MUSC 208 Lab 1

Lab 1 will introduce you to the two programming environments we'll be using throughout MUSC 208: Octave and ChucK.

Introduction to Octave

Octave is a free open source software program for doing math, numerical analysis and Digital Signal Processing (DSP). Octave is mostly compatible with a very expensive commercial program called Matlab. It is also similar to Mathematica but easier to use for digital audio experiments. There are thousands of web pages devoted to learning Octave and Matlab.

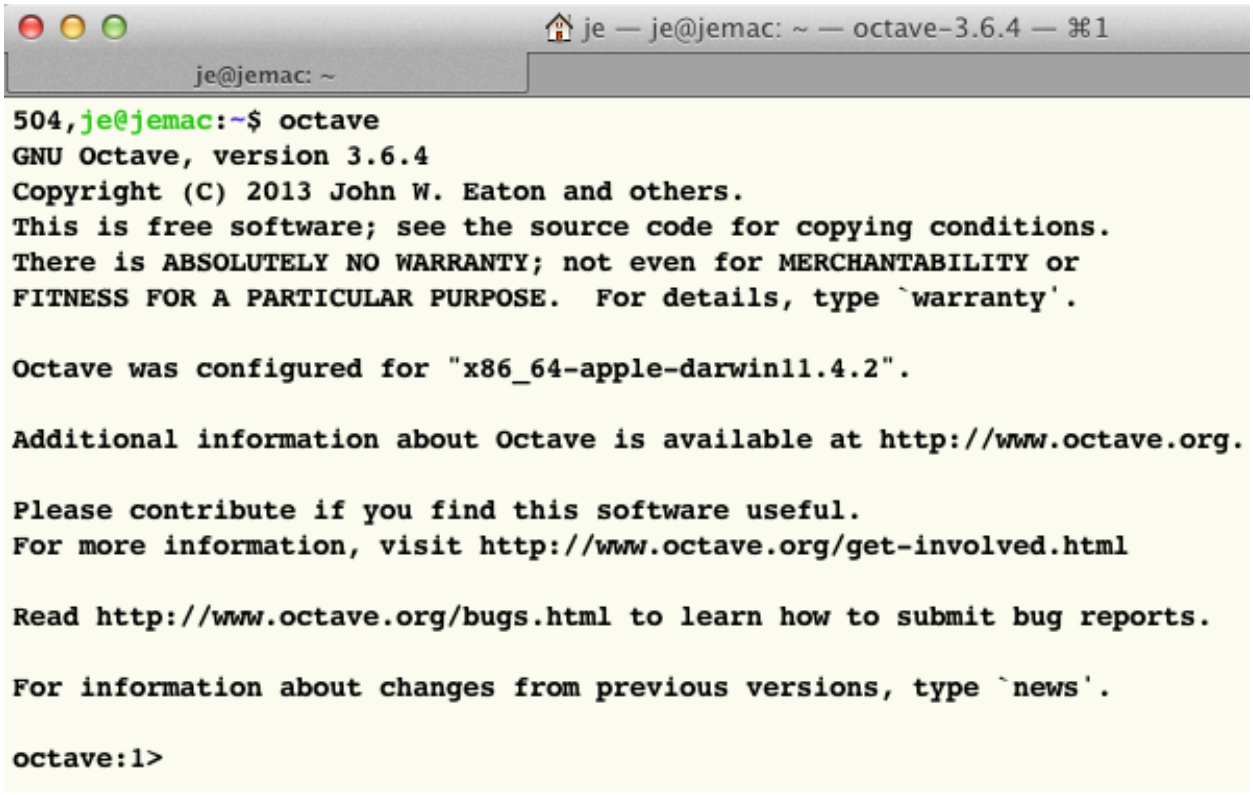
Every major audio editing program has DSP units that create and manipulate sounds. Later in this course you'll be using effects like compressors, equalizers, limiters, delay lines, and reverb. You'll have access to different types software synthesizers that use techniques like: additive synthesis, subtractive synthesis, ring modulation, frequency modulation, and physical modeling. All of these effects and synthesis techniques mathematically manipulate the individual samples in a sound file. Audio sound files often consist of several thousand to several million samples. Octave is especially good at this and will help you understand how several effects and synthesis techniques work.

You're going use Octave to mathematically generate a musical tone, plot it, and play it. But first you and Octave need to get acquainted.

Start Octave

Octave is run from the Terminal command line. Double click /Applications/ Utilities/Terminal. Type octave followed by Enter at the Terminal prompt. The octave prompt is "octave:1." The number after the colon is the command number for this octave session.

Terminal.app



You'll be greeted with octave information and the octave prompt >. Octave runs in a Terminal window as a command line program. You type commands following the Octave prompt and then press Enter to execute them. The first command will print Octave's working directory. The working directory is where Octave will reads and writes files.

pwd (Print Working Directory command)

In the following examples you should type and execute all octave code.

```
octave:1> pwd
ans = /Users/je/Sites/m208w2014_scrivener/_courseMaterials/octaveLabs_208
octave:2>
```

Your directory will be something like: /Users/labuser

Your working directory is where Octave will reads and writes files. Right now it's the lab user home folder.

VERY IMPORTANT: Memorize the pwd command. When things aren't working it's the first thing to check.

Basic Commands +, -, *, /

Addition, subtraction, multiplication, and division, work as you'd expect. Type this command and press return.

```
octave:41> 4*2+15/3-3
ans =  10
```

Spaces between the numbers are easier to read but optional.

Type the up arrow to recall the previous command, add spaces and type return.

```
octave:42> 4 * 2 + 15 / 3 - 3
ans =  10
```

Memorize This: Type the up and down arrows to recall your command history.

Operator Precedence

Multiplication and division take precedence over addition and subtraction as you'd expect. You can use parentheses to make it clear.

```
octave:43> (4 * 2) + (15 / 3) - 3
ans =  10
```

Try these commands.

```
octave:44> 4 * (2 + 15) / 3 - 3
ans =  19.667
octave:45> 4 * (2 + 15 / 3) - 3
ans =  25
octave:46> 4 * (2 + 15) / (3 - 3)
warning: division by zero
ans = Inf
```

^ The Power Operator

$2^{16} = 65536$

```
octave:1> 2^16
ans =  65536
```

Octave Comments

Text following a # symbol is treated as a comment and will be ignored.

```
octave:5> # the value of pi is
octave:5> pi
ans =  3.1416
octave:6> # the value of 2pi is
octave:6> 2*pi
ans =  6.2832
```

Creating Variables

You can assign and work with variables in octave the same way you do in algebra.

```
octave:51> x = 1234;
octave:52> y = 5678;
octave:53> z = x * y;
octave:54> z
z =  7006652
```

One Dimensional Arrays or Vectors

Create an array of consecutive numbers from 0 to 10.

```
octave:76> n = 0:10
n =

    0    1    2    3    4    5    6    7    8    9   10
```

Find the number of elements in the array.

```
octave:77> length(n)
ans =  11
```

Multiply every element in the array by 5 and assign the result to a new array.

```
octave:78> x = n*5;
octave:79> x
x =

    0    5   10   15   20   25   30   35   40   45   50
```

You can create the array x in one step using the following command

```
octave:4> # startNumber:increment:endNumber
octave:4> x = 0:5:50;
octave:5> x
x =

    0    5   10   15   20   25   30   35   40   45   50
```

You can access specific elements in the array using parenthesis, x(1) is the first element of the array x. Index 0 is undefined. Indices must be integers. Try these commands.

```
octave:7> n(1)
ans = 0
octave:8> n(0)
error: subscript indices must be either positive integers or logicals
octave:8> n(1.5)
error: subscript indices must be either positive integers or logicals
octave:17> z = n(3) + x(5)
z =  22
```

You can access sections of an array.

```
octave:18> x(5:8)
ans =

   20   25   30   35
```

You can convert a row vector into a column vector (append single quote).

```
octave:25> n'
ans =

    0
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
```

Surpressing Output

Memorize This: A semicolon at the end of a command will surpress the output.

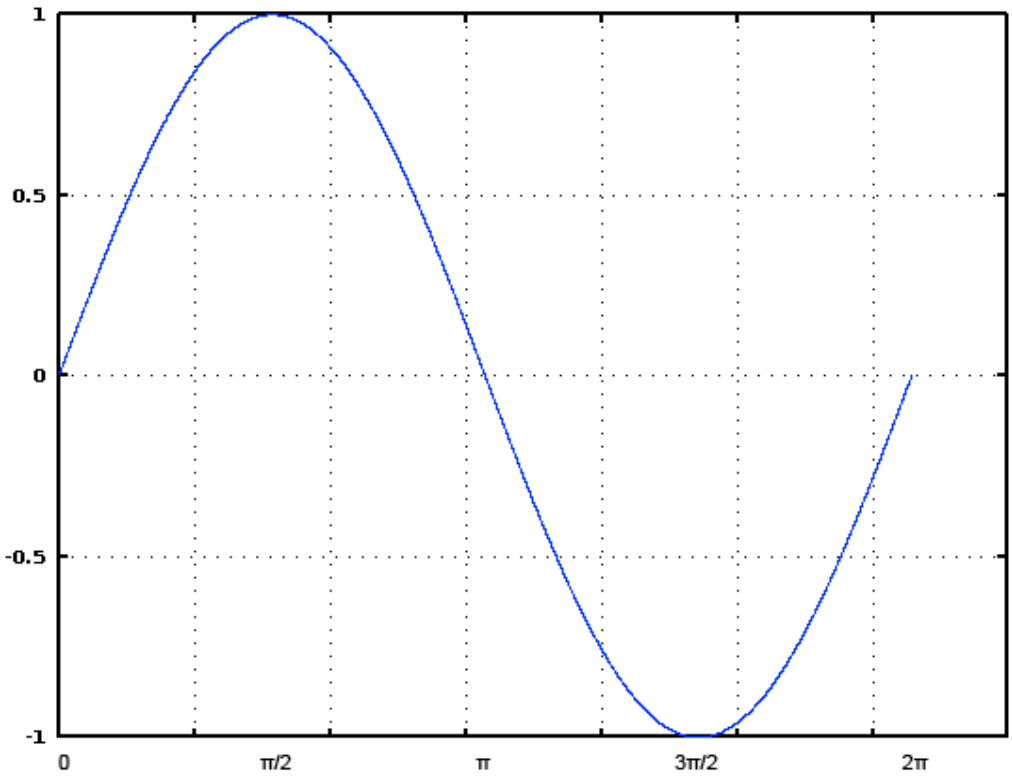
```
octave:14> zero_to_one_in_thousandths = 0 : 0.001 : 1;
octave:15> zero_to_one_in_thousandths(11:18)
ans =

   0.010000   0.011000   0.012000   0.013000   0.014000   0.015000   0.016000   0.017000
```

The General Sine Wave Formula

You may know (or will learn) that all musical sounds can be represented mathematically as a sum of sine waves of different amplitudes and phases that are integer multiples of a fundamental frequency. You've all seen this formula for a sine wave:

$$y(x) = \sin(x)$$

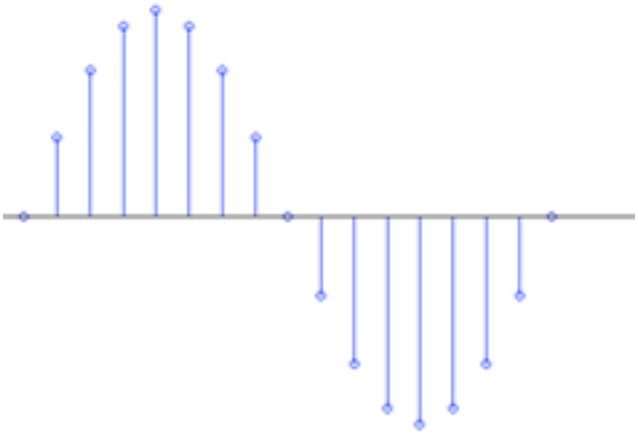
If x is a real number between 0 and 2π, the function y(x) is continuous and represents one period of a sine wave.



If that sine wave was a voltage sent to a speaker and was repeated 440 times a second, you'd hear hear it as the pitch A above middle C on the piano. That note is the modern standard of pitch and is known as "A 440" because its frequency is 440 Hz (Hz stands for Hertz and is the modern term for cycles per second).

Analog / Digital Signals

Analog (electronic voltage) signals are continuous in the mathematical sense that a voltage value exist at every moment of time. Digital signals are discrete and only exist as sample values taken at a uniform sampling rate. Sampled waveforms are often represented as "lollipop" graphs where stems represent the sample rate and small circles represent the sample values. You might think the following graph represents a smooth sine wave but you can't be certain because you don't know what happened in between the samples.



Sample Rate / Sample Period

The sample rate is the number of samples taken per second.
The sample period is the number of seconds between samples.

$$SamplePeriod = \frac{1}{SampleRate}$$

If you were told that the sampling rate used in the above graph was 16 samples per second you'd observe that one period of the wave form was completed in one second resulting in a frequency of 1 Hz.

Frequency / Period

If you were told that the sample period used in the above graph was 0.25 milliseconds (250 microseconds), then you'd observe that it took 16 * 0.25 = 4 milliseconds to complete one period. A 4 ms period represents a frequency of 250 Hz.

$$frequency = \frac{1}{period}$$

```
octave:26> 1 / .004
ans =  250
octave:27> 1/4e-3
ans =  250
```

The Audio CD Sample Rate and Sample Period

The audio CD sample rate is 44100 samples per second.

The audio CD sample period is 22.6757 microseconds (0.0000226757 seconds).

The Sampled Sine Wave Formula

We're now ready to create a sine wave tone in octave using this formula.

$$y(n) = A \sin(2\pi f \frac{n}{SR} + \theta) \text{ or } y(n) = A \sin(2\pi f n T + \theta)$$

n is an integer index representing the sample number;
A is the amplitude;
f is the frequency (2πf is the frequency in radians);
SR is the sample rate (samples/second);
T = 1/SR is the sample period (seconds/sample);
θ is the starting phase in radians (0-2π radians).

Complete this code in Octave

Assume the amplitude is 1 and the phase is 0. With those assumptions the formula simplifies to;

$$y(n) = \sin(2\pi f n T)$$

Enter this code in Octave. The octave prompt has been omitted from each line.

```
# sample rate
SR = 44100;

# sample period
T = 1/SR;

# frequency is 440 Hz
f = 440;

# duration is 1 second
dur = 1;

# create 2 seconds worth of samples
n = 0 : (SR * dur) - 1;

# implement the formula
wav = sin( 2 * pi * f * n * T );
```

Let's look at the sample values.

```
octave:4> wav
wav =

 Columns 1 through 9:

   0.00000   0.03727   0.07448   0.11159   0.14855   0.18530   0.22180   0.25798   0.29381

 Columns 10 through 18:

   0.32923   0.36419   0.39864   0.43254   0.46584   0.49850   0.53046   0.56168   0.59212

 Columns 19 through 27:

   0.62174   0.65050   0.67835   0.70526   0.73119   0.75611   0.77997   0.80275   0.82442

 Columns 28 through 36:

   0.84494   0.86428   0.88243   0.89935   0.91502   0.92942   0.94252   0.95432   0.96479

 Columns 37 through 45:

   0.97393   0.98170   0.98812   0.99316   0.99682   0.99910   0.99999   0.99949   0.99760

 Columns 46 through 54:

   0.99432   0.98966   0.98363   0.97623   0.96748   0.95738   0.94595   0.93321   0.91917

 Columns 55 through 63:

   0.90385   0.88728   0.86948   0.85046   0.83027   0.80892   0.78645   0.76288   0.73826

 Columns 64 through 72:

   0.71261   0.68597   0.65837   0.62987   0.60048   0.57026   0.53925   0.50750   0.47503

 Columns 73 through 81:

   0.44191   0.40817   0.37386   0.33904   0.30374   0.26803   0.23194   0.19552   0.15884

 Columns 82 through 90:

   0.12194   0.08486   0.04767   0.01041  -0.02686  -0.06410  -0.10124  -0.13825  -0.17506

 Columns 91 through 99:

  -0.21163  -0.24791  -0.28384  -0.31938  -0.35447  -0.38907  -0.42313  -0.45660  -0.48944

 Columns 100 through 108:
-- less -- (f)orward, (b)ack, (q)uit
```

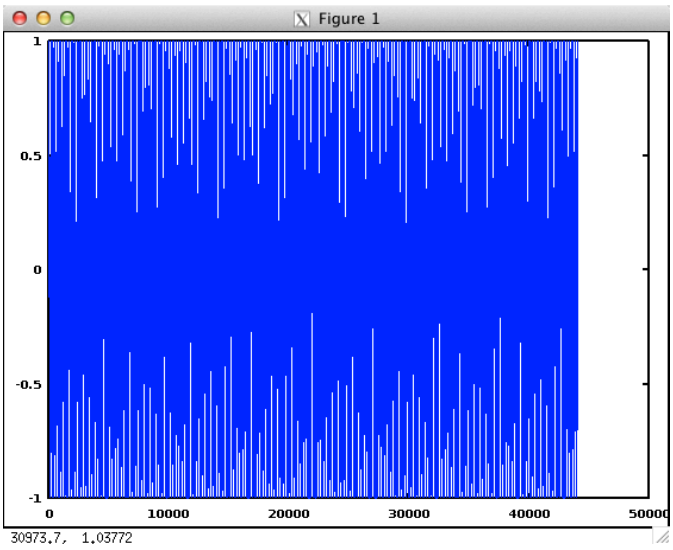
You could type f (or space bar) to see the next page, b for the previous page, or q to exit. You probably don't want to see all 44100 sample values. Type q.

Octave Plots

Type this command.

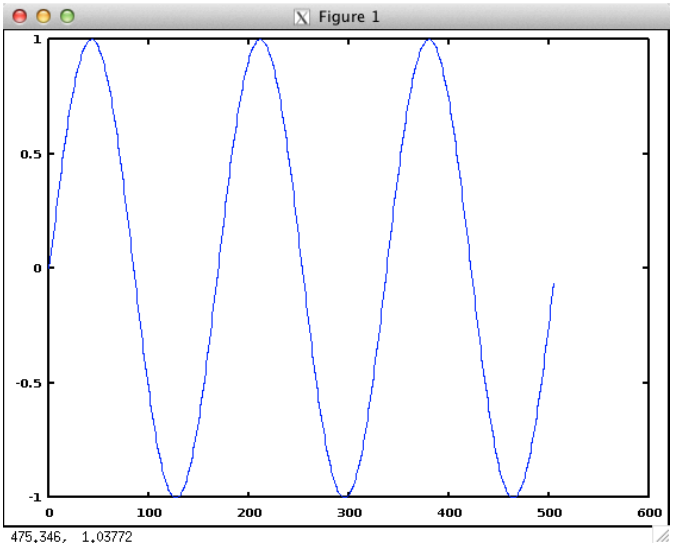
```
octave:7> plot( wav );
```

You should see a graph of all 44100 samples appear.

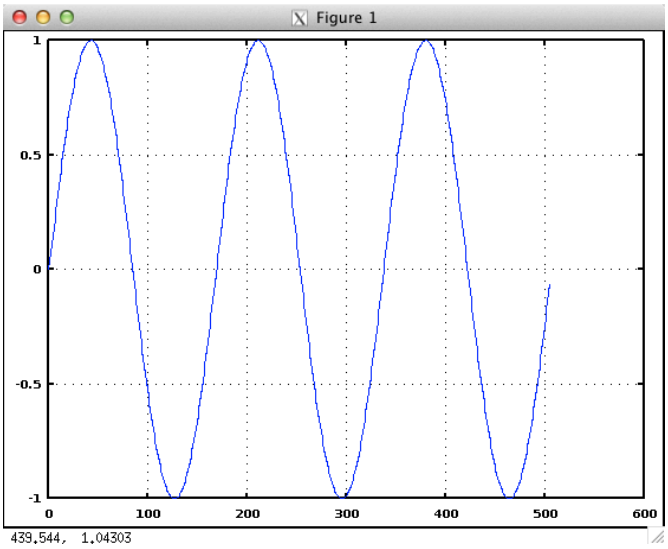


Let's plot the first three periods of this sine wave.

```
octave:10> numberOfSamplesInOnePeriod = SR / f;
octave:11> plot( wav( 1 : numberOfSamplesInOnePeriod * 3 ) );
```



```
octave:12> grid on
```



playsamples

I wrote the playsamples code and added to the Lab version of Octave. Try it.

```
octave:14> playsamples( wav );
```

You should have heard a one second A440 sine tone.

If you want to see what the play samples code looks like type

```
octave:15> edit playsamples.m
```

The code should open in TextWrangler.

You'll learn how to write your own Octave code in the next lab.

ChucK (MiniAudicle Interface)

ChucK is a concurrent, strongly timed audio programming language for real-time synthesis, composition, and performance,[2] which runs on Mac OS X, Linux, Microsoft Windows, and iPhone/iPad. It is designed to favor readability and flexibility for the programmer over other considerations such as raw performance. It natively supports deterministic concurrency and multiple, simultaneous, dynamic control rates. Another key feature is the ability to live code; adding, removing, and modifying code on the fly, while the program is running, without stopping or restarting. It has a highly precise timing/concurrency model, allowing for arbitrarily fine granularity. It offers composers and researchers a powerful and flexible programming tool for building and experimenting with complex audio synthesis programs, and real-time interactive control. http://en.wikipedia.org/wiki/ChucK

ChucK is a general-purpose programming language, intended for real-time audio synthesis and graphics/multimedia programming. It introduces a truly concurrent programming model that embeds timing directly in the program flow (we call this strongly-timed). Other potentially useful features include the ability to write/change programs on-the-fly.

ChucK's programming model provides programmers direct, precise, and readable control over time, durations, rates, and just about anything else involving time. This makes ChucK a potentially fun and highly flexible tools for describing, designing, and implementing sound synthesis and music-making at both low and high levels.http://chuck.cs.princeton.edu/release/files/chuck_manual.pdf

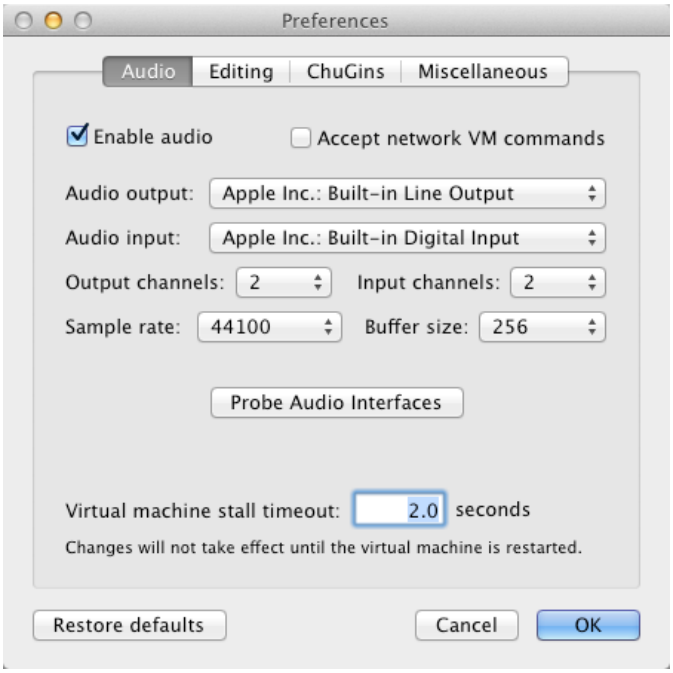
Play the A-440 Sine Wave in ChucK

Open /Applications/MiniAudicle. MiniAudicle is a GUI (graphical user interface) to the ChucK audio programming language.

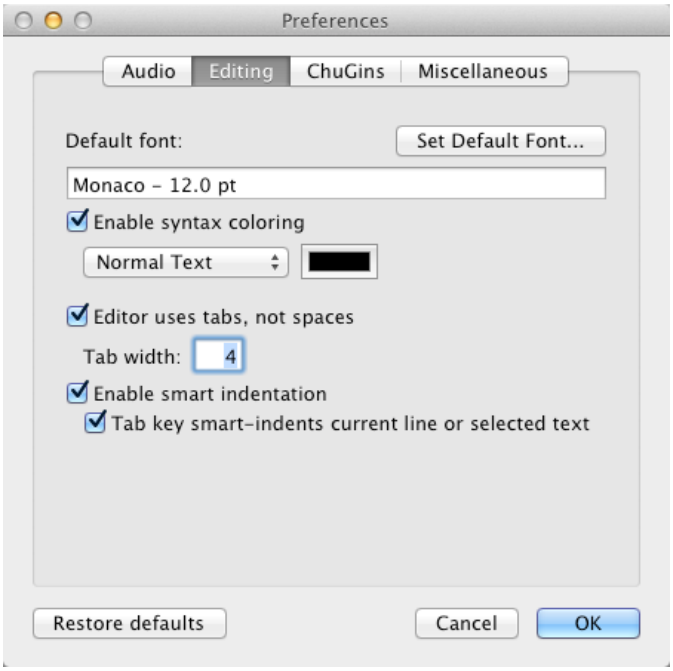
Two windows should appear: the main Untitled window and the Virtual Machine Monitor window.

ChucK Preferences

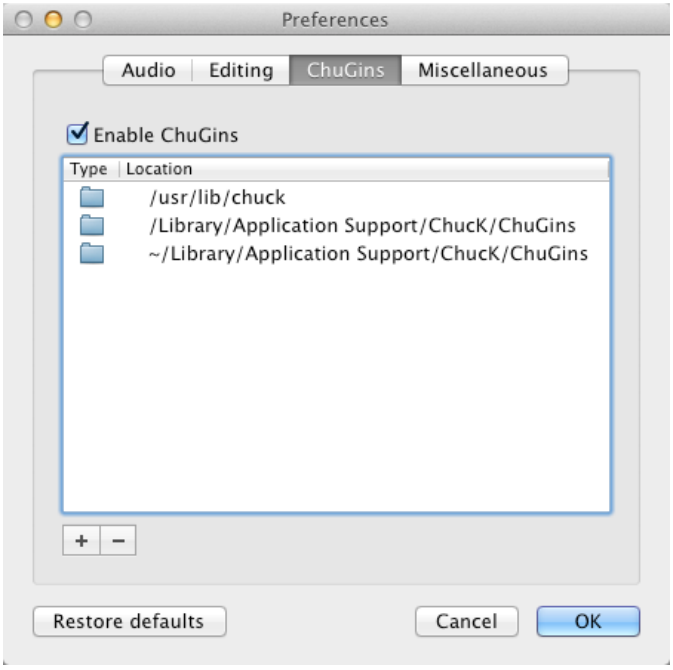
Open Preferences from the miniAudicle menu. You'll see four tabs. The lab setting may not be exactly like these pictures, but they should be similar. The most important one is tab 4 Miscellaneous.



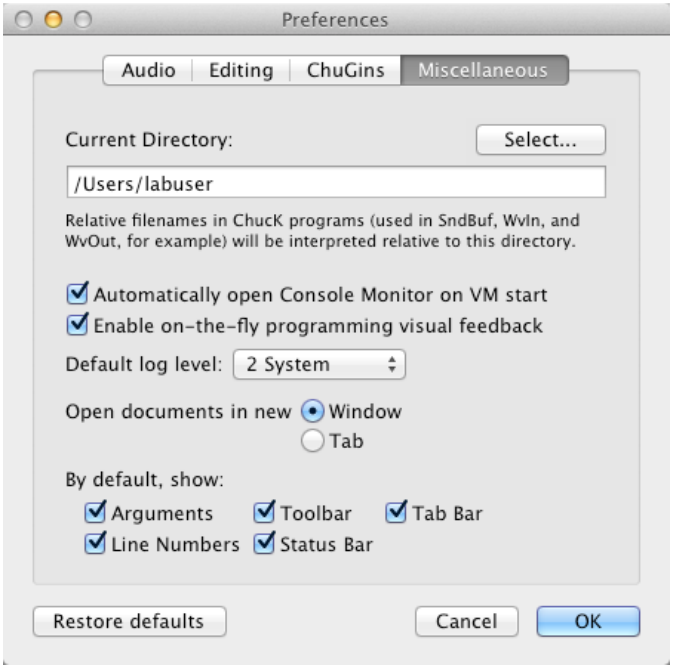
Set to taste. Not saved across restarts in the lab.



Should not change for MUSC 208.

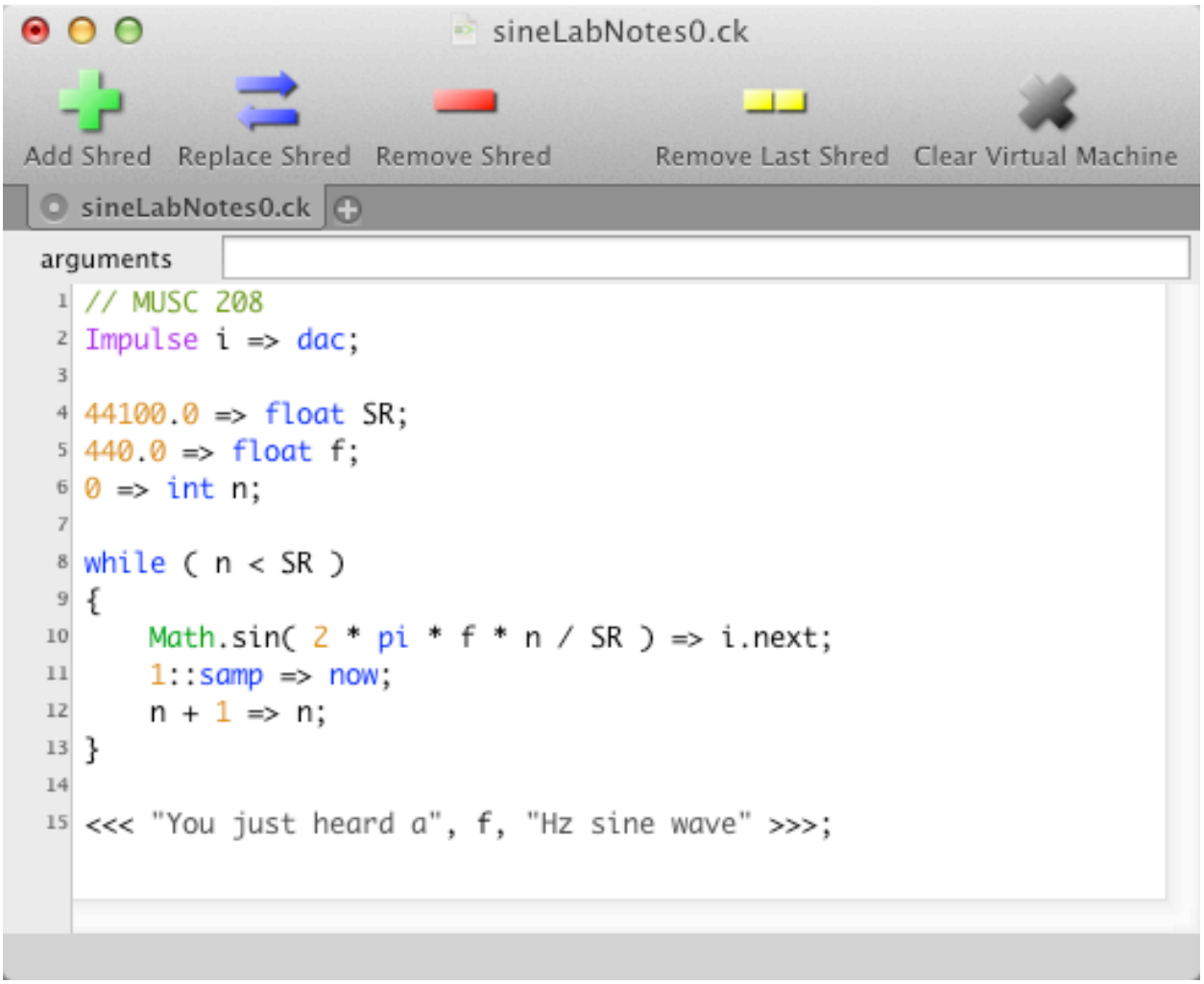


The Miscellaneous tab is important. If you're having problems with miniAudicle check that the Current Directory is set correctly.



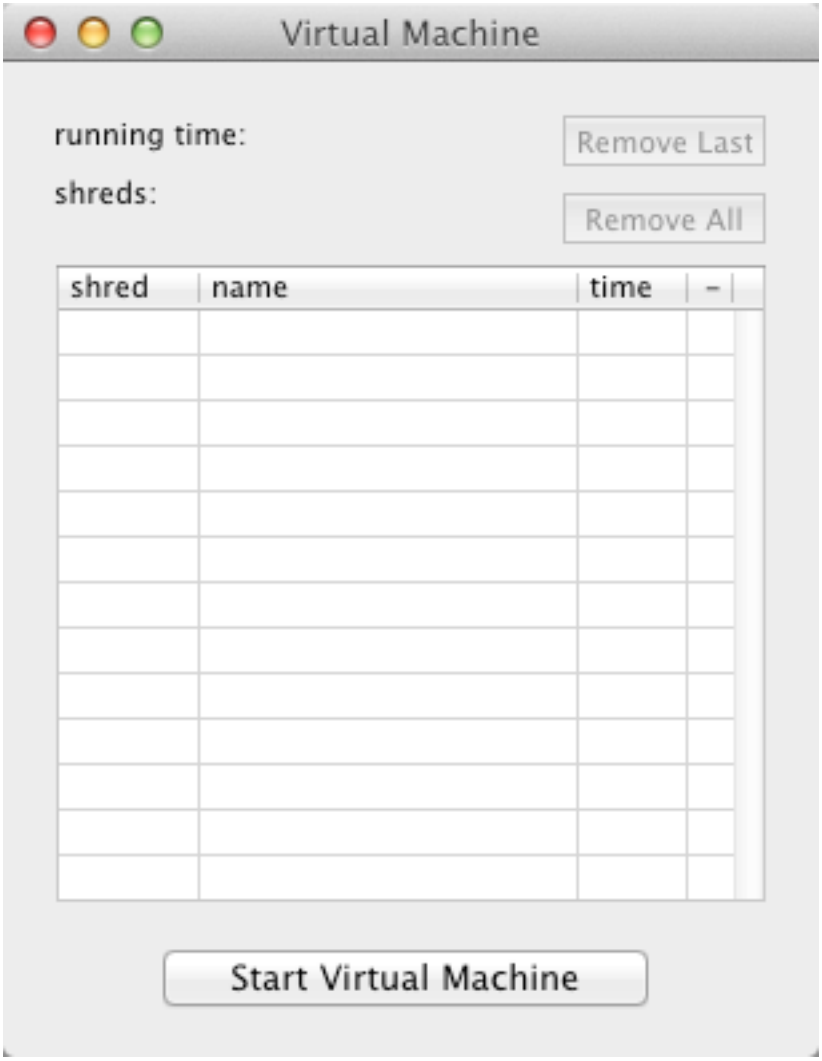
Generate a Sine Wave in ChucK

Type these lines in the miniAudicle window.

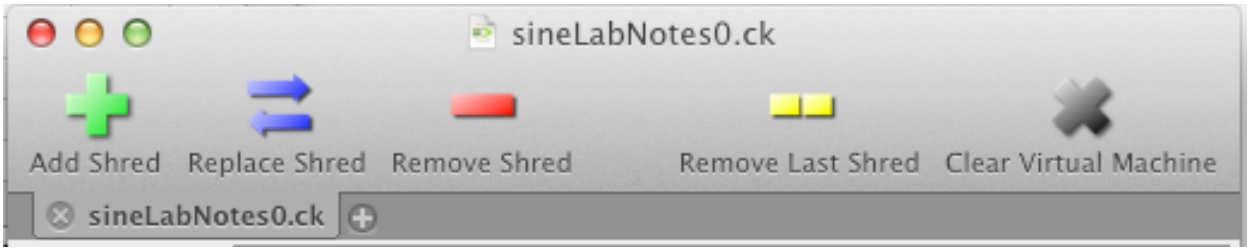

sineLabNotes0.ck
Add Shred
Replace Shred
Remove Shred
Remove Last Shred
Clear Virtual Machine
sineLabNotes0.ck
arguments
1 // MUSC 208
2 Impulse i => dac;
3
4 44100.0 => float SR;
5 440.0 => float f;
6 0 => int n;
7
8 while (n < SR)
9 {
10 Math.sin(2 * pi * f * n / SR) => i.next;
11 1::samp => now;
12 n + 1 => n;
13 }
14
15 <<< "You just heard a", f, "Hz sine wave" >>>;

Run the Program

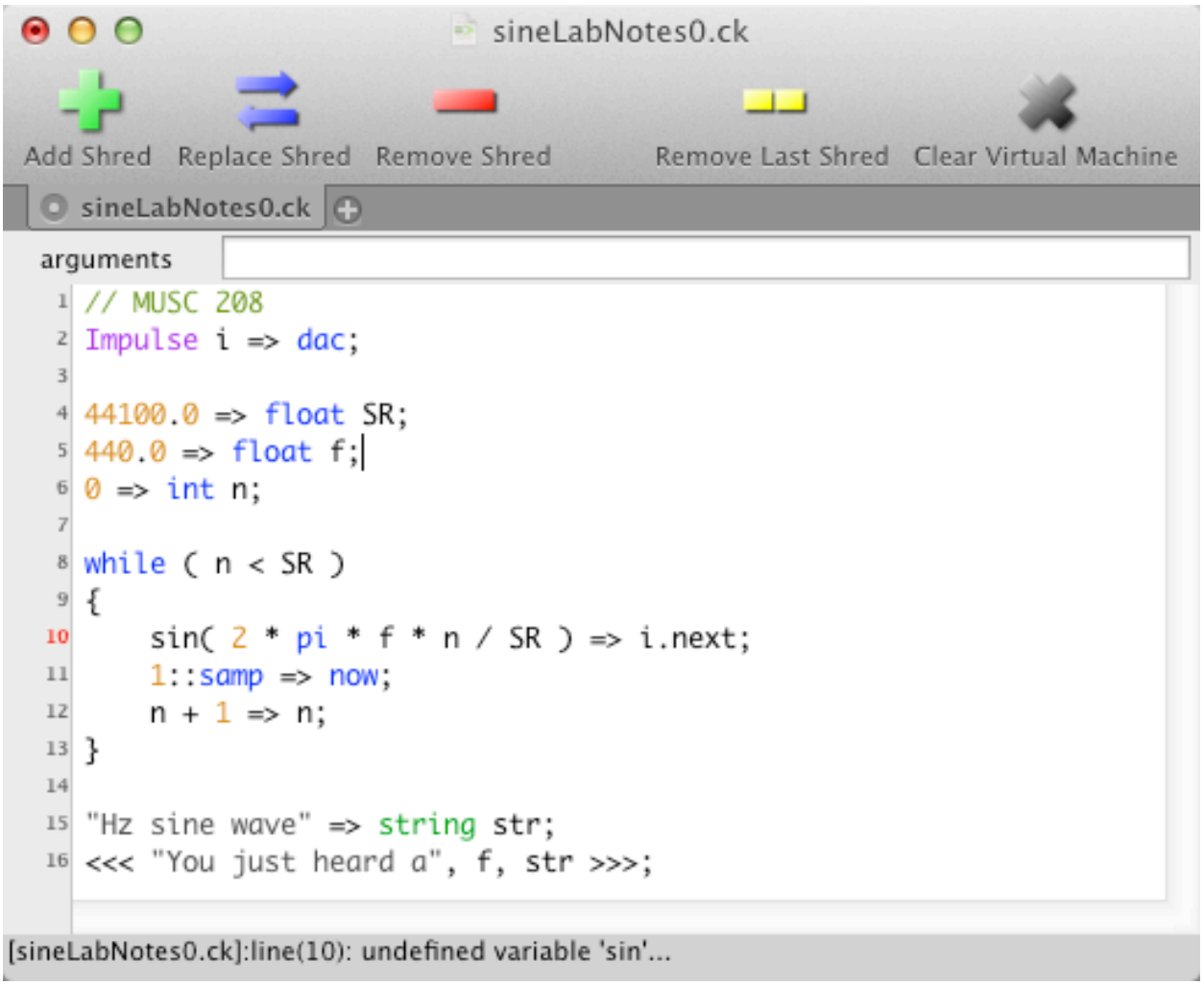
1. Start ChucK by clicking the Start Virtual Machine button in the Virtual Machine window.



2. Click the green "Add Shred" button. If there are no typing errors you'll hear the sine wave.



If there are errors, MiniAudicle points them out by displaying line number where the error occurred in red (line 10) and also displays an error message at the bottom of the window. Here the Math. library prefix is missing. When the program is error free you'll hear the A440 sine tone.



Syntax Coloring

// comments are olive green

numbers are orange

reserved words are blue. They are part of the ChucK programming language and cannot be used to name your variables.

ChucK library function classes (Math.) are bright green. The sin() function is part of the Math library. Libraries contain many useful utility functions that you

can use. The only way you can learn about them is by reading the ChucK documentation.

strings (text enclosed in quotes) are gray;

<<< … >>> whatever is enclosed inside the triple angle brackets will be printed to the Console Monitor window. In this example three items separated by commas will be printed <<< literal string, float variable, string variable >>>.

DSP Terminology

Many ChucK terms come straight out of DSP (Digital Signal Processing) theory.

Impulse

An impulse is a single sample.

DAC

DAC refers to a Digital to Analog Converter or something that converts a discrete digital signal (numbers) to a continuous analog signal (voltage). Here it's just a fancy name for the speaker.

ADC

An ADC refers to an Analog to Digital Converter or something that converts a continuous analog signal (voltage) to a discrete digital signal (numbers). It's what happens you record something into the computer or talk on your iPhone.

float and int

Two types of numbers are used in ChucK, a float (short for floating point) and an int (short for integer). A float is any number that contains decimal places. An int is a simple integer (no decimal places).

Line by Line Description of the Code

=> is the ChucK operator. According to the manual it is derived from the phrase "to chuck something" as in to throw it at.

Line 1 is a comment.

Line 2 defines an Impulse (a single sample), gives it the variable name i, and "chucks" it to the to the dac (speaker).

Line 4 "chucks" the number 44100.0 to the variable SR of type float.
Line "chucks" the number 0 the integer variable n.

Line8 sets up a "while loop" that repeats all lines inside the { } curly brackets (lines 10-12) as long as the value of n (the current sample number) is less than SR (44100).

Line 10 calculates a single sample value of the sine wave and chucks it to i.next. i.next is stored in a special audio memory holding area waiting to be played.

Line 11 chucks time to now. The word **now** is arguable the most important word in the ChucK language. It allows audio to be processed and heard. In this case one sample (1::samp) worth of time, and i.next which has just been sitting there will be sent to dac (the speaker). When 22+ microseconds (1 sample) have passed ChucK suspends audio processing but continues to run the program.

Line 12 adds 1 to the value of n and returns to line 10 and the loop is repeated. When n is greater than or equal to 44100 the loop exits and processing continues until the end of the program.

Line 15 chucks a string, "Hz sine wave" to a variable called str.

Line 16 prints everything inside the triple angle brackets to the Console Monitor window.

ChucK Help

ChucK help is available online at http://chuck.cs.princeton.edu/ and its mirror site http://chuck.stanford.edu/. The author of the ChucK language, Ge Wang, developed the program while working on his PhD at princeton and now teaches at Stanford. ChucK first appeard in 2004.

Here's help for Impulse from http://chuck.cs.princeton.edu/doc/program/ugen.html.

[ugen]: **Impulse**

- *pulse generator - can set the value of the current sample*
- *default for each sample is 0 if not set*

(control parameters)

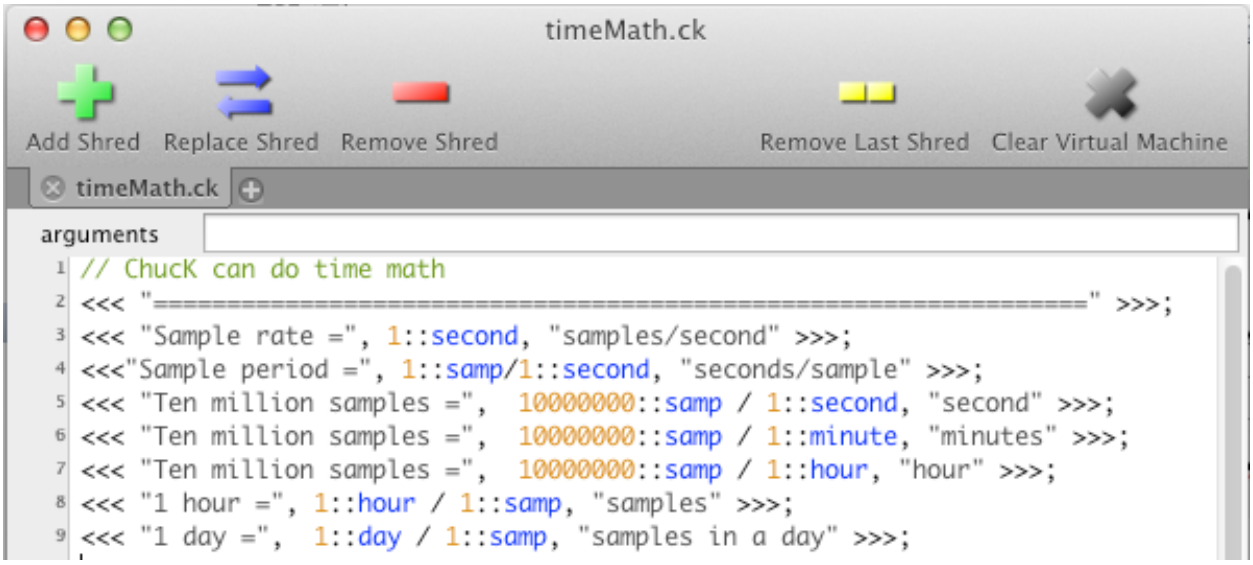
- **.next** - (float , READ/WRITE) - *set value of next sample to be generated. (note: if you are using the **UGen.last** method to read the output of the impulse, the value set by **Impulse.next** does not appear as the output until after the next sample boundary. In this case, there is a consistent 1::samp offset between setting .next and reading that value using .last)*

Time Math

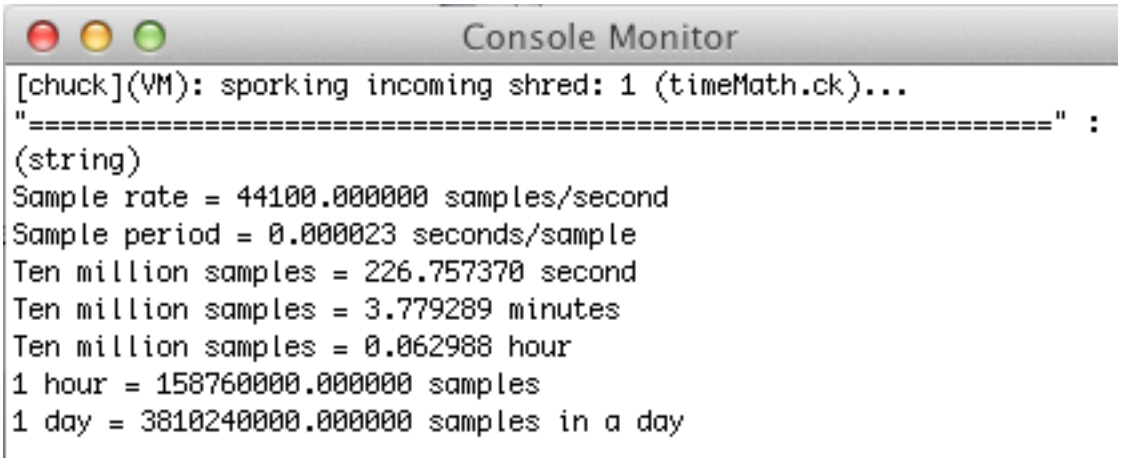
One of the big strengths of ChucK for real time audio performance is its ability to accurately process time by defining precisely when and for how long audio will be heard, time chucked to now. Chuck can processes time in fractions of a sample, samples, milliseconds, seconds, minutes, hours, and days.

ChucK uses a double colon to represent quantities of time, 1::samp, 1::ms, 1::second, 1::minute, 1::hour, 1::day.

ChucK also makes it easy to do time math. Create a new miniAudicle window and enter and execute this code.

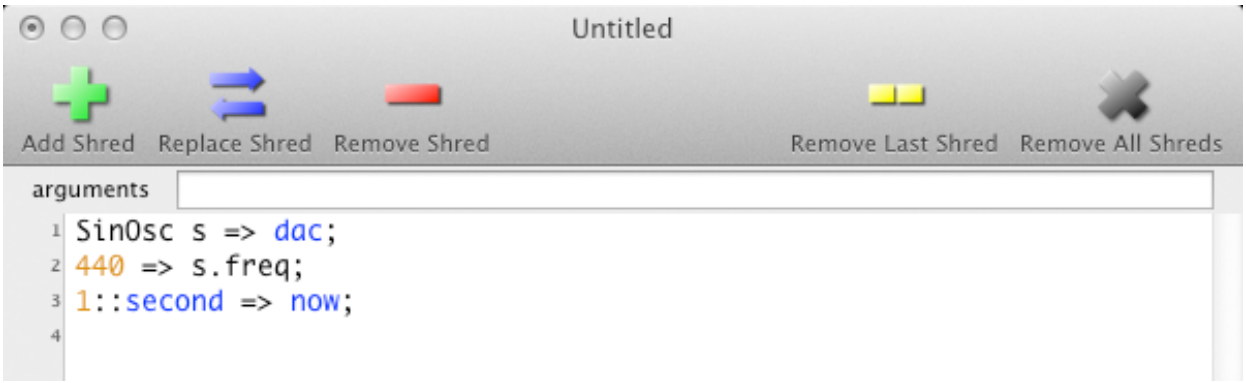


The results will appear in the Console Monitor window.



The Easy Way To Play A Sine Wave In Chuck

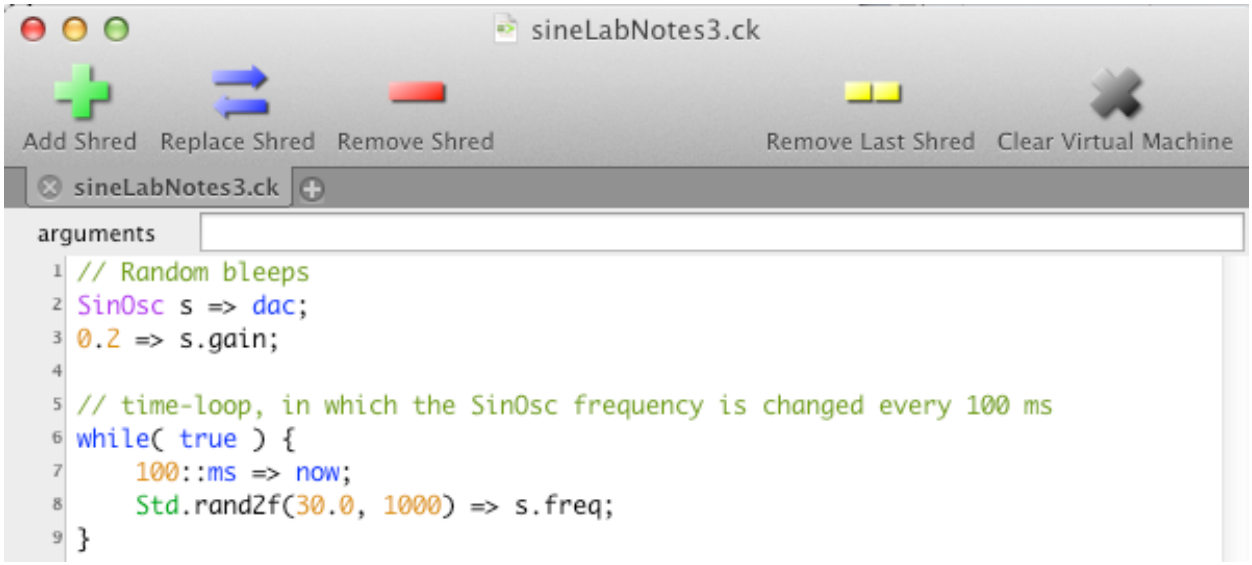
You can do it in three lines. You'll hear a 440 Hz sine wave play for one second at maximum amplitude.



Let's turn down the gain (amplitude). Insert this new (line 3) and click the Add Shred button.

```
1 SinOsc s => dac;
2 440 => s.freq;
3 0.1 => s.gain;
4 1::second => now;
```

Computer Bleeps from Movies in the 50's and 60's



The loop will run forever. Click Remove Shred to stop.

Line 1. a comment

Line 6. Sets up a "while loop" that never ends.

Line 8. The library function Std.rand2f(float min, float max) will generate a random number in the range min..max.

While the program is running change the min max parameters to Std.rand2f and/or the number of ms chucked to now and click the "Add Shred" button. Multiple shreds (threads in computer science terminology) will run in parallel.

Experiment with the other Shred buttons. Their names are descriptive of what you'll hear.

Music 208 Winter 2014
John Ellinger
Carleton College