# AIDE MÉMOIRE ACTIONSCRIPT 3.0

*par Dominique DOLÉ*

*janvier 2011*

# Table des matières

## 1 Les variables

| Nom | Type | Valeurs | Valeur par défaut |
|---|---|---|---|
| String | Texte | | null |
| Number | Nombre à virgule flottante | - 1.79e$^{308}$ à 1.79e$^{308}$ | NaN (Not a Number) |
| Boolean | Booleen | false et true | false |
| int | Nombre entier positif ou négatif | - 2 147 483 $^{648}$ à + 2 147 483 $^{648}$ | 0 |
| uint | Nombre entier positif<br>Couleurs avec ou sans alpha | 0 à + 2 147 483 $^{648}$<br>0xaarrvvbb & 0xrrvvbb | 0 |
| * | N'importe quel type | Peut changer de type au cours du programme | undefined |

## 2 Les déclarations et conversions

```
var a : String = "1";
var b : String = "2";
var c : Number = 3;
var d : Number = 4;
var e : String;
var f : Number;
```

**e = a + b ⇨ 12**

**f = c + d ⇨ 7**

### 2.1 Conversion d'une chaîne en nombre

**f = Number (a) + Number (b); ⇨ 3**

| Type/valeur d'entrée | Exemple | Valeur renvoyée |
|---|---|---|
| non défini | Number(undefined) | NaN |
| null | Number(null) | 0 |
| true | Number(true) | 1 |
| false | Number(false) | 0 |
| NaN | Number(NaN) | NaN |
| Chaîne vide | Number("") | 0 |
| Chaîne qui peut être convertie en nombre | Number("5") | Nombre (exemple : 5) |
| Chaîne qui ne peut pas être convertie en nombre | Number("5a") | NaN |

### 2.2 Conversion d'un nombre en chaine

**e = String (c) + String (d); ⇨ 34**

| Type/valeur d'entrée | Valeur renvoyée |
|---|---|
| non défini | non défini |
| null | "null" |
| true | "true" |
| false | "false" |
| NaN | "NaN" |
| Chaîne | Chaîne |
| Object | Object.toString() |
| Number | Chaîne représentant le nombre |

### 2.3 Conversion minuscule ⇔ majuscule

- Conversion minuscule ⇨ majuscule

  var a : String = "dominique dolé";

  **a.toUpperCase () ⇨ DOMINIQUE DOLÉ**

- Conversion majuscule ⇨ minuscule

  var a : String = "DOMINIQUE DOLÉ";

  **a.toLowerCase () ⇨ dominique dolé**

## 3 Recherches et extractions de chaînes

nota : dans une chaîne, le 1er caractère à pour position (index) 0

var **a** : String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

### 3.1 Longueur d'une chaîne

- Donne la longueur d'une chaîne.

  **a.length ⇨ 26**

### 3.2 Position d'un caractère dans une chaîne

- Recherche un caractère dans une chaine à partir d'une position de départ

  et donne sa position (index) dans la chaîne.

  **a.indexOf ( chaîne recherchée, position de départ )**

  **a.indexOf ( "E", 0 ) ⇨ 4** **a.indexOf ( "E", 2 ) ⇨ 4**

### 3.3 Recherche du caractère correspondant à une position

- Renvoie le caractère correspondant à la position spécifiée par le paramètre "Index"

  **a.charAt ( Index )**

  **a.charAt ( 4 ) ⇨ E**

  **a.charCodeAt ( 4 ) ⇨ 69** // donne le code ASCII du caractère trouvé

### 3.4 Extraction d'une sous-chaîne

- Renvoie d'une chaîne qui contient le caractère ayant pour position "**IndexDébut**" et tous les autres caractères jusqu'au caractère ayant pour position "**IndexFin**" celui-ci étant exclus.

  **a.substring ( IndexDébut, IndexFin )**

  **a.substring ( 4, 6 ) ⇨ EF**

  **a.substring ( 4 ) ⇨ EFGHIJKLMNOPQRSTUVWXYZ**

- Renvoie d'une chaîne qui contient le caractère ayant pour position "**IndexDébut**" indiquée et tous les autres caractères jusqu'à atteindre la longueur spécifiée par "**Longueur**".

  **a.substr ( IndexDébut, Longueur )**

  **a.substr ( 4, 2 ) ⇨ EF**

  **a.substr ( 4 ) ⇨ EFGHIJKLMNOPQRSTUVWXYZ**

  **a.substr ( 0, 3 ) ⇨ ABC** // Les 3 premiers caractères

  **a.substr ( a.length – 3 ) ⇨ XYZ** // Les 3 derniers caractères

- Renvoie une sous-chaînes à chaque occurrence du paramètre "**Séparateur**" et la range dans un tableau.

```
var a : String = "1234-567-89";
var b : Array = new Array ();
a.split ( Séparateur )
b = a.split ( "-" )
```

**b[0] ⇨ 1234** **b[1] ⇨ 567** **b[2] ⇨ 89**

## 4 Les conditions

Chaque condition que l'on va écrire va retourner un Booléen ( true ou false).

### 4.1 Les opérateurs

- Égalité : ==
- Inégalité : !=
- Supériorité : >
- Supériorité ou égalité : >=
- Infériorité : <
- Infériorité ou égalité : <=
- Est : is ex: if ( maVariable is String)

### 4.2 Les opérateurs logiques

- ET logique : && ex: if ( nombre > 10 && nombre < 100 )
- OU logique : || ex: if ( reponse == solution1 || reponse == solution2 )
- PAS (NOT) : ! ex: if ( !ok ) // inverse la valeur de variable "ok" et vérifie que ça vaut "true"

### 4.3 L'instruction if

```
if ( condition 1)
{
    code exécuté si condition 1 est vrai
}
else if ( condition 2)
{
    code exécuté si condition 2 est vrai
}
else
{
    code exécuté dans tous les autres cas
}
```

### 4.4 L'instruction switch

```
Switch ( Capabilities.language)
{
        case "fr" :
                trace (" La langue est le français") ;
        break;

        case "en" :
                trace (" La langue est l' anglais") ;
        break;

        default :
                trace (" Cette langue n'est pas prise en compte") ;
}
```

- L'instruction "**default**" équivaut à l'instruction "else" (sinon).
- On a la possibilité de prolonger une condition sur une autre en supprimant une instruction "break" :

```
Switch ( Capabilities.language)
{
        case "fr" :
        case "en" :
                trace (" Cette langue est prise en charge") ;
        break;

        default :
                trace (" Cette langue n'est pas prise en charge") ;
}
```

### 4.5 L'opérateur ternaire

Cet opérateur permet de glisser une condition au sein d'une instruction :

```
var maVariable = (condition) ? Code exécuté si vrai : code exécuté si faux ;
var messageUtilisateur:String = (champTexte == bonneReponse) ? "Bravo !" : "Perdu !" ;
```

## 5 Les écouteurs d'événement

### 5.1 Événements souris

```
clip.addEventListener (MouseEvent.CLICK, fonctionEcouteur);
```

| | | |
|---|---|---|
| • CLICK | ⇨ | clic |
| • DOUBLE_CLIC | ⇨ | double clic |
| • MOUSE_DOWN | ⇨ | bouton appuyé |
| • MOUSE_MOVE | ⇨ | déplacement souris |
| • MOUSE_OUT | ⇨ | souris en dehors (+ tous les clips inclus) |
| • MOUSE_OVER | ⇨ | souris au dessus (+ tous les clips inclus) |
| • MOUSE_UP | ⇨ | bouton relâché |
| • MOUSE_WHEEL | ⇨ | molette tournée |
| • ROLL_OUT | ⇨ | souris en dehors (uniquement sur le clip écouté) |
| • ROLL_OVER | ⇨ | souris au dessus (uniquement sur le clip écouté) |

```
function  fonctionEcouteur (evt:MouseEvent):void
{
...
}
```

### 5.2 Événements clavier

```
clip.addEventListener (KeyboardEvent.KEY_DOWN, fonctionEcouteur);
```

| | | |
|---|---|---|
| • KEY_DOWN | ⇨ | à l'appui sur une touche |
| • KEY_UP | ⇨ | au relâchement d'une touche |

```
function  fonctionEcouteur (evt:KeyboardEvent):void
{
    // Si appui sur la touche "Entrée"
    if (evt.keyCode == 13)  // code ASCII de la touche
    {
    ...
    }
    ou
    if (evt.keyCode == Keyboard.ENTER)  // constante associée à la touche
    {
    ...
    }
}
```

## 5.3 Événements timer

```
var monTimer:Timer = new Timer(30000, 4);
```

Timer (delai, nombre de répétitions)

delai = temps entre les répétitions et avant le démarrage, en millisecondes (30000 => 30 s)

nombre de répétitions : s'il vaut zéro (ou absent), l'horloge se répète perpétuellement.

```
monTimer.addEventListener (TimerEvent.TIMER, timerEnCours);
monTimer.addEventListener (TimerEvent.TIMER_COMPLETE, timerTermine);
```

- TIMER ⇨ pendant le déroulement du timer
- TIMER_COMPLETE ⇨ à la fin du timer

```
monTimer.start()
```
Il faut appeler la méthode start() car l'horloge ne démarre pas automatiquement.

Dans cet exemple:

La fonction "timerEnCours" démarre après 30 s et se répète 4 fois.

La fonction "timerTermine" est alors exécutée après 120 s (30 x 4).

```
function  timerEnCours (evt:TimerEvent):void
{
        trace ("Répétition N° ", evt.target.currentCount, "sur ", evt.target.repeatCount);
}

function  timerTermine (evt:TimerEvent):void
{
        trace ("Fin du cycle");
}
```

*Nota :*

- l'événement TIMER_COMPLETE ne remet pas à zéro le nombre de répétitions effectuées. Si on doit relancer le timer, il faudra utiliser la méthode reset() avant de le redémarrer par la méthode start().
- la méthode start() démarre, ou redémarre après stop() en tenant de nombre de répétitions déjà faites.
- la méthode stop() arrête le déroulement en mémorisant le nombre de répétitions déjà faites.
- la méthode reset() arrête l'exécution et remet le nombre de répétitions déjà faites à zéro.

## 5.4 Événement à chaque image

```
addEventListener (Event.ENTER_FRAME, aChaqueImage);
function  aChaqueImage (evt:Event):void
{
        ...
}
```

### 5.5 Identification du symbole source de l'événement

Soit un clip nommé "clip_ext" composé de plusieurs symboles dont un nommé "clip_int". On ajoute un écouteur sur "clip_ext" (clip_ext.addEventListener...)

Si on fait action sur "clip_int" :

- evt.**target**.name ⇨ clip_int ⇨ **celui de l'action**
- evt.**currentTarget**.name ⇨ clip_ext ⇨ **celui de l'écouteur**

## 6 Gestion dynamique des occurrences

### 6.1 Accès à une propriété de symboles sur la scène

On veut remplir chaque zone texte (nommée txt_1, txt_2, ...) avec son suffixe (1, 2, ...)

```
this['txt_' + i].text = i;
```

### 6.2 Accès à une propriété de symboles placé à l'intérieur d'un clip

On veut remplir chaque zone texte (nommée txt_1, txt_2, ...) du clip "clip_1" avec son suffixe (1, 2, ...)

```
clip_1['txt_' + i].text = i;
```

### 6.3 Accès à une occurrence

```
// Initialisation des occurrences en leur attribuant un nom
var clip:MovieClip = new MonClip;
clip.name = 'menu_' + i;
// récupération du nom de l'occurrence
nomMenu = evt.currentTarget.name;
// récupération de l'objet d'affichage par son nom
var objMenu:DisplayObject = getChildByName(nomMenu);
// récupération de la position d'index
numMenu = getChildIndex(objMenu);
// accès à l'occurrence par sa position d'index
getChildAt(numMenu)['txt_' + i].text = i;
```

## 7 Positionnement et apparence

### 7.1 Position de la scène et des objets

```
Stage.align = StageAlign.TOP_LEFT; // Alignement en haut à gauche
Stage.scaleMode = StageScaleMode.NO_SCALE; // Dimensions et positions fixes des objets
```

### 7.2 Curseur

```
clip.buttonMode = true; // Curseur en forme de main
```

### 7.3 Interactivité

```
clip.mouseEnabled = false; // Rendre un objet non interactif
```

## 8 Gestions des couleurs

### 8.1 Définition d'une couleur

```
var rouge:String = "0xff0000";
var orange:String = "0xff9900";
var magenta:String = "0xff0099";
```

### 8.2 Application d'une couleur

- Pour un champ texte :

```
txt_info.textColor = rouge; // couleur du texte
txt_info.borderColor = orange; // couleur de la bordure
txt_info.border = true; // affichage de la bordure
```

- Pour une occurrence :

```
var ct:ColorTransform = new ColorTransform(); // Création de l'objet ColorTransform
ct.color = magenta; // couleur à appliquer
clip_1.transform.colorTransform = ct; // application de la couleur
```

## 9 Liens

### 9.1 Chargement d'un fichier

```
var lienVers:String = "img/photo1.jpg";
var adresseLien:URLRequest = new URLRequest (lienVers);
var chargeur:Loader = new Loader();
chargeur.load (adresseLien);
addChild (chargeur);
```

### 9.2 Navigation HTML

```
var lienVers:String = "http://www.domi71fr.free.fr";
var adresseLien:URLRequest = new URLRequest (lienVers);
navigateToURL (adresseLien, "_blank");
```

## 10 Exploitation d'un fichier XML

### 10.1 Chargement

Soit le fichier XML :

```xml
<?xml version="1.0" encoding="utf-8" ?>

<photos>

    <photo id="1">
        <url>img/photo1.jpg</url>
        <titre>Titre de la photo 1</titre>
        <desc>Description de la photo 1</desc>
    </photo>

    <photo id="2">
        <url>img/photo2.jpg</url>
        <titre>Titre de la photo 2</titre>
        <desc>Description de la photo 2</desc>
    </photo>

    <photo id="3">
        <url>img/photo3.jpg</url>
        <titre>Titre de la photo 3</titre>
        <desc>Description de la photo 3</desc>
    </photo>

</photos>
```

```
// Déclaration et chargement du fichier XML
var chargeurXML:URLLoader = new URLLoader (new URLRequest("xml/photos.xml"));
// Écouteur d'événement : chargement du fichier XML terminé
chargeurXML.addEventListener (Event.COMPLETE, xmlCharge);
```

### *10.2 Traitement*

```
function xmlCharge(evt:Event):void
{
    var xmlClasse:XML = new XML (chargeurXML.data); // Chargement des données
    var xmlListe:XMLList = xmlClasse.photo; // Construction de la liste suivant le nom du noeud
    for each (var xmlElement:XML in  xmlListe)
    {
        // Récupération des données (url – titre – desc) de chaque élément (photo)
        var adresseImage:String = xmlElement.url;
        var titreImage:String = xmlElement.titre;
        var descriptionImage:String = xmlElement.desc;
    }
}
```

### *10.3 Exploitation*

```
    var nbIcones:uint = xmlClasse.elements().length(); // Nombre de nœuds parents (photo) => 3
    var nbIcones:uint = xmlListe.elements().length(); // Nombre d'éléments => 9
```

## *10.4 Recherche et filtre*

- Recherche sur la valeur d'un élément :

  On recherche l'élément pour lequel **url = img/photo1.jpg**

  trace (xmlClasse.photo. (**url == "img/photo1.jpg"**));

```
<photo id="1">
  <url>img/photo1.jpg</url>
  <titre>Titre de la photo 1</titre>
  <desc>Description de la photo 1</desc>
</photo>
```

- Recherche sur l'attribut d'un élément :

  On recherche l'élément pour lequel **l'attribut id = "1"**

  trace (xmlClasse.photo. (**@id == "1"**)); // noter l'@ devant le nom de l'attribut

```
<photo id="1">
  <url>img/photo1.jpg</url>
  <titre>Titre de la photo 1</titre>
  <desc>Description de la photo 1</desc>
</photo>
```

- Filtre sur l'élément d'un élément :

  On recherche le sous élément **titre** de l'élément pour lequel **l'attribut id = "1"**

  trace (xmlClasse.photo. (**@id == "1"**))**.titre** ;

  Titre de la photo 1

## *10.5 Code HTML dans le fichier XML*

Si on souhaite utiliser du code HTML dans les éléments,

on doit encadrer le contenu de l'élément par une balise "**CDATA**" dont la syntaxe est :

![CDATA[ … ]]

```
<photo id="1">
    <url>img/photo1.jpg</url>
    <titre><![CDATA[Titre de la <b>photo 1</b>]]></titre>
    <desc>Description de la photo 1</desc>
</photo>
```