Información preliminar a SuperCollider: curso introductorio

Darien Brito

2 de agosto de 2017

Resumen

Este artículo sirve de preámbulo a conceptos claves para entender con facilidad los vídeos del curso. Si no tienes ninguna experiencia previa con programación, asegúrate de leer la información en este documento antes de empezar el tutorial.

Índice general

Índice de figuras

Capítulo 1

Coding

1.1. ¿Qué es coding?

Este tutorial trata sobre el lenguaje de programación SuperCollider y sus capacidades para crear sonidos y música. Estarás por tanto lidiando con código como método creativo. La práctica de escribir código es denominada "coding". Ten en claro por tanto que todo lo que creas está basado en texto. Mientras más rápido escribas y entiendas las palabras necesarias para ejecutar instrucciones con este texto, mejor. Por esta razón, el curso es un tanto agresivo, conciso y rápido, a fin de forzarte a escribir inmediatamente pequeños programas, incluso si no los entiendes del todo.

El código de computadoras es intimidante al principio y puede ser frustrante no poder entender inmediatamente su modo de operación, dado que es un lenguaje un tanto antinatural para la mayoría de las personas. Piensa sin embargo que es un lenguaje creado por seres humanos para seres humanos, *ergo* cualquier persona es capaz de aprenderlo con el entrenamiento adecuado. La mejor forma de hacerlo es copiando, cambiando parámetros, cometiendo errores y construyendo tus propias instrucciones en base a lo que vas aprendiendo.

La razón por la que es interesante aprender a programar es que nos da una capacidad creativa enorme y un poder expresivo gigantesco. En la actualidad, el código ha pasado a ser una herramienta muy cercana a la práctica artística y hay toda una nueva corriente de arte creada con medios digitales expresados con código. En audio, en concreto, entender los principios de síntesis a nivel de programación te permitirán dominar técnicas en cualquier otra plataforma

y a generar cientos de sonidos con pequeños cambios de parámetros en lugar de usar incontables horas moviendo diales y haciendo click en botones.

Para toda tarea necesitamos tener un conocimiento básico de ciertas nociones antes de comenzar a realizar la tarea en sí. Imagina que estás por ejemplo cocinando algo tan simple como un plato de spagetti. El mínimo conocimiento previo que necesitas para hacer este plato es saber hervir el agua, debes tener los productos adecuados, como la pasta, cebollas, ajo y además tener las herramientas adecuadas, como una olla, una sartén, un escurridor, un plato y utensillos. Con SuperCollider, es exactamente lo mismo. Antes de hacer sonidos, necesitamos primero saber qué productos tenemos disponibles y que herramientas son las adecuadas para cada tarea. Por esto, sé paciente y trata de entender los bloques básicos de programación que se presentan en el curso antes de generar nuestro primer sonido.

1.2. ¿Cómo aprender?

Como he sugerido ya, la mejor forma de aprender a programar es copiando[1] y haciendo pequeños cambios en el código. Por ejemplo, en el primer capítulo de este tutorial verás el programa "¡Hola Mundo!". El código para este programa se ve así:

```
``Hola mundo!''.postln;
```

Inmediatamente puedes probar cambiando algo en este código:

```
``Chao mundo!''.postln;
```

Esta pequeña alteración, si ejecutada, te permite concluir que lo que va entre comillas es el texto a ser impreso y que probablemente no importa realmente lo que vaya escrito dentro:

```
``Esto puede ser cualquier cosa''.postln;
```

Puedes continuar viendo más elementos en esta pequeña línea. ¿Qué pasa si alteras el punto por una coma?

```
``Esto puede ser cualquier cosa'',postln;
```

¡Error! ¿Y si alteramos postln?

```
``Esto puede ser cualquier cosa''.imprimir;
```

[1] Asegúrate de escribir lo que copias y no hacer "copy-paste"

¡Otro error¡ Mmm...

Este ejercicio tan simple ha sido capaz de enseñarnos ya varias cosas muy útiles. Podemos deducir por ejemplo que el punto en la instrucción es necesario y que no puede ser por ejemplo una coma, y que "postln" es una palabra especial para imprimir en la consola y por tanto deben existir otras palabras especiales para realizar acciones diferentes.

Pese a que puedes pensar que esto no tiene nada que ver con hacer sonidos o música, resulta que nos ayuda a comprender cómo funciona una computadora ¡Verás que cuando escribas código para generar audio saber lo que este pequeño programa hace y por qué funciona es muy útil!

1.3. ¿Qué es un repositorio?

Todo el código de soporte para este curso está disponible en un repositorio online alojado en el sitio Github. Ok, ¿pero qué es un repositorio?

Un repositorio es un archivo en donde se almacena información digital. En nuestro caso, es el lugar en donde todo el material de soporte para el curso se aloja. Este material contiene:

- El código de cada capítulo
- Transcripciones de los vídeos
- El código LaTeX para generar este documento
- Este PDF
- El código para generar los sonidos de introducción de cada vídeo

Para acceder al repositorio puedes hacer click en este **link** o puedes ir a la siguiente dirección:

`https://github.com/DarienBrito/Quadro_SCIntro`

1.4. ¿Qué es Open Source?

Open Source es la denominación en inglés para referirse a software cuyo código fuente es libre, es decir, que puede ser visto y modificado por cualquier usuario.

A diferencia de los programas comunes, que funcionan como una caja negra que podemos usar pero a cuyo código fuente no podemos acceder, proyectos de código libre hacen todo disponible sin ninguna restricción. SuperCollider es un proyecto *open source*, así como el repositorio para este curso.

1.5. ¿Qué es GitHub?

De acuerdo a la explicacion en la página oficial de Git:

> GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git.

Una gran cantidad de proyectos *open source* se alojan en GitHub porque es una herramienta muy útil para compartir y modificar código y documentos online. Si quieres aprender más al respecto, puedes leer la documentación en este **link** o yendo a la siguiente dirección:

`http://conociendogithub.readthedocs.io/en/latest/data/introduccion/`

Capítulo 2

Música con programación

2.1. Máquinas para música

Podrías preguntarte: ¿por qué usar un lenguaje de programación para hacer música? e inmediatamente responderte con otra pregunta: ¿por qué no?

Antes de que fuera práctico hacer música electrónica con computadoras, los medios para generar y medir sonidos consistían en circuitos analógicos conectados con cables y controlados con niveles de voltaje. Los primeros usuarios de dichas máquinas no fueron músicos o compositores, sino científicos para quienes eran necesarios ciertos medios a fin de realizar experimentos.

El uso de máquinas electrónicas para hacer música fue impulsado gracias a los esfuerzos de compositores como **Pierre Schaeffer** o **Karlheinz Stockhausen**, que dedicaron gran parte de su práctica artística a expandir posibilidades sonoras y a desarrollar junto con ingenieros nuevos instrumentos electrónicos.

Posteriormente, alrededor de los años 60, pioneros como **Bob Moog** y **Don Buchla** produjeron máquinas analógicas específicamente para propósitos musicales y popularizaron su uso en todo tipo de géneros.

Lógicamente, una vez el poder de procesamiento y las interfaces para computadoras evolucionaron lo suficiente, la producción de música electrónica comenzó a trasladarse a medios digitales. Pioneros en este campo como **Gottfried Michael Koenig** y **Iannis Xenakis** realizaron conceptos e ideas radicales que transformaron el modo en el que escuchamos y producimos música.

2.2. Medios

Existen numerosos predecesores de los programas y aplicaciones para audio actuales, entre ellos Kyma[1], Common Music[2] y CSound[3], que fueron creados en los años 80. Estos programas son ambientes de programación y siguen en uso hoy. El paradigma para hacer música en computadoras cambió con la introducción de la línea de tiempo tradicional e interfaces más amigables como en programas como Ableton Live, Logic o Pro Tools.

En esencia, estos programas no son sino código disfrazado con interfaces más accesibles. La desventaja es que, si creamos música exclusivamente con estas herramientas y tenemos una idea que no forma parte del programa, ¡no podemos realizar nuestra idea! Usar un lenguaje de programación para hacer música y sonidos nos da libertad absoluta para implementar ideas que de otra forma no son posibles.

Programar no es para todo el mundo y eso está bien, porque a veces no necesitamos profundizar tanto en algo para tener resultados. Sin embargo, si sientes que los métodos tradicionales para realizar música no te satisfacen, **SuperCollider** es una herramienta perfecta para experimentar.

2.3. Música con computadoras

Existen obvias y no tan obvias diferencias entre hacer música con medios puramente acústicos, medios electrónicos analógicos y computadoras. ¿Qué es especial acerca de hacer música con computadoras?

- Velocidad de ejecución
- Precisión
- Exploración de ideas formalizadas
- Experimentación con varios modelos compositivos
- Experimentación con procesos generativos
- Creación y re-utilización de instrumentos sonoros

En esencia, hacer cosas que sin una computadora son imposibles de realizar. En el proceso, la computadora puede tomar vario roles. Podría ser por ejemplo:

[1] http://kyma.symbolicsound.com/
[2] http://commonmusic.sourceforge.net/
[3] http://csound.github.io/

- Una herramienta para composición
- Un generador de material sonoro
- Un asistente para organizar material sonoro
- Un asistente para transformación de material (efectos)
- Un performer automático, supervisado o no
- Un medio para analizar música
- Un instrumento musical
- Un medio para reproducir música

2.4. Programación y música

Cuando comenzamos a trabajar con música generada con computadoras, nuestras percepciones cambian y eventualmente desarrollamos un nuevo oído musical. Sonidos que antes no nos parecían especiales comienzan a interesarnos y nuevas estructuras sonoras se forman en nuestra cabeza. Nuestra sensitividad también muta, poco a poco somos capaces de escuchar más y mejor, pues los detalles que podemos controlar y a los que tenemos que prestar atención al componer música digital son en muchos casos más finos y sutiles que al lidiar con instrumentos convencionales. Nuevas preguntas aparecen y nuestras pre-concepciones sobre lo que es musical se aligeran.

Por ejemplo podemos empezar a preguntarnos: ¿hay diferencias entre componer música y componer sonidos? Sin notas definidas, ¿tiene sentido concebir música más alla de escalas y ritmos? ¿podemos controlar sonidos en niveles de tiempo menores al de la nota más rápida? Citando al compositor islandés Bjarni Gunnarsson, quien es a propósito un brillante usuario de SuperCollider:

> Could we think of the sounds a music creates instead of which music sounds create? [4]

Generar arte con algoritmos tiene consecuencias en nuestra forma de pensar y nuestra estética, por esto no es soprendente que nuestro trabajo cambie, a veces drásticamente, una vez que dominamos cierta herramienta. En nuestro caso, esta herramienta es SuperCollider.

[4] ¿Sería posible pensar en los sonidos que una música crea en lugar de qué música los sonidos crean?

Capítulo 3

SuperCollider

3.1. ¿Qué es SuperCollider?

SuperCollider es un lenguaje de programación general orientado a objetos. Esto significa que usando texto y expresiones determinadas, podemos realizar cualquier tarea computacional usando un paradigma de jerarquias y herencia de clases. Su objetivo principal es sintetizar audio en tiempo real y componer sonidos y obras musicales mediante algoritmos.

Dentro del contexto de software para síntesis de audio, es uno de los más versátiles y poderosos ambientes de programación gracias a su sintaxis compacta y a la inmensidad de aplicaciones y extensiones generadas por sus usuarios. SuperCollider es software de código libre y es absolutamente gratuito. Fue originalmente creado por James McCartney y esta constantemente evolucionando gracias a su muy activa comunidad.

Es difícil explicar lo que SuperCollider es capaz de hacer precisamente por la inmensidad de opciones que ofrece. Algunos ejemplos de aplicaciones comunes:

- Síntesis de audio
- Composiciones automáticas
- Creación de instrumentos digitales
- Live coding
- Creación de música generativa
- Creación de instalaciones sonoras
- Interacción con interfaces y sensores

- Desarrollo de software para aplicaciones científicas
- Musíca en red

Sin embargo, podemos realizar proyectos menos comunes y más atrevidos. Por dar un ejemplo bizarro: digamos que queremos sonificar todos los tweets de una cuenta de Twitter en base a ciertas reglas propias y queremos que otras computadoras alrededor del mundo reconozcan mensajes que les digan cuando tocar ciertos sonidos en base a estos tweets. Mientras hacemos esto, queremos enviar datos a diez microcontroladores ubicados alrededor del mundo, que reaccionan de acuerdo al análisis de la señal del mercado de acciones de Google durante los últimos diez años. Cada uno de estos motores toca una campana en diferentes ciudades del mundo, sincronizadas mediante nuestra laptop en casa... ¿cómo haríamos todo esto con software estándar? Pues bien, para esto podemos usar SuperCollider.

Como ves, las aplicaciones son difíciles de definir y engloban una gran cantidad de posibilidades. A modo de ejemplo, puedes ir al link: ***Debris[2] //volatility*** para escuchar una pieza de música electrónica que compuse en el año 2016, cuyos sonidos fueron realizados íntegramente en SuperCollider, usando herramientas propias que escribí. Puedes también accederla en el siguiente link:

`http://darienbrito.com/works/debris2-volatility/`

3.2. Arquitectura

Va mas allá de esta introducción el explicar en detalle cuál es la arquitectura del software en sí. Aprender eso será necesario cuando domines los conceptos básicos y comiences a desarrollar tus propias extensiones y herramientas.

Por ahora basta con saber que SuperCollider consta de tres programas:

1. El intérprete (o cliente). Es la parte que se encarga de traducir las instrucciones que escribimos en nuestros programas de modo que el servidor las entienda.
2. El servidor. Es la parte que se encarga de producir sonidos y valores.
3. La interface. Es la parte en donde escribimos código, accedemos a los archivos de ayuda y supervisamos la consola[1].

Estos tres programas funcionan en sincronía cada vez que usas SuperCollider y pueden ser controlados independientemente desde la interface.

[1]La consola es un monitor que nos muestra mensajes y notificaciones cuando ejecutamos programas. Lo usamos para controlar que nuestro código funciona bien.

3.3. Interface

La interface de SuperCollider nos presenta tres áreas:

1. El espacio de trabajo, en donde escribimos código.
2. La ventana de ayuda, en donde encontramos documentación.
3. La consola, en donde monitoreamos nuestros programas.

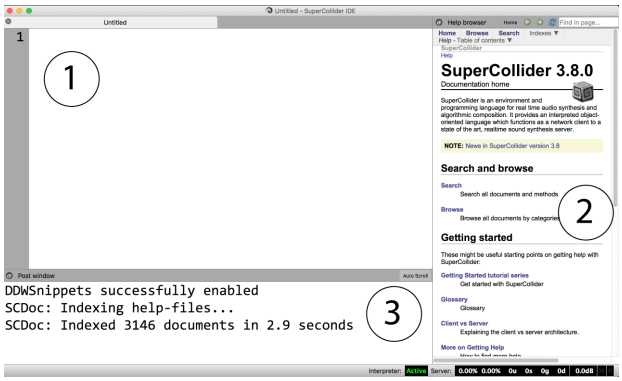

Figura 3.1: La interface

Podemos configurar el modo en el que queremos ordenar esta interface haciendo click en el menú de cada ventana:

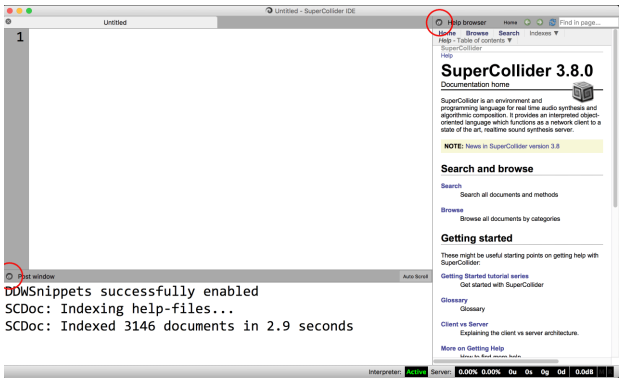

Figura 3.2: Orden de elementos

- Undock/Dock: monta y desmonta la ventana. Podemos arrastrarla y situarla en el sitio desado.
- Detach/Attach: crea una ventana flotante independiente.
- Close: cierra la ventana.

Si cerramos una de estas ventanas por accidente, podemos recuperarla yendo a la pestaña view/docklets:

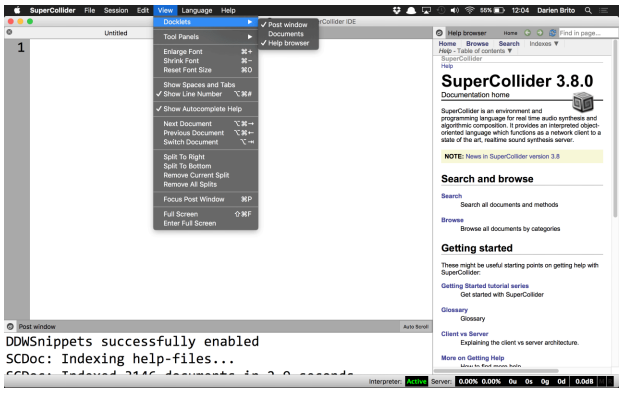

Figura 3.3: Activar docklets

"Post window" es nuestra consola y "Help browser" nuestra ventana de ayuda. "Documents" es un menú adicional que nos permite ver el directorio actual del documento en el que estamos trabajando.

Si queremos alterar los colores de la interface, podemos ir a: SuperCollider/preferences/editor/Fonts&Colors y escoger la configuración deseada. En el caso de este tutorial, yo he escogido el preset *Black* y la fuente *consolas*.

3.4. Nociones preliminares

Finalmente, aquí una guía con síntaxis y objetos que veremos en nuestro curso introductorio. Si la explicación en esta guía no tiene mucho sentido en este momento no hay problema. Esta aquí para prepararte y se complementa con las explicaciones del vídeo.

- **Integer**: un número íntegro. ej: 1, 10, 100
- **Float**: un número decimal. ej: 1.0, 10.0, 100.0

- **String**: una cadena de caracteres, palabras, frase, oración o una secuencia ordenada de longitud arbitraria. Se crean con los símbolos para comillas “ ”. Por tanto, todo dentro de estos símbolos se convierte en una String, incluso números. ej: “hola”, “otra cosa”, “123.1”.

- **Comentarios**: texto que no es evaluado por el intéprete. Se usan para dejar notas o aclaraciones en un programa. Se crean con los símbolos:

  ```
  //
  ```

  Todo lo posterior a estos símbolos se convierte en un comentario a menos que creemos una nueva linea. A traves de nuestro tutorial, encontrarás varios comentarios que aclaran líneas de código. ej:

  ```
  SinOsc.ar() // este es un oscilador sinusoide
  ```

  Para comentarios multi-línea podemos usar los símbolos:

  ```
  /*   */
  ```

  Todo lo que va dentro de estos símbolos se convierte en un comentario. ej:

  ```
  /*
  Todo esto es un comentario.

  Puedo crear varias líneas.
  */
  ```

- **Array**: una colección de elementos. Un Array se crea con los símbolos:

  ```
  [ ]
  ```

  Un array sirve para almacenar varios objetos en una sola caja, por así decirlo. Podemos acceder a los elementos de esta caja usando un número índice, que es la posición del objeto en el Array. ej:

  ```
  [1, 10.0, ‘‘hola’’]
  ```

- **Punto y coma (;)**: El punto y coma sirve para decirle al intérprete que una instrucción determinada a concluido. A través de los programas que escribamos verás como separar instrucciones usando este símbolo.

- **var**: una variable. Se usa para referirnos a un objeto usando una palabra clave arbitraria. Esta palabra puede ser cualquier cosa siempre y cuando no inicie con mayúsculas o números y no tenga caracteres especiales como acentos. Para asignar un valor a una variable, usamos los signos de igual. ej:

  ```
  var miVariable = 10;
  ```

- **postln**: un método. Métodos son funciones que los objetos entienden a fin de realizar una acción. El método postln se encarga de imprimir el objeto en la consola. Para usar un método, necesitamos usar antes un punto, a fin de que el intérprete sea capaz de entender en donde inicia el método. ej:

  ```
  ‘‘¡Hola Mundo’’.postln;
  ```

- **Funciones**: una función es una serie de instrucciones encapsuladas en un objeto. Se usan cuando queremos crear porciones de código re-utlizables. Se crean con los símbolos:

  ```
  { }
  ```

  Podemos asignar una función a una variable y reutilizarla a través de nuestro programa. ej:

  ```
  var f = { ‘‘Hola mundo’’.postln };
  ```

  Para evaluar una función usamos el método value. ej:

  ```
  f.value;
  ```

- **arg**: un argumento. Se usa para crear una compuerta a través de la cual pasar un objeto. Usamos argumentos cuando tenemos funciones que requieren valores flexibles para operar. ej:

  ```
  var f = {arg i; i.postln }; // imprime el objeto en la consola
  f.value(1); // imprime 1
  f.value(10); // imprime 10
  ```

3.5. Usando los documentos de soporte

Como ya hemos dicho, existen documentos de soporte en el **repositorio** de nuestro tutorial. La idea de estos documentos es que tengas a la mano todo lo que vamos viendo en cada vídeo. Te recomiendo realizar variaciones de cada ejercicio en los documentos por tu cuenta, a fin de entender los conceptos tal y como se van presentando. Para este propósito, he dejado un espacio entre líneas de código, de modo que te invite a experimentar un poco.

Te deseo buena suerte con este tutorial y espero que la información que encuentres te sea de ayuda. No dudes en escribir a la dirección dada en la plataforma con sugerencias o comentarios. ¡Feliz programación!