

# Tutorat Flash

Novembre 2008

**UFR STGI** | **Département Multimédia**  
4 Place Tharradin – BP 71427 | M. Benoit DELDICQUE  
25211 Montbéliard CEDEX | Tutorat Flash-ActionScript

## SOMMAIRE

Principes de base .....	2
Terminologie .....	2
Tracer un message .....	3
Opérateurs mathématiques .....	3
Egalités et comparaisons .....	4
Actions conditionnelles .....	5
Boucles .....	6
Construction d'une classe .....	7
Organisation des dossiers.....	7
Modificateurs d'accès .....	8
Syntaxe de base .....	9
Utilisation d'autres classes dans une classe .....	10

## Principes de base

### Terminologie

#### Variables

Les variables sont des emplacements commodes pour stocker des données dans le code. On peut les nommer à notre guise à condition que leur nom ne soit pas déjà réservé par ActionScript et qu'il débute par une lettre, un souligné (touche `_`) ou un signe dollar (mais non par un nombre). Vous pouvez affecter une valeur à une variable en utilisant le signe égal (`=`).

#### Propriétés

Une propriété est un attribut d'objet qui peut être lu et/ou défini. Certaines peuvent être testées et définies (propriété `x` d'un objet, par exemple), d'autres peuvent seulement être lues (propriété `length` d'un champ texte, par exemple).

#### Fonctions

Les fonctions sont des blocs de code qui font quelque chose. On peut appeler ou invoquer une fonction par son nom.

#### Méthode

Une méthode est une fonction associée à un objet qui opère sur l'objet. Par exemple, la méthode `replaceSelectedText()` d'un objet champ texte peut être utilisée pour remplacer le texte sélectionné dans le champ.

#### Instructions

Les commandes d'ActionScript sont entrées comme une série d'une ou plusieurs instructions. Une instruction se termine par un point-virgule.

#### Commentaires

Les commentaires sont des notes placées dans le code. Un commentaire d'une seule ligne sera précédé d'un double slash (`//`). Pour effectuer un commentaire sur plusieurs lignes, la note doit débiter par un slash étoile (`/*`) et terminer par un étoile slash (`*/`).

## Tracer un message

Utilisez la fonction `trace()` pour obtenir la valeur d'une variable ou n'importe quelles autres données. Un trace peut contenir du texte et des variables. On utilise le signe plus (+) pour combiner les valeurs à afficher.

```
trace("Mon nom est " + nomUser + ".") ;
```

## Opérateurs mathématiques

Utilisez les opérateurs d'affectation composites pour modifier une variable ou un propriété par incréments. Plusieurs syntaxes existent pour une même opération.

- Ajouter 6 à une valeur

```
quantite = quantite + 6 ; OU quantite += 6 ;
```

- Soustraire 6 à une valeur

```
quantite = quantite - 6 ; OU quantite -= 6 ;
```

- Multiplier par 6 une valeur

```
quantite = quantite * 6 ; OU quantite *= 6 ;
```

- Diviser par 6 une valeur

```
quantite = quantite / 6 ; OU quantite /= 6 ;
```

Il existe quelques opérations pour l'ajout ou la soustraction d'une valeur avec le chiffre 1.

- `quantite++` ; ajoute 1 à `quantite`
- `quantite--` ; retranche 1 à `quantite`

## Egalités et comparaisons

Employez l'opérateur d'égalité (ou d'inégalité) ou l'opérateur de stricte égalité (ou de stricte inégalité) pour comparer deux valeurs.

### Égalités

Les expressions d'égalités retournent toujours une valeur booléenne.

- `trace(5 == 6) ; // Affiche false`
- `trace(6 == 6) ; // Affiche true`

Les expressions d'inégalités retournent aussi une valeur booléenne.

- `trace(5 != 6) ; // Affiche true`
- `trace(6 != 6) ; // Affiche false`

Les expressions d'égalités ou d'inégalités strictes retournent une valeur booléenne et tiennent compte du type de la variable testée.

- `trace(6 === 6) ; // Affiche true`
- `trace(6 === "6") ; // Affiche false`
- `trace(6 !== 6) ; // Affiche false`
- `trace(6 !== "6") ; // Affiche true`

### Comparaisons

Naturellement, vous pouvez effectuer des comparaisons en utilisant les opérateurs de comparaison bien connus. Les comparaisons ne portent que sur des valeurs de type `Number`. Par exemple, vous pouvez utiliser les opérateurs `<` et `>` pour vérifier si une valeur est inférieure ou supérieure.

- `trace(5 < 6) ; // Affiche true`
- `trace(5 > 5) ; // Affiche false`

## Actions conditionnelles

Pour effectuer une action uniquement lorsqu'une condition est vraie, utilisez une instruction `if` ou `switch`. Le test d'une condition retourne toujours `true` ou `false`.

Vérifier si le contenu d'une variable est le texte « tortue ».

```
if (nomAnimal == "tortue") {
    // Instructions exécutées si le texte est tortue
} else {
    // Instructions exécutées si le texte est différent de tortue
}
```

Pour exécuter des instructions en fonction de la valeur d'une variable ou si vous si vous avez une condition qui peut avoir plus de deux états utilisez l'instruction `switch`.

```
switch(nomAnimal) {
    case "tortue" :
        // Instructions exécutées si le texte est tortue
    case "colombe" :
        // Instructions exécutées si le texte est colombe
    default :
        // Instructions exécutées si le texte ne correspond pas
}
```

Pour prendre une décision basée sur plusieurs conditions on utilise les opérateurs logiques ET (`&&`), OU (`||`) et NON (`!`) pour créer des instructions conditionnelles composites.

Par exemple, pour souhaiter un anniversaire on teste la date et le mois.

```
var maintenant:Date = new Date() ;
if( (maintenant.getDate() == 17) && (maintenant.getMonth() == 3) ) {
    trace("Joyeux anniversaire") ;
}
```

L'opérateur NON (`!`) est très utile pour tester des valeurs booléennes.

<pre>if(_sprite.visible) {     // Le sprite est visible } else {     // Le sprite est masqué }</pre>		<pre>if(!_sprite.visible) {     // Le sprite est masqué } else {     // Le sprite est visible }</pre>
--	--	---

## Boucles

Pour effectuer une tâche plusieurs fois, utilisez une instruction de boucle comme `for` ou `while`.

### For

La syntaxe d'une boucle `for` comporte cinq parties fondamentales :

- Le mot-clé `for`.
- L'expression d'initialisation, un indice de type `Number` généralement nommé « `i` » et initialisé à 0.
- L'expression de test : égalité ou comparaison.
- L'expression de mise à jour : changement de la valeur de l'expression d'initialisation.
- Le corps de l'instruction.

```
for(var i:int=0; i<1000; i++) {  
    trace(i);  
}
```

### While

La boucle `while` est similaire à la boucle `for` sauf qu'elle est exécutée au minimum une fois.

La syntaxe d'une boucle `while` comporte quatre parties fondamentales :

- Le mot-clé `do`.
- Le corps de l'instruction.
- Le mot clé `while`
- L'expression de test : égalité ou comparaison.

```
do {  
    // Instructions exécutées  
}  
while (nomAnimal != "tortue");
```

## Construction d'une classe

### Organisation des dossiers

Les classes sont des fichiers textes qui sont placés sur le disque dur dans un chemin précis.

La définition d'un chemin d'une classe est en relation directe avec le fichier .fla qui l'utilise.

L'organisation des fichiers d'un projet Flash proposée ainsi est celle qui est la plus utilisée par les développeurs Flash.

Sur votre disque dur vous avez créé un dossier dans lequel est rangé votre fichier .fla. Dans ce même dossier, appelé dossier racine du projet, vous créez un dossier « com » qui contiendra les différents collaborateurs qui ont créé es classes utilisées pour votre projet. Ainsi, un nouveau dossier sera créé dans « com », il portera votre nom ou pseudonyme. C'est dans ce dossier que vous placerez vos fichiers de classe.

## Modificateurs d'accès

Les modificateurs d'accès sont assignables aux variables, aux propriétés et aux fonctions d'une classe. Ils définissent de quelle manière les variables et les fonctions sont accessibles.

### Public

Le modificateur d'accès `public` autorise un accès à une variable, une propriété ou une fonction à être appelés depuis n'importe quelle partie ou classe de votre animation Flash.

### Private

Le modificateur d'accès `private` autorise un accès à une variable, une propriété ou une fonction mais de manière limitée. L'appel à une variable, une propriété ou une fonction pourra être effectué uniquement dans la classe où elle a été définie.

Par convention, les noms de variables privées commencent par un souligné (`_`). Ex : `private var _age:Number;`

### Static

Le mot clé `static` autorise un accès à une variable ou une en lecture seule uniquement. Par exemple, PI peut être définie comme variable statique puisqu'elle ne change pas. L'attribut statique précède le modificateur. Ex : `static public var titre:String = "Mon titre";`

Les variables publiques ou privées sont accessibles en lecture-écriture selon les accès qui leurs sont conférés (`public` ou `private`).

## Syntaxe de base

### Package

Dans ActionScript 3.0, toutes les classes doivent être placées dans des packages. Ces packages contiennent le chemin relatif qui les désigne.

Voici un exemple de syntaxe :

```
package com.benoitdeldicque.essai {  
    // code de la classe  
}
```

Le fichier de cette classe sera donc rangé dans un dossier « essai » contenu dans le dossier parent « benoitdeldicque », lui-même contenu dans son parent « com ». Dans le même dossier que le dossier « com » se situe le fichier .fla.

### Class

Une fois le package correctement déclaré, il faut déclarer la classe. Le nom de la classe doit, par convention, commencer par une majuscule.

Il est important de noter que le fichier ActionScript doit avoir obligatoirement le même nom que votre classe (sensible à la casse).

Voici un exemple de syntaxe pour le fichier Test.as :

```
package com.benoitdeldicque.essai {  
    public class Test {  
        // code de la classe  
    }  
}
```

## Propriétés et méthodes

Lorsque la classe est déclarée, il est nécessaire de déclarer les propriétés et méthodes qui lui sont associées.

Généralement, le nom de la première méthode est similaire au nom de la classe. Cette méthode sera appelée le constructeur de la classe.

Voici un exemple de syntaxe pour le fichier Test.as complété :

```
package com.benoitdeldicque.essai {
    public class Test {
        // déclaration des propriétés
        public var actif:Boolean = true;
        public var age:Number;
        private var _pi:Number = 3.1415;

        // déclaration du constructeur
        public function Test():void {
            // code du constructeur
        }

        // déclaration d'une méthode
        public function getAge():Number {
            // code de la méthode
        }
    }
}
```

## Utilisation d'autres classes dans une classe

Lorsque vous créez vos classes, il peut vous arriver d'avoir besoin d'utiliser d'autres classes développées par vous-même ou par tierces personnes. Il devient alors nécessaire d'importer les classes pour pouvoir les utiliser dans votre classe. L'import n'est pas nécessaire si la classe se trouve dans le même dossier que la classe qui en fait appel.

Dans tous les autres cas, vous devez importer les classes de en utilisant la commande `import`. Ex : `import com.autrepersonne.sonessai.*;`