# Música por computadora

Ernesto Romero y Hernani Villaseñor
Centro Multimedia 2012

## Sesión 1

1- Introducción a SuperCollider:
    Historia, relevancia, figuras destacadas, ejemplos de obras de distintos estilos y disciplinas.
2- Elementos estructurales de la aplicación:
    IDE, servidor y cliente.
3- Programación orientada a objetos:
    3.1- Sintáxis, objetos, métodos, argumentos(), rates, audio rate .ar, control rate .kr, *strings* y símbolos.

### 1 - Introducción a SuperCollider

SuperCollider es un ambiente y lenguaje de programación para síntesis en tiempo real y composición algorítmica. Está provisto de un lenguaje orientado a objetos que es interpretado y funciona como un cliente en red, con un servidor de síntesis en tiempo real de alto nivel de desempeño.

#### Historia

SuperCollider fue creado por James McCartney en 1996, la primer versión corría en Power Macintosh y costaba 250 dólares. Posteriormente fue liberada para convertirse en una aplicación de uso libre y de código abierto. Actualmente es un programa multiplataforma ya que corre en Mac OSX, Windows y Linux.

Originalmente SC constaba de dos programas separados: Synth-O-Matic, -escrito en 1990 por McCartney- y por el objeto de MAX llamado Pyrite, el cual contenía el lenguaje intérprete. A partir de la versión 2, SC se compone de los programas scsynth como servidor y sclang como lenguaje.

**Relevancia**

SC es una de las principales herramientas usadas en el ámbito de la música electrónica, el arte sonoro y la multimedia. Es utilizado para la enseñanza en las aulas de muchas universidades y centros académicos alrededor del mundo. También tiene un papel importante en la investigación científica en ramas como la acústica y la psicoacústica.

**Figuras destacadas**

La comunidad que se ha formado en torno a esta herramienta es basta, algunas figuras destacadas son:

Nick Collins, Fredrik Olofsson, Cylob, Andrea Valle, Sergio Luque, Roberto Morales, Juan Sebastián Lach, Dan Stowell, Julian Rohrhuber, Alberto de Campo y Thor Magnusson.

**Ejemplos de obras de distintos estilos y disciplinas**

- sc140, códigos de 140 caracteres, 22 códigos en colaboración con la revista Wire, los cuales hacen referencia a el número de caractéres permitidos en un tweet.

- LiveCoding HackPact de Fredrick Olofsson.

- “Virtual electronic poem” de Andrea Valle, una recreación en SuperCollider de -“Poema electrónico” de Edgar Varèse.

**2 - Elementos estructurales de la aplicación**

**IDE (Entorno de Desarrollo Integrado)**

Ambiente de desarrollo integrado o IDE[1], es una aplicación que nos permite ingresar comandos y pedir que estos se ejecuten, también nos muestra los resultados, errores y avisos del programa. En MacOS y en Windows SC incluye su propia IDE. En Linux es necesario abrir una IDE independiente de SC y pedirle que entre en el modo de SuperCollider. Los IDE mas usados en Linux son: Gedit, Emacs y Vim. Recientemente con el desarrollo de la IDE basada en Qt, todas la plataformas son visualizadas de manera igual.
Cada IDE tiene sus atajos o *shortcuts* para realizar las acciones esenciales de SuperCollider. A continuación presentamos una lista de estas acciones y sus atajos correspondientes.

Antes prendemos y apagamos SC con el siguiente código:

```
// prende supercollider
s.boot;
// apaga supercollider
s.quit;
```

1 Integrated Development Environment

| | MacOS | Windows | Gedit | Emacs |
|---|---|---|---|---|
| Evaluar selección | Enter (no Return) | Ctrl+Intro | Ctrl+E | Ctrl+C + Ctrl+C |
| Detener procesos | cmd+. (cmd+punto) | Alt+. (Alt+punto) | Esc | Ctrl+C + Ctrl+S |
| Abrir archivo de ayuda | cmd + D | F1 | Ctrl+U | Ctrl+H |

SC muestra dos ventanas, la *Post Window* que es una ventana donde se imprime el resultado del proceso de lo que realiza el programa, así como los errores y una ventana donde escribimos el código a manera de texto. Además podemos visualizar algunas GUIs[1] como los servidores, medidores y el estetiscopio.

```
s.makeGui // crea una gui del servidor activo
s.meter // crea una gui de los medidores de entrada/salida
s.scope // crea una gui del estetiscopio
```

**Servidor**

En SC existen dos servidores: servidor interno y servidor externo. Para producir sonido es necesario prender cualquiera de estos.

El servidor interno corre en el mismo proceso que la aplicación SuperCollider también llamada “sclang”. Este servidor es interno al programa lo cual tiene ciertas ventajas derivadas de una mayor comunicación con el sclang.

El servidor local corre en la misma máquina que la aplicación Supercollider, aunque es un programa diferente llamado “scsynth”. La ventaja de usar este servidor es que en caso de que el servidor se caiga, la aplicación seguirá corriendo y viceversa.

**Cliente**

SC funciona bajo el modelo de cliente/servidor, ambos trabajan dentro de una red, es decir el usuario escribe programas mediante los cuales el cliente solicita al servidor que haga algo.

**3 Programación orientada a objetos**

**Sintaxis**

La sintaxis es la estructura en que debemos acomodar los elementos del lenguaje de SC para que este entienda nuestros códigos. La sintaxis es dada por el programa y tenemos que

1 *Graphic User Interface* o Interfaz gráfica de usuario.

aprenderla para poder escribir cosas con sentido y comunicarnos con SC. Es difícil entender la sintaxis sin comenzar con ejemplos que involucren a la partes del lenguaje a ordenar. Aquí una lista de elementos básicos de la sintaxis del programa. Durante el desarrollo de las sesiones se irá describiendo su uso dentro de un contexto específico.

| | |
|---|---|
| `//` | las dos barras diagonales definen un comentario. |
| `/*` | una diagonal seguida de un asterisco abre una sección de comentario. |
| `*/` | el asterisco seguido de diagonal cierra una sección de comentario previamente abierta. |
| `{}` | las llaves definen una función. |
| `[]` | los corchetes definen un arreglo. |
| `()` | los parentesis definen un argumento o un bloque de código. |
| `\|\|` | los pipes definen argumentos. |
| `Algo` | Una palabra que inicia con mayúscula representa un objeto. |
| `.otraCosa` | Un punto seguido de una palabra que inicia con minúscula representa un mensaje o método. |
| `;` | el punto y coma indica una ruptura o *break* en el código. |
| `,` | la coma separa argumentos. |
| `"algo"` | cualquier cosa escrita entre comillas es un *String.* |
| `\algoMas` | cualquier palabra escrita después de una diagonal es un símbolo. |
| `~` | la tilde indica una variable global. |

**Comentarios**

Los caracteres `//` o `/* */` sirven para decirle a SC que vamos a escribir algo que no se interpretará como código. Esto se llama comentar. SC hará caso omiso de lo que comentemos. Hay dos maneras de comentar:

```
// Este es un comentario breve

/* Este es un comentario largo y tiene que cerrarse */
```

**Objetos**

Los elementos básicos de SC son llamados objetos. Casi todo en SC es un objeto. Los objetos son capaces de realizar ciertas tareas, y es la conjugación de varios objetos lo que nos permite realizar tareas complejas. Cuando escribimos un objeto siempre empieza con mayúscula.

Existe una clase especial de objeto: los `UGens`[1]. Estos son los objetos que generan sonido en SC y cuyas tareas son manejadas por el servidor, por ejemplo: `Pulse` o `SinOsc`.

1 Unit Generators o Unidades Generadoras

**Métodos**

Los métodos son también llamados mensajes y es a través de ellos que podemos decirle a los objetos lo qué queremos que hagan y cómo queremos que lo hagan. Al mismo objeto se le pueden enviar distintos métodos, pero un objeto no aceptará cualquier método, sino solo un grupo de métodos que están asociados a el. Cuando escribimos un método siempre va después de un punto y empieza con minúscula.

Ejemplo: Los números enteros en SC son objetos o instancias del objeto Integer. Enviemos al objeto `7` los mensajes `odd` y `plot`.

```
7.odd // odd pregunta si es el objeto es un número impar

7.plot /* plot hace una gráfica en 2D del objeto. Un número
          entero no puede arrojar una gráfica 2D por lo que al
          declarar esta línea obtenemos un error en la ventana
          Post. */
```

`.odd`, pero, por ejemplo si ponemos `.plot`, este nos arrojará un error, ya que es un mensaje que los números no aceptan.

También hay métodos que pueden escribirse en diferentes objetos.

Ejemplo:

```
// el mensaje .postln imprime información en la post window
"palabras".postln
```

**Argumentos**

Los argumentos son parte de un método y nos ayudan a decirle al objeto, específicamente, de que forma queremos que realice una tarea. Los argumentos se escriben entre paréntesis y van separados por comas en caso de ser más de uno. Ejemplo:

```
// nos redondea el número entero
10.0975.round (1)

// nos redondea a un decimal
10.0975.round (0.1)

/*
Los argumentos se escriben entre paréntesis y se separan por
comas si son mas de uno.
```

```
El mensaje linlin nos convierte un número dentro de un rango en
su equivalente dentro de otro rango. 98 es, dentro del rango de
0 a 100, igual a 0.98 dentro del rango de 0 a 1.0.
*/
98.linlin(0, 100, 0,1.0)
```

**Rates**

En SuperCollider la información de los `UGens` es tratada de dos formas: como audio o como control. Para esto están definidos dos mensajes que indican al objeto de que forma será tratado: `.ar` y `.kr`

**Audio rate** `.ar`

Los UGens que reciben el mensaje `.ar` corren a velocidad de audio: 44100 muestras por segundo. Hay que mandar el mensaje `.ar` a estos `UGens` cuando quieran ser tratados como señal (para que se escuchen).

**Control rate** `.kr`

Los `UGens` que reciben el mensaje `.kr` corren a velocidad de control: 689 muestras por segundo. Es por esto que los `UGens` de control son más baratos computacionalmente hablando que sus contrapartes a velocidad de *audio rate*. Notar que un `UGen` con el mensaje `.kr` no se escuha.

Los `UGens` de control los usamos como moduladores, esto es, como señales que le dan forma a una señal de audio. En otras palabras los `UGens` con `.kr` controlan los argumentos de los `UGens` con `.ar`

**Strings**

Los strings son cualquier cadena de caracteres (i.e. palabras, números), que se escriben entre comillas y no tienen ningún significado para SC. Ejemplo:

```
"SuperCollider no sabe que estamos hablando de él".postln
```

**Símbolos**

Los símbolos son nombres garantizados como únicos. No hay dos cosas nombradas con el mismo símbolo. Se escriben con una diagonal invertida antes de una cadena de caracteres.

Ejemplo: `\unSimbolo`

Los símbolos son usados normalmente para nombrar estructuras de código predefinidas, como puede ser `SynthDefs` o `Tdefs`, por ejemplo: `Tdef(\nombre), SynthDef(\name).`

**Notación**

La tipografía Arial 11 se usa para el texto del tutorial, la tipografía `Courier 12` para el código.

**Referencias**

McCartney, J. (1996). SuperCollider: a new real time synthesis langauge. ICMC.

Netri, E. y Romero, E. (2008). Curso de SuperCollider Principiantes. Centro Multimedia: México.

Polishook, M. (2004). Introductory Tutorial: For SuperCollider 3. Archivo de ayuda de SuperCollider.

**Recursos**

http://cmm.cenart.gob.mx/tallerdeaudio
http://supercollider.sourceforge.net