07 Fonctions

Définition

Une fonction est un groupe de ligne(s) de code de programmation (un « sous-programme ») destiné à exécuter une ou plusieurs tâches et que l'on pourra utiliser à plusieurs reprises (factorisation du code).

En outre, l'usage des fonctions améliorera grandement la lisibilité de votre script.

En JavaScript, comme d'ailleurs dans tous les langages, il existe deux types de fonctions :

- Les fonctions propres à JavaScript (appelées aussi fonctions *natives*). On les appelle aussi des *méthodes*. Elles sont associées à un objet bien particulier comme c'est le cas de la méthode `alert()` avec l'objet `window`.
- Les fonctions que vous écrivez pour les besoins de votre script et que nous abordons dans ce cours.

Déclaration des fonctions

Pour déclarer ou définir une fonction, on utilise l'instruction `function`.

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

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

Le nom de la fonction suit les mêmes règles régissant le nom de variables (nombre de caractères indéfini, commence par une lettre, peut inclure des chiffres...). Pour rappel, JavaScript est sensible à la case. Ainsi `mafonction()` ne sera pas égal à `maFonction()`. En outre, tous les noms des 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 dans la partie JavaScript avancé.

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 votre code entre elles.

Appel d'une fonction

L'appel d'une fonction se fait simplement par le nom de la fonction (avec les parenthèses) : par exemple `nom_de_la_fonction()`.

Il faudra veiller en toute logique (car l'interpréteur lit votre script de haut vers le bas) à ce que votre fonction soit bien définie avant d'être appelée.

Passer une valeur à une fonction

On peut passer des valeurs aux fonctions JavaScript. La valeur ainsi passée sera utilisée par la fonction.

Pour passer une valeur, aussi appelé **paramètre** ou **argument** à une fonction, on fournit le nom d'une variable dans la déclaration de la fonction.

Exemple :

Ecrire une fonction qui affiche un message d’alerte dont le texte peut changer :

```
function exemple(texte)
{
   alert(texte);
}
```

Le nom de la variable est *texte* et est définie comme un paramètre de la fonction.

Dans l'appel de la fonction, on lui fournit le texte :

exemple("Salut à tous");

La fonction `exemple()` pourrait être appelée sur l'événement onLoad de la balise .

Arguments multiples

On peut passer bien sûr un ou plusieurs arguments : dans ce cas ils doivent être séparés par des virgules. Les types peuvent être différents.

```
/* Arguments multiples */
function maFonction(arg1, arg2, arg3)
{
    ... code des instructions ...
}
```

Et pour l'appel de la fonction :

```
maFonction(arg1, arg2, arg3);
```

Arguments facultatifs et affectation par défaut

En Javascript, tous les arguments d’une fonction sont facultatifs.

```
function maFonction(obligatoire, facultatif)
{
   // Affiche 'Argument 1 : Paul'
   console.log('Argument 1 : '+obligatoire);

   // 1er appel : erreur car 2ème argument non envoyé
   // 2ème  appel : affiche 'Argument 2 (facultatif) : Anne'
   console.log('Argument 2 (facultatif) : '+facultatif);
}

maFonction('Paul'); // 1er appel

maFonction('Paul', 'Anne'); // 2ème appel
```

Afin d’éviter l'erreur provoquée parce que l’on a un argument non défini, on va ajouter au sein de la fonction une condition de test sur le type des arguments facultatifs. On pourra alors par la même occasion affecter une valeur par défaut aux arguments que l’on souhaite rendre facultatifs. Dans l'exemple ci-dessous l’argument facultatif reçoit la valeur Anne :

```
function maFonction(obligatoire, facultatif)
{
   if (typeof facultatif == 'undefined')
   {
      facultatif = 'Anne';
   }

   // Affiche 'Argument 1 : Paul'
   console.log('Argument 1 : '+obligatoire);

    // 1er appel : erreur car 2ème argument non envoyé
   // 2ème  appel : affiche 'Argument 2 (facultatif) : Anne'
   console.log('Argument 2 (facultatif) : '+facultatif);
}

maFonction('Paul'); // 1er appel

maFonction('Paul', 'Anne'); // 2ème appel
```

Retourner une valeur

Le principe est simple (la pratique parfois moins). Pour renvoyer un résultat, il suffit d'écrire l'instruction `return` suivi de l'expression à renvoyer. Notez qu'il ne faut pas entourer l'expression de parenthèses.

Exemple :

```
// Fonction qui retourne le carré d'un nombre
function carre(nombre)
{
  let resultat = nombre*nombre;
  return resultat;
}
```

Dans ce cas, il faut appeler la fonction en affectant une variable qui stockera le résultat retourné par la fonction :

```
let resultat2 = carre(nombre);
```

Notez que la variable qui stocke le retour n'a pas pour obligation de porter le même nom que celui de la variable qui suit l'instruction return. Précisons que l'instruction return est facultative et qu'on peut trouver plusieurs return dans une même fonction (par exemple dans un bloc de conditions où chaque cas renvoie quelque chose). S'il n'y a aucune instruction return, l'accolade fermante s'y substitue dans certains cas selon la portée des variables (cf. paragraphe suivant).

Portée des variables

Avec les fonctions, le bon usage des variables locales et globales prend toute son importance. Une variable déclarée dans une fonction par le mot clé `let` aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script : il s'agit donc d'une variable locale :

```
function carre(nombre)
{
    let resultat = nombre*nombre;
}
```

Par contre, si la variable est déclarée contextuellement (c'est-à-dire sans le mot-clé `let`) dans une fonction, sa portée devient globale une fois que la fonction aura été exécutée : l'instruction `return` devient alors facultative.

```
function cube(nombre)
{
    resultat = nombre*nombre*nombre;
}
```

La variable resultat déclarée contextuellement sera ici une variable globale.

Exercice

Testez les exemples ci-dessous :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Exercice</title>
  </head>
  <body>
    <h1>Portée d'une variable dans une fonction | Javascript</h1>
    <hr>
    <script>
      function maFonction() {
        let plop1 = 123;
        return plop1;
      }

      function maFonction2() {
        plop2 = 456; // variable globale implicite (à éviter !)
      }

      plop1 = maFonction();
      console.log("fonction 1 / plop1 : " + plop1);

      maFonction2();
      console.log("maFonction2 > plop2 : " + plop2);
    </script>
  </body>
</html>
```

Expressions de fonctions

Une fonction peut être déclarée avec un nom précédé du mot clé `let`. On appelle cela une expression de fonction. On peut ensuite appeler la fonction grâce à ce nom :

```
// Déclaration de la fonction carre() :
let carre = function(nombre)
{
    let resultat = nombre*nombre;
}

// Appel de la fonction carre() :
let k = carre(2);
```

Il s'agit donc là d'une syntaxe alternative pour nommer une fonction. Les expressions de fonction trouvent leur utilité dans les cas de récursivité ou pour passer une fonction en argument à une fonction.

Récursivité

Une fonction peut faire référence à elle-même et s'appeler elle-même. Une fonction qui s'appelle elle-même est appelée une fonction récursive. Sous certains aspects, une récursion est semblable à une boucle : toutes les deux exécutent le même code plusieurs fois et toutes les deux requièrent une condition d'arrêt (pour éviter une boucle ou une récursion infinie).

Par exemple, ce code utilise une boucle :

```
let x = 0;

while (x < 5)
{
    console.log("x : "+x);
    x++;
}
```

On pourra convertir ce code en une fonction récursive de la façon suivante :

```
function boucle(x)
{
    if (x >= 10)
    {
        return;
    }

    console.log("x : "+x);

    boucle(x + 1); // appel récursif
}

boucle(0); // appel initial de la fonction
```

La récursivité est souvent utilisée pour parcourir une arborescence (par exemple le D.O.M.).

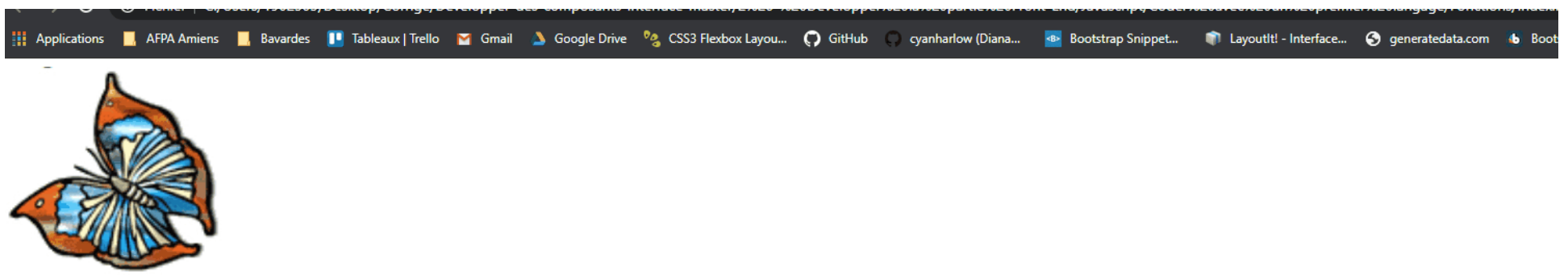
Exercices

Exercice 1

Créer les 2 fonctions suivantes :

- `produit(x, y)` qui retourne le produit des 2 variables `x`, `y` passées en paramètre.
- `afficheImg(image)` qui affiche l'image passée en paramètre. (Le paramètre image correspond au chemin de votre image)

Créer la page HTML correspondant au résultat ci-dessous :



Vous aurez besoin de cette image

Ressource

Veuillez utiliser la propriété **innerHTML** pour réaliser cet exercice :

- HTML DOM innerHTML

Exercice 2 - String Token

Concevez la fonction `strtok()` qui prend 3 paramètres `str1`, `str2`, `n` en entrée et renvoie une chaîne de caractères : str1 est composée d'une liste de mots séparés par le caractère `str2`. `strtok()` sert à extraire le nième mot de `str1`.

Exemple :

Pour *str1 = « robert ;dupont ;amiens ;80000 », strtok (str1, « ; », 3)* doit retourner *amiens*.

Indice : *utilisez la méthode* `split()`.