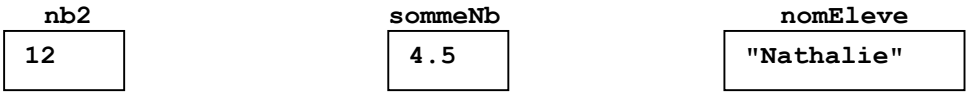
On vous présente ici des notions qui sont communes au plupart des langages de programmation. Ici on vous illustre Java et JavaScript en vous montrant les petites différences entre ces deux langages. Ces langages comme tous les autres langages modernes sont dérivés du langage C (leur ancêtre).

1. Concept de variable

Une **variable** est un emplacement en mémoire représenté par un nom (une suite de caractères) que l'on utilise pour conserver une donnée.

- Il est important de choisir le nom des variables de manière à rappeler la signification de la donnée qu'elle représente.
- Par convention, un nom de variable commence par une minuscule, chaque nouveau mot commence par une majuscule. On peut inclure des chiffres dans un nom mais on devrait éviter les accents.
- Une variable est représentée par : un **nom**, une **valeur** et un **type**.

Exemples :



2. Concept de constante

Une valeur **constante** peut être écrite telle quelle dans un programme. C'est une valeur qui ne changera jamais.

Exemples :

- `0.07`
- `235`
- `"Bonjour"`

Mais, dans bien des cas, il est préférable de donner un nom **symbolique** aux **constantes**, comme pour les variables. Par convention, les noms de constantes sont en majuscules, chaque nouveau mot étant séparé du précédent par le caractère _.

```
TAUX_TVA = 0.07;
MESSAGE = "Bonjour";
```

3. Le typage

JavaScript est un langage faiblement typé, son type dépend de la valeur qu'on va y mettre. La plupart des langages sont fortement typés comme Java, C++, C# et autres, toute donnée (***variable*** ou ***constante***) doit obligatoirement être déclarée avec un **type** donné.

Le **type** d'une donnée indique :

- le domaine des valeurs qu'elle peut prendre ;
- les opérations permises sur cette donnée.

En effet, un programme manipule des données qui sont soit de type **numérique** (entier, réel, …) soit de type **non numérique** (caractères ou chaînes de caractères).

Les types de base du langage Java (comme exemple seulement), en JavaScript on ne déclare pas les variables :

type	équivalent en français	taille en mémoire	exemples de valeurs

char	caractère unicode	16 bits	`'a'`
			`'$'`
			`'É'`

byte	entier	8 bits	Domaine : [-128, +127]
short	entier court	16 bits	Domaine : [-32768, +32767]
int	entier	32 bits	Domaine : [-2^{31}, $+2^{31}-1$]
long	entier long	64 bits	Domaine : [-2^{63}, $+2^{63}-1$]

float	réel	32 bits	`123.5f`
			`25.f` ou `25.0f`
			`-54.2f`
			`12E+2f (1200)`

double	réel double précision	64 bits	`123.5`
			`-54.26589`
			`0.5` ou `.5`

boolean	booléen (oui, non)	8 bits	`true` `false`
String	chaîne de caractères		`"Montréal, Qc"` `"9155 St-Hubert"`
On notera que `String` commence par une majuscule car ne n'est pas un type de base (ou type *primitif*) du langage Java.			

Exercice :

Pour chacun des nombres suivants, indiquez si le nombre est un entier, un réel ou s'il est invalide par l'interpréteur :

	nombre	entier	réel	invalide (pourquoi ?)
a	`234`			
b	`123.`			
c	`2E+4`			
d	`-17.2`			
e	`0.0`			
f	`-.25`			
g	`12.5E+2`			
h	`-100`			
i	`-112.5`			
j	`12 E23`			
k	`12E-3`			
l	`12E -3`			
m	`- 100`			
n	`0.125E`			

4. Déclarations

Toute donnée (***variable ou constante***) doit être **déclarée une et une seule fois**, avant d'être utilisée. La déclaration permet au programme :

- ☞ de réserver l'espace mémoire ;
- ☞ d'initialiser la donnée.

Syntaxe :

Déclaration d'une variable :
En JAVA `String nom;` en JavaScript `var nom ;`
Déclaration et initialisation d'une variable :
En JAVA `String nom = valeur;` en JavaScript `var nom = valeur ;`
Déclaration et initialisation d'une constante symbolique :
En JAVA `final String nom = valeur;` en JavaScript `const nom = valeur ;`

Exemples :

En JavaScript

```
var age;
age = 32 ; //ceci est une affectation

ou

var age = 32 ; //ceci est une initialisation

age est du type int (entier)

age = 32.5 ; //age est maintenant un float (réel)

ager = "Mon âge"; //age est maintenant un String (chaîne de caractères)
```

C'est ça la signification de faiblement typé

En Java et autres langages comme C++, C#, etc.

```
int age = 32; //donc un entier

age = "Mon âge"; //ERREUR age peut recevoir seulement des entiers
```

C'est ça la signification de fortement typé

5. La notion d'expression

La combinaison d'opérateurs et d'opérandes constitue **une expression** dont le résultat est une valeur.

- ☞ Un **opérateur** permet de faire un calcul.
- ☞ Un **opérande** peut être soit une constante, soit une variable.

Les différents opérateurs arithmétiques :

opérateur	exemple	remarques	valeur si `x = 7` et `y = 2`
`+`	`x + y`	Addition	`9`
`-`	`x - y`	Soustraction	`5`
`*`	`x * y`	Multiplication	`14`
`/`	`x / y`	Division Attention : Si les opérandes sont entiers, il s'agit de la **division entière**	`3`
`%`	`x % y`	Modulo (reste de la division)	`1`

La priorité des opérateurs :

Lors de l'évaluation d'une expression :

1 - On doit traiter les parenthèses en premier lieu
2 - La priorité des opérateurs intervient ensuite
3 - Pour les opérateurs de même priorité, l'évaluation se fait de gauche à droite

Table de priorité des opérateurs arithmétiques :

opérateurs		priorité
`( )`	Parenthèses	1
`+` `-`	Opérateur unaire d'addition Opérateur unaire de soustraction	2
`*, /, %`		3
`+` `-`	Opérateur binaire d'addition Opérateur binaire de soustraction	4

Voir la table de priorité des opérateurs en annexe.

Exemple :

```
          2 * 2 + ( 2 + 1 ) * ( 2 + 1 ) / 3
ordre :     3   6     1     4     2     5
```

Exercices :

Évaluez les expressions suivantes :

expression	résultat
`3.7 + (8 / 2) * (8 / 2)`	
`13 + 4 * 2 – 3 * (8 % 3 + 5)`	
`- (5 * 2) * (5 * 2) * 10 + 18 / (4 + 8 / 2 / 2)`	
`12 + 3 * 4 / 3`	
`24 + 36 / (6 * 3 / (7 + 2))`	
`3 + 5 * 6 / 3 - 5`	
`1.0 / (1.0 / 2)`	
`1.0 / 1.0 / 2.0`	

Expliquez la différence entre les 4 opérations suivantes :

opération	différence
`14 / 3`	
`14.0 / 3.0`	
`14.0 / 3`	
`14 % 3`	

6. La notion d'affectation

L'**affectation** est une opération fondamentale dans les langages de programmation. C'est elle qui permet de placer une valeur en mémoire. Par contre, il faut faire attention au type des variables des chaque côté de l'assignation (un réel ne peut pas être conservé dans un entier).

Exemples :

notation conceptuelle	notation en Java	explications
$x \leftarrow 3$	`x = 3;`	Conserve la valeur **3** à l'emplacement désigné par **x**
$y \leftarrow 4$ $x \leftarrow y$	`y = 4;` `x = y;`	Conserve la valeur de la variable **y (4)** à l'emplacement désigné par **x**
$y \leftarrow 4$ $x \leftarrow y * 2$	`y = 4;` `x = y * 2;`	Conserve la valeur de l'expression **y * 2 (8)** à l'emplacement désigné par **x**

Exercices :

Indiquez quelles sont les valeurs des variables entières **i**, **j** et **k** après l'exécution de chacune des instructions suivantes : *(les variables conservent leur valeur d'une ligne à l'autre)*

instruction	valeur de i	valeur de j	valeur de k
`i = 13;`			
`j = 5;`			
`i = j + 1;`			
`i = i % 5;`			
`j = j * i + 2;`			
`k = 1;`			
`k = (i * 7 + 3) / 5;`			

Sachant que `a = 4`, `b = 6`, `c = 2` et `d = 3`, calculez les expressions suivantes :

expression	résultat
`- a * ((2.0 + b) / c) * 2.5 * c`	
`(a * (b + d) - 2) * 3`	

Il est possible d'utiliser les opérateurs spéciaux pour simplifier l'écriture d'expressions courantes : des opérateurs d'affection d'addition, de soustraction, de multiplication, de division et de modulo, ainsi que des opérateurs unaires de post-incrémentation et de post-décrémentation.

Exemples :

opérateur	exemple d'utilisation	remplace l'expression
`+=`	`somme += nombre;`	`somme = somme + nombre;`
`-=`	`total -= 2;`	`total = total - 2;`
`*=`	`resultat *= valeur;`	`resultat = resultat * valeur;`
`/=`	`nombre /= facteur;`	`nombre = nombre / facteur;`
`%=`	`nombre %= 4;`	`nombre = nombre % 4;`
`++`	`compteur++;`	`compteur = compteur + 1;` *ou* `compteur += 1;`
`--`	`nbFois--;`	`nbFois = nbFois - 1;` *ou* `nbFois -= 1;`
Dans le cas des deux derniers opérateurs illustrés, leur exécution se produit une fois qu'on a terminé d'utiliser la variable dans l'expression courante. Ainsi, pour l'expression `total = nombre++ - 5`, la valeur de nombre sera augmentée de 1 seulement après que le résultat de l'expression `nombre - 5` ait été affecté à la variable `total`.		

Exercices :

Indiquez quelles sont les valeurs des variables entières **a**, **b** et **c** après l'exécution de chacune des instructions suivantes : *(les variables conservent leur valeur d'une ligne à l'autre)*

	instruction	valeur de a	valeur de b	valeur de c
a	`a = 5;`			
b	`b = 3;`			
c	`c = a + b++;`			
d	`a--;`			
e	`c -= 10;`			
f	`a += b;`			
g	`a /= b--;`			
h	`a = b++ + c;`			
i	`b *= c--;`			
j	`a /= 2;`			
k	`b %= c;`			
l	`a = b = c;`			

Expressions logiques et instructions de sélection

Dans certains programmes, on a besoin de prendre des décisions lors du traitement des données, pour cela on a besoin d'exécuter une action ou un groupe d'actions sous certaines conditions.
Une condition peut être exprimée sous forme d'une expression logique qui peut prendre la valeur vraie ou fausse (selon que la condition est vérifiée ou pas).

1. La notion d'expression logique

La plupart des programmes réalisent des calculs arithmétiques ou logiques.

Une expression logique est la combinaison d'opérateurs relationnels et/ou d'opérateurs logiques et d'opérandes dont le résultat est une valeur logique (ou booléenne).

Une valeur logique représente une vérité (**vrai** ou **faux**) à un instant donné.

OPÉRATEURS RELATIONNELS (DE COMPARAISON) du langage Java

opérateur en Java	description	exemple	valeur si x = 7 et y = 2
`==`	égal	`x == y`	faux
`!=`	différent	`x != y`	vrai
`>`	supérieur	`x > y`	vrai
`>=`	supérieur ou égal	`x >= y`	vrai
`<`	inférieur	`x < y`	faux
`<=`	inférieur ou égal	`x <= y`	faux

Exercice :

Évaluez les expressions logiques ci-dessous en tenant compte des tables de vérité et des priorités des opérateurs :

	expression	résultat
a	`(12 > 8)`	
b	`(14 < 12)`	
c	`3 <= 3`	
d	`(3 == 3)`	
e	`(12 >= 8)`	
f	`14 != 12`	
g	`(3 != 3.0)`	

OPÉRATEURS LOGIQUES

Tables de vérité des différents opérateurs logiques

L'opérateur logique ET :

condition 1	condition 2	condition 1 ET condition 2
faux	faux	faux
faux	vrai	faux
vrai	faux	faux
vrai	vrai	vrai

L'opérateur logique NON :

condition 1	NON condition 1
faux	vrai
vrai	faux

L'opérateur logique OU :

condition 1	condition 2	condition 1 OU condition 2
faux	faux	faux
faux	vrai	vrai
vrai	faux	vrai
vrai	vrai	vrai

Les opérateurs logiques en Java :

description	notation en Java
ET	`&&`
OU	`||`
NON	`!`

Voir la table de priorité des opérateurs en annexe.

Exercice :

Sachant que `a = 2`, `b = 3`, `c = '5'` et `d = 3`, indiquez si chacune des expressions conditionnelles suivantes est vraie, fausse ou invalide pour le compilateur Java :

	expression	vraie	fausse	invalide
a	`a < b`			
b	`d - b == a`			
c	`a <= 2 > b`			
d	`b != d || a <= b`			
e	`b != d || a <= b - 1`			
f	`b == d || a <= b + 1`			
g	`(b != d || a <= b) + 1`			
i	`d >= b / a + 1.0`			
j	`c == '5'`			
k	`a = 0`			

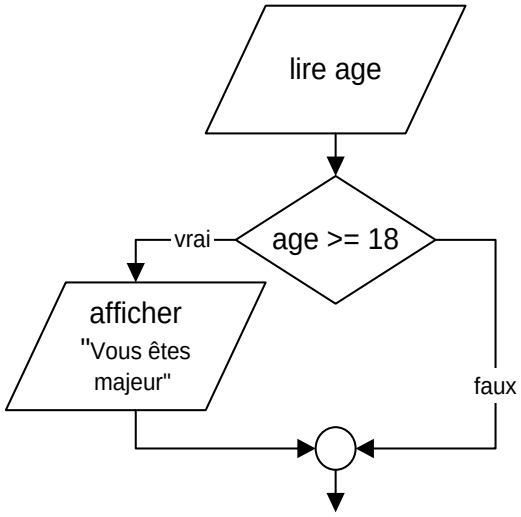
1. La structure de sélection if

La structure de sélection *simple* permet de sélectionner une action ou un groupe d'actions.

Soit l'algorithme suivant :

```
lire age
si (age >= 18)
      afficher "Vous êtes majeur"
fin si
```

Ordinogramme :



Étant donné le tableau des valeurs pour ***age***, indiquez la valeur qui sera affichée :

exécution numéro	***age***	**résultat affiché**
1	12	
2	14	
3	18	
4	21	
5	25	

Écrivez l'algorithme qui permet de lire la note d'un étudiant (variable ***noteEtudiant***). Si la note est inférieure à 60, afficher le message "**Échec**".

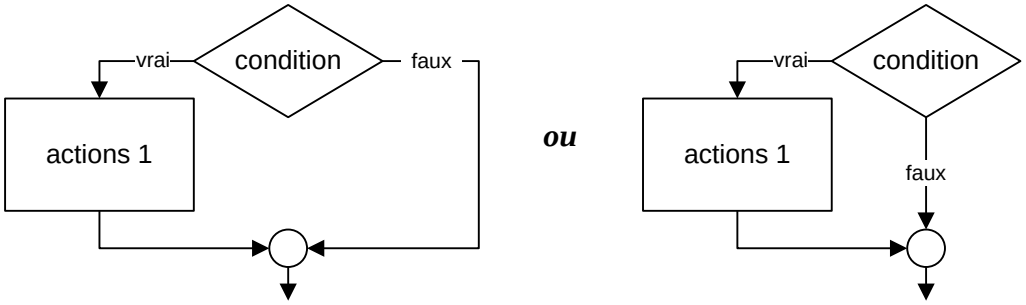
Algorithme :

Étant donné le tableau des valeurs pour ***noteEtudiant***, indiquez le message qui sera affiché :

exécution numéro	***noteEtudiant***	**résultat affiché**
1	56	
2	61	
3	43	
4	60	

Pour afficher dans la console en JavaScript on fait `console.log` et en Java `System.out.println`

L'instruction if en JavaScript



Syntaxe :

```
if (condition)
{
   actions 1
}
```

Exemple :

On doit écrire l'instruction `if` qui correspond aux énoncés suivants :

Si ***noteEtudiant*** est inférieur à 60, afficher le message "**Échec**" :

```
if (noteEtudiant < 60)
{
   console.log("Échec");
}
```

Si ***salaireNet*** est inférieur à 10000 ou supérieur ou égal à 100000, afficher le message "**Le salaire est hors norme**"

```
if (salaireNet < 10000 || salaireNet >= 100000)
{
   console.log("Le salaire est hors norme");
}
```

Étant donné le tableau des valeurs pour ***salaireNet***, indiquez le message qui sera affiché :

exécution numéro	salaireNet	résultat affiché
1	9900	
2	100000	
3	10000	
4	12000	
5	130000	

Exercice :

Écrivez l'instruction **`if`** qui correspond à chaque énoncé :

1. On veut afficher **`"Femme mariée"`** si **`statut`** est égal à **`'M'`** et si **`sexe`** est égal à **`'F'`**

2. On veut additionner 1 à **`nbPersonnesAdultes`** si **`age`** est situé entre 25 et 65 ans

3. Si la **`categorie`** d'un appel téléphonique est **`'L'`** (pour Local), on veut mettre **`tarif`** à 0.45 et afficher **`"Tarif local = 0.45"`**

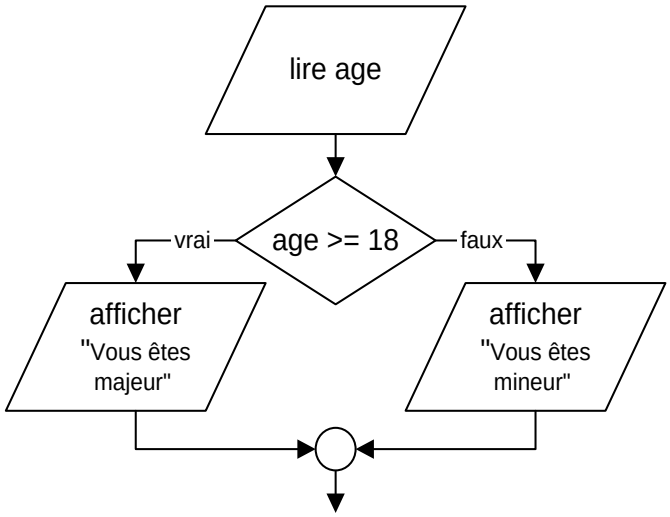
2. La structure de sélection if/else

La structure de sélection *double* permet de choisir entre deux actions ou groupes d'actions.

Soit l'algorithme suivant :

```
lire age
si (age >= 18)
      afficher "Vous êtes majeur"
sinon
      afficher "Vous êtes mineur"
fin si
```

<u>Ordinogramme</u> :



Étant donné le tableau des valeurs pour ***age***, indiquez la valeur qui sera affichée :

exécution numéro	***age***	**résultat affiché**
1	12	
2	14	
3	18	
4	21	
5	25	

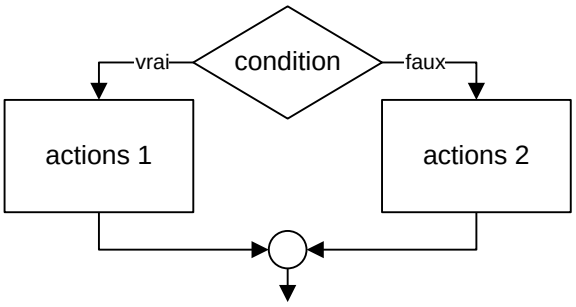
Écrivez l'algorithme qui permet de lire la note d'un étudiant. Si la note (variable ***noteEtudiant***) est inférieure à 60, on doit afficher le message "**Échec**", autrement on doit afficher "**Réussite**".

Algorithme :

Étant donné le tableau des valeurs pour ***noteEtudiant***, indiquez le message qui sera affiché :

exécution numéro	***noteEtudiant***	**résultat affiché**
1	35	
2	75	
3	59	
4	60	

L'instruction de sélection (if…else)



Syntaxe :

```
if (condition)
{
   actions 1
}
else
{
   actions 2
}
```

Exemple :

On doit écrire l'instruction `if` qui correspond à l'énoncé suivant :

Si ***age*** est supérieur ou égal à 18, on doit afficher la phrase **"Vous êtes majeur"**, sinon on doit afficher **"Vous êtes mineur"**

```
if (age >= 18)
{
   console.log("Vous êtes majeur");
}
else
{
   console.log("Vous êtes mineur");
}
```

Remarques :

- Dans une instruction `if`, lorsqu'il n'y a qu'**une seule** instruction dans un bloc entre `{ }`, on peut omettre les `{ }`

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

- Il faut faire bien attention de **ne pas** mettre `;` sur la ligne du if, car celui-ci serait considéré comme terminé et l'instruction ou le bloc d'instructions suivant serait exécuté **dans tous les cas** :

```
if (age >= 18);
   console.log("Vous avez le droit de vote");
```

Exercice :

Écrivez l'instruction **`if`** qui correspond à chaque énoncé :

1. On veut afficher le plus grand de deux nombres **`nbr1`** et **`nbr2`**

2. On veut additionner 1 à **`nbPersonnesAdultes`** si **`age`** est situé entre 25 et 65 ans, sinon on additionne 1 à **`nbPersonnesJeunes`**

3. Si la **`categorie`** d'un appel téléphonique est **`'L'`** (pour Local), mettre **`tarif`** à 0.45 et afficher **`"Tarif local = 0.45"`**, sinon mettre **`tarif`** à 1.53 et afficher **`"Tarif outremer = 1.53"`**

Complétez le programme Java suivant qui permet de lire le salaire net d'un employé. Si le salaire net, ***salaireNet***, est inférieur à 10000 ou supérieur ou égal à 100000, on doit afficher le message "**Le salaire est hors norme**"

```
/*
 * CalculSalaire.js ou .java
 * Ce programme
 *
 * Auteur :
 * Créé le
 */


        // déclaration de vos variables et constantes




        // lecture des données en entrée


        // traitements et affichage des résultats
```

Écrivez le programme qui permet :

- de lire la quantité achetée d'un produit ainsi que le prix unitaire du produit
- de calculer et d'afficher le **coût brut** de l'achat, le montant du **rabais** et le **coût net** de l'achat
- on sait qu'on accorde un rabais de 10% quand on achète plus de 5 articles

Algorithme :

Données en entrée	Données en sortie

Intermédiaires	Constantes
	TAUX_RABAIS = 10% **SEUIL_RABAIS = 5**

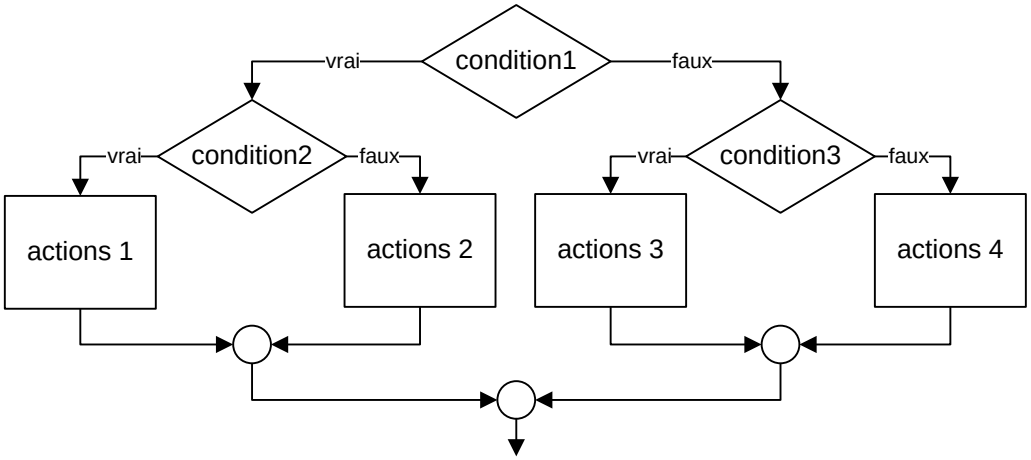
```
/*
 * CalculCout.js ou .java
 * Ce programme lit une quantité et un prix, puis calcule et affiche
 * le coût brut de l'achat, le montant du rabais et le coût net de l'achat
 * Auteur :
 * Créé le
 */

      // déclaration de vos variables et constantes

      // lecture des données en entrée
```

```
        // calculs
```

```
        // affichage des résultats
```

Plusieurs instructions (if…else) imbriquées

Le contenu d'un **if** et d'un **else** peut être un autre if. Dans ce cas, on parle d'instructions **if imbriquées.**



Syntaxe :

```
if (condition1)
{
   if (condition2)
   {
      actions 1
   }
   else
   {
      actions 2
   }
}
else
{
   if (condition3)
   {
      actions 3
   }
   else
   {
      actions 4
   }
}
```

Attention à l'indentation car ça risque d'être illisible ! :

```
if (condition1) {
if (condition2) {
actions 1
}
else {
actions 2
}
else {
if (condition3) {
actions 3
}
else {
actions 4
}
}
```

Exemple :

On doit écrire les instructions qui correspondent à l'énoncé suivant :

Si ***sexe*** est égal à 'F' (pour Femme) et que ***age*** est supérieur ou égal à 18, on doit afficher la phrase **"C'est une femme majeure"**, sinon on doit afficher **"C'est une femme mineure"** Si ce n'est pas une femme et que ***statut*** est égal à 'M' (pour marié), alors on doit additionner 1 à ***nombreHommesMaries*** et afficher la phrase **"C'est un homme marié"**, autrement on doit afficher **"C'est un homme libre"**.

```
if (sexe == 'F')
{
   if (age >= 18)
   {
      console.log("C'est une femme majeure");
   }
   else
   {
      console.log("C'est une femme mineure");
   }
}
else
{
   if (statut == 'M')
   {
      nombreHommesMaries++;
      console.log("C'est un homme marié");
   }
   else
   {
      console.log("C'est un homme libre");
   }
}
```

Remarques :

- ☞ Il est important d'associer les **`else`** avec les bons **`if`** : un **`else`** se rapporte toujours au **`if`** le plus proche.

Complétez le programme Java suivant :

```
/*
 * Meteo.js ou .java
 * Ce programme lit la temperature puis affiche un message :
 *    temperature >= 20 affiche "Il fait chaud"
 *    temperature < 20 et >= 0 affiche "Il ne fait ni chaud ni froid"
 *    temperature < 0 affiche "Il fait froid"
 * Auteur :
 * Créé le
 */

       // déclaration de vos variables et constantes




       // lecture des données en entrée


       // traitements et affichage des résultats
```

3. La structure de sélection switch

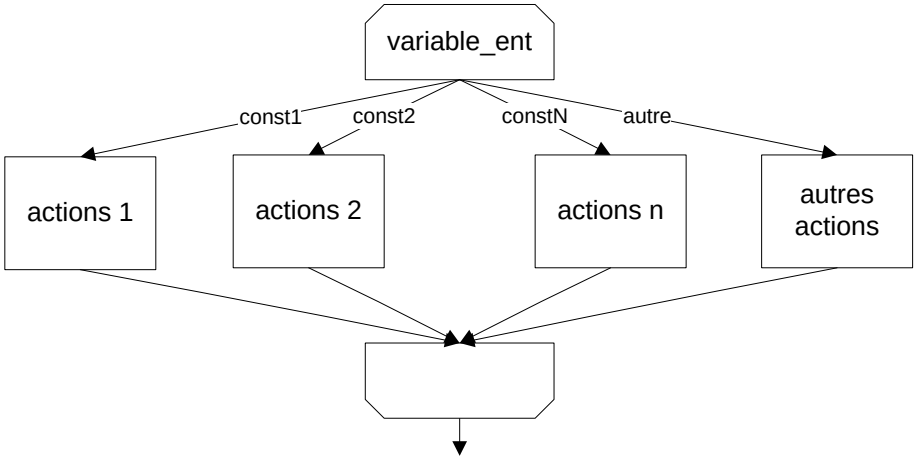
La structure de sélection *multiple* permet d'éviter dans certains cas une cascade d'instructions de sélection imbriquées (ou if imbriqués)

Exemple :

Écrire l'algorithme qui permet de lire le type d'appel téléphonique et d'afficher un message selon le type d'appel.
Si le type saisi vaut **1** alors le programme affiche le message « *appel Local* », sinon s'il vaut **2** alors le programme affiche le message « *appel Interurbain* », sinon s'il vaut **3** alors le programme affiche le message « *appel International* », sinon s'il vaut **4** alors le programme affiche le message « *appel de nuit* ». Autrement le type de l'appel est invalide.

Algorithme :

```
lire typeAppel
selon que typeAppel vaut
    1 :         afficher "appel Local"
    2 :         afficher "appel Interurbain"
    3 :         afficher "appel International"
    4 :         afficher "appel de nuit"
    autre :     afficher "type d'appel invalide"
fin selon que
```



L'instruction switch

Syntaxe :

```
switch (variable_ent)
{
   case const1:
                  actions 1
                  break;

   case const2:
                  actions 2
                  break;

   ...

   case constN:
                  actions N
                  break;

   default:                                  } partie optionnelle
                  autres actions
}
```

`variable-ent` : est une variable (ou une expression

`const1` à `constN` : sont des **constantes**

Fonctionnement :

`variable-ent` est comparée successivement à chaque constante :

si `variable-ent` == `const1`,
alors exécuter `actions 1` puis sortir du switch **(`break`)**

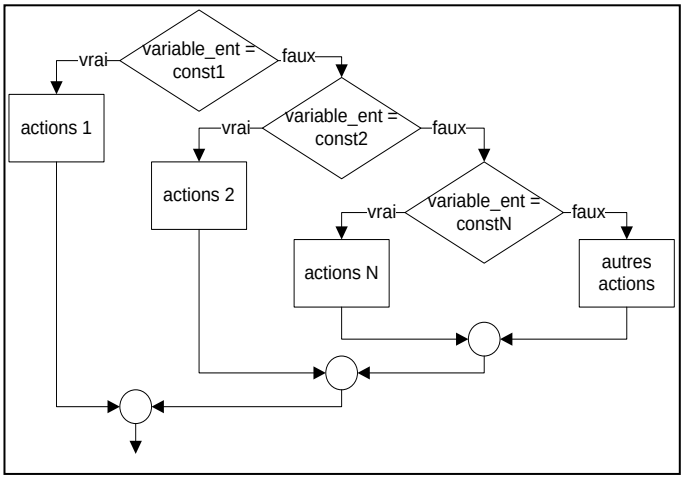
si `variable-ent` == `const2`
alors exécuter `actions 2` puis sortir du switch **(`break`)**

si **aucune égalité** n'est vérifiée,
alors exécuter les instructions de la partie `default`

Attention, s'il n'y a pas de `break` à la fin d'un bloc `actions`, l'exécution se poursuit avec les blocs `actions` suivants jusqu'à la rencontre d'un `break` ou la fin du `switch`!

L'instruction **switch** est équivalente à la forme suivante **:**

```
if (variable_ent == const1)
{
   actions 1
}
else if (variable_ent == const2)
{
   actions 2
}

...

else if (variable_ent == constN)
{
   actions N
}
else
{
   autres actions
}
```



Exemple 1 :

```
var reponse = 'O'; //en JAVA serait char reponse = 'O';
switch (reponse)
{
   case 'O':
      console.log("Vous avez répondu Oui");
      break;

   case 'N':
      console.log("Vous avez répondu Non");
      break;

   default:
      console.log("Votre réponse est incorrecte");
}
```

Exemple 2 :

```
var catEmploye = 'P';  //en JAVA serait char catEmploye = 'P';
switch (catEmploye)
{
   case 'C':      console.log("Catégorie des cadres");
                  salaire = salaire + 500;
                  break;

   case 'P':      console.log("Catégorie des professeurs");
                  salaire = salaire + 300;
                  break;

   case 'S':      console.log("Catégorie des secrétaires");
                  salaire = salaire + 350;
                  break;

   default:       console.log(catEmploye + " est invalide");
}
```

Exemple 3 : Exécuter la même action pour plusieurs cas

```
var noMois = 8; //en JAVA serait int noMois = 8;
switch (noMois)
{
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      console.log("Mois de 31 jours");
      break;

   case 4:
   case 6:
   case 9:
   case 11:
      console.log("Mois de 30 jours");
      break;

   case 2:
      console.log("Mois de 28 ou 29 jours");
      break;

   default:
      console.log(noMois + " n'est pas un mois valide");
}
```

Exemple 4 : Exécuter la même action pour plusieurs cas, comme exemple 3

```
var lettre = 'A'; //en JAVA serait char lettre = 'A';
switch (lettre)
{
   case 'A': case 'E': case 'I': case 'O': case 'U': case 'Y':
      console.log(lettre + " est une voyelle");
      break;

   default:
      console.log(lettre + " n'est pas une voyelle");
}
```

Exemple 5 : Avec des constantes symboliques

```
const PRINTEMPS = 1,//en JAVA déclarer final int PRINTEMPS=1, …
          ETE = 2,
          AUTOMNE = 3,
          HIVER = 4;
var saison = AUTOMNE; //en JAVA int saison = AUTOMNE;

switch (saison)
{
   case PRINTEMPS:
      console.log("Enfin le printemps !");
      break;

   case HIVER:
      console.log("Pas déjà l'hiver !");
      break;

   default:
      console.log("Il fait beau !");
}
```

Le résultat affiché est :

```
Il fait beau !
```

Exemple 6 : Avec une expression à évaluer

```
var nombre = 3;  // en JAVA int nombre = 3;

switch (nombre * 2 + 1)
{
   case 3:
      console.log("Afrique");
      break;

   case 7:
      console.log("Amérique");
      break;

   case 9:
      console.log("Europe");
      break;

   default:
      console.log("Nulle part");
}
```

Le résultat affiché est :

```
Amérique
```

Exemple 7 : Si on omet les **break**...

```
const PRINTEMPS = 1,
          ETE = 2,
          AUTOMNE = 3,
          HIVER = 4:
var saison = AUTOMNE;

switch (saison)
{
   case PRINTEMPS:
      console.log("Enfin le printemps !");

   case HIVER:
      console.log("Pas déjà l'hiver !");

   default:
      console.log("Il fait beau !");
}
```

Dans l'exemple 7, comme il n'y a aucun **break**, l'exécution se poursuit **toujours** jusqu'à la fin du **switch**. La rencontre d'un nouveau **case** ne provoque pas un saut par-dessus les actions de ce **case**.

si saison vaut	phrase(s) affichée(s)
1	Enfin le printemps ! Pas déjà l'hiver ! Il fait beau !
2	Il fait beau !
3	Il fait beau !
4	Pas déjà l'hiver ! Il fait beau !
autre valeur	Il fait beau !

Exercice :

Écrivez l'instruction `switch` qui correspond à chaque énoncé :

1. On veut afficher le message "**Code valide**" si ***codeEntre*** a la valeur 1 ou la valeur 2 ou la valeur 3

2. On veut afficher le message "**Code erroné**" si ***codeEntre*** a une autre valeur que la valeur 1 ou la valeur 2 ou la valeur 3

3. On veut afficher le message "**Meilleur choix**" si ***codeEntre*** a la valeur 1 ou 2, mais on veut quand même afficher le message "**Code erroné**" si ***codeEntre*** a une autre valeur que la valeur 1 ou la valeur 2 ou la valeur 3

Complétez le programme Java suivant :

```
/*
 * Logement.js ou .java
 * Ce programme lit un codeLogement, puis affiche un message :
 *    codeLogement = 'P' affiche "Vous êtes propriétaire"
 *    codeLogement = 'L' affiche "Vous êtes locataire"
 *    codeLogement = 'E' ou 'X' affiche "Vous habitez chez vos parents"
 *    autre valeur de codeLogement affiche "Vous êtes un sans-abri"
 * Auteur :
 * Créé le
 */

      // déclaration de vos variables et constantes
   var codeLogement; //en JAVA char codeLogement; String messageSortie;
   var messageSortie;  // y mettre votre message pour afficher
                       // ce message une seule fois à la fin

      // lecture des données en entrée

      // traitements et affichage des résultats
```

FIN DE LA PARTIE 1