SynthDefs

Basic concept: Instances of an instrument

- One synthesizer per note
- Only exists in that lifetime
- Efficiency

See instantiation in action

You can follow the spawning of synths by running `s.plotTree`

```
(
// Play random pattern
Pbind(\degree, Pwhite(0,10)).play;
// Look at the node tree to see the synths spawned
s.plotTree;
)
```

SynthDef vs. Synth

Imagine a cookie.

A SynthDef is the recipe for the cookie.

A Synth is one of the actual cookies.

What's in a SynthDef?

Unit generators aka.a UGens.

Building blocks that generate or modify signals of all kinds.

SuperCollider comes with over 1000 over them if you have sc3-plugins installed.

Skeleton of a SynthDef

A SynthDef is declared by adding a name (`\name` below) and a function (`{}` braces) containing a synthesizer patch.

```
SynthDef(\name, {}).add
```

Playing a synthdef

It can be useful while testing to be able to play the SynthDef.

Use the `.play` message to do this.

```
SynthDef(\name, {}).play
```

Adding a synthdef to the server

Use the `.add` message to add a synthdef to the server and make it available for the sound engine.

```
SynthDef(\name, { arg freq;
        // ...
})
```

UGen rates

- `.ar` – Audio rate
- `.kr` - Control rate
  - Lower "resolution"
  - more effective
  - Used for LFOs, modulators etc.
- `.ir` - Initial rate (aka. Scalar)
  - Set once on initialization and can't be changed
  - Used for SampleRate etc.

Variables

Variables are defined using the `var` keyword.

They can contain data and/or UGens:

```
// A variable containing our frequency
var freq = 100;

// A variable containing our signal
var sig = SinOsc.ar(freq);

// Overwrite the signal with a version that has half the amplitude
sig = sig * 0.5;
```

Arguments

The interface of your Synth.

Each argument you add will be controllable from a pattern.

Argument defaults

Sometimes it's sensible to have a default value

When to use arguments and when to use variables?

Outputting sound

Use Out.ar to output sound from a synth

```
var sig = WhiteNoise.ar();
Out.ar(0, sig);
```

Got problems? Poll them!

The most important tool for debugging synths is poll. Use it by attaching `.poll` to the part of your synth you want to debug.

```
(
SynthDef(\problematic, {
    WhiteNoise.kr().poll
}).play
)
```

Let's make a SynthDef!

```
SynthDef(\sawtooth, {
        // Arguments
        arg freq=442, cutoff=500, amp=0.5, out=0;
        // The sound generator
        var sig = Saw.ar(freq);

        // Scale amplitude
        sig = sig * amp;

        // Filter high frequencies
        sig = LPF.ar(sig, cutoff);

        Out.ar(out, sig)

}).play;
```

Adding an envelope and duration

```
// Kill synth when done
var done = 2;
// Percussive envelope
var env = Env.perc(attack, decay);
env = env.kr(done, gate, dur);

...

sig = sig * env;
Out.ar(out, sig )
```

Hint: Don't forget to add `attack`, `decay`, `gate` and `dur` to list of arguments.

```
SynthDef(\sawtooth, {
        // Arguments
        arg freq=442, cutoff=500, amp=0.5, out=0, attack=0.001, decay=0.99, dur=1, gate=1;

        // Kill synth when done
        var done = 2;
        // Percussive envelope
        var env = Env.perc(attack, decay).kr(done, gate, dur);

        // The sound generator
        var sig = Saw.ar(freq);

        // Scale amplitude and apply envelope
        sig = sig * amp * env;

        // Filter high frequencies
        sig = LPF.ar(sig, cutoff);

        Out.ar(out, sig)

}).play;
```

Exercise: Modify your Synth!

Try changing your SynthDef by exchanging the `Saw` to another type of oscillator.

Browse the help system to find a replacement.

Hint: Pulse, DPW3Tri and SinOscFB are fun alternatives.

Make sure to read the help file.