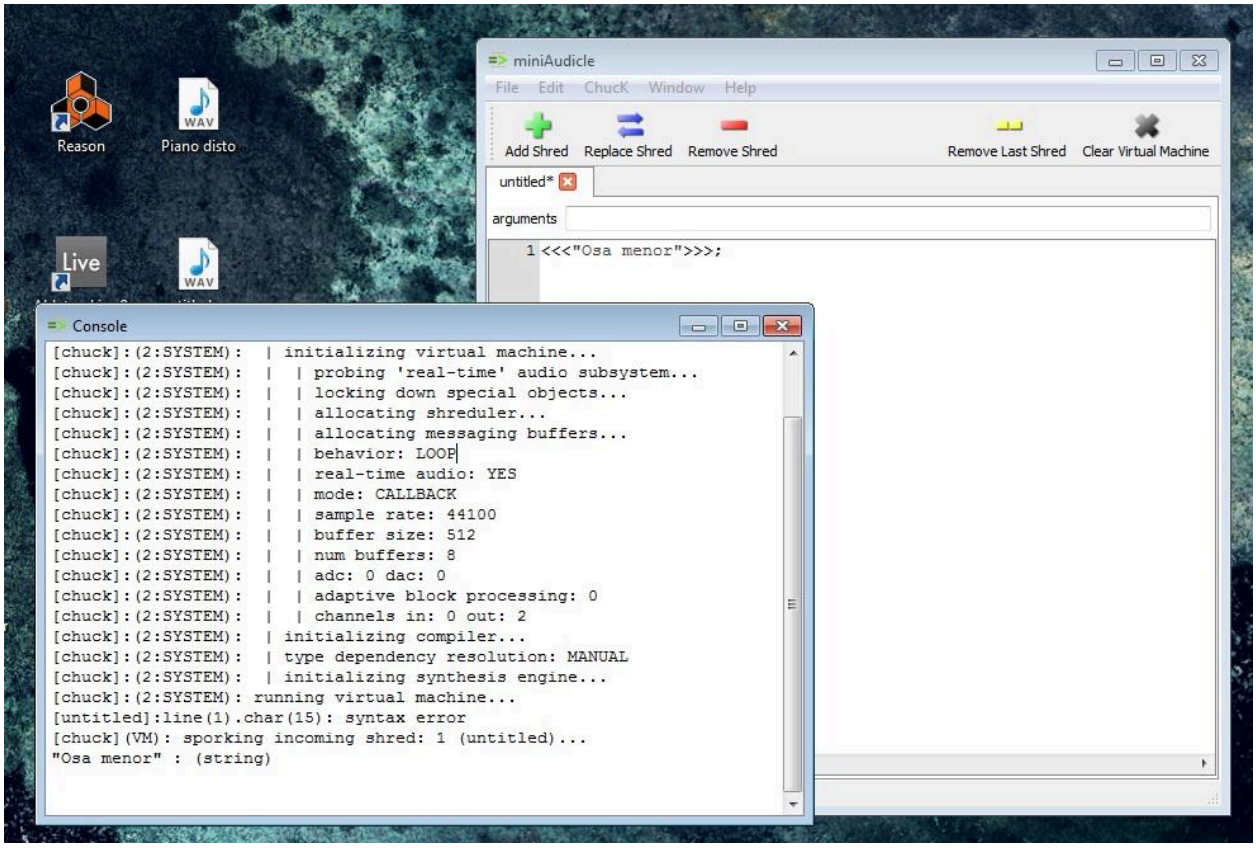
ChucK es un lenguaje orientado a la composición musical. Contiene, entre otros paradigmas, el de POO. Fue creado por Ge Wang y Perry R. Cook, basado ligeramente en C.

En este manual se transcribe de manera cronológica del cuaderno donde fue apuntado por Óscar A. Montiel.

Primero aprenderemos a hacer strings. Para ello se escribe básicamente:

<<<"Osa Menor">>>; //Se escriben los <<<"">>> para denotar que es un string que se verá en la consola.



De igual forma sirve la concatenación:

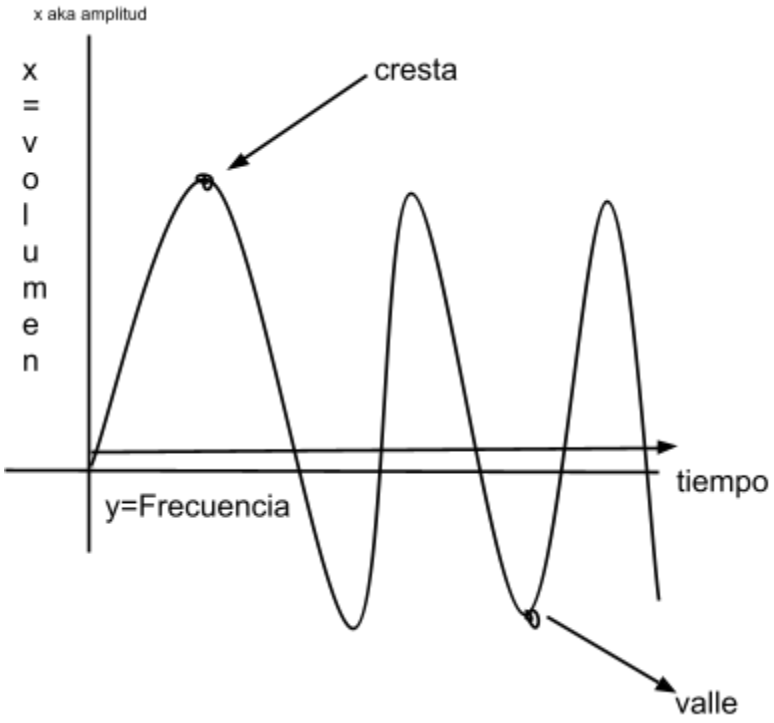
<<<"Osa Menor se casó con "+variableDelNombreDeSuEsposa>>>;

Normalmente se usa para juntar strings con variables, y ambas serán representadas en la consola.

Una cosa importante, bueno, mejor dicho **vital**, es que debemos iniciar la máquina virtual, es

decir, donde se ejecutará el código. Simplemente se le pone en *ChucK - Start Virtual Machine*. Cuando ejecutamos el código (*Add Shred)*, se crea un *Shred* que es como el código almacenado en una caja que se ejecuta. Uno puede tener tantos *shreds* (códigos) diferentes trabajando de manera independiente, pero eso se verá después.

Ahora pasaremos a lo que es una onda senoidal:



Perdón por la onda tan rara, pero demuestra lo que es. Básicamente, una onda senoidal es una onda con curvas bonitas, donde su frecuencia (velocidad de oscilación) afecta el pitch, y la altura (amplitud) a donde llegue su cresta (punto más alto de la onda) es su volumen.
Sin embargo, hay un concepto clave: sin *tiempo* no puede haber sonido.

Bueno, vayamos a la parte de programación. En ChucK, para empezar a escuchar soniditos:

SinOsc mi_sonido => dac;

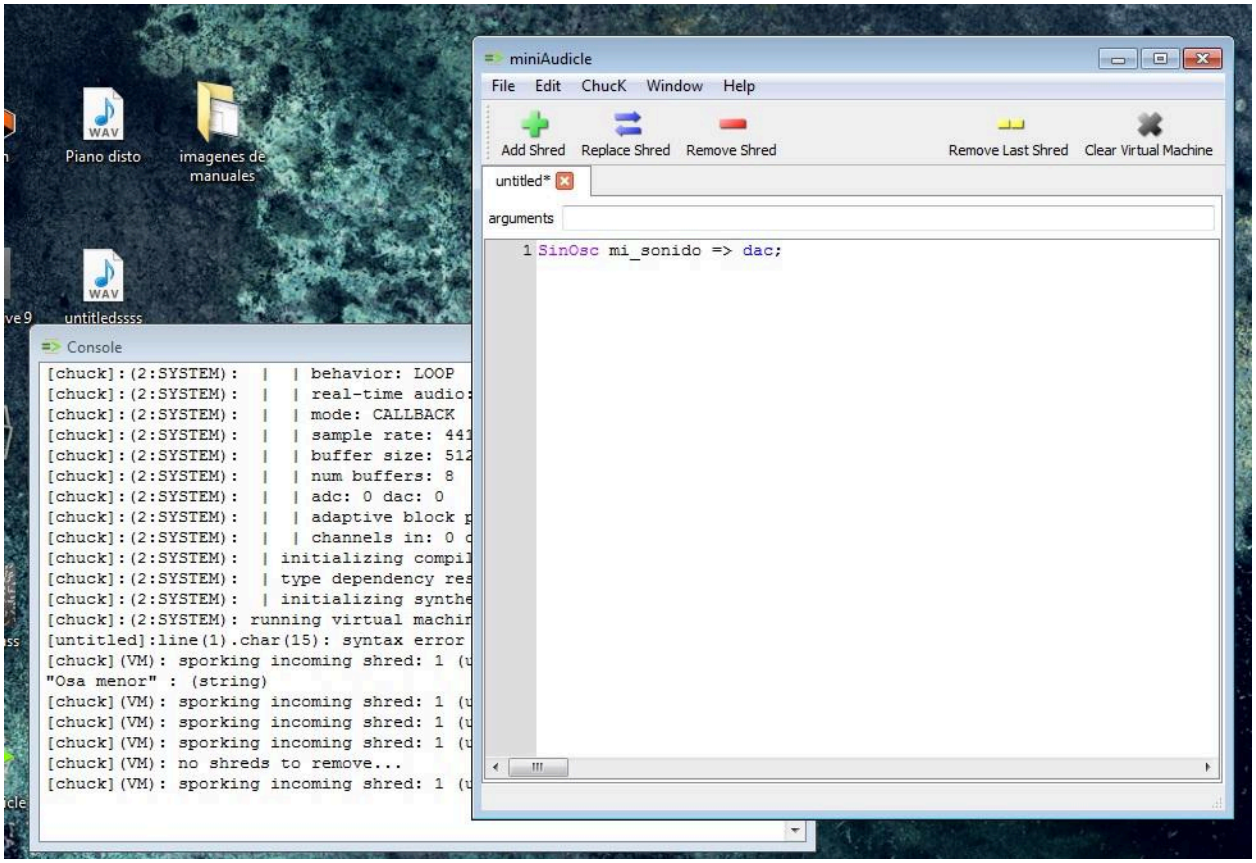
- *SinOsc* es el nombre de una *clase* dentro de ChucK que contiene todos los datos para generar una onda senoidal.

- *mi_sonido* no es más que el nombre de la variable. O sea, de la clase *SinOsc* “nace” un

objeto llamado *mi_sonido*.

- => es la manera de decir “es igual a”, como el = pero en ChucK es =>.
- *dac* es para que suene el sonido.

En ChucK, se “conectan” los diferentes módulos, como si fueran hardware siendo conectados vía cable. Por eso, cuando se dice *mi_sonido => dac;* es como estar conectando un aparato de ondas senoidales *SinOsc* llamado *mi_sonido* conectado a un altavoz, o *dac* (digital to analog converter). Así puede sonar :)



Sin embargo, si le puchamos Add Shred (que es para ejecutar el código) no se escuchará nada. ¿Por qué? Porque no hemos agregado ni la **frecuencia, volumen** ni el ingrediente principal que es **el tiempo.**

Volumen: Para ponerle un volumen a nuestro sonido *mi_sonido,* se usa un método del objeto llamado *.gain*:

0.6 => mi_sonido.gain; //0.6 es el volumen (del 0 al 1) ;)

Frecuencia (o pitch): Se usa el método *.freq*.

220.776 => mi_sonido.freq; //La frecuencia corresponde a una nota de una escala (diatónica,

melancólica, la que sea), pero, dado que puede tener decimal, alcanza notas microtonales :)

Tiempo: Para usar el tiempo en ChucK no se usa un método específico, sino un código especial para ello.
Tenemos los siguientes *keywords* para manejar el tiempo:

- *samp // duración de 1 sample en tiempo ChucK.*
- *ms //duración en milisegundos (1000ms es un segundo).*
- *second //duración de 1 segundo.*
- *minute //duración de 1 minuto.*
- *hour //duración de 1 hora.*
- *day //duración de 1 día.*
- *week //duración de 1 semana.*

Para empezar, podemos aplicar tiempo a *mi_sonido* de la siguiente manera:

1::second => now //1 segundo (1::second) al(=>) tiempo actual (now). Es un poco difícil de explicar, pero cuando se programa se entiende intuitivamente. Los :: siempre se ponen. Porque sí.

Tipos de datos:

En ChucK, como en la mayoría de los lenguajes de programación, podemos encontrar los *floats* (decimales) y los *int* (números enteros). Ambos pueden ser + ó -. La manera de asignar valores a variables de algún tipo es la siguiente:

int entero; //Primero se pone el tipo de dato, en este caso int (número entero), y, luego, el nombre que queramos.

float decimal; //número decimal.

Ahora, para aplicarle un valor, se hace lo siguiente:

2 => entero; //entero valdrá 2.

3.1 => decimal; //decimal valdrá 3.1.

¡Fácil! De igual forma podemos aplicar valores inmediatamente a la creación de una variable:

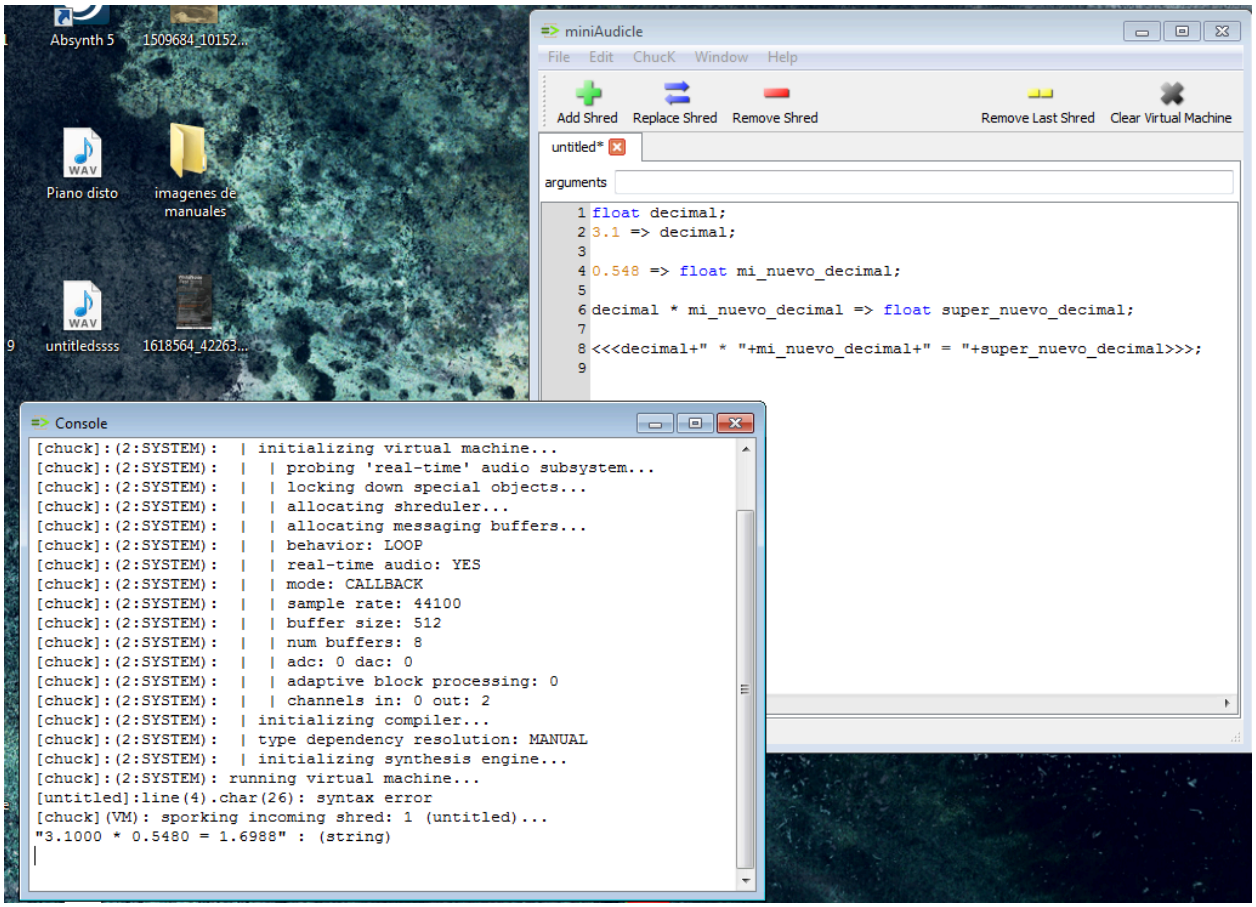
2 => int mi_nuevo_entero;

0.548 => float mi_nuevo_decimal;

También podemos realizar operaciones aritméticas con las variables y, de paso, asignarla a una nueva variable:

*decimal * mi_nuevo_decimal => float super_nuevo_decimal;*

*<<<decimal+" * "+mi_nuevo_decimal+" = "+super_nuevo_decimal>>>;*

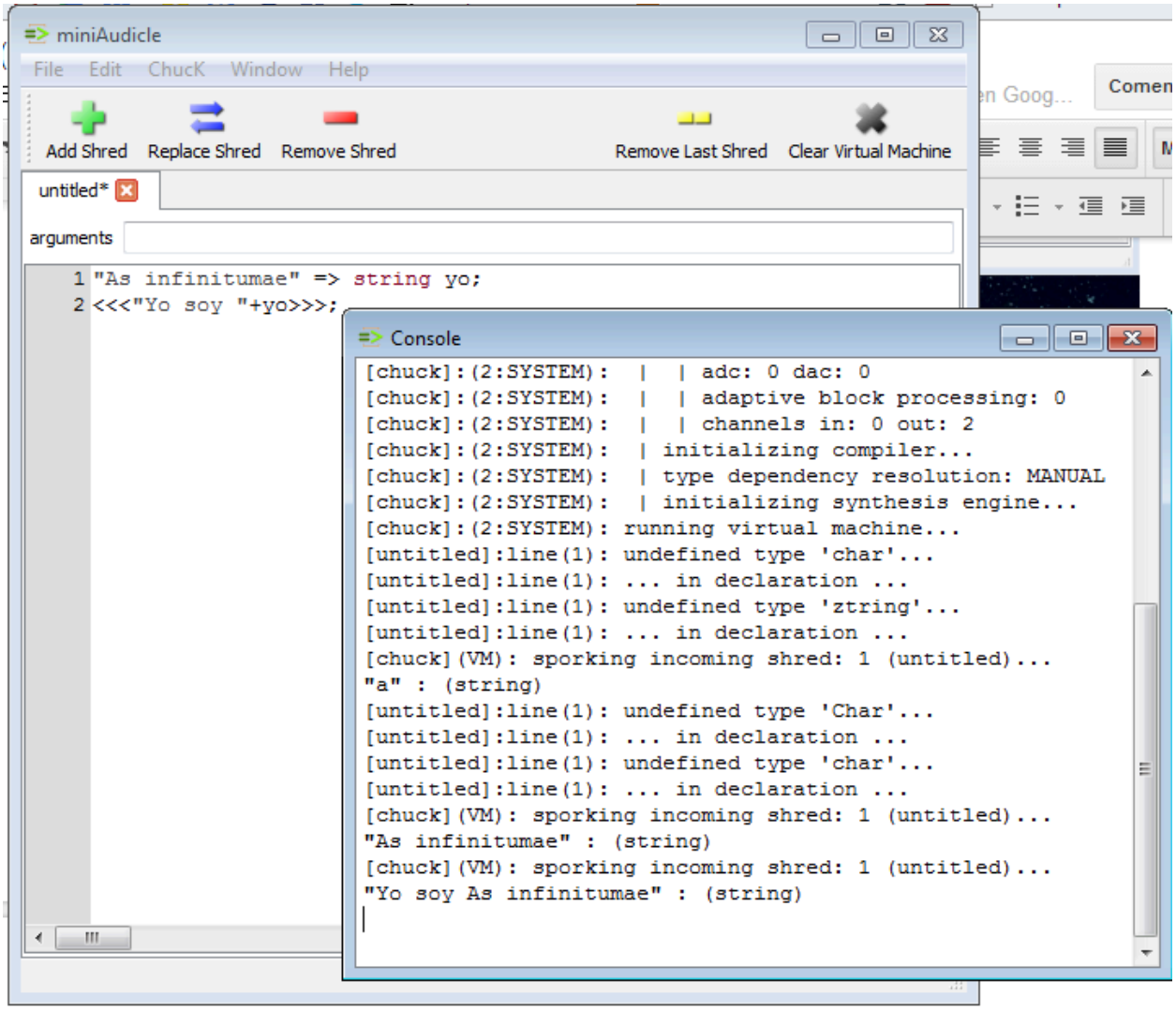


Tenemos también otro tipo de datos, unos que manejan letras (*strings):*

- *char //Manejan un sólo caracter. ejem “a”;*
- *string //Manejan una palabra, o palabras. ejem “Ad infinitum”;*

He aquí unos cuantos ejemplos:

“Ad infinitumae” => string yo;
<<<”Yo soy “+yo>>>;



Tiempo:

La manera en la que ChucK maneja el tiempo es lo que lo hace destacable de entre los lenguajes de programación orientados a la música. Además de su sintaxis simple, ChucK mantiene un *reloj interno* (que se activa al iniciar el Virtual Machine). *now* es la palabra/variable reservada que mantiene la cuenta de dicho reloj dentro del programa. Es por eso que cuando escribimos:

SinOsc sine => dac; //Todo lo que queramos que suene va conectado al dac.

while (true) //Es un bucle "infinito".
{
441 => sine.freq; //Aquí asigna qué frecuencia (o nota) queremos que suene.
0.3 => sine.gain; //Aquí volumen

*1::second => now; /*Aquí decimos que queremos que dure 1 segundo el código escrito arriba. cualquier cosa por debajo de ésto ya no tiene efecto, ya que el código se lee línea por línea, de arriba a abajo.*/*

220 => sine.freq; //Asignamos un nuevo valor de frecuencia a sine.
500::ms => now; //Ahora sonará ½ segundo, o 500 milisegundos. Terminando regresará nuevamente hasta arriba y seguirá haciendo lo mismo hasta que detengamos el shred actual.
}

Usar *for*:

El uso de *for* es el básico. Sin embargo, dado que en ChucK se usa el => en vez del =, el código *for* quedaría así:

for (0 => int i; i < 10; i + 1 => i)

Convertir MIDI a frecuencias y viceversa:

Las notas musicales suenan como tal debido a la frecuencia con la que oscilan. Un Do es diferente a un Sol porque una tiene una frecuencia de 130.81hz y 196, respectivamente (en escala 3). ChucK puede usar esos números para usar el método *.freq*. Pero también puede usar el protocolo MIDI. En el MIDI, las notas, en vez de llamarse Do, Re, Mi, etc., se llaman 1, 2 , 3, hasta el 127 (cada 12 números es una octava).
ChucK puede convertir frecuencias a MIDI y MIDI a frecuencias con el siguiente código:

Std.mtof(int valor);
//Std es la librería estándar en ChucK. mtof es MIDI to frequenci. int valor es donde se pone el número MIDI.

Std.ftom(float valor)
//Ahora, en vez de poner un int, normalmente se pone un float, usando la frecuencias que queramos (o sea la nota), y ésta se convertirá al MIDI correspondiente.

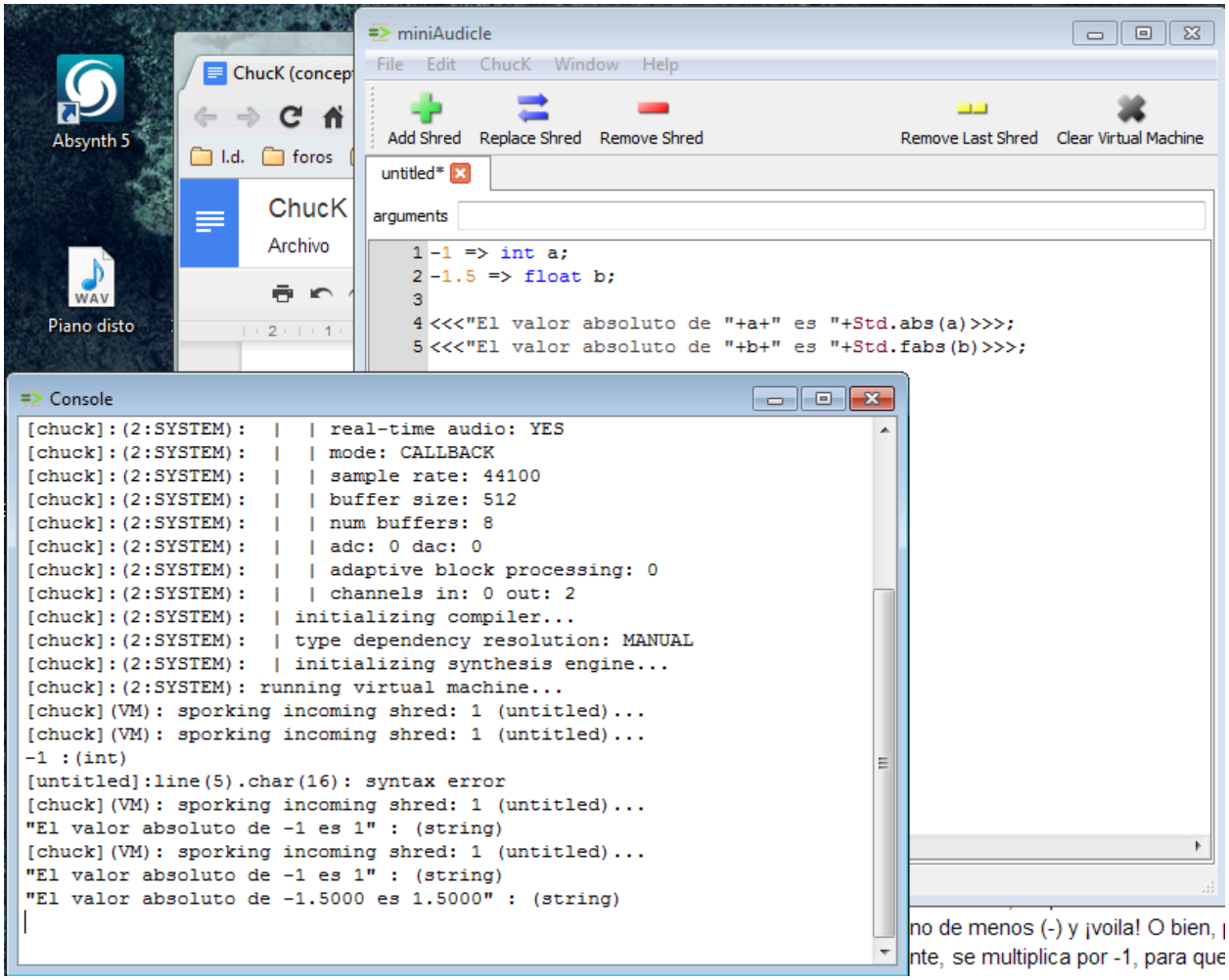
Así podemos escuchar algún sonido en una nota específica, ya sea MIDI o su respectiva frecuencia :)

Convertir a valor absoluto algún… valor:

El valor absoluto simplemente es cualquier número pero positivo. Para obtener el valor absoluto de, digamos, 8, sólo se deja como está, ya que es positivo. Ahora, si queremos el valor absoluto de -8, lo único que tenemos que hacer es quitarle el signo de menos (-) y ¡voila! O bien, para los pendejos matemático que quieren todo matemáticamente, se multiplica por -1, para que pase a ser positivo. ¿Por qué es "absoluto" el valor? No sé, pero así es.
En ChucK tenemos un par de métodos para hacer tal labor:

Std.abs(int valor negativo);
//.abs quiere decir absoluto.

Std.fabs(float valor negativo) ;
//Lo mismo pero en float (o decimal para los hippies).



Convertir *float* a *int* (casting):

"Casting", en la jerga de los programadores, se define como un valor de un tipo dado (digamos *float*) queremos que salga entero (*int*).
Para hacer éso en ChucK se hace lo siguiente:

Std.ftoi(valor float);
//float to integer.

Simplemente se removerán los 0 extras o lo que esté después del punto decimal, para que quede como un puro e inexacto entero ;)

Librería Math:

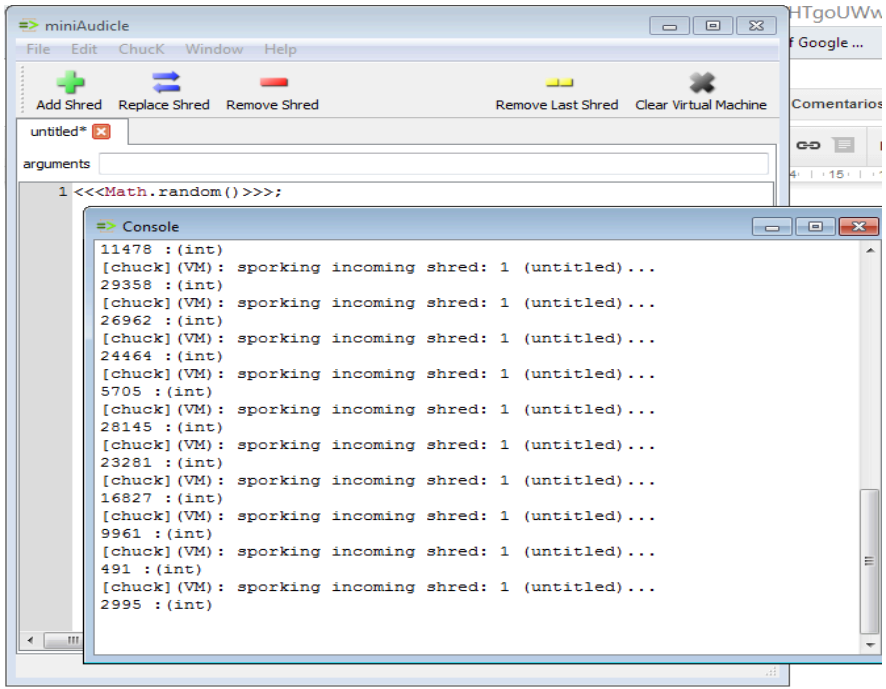
¡Matemáticas! Para muchos la ciencia más bella, madre de todas, para otros una mierda que te enseñan en la escuela.

Random:

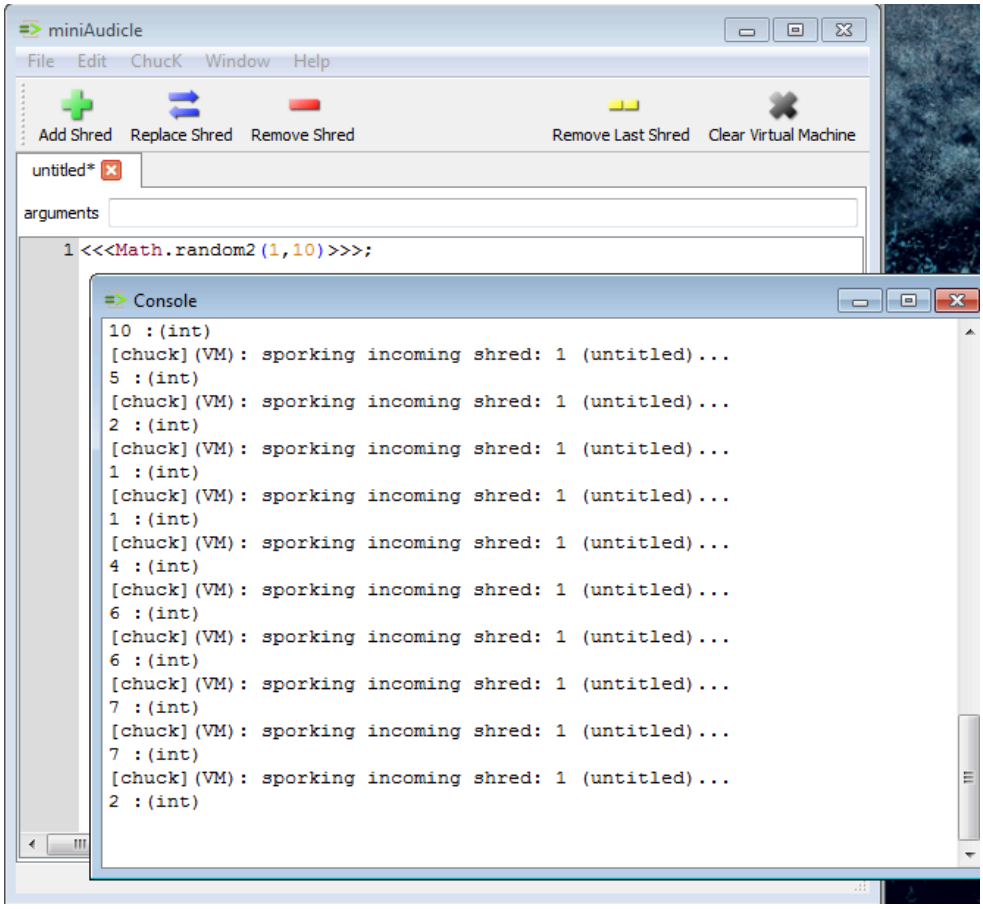
Usar la función *random* en ChucK dará como resultado un número elegido "al azar" mediante una función matemática. Cabe resaltar que no será 100% al azar, ya que, en realidad, el resultado que arrojaría está basado en varios parámetros establecidos. ¿Hay algo verdaderamente aleatorio en el Universo? Lo dudo.

Hay diferentes tipos de *random* en ChucK

Math.random(); //Genera un número entero entre 0 y un numerote.



Math.random2(int,int); //Genera un número entre el rango especificado en los parámetros incluyéndose ellos mismos. Cabe destacar que sólo serán enteros, sin decimales.

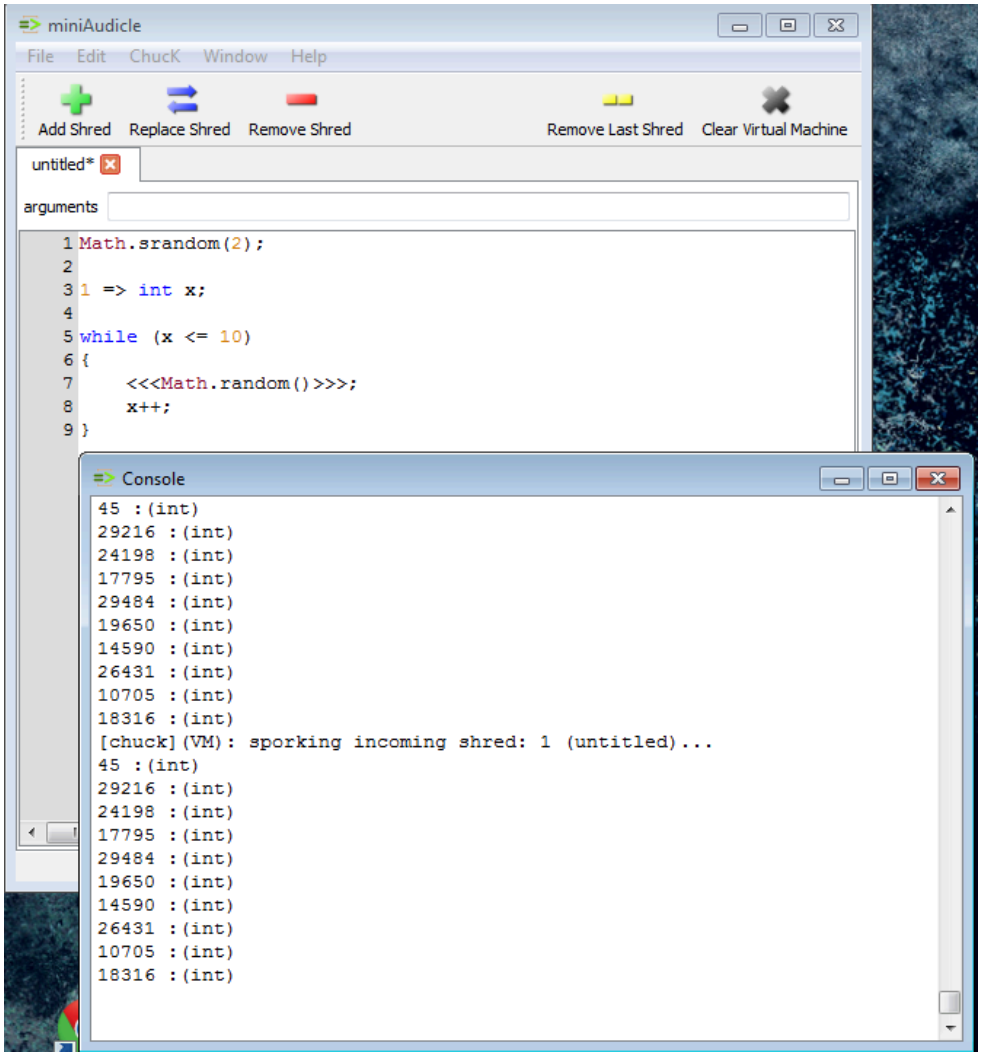


Math.randomf(); //Dará decimales aleatorios entre 0 y 1. Todos decimales.

Math.random2f(float,float); //Dará resultados decimales entre el rasgo especificado.

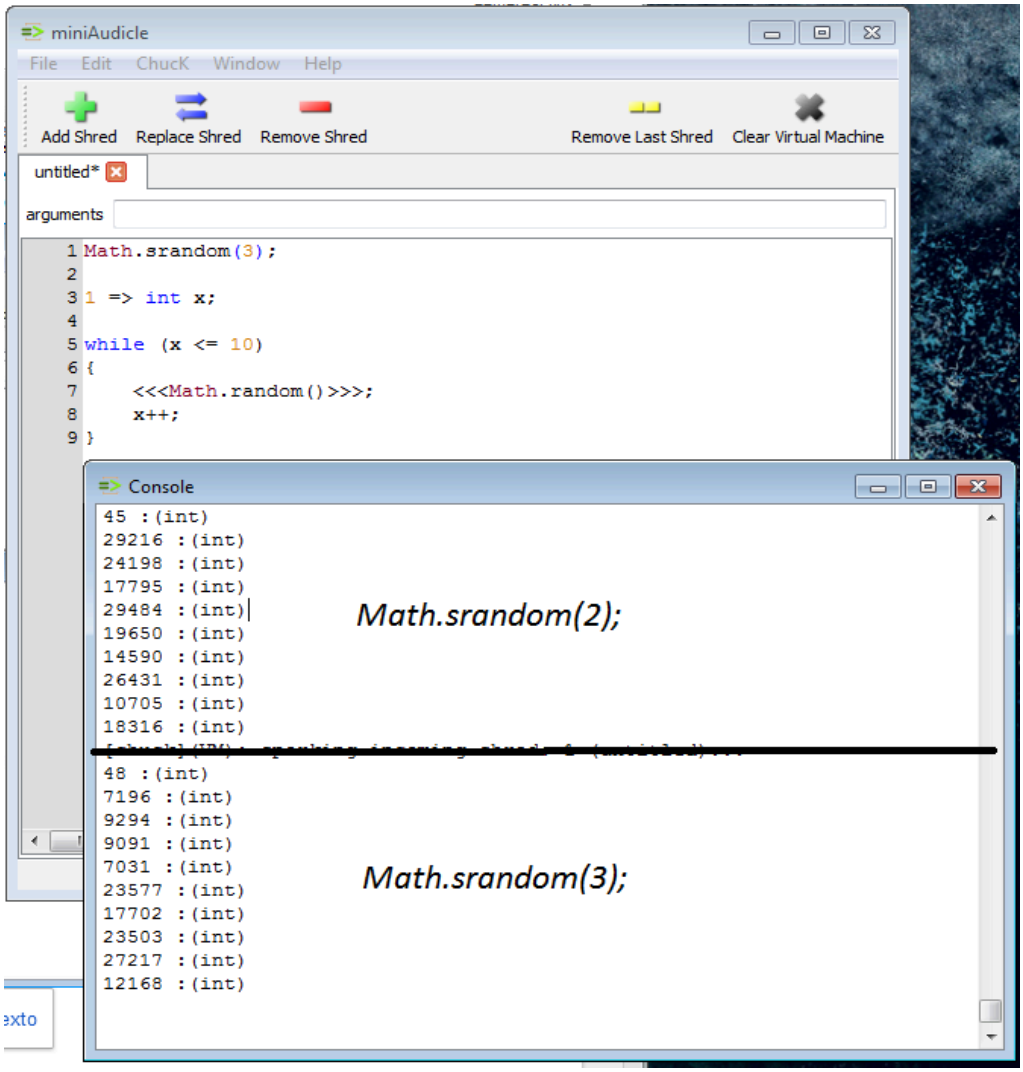
Para que funcione *.random* de cualquier tipo, éste se basa en un *seed* que es como un número tomado de alguna parte del universo para arrojar números aleatorios. Por ejemplo, si todo el tiempo usara tu edad (la mía, actual 19, por ejemplo) arrojaría números “al azar” basándose sólo en el 19. Si lo cambiáramos a 18, arrojaría otros.
Para controlar éso, ChucK cuenta con:

Math.srandom(int); //Específica un seed de donde se puede basar los random. Si, por ejemplo, en un while ponemos de seed 2, todas las veces que ejecutemos ese código saldrán los mismos números aleatorios. Por otro lado, si lo vamos cambiando, serán siempre "aleatorios":



En cualquier ejecución del código, los números "al azar" serán siempre los mismos. ¿Qué ironía, no?

Si cambiamos el *seed* a 3:



Por eso es recomendable andar cambiando el *seed* constantemente, para evitar mantener la rutina y generar, verdaderamente, ¡números pseudorandom!

Hay otro tipo de random que se llama *Math.sin();* que devuelve un número comprendido entre -1 y 1, por lo que, visualizando, es una oscilación. Lo veremos más adelante.

Panning:

El pan o panning es simplemente por dónde sale el sonido en las bocinas. O sea, si por el derecho o el izquierdo, en un sistema estéreo, o en más en uno surround.
Para modificar tal configuración en ChucK se escribe lo siguiente:

```
SinOsc sine => dac.right; //Asigno un SinOsc al speaker derecho.
SinOsc sine1 => dac.left; //Asigno otro pero al speaker izquierdo.
0 => int x;

while (x < 10)
{
   Math.random2(441/2,441) => sine.freq; //Establezco una frecuencia al azar.
   Math.random2(441/2,441)*2 => sine1.freq;
   //Hago lo mesmo pero lo multiplico por 2 para que suena diferente

    0.3 => sine.gain; //Volumen del sine.
    0.3 => sine1.gain; //del otro.
   500::ms => now;

   x++;
}
```

Así suena algo diferente en cada speaker (en estereo).

Más adelante haremos cosas con paneo mucho más interesantes :)

Declarar arreglos:

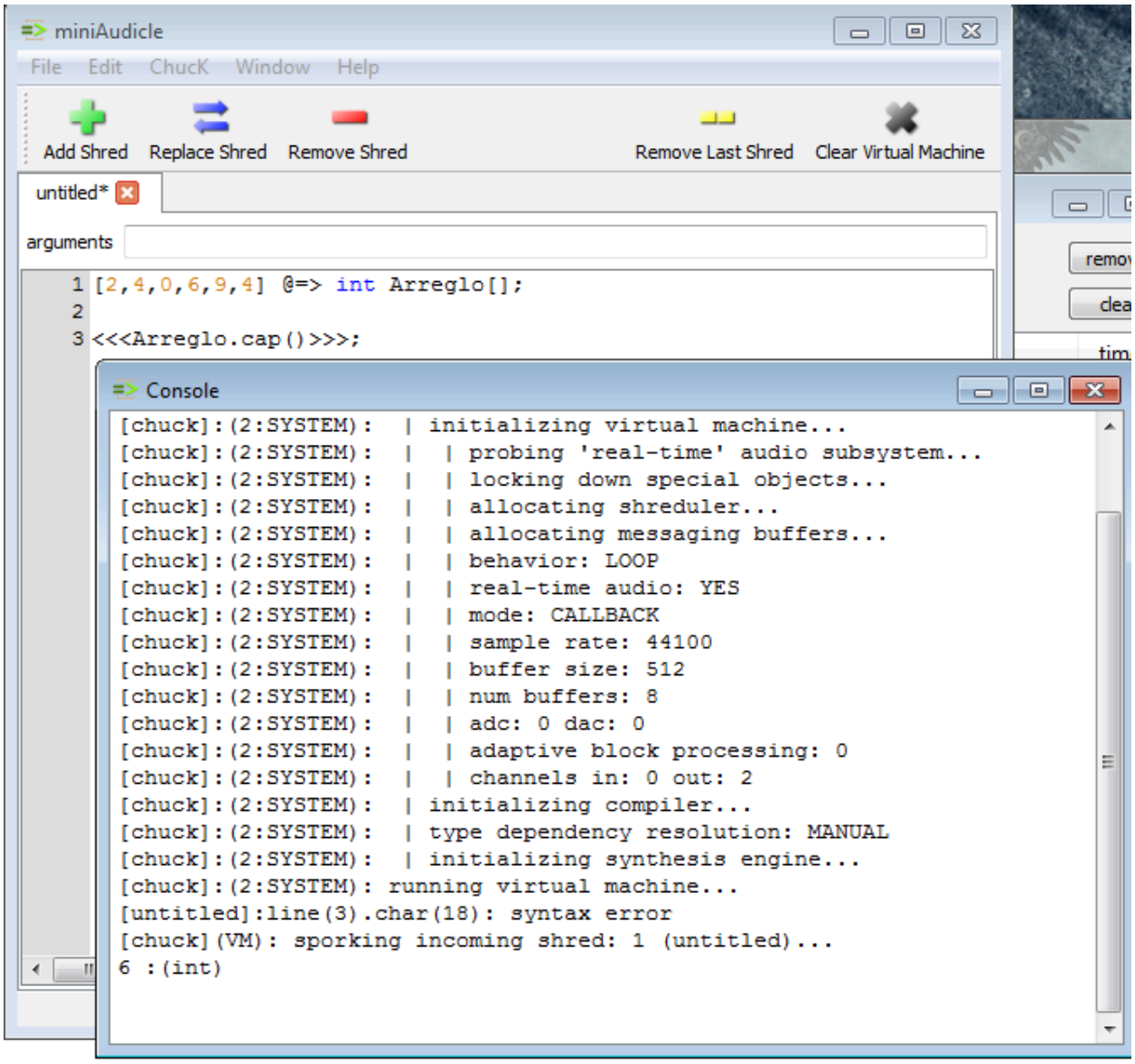
Un arreglo es básicamente como un almacén con cajas que contienen una variable. Por ejemplo, podemos tener un almacén que se llame *almacén* y dentro de ella tenemos, dentro de cajas, números enteros guardados, como por ejemplo un *2*, *3*, *4*, etc.
En ChucK, para hacer un arreglo se escribe lo siguiente:

```
[2,4,0,6,9,4] @=> int Arreglo[];
//Se escribe así, pues. Siempre se pone el @, porque sí. Y debemos tomar en cuenta qué tipo de variables ponemos dentro de los corchetes, ya queno podemos combinar enteros con
```

decimales (floats), pero sí decimales con enteros (ya que, técnicamente, un entero es un decimal). Por eso ponemos el int, o float, según sea el caso. Después el nombre que queramos del arreglo (en este caso "Arreglo") y le ponemos un par de corchetes, y ¡voila! Un arreglo en ChucK.

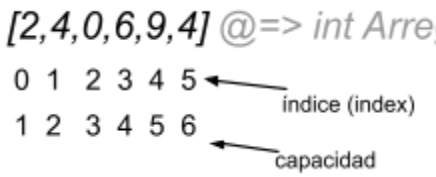
Tenemos varios métodos con los que jugar usando un arreglo. Uno de ellos es para conocer la capacidad:

Arreglo.cap(); //Regresa cuántos datos tiene dentro de sí.



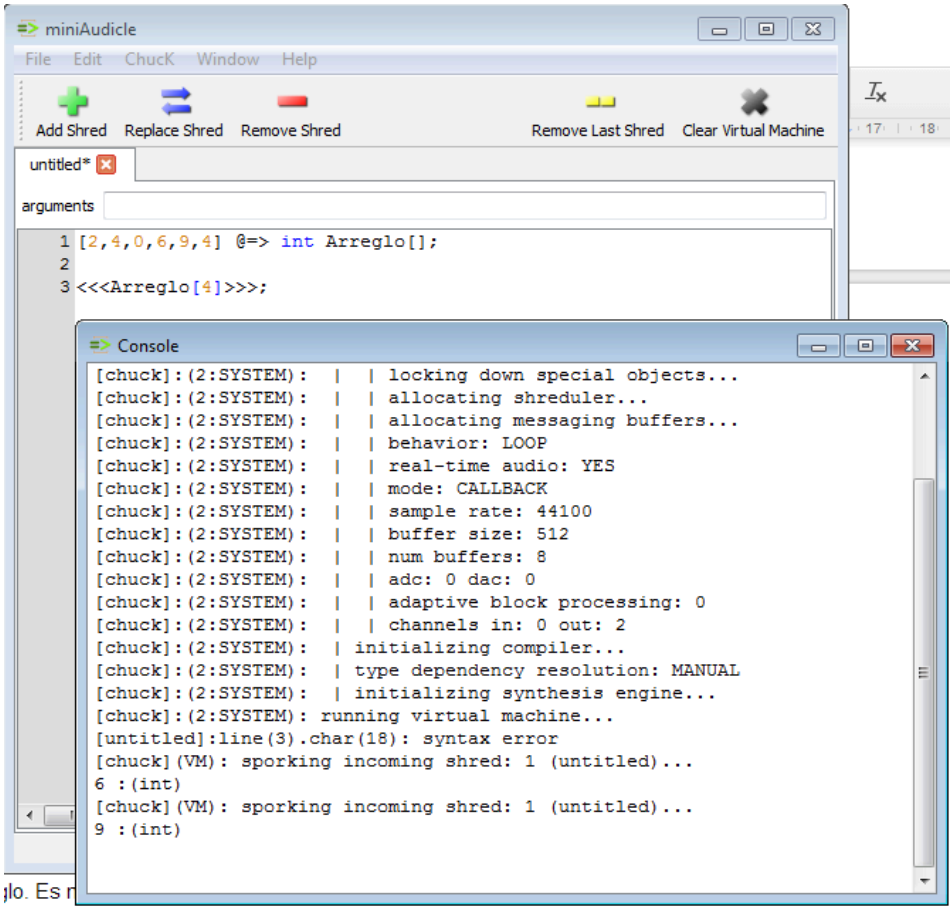
Podemos usarlo ampliamente con loops *for*, pero éso se verá más tarde ;)

Ahora, si quisiéramos saber dónde se encuentra un valor específico dentro de ese arreglo, ¿cómo le haríamos?



Como había mencionado, cada variable se guarda en una cajita. Para identificar la caja se usa el índice. Cabe resaltar que las cajas empiezan a numerarse desde 0, por lo que el valor *2* se encuentra en la caja *0*.
Regresando a nuestro problema, ¿qué pasa si yo quiero saber el valor de la caja 4 en mi arreglo. Es muy fácil:

<<<Arreglo[4]>>>;
//Ponemos el nombre del arreglo que queramos revisar y, dentro de los corchetes, el nombre de la caja (índice) que nos interesa. Ese valor se almacenará dentro de otro valor (si es que uno quiere).



Y así se aprenden algunos conceptos básicos de programación :^)