## Instructions et blocs d'instructions

Les scripts JavaScript trouvent leur place au sein de pages HTML. Ainsi, en html5, le code JavaScript doit être encadré par une simple balise `<script>`.

Dans le langage JavaScript, les blocs d'instructions sont délimités par des accolades : `{` est l'équivalent d'un *Début*, `}` d'une *Fin*.

Chaque instruction est impérativement suivie d'un point-virgule (`;`).

## Types et variables

JavaScript autorise la manipulation de types classiques : booléens, entiers, réels, caractères et chaîne de caractères et tableaux. Cela dit les types ne sont pas déclarés, uniquement les variables.

```
var x;              // simple déclaration/
var pi     = 3.14; /* déclaration
                      et affectation d'une valeur */
var prenom = 'nicolas';
```

Pour écrire des commentaires dans le code JavaScript, deux syntaxes sont possibles :

- `//`est réservé à un commentaire qui s'étend jusqu'à la fin de la ligne mais pas au-delà.
- `/*` et `*/` permettent d'encadrer un commentaire, éventuellement sur plusieurs lignes.

La déclaration d'une variable se fait à l'aide des mots-clefs `var` ou `let`, la différence étant liée à la *portée de la variable*, c'est-à-dire sur la portion du script dans laquelle la variable est connue. Avec `var` la variable est connue dans toute la fonction dans laquelle elle est définie et pas au-delà. Avec `let` la portée de la variable se restreint au bloc dans laquelle la variable est définie, c'est-à-dire au jeu d'accolades ouvrante/fermante le plus proche contenant la déclaration. Une variable définie avec `var` en dehors de toute fonction est connue dans l'ensemble du script, c'est une variable globale.

Le symbole **=** permet l'affectation d'une valeur à une variable. Un test d'égalité, qui lui ne modifie en rien les variables, sera exprimé par le symbole == (qui ne test pas le type des variables). Le symbole === test l'égalité et le type des variables.

## Opérateurs

Les opérateurs arithmétiques classiques : **+**, **-**, *****, /, **%** pour l'opération *modulo*, et l'opérateur de concaténation entre chaînes de caractères noté par un *plus* (`+`, comme pour l'addition entre entiers).

```
var i = 10;
i = i + 1;
i = i + 9;
var texte = 'La variable i vaut '+i;
texte = texte+"\n";
```

Il y a également les habituels opérateurs logiques : le ET (noté `&&`), le OU (noté `||`) et le NON (noté `!`).

Les comparaisons comme `<` peuvent porter sur des nombres mais aussi sur des chaînes de caractères (il s'agit alors d'une comparaison alphabétique).

## Instruction de sortie et chaînes de caractères

L'affichage est réalisé à l'aide des méthodes `write` et `writeln` de l'objet `document` : il s'agit d'écrire du texte ou des balises au sein du code HTML, à l'endroit où est placé le script.

Le langage JavaScript permet de délimiter les chaînes de caractères soit par des apostrophes, soit par des guillemets. Il est pratique d'utiliser les guillemets lorsque le texte contient une apostrophe, et vice-versa. Il est aussi possible d'utiliser un *backslash* (signe \) pour neutraliser la signification JavaScript d'un caractère.

```
var prenom = 'Toto';
document.write("salut ");
document.writeln(prenom);
document.writeln("<p>ça va aujourd'hui ?</p>");
document.writeln('<p>oui, ça va aujourd\'hui !</p>');
```

La différence entre `document.write` et `document.writeln` : la seconde instruction passe après à la ligne après affichage, pas la première. Cependant ces passages à la ligne ne sont pas pris en compte par les navigateurs, lesquels ne s'intéressent finalement qu'aux éléments html et à leurs styles css. Un moyen de visualiser la différence est de produire des affichages au sein d'un environnement `pre` qui respecte les passages à la ligne.

```
<pre>
<script>
document.write("salut ");
document.writeln("Toto");
document.writeln("ça va aujourd'hui ?");
document.writeln('oui, ça va aujourd\'hui !');
</script>
</pre>
```

L'exemple précédent va afficher *salut* et *Toto* sur la même ligne :

```
salut Toto
ça va aujourd'hui ?
oui, ça va aujourd'hui !
```

## Fonctions mathématiques

JavaScript met à disposition des fonctions mathématiques à travers l'objet `Math` dont les principales sont :

```
// les arrondis : en dessous, au plus proche, au dessus
Math.floor(...)
Math.round(...)
Math.ceil(...)

// de l'aléatoire : un nombre entre 0 et 1
Math.random()

// de la géométrie : la valeur de Pi
Math.PI
```

`Math.floor` et `Math.random` permettent de choisir au hasard une case dans un tableau. `Math.PI` permet de tracer un cercle dans le *canvas*.

## Structures de contrôle en JavaScript

La boucle *si alors sinon* :

```
if (une condition ici) {
  // des instructions ici
} else {
  // des instructions ici
}
```

La boucle *tant que* :

```
while (une condition ici) {
  // des instructions ici
}
```

ce qui donne sur un exemple comptant de un à dix :

```
var i=0;                  # initialisation d'une variable i
while (i<10) {            # tant que et sa condition
    document.writeln(i); # affichage
    i = i+1;              # incrémentation de la variable i
}
```

La boucle *pour*, trois éléments doivent être fournis : une variable avec son initialisation, une condition de type *tant que* et une instruction à effectuer à chaque passage dans la boucle (typiquement une mise à jour de la variable associée à la boucle).

```
for ( initialisation ; condition pour continuer ; progression) {
    # des instructions ici
}
```

Exemple :

```
for (var i=0 ; i<10 ; i++) {
    document.writeln(i); # affichage
}
```

## Fonctions et procédures en JavaScript

Les procédures et les fonctions sont semblables sur bien des points. Elles permettent d'isoler un bout de code, de le nommer et ainsi de le rendre réutilisable à différents moments. Elles peuvent utiliser des paramètres et contenir des instructions quelconques.

En revanche, une fonction doit toujours se terminer par le renvoi d'un résultat (à l'aide du mot-clef `return` en JavaScript), ce qui n'est jamais le cas pour une procédure. Cela signifie en particulier qu'un appel à une fonction

se trouvera souvent dans la partie droite d'une affectation, ce qui ne sera jamais le cas pour un appel de procédure.

## Définition et utilisation d'une procédure

```
// forme générale d'une procédure
function nom_de_la_procédure (paramètres) {
  // des instructions ici
}
# définition d'une procédure
function est_voyelle_procédure_tantque (lettre) {

    var voyelles = ['a','e','i','o','u','y'];

    var i = 0;
    while ((i<voyelles.length) && (lettre != voyelles[i])) {
        i = i+1;
    }

    if (i<voyelles.length) {

        document.writeln('« ',lettre,' » est une voyelle.');

    } else {

        document.writeln('« ',lettre,' » N\'est PAS une voyelle.');

    }
}

# appels à la procédure
est_voyelle_procédure_tantque('a');
est_voyelle_procédure_tantque('b');
```

## Définition et utilisation d'une fonction

```
// forme générale d'une fonction
function nom_de_la_fonction (paramètres) {
  // des instructions ici
  return(résultat)
}
# définition d'une fonction
function est_voyelle_fonction_tantque (lettre) {

    var voyelles = ['a','e','i','o','u','y'];

    var i = 0;
    while ((i<voyelles.length) && (lettre != voyelles[i])) {
        i = i+1;
    }

    return (i<voyelles.length);

}

# appel à la fonction dans la condition d'un if
for (lettre of ['a','b']) {
    if (est_voyelle_fonction_tantque(lettre)) {
        document.writeln('« ',lettre,' » est une voyelle.');
    } else {
        document.writeln('« ',lettre,' » N\'est PAS une voyelle.');
    }
}
```

# Gestion des événements et des actions en JavaScript

Le couple HTML + CSS amène à des documents essentiellement statiques. Certains calculs peuvent être automatisés avec `document.writeln`. Regardons comment produire des documents de manière dynamique :

- côté serveur : il est possible d'utiliser un langage comme PHP associé à une base de données SQL;
- côté client : ce sont les actions de l'utilisateur, puis la modification du document en fonction de ces actions et à l'aide de JavaScript, qui produisent une page web dynamique.

## Événements que l'on peut capturer avec JavaScript

Voici les principaux événements qu'il est possible de traiter.

- `onload` : événements déclenchés à l'arrivée sur la page,
- `onunload` : événements déclenchés au départ de la page,
- `onclick`, `onmousedown`, `onmouseup`, `onmousemove`, `onmouseover`, `onmouseout` : événements associés aux clics et déplacements de la souris,
- `onkeypress`, `onkeydown`, `onkeyup` : événements provoqués par l'appui d'une touche au clavier,
- `onsubmit`, `onchange` : événements associés à la manipulation d'un formulaire par l'utilisateur.

Ces événements sont également des noms d'attributs HTML, le contenu de ces attributs sera du JavaScript. De manière générale, JavaScript permet d'associer une instruction JavaScript à chaque événement : soit l'appel à une fonction de base JavaScript, soit l'appel à l'une de nos propres procédures.

La boîte de dialogue **prompt**() permet d'afficher un message, un champ à remplir (input de type texte) et un bouton "OK". Cette fonction retourne la valeur qui a été entré dans le champ par l'utilisateur.
La boîte de dialogue **confirm()** permet d'afficher un message et deux boutons : un bouton "OK" et un bouton "Annuler". Cette fonction retourne une valeur booléenne qui vaut "true" si c'est le bouton OK qui est cliqué et retourne "false" si c'est le bouton "Annuler" qui est cliqué.

## Fonctions JavaScript associables à une action de l'utilisateur

Nous venons de capturer un événement, il reste à lui associer une action, c'est-à-dire un code JavaScript à exécuter. Nous pourrons en particulier jouer sur les actions JavaScript suivantes :

- `alert` ouvre une fenêtre avec un message donné,
- modifier `window.location` pour aller vers une autre page,
- `document.write` écrit un texte donné,
- `setTimeout` et `setInterval` permettent respectivement de différer et de répéter une instruction ;
- `focus` permet d'activer un élément de la page (comme la case à remplir obligatoirement dans un formulaire), `blur` de le désactiver.

Exemples :

```
// déclenchement d'une alerte immédiate (alert)
<button onclick="alert('Coucou !')">cliquez ici !<button>

// déclenchement d'une alerte dans une demi-seconde (setTimeout)
<button onclick="setTimeout('alert(\'Surprise !\')',500)">
cliquez ici !<button>

// répétition toutes les 5 secondes (setInterval)
var clock_id = setInterval('alert(\'hi hi !\');',5000);
...
clearInterval(clock_id); // fin de la répétition
```

## Informations sur un événement capturé

Dans le cas particulier de la frappe d'une touche au clavier, l'événement peut être capturé et associé à l'une de nos fonctions `gere_frappe` comme suit :

```
<body onkeyup="gere_frappe(event);">
```

`event` est une variable qui décrit l'événement qui s'est produit, en particulier dans le cas d'une touche pressée au clavier, elle contient l'information nous permettant de connaître cette touche.

Dans le cas d'un caractère imprimable, il suffit d'utiliser `event.key` nous donne le caractère lui-même. Dans les autres cas, `event.key` fournit une chaîne de caractères correspondant à la touche pressée :

- `Delete` et `Backspace` pour les suppressions,
- `Enter` pour la touche entrée,
- `ArrowLeft`, `ArrowUp`, `ArrowRight`, `ArrowDown` pour les flèches,
- etc.

Enfin, dans le cas d'un clic à la souris dans la page, l'objet `event` nous fournit les coordonnées du pixel cliqué à travers `event.pageX` et `event.pageY`.