemeVariable;
```

Fig.9

2. Mots réservés

Comme dans la plupart des langages informatiques JavaScript spécifie une liste de mots réservés à son usage exclusif. Ceux-ci ne peuvent être utilisés par le développeur comme nom de variable (ni de fonction, de méthode, d'objet ou de propriétés).

Liste des mots réservés JavaScript :

Tableau n°1

abstract	else	instanceof	super
boolean	enum	int	switch
break	export	interface	Synchronized
byte	extends	let	this
case	false	long	throw
catch	final	native	throws
char	finally	new	transient
class	float	null	true
const	for	package	try
continue	function	private	typeof
debugger	goto	protected	var
default	if	public	void
delete	implements	return	volatile
do	import	short	while
double	in	static	with

II. Les opérateurs

Les opérateurs permettent des opérations sur les nombres et les chaînes :
- opérateurs arithmétiques pour réaliser des calculs ;
- opérateurs d'assignement pour assigner des variables ;
- opérateurs de comparaisons ;
- opérateurs logiques pour combiner des comparaisons.

A. Opérateurs arithmétiques

Voici le tableau des opérateurs arithmétiques :

Tableau n°2

OPERATEUR	DESCRIPTION DE L'OPÉRATEUR	EXEMPLE (X=5)	VALEUR RÉSULTAT DE X	VALEUR RÉSULTAT DE Y
+	Addition	x=y+2	7	5
-	Soustraction	x=y-2	3	5
*	Multiplication	x=y*2	10	5
/	Division	x=y/2	2.5	5
%	Modulo (reste de la division euclidienne)	x=y%2	1	5
++	Incrémentation	x=++y	6	6
		x=y++	5	6
--	Décrémentation	x=--y	4	4
		x=y--	5	4

Réalisez l'exercice 1 en autocorrection.

JavaScript étant un langage dérivé de l'anglais, **le caractère de séparation des décimales est le «.»** (22,2 est représenté par 22.2 en JavaScript).

B. Opérateurs d'assignement

Les variables d'assignement permettent d'affecter une valeur aux variables.

Tableau n°3

OPERATEUR	EXEMPLE	ÉQUIVALENT À
=	x=y	
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Réalisez l'exercice 2 en autocorrection.

C. Utilisation du signe + pour concaténer des chaînes de caractères

Pour concaténer des chaines de caractères, on utilise le signe +.

Exemple

```
var a = "Bonjour";
var b = "à";
var bienvenue = a + b + "tous";
document.write(bienvenue + "<br/>");
var bienvenue2 = a + " " + b + " tous";
document.write(bienvenue2 + "<br/>");
```

Fig. 10

Résultat attendu :

```
Bonjouràtous
Bonjour à tous
```

Fig. 11

Réalisez l'exercice 3 en autocorrection.

D. Insérer des caractères spéciaux dans un texte à l'aide du caractère back-slash

Le caractère « \ » (back slash) permet d'insérer des caractères comme les apostrophes, des retours à la ligne et d'autres types de caractères spéciaux.

Par exemple, si vous utilisez le guillemet double («) pour délimiter un texte et, que ce même texte contient également un guillemet double, vous pourrez le faire en précédant cette *double-quote* du caractère back-slash (barre oblique inversée) « \ ».

Code

```
<head>
<script type="text/javascript">
var nom = "toto";
var texte = "Votre \"nom\" est " + nom);
document.write(nom + "<br />");
</script>
</head>
```

Fig. 12

Résultat attendu

Fig. 13

Voici la liste des caractères spéciaux que vous pouvez ajouter dans un texte à l'aide du caractère back-slash "\" (barre oblique inversée) :

Tableau n°4

CODE	OUTPUTS
\'	Guillemet simple
\"	double guillemet
\\	Back-slash (barre oblique inversée)
\n	Nouvelle ligne
\r	Retour chariot
\t	tabulation
\b	Espace arrière
\f	form feed

Réalisez l'exercice 4 en autocorrection.

E. Opérateurs de comparaison

Réalisez l'exercice 5 en autocorrection.

Les expressions de comparaisons doivent être spécifiées entre parenthèses. Soit A = 10 et B = 20 :

Tableau n°5

OPERATEUR	DESCRIPTION	EXEMPLE
==	Détermine si les valeurs de deux opérandes sont égales, si oui, la comparaison renvoie la valeur true, sinon false.	(A == B) renvoie la valeur *false*
!=	Détermine si les valeurs de deux opérandes sont **différentes**. Si oui, la comparaison renvoie la valeur true, sinon false.	(A != B) renvoie la valeur *true*
>	Détermine la valeur de l'opérande de gauche qui est **supérieure** à la valeur de l'opérande de droite. Si oui, renvoie la valeur true, sinon, renvoie la valeur false.	(A > B) renvoie la valeur *false*
<	Détermine si la valeur de l'opérande de gauche est plus petite que la valeur de l'opérande de droite. Si oui, renvoie la valeur true, sinon, renvoie la valeur false.	(A < B) renvoie la valeur *true*
>=	Détermine si la valeur de l'opérande de gauche est **supérieure ou égale** à la valeur de l'opérande de droite. Si oui, renvoie la valeur true, sinon, renvoie la valeur false.	(A >= B) renvoie la valeur *false*
<=	Détermine si la valeur de l'opérande de gauche est **supérieure ou égale** à la valeur de l'opérande de droite. Si oui, renvoie la valeur true, sinon, renvoie la valeur false.	(A <= B) renvoie la valeur *true*

F. Opérateurs logiques

Soit A = 10 et B = 20 :

Tableau n°6

OPÉRATEUR	DESCRIPTION	EXEMPLE
&&	On appelle cet opérateur, l'opérateur **AND** (et). Il permet de concaténer des conditions.	(A && B) renvoie la valeur true.
||	On appelle cet opérateur, l'opérateur **OR** (ou). Renvoie *true* si l'un des deux opérateurs contient une valeur différente de zéro, sinon renvoie *false*.	(A || B) renvoie la valeur true.
!	On appelle cet opérateur, l'opérateur **NOT** (non). On l'utilise pour inverser le résultat de d'une condition.	!(A && B) renvoie la valeur false.

Réalisez l'exercice 6 en autocorrection.

III. Les structures conditionnelles

Les structures conditionnelles permettent d'exécuter un bloc d'instructions selon certaines conditions.

JavaScript fournit plusieurs instructions conditionnelles :
- if ;
- switch ;
- if ternaire.

A. Instruction if

L'instruction if permet d'exécuter des instructions selon qu'une condition est vérifiée ou non.

Elle peut prendre plusieurs formes :

1. if …
2. if … else …
3. if … else if …

Les instructions exécutées conditionnellement doivent être incluses entre accolades et éventuellement omises s'il n'y en a qu'une.

1. Syntaxe

Conseil :
Pour une meilleure lisibilité, mettez vos blocs d'instructions conditionnelles entre accolades, cf. Fig. 14.

```
if (expression) {
 ...
}
```

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

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

Fig. 14

Exemple:

```
var x=0;

if (x) {
   message = "Bonjour à tous";
} else {
   message = "Au revoir à tous";
}

document.write(message); /* doit afficher "Au revoir à tous" */
```

Fig. 15

Réalisez l'exercice 7 en autocorrection.

B. Switch...case

Généralement, vous utiliserez l'instruction if pour vos structures conditionnelles. Cependant, lorsque plusieurs séries d'instructions dépendent de la valeur d'une même variable, il est plus concis et lisible d'utiliser l'instruction *switch*.

1. Syntaxe

```
switch (expression) {
 case condition 1 :
  ...
  break;
 case condition 2 :
  ...
  break;
 ...
 case condition n :
  ...
  break;
 default :
  ...
}
```

Fig. 16

Voici comment marche l'instruction **switch**. D'abord, vous spécifiez une expression (le plus souvent une variable), celle-ci est évaluée une seule fois. Sa valeur est ensuite comparée avec celle de chaque cas (case). Si un cas correspond, le block associé est exécuté jusqu'à ce qu'il rencontre l'instruction **break**, le cas par défaut (**default:)** ou la fin du block **switch**.

L'instruction **default** indique à l'interpréteur ce qu'il faut exécuter et ne correspond en aucun cas **à la valeur de l'expression.**

Exemple

```
var mention='A';
document.write("Début du bloc switch <br />");
switch (mention)
{
 case 'A': document.write("Excellent travail<br />");
     break;
 case 'B': document.write("Bon travail<br />");
     break;
 case 'C': document.write("Travail moyen<br />");
     break;
 case 'D': document.write("Insuffisant<br />");
     break;
 case 'F': document.write("Très insuffisant<br />");
     break;
 default: document.write("Mention inconnue<br />")
}
document.write("Fin de bloc");
```

Fig. 17

Le code ci-dessus produit le résultat suivant :

```
Début du bloc switch
Excellent travail
Fin de bloc
```

Fig. 18

Exemple

```
var mention='B';
document.write("Début du bloc switch <br />");
switch (mention)
{
 case 'A': document.write("Excellent travail");
 case 'B': document.write("Bon travail<br />");
 case 'C': document.write("Travail moyen<br />");
 case 'D': document.write("Insuffisant<br />");
 case 'F': document.write("Très insuffisant<br />");
 default: document.write("Mention inconnue<br />")
}
document.write("Fin de bloc");
```

Fig. 19

Le code ci-dessus produit le résultat suivant :

```
Début du bloc switch
Bon travail
Insuffisant
Très insuffisant
Mention inconnue
Fin de bloc
```

Fig. 20

Réalisez l'exercice 8 en autocorrection.

C. Une structure conditionnelle très concise : l'opérateur ternaire

Si vous devez écrire un bloc de type if … else … ne spécifiant qu'une instruction selon le cas, JavaScript vous fournit une instruction conditionnelle plus concise.

1. Syntaxe

```
(expression) ? instruction1 : instruction2;
```

Fig. 21

2. Explication

L'instruction1 est exécutée si l'expression renvoie la valeur true, sinon exécute l'instruction2.

Exemple

```
var mention=true;
mention ? document.write('Oui') : document.write('Non');
```

Fig. 22

Résultat attendu

```
Oui
```

Fig. 23

Réalisez l'exercice 9 en autocorrection.

IV. Les boucles

Lorsque vous écrivez des programmes, il se peut que vous ayez besoin de répéter un certain nombre de fois la même suite d'instructions. Dans cette situation, vous aurez besoin d'écrire une boucle pour réduire le nombre de lignes nécessaire.

JavaScript propose plusieurs solutions, à vous de choisir la plus appropriée selon le cas qui se présente à vous :
- while ;
- do…while ;
- for.

Deux autres instructions permettent de contrôler l'exécution des boucles :
- break ;
- continue.

A. Boucle While

L'instruction **while** permet de réitérer l'exécution d'un bloc d'instruction **tant qu'une condition est vérifiée**.

1. Syntaxe

```
while (expression){
  Instruction(s) à executer si l'expression renvoie la valeur true
}
```

Fig. 24

Exemple

```
var i=0;
while (i<5){
   document.write("Le nombre est " + i + "<br />");
   i++;
}
```

Fig. 25

Afin d'éviter une boucle infinie, n'oubliez-pas d'incrémenter la variable i.

Résultat attendu

```
Le nombre est 0
Le nombre est 1
Le nombre est 2
Le nombre est 3
Le nombre est 4
```

Fig. 26

Réalisez l'exercice 10 en autocorrection.

B. Boucle do...while

La boucle **do…while** est une variante de la boucle **while**. Cette boucle exécute une suite d'instructions avant de vérifier si la condition de fin de boucle renvoie à la valeur true.

La suite d'instructions sera donc exécutée au moins une fois.

1. Syntaxe

```
do
 {
 Instructions
 }
while (variable<valeur de fin de boucle);
```

Fig. 27

Exemple

```
var i = 0;
do {
    document.write('le nombre est ' + i + '<br/>');
    i++;
} while (i < 5);
```

Fig. 28

Résultat attendu

```
Le nombre est 0
Le nombre est 1
Le nombre est 2
Le nombre est 3
Le nombre est 4
```

Fig. 29

Réalisez l'exercice 11 en autocorrection.

C. Boucle For

La boucle for accepte en argument un compteur qui sera incrémenté à chaque itération.

1. Syntaxe

```
for (initialisation; test; incrémentation) {
 Instructions à exécuter si le test renvoie la valeur true
}
```

Fig.30

Exemple

```
var count;
document.write("Début de boucle" + "<br />");
for(count = 0; count < 2; count++){
 document.write("Valeur du compteur : " + count );
 document.write("<br />");
}
document.write("Fin de boucle!");
```

Fig.31

Résultat attendu

```
Début de boucle
Valeur du compteur : 0
Valeur du compteur : 1
Valeur du compteur : 2
Fin de boucle!
```

Fig.32

Réalisez l'exercice 12 en autocorrection.

D. Instructions de contrôle de boucle : Break et Continue

1. Instruction Break

L'instruction **break** permet d'interrompre *immédiatement* une boucle. L'interpréteur rencontrant cette instruction interrompt la boucle en cours et "saute" à la première instruction après cette même boucle.

Exemple

```
<script type="text/JavaScript">
<!--
var x = 0;
document.write("Début de boucle<br /> ");
while (x < 5)
{
 if (x == 2){
   break; // Interrompt la boucle
 }
 x = x + 1;
 document.write( "Valeur de x : " + x + "<br />");
}
document.write("Fin de boucle!<br /> ");
//-->
</script>
```

Fig. 33

Résultat attendu

```
Début de boucle
Valeur de x : 0
Valeur de x : 1
Fin de boucle!
```

Fig. 34

2. Instruction Continue

L'instruction **Continue** permet de réitérer immédiatement une boucle en ignorant les instructions suivantes. La condition de fin de boucle est immédiatement vérifiée.

Exemple

```
for (i=0;i<=5;i++) {
   if (i==3) {
      continue;
   }
   document.write(i + "<br />");
}
```

Fig. 35

Résultat attendu

```
0
1
2
4
5
```

Fig. 36

Réalisez l'exercice 13 en autocorrection.

V. Affichage des boîtes de dialogue

Les boîtes de dialogues permettent d'interagir avec l'utilisateur.

JavaScript propose 3 types de boîtes de dialogue :
- d'alerte ;
- de confirmation ;
- d'entrée.

A. Instruction Alert

L'instruction **alert** permet d'afficher un message d'avertissement.

```
alert("coucou");
```

Fig. 37

Exemple

```
alert("Ceci est une alerte");
```

Fig. 38

Résultat attendu

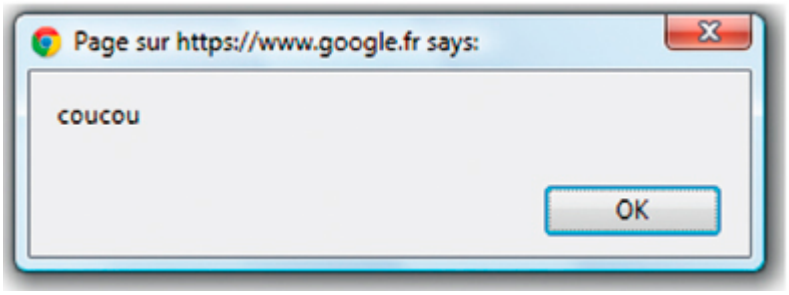

Fig. 39

B. Instruction Confirm

L'instruction **confirm** permet d'afficher une demande de confirmation.

1. Syntaxe

```
confirm("message")
```

Fig. 40

Exemple

```
var input =
confirm("Continuer?");
if (input == true ){
 alert("Je continue!");
}else{
 alert("Je ne continue pas!");
}
```

Fig. 41

Résultat attendu

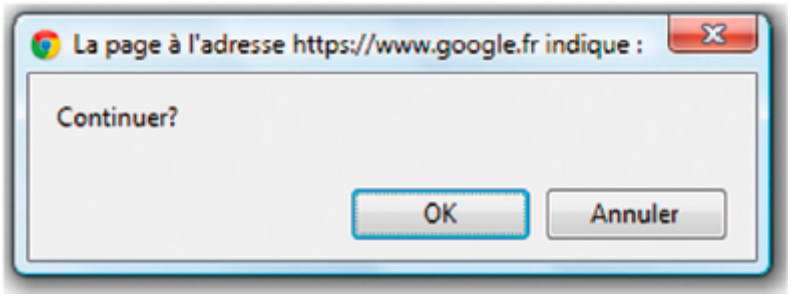

Fig. 42

C. Instruction Prompt

L'instruction **prompt** affiche une boite de dialogues de saisie.

1. Syntaxe

```
prompt(message, initialisation)
```

Fig. 43

Le paramètre initialisation est optionnel. Il permet de pré remplir le texte à saisir.

Exemple

```
var input = prompt("Enter votre nom : ", "votre nom ici");
alert("Vous avez entré : " + input );
```

Fig. 44

Résultat attendu

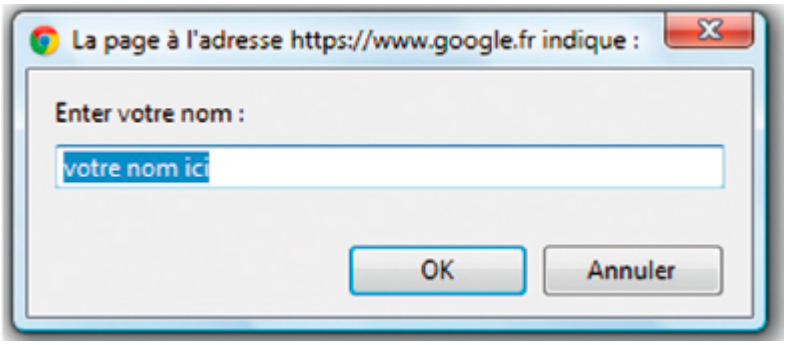

Fig. 45

Réalisez l'exercice 14 en autocorrection.

VI. Gestion des erreurs Javascript avec Try... Catch and Throw

A. Try...catch

L'instruction try...catch permet de traiter les erreurs JavaScript qui pourraient apparaître dans un bloc d'instruction. Le traitement des erreurs permet de ne pas interrompre le programme.

1. Syntaxe

```
try
 {
 instruction(s)
 }
catch(erreur)
 {
 traitement de l'erreur
 }
```

Fig. 46

La variable erreur est positionnée par l'interpréteur, elle indique l'erreur.

Exemple

```
var texte="";
try
{
xxxx // cette instruction inconnue générera une erreur
}
catch(erreur)
{
texte="Erreur rencontrée.\n\n";
texte+="Description de l'erreur: " + erreur.message + "\n\n";
texte+="Cliquez sur le bouton OK pour continuer.\n\n";
alert(texte);
}
```

Fig. 47

Résultat attendu

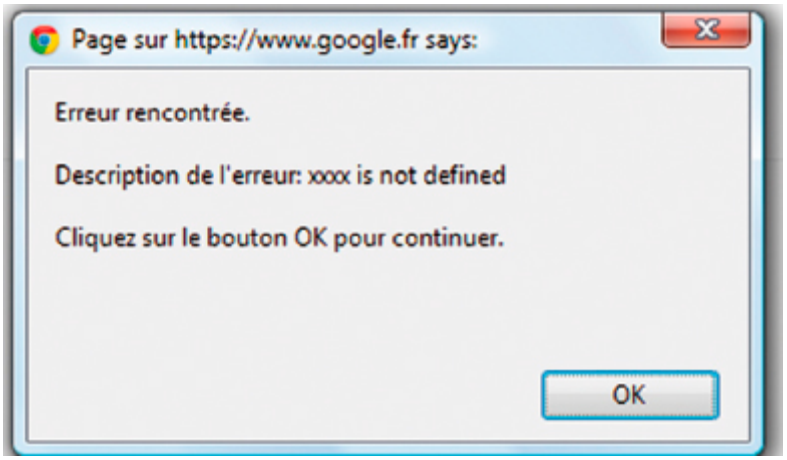

Fig. 48

Réalisez l'exercice 15 en autocorrection.

B. Instruction Throw

L'instruction **throw** permet de générer une erreur JavaScript. Utilisée avec l'instruction **try…catch**, cette instruction vous permet de gérer le déroulement de votre programme et d'afficher des messages d'erreurs explicites.

1. Syntaxe

```
throw erreur
```

Fig. 49

L'erreur peut être un texte, un numérique, un booléen ou un objet.

Exemple

```
var x=prompt("Entrez un nombre entre 5 et 10:","");
try
  {
  if(x>10)
    {
    throw "Err1";
    }
  else if(x<5)
    {
    throw "Err2";
    }
  else if(isNaN(x)) // n'est pas numérique
    {
    throw "Err3";
    }
  }
catch(err)
  {
  if(err=="Err1")
    {
    document.write("Erreur! La valeur est trop importante.");
    }
  if(err=="Err2")
    {
    document.write("Erreur! La valeur est trop basse.");
    }
  if(err=="Err3")
    {
    document.write("Erreur! La valeur spécifiée n'est pas un numérique.");
    }
  }
```

Fig. 50

VII. Les fonctions

Les fonctions sont un groupe d'instructions réutilisables. Elles permettent de mieux structurer le code, de le rendre plus lisible.

JavaScript fournit au développeur un certain nombre de fonctions natives, mais vous pouvez également écrire vos propres fonctions.

A. Définition d'une fonction

Avant d'être utilisée, une fonction doit être définie à l'aide de l'instruction **function**.

1. Syntaxe

```
function nomDeLaFonction(liste de paramètre)
{
 instructions
}
```

Fig. 51

La liste de paramètres est une suite de noms de variable ; chaque paramètre étant séparé par une virgule (,).

B. Appel d'une fonction

Pour appeler une fonction, vous utilisez son nom en lui passant entre parenthèse les variables nécessaires.

Exemple

```
function bonjour(nom) {
    alert("Bonjour " + nom);
}
bonjour("fabienne");
```

Fig. 52

Résultat attendu

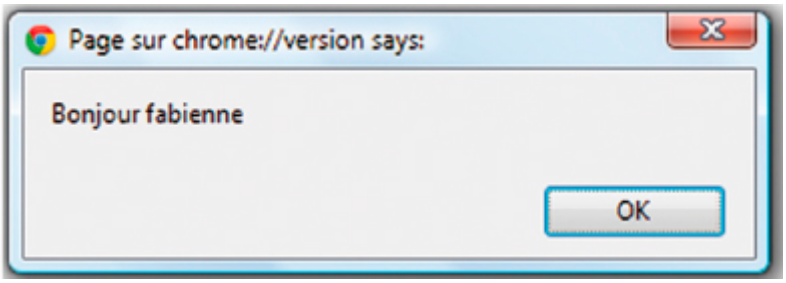

Fig. 53

Si un paramètre est omis, il prend la valeur *null*.

C. Fonction retournant une valeur

La fonction appelée peut retourner une valeur grâce à l'instruction **return**.

Exemple

```
function validate(val) {
    if (!val || typeof val != "number") {
        return "valeur incorrecte";
    } else is (val < 100) {
        return "valeur correcte";
    } else {
        return "valeur incorrecte";
    }
}

document.write(validate() + "<br/>");
document.write(validate(200) + "<br/>");
document.write(validate(5) + "<br/>");
document.write(validate("test") + "<br/>");
```

Fig. 54

Résultat attendu

```
valeur incorrecte
valeur incorrecte
valeur correcte
valeur incorrecte
```

Fig. 55

Réalisez l'exercice 16 en autocorrection.