TP n°3 : Introduction à JavaScript

1 Variables et constantes

1.1 Types de variables

On retrouve en JavaScript les mêmes types de variables élémentaires que dans les autres langages de programmation, à savoir des :

— entiers ;
— flottants ;
— chaînes de caractères ;
— booléens ;
— listes ;
— etc.

Toutefois, on remarquera des différences avec le Python, notamment pour les entiers et les flottants qui ont le même type en JavaScript ; ou les listes qui sont des objets.

Le tableau ci-dessous donne des correspondances de types de données en JavaScript.

Type de donnée	Booléen	Entier	Flottant	Chaîne de caractères	Listes
JavaScript	`boolean`	`number`	`number`	`string`	`object`

Il existe de nombreux autres types comme `null` qui sert pour les valeurs inconnues ou encore `undefined` pour les valeurs non attribuées mais nous ne nous attarderons pas dessus.

1.2 Différences entre variables et constantes

Les constantes, contrairement aux variables, ne sont pas mutables, i.e. que leur valeur ne peut pas être modifiée. Si l'on demande une modification de la valeur d'une constante, on obtient alors une erreur nous indiquant que cela n'est pas possible.

Les constantes peuvent être très pratiques pour conserver des données dont on veut s'assurer qu'elles ne changeront pas au cours de l'exécution du programme. Par exemples : la date de naissance d'une personne, son nom, etc.

1.3 Déclaration d'une constante

Une constante est déclarée par le mot clé `const`. Par exemple

```
const prenom = "Raton" ;
const nom = "Laveur" ;
```

Remarque : on peut voir que la fin des deux instructions ci-dessus est symbolisée par des « ; ». Ceux-ci s'avèrent **indispensables** en JavaScript, sans eux, nous aurions une erreur.

1.4 Déclaration et modification d'une variable

Une variable est déclarée par le mot clé `let`. Une fois déclarée, il n'y a plus besoin de mot clé pour la modifier. Par exemple

```
let age = 1 ;
age = 3 ;
let liste = [age] ;
```

déclare et initialise la valeur de `age` à 1 puis lui réaffecte la valeur 3. Comme en Python, les listes sont symbolisées par des `[]`.

Les modifications raccourcies existent aussi en JavaScript :

Opération	Algorithme	JavaScript	JavaScript raccourci
Addition	$a \leftarrow a + k$	`a = a+k`	`a += k`
Incrément de 1	$a \leftarrow a + 1$	`a = a+1`	`a++`
Soustraction	$a \leftarrow a - k$	`a = a-k`	`a -= k`
Décrément de 1	$a \leftarrow a - 1$	`a = a-1`	`a--`
Multiplication	$a \leftarrow a \times k$	`a = a*k`	`a *= k`
Division	$a \leftarrow a/k$	`a = a/k`	`a /= k`
Puissance	$a \leftarrow a^k$	`a = a**k`	`a **= k`

Exercice 1.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Modifier le fichier JavaScript afin d'y déclarer les variables et constantes suivantes :
   — votre nom (constante) ;
   — votre prénom (constante) ;
   — votre âge (variable) ;
   — une liste (variable) judicieusement nommée « liste » contenant les trois précédentes.
3. Ajouter une ligne permettant d'incrémenter la valeur de votre âge de 1.
4. Observer le code source de la page affichée à l'aide de Firefox.

Exercice 2. [Battlestar Galactica] Battlestar Galactica est une excellente série télévisée des années 2000. Elle est composée de quatre saisons ; la première comporte treize épisodes, les trois autres, vingt chacune ; chaque épisode dure 43 minutes.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Modifier le fichier JavaScript afin d'y déclarer les variables et constantes suivantes :
   — durée de l'épisode ;
   — nombre d'épisode par saison.
3. À l'aide des deux variables ou constantes précédentes, déclarer et calculer les deux quantités suivantes :
   — le nombre total d'épisodes : `nombre_total_episodes` ;
   — le nombre total d'heures et de minutes de visionnage : `nombre_heures_visionnage` et `nombre_minutes_visionnage`.

2 Tests

2.1 Le if, le else if et le else

Les tests (ou instructions conditionnelles) se font en JavaScript à l'aide des commandes `if`, `else if` et `else`. La syntaxe est la suivante :

```
if (condition_1) {
    instructions_1 ;
}
else if (condition_2) {
     instructions_2 ;
}
else {
     instructions_3 ;
}
```

Les conditions à tester sont entre parenthèses et les instructions entre accolades. Bien que les indentations ne soient pas nécessaires, elles aident à la lisibilité du code et il est conseillé de les appliquer.

Remarque : une variable ou une constante déclarée à l'aide de `let` ou `const` dans un bloc de code délimité par des accolades `{}` n'a d'existence qu'au sein de celui-ci, pas l'extérieur. On peut parler de caractère local des variables et constantes.

2.2 Opérateurs de comparaison

Comparaison	Inégalité	Égalité	Différence
Opérateurs JavaScript	<, <=, >, >=	==, ===	!=, !==

En JavaScript, il existe deux opérateurs d'égalité. Le `==` teste une égalité de valeur, ainsi `1=="1"` aura pour valeur `true`. Le `===` teste une égalité de valeur et de type, ainsi `1==="1"` aura pour valeur `false`. Il en va de même pour `!=` et `!==`.

2.3 Opérateurs logiques

Opérateur logique	et	ou	non
Opérateur JavaScript	&&	||	!

```
let booleen1 = true ;
let booleen2 = false ;

let booleen3 = booleen1 && booleen2 ; // aura la valeur false
let booleen4 = booleen1 || booleen2 ; // aura la valeur true
let booleen5 = !booleen1 ; //aura la valeur false
```

Exercice 3. [Vous connaissez la réponse]

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire dans la section modifiable du fichier JavaScript un test qui
   (a) appelle la fonction `bonne_reponse()` si la valeur de la variable `reponse` est égale à celle de la constante `la_reponse` ;
   (b) appelle la fonction `mauvaise_reponse()` sinon.

   Indication : pour appeler une fonction sans argument, il suffit d'utiliser l'instruction `nom_fonction()`.

Exercice 4. [Identifiant et mot de passe]

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire dans la section modifiable du fichier JavaScript un test qui
   (a) appelle la fonction `formulaire_valide()` si tous les champs sont complétés et les mots de passe identiques ;
   (b) appelle la fonction `champ_vide()` si au moins l'un des champs est vide ;
   (c) appelle la fonction `mots_de_passe_differents()` si les mots de passe sont différents.

3 Boucles

3.1 Boucles `for`

3.1.1 Boucle `for`

La boucle `for` en JavaScript a la syntaxe suivante :

```
for (éléments à parcourir) {
      instructions ;
}
```

```
for (let i=0; i<10; i++) {
      console.log(i) ;
}
```

On reconnaît dans le `let i=0` l'initialisation de la valeur de la variable que l'on parcourra ; dans le `i<10` un critère d'arrêt de la boucle ; et dans le `i++` la façon dont la variable évolue à chaque étape de la boucle, ici un incrément de 1. L'instruction `console.log(i)` permet d'afficher la valeur de `i` dans la console ; nous ne l'utiliserons pas cependant, elle est sert juste d'exemple.

Exercice 5. [Je compte jusqu'à 99]

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire dans la section modifiable du fichier JavaScript une boucle appelant la fonction `ajout_paragraphe()` afin que soient affichés les nombres de 0 à 99 sur la page Web. `ajout_paragraphe()` prend comme argument le contenu du paragraphe à afficher.

3.1.2 Boucle `for in`

La boucle `for in` permet un parcours plus intuitif de la boucle sur les indices d'une liste. Par exemples :

```
const liste_valeurs = [4,5,6] ;

for (let i in liste_valeurs){
     console.log(i) ;
}
```

```
const liste_valeurs = [4,5,6] ;

for (let i in liste_valeurs){
     console.log(liste_valeurs[i]) ;
}
```

Dans les deux exemples, `i` parcourt les valeurs des indices possibles de la liste `liste_valeurs`. `i` va donc successivement prendre les valeurs 0, 1 et 2. Ce sont elles qui seront affichées dans l'exemple de gauche. Les valeurs de listes ne seront affichées que si on les appelle à l'aide de l'instruction `liste_valeurs[i]` comme dans l'exemple de droite.

Exercice 6. [Moyenne 2, le retour]

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire dans la section modifiable du fichier JavaScript un script calculant, à l'aide d'une boucle `for in`, la moyenne d'une liste de valeurs judicieusement nommée `liste_valeurs` que vous aurez déclarée au préalable (valeurs au choix). On nommera habilement la moyenne : `moyenne`. *Indication :* la longueur d'une liste peut être obtenue grâce à `length` de la façon suivante : `nom_liste.length`.

3.1.3 Boucle `for of`

La boucle `for of` permet un parcours plus intuitif et direct de la boucle sur les éléments d'une liste. Par exemple :

```
const liste_valeurs = [4,5,6] ;

for (let element of liste_valeurs){
     console.log(element) ;
}
```

Ici, la variable `element` prendra successivement les valeurs contenues dans la liste. Elle vaudra donc, 4 puis 5 et enfin 6.

Exercice 7. [Minimum 2, le retour]

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire dans la section modifiable du fichier JavaScript un script déterminant, à l'aide d'une boucle `for of`, le minimum d'une liste de valeurs judicieusement nommée `liste_valeurs` que vous aurez déclarée au préalable (valeurs au choix). On nommera habilement le minimum : `minimum`.

3.2 Boucle `while`

La boucle `while` en JavaScript a la syntaxe suivante :

```
while (conditions) {
      instructions ;
}
```

Par exemple :

```
let i = 0 ;

while (i<10) {
      console.log(i) ;
      i++ ;
}
```

Exercice 8. [Dichotomie 2, le retour]

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire dans la section modifiable du fichier JavaScript un script trouvant la valeur de la variable `nombre_mystere` à l'aide d'un algorithme de dichotomie. `nombre_mystere` est un nombre entier aléatoirement tiré entre les variables `borne_inf` et `borne_sup`. La recherche s'effectuera à l'aide d'une variable nommée `nombre` que l'on déclarera et initialisera. On stockera toutes valeurs que pendra cette variable dans une liste nommée `liste_valeurs`.

Indications :

— pour ajouter une valeur à la fin d'une liste, on utilisera l'instruction `nom_liste.push(valeur)` ;
— pour obtenir la partie entière d'un nombre, on utilisera la commande `Math.floor(valeur)`.

4 Fonctions

Les fonctions en JavaScript ont la syntaxe suivante :

```
function nom_fonction(paramètres) {
      instructions ;
      return valeurs ;
}
```

On remarquera que comme en Python, le `return` n'est pas obligatoire mais n'intervient que si l'on souhaite renvoyer une ou plusieurs valeurs en sortie.

Exemples :

```
function fonction_somme(n) {
      let somme = 0 ;
      for (let i=1; i<=n; i++) {
           somme += i ;
      }
      return somme ;
}
```

```
function fonction_produit(n) {
      let produit = 1 ;
      for (let i=2; i<=n; i++) {
           produit *= i ;
      }
      return produit ;
}
```

Ces deux fonctions calculent respectivement la somme et le produit des n premiers entiers naturels :

$$1+2+\cdots+n \quad \text{et} \quad 1\times 2\times\cdots\times n.$$

Exercice 9. Reprendre les fichiers de l'exercice 6 et coder dans la section modifiable une fonction calculant la moyenne d'une liste de valeurs.

Exercice 10. Reprendre les fichiers de l'exercice 7 et coder dans la section modifiable une fonction déterminant le minimum d'une liste de valeurs.

Exercice 11. Reprendre les fichiers de l'exercice 8 et coder dans la section modifiable un algorithme de dichotomie à l'aide d'une ou plusieurs fonctions.