Tutoriel SuperCollider

Chapitre 4

Par Celeste Hutchins

2005

www.celesteh.com

Contrôles

Dans notre dernier chapitre, nous avons commencé à créer quelque chose de plus musical en jouant chaque note d'une gamme, mais dans le désordre. Il peut arriver que nous voulions simplement sauter une note. Nous avons besoin d'un moyen de prendre des décisions. L'un de ces moyens est **if**. `if` possède son propre fichier d'aide. Mettez en surbrillance la lettre minuscule if et appuyez sur pomme-majuscule- ? If est également expliqué brièvement dans le fichier d'aide de **Boolean**.

Un **booléen** est une valeur qui est soit vraie, soit fausse. `true` et `false` sont des mots réservés dans SuperCollider. Nous pouvons envoyer un message `if` aux booléens.

```
(

     ([true, false].choose).if(
          {
               "true".postln ;

          }, {
               "false".postln ;
          }
     ) ;
)
```

Si vous exécutez le code ci-dessus plusieurs fois, "true" et "false" s'afficheront à peu près le même nombre de fois dans un ordre aléatoire, parce que `[true, false].choose` devrait être vrai la moitié du temps et faux l'autre moitié. Le résultat de cette expression est un booléen. Nous envoyons un message if au booléen, qui a deux arguments, tous deux des fonctions. La première fonction est évaluée si le booléen est vrai. La seconde fonction est évaluée si le booléen est faux.

```
booléen.if(trueFunction, falseFunction) ;
```

Vous pouvez omettre la fonction false si vous le souhaitez.

La syntaxe que nous avons utilisée, `object.message(argument1, argument2, . . . argumentN) ;`, est la syntaxe la plus couramment utilisée dans les programmes SuperCollider. C'est ce qu'on appelle la **notation du récepteur**. Cependant, il existe plus d'une syntaxe correcte dans SuperCollider. Il existe également une syntaxe appelée **notation fonctionnelle**. Elle est plus couramment utilisée avec les messages if qu'avec la notation du récepteur. Lorsque vous voyez if dans les fichiers d'aide, les exemples utilisent presque toujours la notation fonctionnelle. La notation fonctionnelle est la suivante :

```
message(object, argument1, argument2, . . . argumentN) ;
```

Les deux notations sont équivalentes. Vous pouvez remplacer l'une par l'autre à n'importe quel endroit de n'importe quel programme et cela ne changera pas le programme. Ce que cela signifie pour if, c'est que l'on voit très souvent :

```
if(boolean, trueFunc, falseFunc) ;
```

Notre exemple deviendrait donc :

```
(
     if([true, false].choose,
          {"true".postln ;

     }, {
          "false".postln ;
     }) ;
)
```

Il fonctionne exactement de la même manière.

Pourquoi y a-t-il plusieurs notations correctes ? C'est déroutant !

SuperCollider est basé sur de nombreux autres langages de programmation, mais le langage auquel il emprunte le plus est celui appelé Smalltalk. Smalltalk, comme SuperCollider, est un langage orienté objet. Lorsque j'ai pris des cours de programmation, mon professeur m'a dit que Smalltalk était le meilleur langage orienté objet et que la seule raison pour laquelle il n'était pas le plus populaire était que sa syntaxe était insensée.

James McCartney, l'auteur de SuperCollider, essayait peut-être de nous épargner les horreurs de la syntaxe Smalltalk et de nous laisser utiliser la notation du récepteur, qui est commune à de nombreux langages orientés objet. La notation fonctionnelle, cependant, persiste dans if, probablement parce que d'autres langages ont des façons différentes de penser à if.

Revenons à notre programme musical et donnons-lui 50 % de chances de jouer une note et 50 % de chances de se reposer :

```
(

    var func, arr ;

    func = { arg ratio_arr, baseFreq = 440, detune = 10
        ; var pitch ;

        Routine.new({
            ratio_arr.scramble.do({ arg ratio, index ;
                if( [true, false].choose, {
                    pitch = (ratio * baseFreq) + detune ;
                    Synth.new("example3", [\freq, pitch, \dur, 1]) ;
```

```
                    }) ;
                    1.attendre ;
                })
                ;
        }) ;
    } ;

    arr = [ 1/1, 3/2, 4/3, 9/8, 16/9, 5/4, 8/5] ;

    func.value(arr, 440, 10).play ;
)
```

Si nous voulons lui donner 33,3 % de chances de se reposer (66 % de chances de jouer), nous pourrions modifier if pour qu'il ressemble à `if ([true, true, false].choose, {` et développer notre tableau à chaque fois que nous voulons modifier la probabilité. Mais que se passe-t-il si nous voulons que quelque chose joue dans 99 % des cas ? Il nous faudrait 99 vrais et un faux. Heureusement, il existe un message que vous pouvez utiliser et qui renvoie un booléen
en fonction du pourcentage. Pour jouer 66% du temps, nous changerions notre if en `if (0.66.coin, {`

```
(

    var func, arr ;

    func = { arg ratio_arr, baseFreq = 440, detune = 10
        ; var pitch ;

        Routine.new({
            ratio_arr.scramble.do({ arg ratio, index ;
                if( ((2/3).coin), {
                    pitch = (ratio * baseFreq) + detune ;
                    Synth.new("example3", [\freq, pitch, \dur, 1]) ;
```

```
                    }) ;
                    1.attendre ;
                })
                ;
        }) ;
    } ;

    arr = [ 1/1, 3/2, 4/3, 9/8, 16/9, 5/4, 8/5] ;

    func.value(arr, 440, 10).play ;
)
```

`coin` est un message que vous pouvez transmettre à SimpleNumber. Il renvoie une valeur vraie ou fausse. Le nombre qui reçoit le message est le pourcentage de probabilité qu'il renvoie une valeur vraie.

Expressions booléennes

De nombreuses opérations arithmétiques renvoient des booléens.
équivalence avec **==**

```
(
    a = 3 ;
    si (a == 3, {
        "true".postln ;
    }, {
        "false".postln ;
    }) ;
)
```

Notez qu'il s'agit de deux signes égaux l'un à côté de l'autre lorsque nous testons l'équivalence. Si vous n'utilisez qu'un seul signe égal, cela signifie qu'il s'agit d'une affectation. Il m'arrive souvent de taper accidentellement un signe égal alors que je voulais en taper deux.

Nous pouvons tester si la valeur **est supérieure** ou **inférieure à** :

```
(
     a = 3 ;
     si (a > 4, {
          "true".postln ;
     }, {
          "false".postln ;
     }) ;
)

(
     a = 3 ;
     si (a < 4, {
          "true".postln ;
     }, {
          "false".postln ;
     }) ;
)
```

Nous pouvons effectuer des opérations booléennes. Les plus importantes sont **not**, **and**, et **or**.

La façon la plus simple de les illustrer est d'utiliser des **tables de vérité**. Une table de vérité vous montre toutes les combinaisons possibles de variables vraies et fausses et les résultats qui en découlent. Toute variable booléenne peut être soit vraie, soit fausse. Voici une table de vérité pour **not** :

Non :

vrai	**faux**
faux	vrai

Not est un **opérateur unaire**. Cela signifie qu'il n'implique qu'un seul objet. Le haut du tableau montre une entrée vraie et une entrée fausse. Le bas du tableau montre le résultat. `true.not` renvoie false et `false.not` renvoie true.

Et est un **opérateur binaire**. Comme +, -, *, / et %, il opère sur deux objets. Disons que nous avons deux variables, a et b, et que l'une ou l'autre peut être vraie ou fausse. Nous pouvons placer a en haut du tableau et b sur le côté gauche. Au milieu, nous plaçons les résultats possibles de a **et** b.

	vrai	**faux**
vrai	vrai	faux
faux	faux	faux

Ou est également binaire :

	vrai	**faux**
vrai	vrai	vrai
faux	vrai	faux

Comment les coder ? Reprenons l'exemple de not. Not est représenté par **!**

```
(
    a = 2 ;
```

```
    si ( !(a == 4) , {
        "true".postln ;
    }, {
        "false".postln ;
    }) ;
)
```

Il ne s'agit pas seulement d'une négation. Il transforme un faux en vrai et un vrai en faux. Il peut également être combiné avec l'équivalence pour tester la **non-équivalence**.

```
(
    a = 2 ;

    si (a != 4 , {
        "true".postln ;
    }, {
        "false".postln ;
    }) ;
)
```

Les deux derniers exemples sont identiques.

Et est représenté par **&&** :

```
(
    a = 3 ;
    b = 4 ;

    if ( (a > 2) && (b < 5), {
        "true".postln ;
    }, {
        "false".postln ;
```

```
    }) ;
)
```

`(a > 2)` **et** `(b < 5)` doivent tous deux être vrais pour que cette expression soit évaluée comme vraie. Si l'un d'entre eux est faux, l'ensemble est faux.

Ou est représenté par `||` (Ce sont des lignes verticales. Sur votre Macintosh doté d'un clavier américain, elles se trouvent au-dessus de la barre oblique) **:**

```
(
    a = 3 ;
    b = 4 ;

    if ( (a > 2) || (b < 5), {
        "true".postln ;
    }, {
        "false".postln ;
    }) ;
)

(
    a = 3 ;
    b = 4 ;

    if ( (a < 2) || (b < 5), {
        "true".postln ;
    }, {
        "false".postln ;
    }) ;
)

(
```

```
	a = 3 ;
	b = 4 ;

	if ( (a > 2) || (b == 5), {
		"true".postln ;
	}, {
		"false".postln ;
	}) ;
)
```

Pour que ces expressions soient vraies, une seule de leurs parties doit l'être. Si aucune partie d'un ou n'est vraie, l'ensemble est faux.

Tandis que

Nous avons utilisé des expressions booléennes pour contrôler le flux d'exécution d'un programme avec if. Une autre **structure de contrôle** est le **while**. While est un message transmis à une fonction. La fonction doit renvoyer soit vrai, soit faux. Il s'agit d'une fonction de test. Une autre fonction est passée en argument. `test_function.while(loop_function) ;` Si la fonction de test est vraie, la fonction de boucle est exécutée. Ensuite, la fonction de test est à nouveau exécutée. Si elle renvoie à nouveau un résultat positif, la fonction en boucle est à nouveau exécutée. Cette opération se poursuit jusqu'à ce que la fonction de test renvoie un résultat faux. *Tant que* la condition est vraie, la boucle est exécutée.

```
(

	var add_amt, max_add, total ;

	max_add = 0,5 ;
	total = 0 ;
```

```
    {total < 1}.while({

        add_amt = max_add.rand ;
        // . . .
        total = total + add_amt ;
        "in while loop".postln ;
    })
    ;
)
```

While estmessage envoyé à une fonction. Rappelez-vousnotation du récepteur et la notation fonctionnelle sont équivalentes. Les deux lignes suivantes sont identiques :

```
test_func.while(body_func) ;
while(test_func, body_func) ;
```

Il existe d'autres **structures de contrôle** détaillées dans un fichier d'aide intitulé "Structures de contrôle". Mettez en surbrillance "Structures de contrôle" et appuyez sur shift-apple- ?

Problèmes

1. Réécrivez hello world du premier chapitre en utilisant la notation fonctionnelle.

2. Rédigez des énoncés de type "si" en utilisant et ou et non en utilisant les valeurs booléennes de vrai et de faux pour illustrer les tables de vérité, en utilisant toutes les combinaisons possibles de vrai et de faux. Par exemple :

```
(
        if ( ! true, {
        "true".postln ;
        }, {
```

```
"false".postln ;
}) ;
```

)

Qu'attendez-vous de chacun d'eux ? Les résultats obtenus correspondent-ils à vos attentes ?

3. Écrivez une fonction qui renvoie une routine. La fonction doit prendre trois arguments : un tableau de hauteurs à choisir, un tableau de durées à choisir et une durée totale pour la routine. Jouez les hauteurs en rythme pendant la durée, puis arrêtez-vous. Vous pouvez utiliser des tableaux de tableaux pour créer des motifs rythmiques ou des thèmes répétitifs.

Projet

Certaines chansons, comme *Bingo* ou certaines chansons à boire allemandes, laissent de côté certaines hauteurs de son lors des répétitions. Écrivez un morceau d'une ou deux minutes qui répète une courte phrase mais qui laisse des notes en dehors des répétitions.