05 Instructions conditionnelles et alternatives

L'action conditionnée

L'action conditionnée est une instruction élémentaire ou une suite d'instructions exécutées en séquence si l'état du système l'autorise. Le(s) critère(s) à respecter pour exécuter l'action s'exprime(nt) à l'aide d'une condition (= prédicat) évaluable au moment précis où l'action doit, le cas échéant, intervenir.

Lors de l'exécution du programme, le processeur est donc amené à évaluer la condition. La condition évaluée constitue alors un énoncé (ou proposition) vrai ou faux.

Schéma :

```
Si _prédicat_ alors
   _Instruction 1_
   _Instruction 2_
   ...
   _Instruction N_
Fin si
```

Exemple 1

```
Si Température > 38 alors
   Écrire "Le patient a de la fièvre"
Fin si
```

Exemple 2

```
Si Température > 41 et Tension > 25 alors
   Écrire "Le patient va perdre patience"
Fin si
```

Exemple 3

```
Si non Patient alors
   Écrire "Éconduire l'olibrius"
Fin si
```

Exemple 4

```
Si Température > 42 ou (Tension < 25 et Pouls > 180) alors
   Écrire "Prévenir la famille"
Fin si
```

Exemple 5

```
Si Température > 40 ou Tension >= 25 alors
   Écrire "Hospitaliser le patient"
Fin si
```

Exemple 6

```
Si Patient ET Pouls = 0 alors
    Écrire "Appeler le curé"
Fin si
```

Syntaxe Javascript :

```
if (prédicat)
{
   Instruction 1
   Instruction 2
   ...
   Instruction N
}
```

Les conditions (expressions `if`, `else`, `switch`)

A un moment ou à un autre de la programmation, on aura besoin de tester une condition. Ce qui permettra d'exécuter ou non une série d'instructions.

Condition simple

Dans sa formulation la plus simple, l'expression `if` se présente comme suit :

```
if (conditionvraie)
{
   // une (ou plusieurs) instructions;
}
```

- Si la condition est vérifiée, les instructions s'exécutent.
- Si elle ne l'est pas, les instructions ne s'exécutent pas et le programme passe à la commande suivant l'accolade de fermeture.

Exemple :

```
let reponse = "oui";

if (reponse === "oui")
{
    console.log("Bonne réponse !");    // Affichera 'Bonne réponse' dans la console
}
```

> Dans le cas où il n'y a qu'une seule instruction, les accolades peuvent être omises, mais ce n'est pas recommandé pour la lisibilité du code :

```
if (reponse === "oui")
   console.log("Bonne réponse !");
```

Conditions multiples : les opérateurs logiques

Les opérateurs logiques **ET** (signe `&&`) et **OU** (signe `||`) permettent de tester une association de conditions :

- Utilisation du **ET** : `if ((condition1) && (condition2))` teste si la condition 1 **ET** la condition 2 sont réalisées (les deux à la fois)
- Utilisation du **OU** : `if ((condition1) || (condition2))` teste si au moins UNE des 2 conditions est vraie. Le fonctionnement est le suivant :
  - si la 1ère condition est vraie, on ne teste pas la 2ème ; si la 1ère est fausse, on teste si la deuxième est vraie.
  - Si aucune des 2 n'est vraie, *faux* est renvoyé.

Exemple 1, avec ET :

```
// Condition avec ET
if (age > 18 && permis == "Voiture")
{
    console.log("Vous avez plus de 18 ans ET vous avez un permis de conduire voiture, vous pouvez conduire");
}
```

Dans cet exemple, les usagers ayant un permis moto ne peuvent donc pas conduire, quel que soit leur âge !

> IMPORTANT : Remarquez le signe `==` pour la comparaison *permis == "Voiture"*. Ce signe "double égal" est l'opérateur de comparaison, à ne pas confondre avec `=` seul qui est l'opérateur d'affectation. Retenez également qu'en JavaScript moderne il est important de privilégier l'opérateur de comparaison strict `===` !

Exemple 2, avec OU :

```
// Condition avec OU
if (age > 18 || permis === "Voiture")
{
    console.log("Vous avez plus de 18 ans");
}
```

Dans cet exemple, cette fois un usager ayant un permis de conduire (Voiture) ou ayant plus de 18 ans est majeur !

Il faut rester vigilant sur la façon d'écrire les conditions. Des conditions mal définies peuvent fausser un algorithme !

Autres opérateurs logiques de comparaison

Dans les exemples du paragraphe précédent, le signe `>` a servi d'opérateur, c'est-à-dire que l'on a considéré que l'âge est supérieur à 18 ans, donc à partir de 19 ans, 18 ans n'étant pas pris en compte.

Pour prendre en compte l'âge de 18 ans, il est nécessaire d'ajouter le signe égal :

```
if (age >= 18 || permis === "Voiture")
{
    console.log("Vous avez plus de 18 ans OU vous avez un permis de conduire voiture, vous pouvez conduire");
}
```

Signe	Signification	Exemple	Condition posée
==	Comparaison	x == 11	Est-ce que x vaut 11 ?
===	Égalité stricte (valeur et type)	x === 11	Est-ce que x est un nombre égal à 11 ?
<	Inférieur (strictement)	x < 11	Est-ce que x est strictement inférieur à 11 ? - Oui si x vaut jusqu'à 10 - Non si x vaut 11 ou plus.
<=	Inférieur ou égal	x <= 11	Est-ce que x est inférieur ou égal à 11 ? - Oui si x vaut jusqu'à 11 - Non si x vaut plus de 11.
>	Supérieur (strictement)	x > 11	Est-ce que x est strictement supérieur à 11 ? - Oui si x vaut au moins 12 - Non si x vaut 11 ou moins.
>=	Supérieur ou égal	x >= 11	Est-ce que x est supérieur ou égal à 11 ? - Oui si x vaut au moins 11 - Non si x vaut moins de 11.
!=	Différence	x != "toto"	Est-ce que x est différent de la chaîne "toto" ?
!==	Différence stricte (valeur ou type différent)	x !== 11	Est-ce que x est différent (en valeur ou en type) de 11 ?

En JavaScript moderne, on recommande d'utiliser `===` et `!==` plutôt que `==` et `!=`.

Conditions avec `else`

Si l'on reprend l'exemple du paragraphe *condition simple*, on voit qu'un autre choix, *non*, est possible. Pour traiter ce second cas, on pourrait logiquement ajouter une seconde condition :

```
let reponse = "oui";

// 1er cas
if (reponse === "oui")
{
    console.log("Bonne réponse !");
}

// 2ème cas
if (reponse === "non")
{
    console.log("Mauvaise réponse !");
}
```

Ce code n'est pas faux mais pour un choix simple à deux possibilités comme ici, c'est-à-dire booléen, il existe une instruction permettant de le simplifier : `else` que l'on traduit par *sinon* :

- Si la condition est vérifiée (true), le bloc d'instructions 1 s'exécute.
- Si elle ne l'est pas (false), le bloc d'instructions 2 s'exécute.

```
if (condition vraie)
{
   instructions 1;
}
else
{
   instructions 2;
}
```

Exemple :

```
if (reponse === "oui")
{
   console.log("Bonne réponse !");
}
else
{
   console.log("Mauvaise réponse !");
}
```

Cet exemple est le plus simple mais on peut néanmoins appliquer une condition sur le else :

Exemple :

```
if (reponse === "A")
{
   console.log("Bonne réponse !");
}
else if (reponse === "B")
{
   console.log("Mauvaise réponse !");
}
```

Tester l'exemple avec les 2 possibilités.

Il se pose cependant un problème : si aucun des deux cas n'est avéré, il ne se passera rien ou plutôt on ne rentrera dans aucune des 2 conditions.

Reprenons l'exemple et testons la réponse *C* :

```
let reponse = "C";

if (reponse === "A")
{
   console.log("Bonne réponse !");
}
else if (reponse === "B")
{
   console.log("Mauvaise réponse !");
}
```

Pour éviter cela et gérer tous les cas possibles, il est recommandé d'ajouter un `else` sans condition qui permettra donc de traiter tous les autres cas (notez bien que ce `else` final reste facultatif) :

```
let reponse = "C";

if (reponse === "A")
{
   console.log("Bonne réponse !");
}
else if (reponse === "B")
{
   console.log("Mauvaise réponse !");
}
else
{
   console.log("Réponse inconnue.");
}
```

L'instruction `switch`

L'instruction `switch` permet d'écrire un ensemble de conditions sous une autre forme. Elle est introduite par le mot-clé `switch` qui reçoit en argument la variable à tester, puis, entre accolades, les mot-clés case reçoivent les différentes valeurs attendues pour cette variable. Pour chaque cas on exécute alors des instructions. Chaque bloc case doit se terminer obligatoirement par l'instruction `break` qui permet de sortir du `switch` une fois la condition réalisée (si absente toutes les cas sont exécutés !).

```
let variable = "1";

switch (variable)
{
  case "1" :
     console.log("Cas 1");
     break;

  case "2" :
     console.log("Cas 2");
     break;

  case "3":
     console.log("Cas 3");
     break;
}
```

Exemple :

```
let modele = "Clio";

switch (modele)
{
  case "208" :
     console.log("Modèle 208 : marque Peugeot");
     break;

  case "Clio" :
     console.log("Modèle Clio : marque Renault");
     break;

  case "C3" :
     console.log("Modèle C3 : marque Citroën");
     break;
}
```

L'instruction `default` peut être ajoutée de façon facultative. Elle se place à la fin du bloc `switch` après tous les `case`. Son rôle est similaire au else des conditions avec `if`, c'est-à-dire que si aucune condition n'est réalisée dans les `case` (on n'est pas sorti du `switch` car on n'a pas rencontré de `break`) ce sont les instructions contenues dans ce bloc `default` qui s'appliqueront :

```
let modele = "A4";

switch (modele)
{
  case "208" :
     console.log("Modèle 208 : marque Peugeot");
     break;

  case "Clio" :
     console.log("Modèle Clio : marque Renault");
     break;

  case "C3" :
     console.log("Modèle C3 : marque Citroën");
     break;

  default:
     console.log("Modèle "+modele+": marque inconnue");
}
```

Notez qu'il n'y a pas d'instruction `break` pour terminer le bloc `default`.

Enfin, sachez qu'il est possible de grouper des cas auxquels on appliquera des instructions communes. Dans l'exemple ci-après, les Laguna et Clio sont 2 modèles de la marque Renault :

```
let modele = "Clio";

switch (modele)
{
  case "508" :
     console.log("Modèle 508 : marque Peugeot");
     break;

  case "Clio" :
  case "Laguna" :
     console.log("Modèle "+modele+" : marque Renault");
     break;

  case "C5" :
     console.log("Modèle C5 : marque Citroën");
     break;

  default:
     console.log("Modèle "+modele+": marque inconnue");
}
```

Imbrication de conditions

Un bloc de conditions peut contenir un ou plusieurs autres blocs de conditions (`if` ou `switch`) :

```
let reponse = "oui";
        let score = 19;

if (reponse === "oui")
{
   console.log("Bonne réponse!");

   score++; // Augmente le score de 1

   if (score === 20)
   {
       console.log("Vous avez gagné !");
   } // fin du 2ème if

} // fin du 1er if
```

Condition ternaire

Il existe une autre forme d'écriture des conditions. Cette forme est dite « ternaire » et revêt diverses appellations : condition ternaire, forme ternaire, écriture ternaire ou encore opérateur (ternaire) conditionnel.

Une condition ternaire s'écrit sous cette forme :

```
(condition) ? instruction 1 : instruction 2
```

Fonctionnement :

- Si la condition entre parenthèses est vraie, l'instruction 1 est exécutée,
- sinon (condition entre parenthèses est vraie (évaluée à TRUE), sinon - condition entre parenthèses évaluée à FALSE - c'est l'instruction 2 qui est exécutée.

Exemple :

```
let age = 19;

(age >= 18) ? console.log("Vous êtes majeur") : console.log("Vous êtes mineur");
```

Bien que l'on puisse rencontrer parfois cette forme d'écriture, son utilisation est en général déconseillée pour des raisons de lisibilité du code.

Exercices

Exercice 1 - Parité

Ecrivez un programme qui demande un nombre à l'utilisateur puis qui teste si ce nombre est pair. Le programme doit afficher le résultat *nombre pair* ou *nombre impair*. Vous devez utiliser l'opérateur modulo `%` qui donne le reste d'une division. `a%2` donne le reste de la division de `a` par 2, si ce reste est égal à zéro, `a` est divisible par 2.

Exercice 2 - Age

Ecrivez un programme qui demande l'année de naissance à l'utilisateur. En réponse votre programme doit afficher l'âge de l'utilisateur et indiquer si l'utilisateur est majeur ou mineur.

Exercice 3 - Calculette

Faire la saisie de 2 nombres entiers, puis la saisie d'un opérateur `+`, `-`, `*` ou `/`.

Si l'utilisateur entre un opérateur erroné, le programme affichera un message d'erreur.

Dans le cas contraire, le programme effectuera l'opération demandée (en prévoyant le cas d'erreur *division par 0*), puis affichera le résultat.