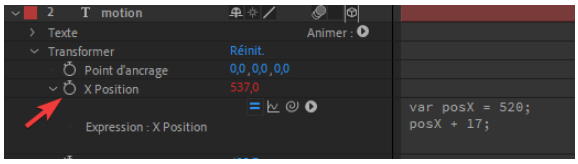
Petit précis des expressions dans After Effects
Les bases

Le mode de rédaction

Les expressions sont des lignes de codes que le logiciel *After Effects* interprète pour effectuer une multitude d'opérations d'animations ou de transformations.
Les expressions sont dérivées du langage JavaScript. Elles sont interprétées ligne par ligne.

Pour entrer dans le mode expression, il faut faire un `alt + clic` ou `option + clic` sur le chronomètre de la propriété sur laquelle on veut appliquer l'expression.



Le point-virgule

Une ligne expression doit se terminer par un point-virgule (`;`). On peut le négliger si l'expression ne comporte qu'une ligne, mais il est à la fois plus prudent et plus clair de prendre l'habitude de toujours insérer un `;` à la fin de toute ligne d'expression

Principe de valeur(s)

Le but de toute expression est de communiquer au paramètre sur laquelle elle est appliquée une valeur numérique.

Par exemple, si l'on applique une expression sur le paramètre `Opacité`, la valeur attendue sera un chiffre.

Si l'on applique une expression sur le paramètre `Échelle`, un couple de valeurs (échelle horizontale, échelle verticale) sera attendu.

Si l'on applique une expression sur le paramètre `Position` d'un calque 3D, c'est un triplé de valeurs qu'il faudra fournir (position horizontale, position verticale, position de profondeur)

Lorsqu'on ne doit fournir qu'une valeur, un simple chiffre suffit. Exemple :

```
Opacité : 100%
```

Pour le cas de deux ou trois valeurs, il faut fournir ce qu'on appelle un tableau de valeurs. Exemples :

`Échelle : [50, 100]` (soit 50% sur l'échelle horizontale, 100% sur l'échelle verticale)

`Position : [960, 540, 0]` (soit position horizontale à 960 pixels, position verticale à 540 pixels, et position en profondeur de 0 pixels)

Notion et types de variables

Une variable est comme une boite portant une étiquette et contenant une valeur. Cela permet à la fois de répéter l'accès à une valeur plusieurs fois, et de faire varier la valeur à l'intérieur de la variable à n'importe quel moment. Exemples :

```
var depart = 540;
```

Attention : on ne spécifie jamais les unités !

Variable numérique

Elle est définie par un simple chiffre :

```
var duree = 12;
```

⚠ Attention à la notation décimale !

Les expressions –comme l'intégralité des langages informatiques– utilisent la notation anglo-saxonne pour les décimales. Ainsi, ce n'est pas la virgule qui sépare les entiers des décimaux, mais le point.
De plus, la virgule est utilisée pour séparer des valeurs au sein de fonctions ou tableaux. Il est très important de ne pas confondre les deux. Exemple :

`var decimaleOK = 16.4` → correspond à la valeur numérique 16,4

`var erreurDecimale = 16,4` → provoquera une erreur dans l'expression

Variable de texte

Elle est définie en mettant le texte entre guillemets :

```
var monTexte = "Hello le monde";
```

Variable booléenne

C'est un type de variable qui ne peux avoir que deux valeurs : 0 et 1. On spécifie cela en utilisant les mots-clé `true` (vrai) et `false` (faux). On pourrait parler de variables «interrupteur».

```
var estActif = false;
```

Les tableaux

Il s'agit de listes de valeurs réunies dans une seule variable. On peut y mettre toute sortes de valeurs, y compris d'autres variables. Chaque valeur est séparée par une virgule :

```
var coordonnees = [960, 540, 0];
```

```
var tableVariables = [x, estActif, monTexte];
```

Un tableau peut avoir plusieurs dimensions. Dans ce cas, il faut insérer un tableau à l'intérieur du tableau :

```
var multiTable = [960, [x, y]];
```

Opérations algébriques

Les expressions permettent d'effectuer des opérations mathématiques, notamment à l'aide des opérateurs suivants :

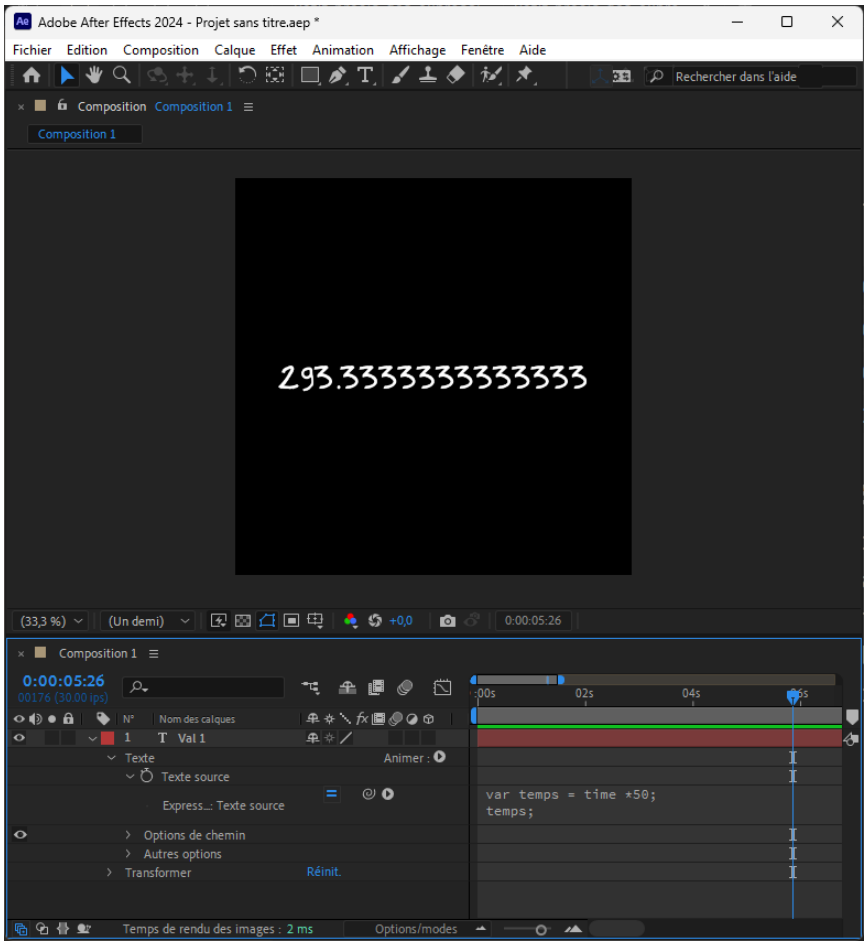
Opérateur	Opération
`+`	Addition
`-`	Soustraction
`*`	Multiplication
`/`	Division
`*-1`	Passe en négatif la valeur

Exemples :

- `position + 50`
- `scale *3`
- `rotation /2`
- `time *-1`

Arrondir les valeurs

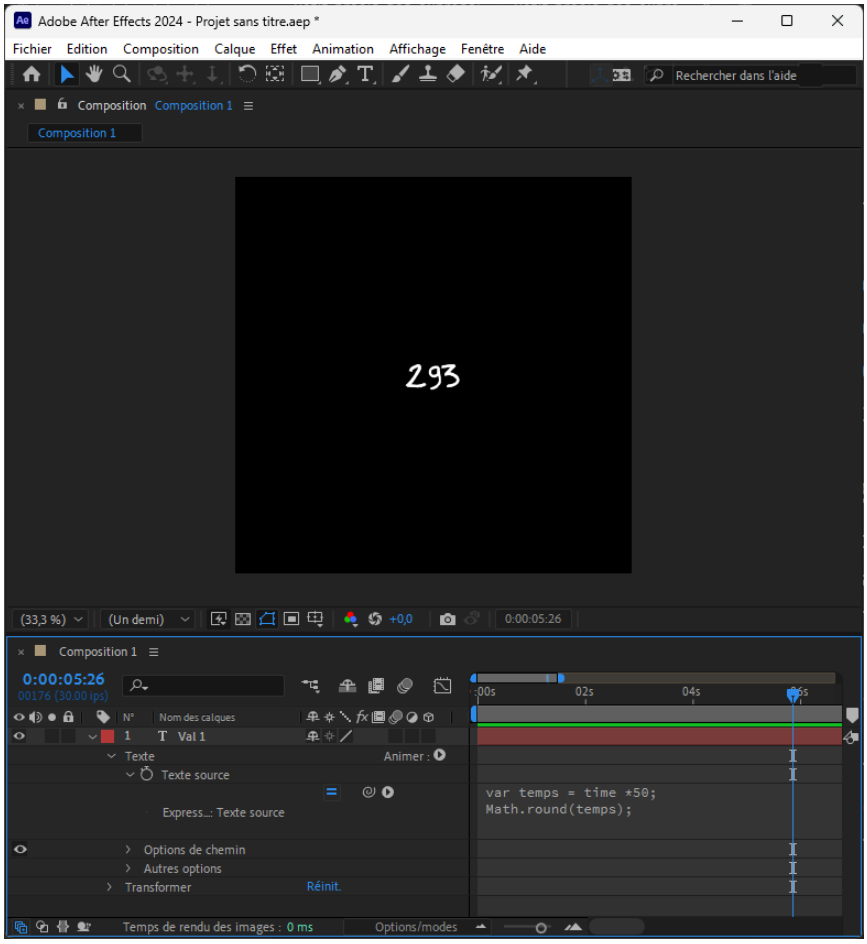
Puisque les calculs sont faits par l'ordinateur, il a tendance à utiliser un nombre impressionnant de décimales.



Il est parfois nécessaires, pour des raisons de clarté ou simplement d'affichage, de réduire ce nombre de décimales. Voici quelques outils pour cela :

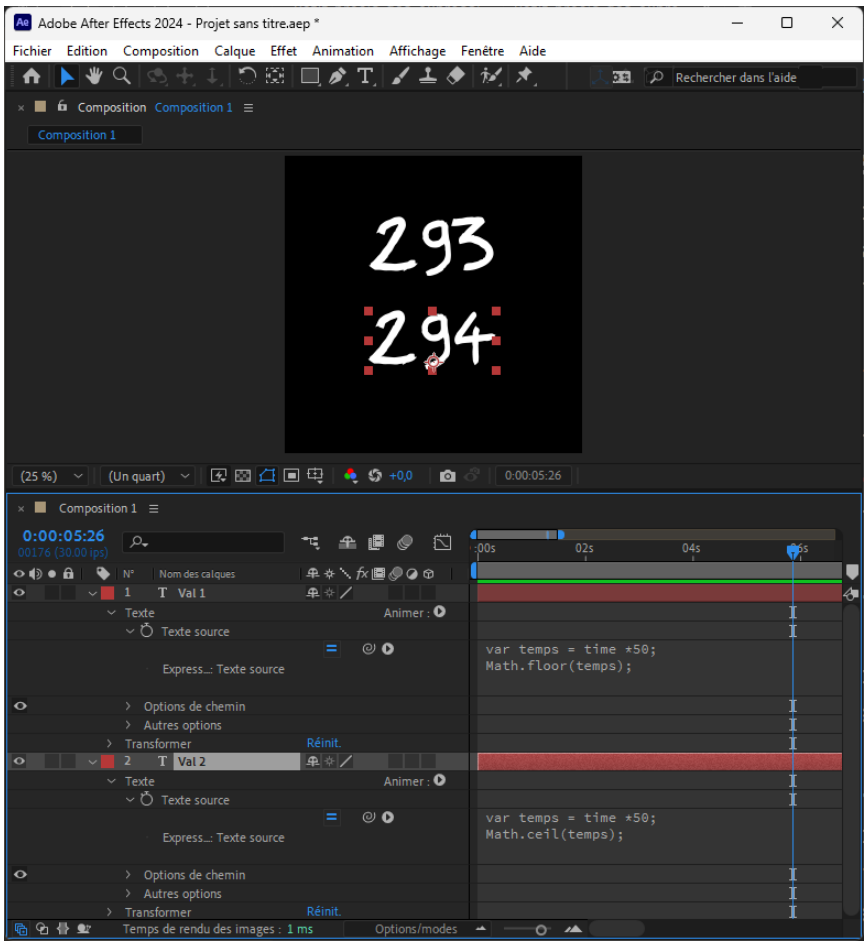
Math.round()

La fonction `Math.round()` est sans doute la plus utilisée. Elle arrondit à l'entier le plus proche.



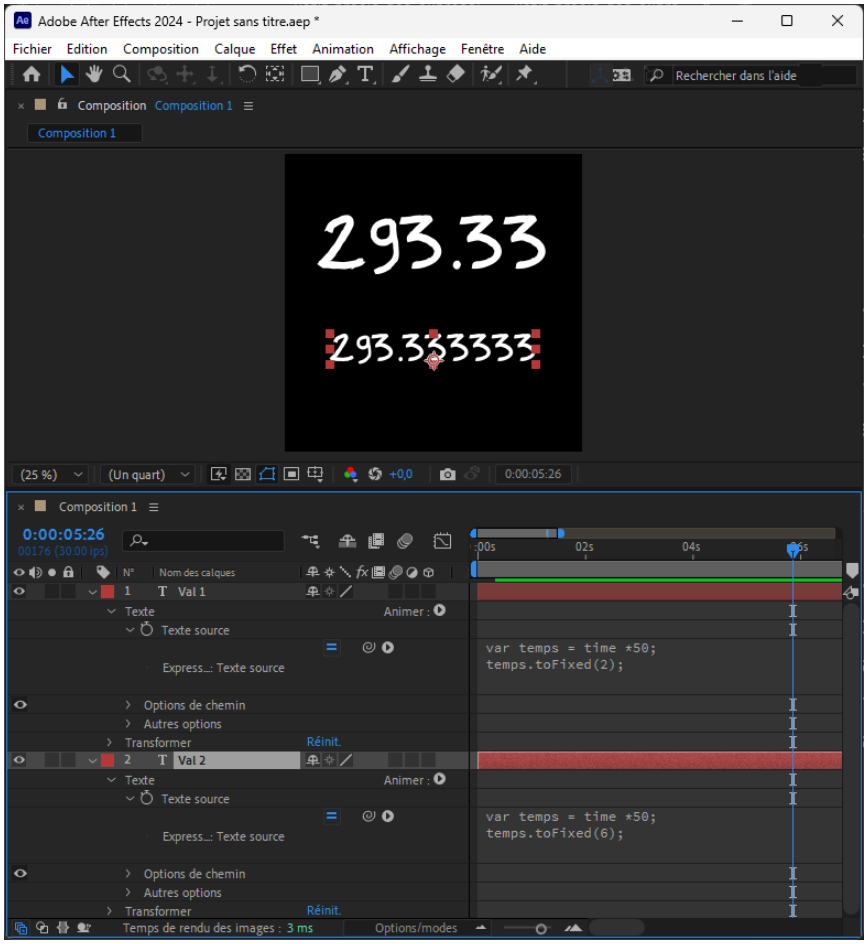
Math.floor() / Math.ceil()

La fonction `Math.floor()` arrondi la valeur à l'entier inférieur. À l'opposé, la fonction `Math.ceil()` arrondi à l'entier supérieur.



toFixed()

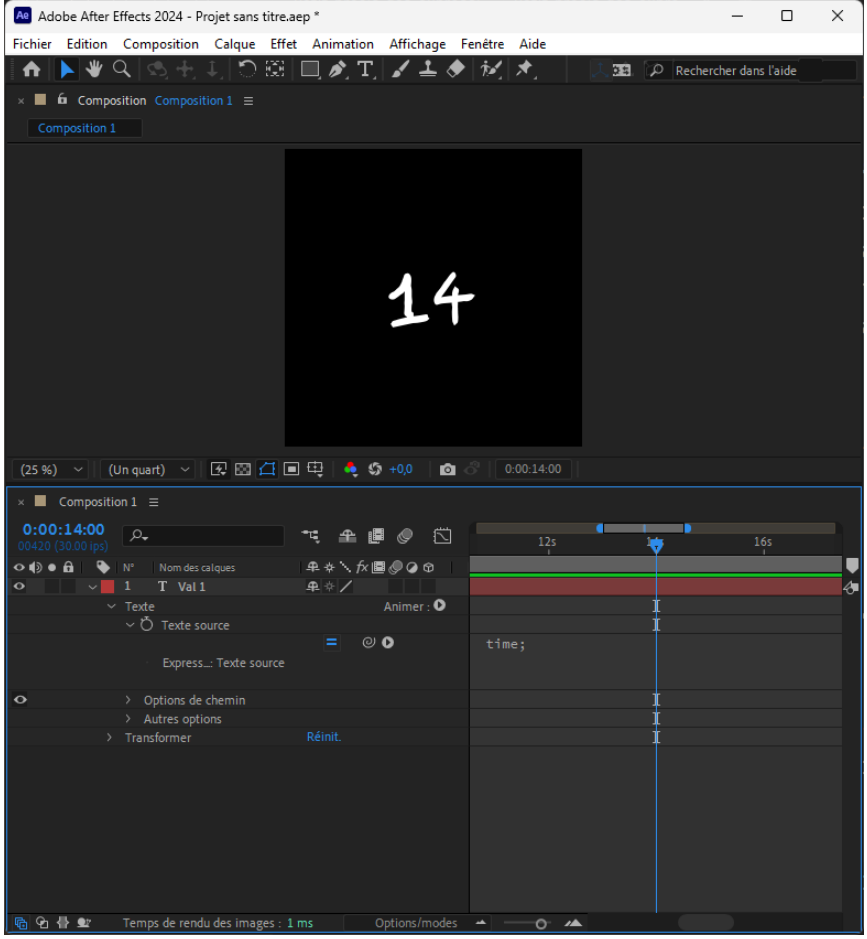
La fonction `.toFixed()` s'utilise à la fin de l'élément de valeur. Elle permet de déterminer combien de décimales on désire utiliser.
Elle attend un nombre à l'intérieur des parenthèses, qui spécifie le nombre de décimales à utiliser.



L'expression `time`

L'expression `time` renvoie comme valeur le nombre de secondes où se trouve la tête de lecture dans la composition. Exemples :

- Si la tête de lecture est au début de la composition, `time` donnera `0`
- Si la tête de lecture est à la 14e seconde, `time` donnera `14`



Si vous utilisez l'expression `time` dans une propriété, la valeur de celle-ci augmente donc de 1 à chaque seconde (en utilisant toutes les décimales intermédiaires).

Pour augmenter sa vitesse, il suffit de la multiplier :

`time *50`

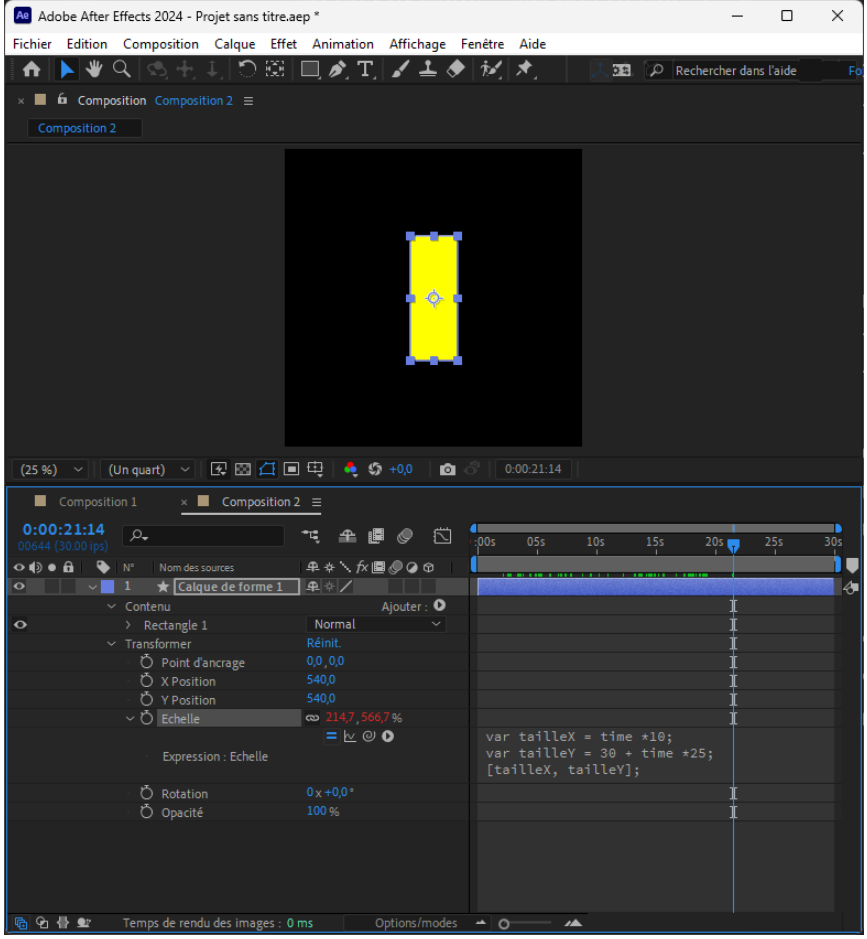
La valeur de `time` correspondant au temps de la composition, sa valeur de départ est à `0`. Si on désire que celle-ci soit autre, il suffit d'ajouter la valeur nécessaire :

`32 + time *50`

La propriété sur laquelle cette expression est appliquée augmentera de 50 unités chaque seconde en commençant à 32.

Si l'on désire appliquer cette expression sur une propriété à plusieurs dimensions, il faudra utiliser des variables. Par exemple, sur l'échelle :

```
var tailleX = time *10;
var tailleX = 30 + time *20;
[tailleX, tailleY]
```



L'expression `wiggle()`

La fonction `wiggle` retourne une valeur aléatoire. Elle possède plusieurs paramètres (à mettre entre ses parenthèses).

Utilisation simple

Au minimum, cette fonction nécessite deux paramètres :

`wiggle (fréquence, amplitude)`

La fréquence définit à quel rythme la valeur aléatoire doit changer. Son unité est la seconde.

L'amplitude définit le maximum de la valeur tirée aléatoirement, à la fois en positif et en négatif. Par exemple, si partant de zéro vous définissez une amplitude de 100, la valeur aléatoire s'étendra de −100 à 100.

Contrairement à `time`, la fonction `wiggle` peut s'appliquer sur n'importe quel paramètre, qu'il ait une seule valeur (opacité, rotation), deux (position, échelle) ou trois (calque 3D).

`wiggle (2, 50);` tirera une valeur aléatoire entre −50 et 50, deux fois par seconde.

`wiggle (0.25, 200);` tirera une valeur aléatoire entre −200 et 200, toutes les 4 secondes.

Si l'on veut appliquer des effets de `wiggle` différents sur une propriété à plusieurs valeurs, il faut utiliser des variables :

```
var aleatX = wiggle(2, 50);
var aleatY = wiggle(4, 300);

[aleatX, aleatY];
```

Si l'on veut appliquer le `wiggle` sur une seule valeur sans modifier l'autre, il suffit de réutiliser la valeur d'origine. Exemple :

```
var aleatX = wiggle(2, 100);
var normY = transform.position[1];

[aleatX, normY];
```

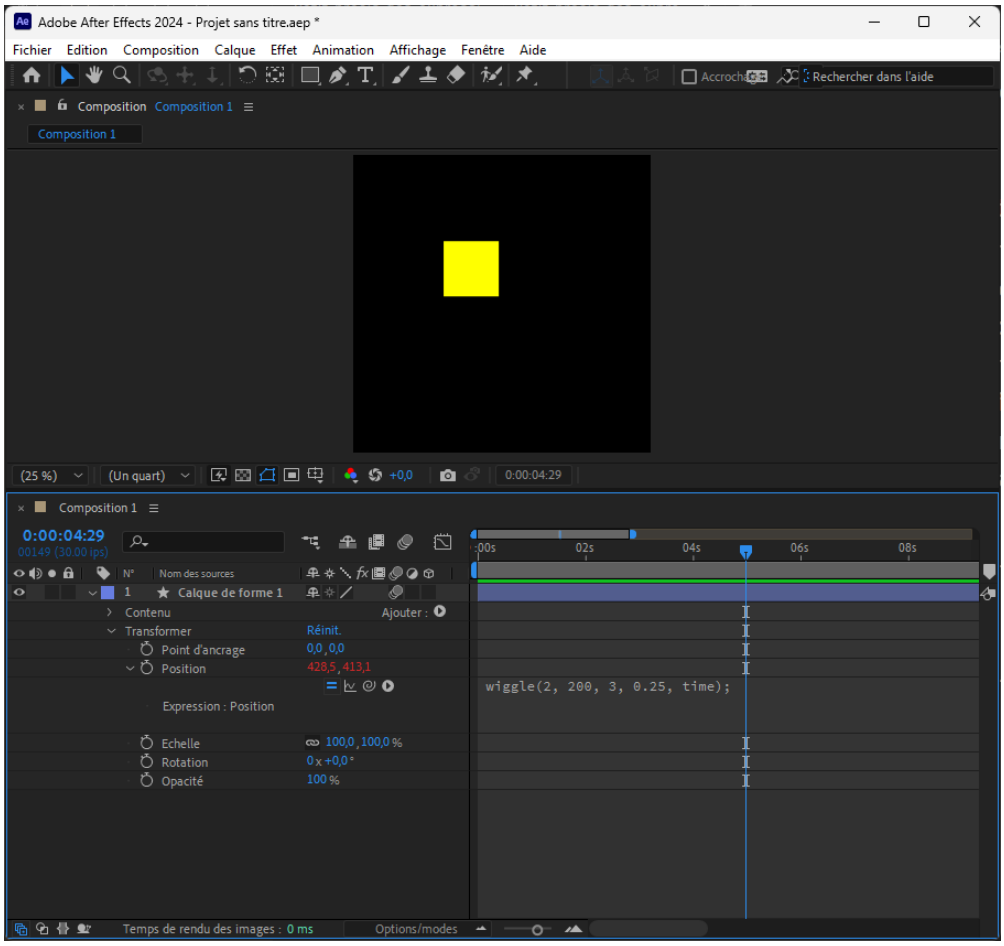
%

Utilisation avancée

La fonction `wiggle` permet d'utiliser plus de paramètres, si besoin. En plus de la fréquence et de l'amplitude, on peut lui fournir également les paramètres suivants :

- Un octave : cela définit le détail du `wiggle`. Plus l'octave est bas, plus le `wiggle` sera doux, et plus l'octave est élevé, plus le `wiggle` sera sec. Par défaut, l'octave a une valeur de 1.
- Un multiplicateur d'amplitude : cela définit par quelle valeur chaque amplitude doit être multipliée. Si ce nombre est positif, chaque amplitude augmentera en force. À contrario, un nombre décimal permettra de décroitre la force de l'amplitude. Par exemple, un multiplicateur de 0.5 fera diminuer de moitié chaque amplitude. Par défaut, le multiplicateur a une valeur de 1.
- Temps : récupère la position temporelle de la composition. Par défaut, le temps a la valeur de l'expression `time`. Si on la change, il est possible de ralentir ou d'accélérer l'effet.

En résumé :
`wiggle(fréquence, amplitude, octave, multiplicateur_amplitude, temps)`



Boucle de `wiggle`

Pour effectuer une boucle de `wiggle`, nous allons l'utiliser conjointement avec une autre expression : `linear` (voir page pour plus d'informations).

Voici un exemple d'expression `wiggle` bouclée :

```
var freq = 0.5;
var ampl = 500;
var nbBoucles = 5;
var temps = time % nbBoucles;

var wiggle1 = wiggle(freq, ampl, 1, 0.5, temps);
var wiggle2 = wiggle(freq, ampl, 1, 0.5, temps-nbBoucles);

linear(temps, 0, nbBoucles, wiggle1, wiggle2);
```

Décortiquons maintenant ce code :

`var freq = 0.5;`
Met en variable une fréquence de 0.5 par seconde (soit 1 tirage toutes les 2 secondes).

`var ampl = 500;`
Met en variable une amplitude de 500 (l'unité n'est pas connue).

`var nbBoucles = 5;`
Met en variable le nombre de fois qu'on veut faire boucler le `wiggle`.

`var temps = time % nbBoucles;`
Permet de savoir à quelle étape des boucles on se trouve. Cette instruction remet la valeur de `temps` à 0 à chaque fois qu'elle atteint la valeur de 5. L'opérateur `%` (appelé *modulo*) permet de récupérer le reste de la division du premier nombre par le second (en savoir plus : lire l'article Wikipedia)

`var wiggle1 = wiggle(freq, ampl, 1, 0.5, temps);`
Ce `wiggle` mis en variable utilise les deux valeurs de fréquence et d'amplitude définis plus haut. Le reste de ses paramètres sont par défaut.

`var wiggle2 = wiggle(freq, ampl, 1, 0.5, temps-nbBoucles);`
Ce `wiggle` mis en variable est identique à celui ci-dessus, à la différence que sa valeur de `temps` est inversée. En d'autres termes, ce `wiggle` fonctionne à l'envers du premier.

`linear(temps, 0, nbBoucles, wiggle1, wiggle2);`
L'expression `linear` permet de *mélanger* les deux `wiggle`. Comme le temps défile de 0 à 5 secondes, la valeur retournée sert de valeur minimum au premier `wiggle` et de maximum au second.

L'expression `posterizeTime()`

Cette expression permet de modifier la cadence d'interpolation du paramètre sur lequel elle est appliquée.

Il existe un effet identique dans la collection d'effets d'After Effects, sous le nom de `postérisation temporelle`. Néanmoins cet effet impacte la totalité des paramètres du calque. L'expression permet d'appliquer cet effet sur un paramètre uniquement.

```
posterizeTime(12);
transform.scale.value;
```

L'interpolation d'animation d'échelle du calque sera animée à 12 images par seconde, quel que soit le réglage de la composition.

```
posterizeTime(8);
wiggle(2, 100);
```

L'effet de `wiggle` sera appliqué avec une cadence d'animation de 8 images par seconde, indépendamment du réglage de la composition.

Les générateurs aléatoires

L'expression `random()`

La fonction `random` retourne un nombre tiré aléatoirement. La fonction attend un paramètre entre parenthèses qui définit la valeur maximum autorisée. Une nouvelle valeur est tirée à chaque image du temps de la composition.

`random(25);`

Une valeur aléatoire sera générée entre 0 et 25.

On peut également fournir à la fonction un second paramètre qui détermine la valeur minimum attendue :

`random(5, 75)`

Une valeur aléatoire sera générée entre 5 et 75.

Il est également possible de générer des liste de nombres aléatoires, en notant les valeurs maximum (et minimum) à l'intérieur de tableaux. Exemple :

`random([25, 50]);`

Cette expression tirera chaque image un couple de valeurs allant pour la première de 0 à 25 et pour la seconde de 0 à 50.

`random([5, 20], [30, 100]);`

Cet exemple permet le tirage d'un couple de valeurs avec les paramètres suivants :

- La première valeur sera tirée entre 5 et 30
- La seconde valeur sera tirée entre 30 et 100

Il est possible de limiter la fréquence des tirages en combinant cette expression avec `posterizeTime`. Exemple :

```
posterizeTime(4);
random(5, 125);
```

Quatre fois par seconde, un tirage aléatoire est effectué et retourne une valeur entre 5 et 125.

L'expression `gaussRandom()`

Cette expression est pratiquement identique à `random` à la différence que les chances de tirer un nombre s'applique à une courbe de Gauss.

L'expression attend deux valeurs (obligatoires); un minimum et un maximum :

```
gaussRandom(50, 200);
```

Le principe est que le générateur aura plus de chance de tirer une valeur dans la zone médiane (autour de 125 dans l'exemple donné ci-dessus) qu'aux alentours des valeurs minimum et maximum.

Le résultat donnera un effet plus naturel que le simple `random`.

L'expression `seedRandom()`

En conjonction avec l'expression `random`, la fonction `seedRandom` permet de forcer le tirage aléatoire sur un schéma répétitif.

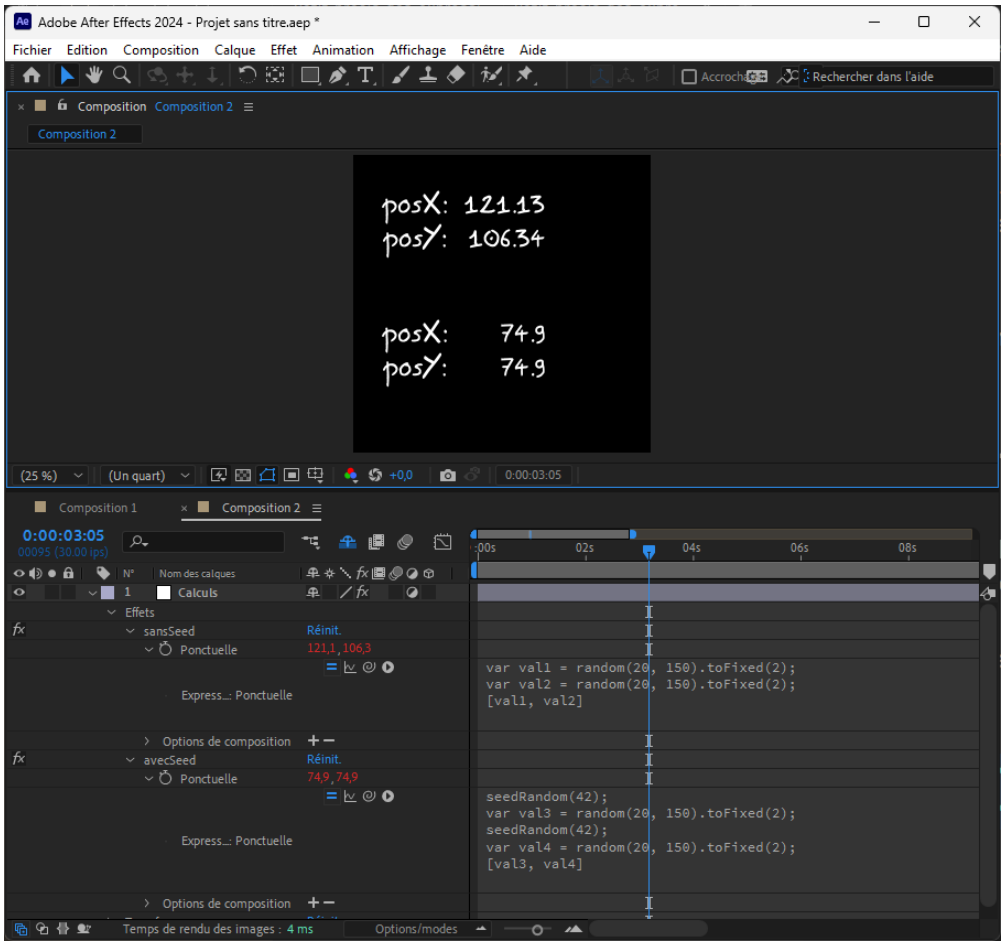
Dans l'exemple suivant :

```
var posX = random(20, 150);
var posY = random(20, 150);

[posX, posY];
```

Bien que les valeurs min et max données au `random` soient identiques, ils vont produire des valeurs différentes. En revanche, dans cet exemple :

```
seedRandom(42);
var posX = random(20, 150);
seedRandom(42);
var posY = random(20, 150);

[posX, posY];
```

Les valeurs fournies aux variables `posX` et `posY` seront strictement identiques.



À l'inverse, fournir un `seedRandom` différent pour chaque tirage garantie un tirage vraiment aléatoire. Le nombre fourni entre parenthèses n'a aucune valeur réelle, l'important c'est qu'il soit unique pour plus d'aléatoire ou répété pour garder des tirages identiques.

Ainsi, il est assez fréquent d'utiliser le numéro des calques comme source pour le `seedRandom` afin de garantir une valeur différente sans avoir à la choisir :

```
seedRandom(index);
random(100);
```

La valeur du mot-clé `index` renvoie au numéro du calque dans la pile de la composition.

On peut aller plus loin avec un deuxième paramètre accepté par la fonction `seedRandom` :

`seedRandom(valeur, timeless=false/true)`

La valeur du paramètre `timeless` est `false` par défaut. Cela signifie que le tirage de `random` sera aléatoire à chaque image de la composition. Mais, s'il est réglé sur `true`, la valeur du tirage sera définie à la première image de la composition et ne changera plus.

Exemples :

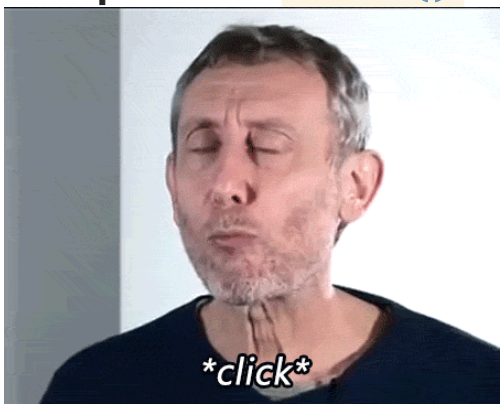
```
seedRandom(42, timeless=true);
random(100);
```

La valeur du `random` sera définie aléatoirement à la première image, et ne changera plus.

```
seedRandom(42, timeless=false);
posterizeTime(2);
random(100);
```

Cette fois, la valeur du `random` changera mais seulement 2 fois par seconde.

L'expression `noise()`



L'expression `noise` est relativement identique à `wiggle`, si ce n'est qu'elle n'utilise pas l'amplitude. Son principe est de générer une valeur comprise entre −1 et 1 en utilisant le bruit de Perlin.

La fonction n'attend qu'un argument qui est un index aléatoire. Exemple :

`noise(time);`

Cela va générer un nombre aléatoire entre −1 et 1 à chaque seconde.

Si l'on veut augmenter l'amplitude, on multiplie la valeur retournée :

`noise(time) *1000;`

Cela va retourner une valeur comprise entre −1000 et 1000 chaque seconde.

```
var posX = 960 + noise(time) *500;
var posY = 540 + noise(time) *500;
[posX, posY];
```

Appliqué sur un calque d'une composition de 1960×1080 (HD), cette expression provoquera un déplacement aléatoire en diagonale.

L'expression `key()`

Il est possible de récupérer et d'utiliser les numéros de points-clé d'une composition à l'aide de deux expressions ; la fonction `key()` qui permet de déplacer la tête de lecture sur un point-clé spécifique, et `numkeys` qui permet de connaitre le nombre total

de points-clé d'une composition.

Exemple d'utilisation :

```
var imgAleat = Math.round(random(1, numkeys));
key(imgAleat);
```

Les expressions `loop`

Il existe deux versions de cette expression permettant de créer des boucles d'animations. La plus utilisée est `loopOut()` qui démarre la boucle à partir du dernier point-clé de l'animation à faire boucler.

Au contraire, `loopIn()` créera la boucle d'animation avant le premier point-clé de l'animation.

Les deux fonctionnant de manière identique, les exemples seront donnés pour l'expression `loopOut()`

L'expression `loopOut()`

L'expression prend deux paramètres :
`loopOut("type", modificateur)`

Types de boucles

Il en existe quatre différents. Chacun définit un comportement spécifique de la boucle. Attention : il faut écrire le type entre guillemets !

`loopOut("cycle")`

C'est le comportement par défaut (on peut donc l'omettre lorsqu'on écrit cette expression).

Lorsque l'animation atteint le dernier point-clé, elle recommence en sautant directement au premier point-clé.

`loopOut("pingpong")`

Ce type d'animation créera une boucle qui revient en arrière lorsqu'elle a atteint le dernier point-clé, pour lire l'animation à rebours jusqu'à atteindre le premier point-clé, et reprendre la lecture dans le bon sens.

`loopOut("continue")`

Ce type d'animation va boucler uniquement sur le dernier point-clé de l'animation, se contentant de continuer indéfiniment l'animation de paramètre impliqué, en gardant sa vélocité et en augmentant sa valeur.

`loopOut("offset")`

Ce type d'animation créera une boucle ne concernant que les deux derniers points-clé, ne répétant que ceux-ci.

Les modificateurs

`loopOutDuration`

Il existe deux variantes de boucles que sont `loopInDuration()` et `loopOutDuration()`. Leur spécificité est qu'elles appliquent une boucle sur une durée de l'animation.

`loopOutDuration("pingpong", 2)` va créer une boucle d'animation sur les deux dernières secondes de l'animation concernée.

`loopInDuration("pingpong", 2)` va créer une boucle d'animation sur les deux premières secondes de l'animation concernée.

L'expression `linear()`

La fonction `linear()` permet de convertir une échelle de valeurs vers une autre. Elle attend cinq paramètres :

`linear(variable, entreeMin, entreeMax, sortieMin, sortieMax);`

Par exemple, il est très facilement possible, grâce à cette fonction, de convertir un pourcentage en degrés (dans l'exemple, le pourcentage est fourni par la variable `controle`) :

`linear(controle, 0, 100, 0, 360);`

Dans cet exemple, la valeur contenue dans la variable `controle` peut aller de 0 à 100. En retour, la fonction `linear` communiquera une valeur allant de 0 à 360.

À noter : la fonction ne tiendra compte que des valeurs d'entrée minimum et maximum ; dans l'exemple donné ci-dessus, si la valeur d'entrée est plus petite que 0 ou plus grande que 100, elle restera bloquée sur les minimum (0) et maximum (100) fournis.

L'expression `ease()`

La fonction `ease()` est identique dans son fonctionnement à la fonction précédente `linear()`. La différence réside dans le fait qu'elle retourne les valeurs converties en utilisant un lissage de vitesse. D'ailleurs, cette expression se décline en trois version :

- `ease()` → Effectuera un lissage sur toutes les étapes de l'animation de la valeur utilisée (identique à `lissage de vitesse` des points-clé)
- `easeIn()` → Effectuera un lissage de vitesse sur le premier point d'animation de la valeur utilisée (identique à `lissage à l'éloignement` des points-clé)
- `easeOut()` → Effectuera un lissage de vitesse sur le dernier point d'animation de la valeur utilisée (identique à `lissage à l'approche` des points-clé)

L'expression `clamp()`

Cette fonction restreint toute valeur donnée à une limite basse et une limite haute.

Exemple :

```
clamp (variable, 100, 250);
```

Dans cet exemple, peu importe la valeur que représente `variable`, elle ne pourra pas descendre en dessous de `100` ni monter plus haut que `250`.