03/02/2018

Le langage Javascript

Sami Hadhri

I. Introduction

1. JavaScript n'est pas Java

Pour commencer, tordons le cou à une erreur classique : JavaScript et Java ne sont en aucune manière apparentés. Il s'agit de deux langages différents. En effet, dans le cadre d'un site Web :

- JavaScript est toujours interprété côté client, sur la machine sur laquelle tourne le navigateur qui analyse et affiche la page Web, alors que Java peut être compilé ou interprété côté serveur.
- Un code JavaScript est appelé dans une page Web différemment d'une *applet* Java.
- Il est nécessaire de déclarer les variables en Java, alors que cette déclaration peut être implicite en JavaScript.

2. Trois méthodes d'insertion de code JavaScript dans un document (X)HTML

Il existe trois manières d'insérer du code JavaScript dans un document (X)HTML :

- La manière la plus simple à mettre en œuvre est l'utilisation des attributs prévus dans les recommandations (X)HTML pour cela : **onclick**, **onmouseover**, etc. Cette méthode présente un inconvénient majeur, celui de mêler le code destiné à donner la structure du document (le **HTML**) au code destiné à le mettre en quelque sorte en mouvement, le JavaScript. La mise à jour d'une telle page est difficile, de la même manière que gérer la mise en forme CSS à l'aide de l'attribut **style** est une pratique à déconseiller.
- Une deuxième méthode, analogue à l'utlisation d'une feuille de style interne en CSS, consiste à faire appel à l'élément **script** et à insérer le code JavaScript à l'intérieur de cet élément. Par exemple,

<script>(...)**</script>**

L'attribut **type** est obligatoire. Vous rencontrerez parfois l'atttribut **language** avec la valeur **JavaScript**, mais cet attribut est obsolète et ne devrait donc plus être utilisé.

Cet élément peut être présent dans l'entête du fichier HTML tout comme l'élément **style** des feuilles de style internes. Il est aussi possible de l'insérer au cœur de la page HTML, comme descendant de l'élément **body**, mais cela complique encore une fois la maintenance de la page.

- Une troisième méthode fait elle aussi appel à l'élément **script**, mais cette fois-ci le code JavaScript se trouve dans un fichier externe. Dans ce cas, l'élément **script** précise à l'aide de l'attribut **src**l'emplacement du fichier JavaScript :

<script src="*../scripts/aideSaisie.js*">

II. Syntaxe générale

1. Conventions d'écriture

a. Fin de ligne

Chaque instruction de code doit se terminer par **;**. Par exemple, les deux syntaxes suivantes aboutissent au même résultat :

a
=
b
+
2
;

et...

a=*b*+2 ;

b. Noms de variables et de fonctions

JavaScript est sensible à la casse employée. Par conséquent, **variabletest** et **Variabletest** désignent deux variables différentes.

Il existe un certain nombre de noms réservés, qui sont à peu près les mêmes que dans les autres langages de programmation, comme **var**, **function**, **for**, **if**... Nous les rencontrerons au fur et à mesure.

2. Commentaires

Un commentaire JavaScript permet de placer du texte en-dehors du script : il n'est alors pas interprété. Deux syntaxes sont possibles :

- Quand le commentaire s'étend sur une seule ligne, on peut commencer cette ligne par //.
- Quand le commentaire doit s'étendre sur plusieurs lignes, il doit commencer par **/*** et se terminer par ***/**.

```
//Une ligne
/*Deux
lignes*/
```

III. Variables

1. Types de données

a. Les types

JavaScript est un langage dans lequel il est possible de ne pas déclarer qu'une variable doit être d'un certain de type de données. Cela signifie qu'il suffit d'indiquer **a=2;**, par exemple, pour déclarer premièrement que

l'on définit une variable qui s'appelle **a** et qui a pour valeur 2, mais aussi deuxièmement que cette variable est de type nombre.

Il n'existe que quelques types de données en JavaScript, ce qui en facilite l'usage mais complique le contrôle des valeurs autorisées pour une variable donnée. Ces types prédéfinis sont :

- la chaîne de caractères. Une chaîne de caractères est délimitée par des " ou '. Par exemple, **a="Ceci est une chaîne" ;**. Si l'on doit utiliser des apostrophes ou des guillemets imbriqués, on peut faire appel à l'autre forme : **"C'est une chaîne"**, voire « échapper » les caractères : **'Le duc déclara : "C\'est une affaire d\'honneur".'** Il existe d'autres caractères échappés : **\b** correspond au retour arrière, **\f**au saut de page, **\n** au saut de ligne, **\r** au retour chariot, **\t** à la tabulation horizontale et finalement \\ au caractère \ lui-même.
- les nombres. Il s'agit soit d'un nombre entier (par exemple 3, -456 ou 78900001), soit d'un nombre à virgule flottante (comme 3.1415 ou -745.6). Les nombres sont compris entre 2^{24}-1 et -(2^{24}), ce qui correspond à environ 10^{308}
- les booléens, avec le type **boolean**. Les variables de type booléen ne peuvent prendre que deux valeurs possibles : « vrai » (**true**) ou « faux » (**false)**.

Il existe d'autres types, qui sortent du cadre restreint de ce cours.

b. Conversion de type

JavaScript permet de changer le type d'une variable, avec les fonctions suivantes :

- *Transformation d'une chaîne de caractères en nombre :* **parseInt()** et **parseFloat()** permettent de convertir une chaîne de caractères en nombre (respectivement entier ou à virgule flottante). Si par exemple la variable **x** contient la chaîne de caractères **"34.7"**, **parseInt(x)** renvoie 34, et **parseFloat(x)** renvoie 34,7.
- **string()** transforme un objet en chaîne de caractères.

2. Déclaration ; affectation de valeurs

a. Opérateur d'affectation simple =

On affecte une valeur à l'aide de l'opérateur =. Par exemple, **a="Ceci est une chaîne"**. Cet opérateur permet aussi d'affecter la même valeur à plusieurs variables. Par exemple, **a=b=c=d=e=5;** affecte la valeur 5 aux cinq variables **a**, **b**, **c**, **d** et **e**.

b. Opérateurs d'affectation complexe

Ces opérateurs permettent d'effectuer une opération sur une variable puis de lui affecter le résultat.

- += permet de réaliser une addition : si **a** vaut 5, **a+=2;** affecte 5+2=7 à **a**.
- -= permet de réaliser une soustraction : si **a** vaut 5, **a-=2;** affecte 5-2=3 à **a**.
- ***=** permet de réaliser une multiplication : si **a** vaut 5, **a*=2;** affecte 5*2=10 à **a**.
- /= permet de réaliser une division : si **a** vaut 5, **a/=2;** affecte 5/2=2.5 à **a**.

IV. Fonctions

1. Introduction

Parfois, il est nécessaire de répéter de mêmes portions de codes. Par exemple, on peut souhaiter afficher un résultat donné sous une certaine forme, ou bien faire un calcul répétitif. Dans ce cas, au lieu de réécrire plusieurs fois cette même portion de code, on définit ce que l'on appelle des *fonctions*. Ces fonctions peuvent accepter des *paramètres*, ou arguments, pour leur permettre de réaliser l'action désirée.

2. Déclaration d'une fonction

Une fonction est un ensemble d'instructions que l'on peut appeler séparément. La forme générale de sa déclaration est

```
function nomFonction(liste éventuelle des arguments)
{
(...)
}
```

Une fonction est ensuite appelée dans le code JavaScript par **nomFonction(arguments...)**. Une fonction peut ne pas nécessiter d'arguments (qui en sont les données d'entrée), et elle peut ne pas retourner de valeur en sortie.

Exemple de fonction ne prenant pas d'argument...

```
function quandSommesNous()
{
aujourdhui=new Date ;
alert(aujourdhui);
}
```

b. Valeur retournée

Une fonction peut éventuellement « retourner » une valeur à l'aide de l'instruction **return**. Par exemple,

```
function surfaceRectangle(longueur, largeur)
{
return longueur*largeur ;
}
```

Un appel à cette fonction se fait par exemple ainsi :

alert(**surfaceRectangle**(2.5, 3))

À l'écran, une boîte d'affichage montrera 7.5.

Exercice 1. Écriture d'une fonction simple

Énoncé

Vous utiliserez les fonctions alert(qui affiche une chaîne de caractères) et prompt (qui invite à la saisie d'une donnée), dont voici les syntaxes :

- alert(Chaîne à afficher à l'écran)
- variable=prompt(Question à afficher à l'écran)

1. Ecrire le code pour que le click sur un bouton lance une fonction. Cette fonction demande la saisie d'une largeur, d'une longueur et affiche la surface du rectangle correspondant.
2. Remplacez ce calcul par celui du périmètre (double de la somme de la longueur et de la largeur).

Correction

```
<html>
<head>
<title>Écriture d'une fonction simple</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
// <!--
function surfaceRectangle(){
var largeur; var longueur;
largeur=prompt("Quelle est la largeur du rectangle ?");
longueur=prompt("Quelle est la longueur du rectangle ?");
alert("La surface vaut "+longueur*largeur);
alert("Le périmètre vaut "+2*(Number(longueur)+Number(largeur))); //Il faut ici transformer les chaînes saisies en nombre, faute de quoi l'opérateur + agit comme une concaténation et non une somme
}
// -->
</script>
</head>
<body>
<h1>Écriture d'une fonction simple</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button onclick="surfaceRectangle()">Lancer la fonction</button></p>
</body>
</html>
```

Exercice 2. Écriture d'une fonction renvoyant une valeur

Énoncé

Soit le code suivant :

```
Soit le code suivant :
<html>
<head>
<title>Écriture d'une fonction renvoyant une valeur</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
// <!--
// -->
</script>
</head>
<body>
<h1>Écriture d'une fonction complexe</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button onclick="">Lancer la fonction</button></p>
</body>
</html>
```

1. Créez une fonction qui :
    - demande la saisie d'un rayon ;
    - retourne la surface du cercle de ce rayon
2. Affichez le résultat de l'appel à cette fonction en cliquant sur le bouton.

Correction

```
<html>
<head>
<title>Écriture d'une fonction renvoyant une valeur</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
// <!--
function surfaceCercle() {
var rayon=parseFloat(prompt("Entrez le rayon du cercle : "));
return 3.14*rayon*rayon;
}
// -->
</script>
</head>
```

```
<body>
<h1>Écriture d'une fonction complexe</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button onclick="alert('Surface du cercle :'+surfaceCercle())">Lancer la fonction</button></p>
</body>
</html>
```

3. Valeur de paramètres par défaut

La version 2015 d'ECMAScript (dite aussi ES6) introduit la possibilité de définir des valeurs par défaut pour les paramètres passés à une fonction. On écrira ainsi, par exemple...

```
function surfaceRectangle(longueur=30, largeur=20)
 {
  return longueur*largeur ;
 }
```

4. Portée d'une variable

La portée d'une variable désigne l'ensemble du code dans lequel elle peut être utilisée.

Si une variable est déclarée sans le mot-clef **var**, elle peut être utilisée n'importe où dans le script. On l'appelle alors *variable globale*.

Si une variable est déclarée avec le mot-clef **var**, elle ne peut être utilisée que dans le bloc où elle se trouve. On l'appelle alors *variable locale*. Ainsi, dans l'exemple suivant...

```
var a = 8 ;

function testFonction(){
var pi = 3.14 ;
(...)                                                                    ;
}

function testFonction2(){
(...)                                                                    ;
}
```

... la variable **a** peut être utilisée dans les fonctions **testFonction1** et **testFonction2**, mais la variable **pi** ne peut être utilisée que dans la fonction **testFonction1**.

Cette possibilité de contrôler la portée d'une variable conduit à conseiller l'utilisation du mot-clef **var** dès que cela est possible. Cela permet par exemple d'éviter d'écraser par inadvertance la valeur d'une variable portant le même nom.

Exercice 1. Écriture d'une fonction avec arguments

Énoncé

```
Soit le code suivant :
<html lang="fr">
<head>
<title>Écriture d'une fonction avec arguments</title>
<meta charset="UTF-8">
<script>
</script>
</head>
<body>
<h1>Écriture d'une fonction avec arguments</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="">Lancer la fonction</button></p>

</body>
</html>
```

1. Créez deux variables globales *a* et *b*, initialisées respectivement à 3 et à 2.
2. Créez une fonction multiplie prenant un argument *x* prenant comme valeur par défaut 8, et renvoyant le résultat de la multiplication de *x* par 3
3. Créez une fonction affiche, appelée au clic sur le bouton, qui affiche dans des boîtes d'alerte successivement le résultat de multiplie appliquée *a*, à *b*, puis sans aucun paramètre (en exécutant donc la fonction avec la valeur de *x* par défaut)

Correction

```
<html lang="fr">

<head>
<title>Écriture d'une fonction avec arguments</title>
<meta charset="UTF-8">
<script>
var a=3;
var b=-2;
function multiplie(x=8)
{
return 3*x;
}
function affiche()
{
alert(multiplie(a));
alert(multiplie(b));
```

```
alert(multiplie())
}
</script>
</head>
<body>
<h1>Écriture d'une fonction avec arguments</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="affiche()">Lancer la fonction</button></p>
</body>
</html>
```

V. Tableaux et boucles

1. Tableau à une dimension

Un tableau permet de déclarer une variable qui doit se présenter comme une collection de valeurs. On numérote alors ces valeurs à partir de 0. Par exemple,

//Déclaration d'un tableau de 4 éléments :
var *tableau1* = **new** Array(4) ;
//Déclaration d'un tableau dont le nombre d'éléments est a priori inconnu :
var *tableau2* = **new** Array() ;

Pour affecter des valeurs à un tableau, plusieurs possibilités sont disponibles :

1. On peut affecter les valeurs l'une après l'autre :

   var *tableau1* = **new** Array(4) ;
   tableau1[0]="Beurre" ;
   tableau1[1]="Confiture" ;
   tableau1[2]="Pain" ;
   tableau1[3]="Jus de fruit" ;

2. On peut affecter les valeurs en une seule ligne :

   var *tableau1* = **new** Array(4) ;
   tableau1=["Beurre", "Confiture", "Pain", "Jus de fruit"] ;

3. Il est enfin possible de déclarer et définir le tableau simultanément :

   var *tableau1* = **new** Array('Beurre', 'Confiture', 'Pain', 'Jus de fruit') ;

La propriété **length** d'un tableau renvoie son nombre d'éléments. Par exemple...

var *tableau1* = **new** Array('Beurre', 'Confiture', 'Pain', 'Jus de fruit') ;
alert(*tableau1.length*) ;

... renvoie 4.

Exercice 1. Utilisation d'un tableau

Énoncé

```
Soit le code suivant :
<html>
<head>
<title>Utilisation d'un tableau</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<script type="text/javascript">
// <!--
// -->
</script>
</head>
<body>
<h1>Utilisation d'un tableau</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="">Lancer la fonction</button></p>
</body>
</html>
```

1. Créez un tableau nommé tab dont le premier élément est -2, le deuxième 1 et le troisième 4
2. Créez une fonction additionne prenant un argument x et renvoyant le résultat de l'addition de x à 2
3. Créez une fonction affiche, appelée au clic sur le bouton, qui affiche dans des boîtes d'alerte successivement le résultat de additionne appliqué au premier élément, puis au dernier élément du tableau (en utilisant la propriété length).

Correction

```
<html>
<head>
<title>Utilisation d'un tableau</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<script type="text/javascript">
// <!--
var tab=[-2,1,4];
function additionne(x)
{
return (x+2);
}
function affiche()
{
alert(additionne(tab[0]));
alert(additionne(tab[tab.length-1]));
```

```
}
// -->
</script>
</head>
<body>
<h1>Utilisation d'un tableau</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="affiche()">Lancer la fonction</button></p>
</body>
</html>
```

2. Boucles for

Une boucle permet de réaliser une suite d'instructions un nombre déterminé de fois. Par exemple...

```
var tab = new Array(3) ;
for (i=0;i<3;i++)
  {
    tab[i]=i                                                                    ;
  }
```

Vous noterez que la numérotation commence à 0. La syntaxe générale de la parenthèse après le mot-clef **for** est

1. instruction de départ ;
2. condition de fin ;
3. instruction à exécuter à chaque fin d'itération.

La dernière instruction est souvent limitée à une incrémentation, mais lorsque la boucle ne comporte qu'une seule instruction, il est possible de l'y placer. L'exemple précédent se simplifie donc en...

```
var tab = new Array(3) ;
for (i=0;i<3;tab[i++]=i) ;
```

Exercice 1. Utilisation de boucles

Énoncé

Soit le code suivant :

```
<html lang="fr">
<head>
<title>Exercice sur les boucles for</title>
<meta charset="UTF-8"/>
<script>
</script>
</head>
<body>
<h1>Boucles for</h1>
```

```
<button onclick="boucle()">Cliquez-moi!</button>
<button onclick="boucle2()">Cliquez-moi aussi!</button>
</body>
</html>
```

1. Au click sur le premier bouton, lancez la fonction boucle() qui crée un tableau de 3 éléments et utilise une boucle for pour le remplir de sorte que l'élément *i* du tableau contienne i^2. Affichez le tableau dans une boîte d'alerte.
2. Au click sur le second bouton, lancez la fonction boucle2() qui demande à l'utilisateur *via* un prompt de saisir la longueur souhaitée du tableau, puis crée ce tableau et utilise une boucle for pour le remplir de sorte que l'élément *i* du tableau contienne i^2. Affichez le tableau dans une boîte d'alerte.

Correction

```
<html lang="fr">
<head>
<title>Exercice sur les boucles for</title>
<meta charset="UTF-8"/>
<script>
function boucle(){
var tab = new Array(5);
for (var i=0;i<tab.length;i++)
{
tab[i]=i*i;
}
alert(tab);
}
function boucle2(){
var longueur=parseInt(prompt('Quelle est la longueur souhaitée
du tableau?'));
var tab= new Array();
for (var i=0;i<longueur;i++)
{
tab[i]=i*i;
}
alert(tab);
}
</script>
</head>
<body>
<h1>Boucles for</h1>
<button onclick="boucle()">Cliquez-moi!</button>
<button onclick="boucle2()">Cliquez-moi aussi!</button>
</body>
</html>
```

VI. Opérateurs

1. Opérateurs binaires mathématiques

Les opérateurs binaires sont ceux qui effectuent une opération sur deux variables ou données. Ces opérateurs sont :

- permet de faire la soustraction de deux nombres. JavaScript gère les conversions de types à la volée dans ce cas, et **"7"-2** renvoie **5**
- permet de faire la multiplication de deux nombres. **9*2** renvoie donc **18**. JavaScript gère aussi les conversions de types à la volée dans ce cas, et **"9"*2** renvoie aussi **18**.
- / permet de faire la division du premier nombre par le second. **9/2** renvoie donc **4.5**. JavaScript gère aussi les conversions de types à la volée dans ce cas, et **"7"/2** renvoie **3.5**.
- + permet dans le cas de deux nombres leur addition, et dans le cas de deux chaînes de caractères leur concaténation. Par exemple, **2+3** renvoie 5, mais **"2"+"3"** renvoie **"23"**. Si cet opérateur agit sur une chaîne et un nombre, alors ce dernier est transformé en chaîne : **"un"+2** renvoie donc **"un2"** et **2+"3"** renvoie **"23"**.
- **%** renvoie le reste de la division euclidienne de deux nombres. Par exemple, **9 % 2** renvoie **1**. JavaScript opère la conversion de type aussi dans ce sens : **"9" % 2** renvoie aussi **1**. Cette opérateur fonctionne aussi avec les nombres à virgule flottante : **20.4%4.8** renvoie **1.2** (ou plutôt **1.1999999999999993** selon les arrondis de calcul).

2. Opérateurs binaires de comparaison

Les opérateurs binaires de comparaison permettent de réaliser des tests sur des variables ou des données. Leur valeur de sortie est un booléen. Ces opérateurs sont :

- < permet de tester si le premier nombre est strictement inférieur au second : **1<2** renvoie **true**, mais **1<1**. renvoie **false**.
- <= permet de tester si le premier nombre est inférieur ou égal au second : **1<=2** renvoie **true** ainsi que **1<=1**.
- permet de tester si le premier nombre est strictement supérieur au second : **1>2** renvoie **false**.
- >= permet de tester si le premier nombre est supérieur ou égal au second : **1>=2** renvoie **false**.
- == permet de tester l'égalité de deux nombres : **1==2** renvoie **false**. JavaScript gère la conversion de type et **1=="1"** renvoie **true**.
- **!**= permet de tester si deux nombres sont différents : **1!=2** renvoie **true**.

3. Opérateurs logiques

Ces opérateurs renvoient aussi des booléens, et opèrent sur des booléens.

- **&&** est le "et logique", et renvoie **true** quand les deux quantités sur laquelle il opère valent **true**. Par exemple, **(1<2) && (chaine == "a")** renvoie **true** si la variable **chaine** vaut **"a"**, et **false** dans le cas contraire.
- || est le "ou logique", et renvoie **true** quand au moins une des deux quantités sur laquelle il opèrent vaut **true**. Par exemple, **(1<2) && (chaine == "a")** renvoie **true** dans tous les cas, car **1<2** est toujours vrai.
- **!** est l'opérateur logique de négation. Ainsi, **!(1<2)** renvoie **false**.

VII. Tests logiques

Les tests logiques permettent de réaliser des opérations en fonction des conditions dans lesquelles le code s'exécute. Par exemple, si l'utilisateur a entré un prénom féminin en guise d'identifiant, on peut envisager d'accorder les participes passés en fonction de ce choix.

1. Instruction if

L'instruction **if** est la plus simple possible. Elle permet de tester une condition unique. Par exemple,

```
if (choix==1)
{
alert("Vous avez fait le premier choix") ;
}
```

Elle peut être complétée par l'instruction **else**, qui permet d'indiquer le code à exécuter si la condition n'est pas remplie :

```
if (choix==1)
{
alert("Vous avez fait le premier choix") ;
}
else
{
alert("Vous n'avez pas fait le premier choix") ;
}
```

Exercice 1. Utilisation d'un test logique

Énoncé

Soit le code suivant

```
<html>
<head>
  <title>Utilisation d'un test logique</title>
  <meta  http-equiv="Content-Type"  content="text/html;  charset=ISO-
8859-1">
  <script type="text/javascript">
  // <!--
  // -->
  </script>
</head>
<body>
```

```
<h1>Utilisation d'un test logique</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="">Lancer la fonction</button></p>
</body>
</html>
```

1. Créez un tableau nommé *tab* dont le premier élément est -2, le deuxième 1 et le troisième 4
2. Créez une fonction soustrait prenant un argument *x* et renvoyant le résultat de la soustraction de *x*-2 si *x* est positif ou nul, la chaîne de caractères "Nombre négatif!" sinon.
3. Créez une fonction affiche, appelée au clic sur le bouton, qui affiche dans des boîtes d'alerte successivement le résultat de soustrait appliqué au premier élément, puis au dernier élément du tableau (en utilisant la propriété length).

Correction

```
<html>
<head>
<title>Utilisation d'un test logique</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<script type="text/javascript">
// <!--
var tab=[-2,1,4];
function soustrait(x)
{
if (x >= 0) return (x-2); else return "Nombre négatif!";
}
function affiche()
{
alert(soustrait(tab[0]));
alert(soustrait(tab[tab.length-1]));
}
// -->
</script>
</head>
<body>
```

```
<h1>Utilisation d'un test logique</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="affiche()">Lancer la fonction</button></p>
</body>
</html>
```

2. Instruction switch

Cette instruction permet de réaliser des branchements conditionnels. Elle s'utilise avec le mot-clef **case**. Chaque cas est analysé, puis si le test est concluant, la liste d'instructions suivant le **case** est exécutée. Cette exécution se poursuit jusqu'à rencontrer l'instruction **break**. On indique la liste des instructions à réaliser par défaut avec **default**. Par exemple,

switch (choix)
{
case 1:
alert("Vous avez fait le premier choix") ;
break ;
case 2:
alert("Vous avez fait le deuxième choix") ;
break ;
case 3:
alert("Vous avez fait le troisième choix") ;
break ;
default :
alert("Vous avez fait un choix au moins égal à 4") ;
}

Si on avait omis le **break** à la fin du troisième **case**, si l'utilisateur avait saisi 3 on aurait eu coup sur coup l'affichage des messages "Vous avez fait le troisième choix" et "Vous avez fait un choix au moins égal à 4".

Exercice 1. Utilisation de switch

Énoncé

Soit le code suivant :

```
<html>
<head>
<title>Utilisation de switch et de l'objet Date</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<script type="text/javascript">
// <!--
// -->
</script>
```

```
</head>
<body>
<h1>Utilisation de switch et de l'objet Date</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="">Lancer la fonction</button></p>
</body>
</html>
```

Au click sur le bouton, lancer la fonction jourDeLaSemaine(). Cette fonction détermine le jour de la semaine et affiche selon le cas dimanche, lundi, mardi... etc. jusqu'à samedi.

Correction

```
<html>
<head>
<title>Utilisation de switch et de l'objet Date</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
// <!--
function jourDeLaSemaine(){
var aujourdhui= new Date;
jour=aujourdhui.getDay();
var jourFrancais ;
switch (jour) {
case 0 : jourFrancais="dimanche";break;
case 1 : jourFrancais="lundi";break;
case 2 : jourFrancais="mardi";break;
case 3 : jourFrancais="mercredi";break;
case 4 : jourFrancais="jeudi";break;
case 5 : jourFrancais="vendredi";break;
case 6 : jourFrancais="samedi";break;
default: jourFrancais="jour inexistant"; break;
}
alert("Nous sommes "+jourFrancais+".");
}
// -->
</script>
</head>
<body>
<h1>Utilisation de switch et de l'objet Date</h1>
<p>Cliquez sur le bouton pour lancer la fonction : <button
onclick="jourDeLaSemaine()">Lancer la fonction</button></p>
</body>
</html>
```