#5 : Javascript

1) Introduction

Javascript est un langage de programmation de script très utilisé dans le développement des sites web. Il s'exécute dans presque tous les navigateurs web.

Ce langage a été créé en 1995 pour le compte de la société Netscape. Malgré son nom, il n'a presque rien à voir avec le langage Java développée par Sun.

En ce qui nous concerne, nous n'allons pas (pour l'instant) nous intéresser au développement web avec JavaScript, mais apprendre les bases de la programmation (variable, fonction, condition, boucle). Malgré tout, puisque JavaScript s'exécute dans un navigateur web (pour nous Firefox) nous allons devoir écrire quelques lignes de HTML.

2) Premier programme

Nous allons créer une page **HTML5** nommée **index.html** en récupérant le fichier **modele.html** des séances précédentes. Ajoutons ensuite une ligne de code qui va "demander" à la page web d'exécuter du code JavaScript :

```
<!doctype html>
<html lang="fr">
<head>
      <meta charset="utf-8">
      <title>Programmation JavaScript</title>
      <script src="monProgramme.js"></script>
</head>
<body>
</body>
</html>
```

Nous avons donc ajouté la balise <**script**>, cette balise accepte un attribut "**src**" qui correspond au chemin du fichier JavaScript (extension .js) qui doit être exécuté. Dans notre exemple, notre fichier "JavaScript" sera dans le même dossier que notre fichier "HTML" et se nommera "**monProgramme.js**". Nous n'aurons plus à modifier ce fichier (sauf si vous décidez de modifier le nom du fichier JavaScript).

Nous allons maintenant créer et enregistrer notre fichier "**monProgramme.js**" dans notepad++ :

```
document.write("Hello World !");
```

Enregistrez votre fichier et ouvrez le fichier HTML que nous avons créé à l'aide du navigateur Firefox. Observez le résultat.

À ce stade, vous devez juste avoir compris que le code **document.write** vous permet d'afficher la chaîne de caractère contenue entre les guillemets (dans notre exemple : Hello World !). Il est aussi important de noter qu'**en JavaScript, une ligne de code doit se terminer par un point virgule**.

Pour votre information, il est aussi possible d'inclure le code JavaScript directement dans le code HTML :

```
<!doctype html>
<html lang="fr">
<head>
      <meta charset="utf-8">
      <title>Programmation JavaScript</title>
      <script>
            document.write ("Hello World !");
      </script>
</head>
<body>
</body>
</html>
```

Nous n'utiliserons pas cette méthode, le mélange code HTML et code JavaScript dans un même fichier risque de compliquer inutilement les choses, surtout pour un débutant. Par la suite, je ne vous reparlerai plus du fichier HTML qui restera identique, nous nous concentrerons uniquement sur notre fichier JavaScript.

3) Les variables en JavaScript

Vous devez tout d'abord **déclarer** votre variable en utilisant le mot clé var.

```
var point_de_vie;
```

Puis vous pouvez **attribuer** une valeur à votre variable

```
point_de_vie=15;
```

Ces 2 actions peuvent être couplées :

```
var point_de_vie=15;
```

Voici un premier exemple : modifiez et enregistrez notre fichier "**monProgramme.js**":

```
var point_de_vie=15;
document.write(point_de_vie);
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

On peut aussi améliorer le message avec le code suivant :

```
var point_de_vie=15;
document.write("point_de_vie a pour valeur ", point_de_vie);
```

Remarquez juste la virgule entre la chaîne de caractères (entre guillemets) et la variable (sans guillemets). Je pense que vous avez tous compris qu'au moment de l'exécution du code le mot **point_de_vie** est remplacé par la **valeur contenue dans point_de_vie** (ici 15).

JavaScript est un langage **faiblement typé**, il n'est donc pas nécessaire de préciser le type de la variable. Attention, cela ne veut pas dire que votre variable n'a pas de type, juste que le programmeur n'a pas besoin de le préciser.

En JavaScript les types possibles sont : **string** (chaîne de caractères), **boolean,** les nombres (**number**) qui regroupent les entiers (type **integer**) et les nombres à virgule (type **float**). Attention pour le type **float** vous devez **utiliser le point à la place de la virgule**, par exemple le nombre pi ne s'écrit pas 3,14 mais 3.14 (convention anglo-saxonne)).

La fonction **typeof**() renvoie le type de la variable qui a été passé comme argument.

```
var a=4;
document.write ("a a pour valeur ", a, ". Elle est de type ", typeof(a), "</br>");

var b="Hello";
document.write ("b a pour valeur ", b, ". Elle est de type ", typeof(b), "</br>");

var c=true;
document.write ("c a pour valeur ", c, ". Elle est de type ", typeof(c), "</br>");

var d;
document.write ("d a pour valeur ", d, ". Elle est de type ", typeof(d), "</br>");
```

Vous avez dû noter que **pour une variable de type string, la valeur est entre guillemets**.

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Deux choses à bien noter dans cet exemple :

- Vous avez sans doute reconnu la balise **</br>** ("retour à ligne") du HTML. Sans trop entrer dans les détails, document.write vous permet d'écrire du code HTML, il est donc logique d'utiliser la balise **</br>** pour effectuer un retour à la ligne (toute autre balise est aussi utilisable, essayez avec une balise **** par exemple). Enlevez les **</br>** du code pour vous convaincre de leur utilité. Enfin, attention, tout comme le texte, les balises HTML doivent être entre guillemets. La variable et la fonction **typeof** ne sont pas entre guillemets.
- Une variable quand elle a été déclaré, mais qu'aucune valeur ne lui a été attribué, a pour valeur **undefined** et est de **type undefined**.

❑**Exercice** : Faites un nouveau programme javascript qui crée 2 variables contenant pour l'une le texte **« Bonjour, j'ai »** et pour l'autre le nombre **20**. Ensuite le programme doit afficher le contenu de ces variables pour qu'on voit : « Bonjour, j'ai 20 ans bientôt ! ».

Que se passe-t-il quand on utilise une variable qui n'a même pas été déclaré ?

```
var a=4;
document.write ("a a pour valeur ", a);
document.write ("b a pour valeur ", b);
var c="Hello";
document.write ("c a pour valeur ", c);
```

1ère ligne nous **déclarons** la variable a et nous lui **attribuons** la valeur (numérique) 4.

2ème ligne, nous **utilisons** la variable a

3ème ligne nous **utilisons** une variable b qui n'a pas été déclaré

4ème ligne nous **déclarons** la variable c et nous lui **attribuons** la valeur (chaîne) "Hello"

5ème ligne nous **utilisons** c.

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Le programme s'est arrêté à la 3ème ligne. Utiliser une variable non déclarée est une erreur qui provoque l'arrêt du programme.

Problème, rien ne nous signale cette erreur dans le navigateur (c'est assez logique, à la base un navigateur n'est pas un outil de développement !). Nous pourrions installer une extension pour Firefox (firebug) qui nous nous donnerait des informations sur les éventuelles erreurs (on parle de bug).

4) Les boites de dialogue

Pour débugger des programmes il suffit souvent de placer des alertes à différents endroits du code.

```
alert("Une alerte simple");
var myText="Une alerte avec le message dans une variable";
alert(text);
var myNumber=13;
alert("La variable contient la valeur" + myNumber);
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat. Notez qu'ici pour la dernière ligne de code l'opérateur + est un **opérateur de concaténation**. En effet ajouter deux chaînes de caractères revient à **les coller à la suite l'une de l'autre : c'est une concaténation**.

Nous allons maintenant étudier la méthode qui permet à l'utilisateur de **rentrer des valeurs au clavier** : la méthode **prompt**(). Nous allons utiliser une structure de la forme : **var maVariable = prompt (message)**.

En réponse à la méthode prompt, le navigateur affichera une fenêtre avec : un bouton OK, un bouton Annuler, un message et une zone de saisie.

L'utilisateur va alors saisir (au clavier) du texte dans la zone de saisie. La validation avec le bouton OK permettra d'attribuer le texte entré par l'utilisateur à la variable maVariable. Au cas où l'utilisateur ne rentrerai rien ou qu'il appuierai sur Annuler on aura alors maVariable = null (pas de valeur). Voici un exemple :

```
var prenom=prompt("Quel est votre prénom ?");
document.write ("Bonjour ", prenom, ", vous allez bien ?");
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat. La fenêtre s'affiche, l'utilisateur entre son prénom et appuie sur OK, puis le message s'affiche sur la page.

Testez ce qui se passe si l'utilisateur n'entre rien (ou appuie sur le bouton Annuler).

Pour éviter ce genre de chose, nous verrons un peu plus loin l'utilisation du couple if/else (les conditions)

Un autre exemple : une machine à additionner

```
document.write ("Nous allons additionner 2 nombres, a et b </br>");
var a=prompt("Entrer a ");
var b=prompt("Entrer b ");
var resultat=a+b;
document.write ("Résultat ",a," + ",b," = ",resultat);
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Pour essayer de comprendre ce qui se passe, faites un "**typeof**()" sur les variables a et b.

Les variables a et b sont toutes deux de type **string**. Or, nous avons vu que si nous avons affaire à des chaînes de caractère le **signe + est le signe de concaténation** (mise bout à bout de 2 chaînes de caractère). Si nous mettons bout à bout 5 et 15, nous obtenons bien 515.

Pour que notre programme fonctionne, il faut "transformer" notre **chaîne** (variable de type **string**) en **nombre** (variable de type **integer**). Nous allons faire du **transtypage (convertir un type en un autre)**.

Pour se faire, utilisons la méthode **parseInt**() :

```
document.write ("Nous allons additionner 2 nombres, a et b </br>");
var as=prompt("Entrer a ");
var bs=prompt("Entrer b ");
var a=parseInt(as);
var b=parseInt(bs);
var resultat=a+b;
document.write ("Résultat ",a," + ",b," = ",resultat);
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat. as et bs sont de type string, a et b sont de type nombre, le résultat est maintenant correct.

5) Les tests conditionnels : if {} else {}

Vous avez déjà eu l'occasion d'aborder les tests conditionnels dans le cours de mathématiques. Revenons sur cette structure :

```
si (condition vraie) fait { instructions }
sinon (sous entendu la condition est fausse) fait { instructions }
```

Prenons un exemple en JavaScript :

```
var a=5;
if (a==5)
{
      document.write("a est égale à 5");
}
else
{
      document.write("a n'est pas égale à 5");
}
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Modifiez la valeur de la variable a pour afficher l'autre message. Testez. Voici la liste des opérateurs de comparaison :

Égalité : ==	supérieur ou égal : >=
strictement supérieur : >	inférieur ou égal : <=
strictement inférieur : <	différent de : !=

Il est aussi possible de faire des combinaisons de tests avec le ET logique (&&) ou le OU logique (||).

```
var valeur=prompt("Entrez un chiffre entre 5 (exclue) et 10 (incluse)");
if ((valeur>5) && (valeur<=10))
{
      document.write("Bravo, vous avez réussi !");
}
else
{
      document.write("Raté");
}
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

- **Exercice** : Faites un nouveau programme javascript qui demande de rentrer un nombre premier et qui répond **Gagné** ou **Perdu** après avoir testé si le nombre donné est bien premier.

6) Les tableaux

Les tableaux vont vous permettre de stocker plusieurs valeurs (chaîne, nombre) dans une structure unique.

Pour déclarer un tableau, la syntaxe est un peu particulière : var monTableau = new Array();

Le mot clé new est utilisé pour créer des objets (au sens informatique du terme, nous aurons l'occasion de revenir sur cette notion d'objet). Un tableau est donc un objet.

Ensuite, pour remplir le tableau il faut procéder comme suit : monTableau [indice de position] = maValeur

```
monTableau [0] = "pomme";
monTableau [1] = "orange";
```

ou alors avec des nombres :

```
monTableau [0] = 15;
monTableau [1] = 20;
```

pomme aura l'indice 0, orange aura l'indice 1, etc. **L'indice de position commence toujours à zéro.**

Il est aussi possible de déclarer et de remplir le tableau en même temps :

```
monTableau = new Array ("pomme", "orange", "cerise");
```

Un exemple à essayer :

```
var mesFruits = new Array ("orange","pomme","cerise");
document.write (mesFruits[0], " : indice de position 0 </br>");
document.write (mesFruits[1], " : indice de position 1</br>");
document.write (mesFruits[2], " : indice de position 2</br>");
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Il est possible de connaître la taille de votre tableau avec la méthode **length** :

```
var mesFruits = new Array ("orange","pomme","cerise");
document.write (mesFruits[0], " : indice de position 0 </br>");
document.write (mesFruits[1], " : indice de position 1</br>");
document.write (mesFruits[2], " : indice de position 2</br>");
document.write ("Ce tableau possède ", mesFruits.length, " éléments");
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Notez bien qu'il est possible de déclarer et de "remplir" un tableau comme suit :

```
var mesFruits=["orange","pomme","cerise"];
```

La méthode typeof est aussi utilisable avec un tableau.

Il est possible de remplacer l'indice de position par une variable

```
var mesFruits = new Array ("orange","pomme","cerise");
var i=prompt("Choisir un fruit (0 une orange ; 1 une pomme ; 2 une cerise)");
document.write ("Votre choix : une ", mesFruits[i]);
```

Vous ne trouvez pas cela étrange que cela fonctionne ? Comme nous l'avons déjà vu, prompt renvoie une chaîne de caractères, donc ici, i est de type string ! Or l'indice de position doit être de type **number** et pas de type string ! Avec mesFruits[i] nous devrions avoir une erreur.

Eh non, pas d'erreur, car JavaScript est capable de faire du "**transtypage automatique**" : étudions le "raisonnement" de JavaScript : "i est de type string, mais dans mesFruits[i] il doit être de type **number**, de mon propre chef j'applique donc un "**parseInt**" à i". Voilà pourquoi nous n'avons pas d'erreur !

Pour terminer sur les tableaux, nous verrons un peu plus loin que l'instruction **for...in** nous permettra de parcourir un tableau quasi automatiquement.

7) Les fonctions en JavaScript

Pour définir une fonction en JavaScript il faut utiliser le mot clé **function**. Nous allons avoir une structure de la forme :

```
function maFonction(paramètre1, paramètre2,…)
{
	...instructions..
	return y ;
}
```

Essayons un exemple :

```
function calcul (x)
{
	var y;
	y=3*x+2;
	return y;
}
document.write ("y=3x+2 avec x = 4 </br>");
document.write ("Résultat : y = ", calcul(4));
```

Remarquez l'indentation (décalage) du contenu de la fonction (entre l'accolade ouvrante et fermante). Cela a pour but de rendre le code plus lisible. Ce n'est pas obligatoire, mais fortement conseillé.

calcul(4) appelle la fonction **calcul** avec un paramètre (égal à 4). Dans le document.write de la dernière ligne, **calcul(4)** est "remplacé" par la valeur retournée (**return y**) par la fonction calcul.

Essayons un autre exemple un peu plus évolué :

```
function calcul (x)
{
	var y;
	y=3*x+2;
	return y;
}
var valeur = prompt ("y=3x+2, entrez la valeur de x");
document.write ("y=3x+2 avec x = ", valeur, "</br>");
document.write ("Résultat : y = ",calcul(valeur));
```

Ici la valeur du paramètre de la fonction calcul est entrée par l'utilisateur. Pour le reste, je vous laisse analyser le code (n'hésitez pas à poser des questions en cas de problème).

Comme déjà dit précédemment, une fonction peut très bien attendre plusieurs paramètres, essayons ceci :

```
function calcul (x,b)
{
	var y;
	y=3*x+b;
	return y;
}
var valeur1 = prompt ("y=3x+b, entrez la valeur de x");
var valeur2 = prompt ("y=3x+b, entrez la valeur de b");
valeur1=parseInt(valeur1);
valeur2=parseInt(valeur2);
document.write ("y=3x+b avec x = ", valeur1, " et b = ", valeur2, "</br>");
document.write ("Résultat : y = ", calcul(valeur1, valeur2));
```

Expliquez le rôle des lignes contenant les parseInt() :

_ _

_ _

_ _

Enlevez ces 2 lignes et relancez votre test puis expliquez le résultat obtenu :

_ _

_ _

_ _

_ _

Évidemment, le paramètre de la fonction peut-être de type **string**. Il faut aussi savoir que le return n'est pas obligatoire (une fonction peut ne rien renvoyer, juste "faire quelque chose") :

```
function afficheNom (nom)
{
      document.write("Bonjour", nom);
}
afficheNom("Toto");
```

Un simple afficheNom("Toto"); vous permet d'appeler la fonction afficheNom, qui ne renvoie rien, mais qui affiche du texte.

Pour terminer cette partie sur les fonctions et le JavaScript, étudions la différence entre les variables **locales** et les variables **globales** (on parle de la portée d'une variable).

Si vous essayez de faire fonctionner le programme suivant, vous aurez droit à une "belle" erreur, pourquoi ?

```
function maFonction ()
{
      var i=10;
}
document.write ("Voici la valeur de la variable i : ", i);
```

La variable i a été défini dans une fonction, donc elle n'existe pas en dehors de cette fonction.

Une variable définie en dehors d'une fonction est appelée variable globale, elle est accessible partout dans le programme. Il faut faire très attention avec cette histoire de variable locale, vous pouvez vous retrouver devant des bugs difficilement détectables essayez par exemple :

```
var i=5
function changerLaValeurDei (x)
{
      var i=5*x;
}

var a=prompt("Entrer une valeur pour x");
changerLaValeurDei(a);
document.write ("Voici la nouvelle valeur de la variable i : ", i);
```

Expliquez pourquoi ce programme ne change pas la valeur de la variable :

_ _

_ _

_ _

_ _

_ _

_ _

8) La boucle while (tant que)

La notion de boucle est fondamentale en informatique. Une boucle permet d'exécuter plusieurs fois des instructions. La structure de la boucle while est la suivante :

```
while (condition)
{
      instructions à répéter
}
```

Tant que la **condition** reste vraie, les instructions à l'intérieur du bloc (entre les 2 accolades) seront exécutées.

```
var i=0;
while (i<=10)
{
      document.write("i vaut ",i,"</br>");
      i=i+1;
}
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et expliquez le résultat.

_ _

_ _

J'attire votre attention sur la ligne i=i+1 : Nous avons ici aussi un exemple de la non-équivalence du signe égal en mathématiques et en informatique. En mathématiques, écrire i=i+1 n'a aucun sens (cela reviendrait à dire par exemple que $5 = 6$!).

Ici i=i+1 veut dire : "Prends la valeur de i, ajoute un à cette valeur puis attribue cette nouvelle valeur à la variable i", on parle **d'affectation**. Dans le cas spécial où on ajoute une constante à une variable on parle **d'incrémentation**.

Il est tellement courant d'utiliser i=i+1 que les informaticiens ont inventé "un raccourci" : **i++**

Voici un exemple un peu plus complet (table de multiplication) :

```
function table (a)
{
      var i=1;
      document.write("TABLE DES ",a, "</br>");

      while (i<=10)
      {
            var resultat=a*i;
            document.write(a," x ",i," = ",resultat,"</br>");
            i++;
      }
}

var numS=prompt("Entrez un chiffre entre 1 et 10");
var num=parseInt(numS);

while ((num<1) || (num>10) || (isNaN(num)))
{
      numS=prompt("Entrez un chiffre entre 1 et 10");
      num=parseInt(numS);
}

table(num);
```

Petite nouveauté dans cet exemple, la fonction isNaN(num).

Si la variable num n'est pas de type nombre, cette fonction renvoie vraie (true). Si la variable num est de type nombre, la fonction **isNaN** renvoie alors faux (false).

Vous pouvez aussi constater que notre code débute par l'écriture d'une fonction. Si vous avez toute une série de fonctions à écrire, il est préférable de toutes les écrire en début de code.

Expliquez ce que fait la fonction **table** :

_ _

_ _

_ _

Expliquez le fonctionnement de la boucle **while** :

_ _

_ _

_ _

9) La boucle do while

C'est un peu la "cousine" de la boucle while, voici sa structure :

```
do
{
      instructions à répéter
}
while (condition);
```

Ici les instructions sont forcément exécutées au moins une fois (ce qui n'est pas le cas pour la boucle while).

Voici le même exemple que précédemment, mais avec une boucle do while :

```
function table (a)
{
      var i=1;
      document.write("TABLE DES ",a, "</br>");
      while (i<=10)
      {
            var resultat=a*i;
            document.write(a," x ",i," = ",resultat,"</br>");
            i++;
      }
}

do
{
var numS=prompt("Entrez un chiffre entre 1 et 10");
var num=parseInt(numS);
}
while ((num<1) || (num>10) || (isNaN(num)));

table(num);
```

Vous voyez que dans ce cas précis, l'utilisation de la boucle do while simplifie un peu le code.

10) La boucles for (boucle comptée)

Nous avons toujours ici affaire à une boucle, mais les conditions de répétitions sont un peu différentes :

```
for (début ; condition de répétition ; incrémentation)
{
      instructions à répéter
}
```

Reprenons notre exemple des tables de multiplication avec une boucle for, testez ceci :

```
function table (a)
{
      document.write("TABLE DES ",a, "</br>");
      for ( i=1 ; i<=10 ; i++ )
      {
            var resultat=a*i;
            document.write(a," x ",i," = ",resultat,"</br>");
      }
}

do
{
var numS=prompt("Entrez un chiffre entre 1 et 10");
var num=parseInt(numS);
}
while ((num<1) || (num>10) || (isNaN(num)));

table(num);
```

La boucle commence avec i = 1, elle va s'exécuter tant que i<=10 reste vraie, i est augmenté d'une unité à chaque boucle.

11) La boucle for in (énumération)

for in va nous permettre de parcourir un tableau de façon "quasi automatique", testez cet exemple :

```
var fruit=new Array("orange", "cerise", "pomme", "pêche", "poire");
document.write("Liste des fruits </br>");

for (var i in fruit)
{
      document.write(fruit[i], " , ");
}
```

Voici un dernier exemple que je vous laisse tester :

```
var fruit=new Array();
var i=0;
do
{
      fruit[i]=prompt("Entrez un nom de fruit (pour terminer taper fin)");
      i++;
}
while (fruit[i-1]!="fin");

document.write("Voici votre liste de fruits </br>");

for (var j=0 ; j<(fruit.length-1) ; j++)
{
      if (j==(fruit.length-2))
      {
            document.write(fruit[j]);
      }
      else
      {
            document.write(fruit[j], " , ");
      }
}
```

Pourquoi le fruit[i-1] ?

_ _
_ _
_ _
_ _

Pourquoi le fruit.length-1 ?

_ _
_ _
_ _
_ _

Pourquoi le fruit.length-2 ?

_ _
_ _
_ _
_ _

12) Les commentaires en JavaScript

Nos programmes commencent à être de plus en plus complexes. Pour qu'ils restent malgré tout le plus clair possible, nous allons systématiquement ajouter des commentaires. Ces commentaires auront principalement 2 buts :

- aider une tierce personne à comprendre votre code
- assurer une maintenance efficace de votre code (il est parfois difficile de comprendre son propre code des mois après son écriture).

En JavaScript, il existe 2 façons d'écrire des commentaires :

```
// Ceci est un commentaire sur une ligne
/*    Ceci est un commentaires
      sur plusieurs lignes */
```