JavaScript Notions de base

1. Généralités

Le langage Javascript est un langage interprété par les navigateurs. Il est donc un des langages de programmation les plus utilisés

Dans le langage JavaScript, chaque instruction se termine par un point-virgule (;) mais ce dernier n'est pas obligatoire.

Le type d'une variable est déterminé par le contexte d'utilisation : Les variables ne sont pas typées mais il est recommandé de les définir afin d'en préciser la portée (let ou var).

Le langage fait la distinction entre les majuscules et les minuscules (sensible à la casse).

Les instructions et fonctions JavaScript sont de préférence contenues dans un fichier externe d'extension '.js' chargé à partir d'un script HTML (ou PHP) par la balise suivante :

```
<script src="monscript.js"></script>
```

L'attribut src désigne alors le chemin, relatif ou absolu, vers le fichier contenant le script.

Il est cependant possible de placer directement des instructions JavaScript dans une page html :

```
<script>
    /* Votre script ici */
</script>
```

Cette technique est utilisée principalement lorsque la génération de code JavaScript s'effectue côté serveur afin de permettre de transmettre des valeurs du serveur (PHP) vers le client (Javascript).

```
<script>
    let idSession = <?= $idSession ?>
</script>
```

Il faut savoir que :

- Il n'y a pas d'installation, de désinstallation, ni de mise à jour possibles de JavaScript. JavaScript est une technologie embarquée, nativement, dans tous les navigateurs. Cette technologie fait partie du code des navigateurs, les mises à jour de JavaScript se font lors de la mise à jour du navigateur.
- Depuis la version 1.6 (ECMAScript 6) sortie en 2015, une nouvelle édition est publié chaque année.
- Le Javascript peut être désactivé par le client.
- Une erreur dans le code JavaScript n'est pas bloquante, le reste du code est tout simplement ignoré par le navigateur.

2. Les données

En JavaScript les variables ne sont **pas typées** (c'est le langage qui fixe le type de la variable en fonction de son contenu) et leur **déclaration n'est pas obligatoire mais fortement recommandée**.

Tous les types primitifs sont gérés : string (chaîne), number (entier ou réel sans distinction), boolean (booléen). Ces types primitifs sont en réalité des classes d'objets sur lesquels on peut exécuter des méthodes.
Le type undefined est associé au variables qui n'existent pas.
Si une variable contient une chaîne, on pourra lui appliquer l'ensemble des méthodes et propriétés sur une chaîne. Une chaîne peut être délimitée par des guillemets ou des apostrophes.

```
let uneChaine = "bonjour";
uneChaine.length  retourne la longueur de la chaîne
uneChaine.charAt(2)  retourne le troisième caractère de la chaîne.
```

Sur un nombre

```
let n = 2.758
n.toPrecision(2)  retournera 2.7 (deux chiffres affichés)
n.toFixed(2) retournera 2.75 (deux chiffres après la virgule)
```

La déclaration d'une constante s'effectue à l'aide du mot clé const

```
const x = 20
```

2.1. Portée (visibilité) des variables

Selon l'endroit où une variable est déclarée, celle-ci pourra être accessible (visible) de partout dans le script ou bien uniquement dans une portion confinée du code, on parle de « portée » d'une variable. Lorsqu'une variable est déclarée **sans le mot clé let ou var**, c'est-à-dire de façon implicite elle est de **portée globale** (accessible dans tout le script)

La portée d'une variable déclarée de façon explicite (précédée du mot-clé **let** ou **var**), dépend de l'endroit où elle est déclarée :

- Une variable déclarée par let ou var au début du script, avant toute fonction, sera globale.
- Une variable déclarée par let ou var dans une fonction aura une portée locale (limitée à cette fonction)
- Une variable déclarée par let dans un bloc d'instruction (ensemble des instructions comprises entre deux accolades) ne sera utilisable que dans ce bloc

L'instruction let permet une définition plus fine de la portée d'une variable. Elle empêche aussi la redéfinition dans la même portée et l'utilisation d'une variable avant sa déclaration (ce qui est possible avec var). Elle tend donc à remplacer l'instruction var.

Si l'on souhaite savoir si une variable est déclarée : **if (typeof(myVar) != 'undefined') {**

Règle de nommage

Bosses de chameau (majuscule au début de chaque mot à partir du 2ème) pour les variables, propriétés et les méthodes et minuscules pour les événements : onload, onchange, etc.

3. La saisie

La saisie sera en générale réalise en HTML à l'aide d'une balise input.

```
<input id='x' value='0' type="text">
```

Le contenu peut être récupéré en javascript par la commande **x.value** si le script js n'utilise pas de variable x qui viendrait masquer ce champs
Si c'est le cas la récupération s'effectue par la commande **document.getElementById('x').value**

Il est cependant possible de saisir une donnée à l'aide d'une boîte de dialogue piloté par le langage :

```
s = prompt('commentaire', [valeur]);
```

Affiche une boite de dialogue avec le texte 'commentaire' et une zone de saisie initialisée à valeur éventuellement. La fonction retourne la valeur saisie (chaine de caractère). La valeur retournée sera de type texte

Lorsqu'il s'agit de poser une question de type 'oui/non', on peut utiliser la méthode confirm de l'objet document.

```
if (document.confirm('Confirmation de votre demande')) {…}
```

Cette méthode affiche 2 boutons "OK" et "Annuler".
Renvoie true si on clique sur le bouton OK, et false pour le bouton Annuler.

3.1. Conversion

La valeur récupéré est toujours de type texte afin de pouvoir accepter toute valeur saisie. Il est souvent nécessaire de la convertir dans le type souhaité (entier, réel par exemple) pour réaliser des opérations.

Deux fonctions permettent respectivement de convertir la valeur en entier ou en réel

```
let x = parseInt(s);
let y = parseFloat(s);
```

Ces fonctions retournent la valeur NaN si s n'est pas un nombre.
La valeur NaN peut être tester avec la fonction isNaN().

```
let age;
let s = prompt('Quel est votre âge ?');
if (s.length > 0) {
        const n = parseInt(s);
        if (!isNaN(n)) {
            age = n;
        }
}
```

Pour convertir la chaine saisie en entier ou réel il est aussi possible d'utiliser le constructeur Number(s) qui retourne 0 si s n'est pas numérique

```
age = Number(s);
```

4. L'affichage

L'affichage s'effectue là encore par une interaction dans le code HTML, mais il est aussi possible d'afficher des résultat dans une boite de dialogue ou dans le fenêtre console du navigateur

La fonction alert('') permet d'afficher un résultat dans une boîte de dialogue :

```
alert('message')
```

La méthode log() de l'objet console permet d'afficher un message dans la fenêtre console du navigateur. Cela est surtout utile pour le programmeur lorsqu'il veut vérifier le fonctionnement du script.

```
console.log('message')
```

S'il s'agit d'un message d'erreur

```
console.error('message')
```

Il est possible d'écrire dans la page HTML associée mais cela va remplacer son contenu

```
document.write('contenu html')
```

Si l'on veut obtenir un affichage en pourcentage :

```
taux = new Intl.NumberFormat("fr-FR", {style: "percent"}).format(taux);
```

Si on souhaite un affichage dans le format monétaire :

```
montant = new Intl.NumberFormat("fr-FR", {style: "currency", currency: "EUR"}).format(montant);
```

Par défaut, si l'on place une variable à l'intérieur d'une chaine, cette dernière ne sera pas interprétée.

```
let n = 10
console.log('n vaut n');  // affiche n vaut n
```

Il est possible de demander au langage JavaScript d'interpréter les variables placées à l'intérieur d'une chaîne de caractères à l'aide de l'opérateur ${}.
La chaîne doit dans ce cas être délimitée par des apostrophes inversées (AltGr 7) : `

```
let n = 10
console.log(`n vaut ${n}`);  // affiche n vaut 10
```

L'opérateur $ permet d'interpréter aussi une expression

```
console.log (`le candidat est ${n>= 10 ? 'admis', 'éliminé'}`);  // affiche le candidat est admis
```

```
let n = 10
let m = 12
console.log (`le candidat est ${n >= m ? 'admis' : 'éliminé'}`);
```

5. Les opérateurs

Tous les opérateurs mathématiques, les opérateurs logiques et les opérateurs de comparaisons sont utilisables. Dans un langage il faut distinguer l'opérateur d'affectation '=' permettant d'initialiser une variable de l'opérateur d'égalité ('==' ou '===') permettant de comparer deux valeurs.

Affectation ←	a = b
Addition	a + b
Soustraction	a – b
Multiplication	a * b
Division	a / b
Exponentiation	a ** b
Modulo	a % b

Égalité	a == b
Différent	a != b
Supérieur	a > b
Inférieur	a < b
Supérieur ou égal	a >= b
Inférieur ou égal	a <= b
Égalité strict	a === b

Non logique	!
Et logique	&&
Ou logique	Or | |
Ou exclusif	^
Incrémentation	n++
Décrémentation	n--
Concaténation	a + b

Dans un langage non typé, c'est le langage qui déterminer le type de la variable au moment de son initialisation. l'égalité strict permet de comparer à la fois la valeur et le type des données. Dans ce cas deux valeurs seront strictement identique si elle ont la même valeur et le même type :
De ce fait 2 === 2 mais 2 n'est pas strictement identique à '2' : 2 !== '2'

L'incrémentation (+1) peut se faire en notation postfixée (après) n++ ou préfixé (avant) ++n
de même pour l'opération de décrémentation n—ou --n

Exemple :

```
let n = 2
console.log(n++)  // affiche 2 puis n passe à 3
console.log(++n)  // n passe à 4 puis sa valeur est affichée : 4
console.log(n--) // affiche 4 puis n passe à 3
console.log(--n) // n passe à 2 puis sa valeur est affichée : 2
```

L'opérateur + est surchargé pour pouvoir s'appliquer sur des chaînes de caractères. Dans ce cas il réalise une concaténation des 2 chaînes

```
let a = "2"
let b = "3"
console.log( a + b) // affiche 23
let c = 2
let d = 3
console.log(c + d) // affiche 5
```

L'opérateur ternaire permet de renvoyer une valeur de façon conditionnelle

```
let c = a == b ? 0 : 1;
console.log(c);
```

si a et b ont la même valeur c prend la valeur 0 sinon il prend la valeur 1

6. Les structures de contrôles

Le Si Alors Sinon

```
if (a == b) {
    c = 0
} else if  (a > b)  {
    c = a - b;
  d =  d - c;
} else {
   c = b - a;
}
```

Les clauses else if et else sont facultatives
Les accolades ne sont pas utile si elles ne comportent qu'une seule instruction

Le selon

```
switch (choix)   {
    case 'a' : action1; break;
    case 'b' : action2 ; break;
     ….
   default :
      action n;
  };
```

La clause default est vivement recommandé pour des raisons de sécurité

La structure itérative Pour : for

```
for ( let i = 1;  i <= 10;  i++) {
     console.log(`5 * ${i} = ${5 * i}`);
}
```

```
let lesJours = ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi', 'Dimanche'];
for ( let jour of lesJours) {
     console.log(jour);
}
```

of : permet de parcourir les valeurs du tableau

```
let lesJours = ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi', 'Dimanche'];
for ( let i in lesJours) {
     console.log(lesJours[i]);
}
```

in permet de parcourir les indices du tableau qui commence à 0 en Javascript

Remarque : dans la console, il est possible d'afficher le contenu d'un tableau à l'aide de la méthode table

```
console.table(lesJours);
```

La structure itérative Tant que : while

```
let x = 0;
while (x < 10) {
   console.log(++x);
}
console.log(`à la sortie de la boucle x vaut ${x}`);  //  10
```

La structure itérative Répéter tant que : do while

```
let n = Math.ceil(Math.random() * 100);
console.log(n);
do {
    n--;
} while (n % 5 !== 0);
console.log(n);
```

Il est possible d'interrompre une structure itérative avant sa fin prévue en utilisant l'instruction **break**

Elle est souvent utilisée pour arrêter une recherche qui s'effectue en parcourant tout le tableau.
Si on trouve l'élément cherché, il n'est plus utile de continuer à parcourir le reste du tableau.

```
n =t.length;
for (i = 0; i < n ; i++) {
  if (t[i] == valeurCherchee) {
    break;
  }
}
```

Si plusieurs structures répétitives sont imbriquées, l'instruction break ne met fin qu'à la structure itérative en cours.

Si l'on souhaite arrêter toutes les structures itératives imbriquées, il faut placer devant la première structure itérative un label suivi de : (Traitement : for(…) par exemple) et utiliser l'instruction break label

```
Recherche: for (l = 0; l < nbCaracteres; l++) {
   code = lesCaracteres[l] + code.substring(1);
   for (k = 0; k < nbCaracteres; k++) {
      code = code.substring(0,1) + lesCaracteres[k] + code.substring(2);
      for (j = 0; j < nbCaracteres; j++) {
         code =  code.substring(0,2) + lesCaracteres[j] + code.substring(3);
         for (i = 0; i < nbCaracteres; i++) {
            code = code.substring(0,3) + lesCaracteres[i];
            if (code == codeSecret) {
              break Recherche;
            }
         }
      }
   }
}
```

L'instruction continue permet de mettre fin à l'itération en cours pour passer à l'itération suivante. Autrement dit, les instructions placées entre continue et la fin de la boucle ne sont pas exécutées.

```
var n = 0
for (i = 1; i <= 5; i++)  {
     if (i ==  3 ) { continue; }
     n = n + i
}
```

Dans cet exemple la valeur 3 ne sera pas ajoutée à n. n = 1 + 2 + 4 + 5 = 12

Si l'on souhaite reprendre à un autre endroit, on peut recourir à l'instruction continue label

```
let n = 0;
for (const x of lesXs) {
      if (x.saison === saison.value
        && x.annee === annee.value
        && x.distance !== distance.value) {
          n = 1;
          ….
          break;
      }
  }
  if (n === 0 ) {
     alert("…");
  }
```

```
let n = 0;
for (const x of lesXs) {
      if (x.saison !== saison.value) continue;
      if (x.annee !== annee.value ) continue;
      if (x.distance !== distance.value) continue;
      n = 1;
      …
      break;
  }
  if (n === 0 ) {
     alert("…");
  }
```