MUSIC MIXING WITH EARSKEETCH AND PYTHON

Georgia Institute of Technology
ICE Programs
May 27th, 2014
Mr. Michaud
Workshop Agenda

- Mix music with EarSketch/Python API
- Sequence of Concepts for Summer Camp
  - EarSketch API
  - Python Language / Computer Science
- Activities and Schedule for EarSketch Camp
- Materials
  - PowerPoints
  - Assignments/Projects
  - Pre and Post Test
  - EarSketch website
Description of EarSketch

- Programming Environment
- Python Based
- Web Based App or Installed System of Software
- API built in Python for Music Mixing
- NSF Funded project to encourage computational interest through the mixing and sharing of music.
- Curriculum and Social Media Site
- earsketch.gatech.edu
Online IDE: Code Editor

```python
from earsketch import *
init()
setTempo(120)
finish()
```
Online IDE: Workstation View
What we will learn:

**Music Remixing**
- Digital Audio Workstation
- Tracks (Layers)
- Measures
- “Slices” of Music
- Rhythms
- Instrumentation
- Effects

**Computer Science**
- Python Programming
- Sequential Steps
- Functions
- Arguments
- Variables
- Integers
- Strings
- For Loops
What we will create:

- An EarSketch Python program that generates a musical composition or remix.
- The resulting remix should be at least 32 measures long and have 3 Tracks
- Demonstrate use of variables, EarSketch functions, “For Loops”, and creating your own functions.
Reaper

Tracks: Hold Sounds

Measures: Timeline

Playback Controls:
Rewind, Stop, Play, Pause
Human Programming:

doJumpingJacks(5)

singSong("Happy Birthday", Mr. Michaud)
Programming with Python

- Python – text based programming language
- Sequential List of Instructions for Computer to Follow
- Used by Google, Nasa, Robotics . . .
- Example . . . Class Demonstration
Essential Elements we will use in Python:

- **Comments**
  
  # This is a comment – meant for Humans

- **Includes** – loading preset methods or data
  
  from earsketch import *

- **Methods** – telling the computer “what to do”
  
  fitMedia(drums, 1, 1, 5)

- **Variables** – Names for information stored by program
  
  Beat1 = “0+++0+++0+0+0+++”
Include Statements

- Must put these two lines towards the beginning of all your EarSketch remix programs:

  ```python
  from earsketch import *
  ```

- These put the necessary “preloaded” instructions into your program.
Functions and Arguments

- **Function** – Tell the Computer what to do
- **Arguments** – details for the Method
- Example:

  \[ \text{doExercise(JumpingJacks, 5)} \]
EarSketch Python Functions

- **init()**
  Start New Reaper File

- **setTempo(120)**
  Beats per minute in remix

- **println(“Hello”)** -
  Prints message in console

- Example:
  println(“This is my music”)

- Example2:
  println(str(5+5))  # For printing numbers
  # Convert to string using str()
EarSketch Python Functions

- `insertMedia(file, track, measure, scale)`
- `insertMediaSection(file, track, location, start, end, scale)`
- `fitMedia(file, track, start, end)`
- `makeBeat(file, track, measure, beatString)`
- `randint(0,3) # returns a random integer`
"fitMedia" Function

fitMedia(file, track, start, end)

- Location of Media Sound
- Which Track in Reaper
- Start measure.
- End Measure

Example:

```python
fitMedia(HIP_HOP_DRUMS1_2M, 1, 1, 9)
```
“makeBeat” Method

makeBeat(file, track, measure, BeatString)

Location of Media Sound

Which Track in Reaper

What measure.

Example: “0+++0+++0+0+0+++”

Example:

makeBeat(drums, 1, 1, “0+0+0+++00-00+++”)
makeBeat(ELEKTRO_HOUSE_DRUMS3_2M, 1, 1, "0+++0+++0+0+0+++")
"InsertMedia" Method

`insertMedia(file, track, measure, Scale)`

- **File Location of Media Sound**
- **Which Track in Reaper**
- **What measure.**

Scale: True → Matches Tempo
False → Do Not Match Tempo
Usually not used

Example:

```
insertMedia(HIP_HOP_DRUMS1_2M, 1, 1)
```
Setting Volume Effects

- `setEffect(track, VOLUME, GAIN, level, start, level2, end)`

- **Example**

  ```
  setEffect(1, VOL_PAN, VOL_PAN_VOL, -40, 1, 10, 5)
  ```
Other Effects: Delay

DELAY creates a repeated echo-like delay of the original sound. A delay effect plays back the original audio as well as a delayed, quieter version of the original that sounds like an echo. After the first echo it plays an echo of the echo (even quieter), then an echo of the echo of the echo (still quieter), and so on until the echo dies out to nothing. With the delay effect, we can control how much time passes between each echo (delay time). If we set the delay time to match the length of a beat, we can create rhythmic effects with delay.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>minValue</th>
<th>maxValue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY_TIME</td>
<td>The time amount in milliseconds (ms) that the original track is delayed, and the time between successive repeats of the delay.</td>
<td>300.0</td>
<td>0.0</td>
<td>4000.0</td>
<td>1 setEffect(1, DELAY, DELAY_TIME, 1000.0)</td>
</tr>
<tr>
<td>DELAY_FEEDBACK</td>
<td>The relative amount of repeats that the delay generates. Higher values create more &quot;echoes&quot;. Be careful of applying &quot;too much&quot; feedback!</td>
<td>-3.0</td>
<td>-120.0</td>
<td>-1.0</td>
<td>1 setEffect(1, DELAY, DELAY_FEEDBACK, -20.0)</td>
</tr>
<tr>
<td>MIX</td>
<td>The percentage of the affected sound (wet) that is mixed with the original sound (dry). At its minimum value (0.0), no effect can be heard. At its maximum value (1.0), none of the original sound is heard — it is all effect.</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1 setEffect(1, DELAY, MIX, 0.4)</td>
</tr>
<tr>
<td>BYPASS</td>
<td>Whether the effect is &quot;on&quot; (1.0) or &quot;off&quot; (0.0). If the bypass of an effect is &quot;on&quot; (1.0), that means the audio going into the effect passes through, and comes out unaffected. Note that unlike other effect name/parameter pairs, the only valid values for BYPASS are 0.0 and 1.0.</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1 setEffect(1, DELAY, BYPASS, 1.0)</td>
</tr>
</tbody>
</table>

setEffect(1, DELAY, DELAY_TIME, 500, 1)

From [http://earsketch.gatech.edu/category/learning/reference/every-effect-explained](http://earsketch.gatech.edu/category/learning/reference/every-effect-explained)
Other Effects: Distortion

**DISTORTION**

DISTORTION creates a "dirty" or "fuzzy" sound by overdriving the original sound. This compresses or clips the sound wave, adding overtones (higher frequencies related to the original sound). It is common to distort an electric guitar sound by "overdriving" the guitar amplifier. Modern music sometimes uses distortion to add a grungy or gritty effect or feel to the composition.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>minValue</th>
<th>maxValue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTO_GAIN</td>
<td>The amount of overdrive of the original sound.</td>
<td>20.0</td>
<td>0.0</td>
<td>50.0</td>
<td><code>setEffect(1, DISTORTION, DISTO_GAIN, 30, 1)</code></td>
</tr>
<tr>
<td>MIX</td>
<td>The percentage of the effected sound (wet) that is mixed with the original sound (dry). At its minimum value (0.0), no effect can be heard. At its maximum value (1.0), none of the original sound is heard – it is all effect.</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td><code>setEffect(1, DISTORTION, MIX, 0.4)</code></td>
</tr>
<tr>
<td>BYPASS</td>
<td>Whether the effect is &quot;on&quot; (1.0) or &quot;off&quot; (0.0). If the bypass of an effect is &quot;on&quot; (1.0), that means the audio going into the effect passes through, and comes out unaffected. Note that unlike other effect name/parameter pairs, the only valid values for BYPASS are 0.0 and 1.0.</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td><code>setEffect(1, DISTORTION, BYPASS, 1.0)</code></td>
</tr>
</tbody>
</table>

`setEffect(1, DISTORTION, DISTO_GAIN, 30, 1)`

From [http://earsketch.gatech.edu/category/learning/reference/every-effect-explained](http://earsketch.gatech.edu/category/learning/reference/every-effect-explained)
Other Effects: PAN

**PAN**

PAN affects the audio mix between the left and right channels. For example, if you were wearing headphones, changing the panning would affect whether you heard something from the left earcup or the right.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>minValue</th>
<th>maxValue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT_RIGHT</td>
<td>Specifies the left/right location of the original sound within the stereo field (0.0 is center, -100.0 is fully left, 100.0 is fully right).</td>
<td>0.0</td>
<td>-100.0</td>
<td>100.0</td>
<td><code>setEffect(1, PAN, LEFT_RIGHT, -50.0)</code></td>
</tr>
<tr>
<td>BYPASS</td>
<td>Whether the effect is &quot;on&quot; (1.0) or &quot;off&quot; (0.0). If the bypass of an effect is &quot;on&quot; (1.0), that means the audio going into the effect passes through, and comes out unaffected. Note that unlike other effect name/parameter pairs, the only valid values for BYPASS are 0.0 and 1.0.</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td><code>setEffect(1, PAN, BYPASS, 1.0)</code></td>
</tr>
</tbody>
</table>

*setEffect(1, PAN, LEFT_RIGHT, 100, 1)*

From [http://earsketch.gatech.edu/category/learning/reference/every-effect-explained](http://earsketch.gatech.edu/category/learning/reference/every-effect-explained)
Other Effects: PITCHSHIFT

PITCHSHIFT simply lowers or raises the sound by a specific pitch interval (PITCHSHIFT_SHIFT). It can be useful in helping multiple sound files sound better together or, contrastingly, to add a little bit of dissonance, if desired. It can also be used to create a harmony part for the same track by keeping the PITCHSHIFT_MIX setting somewhere towards its central position (0.5), to balance the volume between the effect and the original sound.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>minValue</th>
<th>maxValue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PITCHSHIFT_SHIFT</td>
<td>Specifies the amount to adjust the pitch of the original sound in semitones (and fractions of a semitone, given by values after the decimal point). 12 semitones equal 1 octave.</td>
<td>0.0</td>
<td>-12.0</td>
<td>12.0</td>
<td>`1</td>
</tr>
<tr>
<td>MIX</td>
<td>The percentage of the effected sound (wet) that is mixed with the original sound (dry). At its minimum value (0.0), no effect can be heard. At its maximum value (1.0), none of the original sound is heard – it is all effect.</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>`1</td>
</tr>
<tr>
<td>BYPASS</td>
<td>Whether the effect is “on” (1.0) or “off” (0.0). If the bypass of an effect is “on” (1.0), that means the audio going into the effect passes through, and comes out unaffected. Note that unlike other effect name/parameter pairs, the only valid values for BYPASS are 0.0 and 1.0.</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>`1</td>
</tr>
</tbody>
</table>

setEffect(1, PITCHSHIFT, PITCHSHIFT_SHIFT, 06, 1)

From [http://earsketche.gatech.edu/category/learning/reference/every-effect-explained](http://earsketche.gatech.edu/category/learning/reference/every-effect-explained)
Other Effects: RINGMOD

RINGMOD multiplies the signals from two sounds together: your original sound and a pure sine wave (that sounds like a tuning fork). The effect of this multiplication sounds different at every frequency of the original sound, which creates a completely artificial-sounding result, as this type of sound could never occur naturally. Some parameter settings for this effect will likely produce recognizable-sounding effects similar to ones used in old science-fiction movies. It is useful experimenting with since there are a wide range of sounds that can be generated from your original sound.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>minValue</th>
<th>maxValue</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RINGMOD_MODFREQ</td>
<td>The frequency (in Hz) of the sine wave oscillator that is being multiplied into your original sound.</td>
<td>40.0</td>
<td>0.0</td>
<td>100.0</td>
<td>1</td>
</tr>
<tr>
<td>RINGMOD_FEEDBACK</td>
<td>The amount of effected sound that is fed-back into the effect. High values create more robotic-type sounds and sonic artifacts.</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>1</td>
</tr>
<tr>
<td>MIX</td>
<td>The percentage of the effected sound (wet) that is mixed with the original sound (dry). At its minimum value (0.0), no effect can be heard. At its maximum value (1.0), none of the original sound is heard – it is all effect.</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>BYPASS</td>
<td>Whether the effect is “on” (1.0) or “off” (0.0). If the bypass of an effect is “on” (1.0), that means the audio going into the effect passes through, and comes out unaffected. Note that unlike other effect name/parameter pairs, the only valid values for BYPASS are 0.0 and 1.0.</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1</td>
</tr>
</tbody>
</table>

From (http://earsketch.gatech.edu/category/learning/reference/every-effect-explained)
## Variables in Python

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Whole number</td>
<td>measure = 4</td>
</tr>
<tr>
<td>Float</td>
<td>Decimal number</td>
<td>sub = 1.5</td>
</tr>
<tr>
<td>String</td>
<td>Characters or words</td>
<td>beat = “0+++0+0+”</td>
</tr>
</tbody>
</table>
# Operators in Python

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
<th>Example</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>5 + 5</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>4 - 2</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>3 * 5</td>
<td>15</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>4 / 2</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>Modulo</td>
<td>5 % 2</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>Assigns Value</td>
<td>measure = 4</td>
<td></td>
</tr>
<tr>
<td>==</td>
<td>Is equal to</td>
<td>4 == 4</td>
<td>true</td>
</tr>
<tr>
<td>++</td>
<td>Increase by 1</td>
<td>a = 5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a++</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>println(str(a))</td>
<td></td>
</tr>
</tbody>
</table>
Data Structures in Python: Lists

- Zero Based
- Mutable
- Can change size during execution

```python
beats = [HIPHOP_MUTED_GUITAR_001,
         HIPHOP_MUTED_GUITAR_002,
         HIPHOP_MUTED_GUITAR_003,
         HIPHOP_MUTED_GUITAR_004]
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HIPHOP_MUTED_GUITAR_001</td>
</tr>
<tr>
<td>1</td>
<td>HIPHOP_MUTED_GUITAR_002</td>
</tr>
<tr>
<td>2</td>
<td>HIPHOP_MUTED_GUITAR_003</td>
</tr>
<tr>
<td>3</td>
<td>HIPHOP_MUTED_GUITAR_004</td>
</tr>
</tbody>
</table>
path = HIP_HOP_DRUM_FOLDER

for count in range(1, 9, 2):
    randomFile = selectRandomFile(path)
    fitMedia(randomFile, 1, count, count + 1)
    println(count)
For Loops: Skip Counting Challenge:

Write For Loops to meet the following conditions:

Track 1: Place a Measure of music every 4 measures:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Track 2: Place a Measure of music every 2 measures

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Track 3: Place a Measure of music on multiples of 4 measures

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Track 4: Your choice to make everything stick together!
For Loops: Skip Counting

```
fillDrum = HIP_HOP_SYNTHDRUMS2_2M
beat = "0+++0+++0-000+00"

for measure in range(1, 9, 2):
    makeBeat(fillDrum, 1, measure, beat)
```

- `measure` is the “index variable” = assigned values from the range()
- `(1, 9, 2)` means start counting at 1, end before 9 [meaning 8] and skip count by 2:
  - `(1, 3, 5, 7)`
Conditional Statements in Python

if (condition is true):
    # Execute Code

if (measure % 4 == 0):  # On measures evenly divided by 4
    makeBeat(TECHNO_LOOP_PART_001, 1, measure, fillbeat)
Music Mixing Techniques

- Instrumentation: Bass, Drum, Melodic, Accompaniment
- Richard Devine Clips designed to work together by sub folder
- Keep clips in related keys (relative Major / minor)
- Tempos: (Rough Guide)
  - 88-92: Hip Hop / Funk
  - 94-110: Pop Music
  - 110 to 120: Marches (British: 110, American: 120)
  - 120 to 144: Techno
  - Faster than 144: Fast Jazz/Swing
- Less is more – vary the texture
- Music is organized in groups of 4 (Beats, measures, form)
- Organize clips into lists in code.
- Use makeBeat() and fills every 4 measures for variety and drive
EarSketch Website: Complete Reference

EarSketch
computational music remixing and sharing as a tool to drive engagement and interest in computing

About

EarSketch teaches computer science through music composition and remixing! No prior knowledge of either computer science or music is needed. See "More About EarSketch" below to learn more about the software.

I'm a student, how do I get started?

Access the latest version of EarSketch [here].

After you download EarSketch, you’re ready to begin the curriculum, which you can access through the table of contents on the left. You will first be introduced to various aspects of the software you will be using and the EarSketch website. From there, you will learn the basics of programming in EarSketch and create simple songs. Later lessons will focus on more advanced computer science concepts that will help you make your songs more interesting. It is recommended that you step through the curriculum in the order it appears in the table of contents. Click [here] to begin!

I'm a teacher who wants to teach EarSketch, how do I get started?

Access the latest version of EarSketch [here].

We recommend that you go through the first few modules of the student curriculum to familiarize yourself with the components of EarSketch. This includes Intro to DAWs, EarSketch Social Media Site, Anatomy of an EarSketch Project, Creating Your First Project, and The Structure of an EarSketch Script.

Next, you should access the teacher curriculum, which is designed to help computer science teachers with little or no music knowledge begin teaching EarSketch in their classrooms. It will teach you music concepts, rhythms, pattern and variety, and effects as they relate to music programming in EarSketch. Click [here] to