Functional Algorithmic Music with Euterpea

DONYA QUICK

dquick@stevens.edu | donyaquick@gmail.com

RESEARCH ASSISTANT PROFESSOR OF MUSIC & COMPUTATION
STEVENS INSTITUTE OF TECHNOLOGY

Composing with Euterpea

Virtual instrument design with Euterpea

- Fantasy for Bottles (hand-made score)
- Felis (100% coded from score to sound)

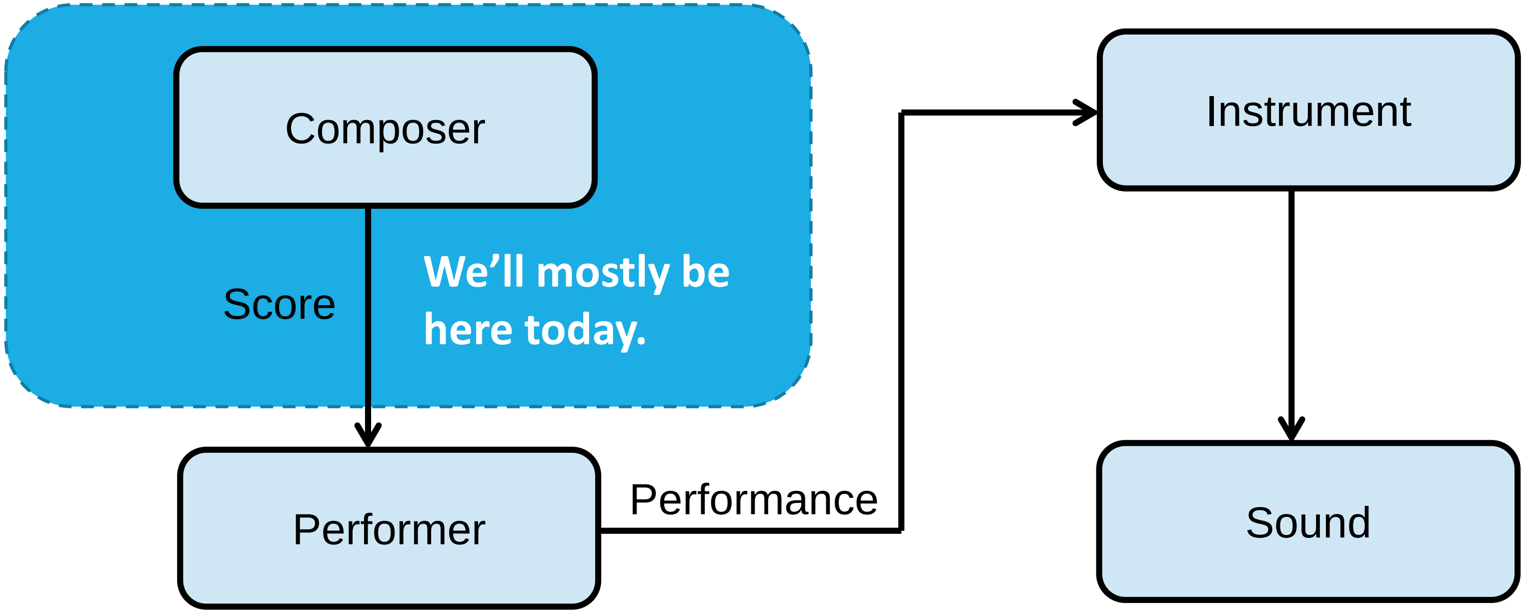
Score-level work with Euterpea

(all virtual instruments rendered in a DAW)

- Zero order
  - arranged output from real-time interactive program
- 33rd and 5th
  - algorithmically generated score
- Dot Matrix
  - algorithmically generated score

There are no human-made notes in the scores for these three examples. It’s all algorithmically generated.

Composition vs. Performance



What is Euterpea?

- ❑ Eutpera is a music library for the **Haskell programming language**.
- ❑ Haskell is a **pure functional** language. (More on this shortly)
- ❑ Export formats Euterpea supports:
  - MIDI - we'll just use this one today
  - WAV using your own virtual instrument definitions
- ❑ Want to try Euterpea? Instructions are here: `http://www.euterpea.com` (along with tutorials & examples)

Functional Programming

- A programming language is ***functional*** if you can pass functions like variables. For example:
  f(x) = x+1
  y = g(f,3)
- Many languages are partially functional (like Python).
- ***Pure functional*** languages, however…are a bit strange.
  - Everything is either a **value** or a **function**.
  - *No* mutable variables (mutable = values change in memory over time)
  - *No* for/while loops
  - Iterative computation happens only through **recursion**
    - Recursion = a function's definition calls itself at some point
- Pure functional coding is a lot like writing blackboard math.

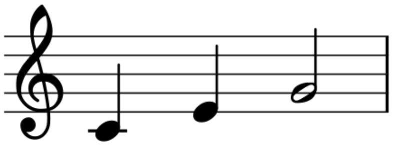
Why Be Purely Functional?

- No mutable variables/loops = fewer bugs
  - Many common kinds of bugs become impossible in compiled code.
  - Because of this, functional programming is increasingly common in rapid prototyping tasks in the corporate world.
- For example: there is no way to accidentally write over a memory location in a language like Haskell*.
- There are some other perks too with *lazy* functional languages (like Haskell)

* Two exceptions: compiler-level bugs (extremely rare) or compiling for the wrong architecture.

Let’s Make Some Notes!

```
m1 = c 4 qn :+: e 4 qn :+: g 4 hn
```



Pitch class

Octave

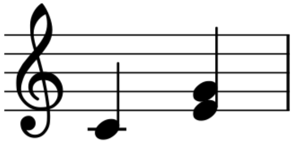
Duration
(qn = “quarter note”)

“followed by…”

“half note”

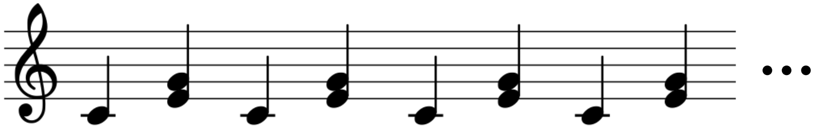
“at the same time as…”

```
m2 = c 4 qn :+: (e 4 qn :=: g 4 qn)
```



And what about this…?

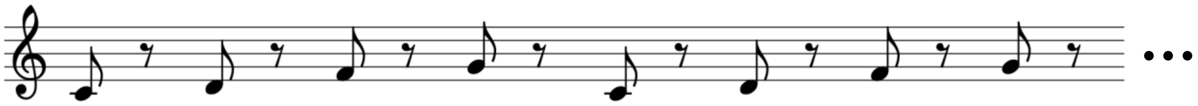
```
m3 = c 4 qn :+: (e 4 qn :=: g 4 qn) :+: m3
```



Recursion: a definition that refers to itself!

A Simple Phase Composition

```
p1 = c 4 en :+: rest en :+: d 4 en :+: rest en :+:
     f 4 en :+: rest en :+: g 4 en :+: rest en :+: p1
```



Recursive definition for infinite music

Move up/down by semitones

Scale tempo by a factor

```
p2 = transpose 3 (tempo 1.01 p1)
p3 = transpose 5 (tempo 1.02 p1)
phase = instrument Marimba (p1 :=: p2 :=: p3)
```

Set a general MIDI instrument

Playing all three parts at the same time (in parallel)

In Case You Were Wondering...

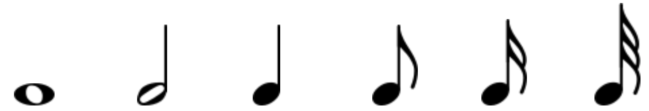
- Making sharps & flats: add `s` for sharp and `f` for flat

  `cs` = C#

  `ef` = E♭

  etc.

- Some other durations:



wn	hn	qn	en	sn	tn
$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$

add `d` for “dotted”

`dqn` = “dotted quarter note” (1.5 times as long as a `qn`)

You can also just write numbers for durations. For example:

```
c 4 (1/6)
f 4 0.456
```

Composing with Euterpea

Let’s Make Some More Music

1

Create a motif

- Just a few notes

2

Transform using

- Transposition
- Retrograde
- Inversion

3

Layer it

- Play another copy at 2/3rds the tempo (different effect to phase composition)

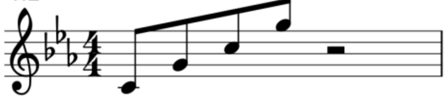
Composing with Euterpea

```
x1 = c 4 en :+: g 4 en :+: c 5 en :+: g 5 en
x2 = x1 :+: transpose 3 x1
x3 = x2 :+: x2 :+: invert x2 :+: retro x2
x4 = forever x3 :=: forever (tempo (2/3) x3)
```

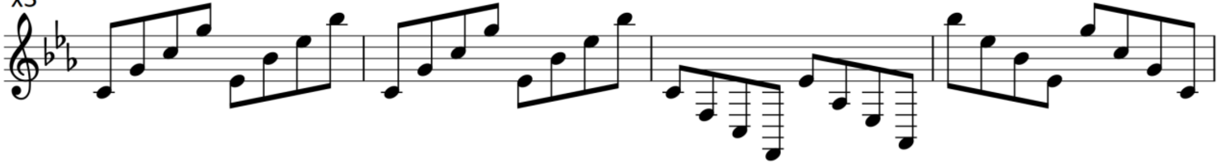
And with some more work…

Custom performance algorithm → Adjust tempo & use a good synth → Jam on it ("Blue Lambda")

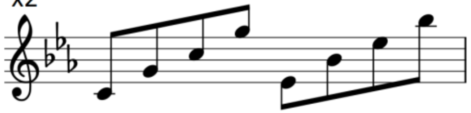
x1



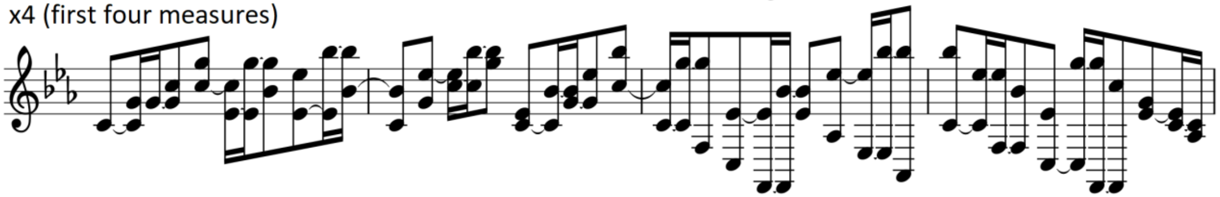
x3



x2



x4 (first four measures)



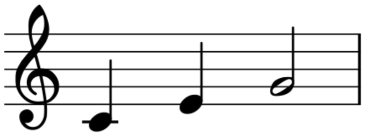
Making Music with Numbers

You can also create notes using **pitch numbers**, which are integers (`Int`) in Euterpea.

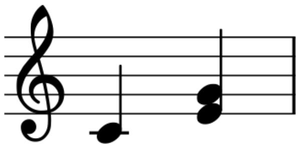
Caution #1: you *can't* mix this approach with the `c 4 qn` method of making notes. You should choose one or the other style when coding your music.

Caution #2: you have to supply some **type information** when using pitch numbers.

```
m1’ :: Music Int
m1’ = note qn 60 :+: note qn 64 :+: note hn 67
```



```
m2’ :: Music Int
m2’ = note qn 60 :+: (note qn 64 :=: note qn 67)
```



This stuff is type information. It lets the computer know about the specific number representation we want in the definitions of `m1’` and `m2’`.

Why Musical Numbers are Handy

- It’s easy to turn a **list** of numbers into notes.
- It’s easy to make lists with lots of numbers.
- Lots of numbers = lots of notes with little code.
- But…let’s start with just a few numbers first.

A list of four numbers.

```
nums1 = [60,64,67,72]
numMusic1 :: Music Int
numMusic1 = line (map (note qn) nums1)
```

Some type information as before

“Put together sequentially…”
(in other words, with :+:)

“make it into a quarter note…”

“for every value in nums1….”

Music from Random Numbers

```
import System.Random
```

Import a library of random number functions

```
rGen = mkStdGen 5
```

Random seed (computers are only pseudo-random)

```
rNums = randomRs (50, 85) rGen
```

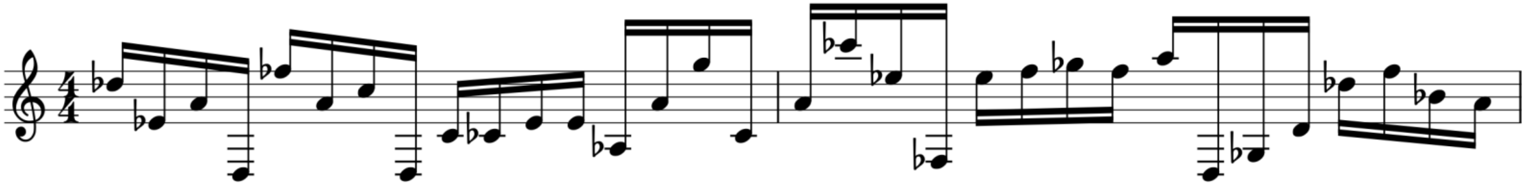
“Generate an infinite list of random numbers between 50 and 85.”

```
randMusic :: Music Int
randMusic = line (map (note sn) rNums)
```

“Put together sequentially…” (in other words, with :+:)

“for every value in nums1….”

“make it into a quarter note…”



The “Thinking Computer” Sound

- Take the code we just did…

```
import System.Random
rGen = mkStdGen 5
rNums = randomRs (50, 85) rGen

randMusic :: Music Int
randMusic = line (map (note sn) rNums)
```

- Speed it up, and make it use a square wave instrument.

```
thinkingComputer =
  tempo 2 (instrument Lead1Square randMusic)
```

With a plain synth: 🔊 And a better one… 🔊

Randomizing Other Features

```
durs1 = choices [qn, en, en] (mkStdGen 30)
pitches1 = choices [60,62,64,67,69] (mkStdGen 31)
vols1 = randomRs (40,120) (mkStdGen 32)
pvPairs1 = zip pitches1 vols1

myMusic1 :: Music (Int, Int)
myMusic1 = line (zipWith (\x y -> note x y) durs1 pvPairs1)
```

…

`myMusic2 =` (basically the same thing with different numbers used)

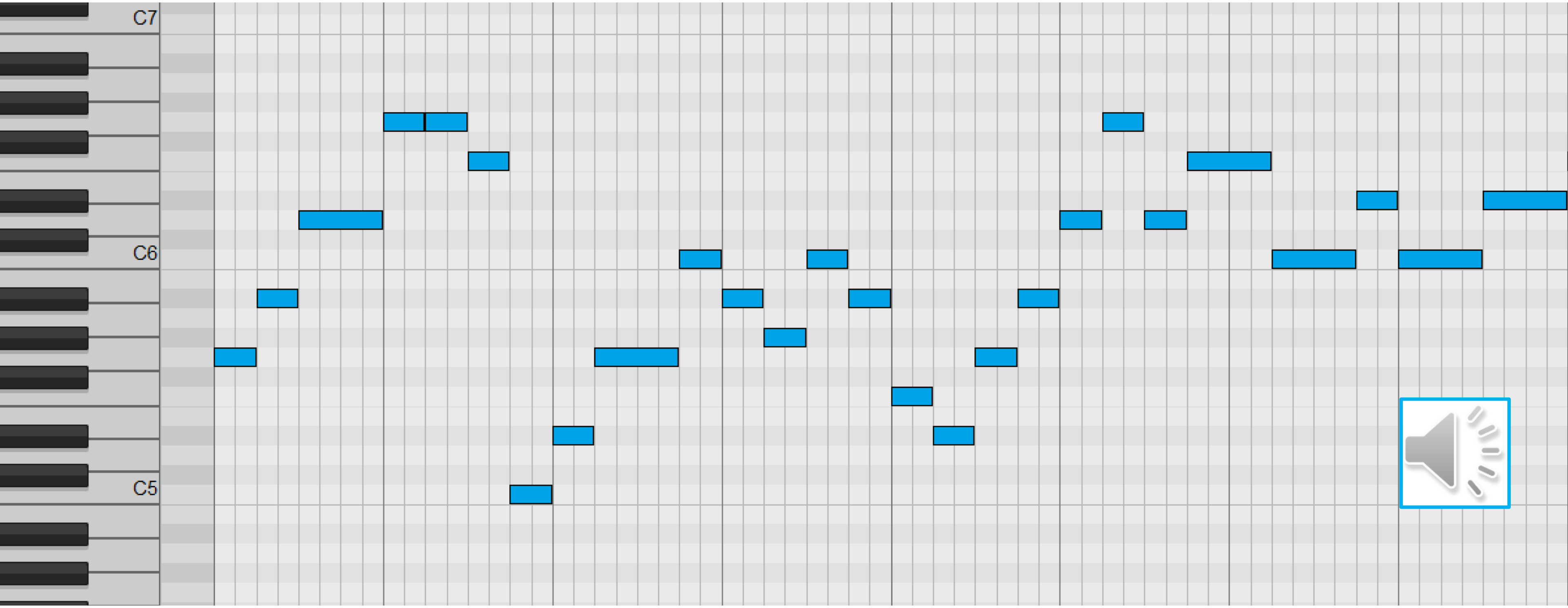
```
duet =
  tempo 1.5 (instrument Vibraphone (myMusic1 :=: myMusic2))
```

…

`choices items g0 =` (infinitely choose values from the `items` list)

Randomness with Structure

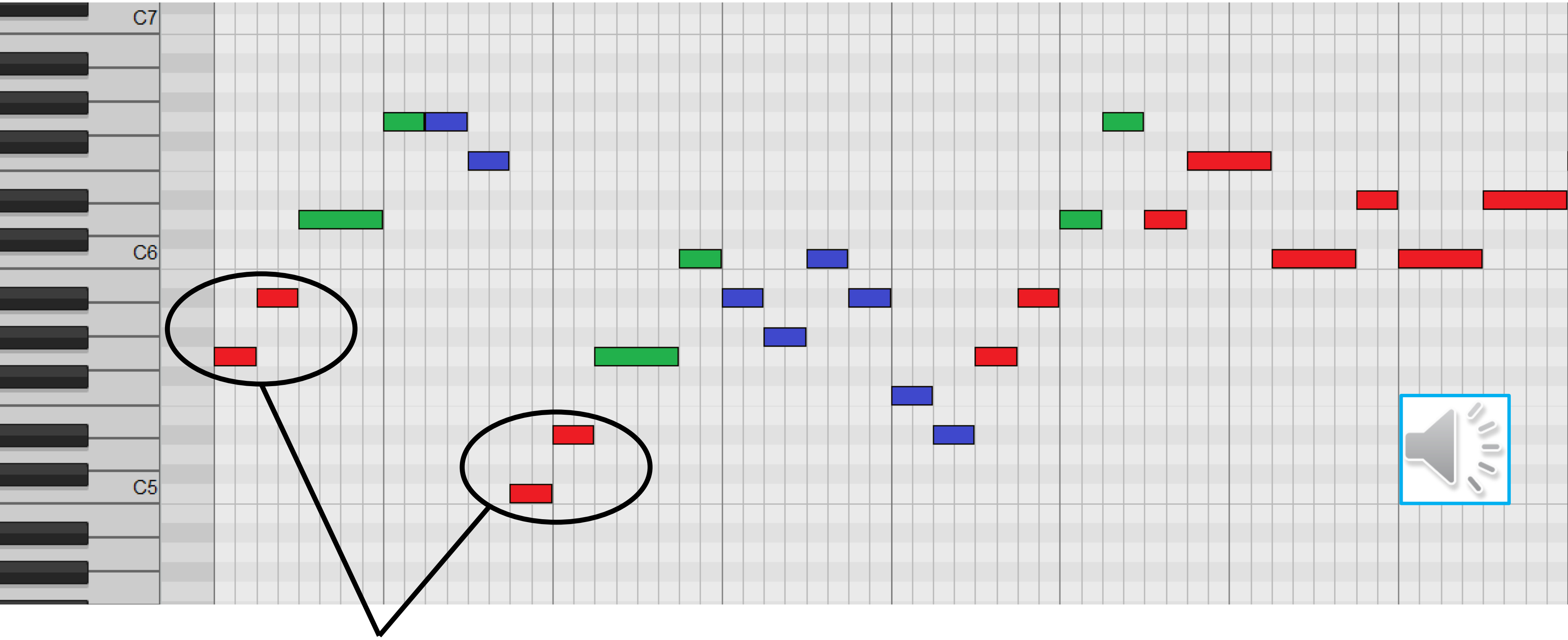
Consider the pitch structure here. Why does it sound less random?
(Durations & volumes are still randomly assigned)



Random C-major as a point of comparison:

Patterns in a Melody

It was stochastically built from 3 intervals! This creates a sort of micro-structure.
General strategy: use randomness to control more complex decision-making.



These are both instances of the same pitch interval pattern.

Composing with Patterns

blueText = name of user-defined function

A marimba part (treble):

```
pats2 = [[0,3], [0,5], [0,-2]]
s2 = scaleToPSpace (50,80) [0,2,4,5,7,9,11]
d2 = 7  -- how far away the next pattern can be
x2 = 60  -- rough starting point
nums2 = pGen2 s2 pats2 x2 d2 (mkStdGen 7)  -- do generation
m2 = line $ map (note sn) nums2
```

C-major scale

A plucked string part (bass):

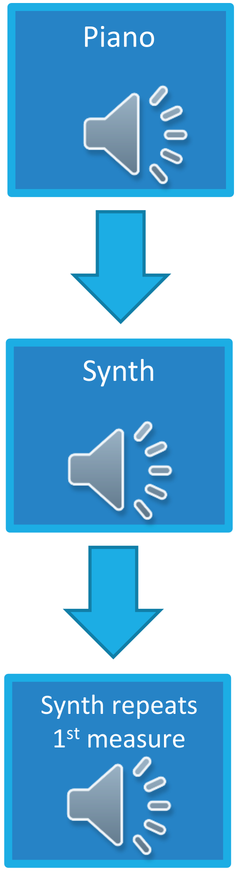
```
pats3 = [[0,7], [0,4], [0,12]]
s3 = scaleToPSpace (30,50) [0,4,7]
d3 = 12  -- patterns can jump around more this time
x3 = 40  -- rough starting point (near the bottom)
nums3 = pGen s3 k3 x3 d3 (mkStdGen 10) -- do generation
m3 = line $ map (\p -> note en p :+: rest en) nums3
m3’ = m3 :=: transpose 12 (rest en :+: m3)
```

A partial scale. Just the root, third, & fifth of C-major

All together:

```
m4 = instrument PizzicatoStrings m3o :=:
     instrument Marimba m2
```

Composing with Patterns



Piano

Synth

Synth repeats 1st measure

A pattern-based melody
(infinite & non-repeating)

Making the instrument more interesting.

Adding larger-scale repetition in addition to the smaller structure.

There's more on this topic here: `euterpea.com/tutorials`
(look for "Pattern-Based Algorithmic Music with Euterpea")

Thank You!

- Euterpea website: www.euterpea.com
  - Setup info, examples, and tutorials
- My website: www.donyaquick.com
  - E-Mail: dquick@stevens.edu | donyaquick@gmail.com
- Compositions using Euterpea: soundcloud.com/donyaquick/
  - Playlists > Made with Euterpea