Cégep de Saint-Hyacinthe
Département d'informatique

Programmation client-serveur

420-2RP-HY

Notes de cours

Le langage JavaScript

Introduction

Enseignante :
Giovana Velarde

Hiver 2022

1 Table des matières

1. INTRODUCTION AU LANGAGE 3

1.1 VARIABLES 3
1.2 TYPES 4
1.3 OPÉRATEURS 7
1.4 CONDITIONS 8
1.4.1 Conditions if… else if… else 8
1.4.2 Structure de contrôle switch 9
1.4.3 Structure conditionnelle ternaire 11
1.5 BOUCLES 11
1.5.1 Boucle for 11
1.5.2 Boucle while 13
1.6 FONCTIONS 14

JavaScript

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives et à ce titre est une partie essentielle des applications web. Avec les technologies HTML et CSS, JavaScript est parfois considéré comme l'une des technologies cœur du World Wide Web. Une grande majorité des sites web l'utilisent, et la majorité des navigateurs web disposent d'un moteur JavaScript dédié pour l'interpréter.[1] Il a été créé en 1995.

JavaScript (qui est souvent abrégé en « JS ») est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web (coté client). Mais il est aussi utilisé dans de nombreux environnements extérieurs aux navigateurs web tels que **Node.js**, Apache CouchDB voire Adobe Acrobat. Le code JavaScript est interprété ou compilé à la volée (JIT : *Just in time.* Compilation juste-à-temps ou compilation JAT). C'est un langage à objets utilisant le concept de prototype, disposant d'un typage faible et dynamique qui permet de programmer suivant plusieurs paradigmes de programmation : fonctionnelle, impérative et orientée objet.[2]

1. Introduction au langage

1.1 Variables

Pour déclarer une nouvelle variable en JavaScript, on utilise le mot clef `var`. Les variables doivent être identifiées par un nom unique qui peut être court ou plus descriptif. Les noms de variables peuvent contenir des lettres, des chiffres, des tirets bas et le signe `$`. Ils sont sensibles aux majuscules et minuscules.

Exemple :

```
var auto = "Toyota";
var prix = 2000;
```

Consignes pour nommer les variables :

- Utiliser la convention de nommage *Camel case*[3] avec la première lettre en minuscules.
- Le nom est sensible a la case (majuscules et minuscules). Exemple : si je déclare `premierNom` et `premiernom`, elles sont deux variables différentes.
- Le nom doit être significatif.
- Certains noms ne peuvent pas être utilisés puisque ce sont des mots clefs réservés au langage (comme `for`, `var`, `while`, . . .).
- Le nom ne peut pas commencer par un chiffre.
- Le nom ne peut pas contenir un espace ni des caractères spéciaux (par exemple : un tiré `-`).

[1] https://fr.wikipedia.org/wiki/JavaScript

[2] https://developer.mozilla.org/fr/docs/Web/JavaScript

[3] https://fr.wikipedia.org/wiki/Camel_case

On peut déclarer plusieurs variables en même temps. Mais c'est une bonne pratique de le déclarer dans des lignes séparées.

Exemple :

```
var personne = "Jean Daigneault", auto = "Volvo", prix = 2000;
```

On peut également déclarer des tableaux (un tableau est structure de données utilisée pour représenter un liste de items). Comme les variables n'ont pas de type, il est possible d'avoir des types différents dans un tableau.

Exemple :

```
var tableau = [4, 2, '3'];
console.log(tableau);
```

```
[4, 2, '3']
```

La taille d'un tableau est dynamique. Elle peut changer.

Exemple :

```
var couleurs = ['rouge', 'blue'];
console.log(couleurs);
couleurs[2] = 'vert';
console.log(couleurs);
```

```
['rouge', 'blue']
['rouge', 'blue', 'vert']
```

Comme un tableau est un objet, il a des propriétés.

Exemple :

```
var couleurs = ['rouge', 'blue', 5];
console.log(couleurs.length);
```

```
3
```

Le mot clef `let` permet aussi de déclarer une variable et le mot clef `const` une constante.

1.2 Types

Il y a trois **types primitifs** principaux en JavaScript : `number` (nombre), `string` (chaîne de caractères), `boolean` (valeur logique). Il y a d'autres types primitifs comme : `null` et `undefined`; et de types plus complexes comme : `function` et `object`. Pour connaître le type d'une variable, on peut utiliser la fonction `typeof()`.

Exemple:

```
var x = 4;
var y = 5.2;
console.log(typeof(x));
console.log(typeof(y));
```

```
number
```

```
number
```

Il n'y a pas de différence entre `4` et `5.2`. Ils sont de type `number`. Mais attention à l'arithmetique en virgule flottante! Chaque fois que c'est possible, **utilisez plutôt des calculs entiers** (par exemple : pour le calcul de prix, vous pouvez calculer en centimes, et non pas en dollars).

```
console.log(1.1+0.2);
console.log(0.1+1.2);
console.log(0.1+0.2);
```

```
1.3
1.3
0.30000000000000004
```

Il y a plusieurs façons de déclarer le type `string`. On peut utiliser les guillemets simples ou doubles. On peut aussi utiliser les guillemets simples dans les guillemets doubles et vice-versa, ce qui permettra l'impression de ceux-ci dans le texte, si désiré.

Exemple:

```
var phrase1 = "allo";
var phrase2 = 'allo';
var phrase3 = "allo 'toi'";
var phrase4 = 'allo "toi"';
console.log(phrase1);
console.log(phrase2);
console.log(phrase3);
console.log(phrase4);
```

```
allo
allo
allo 'toi'
allo "toi"
```

Les variables de types booléens ne peuvent prendre que deux valeurs, `true` ou `false`.

```
var utilisateurEstAdmin = false
console.log(typeof(utilisateurEstAdmin));
```

```
boolean
```

Le type `undefined` est attribué à une variable lorsque celle-ci n'a pas de valeurs. On peut également mettre une variable égale à `undefined`.

Exemple :

```
var a;
var b = undefined;
console.log(typeof(a));
console.log(typeof(b));
```

```
undefined
undefined
```

JavaScript est un langage dit à ***types dynamiques*** et à ***typage faible***. Cela signifie que la même variable peut être utilisée pour contenir des types différents. Vous pouvez initialiser une variable en tant que nombre, puis la réaffecter comme chaîne, ou tout autre type de variable. Ceci offre une grande souplesse, mais peut aussi conduire à un comportement inattendu si vous opérez sans précaution.

Exemple :

```
var z;
console.log(typeof(z));
z = 5;
console.log(typeof(z));
z = "Jean";
console.log(typeof(z));
```

```
undefined
number
string
```

Le type `object` est utilisé pour les objets, les tableaux et le `null`.

Exemple :

```
 var x = [1 ,5 ,2];
console.log(typeof(x));
console.log(typeof(null));
```

```
object
object
```

Un objet permet à une variable de contenir plusieurs valeurs. Par exemple, nous pourrions avoir une variable `étudiant` qui contient le nom, le prénom et le programme d'un étudiant. On peut ensuite accéder à chacune de ses propriétés (attributs) en utilisant le nom de la propriété. Pour déclarer un objet, il suffit d'utiliser les `{}` en ajoutant les propriétés désirées à l'intérieur en forme de liste de paires clé/valeur (`clé:valeur`).

Exemple :

```
var etudiant={nom:"Vigouroux",prenom:"Christian",programme:"Informatique"};
console.log(etudiant);
```

```
{ nom: 'Vigouroux', prenom: 'Christian', programme: 'Informatique' }
```

Pour accéder aux propriétés d'un objet on doit utiliser la notation du point `.` ou les crochets `[]`.

Exemple :

```
console.log(etudiant.nom);
console.log(etudiant['prenom']);
console.log(etudiant["prenom"]);
```

```
Vigouroux
Christian
Christian
```

1.3 Opérateurs

Le tableau suivant résume quelques opérateurs en JavaScript

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (le reste de la division)
**	Exponentiation
++	Incrément
--	Décrément
==	Égale en valeur
===	Égale en valeur et type
!=	Différent en valeur
!==	Différent en valeur et type
<	Plus petit
<=	Plus petit ou égal
>	Plus grand
>=	Plus grand ou égal
&&	Logique et
||	Logique ou
!	Logique non

Table 1 - Opérateurs

Il faut faire attention aux opérateurs `==` et `===`. Quand on utilise le comparateur `==`, celui-ci regarde la valeur seulement. Comme le langage n'est pas typé, on peut comparer `5` et `"5"`.

Exemple :

```
console.log(5 == "5");
console.log("5" == 5);
console.log(5 === "5");
console.log("5" === 5);
```

```
true
true
false
false
```

Lorsqu'on utilise l'opérateur `==`, comme celui-ci ne regarde pas le type, donc `5 == "5"` sera vrai. L'opérateur `==` convertit `5` en `"5"` avant de le comparer avec `"5"`. Cependant, `5 === "5"` sera faux puisque l'opérateur `===` vérifie également le type.

L'utilisation des opérateurs peut également se faire entre types. Il faut cependant faire attention à ce qu'on s'attend à recevoir.

Exemple :

```
console.log(5+10+"allo");
console.log("allo"+3+2);
```

```
15allo
allo32
```

L'opération d'addition entre un string et un nombre converti le nombre en string et concatène les deux valeurs (strings). Mais, l'opération est exécutée séquentiellement de gauche à droite. Donc, dans l'exemple précédent : `5+10+"allo"` calcule `5+10=15` en premier et en suite calcule `15+"allo"="15allo"`. `"allo"+3+2` calcule `"allo"+3="allo3"` en premier et ensuite calcule `"allo3"+2="allo32"`.

1.4 Conditions

Il y a trois façons de faire les conditions en JavaScript :

- Avec les structures de contrôle conditionnelles `if`, `if… else` et `if… else if… else`.
- Avec la structure `switch`.
- Avec la structure conditionnelle ternaire basé sur l'opérateur ternaire `:?`.

1.4.1 Conditions if… else if… else

On peut utiliser les structures de contrôle conditionnelles `if`, `if… else` et `if… else if… else` :

1. Le code dans les `{}` de l'`if` sera exécuté seulement si la condition après le `if` est vraie.
2. Sinon, si condition n'est pas vraie, on regarde la condition après le *`else if`*. Le code dans les `{}` du *`else if`* sera exécuté seulement si la condition après le *`else if`* est vraie. Sinon, si condition n'est pas vraie, on regarde la condition après le `else if` suivant. On peut avoir autant de *`else if`* qu'on veut. Lorsqu'un des *`else if`* est vrai, on ne vérifie pas les suivants.
3. Si tous les *`else if`* sont faux, on exécute la portion de code dans le `else`.

Exemple :

```
age = 25;
if(age < 26){
    console.log("Jeune!");
}
else if (age < 35){
    console.log("Encore un peu jeune!");
}
else{
    console.log ("Pas si jeune!");
}
```

```
Jeune!
```

Dans l'exemple précèdent, la console imprime `Jeune!` puisque `25 < 26`. Le code dans le `else if` et le `else` ne sont pas exécutés puisque le premier `if` est vrai.

Exemple :

```
age = 42;
if(age < 26){
    console.log("Jeune!");
}
else if (age < 35){
    console.log("Encore un peu jeune!");
}
else{
    console.log ("Pas si jeune!");
}
```

```
Pas si jeune!
```

Dans l'exemple précèdent, la console imprime `Pas si jeune!`. Comme `42 > 26`, on n'exécute pas le code dans le premier `if`. Comme `42 > 35`, on n'exécute pas le code dans le `else if`. On exécute donc le code dans le `else`.

1.4.2 Structure de contrôle switch

Une deuxième façon de faire des conditions est d'utiliser l'instruction `switch` qui va nous permettre d'exécuter un code en fonction de la valeur d'une variable. Dans la parenthèse, on met la variable qu'on veut évaluer. La commande `switch` regardera chacun des cas possibles pour la valeur de cette variable. Vous devez définir les cas possibles à l'aide de la commande `case`. Si un cas est vrai, c'est-à-dire, si la variable vaut la valeur dans le `case`, le code après sera exécuté. La commande `break` permet d'arrêter une fonction `switch`. Si la commande `break` n'est pas mise, la fonction `switch` regardera les autres `case`.

Exemple :

```
jour_semaine = 2;
switch (jour_semaine) {
    case 0:
      jour_nom = "Lundi";
      break;
    case 1:
      jour_nom = "Mardi";
      break;
    case 2:
      jour_nom = "Mercredi";
      break;
    case 3:
      jour_nom = "Jeudi";
      break;
    case 4:
      jour_nom = "Vendredi";
      break;
    case 5:
      jour_nom = "Samedi";
      break;
    case 6:
      jour_nom = "Dimanche";
}
```

```
console.log(jour_nom);
```

```
Mercredi
```

Dans l'exemple précèdent, on regarde la valeur de la variable `jour_semaine`. La valeur n'est pas `0`, donc on ne va pas dans le premier `case`. La valeur n'est pas `1`, donc on ne va pas dans le deuxième `case`. Comme la valeur est `2`, `jour_nom = "Mercredi"` et on ne regarde pas les autres cas en raison de l'instruction `break`.

Exemple :

```
jour_semaine = 2;
switch (jour_semaine) {
    case 0:
      jour_nom = "Lundi";
      break;
    case 1:
      jour_nom = "Mardi";
      break;
    case 2:
      jour_nom = "Mercredi";
    case 2:
      jour_nom = "Mercredi2";
      break;
    case 3:
      jour_nom = "Jeudi";
      break;
    case 4:
      jour_nom = "Vendredi";
      break;
    case 5:
      jour_nom = "Samedi";
      break;
    case 6:
      jour_nom = "Dimanche";
}
console.log(jour_nom);
```

```
Mercredi2
```

Dans l'exemple précèdent, on regarde la valeur de la variable `jour_semaine`. La valeur n'est pas `0`, donc on ne va pas dans le premier `case`. La valeur n'est pas `1`, donc on ne va pas dans le deuxième `case`. Comme la valeur est `2`, `jour_nom = "Mercredi"`. Cependant, comme il n'y a pas de `break`, on regarde les autres cas. Ainsi, comme la valeur est `2`, on rentre dans le troisième `case`. On a donc `jour_nom = "Mercredi2"` et on ne regarde pas les autres cas parce qu'il y a l'instruction `break`.

Exemple :

```
jour_semaine = 2;
switch (jour_semaine) {
    case 5:
```

```
      jour_nom = "Samedi";
      break;
    case 6:
      jour_nom = "Dimanche";
      break;
    default:
      jour_nom = "Pas la fin de semaine";
}
console.log(jour_nom);
```

```
Pas la fin de semaine
```

Dans l'exemple précèdent, on regarde la valeur de la variable `jour_semaine`. La valeur n'est pas `5`, donc on ne va pas dans le premier `case`. La valeur n'est pas `6`, donc on ne va pas dans le deuxième `case`. Comme aucun des cas n'est vrai, on rentre dans l'instruction `default`. On rentre dans l'instruction `default` lorsque tous les autres `case` sont faux. Il faut faire attention, si on ne met pas de `break` dans nos `case`s, on ira toujours dans l'instruction `default`.

1.4.3 Structure conditionnelle ternaire

Une troisième façon de faire des conditions est d'utiliser utiliser une syntaxe très condensée basée sur l'opérateur ternaire `? :` qui est un opérateur de comparaison. Une structure ternaire se présente sous la forme : `test ? code à exécuter si true : code à exécuter si false`.

Exemple :

```
var x = 15;
console.log(x >= 10 ? "x supérieur ou égale à 10" : "x inférieur à 10");
var message = x >= 10 ? "x supérieur ou égale à 10" : "x inférieur à 10";
console.log(message);
```

```
x supérieur ou égale à 10
x supérieur ou égale à 10
```

1.5 Boucles

Les boucles exécutent le même code plusieurs fois. Il y a deux types de boucles en JavaScript, la boucle `for` et la boucle `while`.

1.5.1 Boucle for

La boucle for classique se compose de trois items :

1. Le premier item s'exécute avant le début de la boucle.
2. Le deuxième item est la condition d'exécution du bloc
3. Le troisième item s'exécute à la fin de chaque itération.

Dans la boucle, chacun des items est séparé pas un point-virgule.

Exemple:

```
var a = 0;
for(i = 0 ; i < 5 ; i++){
    a += 1;
}
console.log(a);
```

```
5
```

Dans l'exemple précèdent, on définit la variable `i = 0` avant de commencer l'exécution de la boucle. Le `a += 1` s'exécute tant que `i < 5`. À la fin de chaque itération de la boucle, `i` est incrémenté de `1`.

Si la variable que vous incrémentez est définie avant l'exécution de la boucle, vous pouvez omettre le premier item.

Exemple :

```
var a = 0;
var i = 2;
for(; i < 5 ; i++){
    a += 1;
}
console.log(a);
```

```
3
```

Si la variable que vous incrémentez est incrémentée dans la boucle, vous pouvez omettre le troisième item.

Exemple :

```
var a = 0;
var i = 2;
for(; i < 10 ;){
    a += 1;
    i += 1;
}
console.log(a);
```

```
8
```

Il est aussi possible d'utiliser une boucle `for` avec le **mot clef `of`** pour parcourir une **liste** ou un **string**.

Exemples :

```
tableau = ["chat", 1, 3];
for (item of tableau){
    console.log(item);
}
```

```
chat
1
3
```

```
mot = "chat";
for (car of mot){
    console.log(car);
}
```

```
c
h
a
t
```

Pour itérer sur les **attributs d'un `object`**, il est possible d'utiliser la boucle `for` avec **le mot clef `in`** tel que présenté dans l'exemple suivant.

```
var etudiant = {nom:"Vigouroux",prenom:"Christian",programme:"Informatique"};
for (attr in etudiant) {
    console.log(attr);
}
for (attr in etudiant) {
    console.log(etudiant[attr]);
}
```

```
nom
prenom
programme
Vigouroux
Christian
Informatique
```

1.5.2 Boucle while

Une autre façon d'exécuter plusieurs fois le même code est d'utiliser la boucle `while`. La boucle `while` exécutera du code tant qu'une certaine condition est vraie. À chaque itération, elle vérifie la condition. Si la condition est fausse, elle termine et passe à la prochaine ligne.

Exemple :

```
var a = 0;
var i = 2;
while(i < 5){
    a += 1;
    i += 2;
}
console.log(a);
```

```
2
```

La boucle `while` peut également s'utiliser avec l'instruction `do` tel que présenté dans l'exemple suivant. Ce type de boucle permet d'exécuter une première fois le code avant de vérifier si la condition est vraie. Par la suite, si la condition est vraie, l'instruction `do` recommence jusqu'à ce que la condition soit fausse.

Exemple :

```
var a = 0;
var i = 2;
do{
    a += 1;
    i += 2;
} while(i < 2);
```

```
console.log(a);
```

```
1
```

1.6 Fonctions

Les fonctions sont l'un des éléments fondamentaux de JS. Une fonction est essentiellement un **ensemble d'instructions nommé et réutilisable** qui exécute une tâche précise.

Pour déclarer une fonction en JavaScript, on utilise le mot clef `function` et le nom de la fonction. On met ensuite des parenthèses suivies des arguments qu'on veut en entrée. Pour retourner une valeur, on met le mot clef `return` suivie de la valeur qu'on veut retourner. Pour utiliser une fonction, il suffit de mettre son nom, suivi de parenthèses et des arguments qu'on doit donner à cette fonction.

Exemple :

```
function somme(a,b){
    return a+b;
}
console.log(somme(4,5));
console.log(somme(10,2));
```

```
9
12
```

La valeur d'un paramètre par défaut est `undefined`.

```
function saluer(prenom, nomFamille) {
    console.log('Bonjour ' + prenom + ' ' + nomFamille);
}
saluer('Jean');
saluer('Marie');
```

```
Bonjour Jean undefined
Bonjour Marie undefined
```

```
saluer('Jean', 'Cloutier');
```

```
Bonjour Jean Cloutier
```

Le langage JavaScript dispose de nombreuses **fonctions natives** ou prédéfinies qui sont prêtes à l'emploi. Pour être tout à fait précis, les fonctions prédéfinies en JavaScript sont des méthodes. Une méthode est tout simplement le nom donné à une fonction définie au sein d'un objet. Par exemple, le JavaScript dispose d'une fonction nommée `random()` qui appartient à l'objet `Math` et qui va générer un nombre décimal aléatoire entre 0 et 1; ou d'une fonction `log()` qui appartient à l'objet `console` et qui vas permettre d'imprimer sur la console; ou encore d'une fonction `replace()` qui appartient à l'objet `String` et qui va nous permettre de chercher et de remplacer une expression par une autre dans une chaine de caractères. Pour le moment, nous allons considérer que ce sont simplement des fonctions.

```
var message = 'Bonsoir Pierre';
console.log(message.replace('Pierre', 'Mathilde'));
console.log(Math.random());
```

```
Bonsoir Mathilde
0.21293208441161604
```

Créer nos propres fonctions va nous permettre de gagner du temps de développement et de créer des scripts plus facilement maintenables et plus sécurisés.

Les noms des fonctions suivent les mêmes règles que ceux des variables (notation du chameux, commencer par une lettre, ne pas contenir d'espace ni de caractères spéciaux et ne pas être déjà pris nativement par JavaScript).