Techniques élémentaires de programmation AS 3.0

1 Les variables
2 Les tableaux (array) ou listes
3 Les structures conditionnelles
4 Les boucles
5 Les fonctions

Avant-propos

Avant de commencer l'apprentissage de ces notions élémentaires, il est important de comprendre qu'un ordinateur n'est capable d'exécuter/effectuer que 4 types d'instructions.
- La manipulation de variables,
- L'écriture/la lecture de données/fichiers,
- Des boucles,
- Des tests.

1 Les variables

Vous constaterez très vite qu'au cours de la programmation de vos différentes animations, vous aurez besoin de mémoriser certaines informations pour les réutiliser ultérieurement. C'est à partir de ce moment là que vous utiliserez les variables.
Une variable n'est donc ni plus ni moins qu'un contenant comparable à une enveloppe, une boite, un tiroir dans le(a)quel(lle) vous stockez des informations de natures différentes (chiffres, lettres). Elle peut être créée très simplement :
En faisant référence à son nom, dans un script, en la déclarant, et en lui affectant un type et éventuellement une valeur. Le nom que vous donnez à une variable ne doit pas contenir de caractères accentués ou d'espaces, ni de caractères spéciaux, il ne doit pas non plus commencer par un chiffre. Essayez de donner un nom représentatif de ce que doit stocker la variable. Ex. : *tauxtva*, *scoregeneral*, *nbrpoints*.
Une variable peut aussi être définie à partir d'une autre variable, comme le montre le script ci-dessous :

```
14  var francs:int = new int() ;
15  francs = euros*6.55957 ;
```

1.1 Créer, utiliser et modifier une variable

Pour créer une variable dans un script
Faites référence à son nom dans un script, précédé du mot réservé var
Faite suivre le nom d'un : puis du mot-clé correspondant à son type
Utilisez le caractère "="
Saisissez une valeur numérique ou alphanumérique (dans ce dernier cas, utilisez des guillemets) ou
La classe de la variable, String() par exemple
Exemples :

```
14  var francs:int = 20 ;
15  var nom:String = "Leclère" ;
16  var datenaissance:Date = new Date() ;
17  var identifiant:String = new String() ;
18  identifiant = prenom.toLowerCase()+"." + nom.toLowerCase() ;
19  trace(identifiant) ;
20
```

Une variable peut contenir de nombreux autres types de valeurs, voici quelques exemples :
- ⇨ boolean pour les variables logiques, vrai ou faux, false ou true
- ⇨ int pour les valeurs numériques de type entier (positif ou négatif)
- ⇨ uint pour les valeurs numériques de type entier sans signe
- ⇨ Number pour les nombres, positifs ou négatifs, à virgule (tant que possible, il faut préférer int, ou uint, à Number, pour des raisons de rapidité de traitement)
- ⇨ String correspond aux chaines de caractères (stockés en interne en Unicode-UTF16)
- ⇨ Object pour les Object de l'affichage, classe Object
- ⇨ Array pour les tableaux de valeurs
- ⇨ Date pour le stockage de valeurs dans ce format (jour mois quantième hh:mm:ss indication GMT année)

Commentaires :
- ⇨ Ligne 14 : Il s'agit là d'une définition et affectation de variable de type int

- ⇨ Ligne 15 : La variable nom, de type String, est définie et affectée de la valeur»Leclère»
- ⇨ Ligne 16: La variable datenaissance est définie à un type Date, n'a pas de valeur affectée, et prends donc la date courante en guise de valeur par défaut.
- ⇨ Ligne 17 : la variable identifiant est créée, avec le type String mais sans valeur indiquée
- ⇨ Ligne 18 : la variable identifiant est affectée de la valeur de la variable prenom, préalablement convertie en minuscules, à laquelle on ajoute (concatène) le signe ., puis la valeur de nom, également en minuscules
- ⇨ Ligne 19 : la fonction trace() se contente de faire afficher, en mode test uniquement, la valeur fournie entre les parenthèses.

Les variables peuvent encore stocker d'autres informations mais nous n'irons pas plus loin pour l'instant. Les objets et les listes reviennent sur cette notion.

1.2 Affecter une valeur à une variable avec la syntaxe conditionnelle if()

Dans un certain nombre de traitements de vos variables, vous effectuerez des tests à la suite desquels vous affecterez une valeur à la variable en fonction du résultat. Par exemple, si l'objet balle_mc a une position horizontale (x) inférieur à 125, la variable position, de type String, prendra la valeur "Gauche", et "Droite" dans le cas contraire. Votre script ressemblera alors peut-être à celui-ci :

```
1  var position:String = new String() ;
2      position = (Object(root).balle_mc.x > 125 ? "droite":"Gauche") ;
3  trace(position) ;
4  trace(Object(root).balle_mc.x) ;
5
```

Si vous devez donc affecter une valeur en fonction du résultat d'un test, vous pouvez utiliser la structure suivante :
nomvariable = (test logique ? valeur si test vrai : valeur si test faux)
Reportez-vous à la partie suivante pour des explications supplémentaires sur les tests.
Si vous voulez tester cette façon de définir une variable, le fichier Exercice_15.fla est à vous.

1.3 Portée d'une variable.

Que doit-on entendre par portée d'une variable ? Comme vous avez pu le lire dans les explications ci-dessus, une variable est représentée par un mot que vous choisissez librement (en respectant les règles de nomenclature). Que se passe-t-il si 2 variables portent alors le même nom ? Et bien tout dépend de l'endroit où cette déclaration est faite.

1.3.1 Variables locales :

Si l'utilisation d'une variable doit se limiter à un bloc de code qui est délimité par une paire d'accolades, vous devez ajouter le préfixe "var" à vos variables. C'est généralement le cas avec les fonctions.

Dans l'exemple ci-dessus, la variable axe ne peut être utilisée en dehors du script qui contient cette ligne d'instruction. Cette utilisation présente l'avantage de pouvoir utiliser ce nom de variable à différents endroits et/ou différents moments de votre animation.

1.3.2 Variables globales :

A l'inverse des variables locales, les variables globales sont valables de n'importe où, c'est à dire à partir de toutes les timelines (des clips et des différentes séquences).
Pour détruire une variable, vous pouvez utiliser la commande `delete`.
Exemple :

➢ Ouvrez le fichier Exercice_16.fla, le code suivant est déjà saisi :

```
1  var position:String = new String() ;
2  position = "globale";
   function definirLocale (){
3  var position:String = new string() ;
4      position = "locale";
5      trace(position) ;
6  }
7  definirLocale();
8  trace(position) ;
```

```
9
```

- Testez cette animation :

La fenêtre de sortie affiche les valeurs : locale puis globale. Normal !

- Modifiez le script ainsi :

```
1  var position:String = new String() ;
2  position = "globale";
   function definirLocale (){
3      var position1:String = new string() ;
4      position1 = "locale";
5      trace(position1) ;
6  }
7  definirLocale();
8  trace(position1) ;
9
```

- Testez votre animation.

Message d'erreur dans la fenêtre de sortie : "
C'est quoi ce truc position1 ?»
En effet position1 n'a d'existence que dans la fonction definirLocale.

- Plus fort, modifiez le code de la façon suivante :

```
1  var position:String = new String() ;
2  position = "globale";
   function definirLocale (){
3      var position1:String = new string() ;
4      position1 = "locale";
5      trace(position) ;
6  }
7  definirLocale();
8  trace(position) ;
9
```

La fenêtre de sortie affiche globale, 2 fois ! Ca marche, position est une variable globale, tandis que position1 est locale, contenu dans la fonction definirLocale().
Compris ?

2 Les tableaux (array) ou listes

Les tableaux fonctionnent sur le même principe. Il est en effet possible d'associer en programmation, grâce aux variables, non plus une seule valeur à un "mot" mais plusieurs. Cela permet donc de stocker plusieurs valeurs dans une seule boîte.
Tous les étudiants d'une même classe font partie d'un même ensemble. On peut alors regrouper leurs noms et/ou prénoms dans une liste. On fait alors référence à un numéro de cette liste pour faire référence à un élève.
Exemples :

```
1  var classe:Array = ["Julie", "Christine", "Aurélia", "Charlotte",
   "Aurélie", "Claire", "Anabelle", "Mélanie"] ;
2  var scores :Array = [350, 450, 321, 457, 345] ;
3  var identite :Array = ["Dupont", "Marie", 23, 178, 75001, "Paris"] ;
4  var eleve1:Array = ["Touille", "Sacha"];
5  var eleve2:Array = ["Zeblouse", "Agathe"];
6  var eleves:Array = [eleve1, eleve2] ;
7
```

Les valeurs d'un tableau peuvent être des chaines de caractères (ligne 1), des valeurs numériques (ligne 2), ou un mélange des deux (ligne 3), voire même des tableaux (ligne 6), on parle alors de tableaux multidimentionnels.
Une liste est donc constituée d'entrées (éléments ou data), séparées par des virgules. Toutes ces valeurs sont impérativement contenues entre crochets et associées à un mot : le tableau. C'est là le point commun avec les variables. Les valeurs d'un tableau sont elles-mêmes saisies entre guillemets s'il s'agit de textes (ex. *classe* de l'exemple ci-dessus) alors que les chiffres ne le sont pas (ex. *scores* ci-dessus). Si vous souhaitez faire référence à des noms de variables ou de tableau dans un tableau, vous ne devez pas saisir ces mots entre guillemets (ex. le tableau

eleves).
Il est très IMPORTANT de savoir que la première entrée d'un tableau porte un numéro d'index égal à 0. Ainsi, pour faire référence à Christine (dans la première liste fournie en exemple), voici la ligne d'instruction qu'il faudrait saisir (nous décidons de stocker le résultat dans une variable appelée *eleve*).

```
8   var eleve:String = new String() ;
9   eleve = classe[1] ;
10  trace(eleve) ;
11  eleve = classe[4] ;
12  trace(eleve) ;
13
```

Les instructions des lignes 4 et 5 permettront de faire afficher : …….. pour ceux qui suivent !

2.1 Créer un tableau.

- Ouvrez le fichier Exercice_16.fla
- Saisissez la ligne d'instruction ci-dessous à la suite du code déjà présent :

```
13  var semaine:Array = ["Lundi", "Mardi", "mercredi", "Jeudi", "Vendredi",
    "Samedi", "Dimanche"] ;
14
```

2.2 Créer un tableau vide.

- Saisissez la ligne d'instruction ci-dessous

```
14  var caddie:Array = new Array() ;
15
```

Dans certains cas, vous aurez besoin de créer un tableau vide, si vous ne savez pas à l'avance quelles seront les valeurs qu'il contiendra.

2.3 Lire l'entrée d'un tableau.

- Saisissez la ligne d'instruction ci-dessous

```
15  var queljour:String = semaine[3] ;
16  trace("queljour = " + queljour) ;
17
```

Lors du test de l'animation, la fenêtre de sortie fera apparaître queljour = …… Bravo !

- Et si vous ajoutez au code ce qui suit :

```
17  var affichage:String = "bonjour " + classe[5] + ", vous allez entrer dans
    la zone reservée. ";
18  trace(affichage) ;
19
```

Est-ce clair ?
Pour lire l'entrée d'un tableau, il suffit donc de faire simplement référence au nom de ce tableau, suivi de 2 crochets qui contiennent le numéro d'index de l'entrée à récupérer. Généralement, le résultat est stocké dans une variable ou utilisé dans une expression.

2.4 Modifier l'entrée d'un tableau.

- Saisissez la ligne d'instruction ci-dessous

```
19  Semaine[2] = "Mercredi" ;
20
```

Dans cet exemple, nous changeons la valeur de la 3ème entrée de la liste (l'index 2), mercredi est remplacé par Mercredi. Et oui, ce n'était pas une faute de saisie ☺

2.5 Ajouter une entrée à un tableau.

- Saisissez la ligne d'instruction ci-dessous

```
20  caddie.push("Gâteaux") ;
21
```

Comme vous pouvez le constater, la technique est très simple car il suffit de faire référence au nom d'un tableau, lui associer le mot **push** (séparé par un point) et la valeur que vous

souhaitez ajouter à votre liste. Remarquez que cette valeur est contenue entre parenthèses qui font parti du mot **push**. C'est ce qu'on appelle une méthode.
La méthode **push()** ajoute une entrée à un tableau
La méthode **sort()** permet de trier les éléments d'un tableau.
La méthode **reverse()** inverse l'ordre des entrées d'un tableau, etc...

2.6 Supprimer l'entrée d'un tableau.

- Saisissez la ligne d'instruction ci-dessous

```
22  semaine.splice(3, 1) ;
23
```

La méthode **splice()** nécessite 2 paramètres. Le premier (dans notre exemple, le 3) indique le numéro d'index de l'entrée du tableau à supprimer, le deuxième indique le nombre d'entrées à supprimer.
En imaginant que le tableau *semaine* soit composé des jours que nous connaissons tous, l'instruction ci-dessus supprimera la 4ème entrée du tableau, c'est à dire l'index 3, soit "Jeudi". La liste semaine ne contient alors plus que 6 entrées et devient alors :
"Lundi","Mardi","Mercredi","Vendredi","Samedi","Dimanche"

2.7 Connaître le nombre d'entrées dans un tableau.

- Saisissez la ligne d'instruction ci-dessous

```
23  trace(semaine.lenght) ;
24
```

Jusqu'à présent, après chaque "mot" placé derrière le nom d'un tableau, il y avait des parenthèses. Il s'agissait en effet de ce qu'on appelle des méthodes. Dans ce cas, **length** est une propriété. Ce "mot" ne fait rien, il n'agit pas, il renseigne simplement sur le nombre d'entrées contenues dans le tableau. Dans notre exemple, nous avons demandé l'affichage du nombre d'entrées dans le tableau semaine (6 donc pour l'instant !).
Cette notion est très importante car vous aurez souvent besoin de parcourir toutes les entrées d'un tableau lorsque vous effectuerez une recherche ou un traitement quelconque sur/dans un tableau. Vous pourrez alors utiliser une boucle, avec l'instruction **for()**, qui fera appel à la propriété **length**.

2.8 Les tableaux linéaires et tableaux de propriétés

Tous les tableaux que nous avons utilisés jusqu'à présent, sont de type linéaire. Un tableau contient alors une série de valeurs séparées par des virgules. Les tableaux associatifs(précédemment appelés tableau de propriétés) fonctionnent différemment.

2.8.1 Les tableaux associatifs

Rappelons qu'une propriété permet d'obtenir une valeur lorsque vous faites référence au préalable à une instance. Prenons l'exemple d'un clip que vous positionnez sur la scène. Vous avez nommé l'occurrence obtenue **leplan**. Pour connaître ou modifier la position de cette occurrence, vous devez faire référence à la propriété **x**.

```
8  endroitDest = leplan_mc.x ;
9  leplan_mc.x = 150 ;
```

Dans la ligne 8, la variable *endroitDest* prend pour valeur la position horizontale de l'occurrence nommée **leplan_mc**. Dans l'instruction de la ligne 9, la position horizontale de l'occurrence intitulée **leplan_mc** est fixée à 150 pixels du bord gauche de la scène. Il est donc possible de lire, ou d'écrire, les propriétés d'un objet.
Un tableau associatif est donc une série de valeurs accompagnées de propriétés.
Dans l'exemple ci-dessous, nous créons un tableau associatif intitulé "départements" qui contient une série de valeurs (le nom des départements qui ne doivent pas être entre guillemets) accompagnées d'une série de propriétés (le nom des régions qui doit être saisi entre guillemets). Un tel tableau se déclare en tant qu'Object et non en tant qu'Array. Les méthodes et propriétés de la classe Array (push() ou length par exemple) ne sont donc pas disponibles pour des tableaux associatifs.

```
8  var departements:Objects = new Object();
9  departements = {Var:"PACA", Manche: "Normandie", Loiret:"Centre",
```

```
    Calvados: "Normandie", Vosges: "Lorraine"} ;
10  trace(departements.Manche) ;
```

Vous n'utiliserez peut-être pas cette structure tous les jours, mais bon, c'est possible…

3 Les structures conditionnelles

Lorsque vous rédigez un programme, vous avez déjà constaté ou vous constaterez très vite qu'il est indispensable d'effectuer des tests. Pour exécuter une action, vous devez souvent commencer par faire un choix. Vous avez plusieurs techniques de contrôle, nous allons nous attarder sur le principal …

3.1 Le test if() : Si … Alors …

Dans un preloader, l'évaluation de la plateforme qui exécute le .swf fait souvent partie des premiers tests.
Avant tout, il est très important de bien comprendre que le test que doit évaluer l'instruction doit renvoyer une valeur de type booléen (true ou false). Ainsi, vous pouvez saisir comme valeur de test, un état qui vous renvoie true ou false. Nous reviendrons sur cette notion un peu plus tard.
Pour commencer, voici à quoi doit ressembler un test :

```
if(test) {
   instruction à exécuter si le test renvoie la valeur true
}
```

Comme vous pouvez le constater, la technique est très simple, utilisez d'ailleurs le raccourci **`Esc + i + f`** afin d'insérer rapidement la structure complète. Le plus difficile pour un novice est de comprendre la composition du test. Commençons par un exemple simple :

- Exécutez, le fichier Exercice_18.swf disponible dans l'archive Exos_les_swfs.zip situé dans \\KAPOUER\SHS\PUB\Flash ou téléchargeable sur le blog.

Comme vous pouvez le constater dans cet exemple, lorsque la balle sort de l'écran (par la droite), elle revient par la gauche et si elle sort par le haut, elle revient par le bas, magique !
Soyons plus précis en essayant de donner une explication plus apte à être traduite en code AS.
Nous devons tester la position d'une occurrence, par rapport au bord droit de la scène (qui correspond à une coordonnée x= 250) et au bord supérieur (y = 0). Nous savons que la scène mesure 250 pixels de largeur, si une occurrence qui se déplace de la gauche vers la droite de la scène dépasse le 260ème pixel (la balle a une largeur de 18 pixels), nous la replaçons à -10 pixels du bord gauche de la scène (juste en dehors de la scène).
Idem pour le déplacement vertical, le test s'effectue sur y, avec comme valeur de référence -10 , la balle est alors renvoyée verticalement en 160

- Ouvrez le fichier **`Exercice_18. fla`**

Le scénario comporte seulement trois calques, et une seule image clé. Le calque Actions comporte une action d'image accrochée dans cette image clé.

- Ouvrez le panneau Actions :

Le code suivant est le seul de l'animation et se charge de l'animation de la balle rouge …

```
1   Object(root).addEventListener(event.ENTER_FRAME, deplace) ;
2   function deplace (evt:Event) {
3       Object(root).laBalle_mc.x += 5 ;
4       Object(root).laBalle_mc.y -- ;
5       if (Object(root).laBalle_mc.y > 260) {
6         Object(root).laBalle_mc.x = -10 ;
7       }
8       if (Object(root).laBalle_mc.x < -10) {
9         Object(root).laBalle_mc.y = 160 ;
10      }
11  }
12
```

Explications du code :

- ⇨ l'instruction laBalle_mc._x += 5 permet de faire avancer l'occurrence intitulée "laBalle" de 5 pixels vers la droite (c'est de l'écriture abrégée, fort utile !).
- ⇨ Ligne 1 : grâce au gestionnaire mis en place sur l'événement `ENTER_FRAME`, de l'animation principale (`Object(root)`), les instructions, placées dans cette fonction, deplace, sont exécutées à chaque changement d'image, donc à raison de 10 fois par seconde.

- ⇨ Ligne 3 : la position de l'occurrence `laBalle_mc` placée dans la séquence courante est repositionnée à 5 pixels, vers la droite, de sa position actuelle (effet de déplacement).
- ⇨ Ligne 4 : la position de l'occurrence `laBalle_mc` placée dans la séquence courante est repositionnée à 1 pixel, vers le haut, de sa position actuelle (effet de déplacement).
- ⇨ Ligne 5 : on teste la nouvelle position horizontale de la balle dans la séquence, l'animation faisant 250 pixels de large et la balle 18 pixels de diamètre.
- ⇨ Ligne 6 : si le test est vrai (balle hors de la fenêtre), elle est replacée à gauche de la fenêtre
- ⇨ Ligne 7 : fin du test
- ⇨ Ligne 8 : on teste la nouvelle position verticale de la balle dans la séquence, l'animation faisant 150 pixels de large et la balle 18 pixels de diamètre.
- ⇨ Ligne 9 : si le test est vrai (balle hors de la fenêtre), elle est replacée en bas de la fenêtre
- ⇨ Ligne 10 : fin du test
- ⇨ Ligne 11 : fin de la fonction

Avant d'aller plus loin, revenons sur le mot "test" qui figure dans tous les scripts ci-dessus. Dans notre premier exemple, l'occurrence d'un clip se déplace de la gauche vers la droite de la scène. A chaque instant, la position de l'occurrence sur la scène varie, change de valeur. La valeur renvoyée par la propriété x est de plus en plus grande. A un moment donné, l'occurrence va se trouver au delà des 260 pixels. Au test (à la question) "Est-ce que la position horizontale (x) de l'occurrence intitulée 'laBalle_mc' dépasse (est plus grand(e) que) le 261ème pixel du bord gauche de la scène ?", la réponse est : "oui, l'occurrence a bien dépassé", "vrai", "oui", "effectivement". Pour être plus précis, l'ordinateur ne peut donner qu'une réponse : vrai, qui se traduit par true en anglais, c'est ce terme que retient un ordinateur en programmation. Un test peut donc évaluer plusieurs valeurs à conditions qu'elles soient de même type (numérique, alphanumérique, booléen) :

```
if (laBalle1_mc.x > 25) {
if (laBalle2_mc.y < 100) {
if (prix > 25) {
if (eleve == "Christine") {
if (reponse) {
```

Dans les deux premiers exemples, laBalle1_mc._x et laBalle2_mc._y renvoie une valeur (celle de la position horizontale de l'occurrence intitulée "laBalle1_mc" pour la première) numérique, comparable donc..
La variable *prix* a obligatoirement été précédemment initialisée (sinon, le test renverra toujours faux et l'action à exécuter ne se fera pas), on peut donc comparer la valeur de *prix* à 25. Si la valeur de la variable *prix* est comprise entre - l'infini et 25, le test sera faux.
Si la chaîne de caractères contenus dans la variable eleve est bien "Christine", le test sera vrai.
Pour le dernier exemple, la variable reponse doit être du type Boolean pour que le test soit réalisable, les instructions seront exécutées si reponse est à true, plus simple à saisir que reponse==true.

3.2 Les opérateurs de comparaisons

Nous déduisons d'après les exemples ci-dessus que nous pouvons rencontrer plusieurs types d'opérateurs pour les tests :
L'égalité : **==** , attention, à ne pas confondre avec =, opérateur d'affectation
La supériorité : **>**
L'infériorité : **<**
L' inégalité : **!=**
L'infériorité et la supériorité, avec égalité de la valeur comparée**<=** et **>=**
Remarque : Lorsque vous devez tester une position comme nous l'avons fait dans le premier exemple, il ne faut pas utiliser la comparaison d'égalité (==). Si la balle avance en effet de 3 pixels en 3 pixels, le test peut s'avérer toujours faux. Voici la liste des valeurs (exprimées en pixels) que renverrait la position de l'occurrence laBalle1_mc (dans le cas où l'occurrence est initialement positionnée à 0 pixel du bord gauche de la scène) : 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, etc. Si nous testons la position de la balle pour savoir si elle est à 25 pixels (sous entendu "a-t-elle dépassé le 25e pixel du bord gauche de la scène", le résultat serait toujours faux. Vous constaterez souvent qu'il est préférable d'utiliser les opérateurs de comparaison <, >, <= et >= plutôt que d'utiliser celui de l'égalité (==).

Nous pouvons comparer des variables avec des valeurs, des variables avec d'autres variables, des valeurs de propriétés avec des variables, des valeurs de propriétés avec d'autres valeurs de propriétés, nous pourrions continuer ainsi longtemps, la liste est longue. Vous devez simplement comprendre que vous comparez uniquement ce qui est comparable. En résumé et pour simplifier, vous pouvez comparer des chiffres avec des chiffres, des lettres avec des lettres mais pas mélanger les deux (comparer des valeurs numériques avec des valeurs alphanumériques).
Exemples :
`laBalle1.x > raquette_mc.x`
`reponse == 12` (A condition que reponse contienne une valeur numérique)
`reponse == "Janvier"` (A condition que reponse contienne une valeur alphanumérique)
`score < bestscore`
`score <= 12` (A condition que score contienne une valeur numérique)
`reponse != "Toto"` (A condition que reponse contienne une valeur alphanumérique)
`laBalle1.visible == true`
`laBalle1.x > 260`

3.3 Pourquoi == et non = ?

Si nous utilisions le test suivant, `prix=25`, il ne s'agit pas d'une comparaison mais d'une assignation, une affectation. Sortie de son contexte, si vous lisiez cette instructions, vous comprendriez que la variable *prix* est réglée à 25. On attribue 25 à la variable *prix*. Il s'agit donc d'une affirmation. Puisqu'il s'agit d'une affirmation, le "faux" test renverrait dans ce cas toujours la valeur `vrai`. Cette erreur est souvent à l'origine de «bug» que vous ne comprenez pas.

3.4 Plusieurs tests à la fois avec && et ||

Jusqu'à présent, nous n'avons vu que le cas ou vous testez une condition. Il est possible de tester plusieurs conditions que vous devez séparer par `&&` ou `||`(caractère obtenu avec **`AltGr`** `+6` sur Pc ou **`Alt + Maj + L`** sur Mac).
Le ET logique : &&
Vous souhaitez tester si la balle de notre exemple ci-dessus a bien dépassé le bord droit de la scène et vérifier également si une variable *leScore* est plus grande que 70. Vous devriez dans ce cas utiliser le script ci-dessous :

```
1  Object(root).addEventListener(event.ENTER_FRAME, deplace) ;
2  var score:int = new int();
3  function deplace (evt:Event) {
4      Object(root).laBalle1_mc.x += 5 ;
5      if (Object(root).laBalle_mc.y > 260 && score>70) {
6        Object(root).laBalle_mc.x = -10 ;
7      }
8  }
```

Si l'une des deux conditions n'est pas remplie, le test renvoie `false`, l'instruction `laBalle1_mc._x=-10` ne sera pas exécutée.

Le OU logique : ||
A présent, si vous souhaitez tester deux conditions, mais qu'au moins une des deux soit vraie, pour que votre action à exécuter se fasse, vous changez la double esperluette (&) par deux barres verticales (||). Ce qui donne :

```
1  Object(root).addEventListener(event.ENTER_FRAME, deplace) ;
2  var score:int = new int();
3  function deplace (evt:Event) {
4      Object(root).laBalle1_mc.x += 5 ;
5      if (Object(root).laBalle_mc.y > 260 || score>70) {
6        Object(root).laBalle_mc.x = -10 ;
7      }
8  }
```

3.5 Qu'est-ce qu'un booléen ? (boolean ou aussi valeur booléenne)

Un booléen est (représenté par) une valeur qui peut-être `true` ou `false`. La structure du `if()` utilise cette valeur, mais elle n'est pas la seule. Vous pouvez vous même attribuer une valeur booléenne à une variable, vous utilisez une valeur booléenne avec certaines propriétés (ex. : `visible`). Une valeur booléenne est donc très "pratique" car elle est simple à manipuler.

- Exécutez le fichier Exercice_19.swf.
- Ouvrez le fichier Exercice_19.fla et jetez un œil sur le code.

```
1  Object(root).addEventListener(event.ENTER_FRAME, deplace) ;
2  function deplace (evt:Event) {
3      Object(root).laBalle_mc.x += 5 ;
4      Object(root).laBalle_mc.y -- ;
5      if (Object(root).laBalle_mc.y > 260) {
6        Object(root).laBalle_mc.x = -10 ;
7        Object(root).message_txt.visible = false ;
8      }
9      if (Object(root).laBalle_mc.x < -10) {
10       Object(root).laBalle_mc.y = 160 ;
11     }
12 }
13
```

L'instruction insérée en ligne 7 fait appel à une valeur booléenne : la propriété `visible` de l'occurrence `message_txt`, et la définit à `false`, l'occurrence n'est donc plus visible !

3.6 Bascule logique

Remarque : Il est possible d'inverser la valeur d'un booléen en utilisant le caractère `!`. Cela s'avère très pratique pour faire des "valeurs bascules", c'est à dire des valeurs qui doivent changer d'un évènement à un autre. Prenons l'exemple d'un bouton qui devrait fonctionner sur le principe "on/off" à chaque clic.

- Copiez puis ouvrez le fichier Exercice_20.swf
- Copiez puis ouvrez le fichier Exercice_20.fla

Le script du bouton vert est très simple comme vous pouvez le constater, peut-être même trop simple...

Précisons simplement que la variable intitulée *etatJaune* est préalablement déclarée et initialisée à `false` (valeur par défaut).

```
1  Object(root).btVert_btn.addEventListener(MouseEvent.MOUSE_UP, bascule) ;
2  var etatjaune:Boolean = new Boolean() ;
3  function bascule (evt:MouseEvent) {
4      etatjaune = !etatjaune ;
5      if (etatjaune) {
6        Object(root).btJaune_btn.alpha = 0.20 ;
7      }
8      if ( !etatjaune) {
9        Object(root).btJaune_btn.alpha = 1 ;
10     }
11 }
12
```

Explications :

- ⇨ Ligne 1 : ouverture du gestionnaire du bouton vert, réagissant au relachement (`MOUSE_UP`)
- ⇨ Ligne 2 : initialisation de la variable booléenne etatjaune (false par défaut)
- ⇨ Lignes 3 et 4 : lancement de la fonction qui fait la bascule de la variable etatJaune, booléenne, qui passe de `true` à `false` ou de `false` à `true`
- ⇨ Ligne 5 : test de la valeur de la variable *etatJaune* (*etatJaune* vaut `true` ou `false`)
- ⇨ Ligne 6 : exécutée seulement si *etatJaune* vaut `true`, l'opacité de l'occurrence `btjaune_btn` passe à 20%
- ⇨ Ligne 7 : fin du premier test sur etatJaune
- ⇨ Ligne 8 : test de l'inverse de la valeur de la variable

- ⇨ Ligne 9 : exécutée seulement si *etatJaune* vaut `false`, car dans ce cas *!etatJaune* vaut `true`, l'opacité de l'occurrence `btjaune_btn` passe alors à 100%
- ⇨ Ligne 10 : fermeture du second test
- ⇨ Ligne 11 : fermeture de la fonction bascule()

A re-décortiquer, faire et refaire si vous avez un doute …

3.7 Test avec else : Si … Alors … Sinon …

Jusqu'à présent, vous ne faisiez peut-être que de simples tests qui ne vous permettaient pas de prévoir d'action à exécuter si la condition n'est pas remplie. Il existe une structure du code qui vous permet de prévoir cela. Voyons tout de suite un exemple car vous allez très vite comprendre (si cette première partie sur les tests vous a semblé simple).

- ➢ Exécutez les fichiers Exercice_21.swf et Exercice_22.swf
- ➢ Ouvrez les fichiers Exercice_21.fla et Exercice_22.fla

Script de l'image-clé 1 du fichier Exercice_21.fla :

```
1  Object(root).stop() ;
2  Object(root).btOK_btn.addEventListener(MouseEvent.MOUSE_UP, verifier) ;
3  function verifier (evt:MouseEvent) {
4      if (Obect(root).motdepasse_txt.text == "Zoliv") {
5        Object(root).gotoAndStop(3) ;
6      }
7  Object(root).nextFrame() ;
8  }
9
```

Script de l'image-clé 2 du fichier Exercice_21.fla :

```
1  Object(root).stop() ;
2  Object(root).btEncore_btn.addEventListener(MouseEvent.MOUSE_UP,
   nouvelessai) ;
3  function nouvelessai (evt:MouseEvent) {
4      Object(root).gotoAndStop(3) ;
5  }
6
```

Script de l'image-clé 3 du fichier Exercice_21.fla :

```
1  Object(root).stop() ;
2
```

Script de l'image-clé 1 du fichier Exercice_22.fla :

```
1  Object(root).stop() ;
2  Object(root).btOK_btn.addEventListener(MouseEvent.MOUSE_UP, verifier) ;
3  function verifier (evt:MouseEvent) {
4      if (Obect(root).motdepasse_txt.text == "Zoliv") {
5        Object(root).gotoAndStop(3) ;
6      } else {
7        Object(root).consignes_txt.text = "Mot de passe invalide, recommencez
   ! " ;
8        Object(root).motDePasse_txt.text = "" ;
9      }
10 }
11
```

Script de l'image-clé 2 du fichier Exercice_22.fla :

```
1  Object(root).stop() ;
2
```

Explications : Elles vont être simples. En comparant les scripts ci-dessus, associés aux différentes images-clé, vous pouvez constatez quelques différences :

- ⇨ Apparition d'un nouvel élément du langage : else
- ⇨ Disparition du script lié au bouton btEncore
- ⇨ Disparition des images-clé 2 et de leur contenu
- ⇨ Le texte des consignes, consignes_txt, est modifié par AS
- ⇨ Le mot de passe, invalide, est effacé pour permettre la saisie du nouveau

L'expression `else` est placée à la fin du `if`, elle permet de déclarer les instructions à réaliser pour le cas où le test est faux. On obtient une structure du type :

```
if(test) {
    Instructions à exécuter si le test est vrai ;
} else {
    Instructions à exécuter si le test est faux ;
}
```

Ce qui correspond à un SI test est vrai ALORS on fait ça SINON on fait ça, pratique non ? Cette technique s'avère donc très pratique pour contrôler les 2 cas, la bonne et la mauvaise réponse.

3.8 Test(s) imbriqué(s)

Dans certains cas, il ne sera pas aussi simple de faire un choix (ou un autre). Prenons le cas d'un utilisateur qui saisirait un mot de passe. Nous pourrions tester dans un premier temps si le texte saisi contient plus ou moins de 5 caractères, et si c'est le cas, nous pourrions ensuite tester la valeur de la saisie. Ce qui en ActionScript pourrait se traduire par :

```
1  Object(root).stop() ;
2  Object(root).btOK_btn.addEventListener(MouseEvent.MOUSE_UP, verifier) ;
3  function verifier (evt:MouseEvent) {
4      if (Object(root).motdepasse_txt.length = 5) {
5        if (Object(root).motdepasse_txt.text == "Zoliv") {
6              Object(root).gotoAndStop(3) ;
7        } else {
8        Object(root).consignes_txt.text = "Mot de passe invalide, recommencez
   ! " ;
9        Object(root).motDePasse_txt.text = "" ;
10       }
11     } else {
12       Object(root).consignes_txt.text = "Le mot de passe doit contenir 5
   caractères, recommencez ! " ;
13       Object(root).motDePasse_txt.text = "" ;
14     }
15 }
```

C'est le script présent dans l'image-clé 1 du fichier Exercice_23.fla .
Comme vous pouvez l'observer, la structure principale du test est la suivante :

```
if (test) {
   if(autretest) {
instructions à exécuter si autretest est vrai
 } else {
    instructions à exécuter si autretest est faux
}
} else {
   instructions à exécuter si test est faux
}
```

C'est ce qui s'appelle des tests imbriqués, à vous de garder le fil de vos tests !
Remarque : Si vous bloquez la tête de lecture sur une image-clé afin de demander à l'utilisateur de saisir un login et un mot de passe, pensez à retirer la commande "Lire" du menu contextuel. Vous pouvez également (solution présentée dans les exemples choisis), laisser une image clé vide après celle sur laquelle vous placez ce script (ci-dessus) en ajoutant la commande prevFrame() comme action d'image pour cette image clé. Au cas où l'utilisateur tenterait de sélectionner la commande "Lire", la tête de lecture reviendrait automatiquement sur l'image clé qui contient la demande de mot de passe à renseigner.
Il faut penser à tout !

3.9 Exemples de test

3.9.1 Retour sur la bascule logique

- Ouvrez le fichier Exercice_24.swf
- Ouvrez le fichier Exercice_24.fla

Le script du bouton vert n'a que peu changé, mais pourquoi enchaîner deux tests quand un seul suffit...

```
1  Object(root).btVert_btn.addEventListener(MouseEvent.MOUSE_UP, bascule) ;
2  var etatjaune:Boolean = new Boolean() ;
3  function bascule (evt:MouseEvent) {
4      etatjaune = !etatjaune ;
5      if (etatjaune) {
6        Object(root).btJaune_btn.alpha = 0.20 ;
7      } else {
8        Object(root).btJaune_btn.alpha = 1 ;
9      }
10 }
11
```

Explications :

- Ligne 4 : bascule de la variable etatJaune, booléenne, qui passe de `true` à `false` ou de `false` à `true`
- Ligne 5 : test de la valeur de la variable *etatJaune* (*etatJaune* `vaut true` ou `false`)
- Ligne 6 : exécutée seulement si *etatJaune* vaut `true`, l'opacité de l'occurrence `boutonJaune_btn` passe à 20%
- Ligne 8 : exécutée seulement si *etatJaune* ne vaut pas `true`,donc vaut `false`, l'opacité de l'occurrence `boutonJaune_btn` passe à 100%
- Le bouton se comporte donc comme une bascule, changeant l'état du disque jaune. Mais vous pourriez transformer le scénario, pour que le bouton jaune réalise lui-même ce changement d'état ! Fichier Exercice_24_bis.swf, mais pas le fla !

Prêts pour jouer à «Jour ! –Nuit !» En électrotechnique, on appelle cela un télérupteur !

3.9.2 Test de collision

Une «collision» entre 2 objets est vérifiable, pour peu qu'ils s'agissent de deux occurrences nommées placées sur la scène, bien évidemment sur deux calques différents.

- Exécutez le fichier Exercice_25.swf .

La «raquette» grise se déplace à l'aide de la souris.

- Ouvrez le fichier Exercice_25.fla .

Deux occurrences se trouvent sur la scène, voici leurs noms (d'occurrences) : laBalle_mc et laRaquette_mc. L'occurrence laBalle_mc se déplace seule (ou peut-être rendue mobile), voici le script de vérification d'intersection (collision) :

```
11 var chute:int = 3 ;
12 Object(root).addEventListener(Event.ENTER_FRAME, descente) ;
13 function descente(evt:Event) {
14     Object(root).laBalle_mc.y = Object(root).laBalle_mc.y +chute ;
15     if(object(root).laRaquette_mc.hitTestObject(Object(root).laBalle_mc)) {
16       chute = -3 ;
17     }
18     if(Object(root).laBalle_mc.y < 0) {
19       Object(root).laBalle_mc.y = 0 ;
20       Chute = 3 ;
21     }
22     if (Object(root).laBalle_mc.y > 150) {
```

```
if (Object(root).laBalle_mc.hitTestObject(Object(root).laRaquette_mc)) {
   actions à réaliser si test est vrai ;
}
```

Ce test de collision entre 2 objets renvoie donc une valeur booléenne, et, est donc directement exploitable dans une structure conditionnelle.
Attention, cette méthode ne fonctionne qu'avec des objets d'affichage, MovieClip, sprite …
Décortiquez donc cette petite animation, pour comprendre …

3.9.3 Vérifier si un son a atteint la 15ème seconde …

- Ouvrez le fichier Exercice_26.swf .

Les textes apparaissent en lien avec le temps écoulé depuis le début de la lecture du fichier son. Le test utilisé permet donc de synchroniser l'affichage d'un texte ou d'une image avec un

son. Jetez un œil sur la partie consacrée à l'objet `SoundChannel()` dans l'aide de Flash pour plus de détails.
Evidemment, pour les esprits chagrins, il est possible de faire le même genre de manipulations avec un fichier vidéo.

4 Les boucles

Lorsque vous devez effectuer plusieurs fois la même action avec une petite "variante", vous avez la possibilité de faire un copier-coller de votre ligne d'instruction et d'adapter (changer) ensuite le "bout de code" différent. En voici un exemple :

```
1   Object(root).piece1_mc.visible = true ;
2   Object(root).piece2_mc.visible = true ;
3   Object(root).piece3_mc.visible = true ;
4   Object(root).piece4_mc.visible = true ;
5   Object(root).piece5_mc.visible = true ;
6   Object(root).piece6_mc.visible = true ;
7   Object(root).piece7_mc.visible = true ;
8   Object(root).piece8_mc.visible = true ;
9   Object(root).piece9_mc.visible = true ;
10  Object(root).piece10_mc.visible = true ;
11
```

Cela reste un rien fastidieux !
Visiblement, dans l'exemple ci-dessus, seul le nom de l'occurrence change mais l'action est toujours la même : afficher l'occurrence.
Remarques :
- Avant d'aller plus loin, il est important de rappeler ou préciser que l'exécution d'une boucle est très rapide. Tout dépend du nombre d'itérations (de répétitions). Dans le cas présent, il va suffire d'une fraction de seconde.
- Vous ne devez pas confondre une boucle et une exécution en continue effectuée avec la fonction `setInterval()` ou le gestionnaire de l'événement `ENTER_FRAME`.

Revenons à notre exemple. La structure `for()` utilisée en ActionScript, comme dans la plupart des langages de programmation, est très simple à mettre en place, la seule difficulté réside dans la composition de l'instruction à exécuter. Voyons tout de suite comment adapter le "script" ci-dessus.

```
1   for (var i:int = 1 ; i<=10 ; i++) {
2       Object(root)[ "piece"+i+"_mc"].visible = true ;
3   }
```

Explications : Pour "faire comprendre" à Flash qu'il doit concaténer le mot "piece" et la valeur de i et le texte"_mc", il est nécessaire d'utiliser Object(root)[]. La partie fixe du nom de l'objet doit obligatoirement figurer entre guillemets. Cette méthode est donc idéalement adaptée aux traitements par lots d'objets.

5 Les fonctions

Il s'agit de l'une des notions élémentaires les plus importantes en programmation. En effet, il serait impensable, voire impossible de développer des animations réellement interactives sans utiliser les fonctions. Depuis FlashMX, il est devenu obligatoire d'utiliser les fonctions dans presque tous les cas.

5.1 Qu'est-ce que c'est ?

Vous aurez souvent besoin d'exécuter plusieurs lignes d'instructions de façon répétée, vous aurez alors peut-être le réflexe de copier-coller votre "bout" de code. Il serait tellement plus simple d'apprendre à Flash ce qu'il doit faire lorsqu'on lui "dit" un mot. Prenons l'exemple d'une occurrence qu'il faudrait faire avancer de 3 pixels et tourner de 5 degrés. Voilà ce que nous pourrions "dire" à Flash :
Lorsque tu liras (au moment où Flash lit les scripts d'ActionScript) "avance()", tu devras exécuter les deux lignes d'instructions ci-dessous :

```
laBalle_mc.x+=3
laBalle_mc.rotation+=5
```

Il va de soi que Flash ne comprend pas le langage naturel, c'est donc à nous de transcrire ce souhait en code. Nous devons alors créer une fonction que nous "appellerons" par la suite.

5.2 Créer une fonction

Pour commencer, la première question à se poser est : "où placer notre fonction ? ". Il est conseillé de placer ses fonctions sur la ou les premières image(s)-clé de la timeline (sur un calque réservé aux ActionScripts : un calque nommé Actions, par exemple !). Dans la plupart des langages de programmation, la création d'une fonction passe par l'utilisation d'un mot que vous définissez vous-même (n'en choisissez pas un qui soit déjà utilisé par l'ActionScript (ils s'inscrivent en bleu lors de la saisie dans la fenêtre Actions), consultez la liste des mots réservés dans l'aide en ligne de Flash en cas de doutes). Vous devez ouvrir et fermer une paire de parenthèses (nous verrons plus bas pourquoi). Vous accolez alors, à ces parenthèses, une ou plusieurs lignes d'instructions contenues entre des accolades. Ce qui nous donnerait :

```
1  function avance() {
2      Object(root).laBalle_mc.x += 3;
3      Object(root).laBalle_mc.rotation += 5
4  }
5
```

Ainsi, lorsque vous associerez cette fonction à un évènement, vous obtiendrez le script suivant :

```
7  Object(root).addEventListener(Event:ENTER_FRAME, avance) ;
8
```

Le script ci-dessus permet d'exécuter en boucle les 2 lignes d'instructions contenues dans la fonction, à la cadence de l'animation.
Si nous faisons abstraction de ce dernier exemple, maintenant que nous avons créé notre fonction, comment "faire appel" à elle, c'est à dire comment l'utiliser dans un script ?

5.3 Appel d'une fonction

Comme nous venons de le voir, la création d'une fonction est simple, son appel est tout aussi simple. Alors que vous êtes en train de saisir un script, vous pouvez saisir sur une nouvelle ligne, le nom d'une fonction accompagnée de ses parenthèses.

```
8  temps -- ;
9  Object(root).joueur1_mc.x += 5;
10 Object(root).joueur2_mc.x += 3;
11 avance() ;
12 Object(root).leTemps_txt.text = temps ;
```

La fonction avance() sera exécutée à chaque passage de la tête de lecture.
Notez bien que dans notre exemple, si joueur1_mc et joueur2_mc doivent toujours progresser avec ces mêmes pas (+ 5 pixels et + 3 pixels), il serait là aussi intéressant de créer une fonction.
Mais une fonction peut aussi être appelée par une autre fonction :

```
21 Object(root).btJouer_btn.addEventListener(MouseEvent:MOUSE_UP, joue) ;
22 function joue() {
23     avance() ;
24     recalcule() ;
25     affiche() ;
26 }
```

5.4 Fonction avec paramètres

Si nous reprenons notre exemple, nous pouvons constater que la fonction avance() ne peut servir que dans un cas : déplacer de 3 pixels vers la droite une occurrence intitulée "laBalle_mc", tout en lui faisant subir une rotation de 3°. Et si vous devez dans certains cas accélérer ou ralentir le mouvement, va-t-on être obligé de créer une autre fonction,

avancer2() ? Bien sûr que non, il vous suffit de ne pas préciser de valeurs et les passer en paramètres.

```
1  function avance(vitesse, anglerotation) {
2      Object(root).laBalle_mc.x += vitesse;
3      Object(root).laBalle_mc.rotation += anglerotation
4  }
5
```

Ne paniquez pas, ce n'est pas beaucoup plus compliqué !
Attention, les mots choisis et saisis entre les parenthèses ne doivent pas être présents dans le "vocabulaire" de l'ActionScript 3.0 .
Ils sont employés deux fois : dans un premier temps, à l'intérieur des parenthèses, puis dans les lignes d'instructions contenues dans la fonction.
C'est tout à fait normal car la logique est la suivante.
Lorsque vous ferez appel à la fonction, vous saisirez l'instruction suivante :

```
avance(7,4)
```

Qu'y avait-il à la place du chiffre 7 dans la création de la fonction ? Vous avez raison, le mot "*vitesse*". On appelle cela une variable locale. Cette variable "vitesse" était employée dans l'une des instructions contenues dans la fonction. Cela signifie donc que la valeur 7 doit remplacer la variable "*vitesse*". Cela donne l'exécution suivante :

```
laBalle_mc.x += 7;
laBalle_mc.rotation += 4;
```

La création de fonctions avec paramètres présente donc l'avantage de pouvoir utiliser une fonction avec des valeurs de paramètres différents d'un cas à l'autre, modifiant ainsi le résultat obtenu.

```
avance(7,4)
avance(3,2)
avance(6,1)
```

Dans ces 3 exemples, l'occurrence "laBalle_mc" va avancer et tourner avec des valeurs différentes. "Et moi, j'aimerai que cette fonction puisse s'appliquer à plusieurs occurrences !!!". C'est possible, vous devez dans ce cas définir le nom de l'occurrence comme un paramètre également :

```
1  function avance(nomObject, vitesse, anglerotation) {
2      Object(root).nomObject.x += vitesse;
3      Object(root).nomObject.rotation += anglerotation
4  }
5  avance(laBalle1_mc, 5, 10) ;
6  avance(laBalle2_mc, 2, 1) ;
7  avance(laBalle3_mc, -5, -10) ;
```

Vous découvrirez rapidement que l'utilisation des fonctions permet un gain de temps de développement appréciable …

5.5 Exceptions...

Comme vous venez de le voir, l'appel d'une fonction se fait toujours accompagné d'une paire de parenthèses car il est possible de passer des paramètres. Dans un certain nombre de cas, vous ne devez surtout pas saisir ces parenthèses car elles ne seront pas reconnues et engendreront même un bug dans votre programme. C'est par exemple le cas de `setInterval()` qui nécessite l'appel à une fonction. Dans les Composants, dans la fenêtre des paramètres, il est demandé de saisir le nom d'une fonction. Vous devez uniquement saisir le nom de la fonction sans les parenthèses.
Il existe encore d'autres cas que vous découvrirez vous-même …