Les fonctions et les méthodes

1. La déclaration d'une fonction

Une fonction est un groupe de lignes de code de programmation, écrites par le concepteur et destinées à exécuter une tâche bien spécifique. On peut, si besoin est, l'utiliser à plusieurs reprises dans la page. En outre, l'usage des fonctions améliore grandement la lisibilité de votre script.

Pour déclarer ou définir une fonction, on utilise le mot (réservé) `function`.

La syntaxe d'une déclaration de fonction est la suivante :

```
function nom_de_la_fonction(arguments) {
... code des instructions ...
}
```

Remarques :

- Le nom de la fonction suit les mêmes règles que celles qui régissent le nom des variables. Pour rappel, JavaScript est sensible à la casse. Ainsi, function_a() ne sera pas égal à Function_a().
- Tous les noms de fonctions dans un script doivent être uniques.
- La mention des arguments est facultative mais dans ce cas, les parenthèses doivent rester. C'est d'ailleurs grâce à ces parenthèses que l'interpréteur JavaScript distingue les variables des fonctions. Nous reviendrons plus en détail sur les arguments et autres paramètres.
- Lorsqu'une accolade est ouverte, elle doit impérativement, sous peine de message d'erreur, être refermée. Prenez la bonne habitude de fermer directement vos accolades et d'écrire le code entre elles. De nombreux scripts ne fonctionnent pas pour des erreurs d'accolades fermantes.

Le nombre d'accolades ouvertes (voir fonctions, tests conditionnels, etc.) doit toujours être égal au nombre d'accolades fermées.

- Le fait de définir une fonction n'entraîne pas l'exécution des commandes qui la composent. Ce n'est que lors de l'appel de la fonction que le code de programme est exécuté (voir point suivant).
- La déclaration de fonction se place souvent dans l'en-tête du document HTML soit entre les balises `<head> ... </head>`. Elle est ainsi toujours disponible et peut être appelée à tout moment dans le document.

Exemple :

```
<head>
<script>
function message(){
document.write("Bienvenue");
}
</script>
</head>
```

2. L'appel d'une fonction

Le script de la fonction ne s'exécute que lorsque celle-ci est appelée.

L'appel d'une fonction se fait par le nom de la fonction (avec les parenthèses). On y adjoint les arguments éventuels.

Soit par exemple :

```
<script>
message();
</script>
```

Autre possibilité, l'appel de la fonction se réalise souvent au moyen de gestionnaires d'événements comme le chargement de la page, le clic sur un bouton. Nous retenons ici l'événement `onload` qui appelle la fonction `message()` au chargement de la page.

```
<body onload="message();">
```

Il faut veiller à ce que la fonction soit déjà définie et connue de l'interpréteur avant son exécution.

La page complète serait :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
<script>
function message() {
document.write("Bienvenue");
}
</script>
</head>
<body onload="message();">
</body>
</html>
```

3. Le passage de paramètres

Il est possible de passer des paramètres à une fonction, c'est-à-dire de lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer les opérations programmées à l'aide de ces paramètres.

Lorsqu'on passe plusieurs paramètres à une fonction, il faut les séparer par des virgules, aussi bien dans la déclaration que dans l'appel.

Soit une fonction qui calcule le cube d'un nombre :

```
function calcul(nombre)
var cube = nombre*nombre*nombre;
}
```

Cette fonction doit avoir une valeur de départ pour effectuer son opération. Cette valeur lui est fournie en argument lors de l'appel de la fonction.

Lors de l'appel : `calcul(5);`

Ou sous forme de variable :

```
var nombre = 5
calcul(nombre);
```

Appliqué à notre exemple précédent, le script pourrait devenir :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
<script>
function message(texte) {
document.write(texte);
}
</script>
</head>
<body onload="message('Bienvenue');">
</body>
</html>
```

Nous avons entouré la chaîne de caractères 'Bienvenue' par des apostrophes car les guillemets étaient déjà utilisés pour l'attribut `onload="..."`.

4. Les variables locales et globales

Il existe deux types de variables, les variables locales et les variables globales. Une variable globale est accessible partout dans le script. Une variable locale n'est accessible que dans la fonction qui l'a créée.

C'est ce qu'on appelle la portée des variables. Cette distinction est source fréquente d'erreurs et est généralement difficile à déceler dans les scripts.

a. Variables locales

Une variable déclarée dans une fonction (donc à l'intérieur des accolades de la fonction) par le mot-clé var a une portée limitée à cette seule fonction. On ne peut pas exploiter cette variable ailleurs dans le script. On l'appelle donc variable locale.

```
function calcul(nombre)
var cube = nombre*nombre*nombre;
}
```

Ainsi, la variable cube est dans cet exemple une variable locale. Si vous y faites référence ailleurs dans le script, cette variable est inconnue pour l'interpréteur JavaScript (message d'erreur).

Si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée est globale une fois que la fonction a été exécutée.

```
function calcul(nombre) {
cube = nombre*nombre*nombre
}
```

La variable cube déclarée contextuellement est ici une variable globale après l'exécution de la fonction cube().

b. Variables globales

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions, sont toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle.

```
<script>
var cube=1
function calcul(nombre) {
var cube = nombre*nombre*nombre
}
</script>
```

La variable cube est bien globale.

Pour la facilité de gestion des variables, on ne peut que conseiller de les déclarer en début de script (comme dans la plupart des langages de programmation). Cette habitude vous met à l'abri de complications certaines.

5. L'instruction return

Selon l'écriture du code, le résultat d'une fonction peut être retourné par l'instruction return.

Soit la fonction multiple() qui calcule le produit de deux nombres (a et b).

```
function multiple(a,b) {
x = a*b;
return x;
}
```

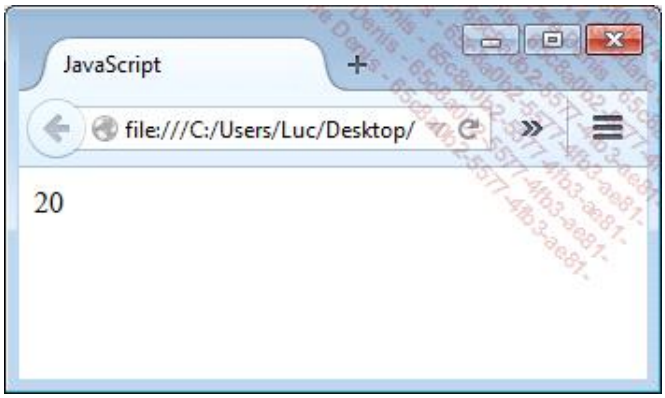
À l'appel de la fonction, on lui passe les paramètres correspondant aux arguments a et b, soit `multiple(4,5)`.

```
resultat=multiple(4,5);
```

Comme la valeur retournée par la fonction `multiple()` est 20, celle-ci est stockée dans la variable `resultat`.

Le script complet devient :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
<script>
function multiple(a,b) {
x = a*b;
return x;
}
</script>
</head>
<body>
<script>
resultat=multiple(4,5);
document.write(resultat);
</script>
</body>
</html>
```



6. Quelques méthodes JavaScript

Les méthodes sont des fonctions prédéfinies dans le langage JavaScript et dédiées à un objet particulier.

Nous avons jusqu'ici utilisé la méthode JavaScript `write()`, spécifique à l'objet `document`.

Passons en revue quelques autres méthodes (de l'objet `window`). Cela permet de varier nos exemples dans la suite de notre exposé.

a. alert()

La méthode `alert()` de l'objet fenêtre affiche une boîte de dialogue. Celle-ci comporte un message qui reproduit la valeur (variable et/ou chaîne de caractères) de l'argument qui lui a été fourni. Cette boîte de dialogue bloque le programme en cours tant que l'utilisateur n'aura pas cliqué sur **OK** pour fermer celle-ci.

Régal des programmeurs débutants en JavaScript, cette méthode est certes spectaculaire mais d'un rôle marginal dans un site Web. Par contre, `alert()` est très utile pour vous aider à déboguer les scripts et y retrouver d'éventuelles erreurs de programmation.

Sa syntaxe est :

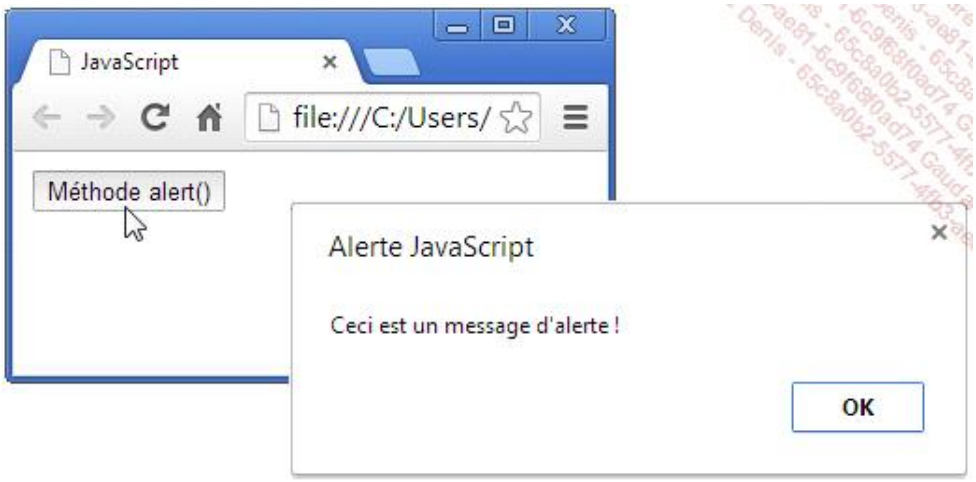
```
alert(variable);
alert("chaîne de caractères");
alert(variable + "chaîne de caractères");
```

Si vous souhaitez écrire sur plusieurs lignes, il faut utiliser le signe \n.

Exemple

Une boîte d'alerte va se déclencher lorsqu'on clique sur le bouton.

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
<script>
function alerte() {
alert ("Ceci est un message d'alerte !");
}
</script>
</head>
<body>
<form>
<input type=button value="Méthode alert()" onclick="alerte()">
</form>
</body>
</html>
```



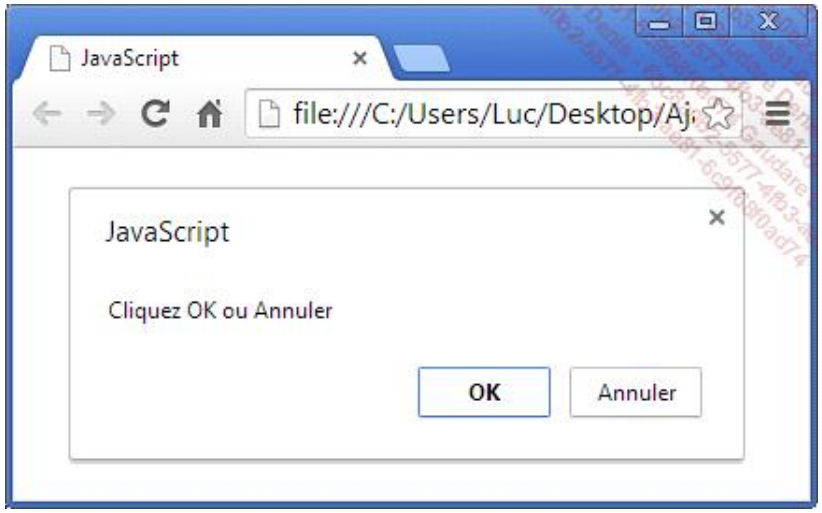
b. confirm()

Cette méthode de l'objet `window` affiche une boîte de dialogue avec deux boutons : **OK** et **Annuler**. En cliquant sur **OK**, la méthode renvoie la valeur `true` et bien entendu `false` si on a cliqué sur **Annuler**. Ce qui peut permettre, par exemple, de définir des options dans un programme.

Exemple

Le script lance une boîte de confirmation. La valeur renvoyée est mise dans la variable `a`. Si la valeur de `a` est `true` (`if(a == true)`), une boîte d'alerte s'affiche. Si ce n'est pas le cas, une autre boîte de dialogue s'affiche.

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
</head>
<body>
<script>
var a = confirm("Cliquez OK ou Annuler");
if(a == true){
alert ("Vous avez cliqué OK");
}
else {
alert ("Vous avez cliqué Annuler");
}
</script>
</body>
</html>
```



c. prompt()

Dans le même style que les précédentes méthodes de l'objet window, JavaScript vous propose une autre boîte de dialogue, appelée boîte d'invite. Elle est composée d'un champ comportant une entrée à compléter par l'utilisateur. Cette entrée peut aussi posséder une valeur par défaut.

La syntaxe est :

```
prompt("texte de la boîte d'invite","valeur par défaut");
```

En cliquant sur **OK**, la méthode renvoie la valeur saisie par l'utilisateur ou la valeur proposée par défaut. Si l'utilisateur clique sur **Annuler**, la valeur `null` est alors renvoyée.

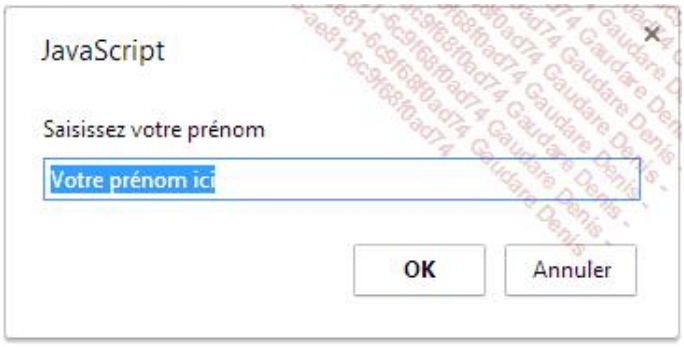
La méthode `prompt()` est parfois utilisée pour saisir des données fournies par l'utilisateur.

Exemple

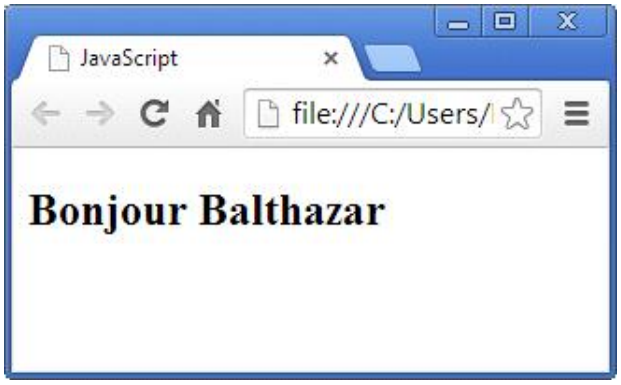
On va demander le prénom du visiteur par une boîte d'invite et l'afficher sur la page web.

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
</head>
<body>
<script>
var prenom = prompt("Saisissez votre prénom", "Votre prénom ici");
document.write("<h2>Bonjour " + prenom + "</h2>");
</script>
</body>
</html>
```

La boîte d'invite :



Le document HTML :



d. setTimeout()

JavaScript met à votre disposition une minuterie (ou plus précisément un compte à rebours) qui permet de déclencher une fonction après un laps de temps déterminé.

La syntaxe de mise en route du temporisateur est :

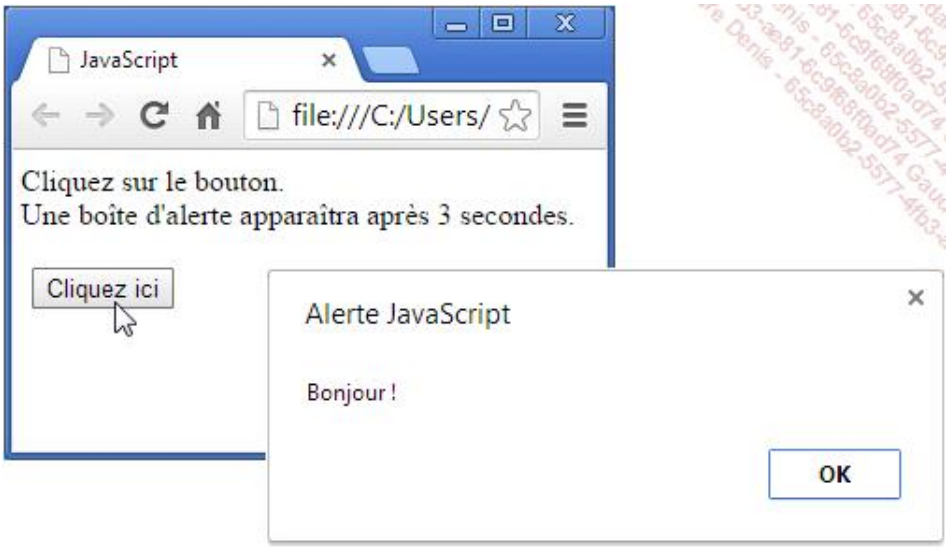
```
nom_du_compteur = setTimeout("fonction_appelée()", temps en millisecondes)
```

Ainsi, `setTimeout("demarrer()",5000)` va lancer la fonction `demarrer()` après 5 secondes.

Exemple

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>JavaScript</title>
<meta charset="UTF-8">
<script>
function display(){
alert("Bonjour !")
}
</script>
</head>
<body>
Cliquez sur le bouton.<br>
Une boîte d’alerte apparaîtra après 3 secondes.
```

```
<form>
<input type="button" onclick="setTimeout('display()',3000)"
value="Cliquez ici">
</form>
</body>
</html>
```



Pour arrêter le temporisateur avant l'expiration du délai fixé, il suffit d'utiliser la méthode `clearTimeout(nom_du_compteur)`.