Functions

A function is a grouping of operations to be performed w
'value' message.

Function definitions are enclosed in curly brackets { }.
Argument declarations, if any, follow the open curly brac
Variable declarations follow argument declarations.
Expressions follow the declarations.

Example:

```
{ arg a, b, c;
    var d;
    d = a * b;
    c + d
}
```

Functions are not evaluated immediately when they occu
passed as objects just like integers or strings.

A function is evaluated by passing it the value messa
arguments.

When evaluated, the function returns the value of its

```
f = { arg a, b; a + b };
f.value(4, 5).postln;
f.value(10, 200).postln;
```

(

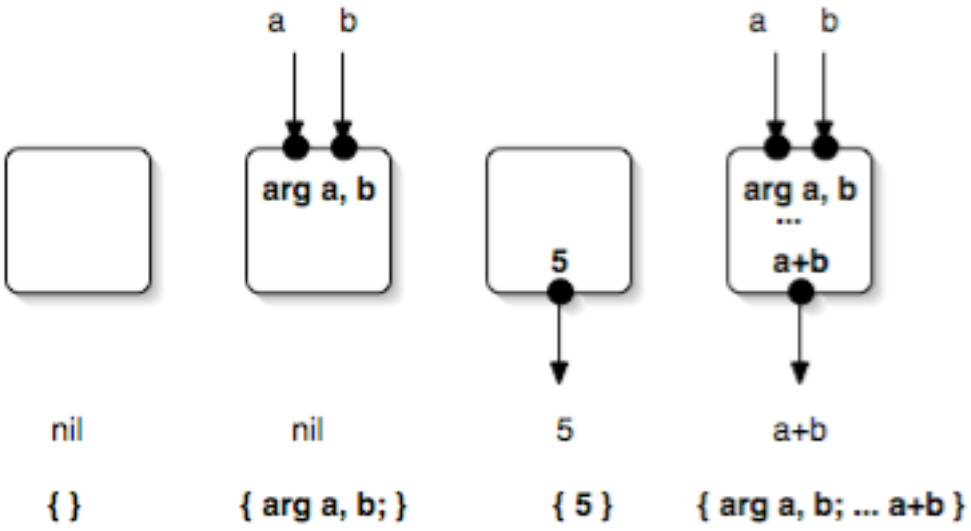
```
    var pitchClass;
    pitchClass = { arg pc;
        pc%12};                    // modulo operation with Simp

    pitchClass.value(4).postln;
    pitchClass.value(45).postln;
    pitchClass.value(60).postln;
)
```

An empty function returns the value nil when evaluated.

```
{  }.value.postln;
```

A function can be thought as a machine able to perform
e.g. a calculator. The calculator can receive input (args)
value, the result of the performed operations. The functi
then be thought as the building of the calculator: once bu
does nothing until a user requests it to work (by passing
to a function).

The following figure depicts an empty function, input with
without input, and then the general case with input and o



Functions

Often, sections of code are enclosed in curly braces (ma
as part of large-scale program organization. This sectio
the properties of Function (such as the abilty to accept th

```
{ SinOsc.ar(800, mul: 0.2) }.play;
```

Arguments

An argument list immediately follows the open curly brac
definition. An argument list begins with the reserved wor
takes no arguments, then the argument list may be omitt
arguments in the list may be initialized to a default literal
equals sign. Arguments which are not explicitly initialized
no value is passed for them.

examples:

```
arg a, b, c=3;
```

arg x='stop', y, z=0;

In general arguments may be initialized to literals or exp
case of Function-play (or SynthDef-play), they may only
literals.

```
// this is okay:
{arg a = Array.geom(4, 100, 3); a * 4 }.value;

// this is not:
(
    {arg freq = Array.geom(4, 100, 3);
        Mix(SinOsc.ar(freq, 0, 0.1))
    }.play; // silence
)

// but this is:
(
    SynthDef(\freqs, { arg freq = #[ 100, 300, 900,
        Out.ar(0, Mix(SinOsc.ar(freq, 0, 0.1)));
    }).play;
)
```

Alternative Argument Syntax

Functions and methods have an alternate style for decla
The arguments are placed between vertical bars such as

{|x=1, y=2, z=3| x + y + z }

is the same as:

```
{ arg x=1, y=2, z=3; x + y + z }
```

The new syntax can make some expressions more conc

Variables

Following the argument declarations are the variable dec
may be declared in any order. Variable lists are preceed
word **var**. There can be multiple var declaration lists if ne
may be initialized to default values in the same way as a

examples:

```
var level=0, slope=1, curve=1;
```

Operations on Functions

The superclass of Function is AbstractFunction. E
the operations that can be performed with numbers
performed on AbstractFunctions and on Functions

Example Function

```
(
    var freqFunc, pitches, midiNote;
```

```
    pitches = [60, 61, 62, 63, 64]; //declare an array of pi

    freqFunc = {
        midiNote = pitches.choose; //pick a pitch from the
         midiNote.midicps;              // return the cps for th
    };

    freqFunc.round.value
)
```

Also, more importantly

```
(
    var a, b, c;
    a = { [100, 200, 300].choose };  // a Function
    b = { 10.rand + 1 };    // another Function
    c = a + b;  // c is a Function.
    c.value.postln;     // evaluate c and print the result
)
```

Some Instance Methods

value(...args)

Evaluates the FunctionDef referred to by the Function. T
passed the args given.

```
{ arg a, b; (a * b).postln }.value(3, 10);
```

defer(delta)

Delay the evaluation of this Function by **delta** in seconds. U

```
{ "2 seconds have passed.".postln; }.defer(2);
```

Looking Ahead: Audio Methods

play(target, outbus, fadetime, addAction)

This is probably the simplest way to get audio in SC3. It
in a **SynthDef** (adding an **Out** ugen if needed), creates
Synth with it, and returns the Synth object. A **Linen** is a
clicks, which is configured to allow the resulting Synth to
argument set, or to respond to a release message. Args
become args in the resulting def.

target - a Node, Server, or Nil. A Server will be converte
group of that server. Nil will be converted to the defa
default Server.
outbus - the output bus to play the audio out on. This is
Out.ar(outbus, theoutput). The default is 0.
fadeTime - a fadein time. The default is 0.02 seconds, v
to avoid a click. This will also be the fadeout time for
not specify.
addAction - see **Synth** for a list of valid addActions. Th
\addToHead.

```
{ SinOsc.ar(440, 0, 0.3) }.play;
```

```
Or,

x = { arg freq = 440; SinOsc.ar(freq, 0, 0
this returns a Synth object;
x.set(\freq, 880); // note you can set the
```

Many of the examples in SC3 make use of the Function

Function.play is often more convienent than SynthDef.p
short examples and quick testing. The latter does have
options, such as lagtimes for controls, etc. Where reuse
flexibility are of greater importance, SynthDef and its va
usually the better choice.

scope(numChannels, outbus, fadeTime, bufsize, zoom)

As **play** above, but plays it on the internal **Server**, and c
open a scope window in which to view the output. Curre
OSX.

numChannels - The number of channels to display in th
starting from **outbus**. The default is 2.
outbus - The output bus to play the audio out on. This is
Out.ar(outbus, theoutput). The default is 0.
fadeTime - A fadein time. The default is 0.02 seconds, v
to avoid a click.
bufsize - The size of the buffer for the ScopeView. The
zoom - A zoom value for the scope's X axis. Larger valu
default is 1.

```
{ FSinOsc.ar(440, 0, 0.3) }.scope(1)
```

plot(duration, server, bounds, minval, maxval, parent)

Calculates **duration** in seconds worth of the output of th
plots it in a GUI window. Unlike **play** and **scope** it will n
Out Ugens, so your function should return a UGen or ar
plot will be calculated in realtime.

duration - The duration of the function to plot in second
0.01.
server - The **Server** on which to calculate the plot. This
your local machine, but does not need to be the inte
default server will be used.
bounds - An instance of **Rect** or **Point** indicating the bo
window.
minval - the minimum value in the plot. Defaults to -1.0.
maxval - the maximum value in the plot. Defaults to 1.0
parent - a window to place the plot in. If nil, one will be

```
{ SinOsc.ar(440) }.plot(0.01);
```