Semaine 2 : Les structures de contrôles et les fonctions en JavaScript

Cours_ UVCI
PRW2103-1

DIABATE Nabègna,
Université Virtuelle de Côte d'Ivoire

juin 2018
1.0

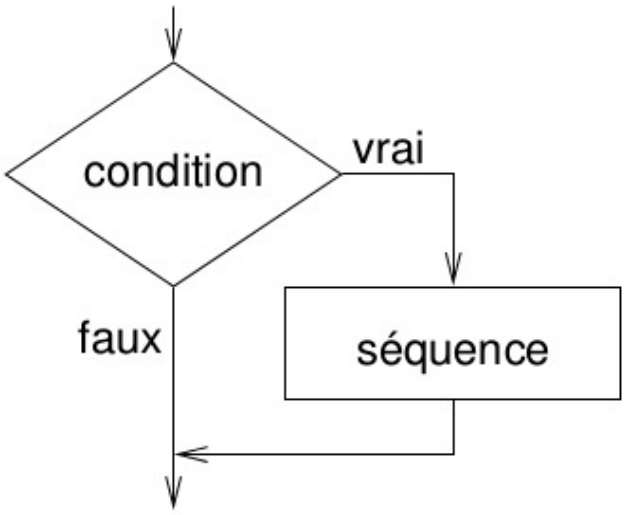
Table des matières

Objectifs

À la fin de cette leçon, vous serez capable de :

- *utiliser* les conditions en programmation JavaScript ;
- *utiliser* les boucles et JavaScript.

Introduction



En programmation informatique, une structure de contrôle est une instruction particulière d'un langage de programmation impératif pouvant dévier le flot de contrôle du programme la contenant lorsqu'elle est exécutée.

Cette leçon est une application des structures de contrôles et des fonctions vues aux cours d'algorithmique et d'algorithmique avancée.

Nous ne nus attarderons donc pas sur l'explication des tournures mais plutôt de leur application avec JavaScript.

Section 1 : Structures conditionnelles

1. 1. Tests conditionnels avec if et else

Les instructions if et else mettent en place une condition dans le code. Elles permettent de différencier plusieurs cas lors de l'exécution.

```
if (condition){
  ...
}else{
  ...
}
```

Dans la condition, on se sert généralement des opérateurs de comparaison mais toute fonction renvoyant un booléen est autorisée.

```
var age = 15;

if (age >= 18){
  document.write("Vous pouvez entrer.");
}else{
  document.write("Vous ne pouvez pas entrer.");
}
```

Notez que ceci est également possible (lorsque le bloc à exécuter ne nécessite qu'une seule ligne) :

```
var age = 15;

if (age >= 18) document.write("Vous pouvez entrer.");
else document.write("Vous ne pouvez pas entrer.");
```

On peut aussi rajouter des conditions intermédiaires entre le if et le else qui permettront de tester plus de cas. Cet exemple montre comment gérer simplement les conditions multiples en JavaScript.

```
var temps = "orageux";
var je_prends;

if (temps == "ensoleillé") je_prends = "mon parasol.";
else if (temps == "nuageux") je_prends = "mon chapeau.";
else if (temps == "pluvieux") je_prends = "mon parapluie.";
else je_prends = "mon paratonnerre.";

document.write("Lorsque le temps est "+temps+", je prends "+je_prends);
// Lorsque le temps est orageux, je prends mon paratonnerre.
```

Il est possible de rallier plusieurs conditions en une seule. Évidemment, la précision sera moindre mais cela se révèle vite incontournable et très pratique. Pour cela il existe deux opérateurs très simples d'utilisation.

- *&&* : il veut dire ET.

- || : il veut dire OU.

Voyons comment cela fonctionne :

```
var temps = "neigeux";
var temperature = -5;
var je_prends;

if (temps == "neigeux" || temperature <= 0) je_prends = "ma luge ou mon gros blouson.";
else if (temps == "neigeux" && temperature <= 0) je_prends = "ma luge et mon gros blouson.";
else je_prends = "mes lunettes de soleil."; // Avec un peu de chance il fait beau :)

document.write("Lorsque le temps est "+temps+" et que la température est de "+temperature+"{{Abbr|°C|degré Celcius}}, je prends "+je_prends);
// Lorsque le temps est orageux et que la température est de -5{{Abbr|°C|degré Celcius}}, je prends ma luge ou mon gros blouson.
```

On peut aussi séparer les ET et les OU avec des parenthèses.

2. 2. L'alternative switch

Il existe une alternative intéressante aux if et else. En effet, lorsque vous avez un nombre important de cas à vérifier, il peut être intéressant de ne pas avoir à recopier une cascade de else if.

L'instruction switch permet de gérer plusieurs cas en fonction de la valeur d'une variable. Son utilisation, bien que pratique, est réputée être (un peu) plus lente que le if et surtout assez restreinte.

Sa syntaxe est la suivante :

```
switch (variable)
{
  case a:  /*...*/  break;
  case b:  /*...*/  break;
  case c:  /*...*/  break;
  default:  /*...*/  break;
}
```

En appliquant un exemple concret, on a :

```
var temps = "soleil";
var je_prends;

switch (temps)
{
  case "soleil": je_prends = "mon parasol."; break;
  case "nuageux": je_prends = "mon chapeau."; break;
  case "pluvieux": je_prends = "mon parapluie."; break;
  default: je_prends = "mon paratonnerre."; break;
}

document.write("Lorsque le temps est "+temps+", je prends "+je_prends);
// Lorsque le temps est soleil, je prends mon parasol.
```

3. 3. Structures ternaires

Les structures ternaires évoquent un procédé raisonnablement complexe mais surtout très pratique et permettant de gagner du temps, de la place et de la lisibilité.

Les ternaires sont en fait un concentré d'un groupe *if* et *else* (sans *else if* au milieu). Cela permet d'écrire une condition en une seule ligne. C'est une alternative aux structures conditionnelles vues précédemment. Voici la syntaxe :

(condition) ? ...bloc du if : ...bloc du else

Le "*?*" remplace donc l'accolade du *if* tandis que les " *:* " remplacent celle du *else*. Un code avec *if* et *else* :

```
1 var age = 15;
2 if (age >= 18) document.write("Vous pouvez entrer.");
3 else document.write("Vous ne pouvez pas entrer.");
```

Un code avec structure ternaire renvoyant le même résultat :

```
1 var age = 15;
2 document.write((age >= 18) ? "Vous pouvez entrer." : "Vous ne pouvez pas entrer."
  );
```

Ici, on a mis la condition en ternaire directement dans le `document.write`.

4. Exercice : Exercices de compréhension

Exercice

1) Une structure conditionnelles peut-être rallier plus de deux conditions

- ◯ Vrai
- ◯ Faux

Exercice

2) Lorsqu'on utilise le mot-clé `if`, on doit forcément utiliser `else` par la suite

- ◯ Vrai
- ◯ Faux

Exercice

3) On désire délibérer sur les notes des étudiants en L2. Si la moyenne annuelle de l'étudiant est supérieure ou égale à 11 alors il est admis sont il est ajourné.

En considérant que la note est saisie dans une variable déclarée par `var NoteEtd` ; complétez le programme suivant en utilisant les structures conditionnelles.

<script>

var NoteEtd = prompt("Quelle est sa note?", "Note de l'etudiant"); // Invite a saisir la note

if {

document.write("Il est admis.");

}

document.write("Il est recalé.");

}

</script>

Exercice

4) On désire modifier le code de l'exercice précédent selon une nouvelle procédure : un étudiant est vraiment admis si l'on confirme qu'il a validé toutes les UE en plus d'avoir la note de 11. S'il a validé toutes les UE la variable ValidUE= 1 sinon ValidUE=0.

```
<script>
  var NoteEtd = prompt("Quelle est sa note?", "Note de l'etudiant"); // Invite
a saisir la note
  if(NoteEtd>=11){
    var ValidUE = prompt("Toutes les UE validées?", " ");
      if(...){
      document.write("Il est admis.");
      }...{
       document.write("Il est recalé.");
      }
  }...
    document.write("Il est recalé.");
  }
</script>
```

<script>

var NoteEtd = prompt("Quelle est sa note?", "Note de l'etudiant"); // Invite a saisir la note

if(NoteEtd>=11){

var ValidUE = prompt("Toutes les UE validées?", " ");

if( ){

document.write("Il est admis.");

} {

document.write("Il est recalé.");

}

}

document.write("Il est recalé.");

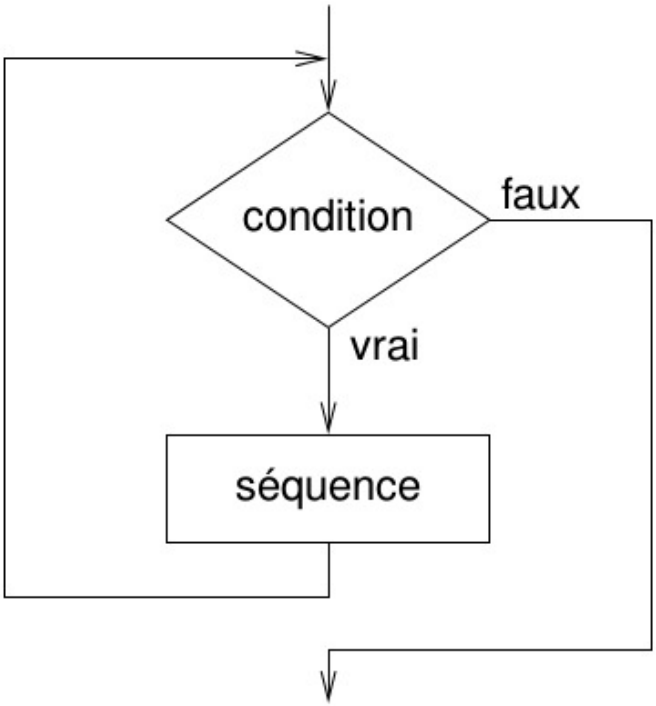
}

</script>

Section 2 : Structures répétitives (les boucles)

II

Une boucle, c'est un bloc d'instructions qui sera répété tant qu'une condition sera vérifiée (= `true`). Une boucle doit être finie ; elle finira à un moment où la condition n'est plus vraie. Une boucle infinie est une erreur et fera planter votre navigateur.



1. 1. La boucle while

L'instruction *`while`* est la boucle la plus simple d'utilisation puisqu'elle ne comporte qu'un seul "`argument`" : la condition ! Voici la syntaxe de *`while`* :

```
while(condition)
{
... (instructions, incrémentation...)
}
```

Voici un exemple simple :

```
1 var i = 0;
2 while (i < 5){
```

```
  document.write("Ligne "+i+"<br />");
  i++;
}
document.write("Boucle terminée !");
```

LE résultat sera ceci :

Ligne 0

Ligne 1

Ligne 2

Ligne 3

Ligne 4

Boucle terminée !

2. 2. L'instruction do... while

La boucle `do while` est la sœur de la boucle `while` puisqu'il s'agit de la même boucle *SAUF que le bloc d'instruction s'effectuera au minimum une fois* et ce même si la condition de la boucle n'est pas vérifiée dès le début.

```
var i = 5;
do
{
  document.write("Ligne "+i+"<br />");
  i++;
} while (i < 5);

document.write("Boucle terminée !");
```

Ici, la condition vaut `false` dès le début. Un `while` normal aurait donc sauté la condition pour passer à la suite du code. Mais le `do while` exécute une fois le code et affiche :

Ligne 5

Boucle terminée !

3. 3. L'instruction for

Cette structure répétitive ne porte pas bien son nom contrairement aux précédentes. Pour faire court, for est un alias de while à la différence qu'elle regroupe en son sein l'initialisation de(s) variable(s) utilisée(s), la condition ainsi que l'incrémentation (décrémentation) de celle(s)-ci.

Syntaxe :

```
for (initialisation; condition; incrémentation)
{
...
}
```

Exemple :

```
for (i = 0; i<5; i++)
{
  document.write("Ligne "+i+"<br />");
}
document.write("Boucle terminée !");
```

Le résultat est le même que celui de la boucle `while`.

Ceci dit, l'utilisation d'une des trois boucles dépend du contexte et des besoins du script.

4. 4. Les instructions break et continue

Il arrive parfois que l'on veuille sortir d'une boucle alors même que la condition de la boucle est encore vraie. Pour cela, on peut faire appel à l'instruction *break*. Lorsque cette instruction est exécutée, la boucle se termine immédiatement et le code la suivant est exécuté.

```
1 var i = 1;
2 while (i <= 10)
3 {
4   if (i == 5) break;
5   i++;
6 }
7 document.write("La valeur de i est "+i);
```

Ce qui affiche : *La valeur de i est 5*. En effet, le programme sort de la boucle lorsque i prend la valeur de 5, donc après 4 tours.

Le rôle de l'instruction *continue* est de sauter le tour actuel et d'en recommencer un nouveau. Cette instruction s'utilise exactement de la même façon que *break*.

5. Exercice

Exercice

1) Lesquelles des structures suivantes sont des structures repétitives

- [ ] Do .... While
- [ ] If....Else
- [ ] for
- [ ] While

Exercice

2) Quelles structure peuvent être utilisée pour afficher un formulaire qui reste toujours affiché tant que l'utilisateur n'as pas saisi un multiple de 5 ?

- [ ] Do .... While
- [ ] for
- [ ] While

Exercice

3) Quelles sont les structures qui ne peuvent fonctionner sans que l'ensemble des éléments suivants ne soit défini ?

- compteur initial

- condition compteur

- incrément

- [ ] Do .... While
- [ ] for
- [ ] While

Exercice

4) Existe-il une instruction qui permet de sortir d'une boucle Do...while avant même que la condition de sortie de la boucle ne soit vérifiée ? Si oui, laquelle ?

- ☐ Oui
- ☐ Non
- ☐ Endif
- ☐ break
- ☐ exit

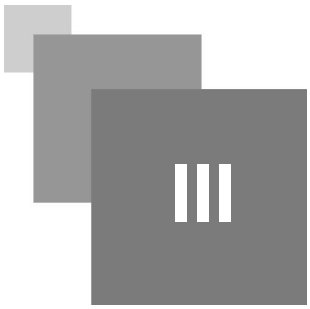
Exercice

5) On veut trouver le plus petit multiple d'un nombre saisi par l'utilisateur et stocké dans une variable nommée maVariable Compris entre deux nombres Nbre1 et Nbre2 tel que Nbre1<Nbre2.

On se propose d'utiliser la boucle `while` sachant le a et b étant deux variables l'opération `a%b` retourne le reste de la division euclidienne de a par b et qu'un un nombre x est multiple de y si `x%y=0`.

Complétez le code suivant pour résoudre le problème posé.

```
var Cpt=1 ;
while ( [Cpt >Nbre1 | Cpt < Nbre1] && [Cpt <Nbre2 | Cpt >Nbre2] ){
if (Cpt % maVariable == 0)      ;
Cpt++;
}
document.write("Le plus petit diviseur trouvé est"+Cpt);
```



Section 3 : Les fonctions en JavaScript

Les fonctions font partie des briques fondamentales de JavaScript. Une fonction est une procédure JavaScript, un ensemble d'instructions effectuant une tâche ou calculant une valeur. Afin d'utiliser une fonction, il est nécessaire de l'avoir auparavant définie au sein de la portée dans laquelle on souhaite l'appeler.

1. 1. Définir des fonctions

Une *définition de fonction* (aussi appelée *déclaration* de fonction ou *instruction* de fonction) est construite avec le mot-clé `function`, suivi par :

- Le *nom* de la fonction.
- Une liste d'*arguments* à passer à la fonction, entre *parenthèses* et séparés par des *virgules*.
- Les *instructions* JavaScript définissant la fonction, entre accolades, *{ }*.

Le code suivant, par exemple, définit une fonction intitulée carré :

```
function carré(nombre) {
  return nombre * nombre;
}
```

La fonction carré prend un seul argument, appelé nombre. La fonction est composée d'une seule instruction qui renvoie l'argument de la fonction (nombre) multiplié par lui-même. L'instruction `return` spécifie la valeur qui est renvoyée par la fonction.

Les arguments ne sont pas obligatoires dans la définition d'une fonction. Ainsi, on pourrait définir une fonction sans argument comme par exemple générer un nombre aléatoire entre 0 et 50 :

```
function alea50() {
  var nb = Math.floor(Math.random()*51);
  return nb;
}
```

Les paramètres primitifs (comme les nombres) sont passés à la fonction par valeur. La valeur est passée à la fonction mais *si cette dernière change la valeur du paramètre, cela n'aura pas d'impact au niveau global ou au niveau de ce qui a appelé la fonction.*

Expression de fonctions

Syntaxiquement, la déclaration de fonction utilisée ci-dessus est une instruction. On peut également créer une fonction grâce à une *expression de fonction*. De telles fonctions peuvent être anonymes (ne pas avoir de nom correspondant). La fonction carré aurait pu être définie de la façon suivante :

```
var carré = function (nombre) { return nombre * nombre };
var x = carré(4); //x reçoit la valeur 16
```

Cependant, un nom peut être utilisé dans une expression de fonction, ce afin de l'utiliser dans la

fonction (*récursivité*) ou afin de l'identifier dans les appels tracés par un éventuel débogueur :

```
var factorielle = function fac(n) { return n < 2 ? 1 : n * fac(n - 1) };

console.log(factorielle(3));
```

En JavaScript, une fonction peut être définie selon une condition. Le fragment de code qui suit définit une fonction seulement si *num* vaut 0 :

```
var maFonction;
if (num === 0){
  maFonction = function(monObjet) {
    monObject.fabricant = "Toyota"
  }
}
```

2. 2. Les appels de fonctions

La seule définition d'une fonction ne permet pas d'exécuter la fonction. Cela permet de lui donner un nom et de définir ce qui doit être fait lorsque la fonction est appelée. Appeler la fonction permet d'effectuer les actions des instructions avec les paramètres indiqués. Par exemple, si on définit la fonction carré, on peut l'appeler de la façon suivante :

```
carré(5);
```

Cette instruction appellera la fonction avec un argument valant 5. La fonction exécute ses instructions et renvoie la valeur 25.

Les fonctions doivent appartenir à la portée dans laquelle elles sont appelées. La portée d'une fonction est la fonction dans laquelle elle est déclarée (son champ d'influence) ou le programme entier si elle est déclarée au niveau le plus haut.

En effet, *on ne peut pas accéder aux variables définies dans une fonction en dehors de cette fonction* : ces variables n'existent que dans la portée de la fonction. En revanche, une fonction peut accéder aux différentes variables et fonctions qui appartiennent à la portée dans laquelle elle est définie. Une fonction définie dans une autre fonction peut également accéder à toutes les variables de la fonction « parente » et à toute autre variable accessible depuis la fonction « parente ».

```
// Les variables suivantes sont globales
var num1 = 20,
    num2 = 3,
    nom = "Licorne";

// Cette fonction est définie dans la portée globale
function multiplier() {
  return num1 * num2;
}

multiplier(); // Renvoie 60

// Un exemple de fonction imbriquée
function getScore () {
  var num1 = 2,
      num2 = 3;

  function ajoute() {
    return nom + " a marqué " + (num1 + num2);
  }

  return ajoute();
}

```

```
getScore(); // Renvoie "Licorne a marqué 5"
```

Aussi, une fonction peut-elle être *récursive*, c'est-à-dire qu'elle peut s'appeler elle-même. Voici la fonction qui calcule récursivement la factorielle d'un nombre :

```
function factorielle(n){
  if ((n === 0) || (n === 1))
    return 1;
  else
    return (n * factorielle(n - 1));
}
```

On peut ensuite calculer les factorielles des nombres 1 à 5 :

```
var a, b, c, d, e;
a = factorielle(1); // a reçoit la valeur 1
b = factorielle(2); // b reçoit la valeur 2
c = factorielle(3); // c reçoit la valeur 6
d = factorielle(4); // d reçoit la valeur 24
e = factorielle(5); // e reçoit la valeur 120
```

3. 3. Les fonctions prédéfinies

JavaScript possède plusieurs fonctions natives, disponibles au plus haut niveau :

- *eval()*

La fonction `eval()` permet d'évaluer du code JavaScript contenu dans une chaîne de caractères.

- *uneval()*

La fonction `uneval()` crée une représentation sous la forme d'une chaîne de caractères pour le code source d'un objet.

- *isFinite()*

La fonction `isFinite()` détermine si la valeur passée est un nombre fini. Si nécessaire, le paramètre sera converti en un nombre.

- *isNaN()*

La fonction `isNaN()` détermine si une valeur est `NaN` ou non. Note : On pourra également utiliser `Number.isNaN()` défini avec ECMAScript 6 ou utiliser `typeof` afin de déterminer si la valeur est *Not-A-Number*.

- *parseFloat()*

La fonction `parseFloat()` convertit une chaîne de caractères en un nombre flottant.

- *parseInt()*

La fonction `parseInt()` convertit une chaîne de caractères et renvoie un entier dans la base donnée.

- *decodeURI()*

La fonction `decodeURI()` décode un Uniform Resource Identifier (URI) créé par `encodeURI()` ou une méthode similaire.

- *decodeURIComponent()*

La fonction `decodeURIComponent()` décode un composant d'un Uniform Resource Identifier (URI) créé par `encodeURIComponent` ou une méthode similaire.

- *encodeURI()*

La fonction `encodeURI()` encode un Uniform Resource Identifier (URI) en remplaçant chaque exemplaire de certains caractères par un, deux, trois voire quatre séquences d'échappement représentant l'encodage UTF-8 du caractères (quatre séquences seront utilisées uniquement lorsque le caractère est composé d'une paire de deux demi-codets).

- *encodeURIComponent()*

La fonction `encodeURIComponent()` encode un composant d'un Uniform Resource Identifier (URI) en remplaçant chaque exemplaire de certains caractères par un, deux, trois voire quatre séquences d'échappement représentant l'encodage UTF-8 du caractères (quatre séquences seront utilisées uniquement lorsque le caractère est composé d'une paire de deux demi-codets).

- *escape()*:

La fonction dépréciée `escape()` calcule une nouvelle chaîne de caractères pour laquelle certains caractères ont été remplacés par leur séquence d'échappement hexadécimale. Les fonctions `encodeURI()` ou `encodeURIComponent()` doivent être utilisées à la place.

- *unescape()*

La fonction dépréciée `unescape()` calcule une nouvelle chaîne de caractères pour laquelle les séquences d'échappement hexadécimales sont remplacées par les caractères qu'elles représentent. Les séquences d'échappement introduites peuvent provenir d'une fonction telle que `escape()`. `unescape` est dépréciée et doit être remplacée par `decodeURI()` ou `decodeURIComponent()`.

4. Exercice

Exercice

1) Une fonction peut-elle prendre plusieurs arguments ?

◯ Oui

◯ Non

Exercice

2) Lesquelles des structures suivantes sont correctes pour définir une fonction en JavaScript ?

☐ function maFonction() {
// code ici.....
}

☐ fonction maFonction() {
// code ici.....
}

☐ function maFonction {
// code ici.....
}

☐ function maFonction(10) {
// code ici.....
}

Exercice

3) Quelle instruction permet à une fonction de retourner une valeur utilisable dans la suite du programme ?

Exercice

4) Une fonction peut-elle avoir pour argument une autre fonction ?

- ◯ Oui
- ◯ Non

Exercice

5) Queml est le type de la variable défini par

```
1 var maJolieVar=maBelleFonction(NombreEntier){ var maChose="Code JavaScript";
  return maChose;}
```

- ◯ number
- ◯ string
- ◯ null
- ◯ ce code est incorrecte