Teaching A Computer To Sing

An After-School Program on Computing + Music

Sponsored by the National Science Foundation
Division of Research on Learning, Award No. 1515767

Advanced Informal STEM Learning (AISL) Program
Science, Technology, Engineering and Mathematics

April 7, 2017

The College of Saint Rose, Albany, NY

Workshop Leaders

Jesse Heines, Prof. Emeritus, Dept. of Computer Science, UMass Lowell
Daniel Walzer, Assistant Prof., Dept. of Music, UMass Lowell
Rachel Crawford, Music Teacher, Bartlett Community Partnership School

This project is supported by Award No. 1515767 from the National Science Foundation (NSF) Division of Undergraduate Education (DUE). It falls under the AISL program: Advanced Informal STEM (Science, Technology, Engineering, and Mathematics) Learning. Any opinions, findings, conclusions, or recommendations expressed in our materials are solely those of the authors and do not necessarily reflect the views of the National Science Foundation.

Workbook Contents

Workshop Leaders

Name: Jesse Heines
Institution: University of Massachusetts Lowell, Lowell, MA
Department: Computer Science
Email Address: jesse_heines@uml.edu

Name: Daniel Walzer
Institution: University of Massachusetts Lowell, Lowell, MA
Department: Music
Email Address: daniel_walzer@uml.edu

Name: Rachel Crawford
Institution: Bartlett Community Partnership School, Lowell, MA
Department: Music
Email Address: rcrawford@lowell.k12.ma.us

Workshop Description

ABSTRACT

This workshop will give attendees direct experience in working with the techniques and tools we use to teach computing in the context of music and music in the context of computing. It will describe and demonstrate the work we have done over the last two years in a middle school, after-school program that combines singing with programming. It will also provide a forum for discussion of how this work can be adapted for participants' schools.

ACTIVITIES

The workshop will begin with a short presentation on what we are trying to accomplish (turning kids on to computing, just as many other programs) and a brief overview of our approach (how we use the 2½ hours that we're with the kids each Tuesday and Thursday after school). We'll entertain some initial questions from attendees, but we'll limit those so that we can get to hands-on work as quickly as possible.

Attendees will first work with EasyABC, a stand-alone program that supports virtually all of the ABC notation standard and translates that notation into a standard music score. This program is freely downloadable for PCs and Macs, but if attendees are unable to install it we will have them work with WebMusicScore or ABCTool, web-based programs that render ABC notation but are considerably less user-friendly than EasyABC.

Attendees will then move their creations to PencilCode, a Scratch-like drag-and-drop programming environment that partially supports ABC notation. They will, for example, translate repeats (|: ... :|) into for loops and 1st and 2nd endings into if statements, work with string concatenation and some simple arithmetic algorithms, and begin to understand the issues involved in generating music from a web page.

Along the way attendees will be exposed to music fundamentals such as note pitch and duration, rests, key signatures, accidentals (sharps and flats), and chords. The session will end with a discussion of how our work can be adapted for use in other schools and perhaps we'll even get people to sing with us!

EXPECTED ATTENDEE BACKGROUND

To get the most out of this workshop, attendees should have a basic familiarity with music notation and programming fundamentals. Attendees who can read music and/or who are familiar with programming environments such as Scratch and PencilCode are especially encouraged to attend. Attendees should bring their own computers so that they can download software and access the relevant program environments on the web.

EXPECTED LEARNING

Attendees will be exposed to samples of the multi-part music scores that our students sing and program, ABC Notation and the EasyABC program designed for transcribing music in that notation, and the PencilCode programming environment and the CoffeeScript language that it uses. They will learn how to use these tools to teach music and computing simultaneously.

HANDOUTS

Attendees will receive samples of all materials discussed, including the "cheat sheets" we have developed to help students learn how to read music and how to code music in ABC Notation. Handouts will also include references to the various websites where additional information can be found.

PRESENTER BACKGROUNDS

Jesse Heines is a Professor Emeritus of Computer Science whose knowledge of music comes from playing trumpet in school bands and orchestras throughout his schooling and singing in barbershop quartets and choruses to this day.

Dan Walzer is an Assistant Professor of Composition for New Media whose knowledge of computing comes from programming and designing interactive media for music performance, gaming, and broadcast.

Rachel Crawford is a music educator who teaches K-8 General Music and serves as the middle school Choir Director. She has worked with students of all ages in studio teaching and after-school programs, but mostly she loves to sing and make up songs with her two young daughters.

Our current work has grown out of earlier work in *Performamatics*, a college-level, interdisciplinary program combining computing and music that was funded by the National Science Foundation from 2007 through 2015.

Acknowledgments

This work is supported by Award No. 1515767 from the National Science Foundation Division of Research on Learning (https://www.nsf.gov/awardsearch/showAward?AWD_ID=1515767). Any opinions, findings, conclusions, or recommendations expressed in this proposal are solely those of the authors and do not necessarily reflect the views of the National Science Foundation.

We also gratefully acknowledge the contributions of many colleagues to this work, particularly Prof. Gena Greher of UMass Lowell and the literally hundreds of students who have taken our courses.

Workshop Schedule and Resource URLs

Introduction

- Examples based on *Shenandoah*
- Sheet Music
- Singing
- ABC Notation
- EasyABC Representation
- Pencil Code Representation

Working with EasyABC

https://sourceforge.net/projects/easyabc/files/EasyABC/1.3.7.6/

- Approaches
- Cheat Sheets
- Documentation: http://abcnotation.com/wiki/abc:standard:v2.1
- Songs

Working with Pencil Code

http://pencilcode.net

- Approaches
- Cheat Sheets
- Documentation: http://book.pencilcode.net, and then click "printable pdf here"
- Songs

Issues and Discussion

Our Custom tacts Library

http://drjay.pencilcode.net/edit/tacts

Sheet Music to Work with in This Workshop

http://tactsworkshop.jesseheines.com

Additional Useful Websites

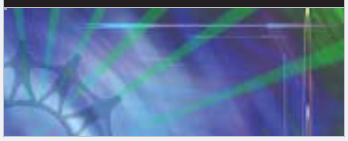
- ABC Notation Home Page
  http://abcnotation.com/
- List of ABC Software Packages
  http://abcnotation.com/software
- Reading Rhythms - Counting Music Interactive Webpage
  http://philtulga.com/counter.html
- MuseScore
  https://musescore.com
- WebMusicScore
  http://www.ronaldhutasuhut.com/wp-content/wms/WebMusicScore.html

National Science Foundation
WHERE DISCOVERIES BEGIN

SEARCH

HOME | RESEARCH AREAS | FUNDING | AWARDS | DOCUMENT LIBRARY | NEWS | ABOUT NSF

Awards



- Search Awards
- Recent Awards
- Presidential and Honorary Awards
- About Awards

How to Manage Your Award

- Grant Policy Manual
- Grant General Conditions
- Cooperative Agreement Conditions
- Special Conditions
- Federal Demonstration Partnership
- Policy Office Website

Award Abstract #1515767

A Middle School After-School Pilot Program Integrating Computer Programming and Music Education

NSF Org:	DRL Division Of Research On Learning
Initial Amendment Date:	July 24, 2015
Latest Amendment Date:	July 24, 2015
Award Number:	1515767
Award Instrument:	Standard Grant
Program Manager:	Alphonse T. DeSena DRL Division Of Research On Learning EHR Direct For Education and Human Resources
Start Date:	August 1, 2015
End Date:	July 31, 2017 (Estimated)
Awarded Amount to Date:	$288,945.00
Investigator(s):	Jesse Heines heines@cs.uml.edu (Principal Investigator) Daniel Walzer (Co-Principal Investigator)
Sponsor:	University of Massachusetts Lowell 600 Suffolk Street Lowell, MA 01854-3643 (978)934-4170
NSF Program(s):	AISL
Program Reference Code(s):	8244
Program Element Code(s):	7259

ABSTRACT

The goal of the project is to research ways in which the teaching of basic computing skills can be integrated into after-school choral programs. The team will study how to adapt the interdisciplinary, computing + music activities developed to date in their NSF-funded Performamatics project with college-aged students to now introduce middle school-aged students to computing in an informal, after-school choral program. They will investigate how to leverage the universal appeal of music to help students who typically shy away from technical studies to gain a foothold in STEM (Science, Technology, Engineering, and Mathematics) by programming choral music. It is funded by the Advancing Informal STEM Learning (AISL) program, which seeks to advance new approaches to, and evidence-based

understanding of, the design and development of STEM learning in informal environments. This includes providing multiple pathways for broadening access to and engagement in STEM learning experiences, advancing innovative research on and assessment of STEM learning in informal environments, and developing understandings of deeper learning by participants.

The team will use a qualitative and quantitative, mixed-methods approach to study four research questions:

- Can middle school-aged children follow the connections from singing to digitized sound to MIDI and back to music and learn to program using the songs they like to sing? To encourage students to become involved with manipulating sounds and programming music on their own computers, the approach will employ Audacity and Scratch, two free music recording, editing, and generation platforms. The team will study how well programming of music helps them acquire STEM skills by assessing the complexity and efficacy of the programs they can learn to code.

- Can programming their individual parts help students learn to sing in three- and four-part harmony? The main focus is on learning of STEM, but research on this question will evaluate whether programming skills can help students learn about music too.

- What resources, models, and tools (RMTs) are necessary to integrate STEM education into a middle school after-school choral program? The team will work with local middle schools to research techniques for integrating computing into after-school choral programs without disrupting their musical focus. They will identify what choral teachers need in order to do this integration, and they will devise and evaluate techniques for adding STEM skills to the students' choral experience.

- Can the involvement of adults who match the students' racial and/or cultural backgrounds have a positive effect on the "people like me don't (or can't) do that?" belief that so often stifles efforts to attract underrepresented groups to STEM? They will actively seek to involve students of underrepresented groups in the program by recruiting adult role models from these groups who are involved with both music and computing. They will use attitudinal surveys to assess whether these adults have any effect on the students' self-efficacy and the "people like me" syndrome that hinders some from engaging in STEM.

Please report errors in award information by writing to: awardsearch@nsf.gov.

Print this page

Top

RESEARCH AREAS FUNDING AWARDS DOCUMENT LIBRARY NEWS ABOUT NSF

Website Policies | Budget and Performance | Inspector General | Privacy | FOIA | No FEAR Act | USA.gov
Accessibility | Plain Language | Contact

National Science Foundation, 4201 Wilson Boulevard, Arlington, Virginia 22230, USA
Tel: (703) 292-5111, FIRS: (800) 877-8339 | TDD: (800) 281-8749

Text Only Version

Computing and Music: What Do They Have in Common?

Computing and music share deep structural similarities. For starters, both rely on notational symbol systems. Programming loops are typically delineated with opening and closing curly brackets { }, parentheses, or levels of indentation. Music loops are delineated with begin and end repeat signs 𝄆 𝄇 or initiated by "D.S." (Italian: *dal segno*), which instructs musicians to "repeat back to the sign," typically designated as 𝄋. As in programming, musical iteration can also make use of loop control variables. For example, Figure 1 shows a loop in which the music changes the second time through.

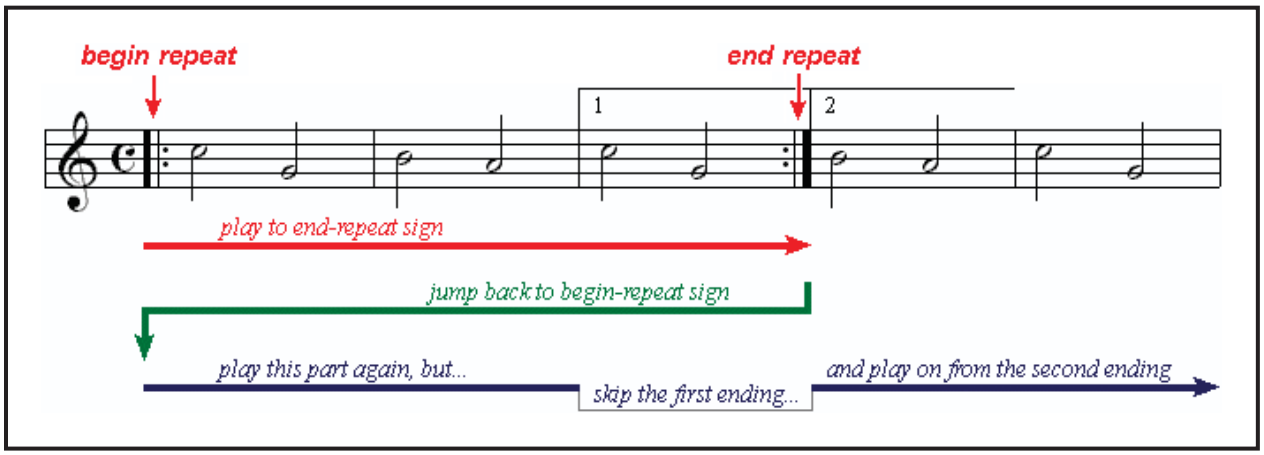


Figure 1. Musical iteration with a loop control variable. [6]

Both computing and music have logic and flow. Figure 2 shows the logic one student saw in The Beatles' All You Need Is Love. If one were to turn this flowchart into a computer program, it would not only contains loops, but if and switch statements as well.

One can also go the other way, converting musical concepts into computer programs. For example, the Scratch [2, 3] program in Figure 3a plays Jimmy Page's famous guitar riff from Led Zeppelin's Kashmir. This code works properly, but consider the many computational thinking (CT) concepts learned by transforming the code in Figure 3a to 3b, which plays exactly the same riff.

Computing and Music (cont'd)

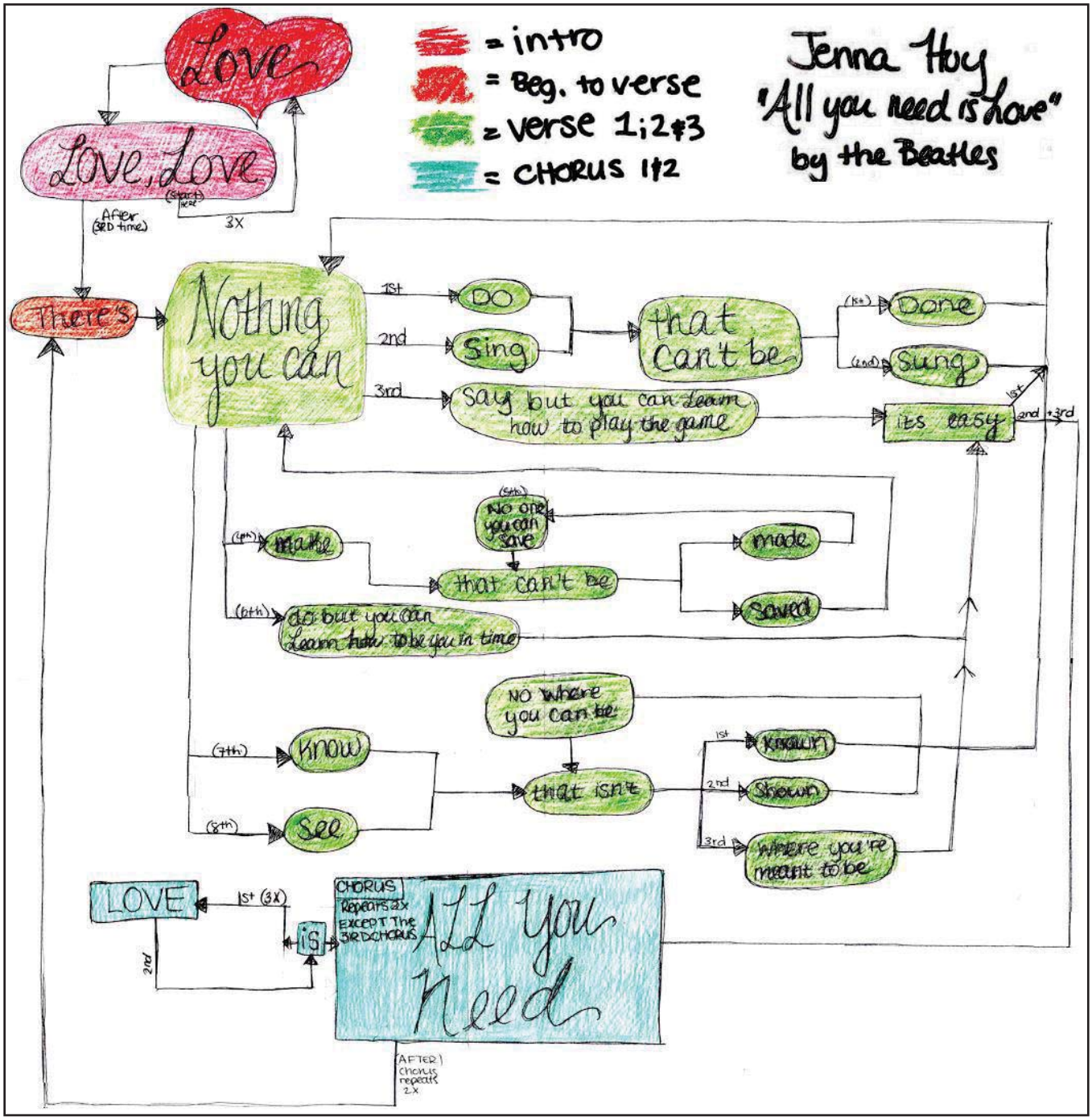
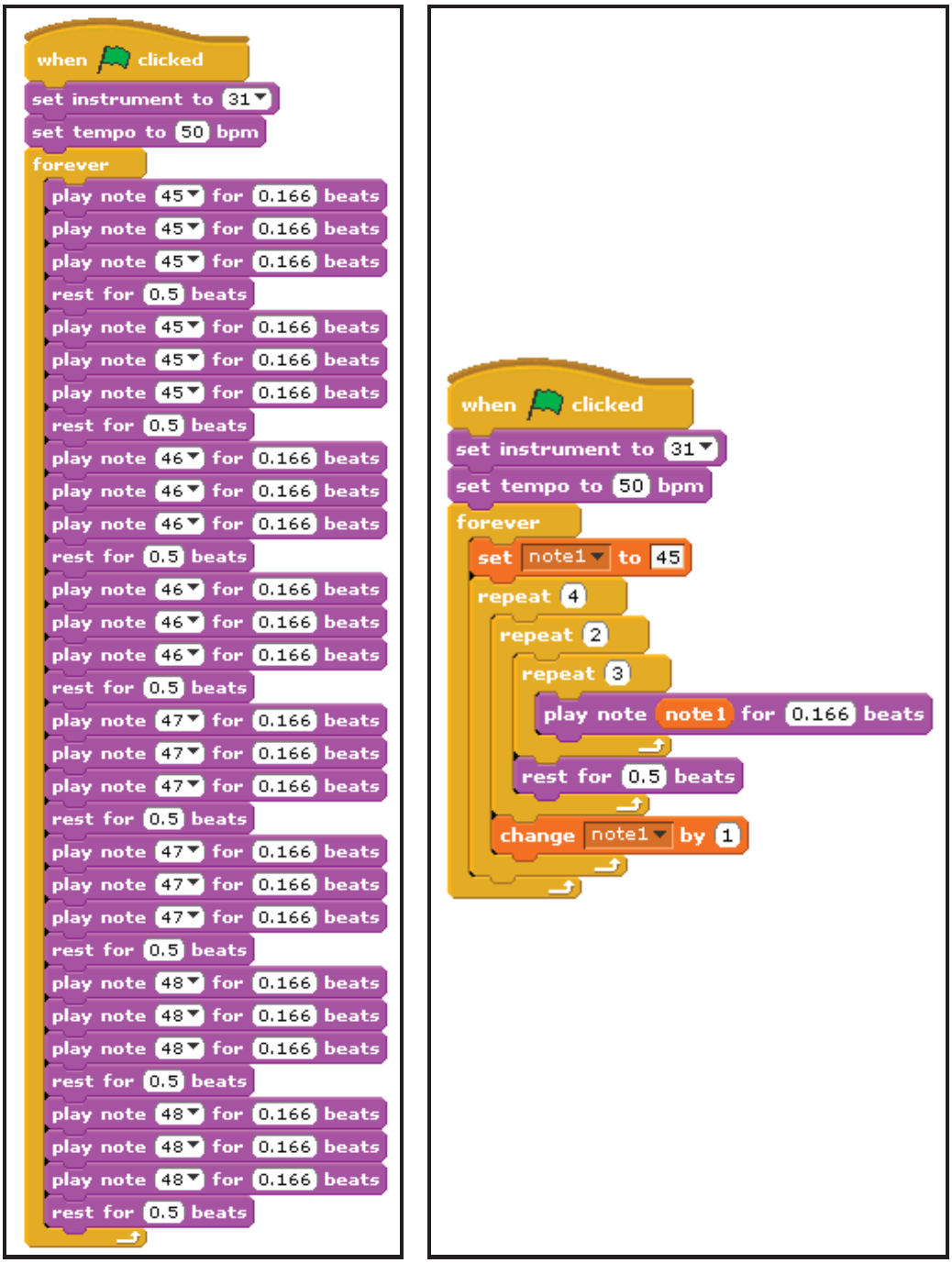


Figure 2. A song flowchart. [1]

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Computing and Music (cont'd)

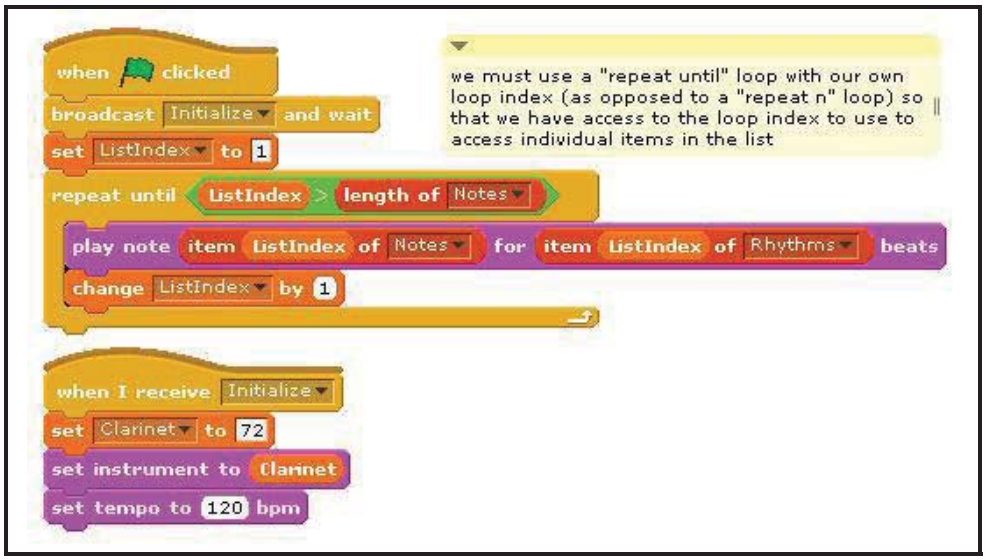


3a 3b

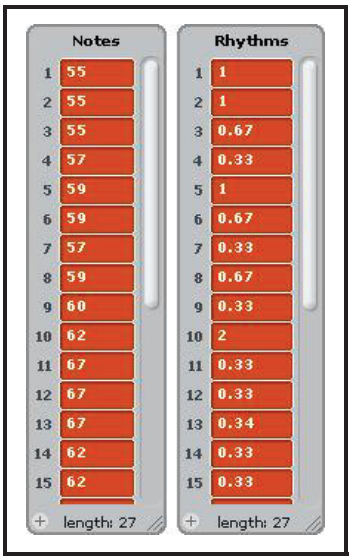
Figure 3. Two versions of Jimmy Page's *Kashmir* riff programmed in Scratch. [4]

Computing and Music (cont'd)

List and array data structures can be used to represent pitches and durations. Figure 4 shows an array (or indexed list) of MIDI note values paired with an array of note durations (in fractions of beats) that plays part of Row, Row, Row Your Boat. Using such structures, one can explore synchronization when the values are read by multiple threads with entrances staggered in time, resulting in the performance of a canon (or round).



4a



4b

Figure 4. Processing Scratch lists of notes and rhythms for *Row, Row, Row Your Boat.* [5]

References Cited

[1] Hoy, J. (2010). *Song flowchart for The Beatles' "All You Need is Love."* Created for a course assignment in "Sound Thinking."

[2] MIT Scratch Team (2009). *Scratch*. scratch.mit.edu accessed Dec. 21, 2009.

[3] Resnick, M., Maloney, J., Monroyhernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). *Scratch Programming for All.* Comm. of the ACM 52(11):60-67.

[4] Ruthmann, S.A. (2009). *Computational Zeppelin*. scratch.mit.edu/projects/alexruthmann/736779 accessed Jan. 5, 2010.

[5] Ruthmann, S.A., & Heines, J.M. (2010). *Exploring Musical and Computational Thinking Through Musical Live Coding in Scratch.* Scratch@MIT. Cambridge, MA.

[6] Smith, D.E. (1997). *Repeats, Second Endings, and Codas*. www.scenicnewengland.net/ uitar/notate/repeat.htm accessed Dec. 25, 2009.

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

THE SUN

July 3, 2016 SUNDAY Lowell, Massachusetts

UMass Lowell and Bartlett in harmony

Daniel Walzer, an assistant professor in UMass Lowell's Music Department, left, works with Jackeline Hernandez, center, and Kathryn Pen, fifth-graders at the Bartlett Community Partnership School in Lowell, as part of a new after-school program at BCPS. The program is teaching them how to write computer code while enhancing their understanding of music. A collaboration between BCPS and UMass Lowell, the "Teaching a Computer to Sing" project shows the students how to sing songs by such artists as Taylor Swift and Shawn Mendes, then teaches them to write code that programs computers to "sing" those songs, too. The project, which will resume in the fall, is funded through a National Science Foundation grant secured by Walzer and UMass Lowell Computer Science Professor Jesse Heines. PHOTO COURTESY JESSE HEINES/UMASS LOWELL

THE SUN
TUESDAY
January 24, 2017 Lowell, Massachusetts

Working on the singing computer program at the Bartlett Community Partnership School in Lowell Thursday are UMass Lowell student Mirza Gracia, 23, and Hazel Rivera, 11, a Bartlett sixth-grader. SUN PHOTO BY DAVID H. BROW

Teach a computer to sing? With UML help, Lowell students are

By Rick Sobey
rsobey@lowellsun.com

LOWELL — Seventh-grader Fernanda Lozano is trying to replicate "Take Me to Church" — the popular song you may have heard over and over again on the radio.

Sitting at a MacBook Air laptop, the 13-year-old girl is working on a program that creates musical scores.

Over comes UMass Lowell senior Nicole Vasconcelos, who tries to help Fernanda tweak her musical creation on the computer.

"The tempo is just a bit too slow," the 22-year-old college student tells the middle-schooler. "Sixty (beats per minute) is too slow, so let's try 80, a little faster."

Fernanda goes up to 80 beats per minute, then brings it down to 76.

"Just play around with the tempo a little bit, but this is awesome," Vasconcelos tells

Please see **COMPUTER/8**

Students teaching a computer to sing

COMPUTER/From Page 1

Fernanda. "You're doing great."

UMass Lowell's "Teach a Computer to Sing" afterschool program has been going on at Lowell's Bartlett Community Partnership School for about 18 months.

The school's music teacher, along with four assistants from the university and a retired computer-science professor, have worked with the students on learning to read music — and then transcribing the music in a computer program.

The afterschool program integrates computer programming and music education.

"We want to get them interested in computers through the music," said Jesse Heines, a retired UMass Lowell professor in computer science. "We want to interest students in pursuing further learning about computing."

The "Teach a Computer to Sing" program begins with the school's music teacher, Rachel Crawford, singing different popular hits with the students. For instance, last week she was singing Bruno Mars' "Just The Way You Are" with the middle-schoolers.

Crawford separates the sopranos and altos in the group, and helps them the learn the different parts. The goal is for the students to notice patterns in the music while singing, and then for them to code the music on the computer, Heines said.

Crawford teaches the students all about pitch and beat.

"It's gone very well," she said. "Every program has rocky starts, but we got over them and it's so successful right now. They're singing at assemblies and starting to see the fruits of their labor.

"It's been awesome working in partnership with a university," Crawford added. "It's a very cool program. If they make a mistake with the coding, there's always a backspace. They're not afraid of it at all."

Fernanda said she does a "bunch of experiments" with the program to try to get the song right.

"It's a lot of fun," added Saif AL-Rekabi, 13, a seventh-grader. "Even if we mess up, we get over it."

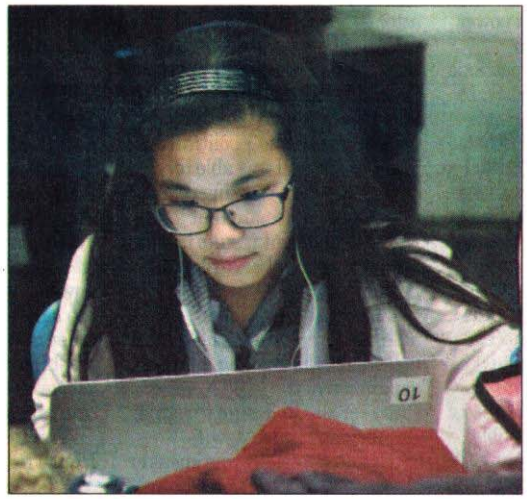
The afterschool program is funded by the National Science Foundation. Funding pays for a stipend for the UMass Lowell students, Heines and Crawford, as well as 30 MacBook Air laptops.

Out of the four assistants from UMass Lowell, three of them are music majors and the other studies computer science.

"The kids have been in school all day, so we don't make it too much of a class," Heines said. "The college students work with them one-on-one, and make it a lot of fun.

"I can't say enough about Rachel, the school and Lowell Public Schools," he added. "They've been super supportive. It wouldn't be possible without them."

UML students and professors that are working with Bartlett Community Partnership School in Lowell on a singing computer program at BCPS include, from left, Mirza Garcia, 23, Professor Jesse Heines, Professor Dan Walzer, Nicole Vasconcelos, 22, and at piano, Christian Hernandez, 22. SUN / DAVID H. BROW

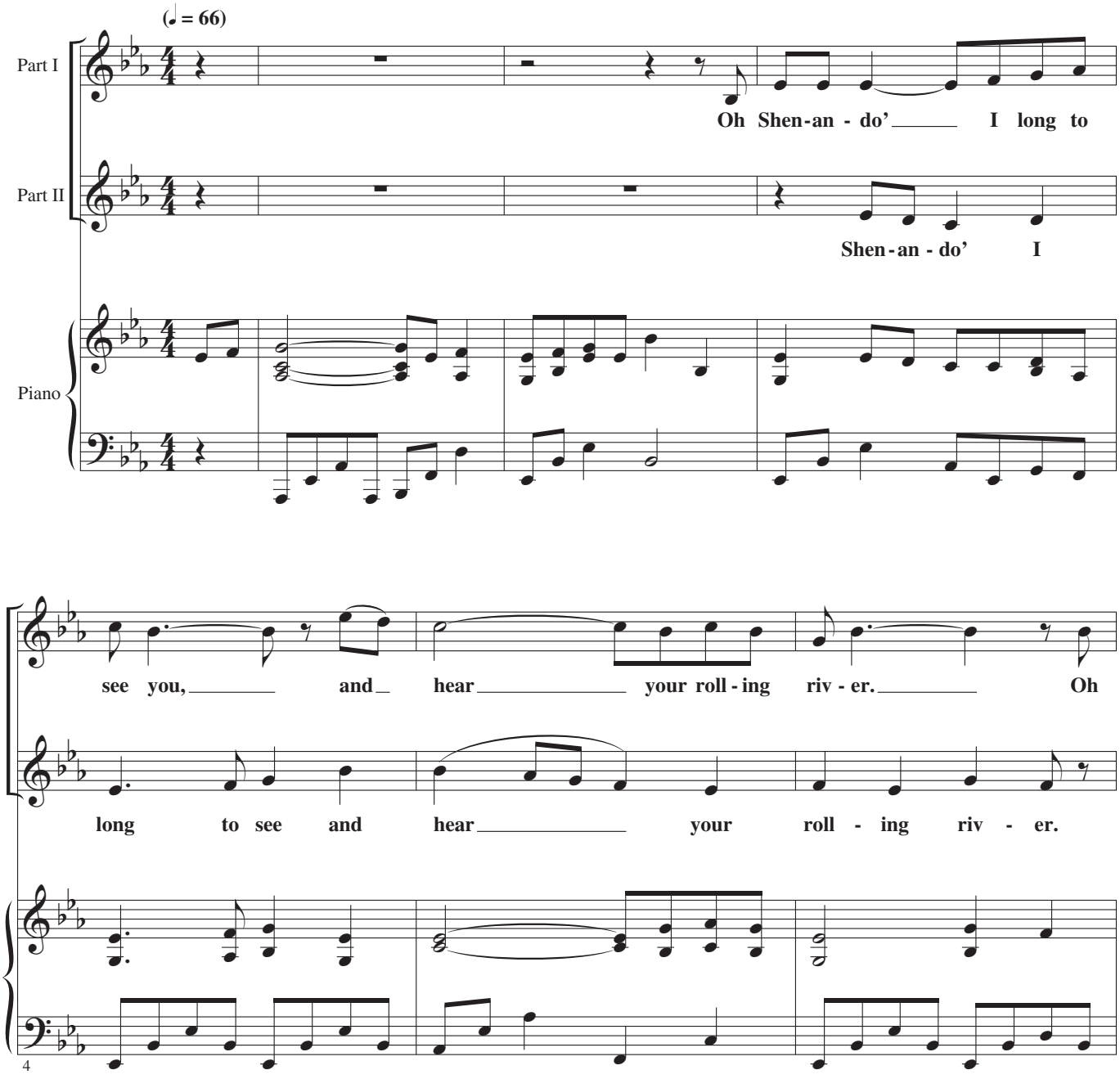


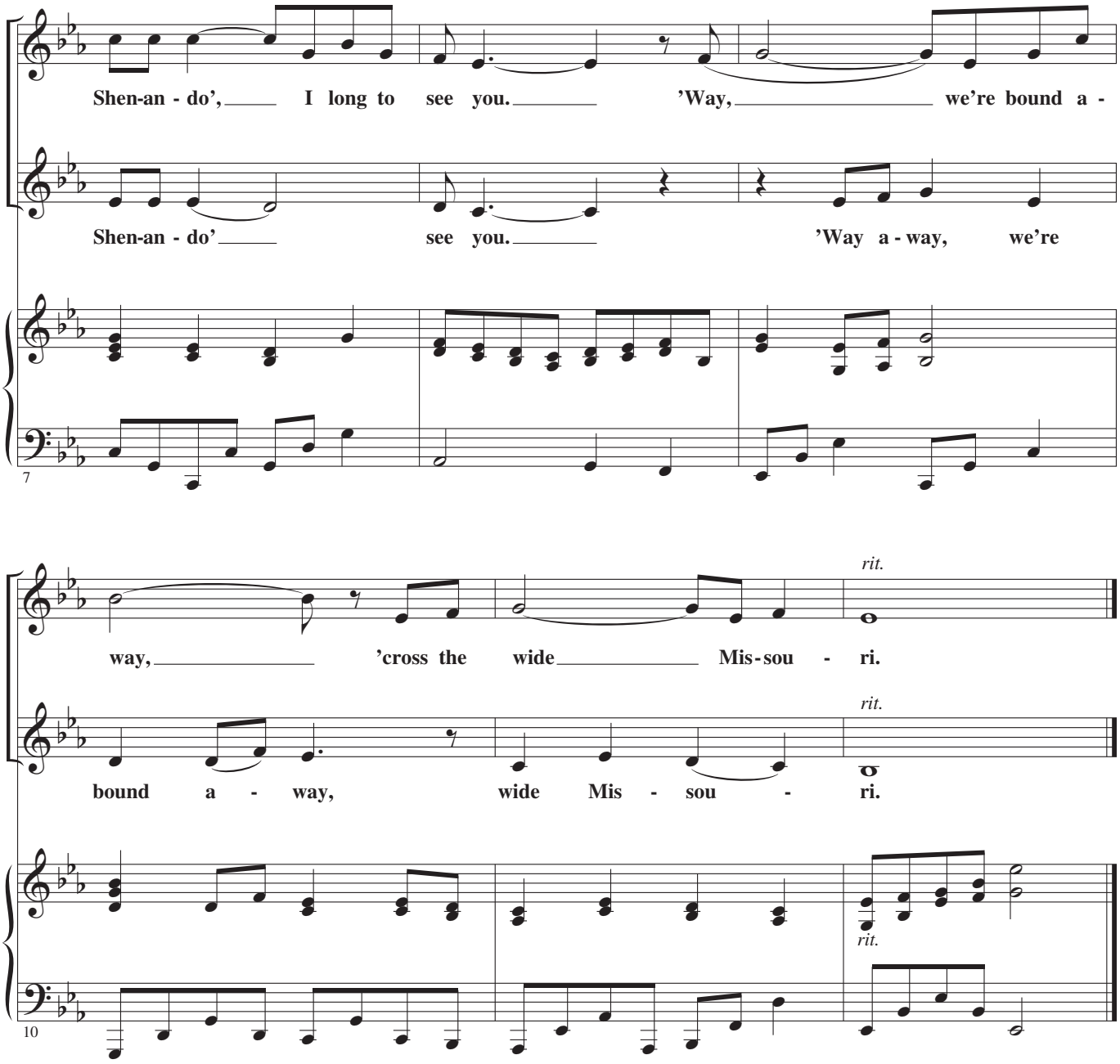
Sixth-grader Nelle Feliciano, 12, works on a computer in the singing computer program. **Watch video at lowellsun.com.** SUN/DAVID H. BROW

SHENANDOAH

2-Part Treble

American Folksong
Arranged by CATHERINE DELANOY




Shen-an - do', I long to see you. 'Way, we're bound a -
Shen-an - do' see you. 'Way a - way, we're
7
way, 'cross the wide Mis - sou - ri.
bound a - way, wide Mis - sou - ri.
rit.
rit.
rit.
10

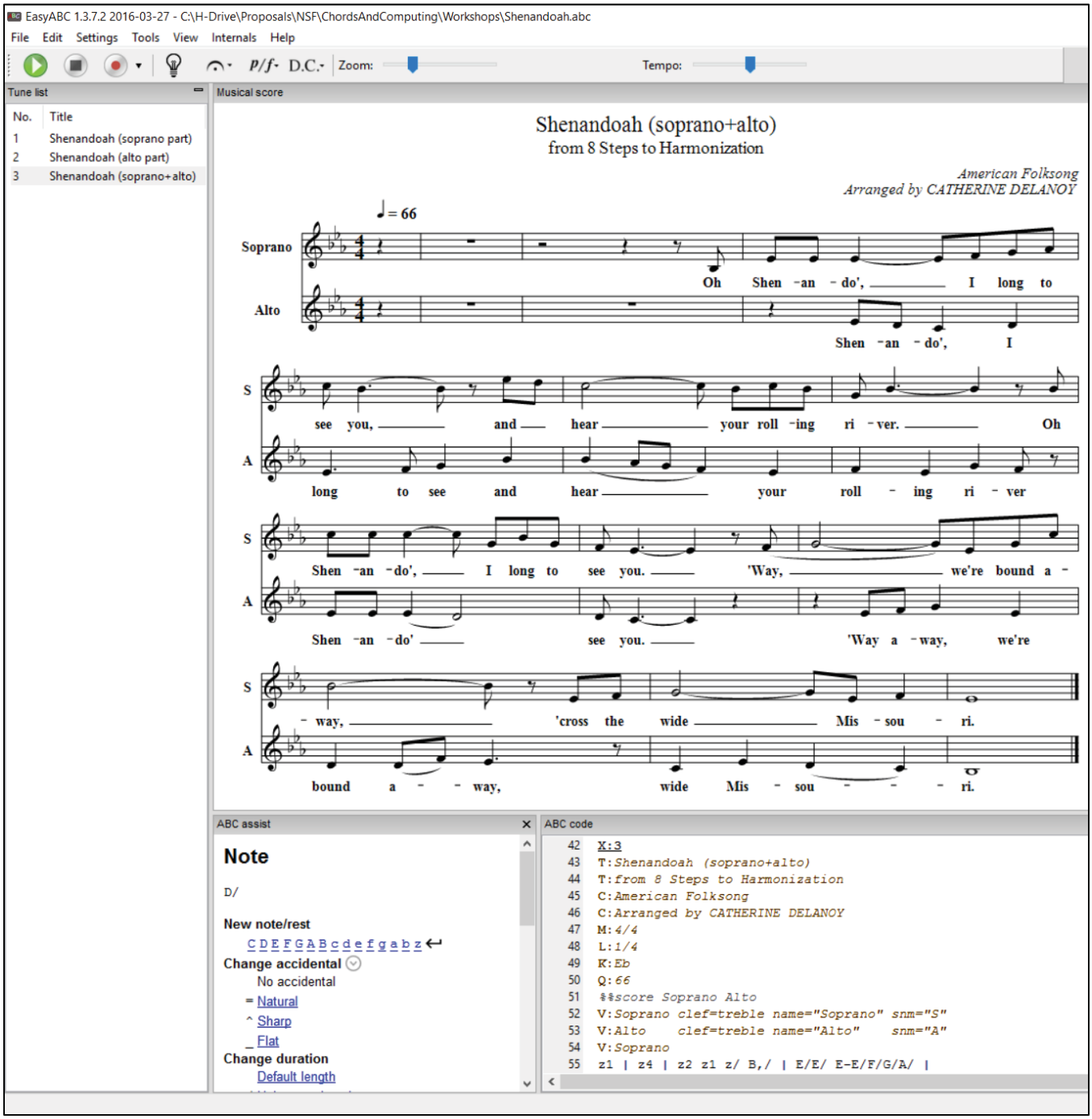
Shenandoah Representations

In ABC Notation

```
%abc-2.1

X:1
T:Shenandoah (soprano part)
T:from 8 Steps to Harmonization
C:American Folksong
C:Arranged by CATHERINE DELANOY
M:4/4
L:1/4
K:Eb
Q:66
z1 | z4 | z2 z1 z/ B,/ | E/E/ E-E/F/G/A/ |
w:Oh Shen-an-do',_ I long to
C'/2 B3/2-B/ z/ E'/-D'/ | C'2-C'/ B/C'/B/ | G/ B3/2-B z/ B/ |
w:see you,_ and_ hear_ your roll-ing ri-ver._ Oh
C'/C'/ C'-C'/ G/B/G/ | F/ E3/2-E z/ (F/ | G2-G/)E/G/C'/ |
w:Shen-an-do',_ I long to see you._ 'Way,__ we're bound a-
B2-B/ z/ E/F/ | G2-G/E/ F | E4 |]
w:way,_ 'cross the wide_ Mis-sou-ri.

X:2
T:Shenandoah (alto part)
T:from 8 Steps to Harmonization
C:American Folksong
C:Arranged by CATHERINE DELANOY
M:4/4
L:1/4
K:Eb
Q:66
%%score Alto
V:Alto    clef=treble name="Alto"    snm="A"
[V:Alto]
z1 | z4 | z4 | z E/D/ C D |
w:Shen-an-do', I
E3/2 F/ G B | (B A/G/ F) E | F E G F/ z/ |
w:long to see and hear___ your roll-ing ri-ver
E/E/ (E D2) | D/ C3/2-C z | z E/F/ G E |
w:Shen-an-do'_ see you._ 'Way a-way, we're
D (D/F/) E3/2 z/ | C E (D C) | B,4 |]
w:bound a--way, wide Mis-sou--ri.

X:3
T:Shenandoah (soprano+alto)
T:from 8 Steps to Harmonization
C:American Folksong
C:Arranged by CATHERINE DELANOY
```

```
M:4/4
L:1/4
K:Eb
Q:66
%%score Soprano Alto
V:Soprano clef=treble name="Soprano" snm="S"
V:Alto    clef=treble name="Alto"    snm="A"
V:Soprano
z1 | z4 | z2 z1 z/ B,/ | E/E/ E-E/F/G/A/ |
w:Oh Shen-an-do',_ I long to
C'/2 B3/2-B/ z/ E'/-D'/ | C'2-C'/ B/C'/B/ | G/ B3/2-B z/ B/ |
w:see you,_ and_ hear_ your roll-ing ri-ver._ Oh
C'/C'/ C'-C'/ G/B/G/ | F/ E3/2-E z/ (F/ | G2-G/)E/G/C'/ |
w:Shen-an-do',_ I long to see you._ 'Way,__ we're bound a-
B2-B/ z/ E/F/ | G2-G/E/ F | E4 |]
w:way,_ 'cross the wide_ Mis-sou-ri.
V:Alto
z1 | z4 | z4 | z E/D/ C D |
w:Shen-an-do', I
E3/2 F/ G B | (B A/G/ F) E | F E G F/ z/ |
w:long to see and hear___ your roll-ing ri-ver
E/E/ (E D2) | D/ C3/2-C z | z E/F/ G E |
w:Shen-an-do'_ see you._ 'Way a-way, we're
D (D/F/) E3/2 z/ | C E (D C) | B,4 |]
w:bound a--way, wide Mis-sou--ri.
```

As Displayed in EasyABC



In Pencil Code with Accidentals

Coded by 5th grade student Breanna

http://breannasings.pencilcode.net/edit/shenandoah

```
 1  write 'Oh Shenan do I long to '
 2  play '_B,/ _E/2_E/2 _E-_E/2F/2G/2_A/2'
 3  write 'see you and '
 4  play 'c/2 _B3/2-_B/2 z/2 _e/2d/2'
 5  write 'hear your rolling '
 6  play 'c2-c/_B/c/2_B/2'
 7  write 'river Oh'
 8  play 'G/2 _B3/2-_B Z/2 B/2'
 9  #B E A need underscore
10  write 'Shenandoah I long to'
11  play 'c/c c-c/2G_B/2G/2'
12  write 'see you way'
13  play 'F/2 _E3/2 z/2 F/2'
```

In Pencil Code Using a Function

Coded by 6th grade student Kathryn (with a little help)

http://kk123.pencilcode.net/edit/shenandoah

```
 1  key= "[K:Eb]"
 2  time= "[M:4/4]"
 3  tempo= "[Q:120]"
 4  prefix=time+key+tempo
 5
 6  p = []
 7  p[1]=new Piano
 8  p[2]=new Piano
 9  p[2].fd 150
10  sync p[1],p[2]
11
12  myplay = ( np, notes ) ->
13    p[np].play prefix + notes
14    write prefix + notes
15
16  if true
17    # myplay 'z2z1'
18    myplay 1, 'z/2 _B,/2'
19    myplay 1, 'E/2 _E/2 _E-E/2 F/2 G/2 _A/2'
20    myplay 1, "C'/2 _B3/2-_B/2 z/2"
21    myplay 1, "E'/2-D'/2 C'2-C'/2 B/2C'/2B/2 G/2 B3/2-B z/2"
22    myplay 1, "B/2 C'/2C'/2 C'-C'/2G/2B/2G/2 F/2 E3/2-E z/2"
23    myplay 1, "F/2 G2-G/2 E/2 G/2 C'/2 B2-B/2 z/2"
24    myplay 1, "E/2 F/2 G2-G/2 E/2 F E4"
25
26  if true
27    myplay 2, "z/2 z E/2 D/2 C D E3/2 F/2 G B"
28    myplay 2, "B A/2 G/2 F E F E G F/2"
29    myplay 2, "z/2 E/2 E/2 E-D2 D/2 C3/2-CZ"
30    myplay 2, "z E/2 F/2 G E D D/2-F/2 E3/2"
31    myplay 2, "z/2 C E D C B"
```

In Pencil Code Using Our Custom tacts Functions

Coded by 6th grade student Nelle (with very little help)

http://kittygurl.pencilcode.net/edit/Shenandoah2

```
 1  # load the custom tacts functions
 2  write '<script src="http://kittygurl.pencilcode.net/code/tacts">
<\/script>'
 3  # note the \ before the / in <\/script>
 4
 5  tacts.CreatePianos 2
 6  tacts.SetKey 'Eb'
 7  tacts.SetTempo 90
 8
 9  tacts.sing 1, 'z z4 z2 z z/2 B,/2 E/2 E/2 E-E/2 F/2 G/2 A/2'
10  tacts.sing 1, 'c/2 B3/2-B/2 z/2 e/2-d/2 c2-c/2 B/2 c/2 B/2'
11  tacts.sing 1, 'G/2 B3/2-B z/2 B/2'
12  tacts.sing 1, 'c/2 c/2 c-c/2 G/2 B/2 G/2 F/2 E3/2-E'
13  tacts.sing 1, 'z/2 F/2 G2-G/2 E/2 G/2 c/2 B2-B/2'
14  tacts.sing 1, 'z/2 E/2 F/2 G2-G/2 E/2 F E4'
15
16  tacts.sing 2, 'z z4 z4 z E/2 D/2 C D E3/2 F/2 G B B-A/2-G/2-F'
17  tacts.sing 2, 'E F E G F/2 z/2 E/2 E/2 E-D2 D/2 C3/2-C z'
18  tacts.sing 2, 'z E/2 F/2 G E D D/2-F/2 E3/2 z/2'
19  tacts.sing 2, 'C E D-C B,4'
```

In Pencil Code Using Our Custom TACTS singsay Function

Coded by Jesse

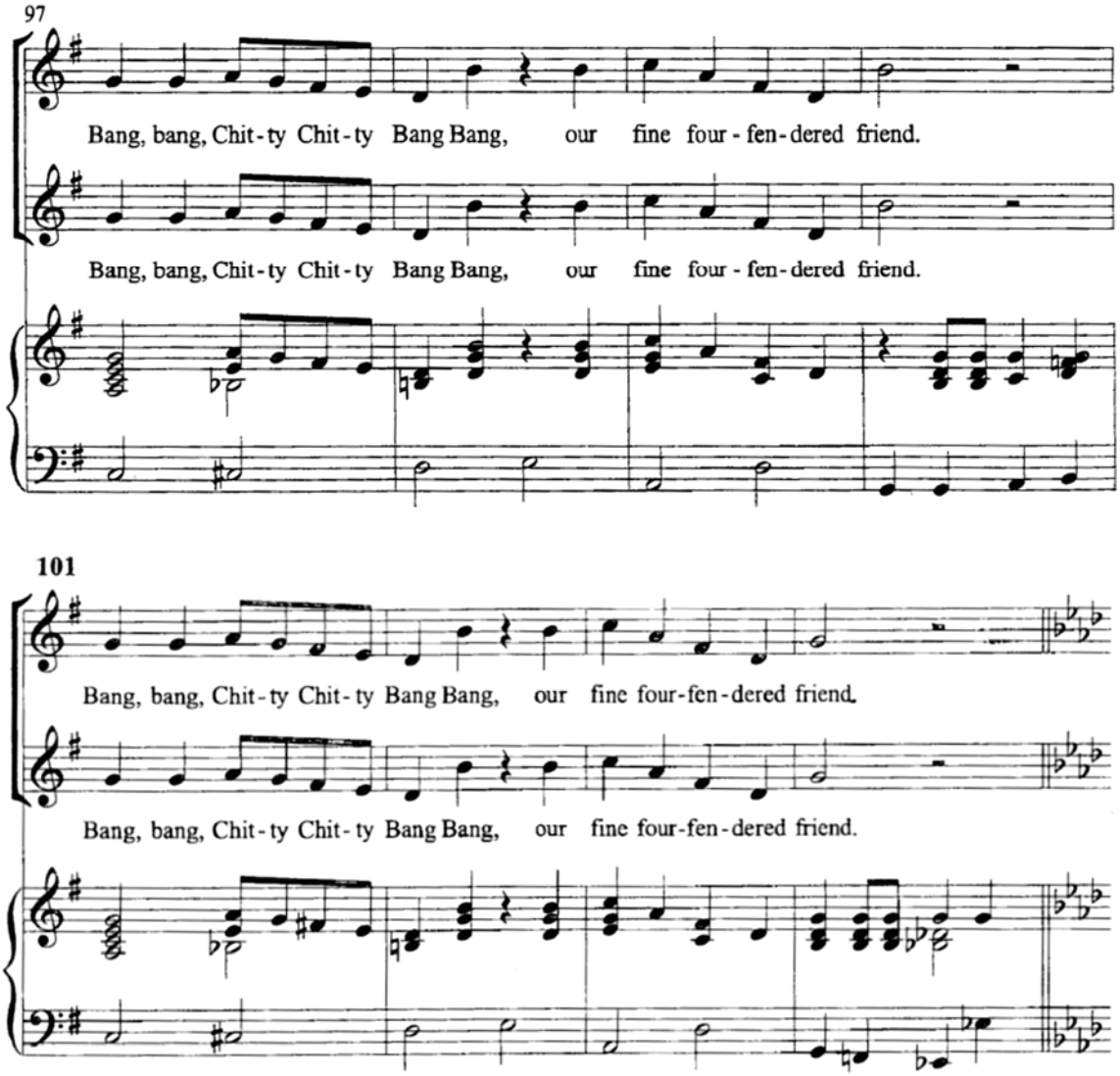
http://drjay.pencilcode.net/edit/Shenando-Say_v2

```
 1  # Shenandoah - 1 part with lyrics
 2  # using the tacts library
 3  # coded by Jesse Heines, February 27, 2017
 4
 5  # load the custom tacts functions
 6  write '<script src="http://drjay.pencilcode.net/code/tacts">
<\/script>'
 7  # note the \ before the / in <\/script>
 8
 9  # set the key, time signature, and tempo
10  #   tempo = quarter notes per minute
11  tacts.SetKey "Eb"     # SET TO YOUR KEY
12  tacts.SetTime "4/4"   # SET TO YOUR METER
13  tacts.SetTempo 66     # SET TO YOUR TEMPO
14
15  # create and initialize 2 pianos
16  tacts.CreatePianos 1  # SET THIS TO THE NUMBER OF PIANOS YOU WANT
17
18  # Part 1 - Melody
19  if true
20    tacts.singsay 1, "z/", ""
21    tacts.singsay 1, "D/", "Oh"
22    tacts.singsay 1, "E/", " Shen"
23    tacts.singsay 1, "E/", "an"
24    tacts.singsay 1, "E3/2", "do'"
25    tacts.singsay 1, "F/", " I"
26    tacts.singsay 1, "G/", " long"
27    tacts.singsay 1, "A/", " to"
28    tacts.singsay 1, "C'/", " see"
29    tacts.singsay 1, "B2", " you,"
30    tacts.singsay 1, "z/", "|"
31    tacts.singsay 1, "E'/", "and"
32    tacts.singsay 1, "D'/", "__"
33    tacts.singsay 1, "C'5/2", " hear"
34    tacts.singsay 1, "B/", " your"
35    tacts.singsay 1, "C'/", " roll"
36    tacts.singsay 1, "B/", "ing"
37    tacts.singsay 1, "G/", " riv"
38    tacts.singsay 1, "B5/2","er."
39    tacts.singsay 1, "z/", "|"
40    tacts.singsay 1, "B/", "Oh"
41    tacts.singsay 1, "C'/", " Shen"
42    tacts.singsay 1, "C'/", "an"
43    tacts.singsay 1, "C'3/2", "do'"
44    tacts.singsay 1, "G/", " I"
45    tacts.singsay 1, "B/", " long"
46    tacts.singsay 1, "G/", " to"
47    tacts.singsay 1, "F/", " see"
```

```
48    tacts.singsay 1, "E5/2", " you,"
49    tacts.singsay 1, "z/", "|"
50    tacts.singsay 1, "F/", "'Way"
51    tacts.singsay 1, "G5/2", "__"
52    tacts.singsay 1, "E/", " we're"
53    tacts.singsay 1, "G/", " bound"
54    tacts.singsay 1, "C'/", " a"
55    tacts.singsay 1, "B5/2", "way,"
56    tacts.singsay 1, "z/", "|"
57    tacts.singsay 1, "E/", "'cross"
58    tacts.singsay 1, "F/", " the"
59    tacts.singsay 1, "G5/2", " wide"
60    tacts.singsay 1, "E/", " Mis"
61    tacts.singsay 1, "F", "sou"
62    tacts.singsay 1, "E4", "ri."
```

First and Second Endings Exercise

Score



Exercise (instructions given verbally)

1. Look at measures 97 to 104.
2. Note that measures 97-99 and 101-103 are identical.
3. Note that there is only one note in measure 100 that differs in measure 104.
4. Code measures 97-99 in a loop so that they play twice.
   a. Be sure to use the type of loop that has a variable counter (`for x in [1..2]`).

b. Be sure to define a variable to hold the key (`key = "[K:G]"`) and then *concatenate* (yes, we really used this CS word with the students) this variable to each string of notes to be played (`play key + "G G A/G/F/E/"`).

5. Add an `if/else` statement to play the `B2` the first time through and the `G2` the second time through.

Student Program Examples

Coded by 6th grade student Jackeline (with a little help)

`http://titi4life.pencilcode.net/edit/FlixAreForKids`

```
 1  key = "[K:G]"
 2  P = new Piano()
 3  for x in [1..2]
 4    P.play key+'G G A/2 G/2 F/2 E/2'
 5    P.play key+'D B z B'
 6    P.play key+'c A F D'
 7    if x is 1
 8      P.play key+'B2 z2'
 9    else
10      P.play key+'G2 z2'
```

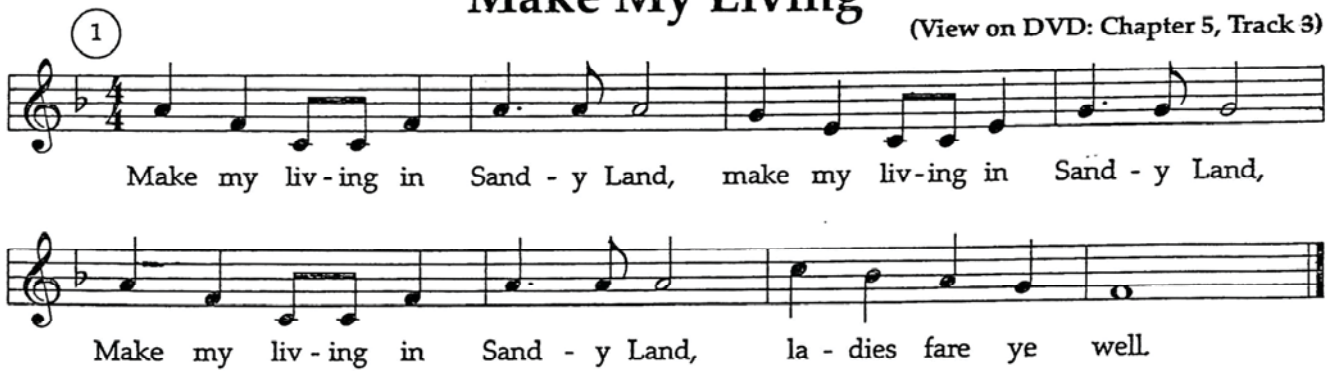
Coded by 7th grade student Saif (with a little help)

`http://safe.pencilcode.net/edit/FLIX-IFStatement`

```
 1  key = "[K:G]"
 2
 3  for x in [1..2]
 4    play key+ "G G A/G/F/E/ D B Z1 B C' A F D"
 5    if x is 1
 6      play key+ 'B2 Z2'
 7    else
 8      play key+ 'G2 Z2'
```

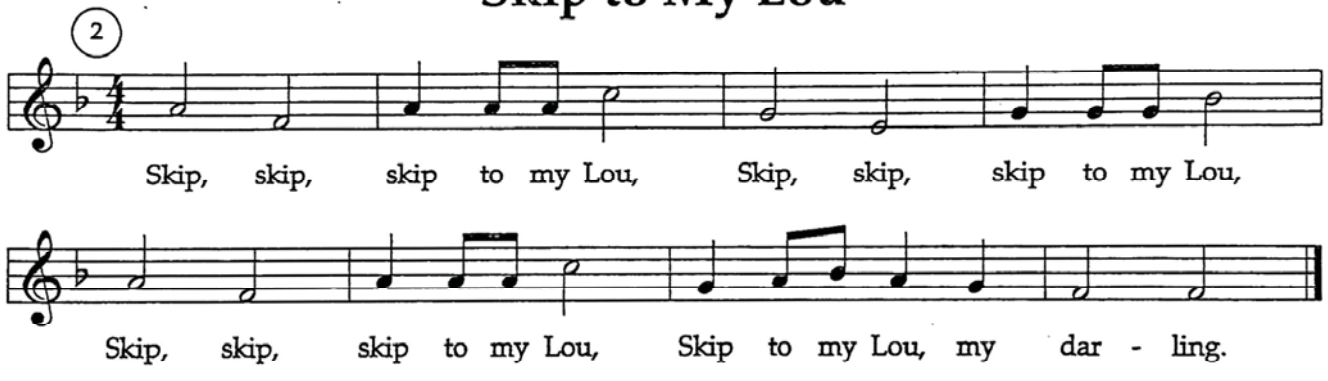
Make My Living

(View on DVD: Chapter 5, Track 3)



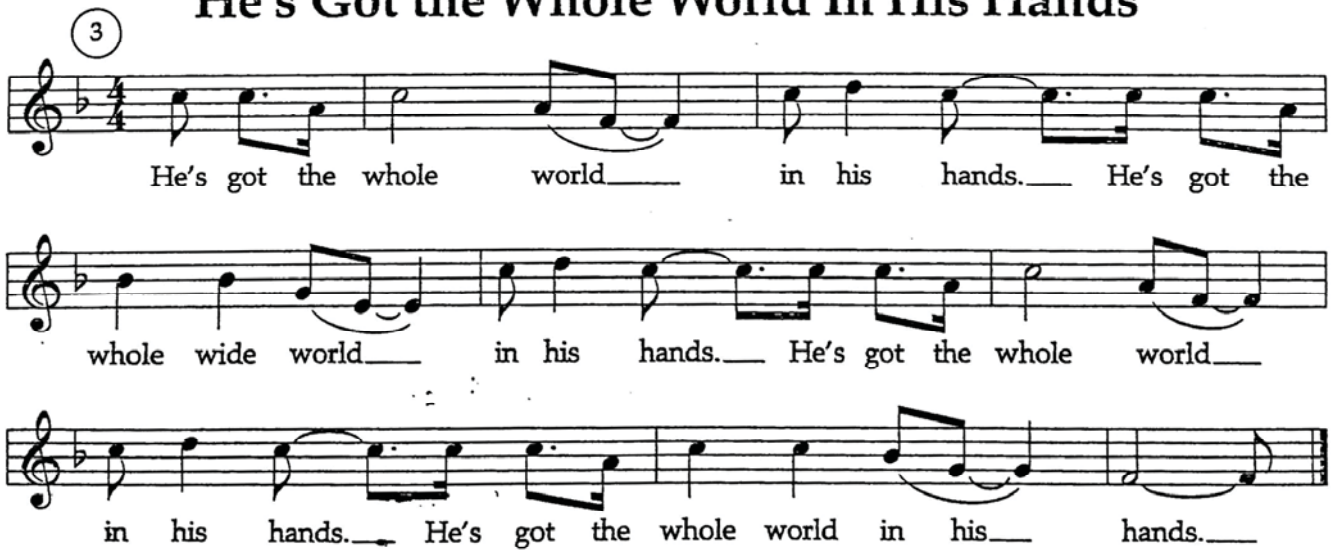
Traditional.

Skip to My Lou



Traditional.

He's Got the Whole World In His Hands

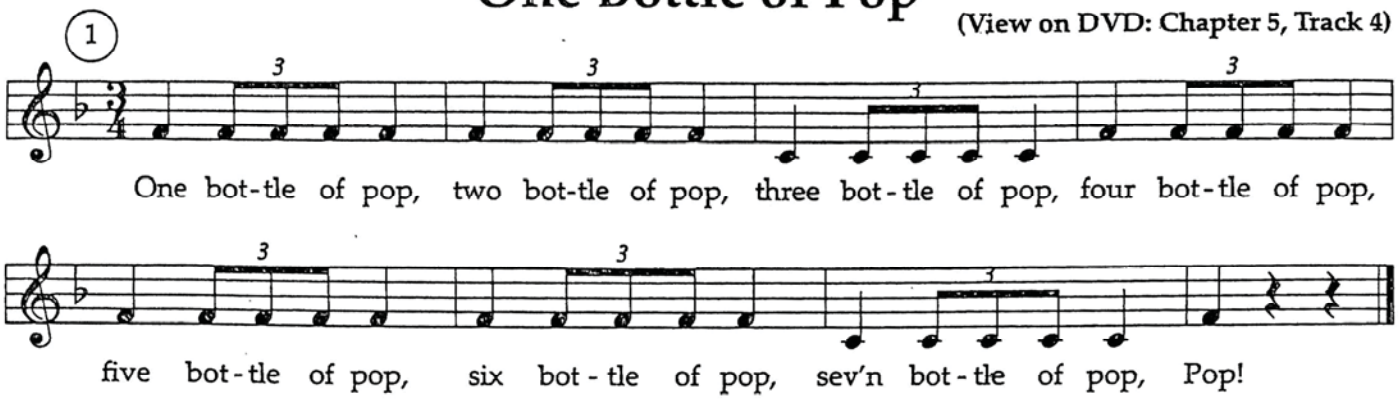


Traditional.

DIRECTIONS:
Sing the three songs together as partner songs. Note: *Make My Living* and *Skip to My Lou* begin on the first full measure of *He's Got the Whole World In His Hands.*

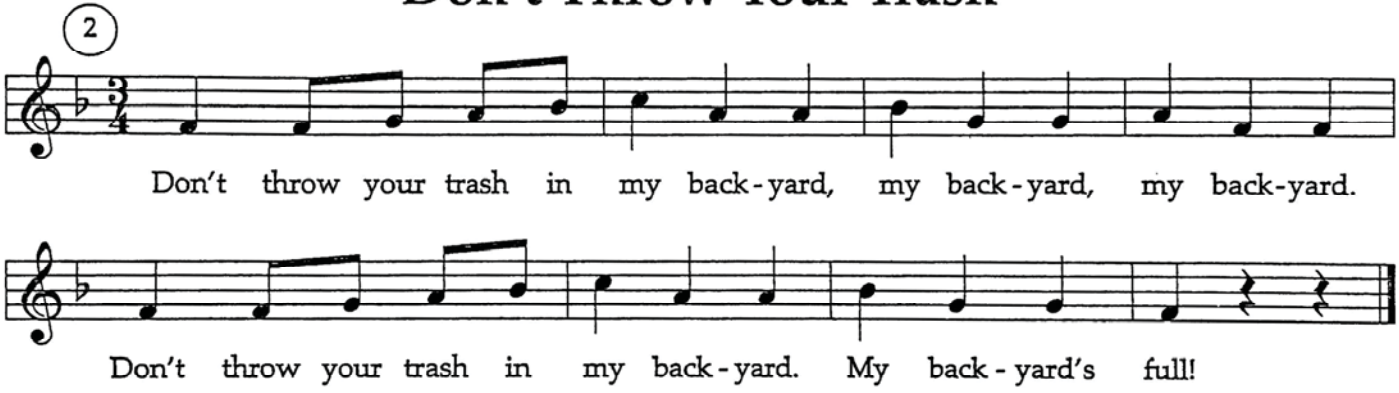
One Bottle of Pop

(View on DVD: Chapter 5, Track 4)



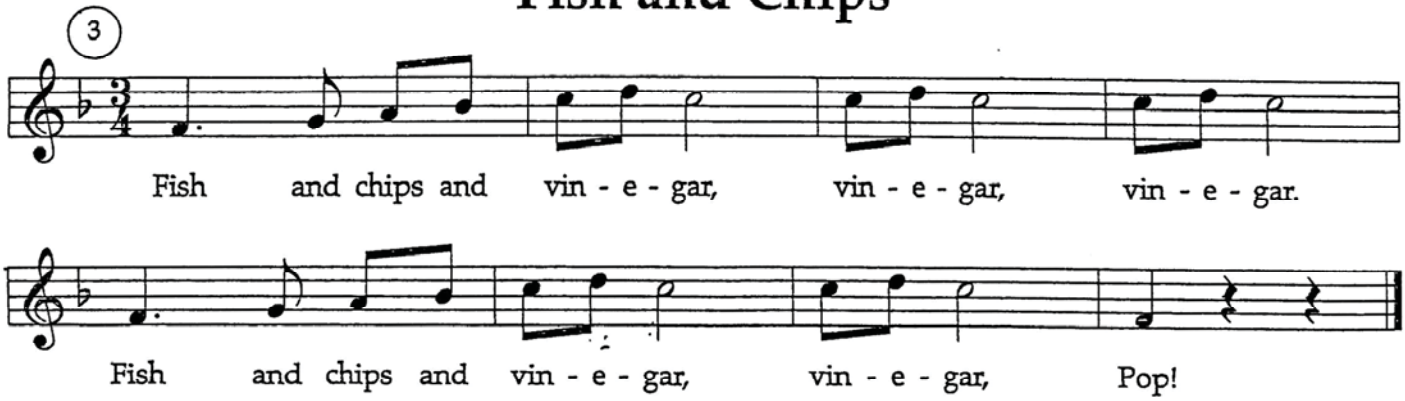
Traditional.

Don't Throw Your Trash



Traditional.

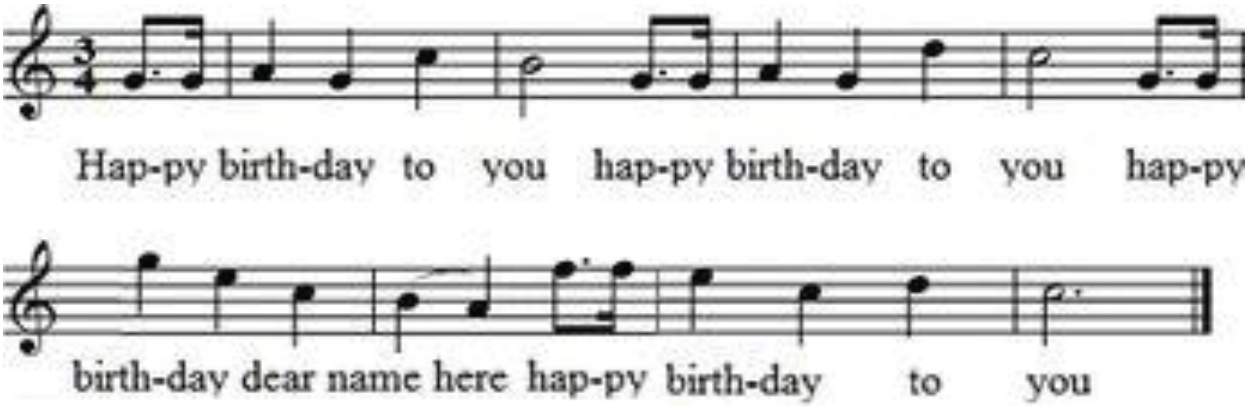
Fish and Chips

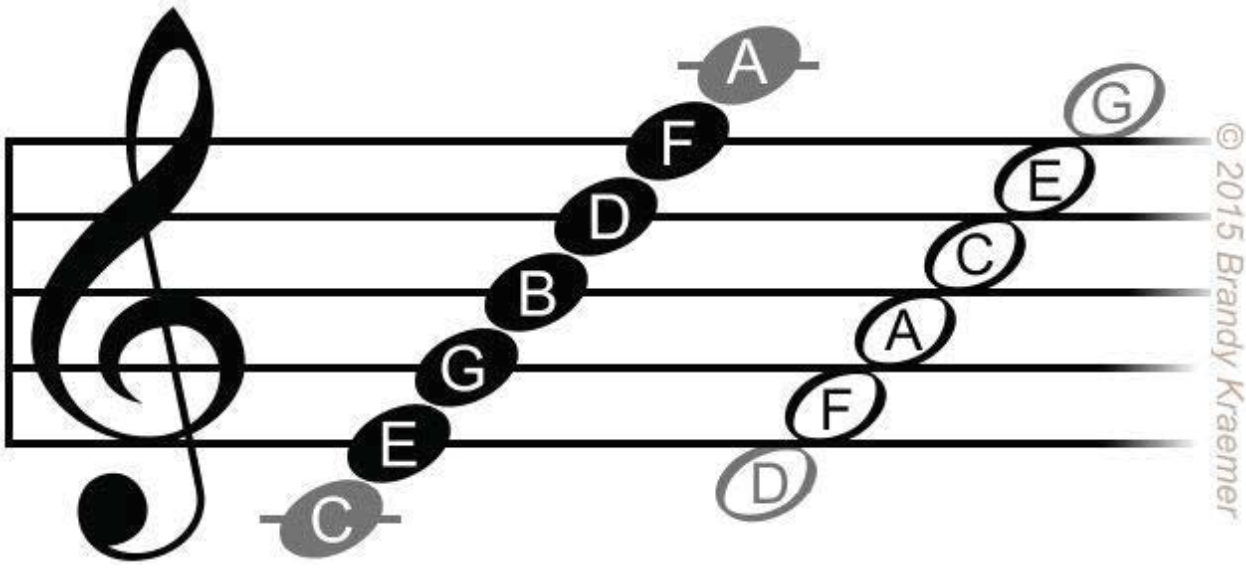


Traditional.

DIRECTIONS:
Sing the three songs together as partner songs.

HAPPY BIRTHDAY

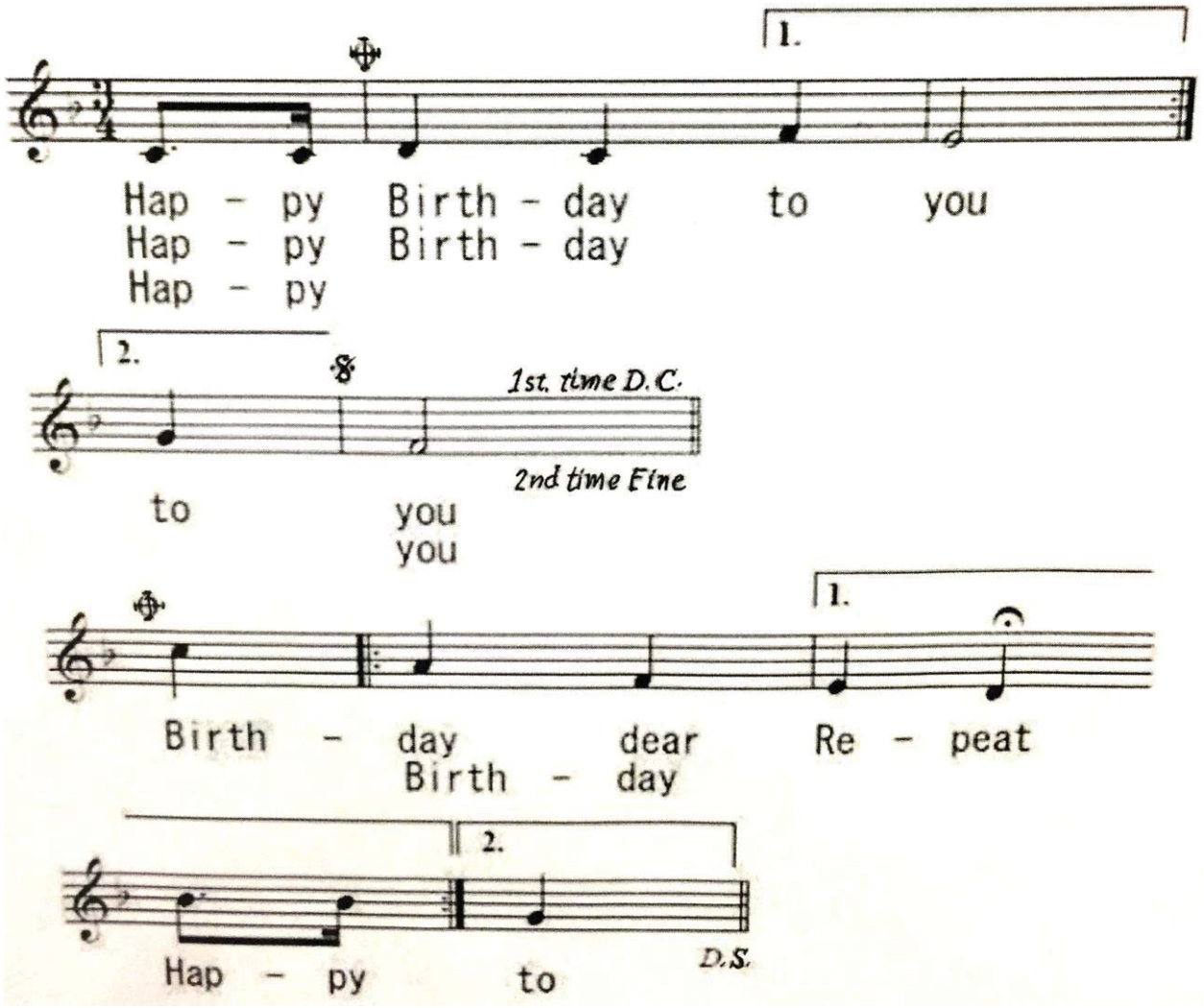




Sources:

- http://www.irish-folk-songs.com/uploads/4/3/3/6/43368469/8824529.jpg?516
- http://f.tqn.com/y/piano/1/S/G/f/-/-/Treble-notes_Grand-Staff.png

HAPPY BIRTHDAY
using Repeats for Minimal Notes



ABC Quick Reference Card

Version 0.6 – ABC v2.1 & some abcm2ps v6 – Steve Merrony

Information & Headers

```
%abc                        %abc-2.1 – First Line of file
B:book
C:composer
D:discography
F:file URL
G:group                     G:flute
H:history
I:instruction
K:key [clef] [middle=x]     K:G, K:AMix – Last Header*
L:note unit length          L:1/8
M:metre                     M:4/4
m:macro                     m: ~n2 = (3o/n/m/ n
N:notes
O:origin
P:parts                     P:A, P:ABAC, P:(A2B)3
Q:tempo                     Q:"allegro" 1/4=120
R:rhythm                    R:R, R:reel
r:remark
S:source
s:symbol line               s: !pp! ** !f!
T:title                     Second Header
U:user defined              U: T = !trill!
V:voice                     V:4 clef=bass**
W:words                     Lyrics after tune
w:words                     Lyrics inline
X:reference number          First Header
Z:transcriber               Also put copyright info here
```

* Clefs

```
treble                      Treble (default)
treble-8                    Treble 8ve below eg.tenors
treble+8                    Treble 8ve above eg.piccolo
bass                        Bass
bass3                       Baritone
alto4                       Tenor
alto                        Alto
alto2                       Mezzosoprano
alto1                       Soprano
none                        No clef
perc                        Percussion
```

** Voices

The voice name is a digit or a word followed by:

```
clef=                 clef of the voice
perc                  percussion staff
name=xxx              name at the left of the first staff
sname=xxx             name that appears left of later staves
merge                 indicates that this voice belongs to
                      the same staff as the previous voice
up or down            forces the note stem direction
gstem=up/down/auto    forces the grace note stem direction
stem=up/down/auto     forces the note stem direction
dyn=up/down/auto      forces the placement of dynamic marks
lyrics=up/down/auto   forces the placement of the lyrics
middle=<note>         specify name of note on middle line
staffscale=n          sets the scale of the associated staff
stafflines=n          sets the number of lines of the staff
```

Extended Information Fields

```
%%abc-creator mxml2abc-0.6
%%abc-version 2.0                 % use for pre-2.1 ABC
%%abc-charset iso-8859-1          % default is UTF-8
%%abc-include mydefs.abh
```

Formatting

Page Format

```
%%pageheight     <length>     %%rightmargin    <length>
%%pagewidth      <length>     %%indent         <length>
%%topmargin      <length>     %%landscape      <logical>
%%botmargin      <length>     %%staffwidth     <length>
%%leftmargin     <length>
```

Fonts (Most take a font name and size as parameters)

```
%%titlefont                   %%measurefont
%%subtitlefont                %%repeatfont
%%composerfont                %%textfont
%%partsfont                   %%voicefont
%%tempofont                   %%vocalfont      % w:
%%gchordfont                  %%wordsfont      % W:
%%headerfont
%%historyfont                 The specifiers $1, $2, $3 and
%%footerfont                  $4 can be used to change the
%%annotationfont              font within an ABC string
%%infofont                    %%setfont-1/2/3/4 <ft> <sz>
```

Spacing

```
%%topspace       <length>     %%titlecaps      <logical>
%%titlespace     <length>     %%subtitlespace  <length>
%%titleleft      <logical>    %%composerspace  <length>
%%aligncomposer  <integer> % -1 left, 0 centre, !1 right
%%musicspace     <length>  % between comp. & 1st staff
%%partsspace     <length>     %%wordsspace     <length>
%%vocalspace     <length>     %%textspace      <length>
%%infospace      <length>
%%staffsep       <length>  % between systems
%%sysstaffsep    <length>  % between staves
%%barsperstaff   <integer>
%%parskipfac     <number>  % space between parts
%%lineskipfac    <number>  % space between lines of text
%%stretchstaff   <logical>    %%stretchlast    <logical>
%%maxshrink      <number>  % shrinking notes
%%maxstaffsep    <length>     %%splittune      <logical>
%%maxsysstaffsep <length>
%%newpage
%%scale          <number>  % scale EVERYTHING
%%staves                   % eg. [(SA) (TB)] for short score
%%vksip          <length>
```

Measures/Bars

```
%%measurefirst     <integer> % number of 1st measure
%%barnumbers       <integer> % numbers every n bars
%%measurenb        <integer> % same as %%barnumbers
%%measurebox       <logical>
%%setbarnb         <integer> % set measure #
%%contbarnb        <logical> % repeat numbering
%%alignbars        <integer> % align bars for next n lines
```

Text

```
%%text           <ABC String>  %%endtext
%%textoption     <string>      %%sep       [# # #]
%%center         <ABC String>  %%header
%%begintext                    %%footer
```

Misc

```
%%autoclef          <logical>
%%beginps …  %%endps
%%bstemdown         <logical>
%%comball           <logical>
%%combinevoices     <logical>
%%dateformat        <string>
%%encoding          <integer> % 0 – 6 for ISO-Latin-n
%%EPS               <string>  % include external EPS file
%%exprabove         <logical>
%%exprbelow         <logical>
%%flatbeams         <logical>
%%freegchord        <logical> % print '#', 'b' and '=' stet
%%gchordbox         <logical>
%%graceslurs        <logical>
%%hyphencont        <logical>
%%infoline          <logical> % rhythm & origin on same line
%%linebreak         $ or ! or <EOL> or <none>
%%musiconly         <logical> % don't output lyrics
%%oneperpage        <logical>
%%partsbox          <logical>
%%postscript        <string>
%%printparts        <logical> % P:
%%printtempo        <logical>
%%sep               <length> <length> <length> % separator
%%shiftunison       <logical>
%%squarebreve       <logical>
%%slurheight        <float>   % <1.0 to flatten
%%staff             <number>  % use nth staff
%%staffnonote       <logical> % don't show empty staves
%%stemheight        <float>
%%straightflags     <logical>
%%timewarn          <logical> % cautionary time sigs
%%tuplets           <integer> <integer> <integer>
%%vocalabove        <logical>
%%withxrefs         <logical> % print X: index in title
%%writefields       <list of field identifiers> [<logical>]
%%writehistory      <logical>
```

Notes

```
C,,     C,      C (middle-C)     c       c'      c''
```

```
^       sharp                 ( )     slur
^^      double sharp          .-      dotted tie
=       natural               .( )    dotted slur
_       flat                  -,      tie below
__      double flat           (' )    slur above
z       rest                  {/g}    grace note
Z       bar's rest            (3      3 notes in time of 2
x       invisible rest        (p:q:r  p notes in time of q
X       invisible bar rest            for next r notes
y       spacer                [ceg]   chord
-       tie                   &       overlay temp voice
```

Repeats and Bars

```
|       |]      ||      [|      |:      :|      ::
[|]             invisible bar line
.|              dotted bar line
[1      [2      1st and 2nd time repeats
```

Chord Symbols

```
m or min        minor e.g. "Am7"A2D2
maj             major
dim             diminished
aug or +        augmented
sus             sustained
7, 9 ...        7th, 9th, etc.
```

Lyrics

```
-       break between syllables within a word
        last syllable is to be held for an extra note
*       one note is skipped
~       appears as a space; puts multiple words under note
\-      appears as hyphen; multiple syllables under note
|       advances to the next bar
```

Decorations

Short Forms

.	Staccato	S	segno
~	Irish roll/turn	O	coda
T	trill	J	slide
H	fermata	R	roll
L	accent or emphasis	u	upbow
M	inverted mordent	v	downbow
P	mordent		

Dynamics

```
!crescendo(!  or !<(!    start of a < crescendo mark
!crescendo)!  or !<)!    end of a < crescendo mark,
                         placed after the last note
!diminuendo(! or !>(!    start of a > diminuendo mark
!diminuendo)! or !>)!    end of a > diminuendo mark,
                         placed after the last note
!pppp! !ppp! !pp! !p! !mp! !mf! !f! !ff! !fff! !ffff! !sfz!
```

Ornaments

```
!trill!          "tr" trill, also !trill(! and !trill)!
!lowermordent!   short squiggle with a line through it
!uppermordent!   short squiggle
!mordent!        same as !lowermordent!
!pralltriller!   same as !uppermordent!
!accent!         > mark
!>!              same as !accent!
!emphasis!       same as !accent!
!fermata!        fermata or hold (arc above dot)
!invertedfermata!
!tenuto!         horizontal bar
!trem1/2/3/4!    tremolo (on 2nd note)
!xstem!          stem up to staff above
!slide!          !turnx!          !invertedturn! !
!invertedturnx!  !arpeggio!
```

Text Annotations

```
"..."
"^..."           above staff      "_..."  below staff
"<..."           left of note     ">..."  right of note
"@x,y ..."       explicit offset
```

Fingering & Phrasing

```
!0! - !5!        fingerings
!plus!           left-hand pizzicato, or rasp for brass
!wedge!          small filled-in wedge mark
!open!           small circle above note indicating
                 open string or harmonic
!thumb!          cello thumb symbol
!snap!           snap-pizzicato mark
!turn!           a turn mark
!roll!           a roll mark (arc) as used in Irish music
!breath!         a breath mark (apostrophe) after note
!shortphrase!    vertical line on the upper part of staff
!mediumphrase!   same, but extending down to the middle
!longphrase!     same, but extending 3/4 of the way down
!upbow!          V mark
!downbow!        squared n mark
```

Repeat/Section Symbols

```
!segno!          2 s-like symbols separated by diagonal
!coda!           a ring with a cross in it
!D.S.!           the letters D.S. (Da Segno)
!D.C.!           the letters D.C. (Da Coda or Da Capo)
!dacoda!         !dacapo!         !fine!
```

Correct Order of Elements

<grace notes><chord symbols><decorations><accidentals>
<NOTE><octave><note length><tie>

General MIDI Instruments

Specify inline with [I: MIDI=program *<n - 1>*]

1 Acoustic Grand	22 Accordion
2 Bright Acoustic	23 Harmonica
3 Electric Grand	24 Tango Accordion
4 Honky-Tonk	25 Acoustic Guitar(nylon)
5 Electric Piano 1	26 Acoustic Guitar(steel)
6 Electric Piano 2	27 Electric Guitar(jazz)
7 Harpsichord	28 Electric Guitar(clean)
8 Clavinet	29 Electric Guitar(muted)
9 Celesta	30 Overdriven Guitar
10 Glockenspiel	31 Distortion Guitar
11 Music Box	32 Guitar Harmonics
12 Vibraphone	33 Acoustic Bass
13 Marimba	34 Electric Bass (finger)
14 Xylophone	35 Electric Bass (pick)
15 Tubular Bells	36 Fretless Bass
16 Dulcimer	37 Slap Bass 1
17 Drawbar Organ	38 Slap Bass 2
18 Percussive Organ	39 Synth Bass 1
19 Rock Organ	40 Synth Bass 2
20 Church Organ	41 Violin
21 Reed Organ	42 Viola
43 Cello	86 Lead 6 (voice)
44 Contrabass	87 Lead 7 (fifths)
45 Tremolo Strings	88 Lead 8 (bass+lead)
46 Pizzicato Strings	89 Pad 1 (new age)
47 Harp	90 Pad 2 (warm)
48 Timpani	91 Pad 3 (polysynth)
49 String Ensemble 1	92 Pad 4 (choir)
50 String Ensemble 2	93 Pad 5 (bowed)
51 Synth Strings 1	94 Pad 6 (metallic)
52 Synth Strings 2	95 Pad 7 (halo)
53 Choir Aahs	96 Pad 8 (sweep)
54 Voice Oohs	97 FX 1 (rain)
55 Synth Voice	98 FX 2 (soundtrack)
56 Orchestra Hit	99 FX 3 (crystal)
57 Trumpet	100 FX 4 (atmosphere)
58 Trombone	101 FX 5 (brightness)
59 Tuba	102 FX 6 (goblins)
60 Muted Trumpet	103 FX 7 (echoes)
61 French Horn	104 FX 8 (sci-fi)
62 Brass Section	105 Sitar
63 Synth Brass 1	106 Banjo
64 Synth Brass 2	107 Shamisen
65 Soprano Sax	108 Koto
66 Alto Sax	109 Kalimba
67 Tenor Sax	110 Bagpipe
68 Baritone Sax	111 Fiddle
69 Oboe	112 Shanai
70 English Horn	113 Tinkle Bell
71 Bassoon	114 Agogo
72 Clarinet	115 Steel Drums
73 Piccolo	116 Woodblock
74 Flute	117 Taiko Drum
75 Recorder	118 Melodic Tom
76 Pan Flute	119 Synth Drum
77 Blown Bottle	120 Reverse Cymbal
78 Skakuhachi	121 Guitar Fret Noise
79 Whistle	122 Breath Noise
80 Ocarina	123 Seashore
81 Lead 1 (square)	124 Bird Tweet
82 Lead 2 (sawtooth)	125 Telephone Ring
83 Lead 3 (calliope)	126 Helicopter
84 Lead 4 (chiff)	127 Applause
85 Lead 5 (charang)	128 Gunshot

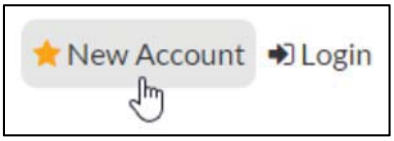
Accented Characters etc.

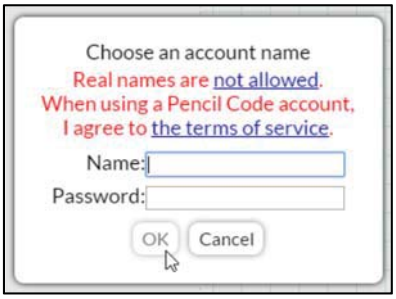
Accent	Examples	Encodings
grave	À à è ò	\`A \`a \`e \`o
acute	Á á é ó	\'A \'a \'e \'o
circumflex	Â â ê ô	\^A \^a \^e \^o
tilde	Ã ã ñ õ	\~A \~a \~n \~o
umlaut	Ä ä ë ö	\"A \"a \"e \"o
cedilla	Ç ç	\,C \,c
ring	Å å	\oA \oa
slash	Ø ø	\/O \/o
breve	Ă ă Ĕ ĕ	\uA \ua \uE \ue
caron	Š š Ž ž	\vS \vs \vZ \vz
double acute	Ő ő Ű ű	\HO \Ho \HU \Hu
ligatures	ß Æ æ œ	\ss \AE \ae \oe
copyright	©	© or \u0022
sharp/nat./flat	♯ ♮ ♭	\u266f \u266e \u266d

GETTING STARTED WITH PENCIL CODE

1. Go to the Pencil Code home page at: **http://pencilcode.net**

2. Click **New Account** to create your own Pencil Code account.

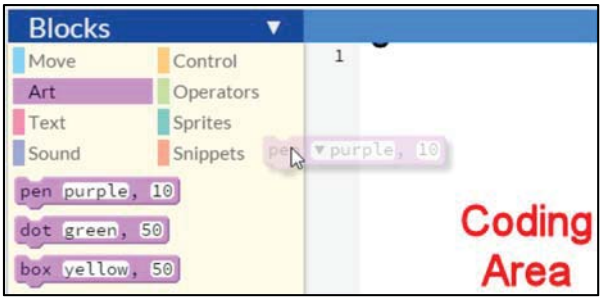


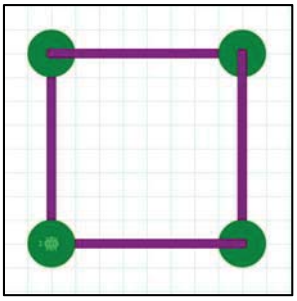


3. Enter an account name for yourself (not your real name) and a password and click the **OK** button.

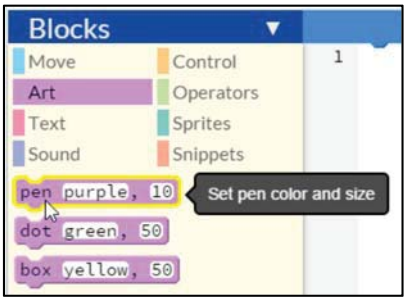
4. Tell one of the teachers or university students your account name and password to add to their list in case you forget it.

5. Drag various blocks into the coding area (as you did with Scratch) to draw a square that looks like the one below.

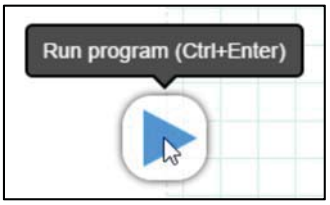




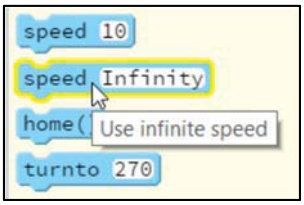
The blocks you will want to use are the **fd** (forward), **rt** (right), and **jumpto** blocks under the **Move** menu, as well as the **pen** and **dot** blocks under the **Art** menu.



6. Once you have sequenced your blocks in the coding area, click the **blue arrow** in the middle of the screen to run your program.

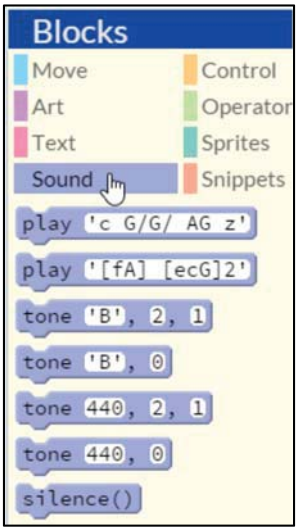


7. You can speed up the animation by adding a **speed** block (under the **Move** menu) at the top of your program.

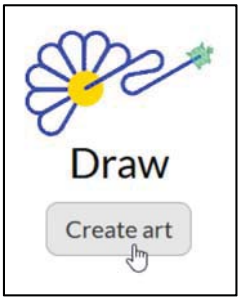


speed 10 will make your program run 10 times faster than normal, and **speed infinity** will make it run as fast as the computer can go.

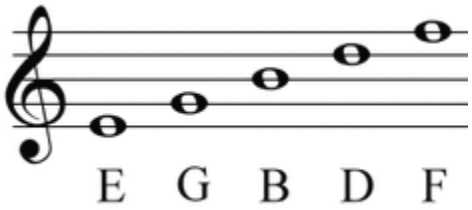
8. As an additional challenge, add blocks from under the **Sound** menu to play notes after each of the corners is drawn.

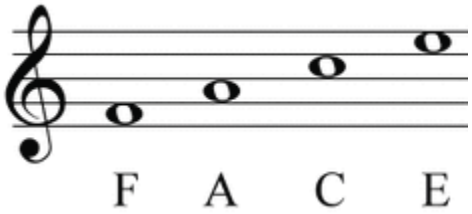


9. Once you do these things, try to make shapes with different numbers of sides or try to "teach the computer to draw" different shapes or figures. See the **Draw** examples by clicking the **Create art** button on the Pencil Code home page for ideas.



NOTES ON THE STAFF





The musical alphabet uses seven letters:

A B C D E F G

After G you start over again at A.

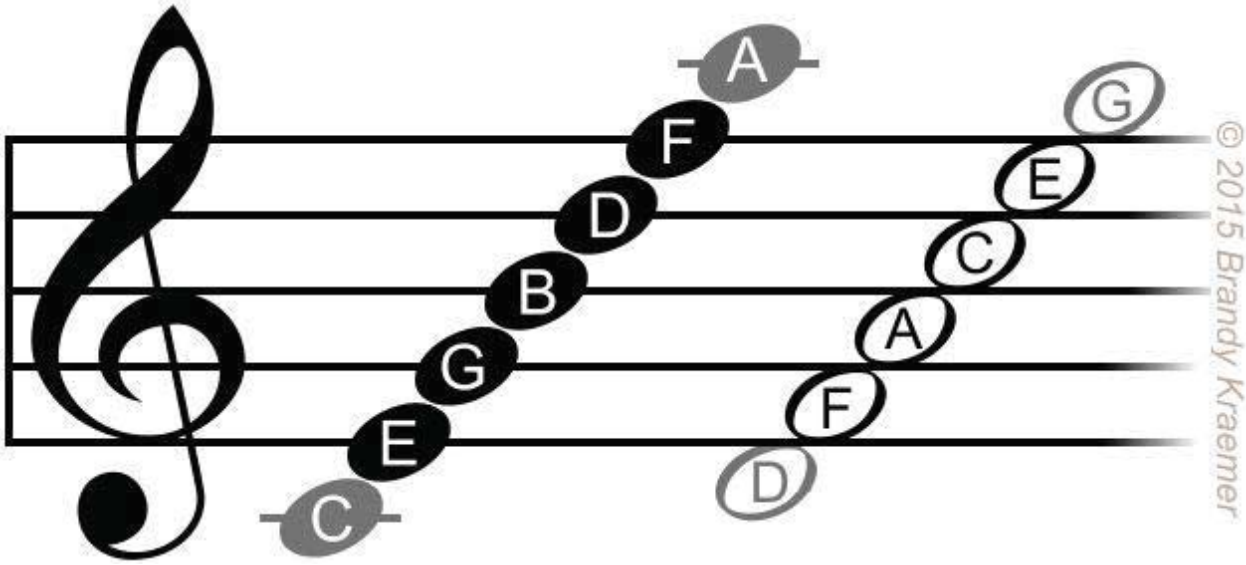
The notes on the staff are on a line or in a space.

Remember the line notes by the saying:

Every **G**ood **B**urger **D**eserves **F**ries

Remember the space notes by spelling out:

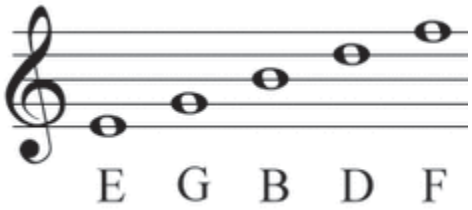
F-A-C-E

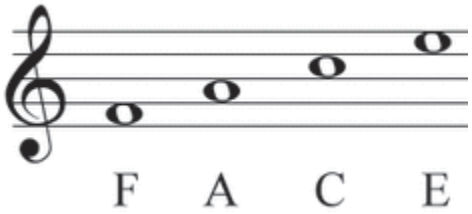


Sources:

- http://musicsmithehb.weebly.com/notes-on-the-staff.html
- http://f.tqn.com/y/piano/1/S/G/f/-/-/Treble-notes_Grand-Staff.png

NOTES AND RESTS





The musical alphabet uses seven letters:

A B C D E F G

After G you start over again at A.

The notes on the staff are on a line or in a space.

Remember the line notes by the saying:

Every **G**ood **B**urger **D**eserves **F**ries

Remember the space notes by spelling out:

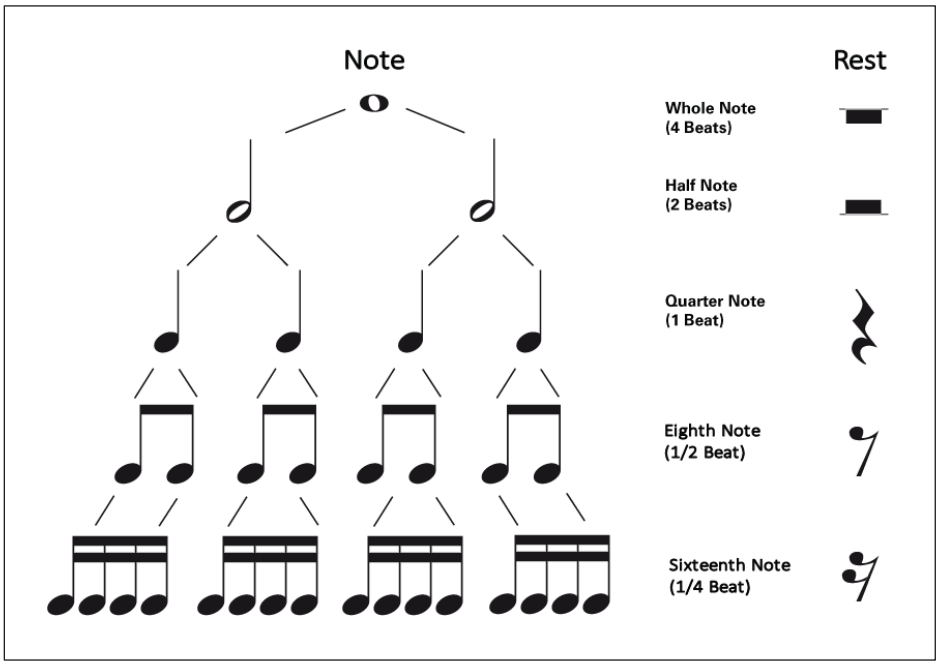
F-A-C-E

Name	Note	Beats	Pencil Code	1 4/4 measure	Rest
Whole	𝅝	4	"G4"	𝅝	"Z4"
Half	𝅗𝅥 𝅗𝅥	2	"G2"	𝅗𝅥 𝅗𝅥	"Z2"
Quarter	♩ ♩	1	"G1" or "G"	♩ ♩ ♩ ♩	"Z1"
Eighth	♪ ♪	½	"G/2"	♪ ♪ ♪ ♪	"Z/2"
Sixteenth	𝅘𝅥𝅯 𝅘𝅥𝅯	¼	"G/4"	𝅘𝅥𝅯 𝅘𝅥𝅯 𝅘𝅥𝅯 𝅘𝅥𝅯	"Z/4"

Sources:

- http://musicsmithehb.weebly.com/notes-on-the-staff.html
- http://musictutorial.in/wp/wp-content/uploads/rest.jpg

Understanding Note and Rest Values, Ties, and Dotted Note Values

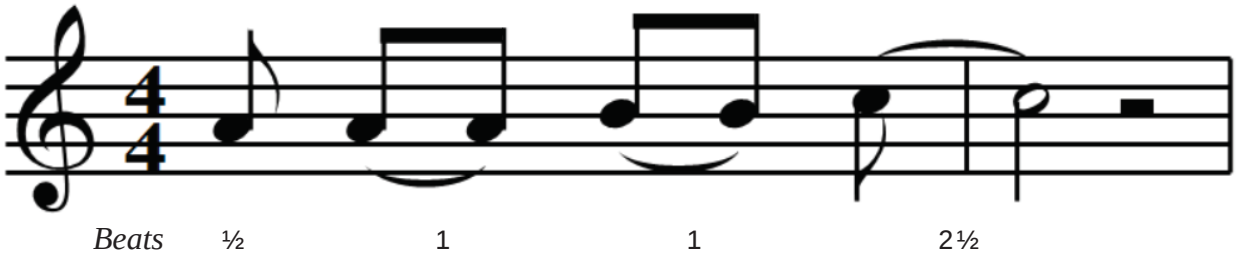


ABC Notation

A4	whole note (4 beats)	z4	whole rest (4 beats)
A2	half note (2 beats)	z2	half rest (2 beats)
A *or* A1	quarter note (1 beat)	z *or* z1	quarter rest (1 beat)
A/ *or* A/2	eighth note (½ beat)	z/ *or* z/2	eighth rest (½ beat)
A/4	sixteenth note (¼ beat)	z/4	sixteenth rest (¼ beat)

Understanding Ties

A ***tie*** indicates that note values are to be added together and held as a single note.

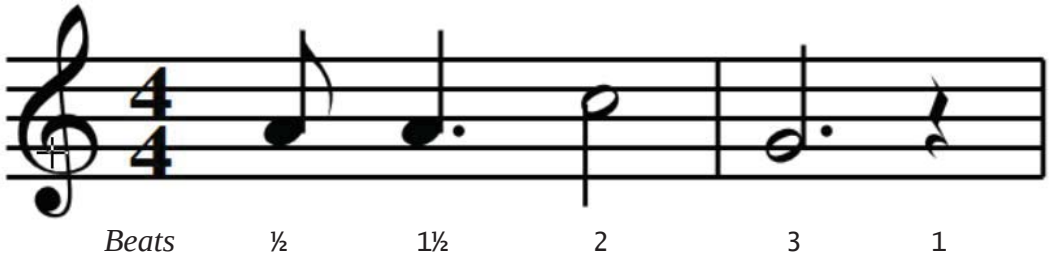


ABC Notation for This Example

```
A/2 (A/2A/2) (B/2B/2) (C/2 | C2) z2 |
```

Understanding Dotted Notes

A ***dot*** after a note indicates that the value of that note is increased by ½. Thus a dotted quarter note has 1½ beats instead of 1, and a dotted quarter note has ¾ beat instead of ½.



ABC Notation for This Example

```
A/2 A3/ C'2 | G3 z |
```

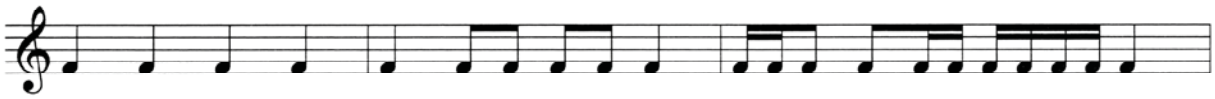
Exercises

Exercise 5-1

Name the following notes and rests.

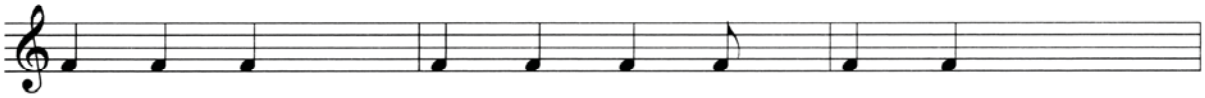
Exercise 5-2

Write the count ("one-e-and-ah") below each of the notes in the following measures.



Exercise 5-3

Fill in the balance of these measures with eighth notes.

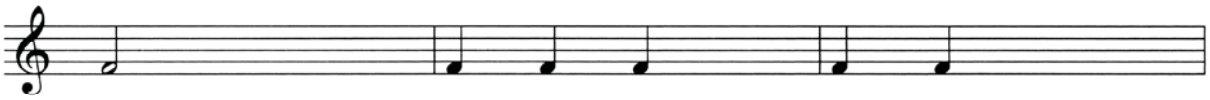


Exercise 5-4

Write the corresponding rests for the following notes.

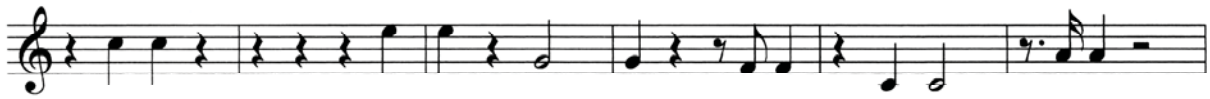
Exercise 5-5

Fill in the balance of these measures with eighth-note triplets.



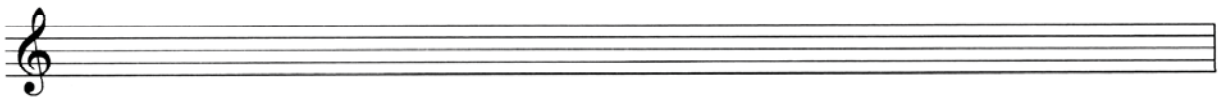
Exercise 5-6

Tie each group of two notes (but not the rests!) together.



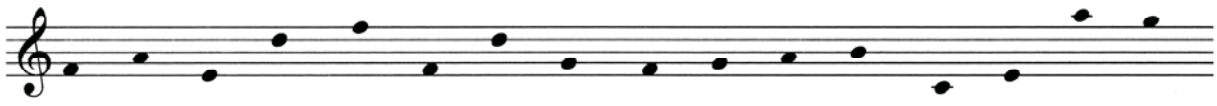
Exercise 5-7

Enter four whole notes, followed by four half notes, followed by four quarter notes, followed by four eighth notes, followed by four sixteenth notes.



Exercise 5-8

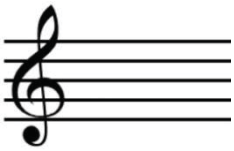
Draw stems and flags on these notes to make them eighth notes; make sure to point the stems in the correct direction.



Understanding Key Signatures

Key signatures determine which sharps and flats are added automatically throughout an entire song. This eliminates the need to put sharp (♯) and flat (♭) symbols before specific notes.

If no key signature is specified, the key signature is C.



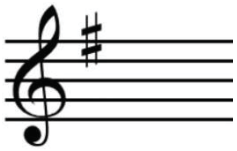
=> Key of C

The sharps go on the staff in the order F – C – G – D – A – E – B. The way to remember this order is to use the sentence "**F**ather **C**rawford **G**oes **D**own **A**nd **E**nds **B**attle."

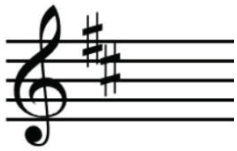
The cool thing about this little memory trick is that the flats go on the staff in the exact opposite order. Therefore, for flats, the sentence is "**B**attle **E**nds **A**nd **D**own **G**oes **C**rawford's **F**ather."

Sharp Keys

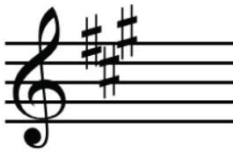
For key signatures containing sharps, the key is always the next higher note from the last sharp. Thus, with 1 sharp on F, the key is F+1 = G. With 2 sharps, F and C, the key is C+1 = D. And so on. Here's the entire list of key signatures containing sharps.



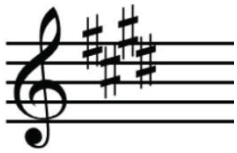
Last sharp on F
=> Key of G



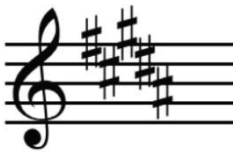
Last sharp on C
=> Key of D



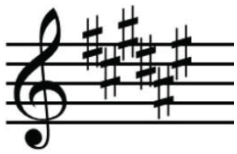
Last sharp on G
=> Key of A



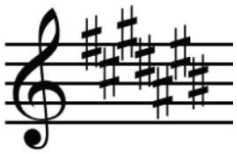
Last sharp on D
=> Key of E



Last sharp on A
=> Key of B



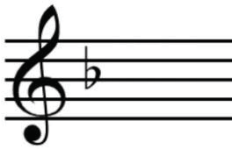
Last sharp on E
=> Key of F#



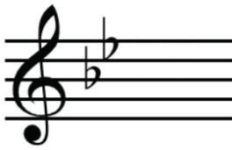
Last sharp on B
=> Key of C#

Flat Keys

Now let's look at key signatures containing flats. This is a little bit different, because in these cases, the key is always three notes down (or four notes) up from the last flat. But there's a trick to this, too: except for 1 flat, in which the key is F, the key is always the next-to-the-last flat. This is easier to see and understand by looking at the staff charts below.



1 flat
=> Key of F



Next-to-last flat on B
=> Key of B♭

Next-to-last flat on E
=> Key of E♭

Next-to-last flat on A
=> Key of A♭

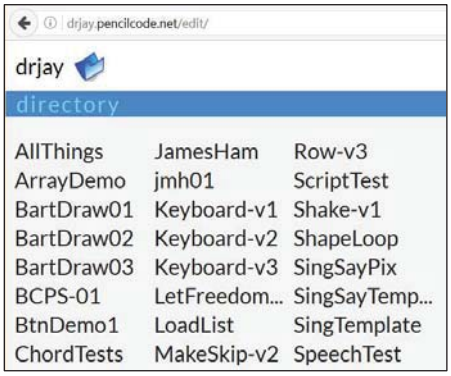
Next-to-last flat on E
=> Key of D♭

Next-to-last flat on G
=> Key of G♭

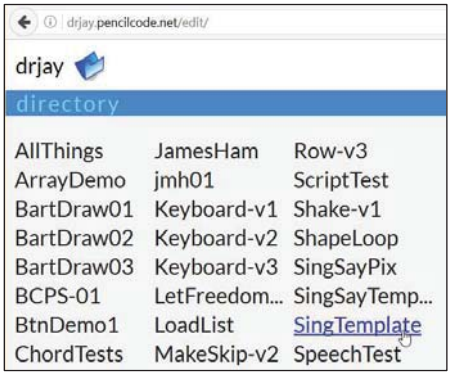
Next-to-last flat on C
=> Key of C♭

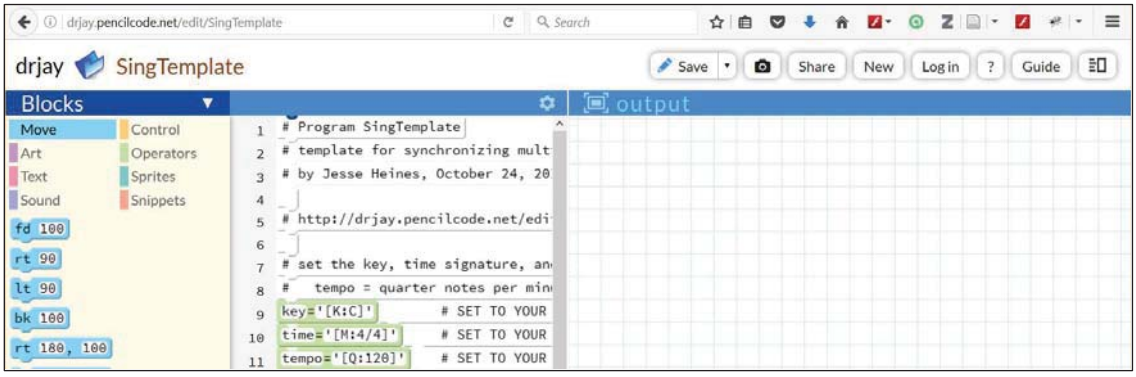
Using the Sing Template

1. Open your browser and go type **`drjay.pencilcode.net`** in the address field. When that page opens you will see a list of programs written by Jesse.

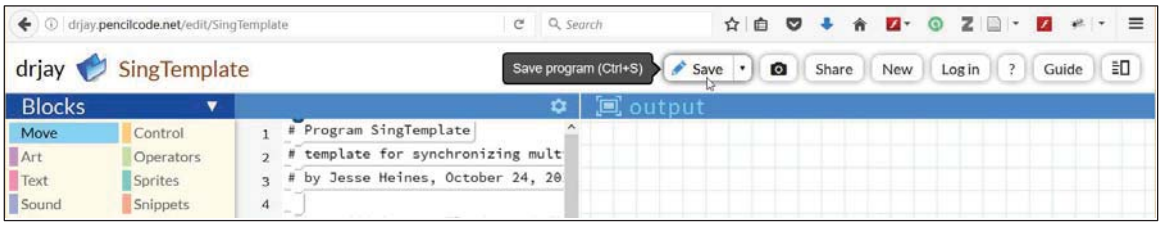


2. Click **`SingTemplate`**. This will open the **`Sing`** template program.





3. With the `SingTemplate` open, click the **`Save`** button.

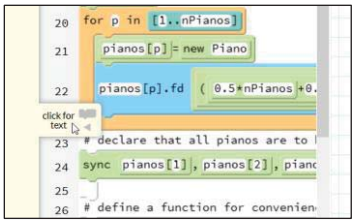


4. A dialog box named **`Log in to save.`** will pop up. Under **`Name: drjay`** click **`Not me? Switch user.`**

5. The dialog box will change to **`Choose an account name to save.`**

6. Enter your own name and password and click the **`OK`** button. This saves the template to your own account.

7. Click the **`click for text`** button. This provides access the full text of the program.



drjay.pencilcode.net/edit/SingTemplate

drjay SingTemplate

{} code

```
1  # Program SingTemplate
2  # template for synchronizing multiple parts
3  # by Jesse Heines, October 24, 2016
4
5  # http://drjay.pencilcode.net/edit/SingTemplate
6
7  # set the key, time signature, and tempo
8  #   tempo = quarter notes per minute
9  key='[K:C]'        # SET TO YOUR KEY
10 time='[M:4/4]'     # SET TO YOUR METER
11 tempo='[Q:120]'    # SET TO YOUR TEMPO
12
13 # combine key, time, and tempo into a variable
14 #   named prefix simply for convenience
15 prefix=time+key+tempo
16
17 # initialize up to 6 pianos to play each part
18 nPianos = 3   # SET THIS TO THE NUMBER OF PIANOS YOU WANT
19 pianos=[]
20 for p in [1..nPianos]
21   pianos[p]=new Piano
22   pianos[p].fd (0.5*nPianos+0.5-p)*150
23 # declare that all pianos are to be synced
24 sync pianos[1],pianos[2],pianos[3],pianos[4],pianos[5]
   ,pianos[6]
25
26 # define a function for convenience, to which
27 # we can pass which piano to play (p) and the
28 # notes to play on that piano
29 sing = (p,notes) ->
30   pianos[p].play prefix+notes
31
32 # REPLACE THE FOLLOWING WITH YOUR OWN CODE
33 # start coding the song here in the format shown
34 sing 1, "C D E F G A B C'"
35 sing 2, "E F G A B C' D' E'"
36 sing 3, "G A B C' D' E' F' G'"
37
```

8. You can now edit the text as follows.

   a. If necessary, replace the C key letter in line 9 with the key letter of your song.

```
 9  key='[K:C]'        # SET TO YOUR KEY
```

   b. If necessary, replace the 4/4 time signature (meter) in line 10 with the time signature of your song.

```
10  time='[M:4/4]'     # SET TO YOUR METER
```

   c. If necessary, replace the 120 beats per minute tempo in line 11 with the tempo of your song.

```
11  tempo='[Q:120]'    # SET TO YOUR TEMPO
```

   d. If necessary, replace 3 in line 18 with the number of pianos (parts) that your song needs.

```
18  nPianos = 3        # SET THIS TO THE NUMBER OF PIANOS YOU WANT
```

   e. Starting at line 34, replace the sample **`sing`** lines with the notes that you want to play on each piano in your program.

```
32  # REPLACE THE FOLLOWING WITH YOUR OWN CODE
33  # start coding the song here in the format shown
34  sing 1, "C D E F G A B C'"
35  sing 2, "E F G A B C' D' E'"
36  sing 3, "G A B C' D' E' F' G'"
```

   You may add as many **`sing`** lines as you like, but they must all follow this pattern:

`sing` ***piano number*****`, "`*****list of notes*****`"`**

   - The first word must be **`sing`**.
     - (This calls the **`sing`** function defined at line 29.)
     - **`sing`** must be followed by a space (press the space bar).
   - Next, type the number of the piano you want to use.
     - Follow that with a comma (**`,`**) and a space.
   - Then type a list of notes inside double quotes (**`"`**).
     - You can put as many notes inside the double quotes as you like, but we recommend that you only put 1 or 2 measures of notes on each line.

   Remember that each piano (part) in your program is *independent*. That is, all the pianos (parts) will play at the same time. If you want one part to start before another, you must add rests to delay the second part.

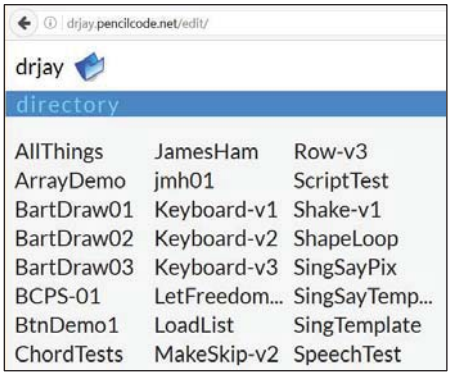
Do not change any other code in the `Sing` template.
Doing so will prevent it from working properly.

Teaching a Computer to Sing

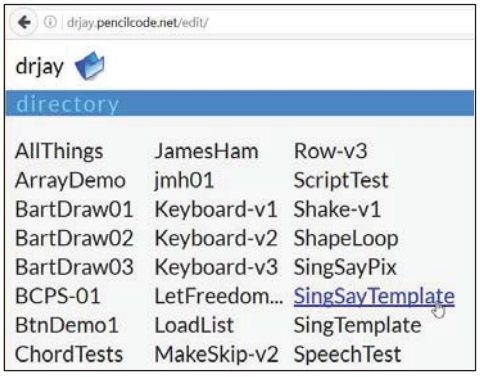
University of Massachusetts Lowell
Bartlett Community Partnership School

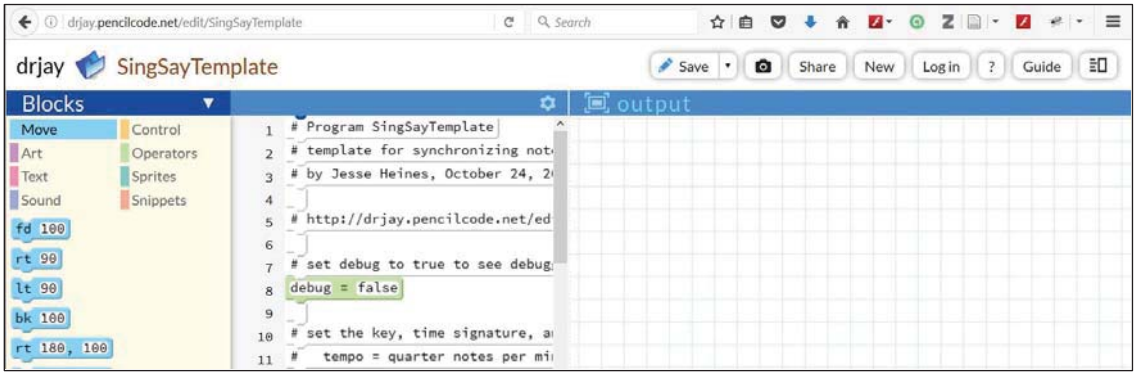
Using the SingSay Template

1. Open your browser and go type **`drjay.pencilcode.net`** in the address field. When that page opens you will see a list of programs written by Jesse.

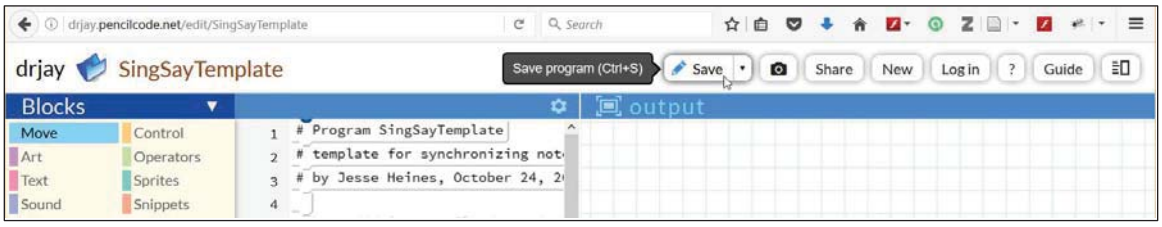


2. Click **`SingSayTemplate`**. This will open the **`SingSay`** template program.





3. With the **SingSayTemplate** open, click the **Save** button.

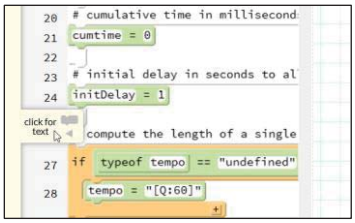


4. A dialog box named **Log in to save.** will pop up. Under **Name: drjay** click **Not me? Switch user.**

5. The dialog box will change to **Choose an account name to save.**

6. Enter your own name and password and click the **OK** button. This saves the template to your own account.

7. Click the **click for text** button. This provides access the full text of the program.



```
# Program SingSayTemplate
# template for synchronizing notes and lyrics
# by Jesse Heines, October 24, 2016

# http://drjay.pencilcode.net/edit/SingSayTemplate

# set debug to true to see debugging information
debug = false

# set the key, time signature, and tempo
#   tempo = quarter notes per minute
key='[K:C]'       # SET TO YOUR KEY
time='[M:4/4]'    # SET TO YOUR METER
tempo='[Q:120]'   # SET TO YOUR TEMPO

# combine key, time, and tempo into a variable
#   named prefix simply for convenience
prefix=time+key+tempo

# cumulative time in milliseconds
cumtime = 0

# initial delay in seconds to allow pianos to display
initDelay = 1

# compute the length of a single beat
if typeof tempo == "undefined"
  tempo = "[Q:60]"
beatlength = /^\[Q:(\d+)\]$/.exec tempo
beatlength = 1000*60/(parseInt beatlength[1])
if debug then write "1 beat = " + beatlength + " ms"

# function to delay for a number of seconds
# call this function  to allow the pianos to be displayed and
#   positioned before starting the music
delay = ( seconds ) ->
  pause seconds
  cumtime += 1000*seconds

# initialize up to 6 pianos to play each part
nPianos = 2   # SET THIS TO THE NUMBER OF PIANOS YOU WANT
pianos=[]
for p in [1..nPianos]
  pianos[p]=new Piano
  pianos[p].fd (0.5*nPianos+0.5-p)*150
# declare that all pianos are to be synced
sync pianos[1],pianos[2],pianos[3],pianos[4],pianos[5],pianos[6]
# wait for the pianos to be displayed and positioned
delay 1

# define a function for convenience, to which
# we can pass which piano to play (p) and the
# notes to play on that piano
sing = (p, notes) ->
  pianos[p].play prefix+notes

# reference for the following code:
# http://stackoverflow.com/questions/6459630/
#   how-to-write-settimeout-with-params-by-coffeescript

```

```
# auxiliary function called by setTimeout
# nPiano = number of piano on which to play notes
# notes = the notes to be played for this phrase
# lyrics = the words to print for this phrase
# note: to start a new line of lyrics, start the line with |
singsayaux = ( nPiano, note, lyrics ) ->
  sing nPiano, note
  if lyrics.substr(0,1) == "|"
    write ""
    append lyrics.substr(1)
  else
    append lyrics

# function to both sing and display lyrics
# seconds = number of seconds to delay since playing the last phrase
# nPiano = number of piano on which to display notes
# notes = the notes to be played for this phrase
# lyrics = the words to print for this phrase
# note: to start a new line of lyrics, start the line with |
singsay = ( nPiano, note, lyrics ) ->
  if true
    callback = -> singsayaux nPiano, note, lyrics
    setTimeout callback, cumtime
  durations = /^[\^_=]*[A-Ga-gZz][',]*((\d*)(\/?)(\d*))$/.exec note
  if debug
    append "|" + durations[2] + "|   |" + durations[3]
    append "|   |" + durations[4] + "|"
  if debug then append note
  if durations == null
    write "Error: duration for note \"" + note + "\" is null"
  else
    noteduration = durations[1]
    if noteduration == ""
      milliseconds = beatlength
      if debug then append "  = " + beatlength + " ms"
    else if noteduration == "/"
      milliseconds = beatlength/2
      if debug then append "  = " + beatlength/2 + " ms"
    else
      durations[2] = 1 if durations[2] == ""
      durations[4] = 1 if durations[4] == ""
      if debug
        append "  =>   |" + durations[2]
        append "|   |" + durations[4] + "|"
      milliseconds = beatlength * parseInt(durations[2]) / parseInt(durations[4])
      if debug then append "  = " + milliseconds + " ms"
  cumtime += milliseconds
  if debug then append " => " + cumtime
  if debug then write ""

# coded notes and lyrics - REPLACE WITH YOUR OWN CODE
singsay 1, "C", "do(low)"
singsay 1, "D", " re"
singsay 1, "E2", " mi"
singsay 1, "F", " fa"
singsay 1, "G", " sol"
singsay 1, "A2", " la"
singsay 1, "B", " ti"
singsay 1, "C'", " do(high)"
```

8. You can now edit the text as follows.
   a. If necessary, replace the C key letter in line 12 with the key letter of your song.

```
12  key='[K:C]'       # SET TO YOUR KEY
```

   b. If necessary, replace the 4/4 time signature (meter) in line 13 with the time signature of your song.

```
13  time='[M:4/4]'    # SET TO YOUR METER
```

   c. If necessary, replace the 120 beats per minute tempo in line 14 with the tempo of your song.

```
14  tempo='[Q:120]'   # SET TO YOUR TEMPO
```

   d. If necessary, replace 3 in line 18 with the number of pianos (parts) that your song needs.

```
41  nPianos = 1       # SET THIS TO THE NUMBER OF PIANOS YOU WANT
```

   e. Starting at line 112, replace the sample **`singsay`** lines with the notes and lyrics that you want to play on each piano in your program.

```
111  # coded notes and lyrics - REPLACE WITH YOUR OWN CODE
112  singsay 1, "C", "do(low)"
113  singsay 1, "D", " re"
114  singsay 1, "E2", " mi"
115  singsay 1, "F", " fa"
116  singsay 1, "G", " sol"
117  singsay 1, "A2", " la"
118  singsay 1, "B", " ti"
119  singsay 1, "C'", " do(high)"
```

   You may add as many **`singsay`** lines as you like, but they must all follow this pattern:

`singsay` ***piano number*****`, "`*****note letter*****`", "`*****lyric*****`"**

   - The first word must be **`singsay`**.
     - (This calls the **`singsay`** function defined at line 80.)
     - **`singsay`** must be followed by a space (press the space bar).
   - Next, type the number of the piano you want to use.
     - Follow that with a comma (`,`) and a space.
   - Next, type the ***single note*** to be played, inside double quotes (").
     - ***Important:*** You may only put ***one note*** inside the double quotes. The note may include the number of beats it is to play (such as `A2` or `B/4`), but only one note is allowed.
     - Again, follow that with a comma (`,`) and a space.
   - Finally, type the word (lyric) to display when the note is played, again inside double quotes (").

   There are two tricks available to help make your lyrics look right on the screen.

   - To separate words, add a space either before or after each word to be displayed.

- To force a line break, that is, to get a word to start on a new line, put a vertical bar (`|`) before the word, like this: `"|word"`.

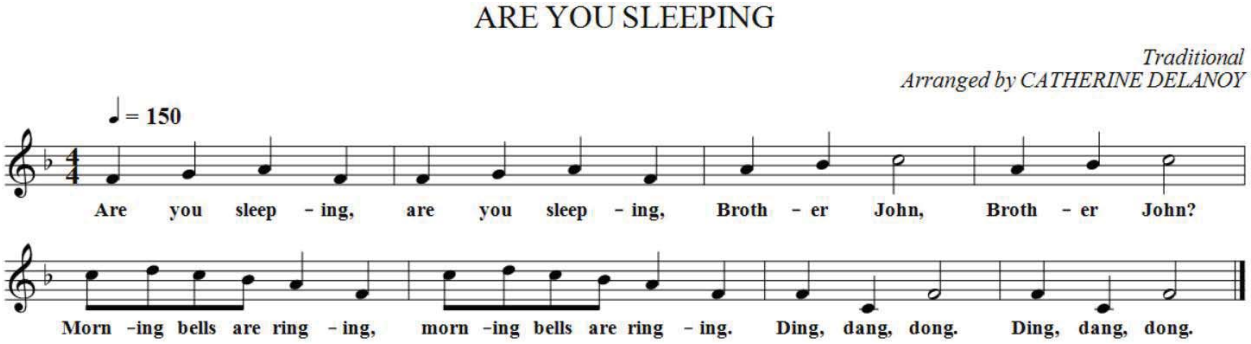
Remember that each piano (part) in your program is *independent*. That is, all the pianos (parts) will play at the same time. If you want one part to start before another, you must add rests to delay the second part.

Do not change any other code in the `SingSay` template.
Doing so will prevent it from working properly.

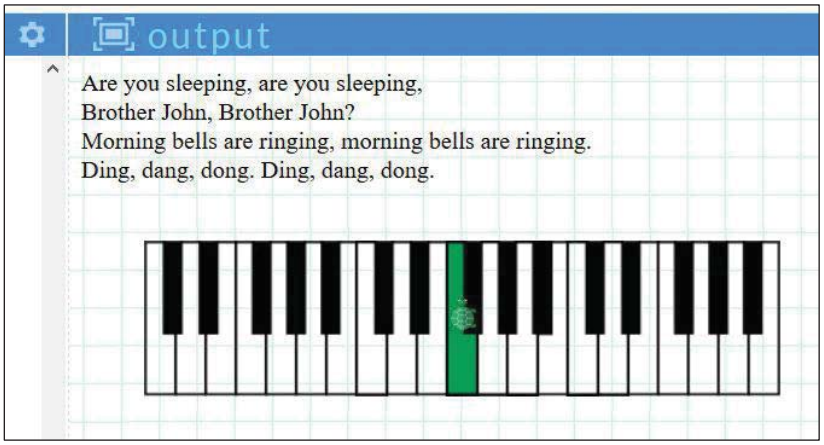
Using the SingSay Template — Example

ABC code

```
%abc-2.1

X:1
T:ARE YOU SLEEPING
C:Traditional
C:Arranged by CATHERINE DELANOY
M:4/4
L:1/4
K:F
Q:150
F G A F | F G A F | \
w:Are you sleep-ing, are you sleep-ing,
A B C'2 | A B C'2 |
w:Broth-er John, Broth-er John?
C'/2D'/2C'/2B/2 A F | C'/2D'/2C'/2B/2 A F | \
w:Morn-ing bells are ring-ing, morn-ing bells are ring-ing.
F C F2 | F C F2 |]
w:Ding, dang, dong. Ding, dang, dong.
```



```
111  # "Are you sleeping" lyrics
112  for k in [1..2]
113    if k == 1
114      singsay 1, "F", "Are"
115    else
116      singsay 1, "F", "are"
117    singsay 1, "G", " you"
118    singsay 1, "A", " sleep"
119    singsay 1, "F", "ing, "
120
121  # "Brother John" lyrics
122  for k in [1..2]
123    if k == 1
124      singsay 1, "A", "|Broth"
125    else
126      singsay 1, "A", "Broth"
127    singsay 1, "B", "er"
128    if k == 1
129      singsay 1, "C'2", " John, "
130    else
131      singsay 1, "C'2", " John?"
132
133  # "Morning bells are ringing" lyrics
134  for k in [1..2]
135    if k == 1
136      singsay 1, "C'/2", "|Morn"
137    else
138      singsay 1, "C'/2", " morn"
139    singsay 1, "D'/2", "ing"
140    singsay 1, "C'/2", " bells"
141    singsay 1, "B/2", " are"
142    singsay 1, "A", " ring"
143    if k == 1
144      singsay 1, "F", "ing,"
145    else
146      singsay 1, "F", "ing."
147
148  # "Ding, dang, dong" lyrics
149  for k in [1..2]
150    if k == 1
151      singsay 1, "F", "|Ding, "
152    else
153      singsay 1, "F", " Ding, "
154    singsay 1, "C", "dang, "
155    singsay 1, "F2", "dong."
```

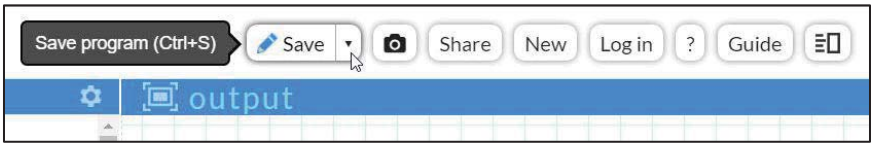


Using TACTS Pencil Code Functions

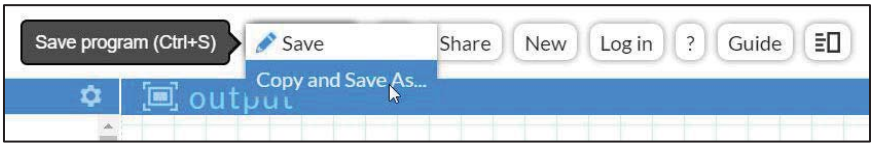
Version 2.1, updated March 5, 2017

Mr. Jesse has simplified the way to use his **sing** and **singsay** functions and added new functions to display and animate your own pictures. All of these functions can now be loaded with a single statement in your Pencil Code programs. This document explains how to do that and the *parameters* (additional information) you need to pass to the functions to make them work.

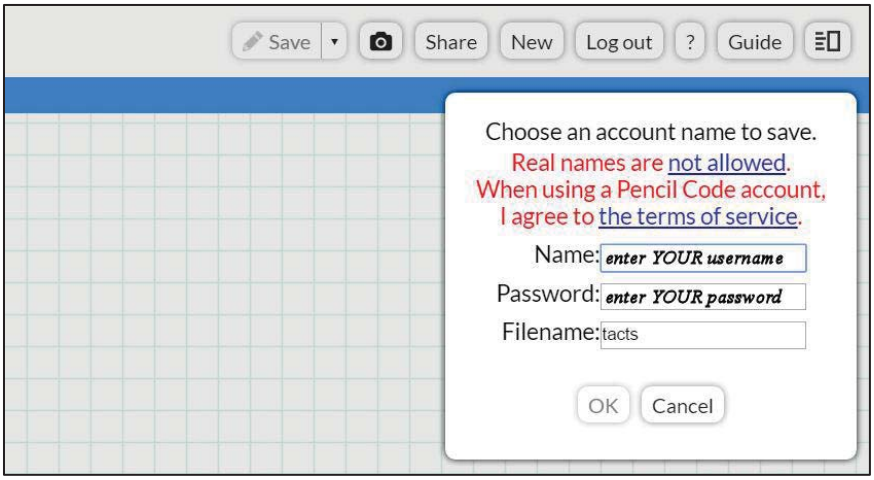
1. Open your browser and type **drjay.pencilcode.net/edit/tacts** in the address field.
2. After that page opens, click the down arrowhead (▼) just to the right of the **Save** button.



3. This will open a dropdown menu. Click **Copy and Save As...**

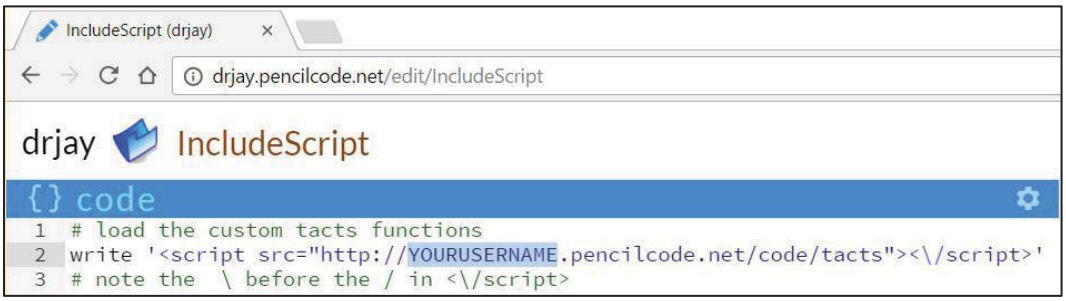


4. This will open a dialog box. Enter ***your*** user **Name** and **Password** and click the **OK** button.



5. This will save the **tacts** functions to your own account.

6. Now type **`drjay.pencilcode.net/edit/IncludeScript`** in the address field of your browser window. After that page opens you will see the following 3-line program.



7. Copy-and-paste these 3 lines (or at least line 2) into your own program. (The critical code is line 2. Lines 1 and 3 are comments.)
8. Replace **YOURUSERNAME** with your own user name.
9. You can now use the **sing** and **singsay** functions as before, with one small change: all functions need to be preceded by **tacts.** (including the dot). Thus, to use the **sing** function, you now type:

```
tacts.sing 1, "C D E F G"
```

   And to use the **singsay** function you now type:

```
tacts.singsay 1, "C", "do"
```

Important Note: The **tacts** functions cannot be mixed with the standard Pencil Code functions. If you choose to use the **tacts** functions in your program, you must do everything with these functions.

Summary of Function Headers

*Note that parentheses are required after all the **Get** functions.*

tacts.CreatePianos *nPianos*, *vertical_separation*
nPianos = tacts.GetNPianos()
tacts.MovePiano *nPiano*, *dx*, *dy*
tacts.SetKey "*key*"
current_key = tacts.GetKey()
tacts.SetTempo *tempo*
current_tempo = tacts.GetTempo()
tacts.SetTime "*timesig*"
current_time_signature = tacts.GetTime()
tacts.sing *nPiano*, "*series_of_notes*"
tacts.singsay *nPiano*, "*note*", "*lyric*"
picture_handle = tacts.ShowPicture "*url*", *width*, *height*, *x*, *y*
tacts.SizePicture *picture_handle*, *multiplier*

Index to Functions

tacts.CreatePianos *nPianos*, *vertical_separation*

Purpose This function creates 1 or more pianos for playing notes. It must be called before calling the **sing** or **singsay** or **MovePiano** functions.

Parameters ***nPianos*** = the number of pianos that you wish to create. If nPianos is not supplied, it will be set to 1.
vertical_separation = the distance (in pixels) that each piano will be separated from each other in the vertical direction. If `vertical_separation` is not supplied, it will be set to 150 (pixels).

Example

```
tacts.CreatePianos 3, 125
```

This function call creates 3 pianos and separates them vertically by 125 pixels.

nPianos = tacts.GetNPianos()

Purpose This function returns the number of pianos that the system knows about.

Parameters This function takes no parameters. Therefore, the name of the function ***must*** be followed by parentheses.

Return Value This function returns the number of active pianos.

Examples

```
write "There are currently " + tacts.GetNPianos() + " pianos."
```

tacts.MovePiano *nPiano*, *dx*, *dy*

Purpose This function moves a piano ***dx*** pixels to the right and ***dy*** pixels down. If ***dx*** or ***dy*** is negative, the piano is moved to the left or up, respectively.

Parameters ***nPiano*** = number of piano to move. This parameter is required and must be less than or equal to the number of pianos created.
dx = number of pixels to move the piano to the right. If ***dx*** is negative, the piano moves to the left. If the ***dx*** is not supplied, it is set to 0, which indicates no move in the horizontal direction.
dy = number of pixels to move the piano down. If ***dx*** is negative, the piano moves up. If the ***dy*** parameter is not supplied, it is set to 0, which indicates no move in the vertical direction.

Example
```
tacts.MovePiano 2, 100, -50
```
This function call moves piano #2 100 pixels to the right and 50 pixels up.

Conditions The number of the piano in the first parameter must be less than or equal to the number of pianos you created with the **tacts.CreatePianos** function.

tacts.SetKey "*key*"

Purpose This function sets the key (in ABC notation format) of the music to be played. If you do not call this function, the key will be set to "C".

Parameters ***key*** = an ABC notation key specification, enclosed in double quotes. This parameter is required.

Examples
```
tacts.SetKey "F"     # key of F (1 flat)
tacts.SetKey "Bb"    # key of B-flat (2 flats)
tacts.SetKey "F#"    # key of F-sharp (6 sharps)
```

current_key = tacts.GetKey()

Purpose This function returns the current key in which music will be played.

Parameters This function takes no parameters. Therefore, the name of the function ***must*** be followed by parentheses.

Return Value This function returns the current key in which music will be played.

Examples
```
write "The current key is " + tacts.GetKey()
```

tacts.SetTempo *tempo*

Purpose This function sets the tempo (number of beats per minute) of the music to be played. If you do not call this function, the tempo will be set to 120 beats per minute.

Parameters ***tempo*** = the number of beats per minute. This parameter is required.

Examples
```
tacts.SetTempo 60    # play at half normal speed
tacts.SetTempo 240   # play at double normal speed
```

current_tempo = tacts.GetTempo()

Purpose This function returns the current tempo with which music will be played.

Parameters This function takes no parameters. Therefore, the name of the function ***must*** be followed by parentheses.

Return Value This function returns the current tempo (in beats per minute) with which music will be played.

Examples
```
write "The current tempo is " + tacts.GetTempo() +
   " beats per minute."
```

tacts.SetTime "*timesig*"

Purpose This function sets the time signature (in ABC notation format) of the music to be played. If you do not call this function, the time signature will be set to "4/4".

Parameters ***timesig*** = an ABC notation time signature specification, enclosed in double quotes. This parameter is required.

Examples
```
tacts.SetTime "3/4"  # 3 beats per measure, quarter note = 1 beat
tacts.SetTime "6/8"  # 6 beats per measure, eighth note = 1 beat
```

current_time_signature = tacts.GetTime()

Purpose This function returns the current time signature with which music will be played.

Parameters This function takes no parameters. Therefore, the name of the function ***must*** be followed by parentheses.

Return Value This function returns the current time signature with which music will be played.

Examples
```
write "The current time signature is " + tacts.GetTime()
```

tacts.sing *nPiano*, "*notes*"

Purpose This function plays a series of notes (in ABC notation) on a specific piano.

Parameters ***nPiano*** = number of piano on which to play the notes. This parameter is required and must be less than or equal to the number of pianos created.
notes = series of notes to play, enclosed in double quotes. This parameter is required.

Examples

```
tacts.sing 2, "Z/2 E/2 E/2E/2 E/2F/2G/2B/2"
```

This function call plays the first singing line of Bruno Mars's song *Lighters*, "This one's for you__ and me,__."

```
forward = "C D E F G A B C'"
reverse = "C' B A G F E D C"
tacts.sing 1, forward
tacts.sing 2, reverse
```

These variable initializations and function calls demonstrate setting variables to series of notes and then playing those series simultaneously on two different pianos.

Conditions You must create at least one piano with the **tacts.CreatePianos** function (explained above) before calling this function.
The number of the piano in the first parameter must be less than or equal to the number of pianos you created with the **tacts.CreatePianos** function.

tacts.singsay *nPiano*, "*note*", "*lyric*"

Purpose This function plays a ***single*** note (in ABC notation) on a specific piano and prints the lyric at the same time.

Parameters ***nPiano*** = number of piano on which to play the note. This parameter is required.
note = the note to play, enclosed in double quotes. This parameter is required.
lyric = the lyric to display, enclosed in double quotes. This parameter is required.

Examples

```
tacts.singsay 1, "Z/2", ""
tacts.singsay 1, "E/2", "This"
tacts.singsay 1, "E/2", " one's"
tacts.singsay 1, "E/2", " for"
tacts.singsay 1, "E/2", " you"
tacts.singsay 1, "F/2", "___"
tacts.singsay 1, "G/2", " and"
tacts.singsay 1, "B/2", " me"
```

These function calls play and show the lyrics for the first singing line of Bruno Mars's song *Lighters*, "This one's for you__ and me."

Note that there can be a blank lyric for a rest, but that must be specified as "". It cannot simply be omitted.

The blank spaces before some words avoid the problem of words running into each other. That is, the above code prints “This one’s for you___ and me” rather than “Thisone’sforyou___andme.”

If you want a word to start on a new line, precede it with a vertical bar (|), as in "|and". This will make the word “and” appear on the next line rather than next to the word that precedes it.

Conditions

You must create at least one piano with the **tacts.CreatePianos** function (explained above) before calling this function.

The number of the piano in the first parameter must be less than or equal to the number of pianos you created with the **tacts.CreatePianos** function.

This function only works with one piano at a time. That is, unlike the **sing** function, it will only work with one part.

picture_handle = tacts.ShowPicture "*url*", *width*, *height*, *x*, *y*

Purpose

This function displays a picture stored on a web server other than the Pencil Code web server. It can be used to display your own pictures. To do that, please see Mr. Jesse about how to get your pictures online.

Parameters

url = the URL or the picture to display, enclosed in double quotes. This parameter is required.

width = the width in pixels at which the picture should be displayed. This parameter is required, but if it is set to 0, the width will be proportional to the height as in the original picture.

height = the height in pixels at which the picture should be displayed. This parameter is required, but if it is set to 0, the height will be proportional to the width as in the original picture.

x = the ***x*** (horizontal) coordinate at which the upper left-hand corner of the picture should appear. This parameter is required.

y = the ***y*** (vertital) coordinate at which the upper left-hand corner of the picture should appear. This parameter is required.

Return Value

This function returns a ***picture_handle*** that can be used to move and resize the picture. Moving a picture can be accomplished using the standard Pencil Code functions in the **Move** category, while resizing can be accomplished using the **tacts.SizePicture** function.

Examples

```
tacts.ShowPicture "tactspictures/Hessell_Taliyah_1242x1242.jpg",
   0, 200, 400, 0
```

This first example displays a picture with a height of 200 pixels and width proportional to the original, at a horizontal location of 400 pixels and a vertical location of 0 pixels (at the top of the display area).

```
pict2 = tacts.ShowPicture "tactspictures/Taliyah_Chan.jpg", 0,
   200, 0, 150
```

This second example displays a picture with a height of 200 pixels and width proportional to the original, at a horizontal location of 0 pixels (at the left of the display area) and a vertical location of 150 pixels.

The advantage of this second example is that it saves a "handle" to the picture so that it can be moved with commands such as the following.

```
for k in [1..4]
  pict2.slide 200
  pict2.rt 90
```

Both of these examples assume that the picture to be displayed is located on the http://tacts.000webhostapp.com/ server. If the picture is located on another server, use the full URL as follows.

```
pict3 = tacts.ShowPicture "http://multifiles.pressherald.com/
   uploads/sites/4/2017/02/1148251_Exchange_Teaching_a_Compute.jpg",
   500, 0, 0, 0
```

If pictures overlap, the pictures loaded last will overlap those loaded earlier.

tacts.SizePicture *picture_handle, multiplier*

Purpose This function changes to size of a picture.

Parameters ***picture_handle*** = the picture handle returned by a call to the **ShowPicture** function. This parameter is required.
multiplier = the proportion by which the picture's size is to be changed. This parameter is required. If the multiplier is greater than 1, the picture size will be increased. If it is less than 1, the picture size will be decreased.

Examples

```
keydown 'G', ->
   tacts.SizePicture pict1, 1.1
keydown 'H', ->
   tacts.SizePicture pict2, 1.1
keydown 'S', ->
   tacts.SizePicture pict1, 0.9
keydown 'D', ->
   tacts.SizePicture pict2, 0.9
```

These function calls set up actions when keys are pressed.

- pressing the G key increases the size of picture #1 by 10% (times 1.1)
- pressing the H key increases the size of picture #2 by 10%
- pressing the S key decreases the size of picture #1 by 10% (times 0.9)
- pressing the D key decreases the size of picture #2 by 10%

By Firas AL-Rekabi, Bartlett Community Partnership School Math Teacher, January 2017

Coding Basics (Variables)

Code	Output
X=10 Y=4 Z=3.5 W=”Hello”	

Coding Basics (Strings)

Code	Output
X=”Hello” Y=’hi’ Z=’A’	

Coding Basics (Comments)

Code	Output
#This program was developed by … #This code is used to ... #I have something missing here	

Code	Output
Write “A” #Write “B” #Write “C” Write “D”	Will print: A D

Coding Basics (write)

Code	Output
x=3 write x write 'x=' write 'x=' + x	Will print: 3 x= x=3

Coding Basics (If statement)

Code	Output

x=10
if x>5
  write "Yes greater"
else
  write "No is not greater"
	Will print **Yes**.

Code	Output

x=3
if x>5
  write "Yes"
else
  write "No"
	Will print **No**.

Code	Output
x=7 if x>5 write "Yes greater"	Will print **Yes greater**.

Code	Output
x=2 if x>5 write "Yes greater"	Will print nothing of the screen.

Code		Code		Code		Code	

x=25
if x>20
  write 'A'
else if x<5
  write 'B'
else if x>11
  write 'C'
else
  write 'D'
	A	x=4
		if x>20
		  write 'A'
		else if x<5
		  write 'B'
		else if x>11
		  write 'C'
		else
		  write 'D'
			B	x=15
				if x>20
				  write 'A'
				else if x<5
				  write 'B'
				else if x>11
				  write 'C'
				else
				  write 'D'
					C	x=11
						if x>20
						  write 'A'
						else if x<5
						  write 'B'
						else if x>11
						  write 'C'
						else
						  write 'D'
							D

Coding Basics (Loops)

Code	Output
for x in [1..10] write “Hi”	Will print **Hi** ten times.

Code	Output
for x in [1..10] write x	Will print the numbers from 1 to 10.

Code	Output
for x in [2..10] by 2 write x	Will print the odd numbers 2 4 6 8 10.

Code	Output
s=0 for x in [1..10] s = s + x write s	Will print the sum from 1 to 10 which is 55.

Code	Output

s=0
for x in [1..10]
  if x>8
  s=s+x
write 's=' + s
	19

Code	Output

s=0
for x in [1..10]
  if (x/2)==round(x/2)
  s=s+x
write 's = ' + s
	30

Teaching a Computer to Sing:
A Middle School, After-School Pilot Program Integrating Computing and Music

Jesse M. Heines
Dept. of Computer Science
Univ. of Massachusetts Lowell
1-978-251-9350
Jesse_Heines@uml.edu

Daniel A. Walzer
Dept. of Music
Univ. of Massachusetts Lowell
1-978-934-3881
Daniel_Walzer@uml.edu

Rachel R. M. Crawford
Bartlett Comm. Partnership School
Lowell (MA) Public Schools
1-978-937-8968
RCrawford@lowell.k12.ma.us

Jill H. Lohmeier
Graduate School of Education
Univ. of Massachusetts Lowell
1-978-934-4617
Jill_Lohmeier@uml.edu

Shanna R. Thompson
Graduate School of Education
Univ. of Massachusetts Lowell
1-978-934-4641
Shanna_Thompson@uml.edu

ABSTRACT
Music is universal. Music is also mathematical, and mathematics is of course the basis of computing. This paper reports on the first year of our attempt to create a voluntary after-school program that introduces middle school students to computing through music. As expected, we had successes and failures. This paper details our goals, activities, changes made in mid-year, evaluation results, and plans for the future.

CCS Concepts
• Social and professional topics → K-12 education.

Keywords
Computing and Music; Middle School; After-School.

1. FROM SINGING TO PROGRAMMING
It probably goes without saying that most educators, regardless of field, think it's important for kids to be computer literate and even to learn at least a little about how to program. Of course we agree. But how do we "hook 'em," especially in an after-school program that doesn't even begin until after they've been in school for seven hours and are ready to just play and hang out with their friends?

"There's nothing like making music and messing with sound to inspire people to learn how to program." [25]

In a nutshell, our approach in this project has been to work with a dynamic music teacher whom the children adore, have her teach them songs with multiple parts and modest complexity, and then have the children manipulate recordings of those songs in Audacity [1] and program them in Scratch [20] and Pencil Code [16]. They then "perform" their creations for each other using a computer projector and the room's sound system, thereby enjoying an outlet for their work and learning from what others have produced.

ACM SIGCSE 2017, March 8-11, 2017, Seattle, WA, USA.

DOI: http://dx.doi.org/10.1145/12345.67890

While this approach may sound straightforward, it wasn't easy to implement. Despite their love of listening to—and singing along with—popular music with their friends, the kids in our program were initially shy about singing in a class setting. The music teacher worked hard to get them to overcome their reluctance. The team struggled to engage them in the computer part of the program, too. Due to our lack of experience with this age group, it took some time for us to "get our footing." However, as discussed in this paper, persistence paid off and the children made significant progress in both singing and programming.

2. ENABLING RESEARCH
We believe that our current research is unique in that it explores the intersection of computing and music at the middle school level. The basis for this research is our prior work in Performamatics [14, 15]. The first of our prior projects explored interdisciplinary connections between computer science and three fine arts fields: art, music, and theater [5, 7, 8, 9, 13, 18]. It then focused on computing+music [8, 19], which for us turned out to be the collaboration that had the most "traction." The popularity of the university courses we built in those projects and the positive research results we observed enticed us to see if we could achieve similar gains with younger students. Other research related to ours is discussed below.

Teaching a Computer to Sing builds on the premise outlined by Magerko et al. [12] that creative coding can enhance musicianship by helping students identify a song's structure. It also builds on the concept of "coding through play" described by Stinson [21], who

taught Scratch coding by having children follow robots through a series of imaginative storytelling activities. The key here is that children saw immediate, tangible results as the robots responded to various commands. Programming music also yielded immediate results, which children learned to identify as "right" or "wrong" by what they heard.

Ho [10] conducted a five-year study to evaluate the ability of information technology to inspire learner-centered music creation practices in young students. Both students and teachers remarked that the use of MIDI improved pitch and rhythmic accuracy when added to traditional choral rehearsal techniques and pedagogy. Our work builds on this finding by having students work with the same music in multiple modes.

Studer [22] explored the use of computers in choral rehearsals and noted that music notation programs are highly useful for isolating each part. We took an analogous approach: having children write programs that "sang" multiple parts (see Section 3.2) to help them learn each one. Studer also found that music notation programs enhance STEM learning along with musical competency. Our project went one step further: writing computer programs to play music using standard computing constructs (see Section 3.1).

3. PROGRAM LOGISTICS

Teaching a Computer to Sing is an after-school program. It ran on Tuesdays and Thursdays from October 13, 2015, through May 26, 2016, on the following schedule:

2:10-2:25 snacks
2:25-3:20 singing with the school music teacher
3:20-3:50 break for homework and snacks
3:50-4:45 computing with university faculty and student assistants
4:45-5:00 dinner.

Students spent the first afternoon session singing songs and then, after a break, spent the second coding those songs using ABC Notation [2, 26] in Pencil Code [16], a block programming environment. Sometimes we used the last 10-15 minutes of the computing session for children to show their work to the entire group. This yielded mixed results, as some were enthusiastic to show their creations while others were too self-conscious to do so or were simply ready to go home. Also, we often felt it was best not to disturb them if they were fully engaged in the day's activities.

3.1 Software Choices

We began the program using Scratch as our music platform, but the middle school students had a lot of trouble working with MIDI. First, converting musical scores to MIDI involved two steps: (a) figuring out a note's alphabetic name (such as C) from the staff, and then (b) converting that note to its MIDI value (such as 60). Second, the resulting code bore little resemblance to the musical notation. For example, it's pretty hard to know that the code in Figure 1(a) plays *Frère Jacques* even if one compares it to the sheet music in Figure 1(b). The Pencil Code version in Figure 1(c) that uses ABC notation still requires a bit of interpretation, but it is clearly easier to explain to students than the Scratch version.

To help students make the transition from sheet music to ABC notation we used WebMusicScore [6]. This is a wonderful free website that allows one to enter ABC notation in one window and see the resultant score in another window. Students can thus compare the resultant score with the original score that they have on paper to ensure that they are the same. If the two scores are not identical, the ABC notation is easily edited to correct any problems.

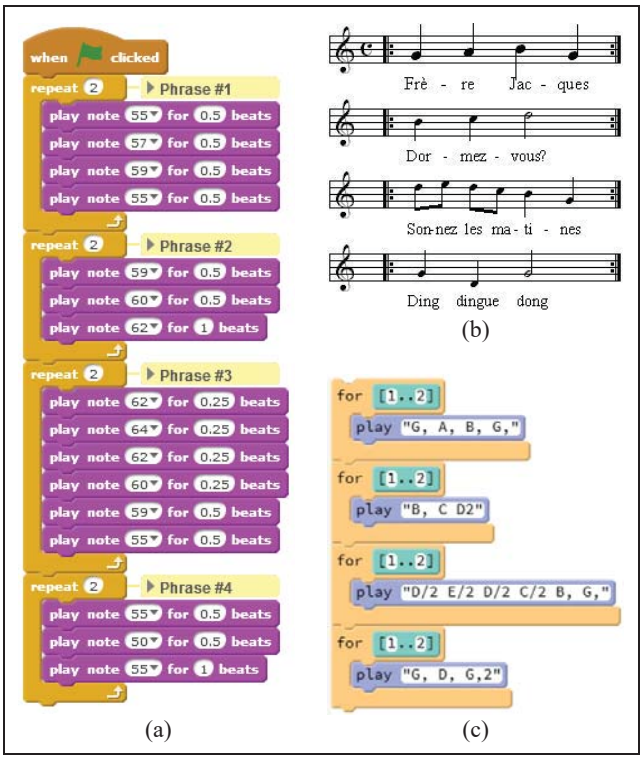


***Figure 1. Frère Jacques* in (a) Scratch, (b) standard music notation, and (c) Pencil Code using ABC Notation.**

Once the correct ABC notation is entered into WebMusicScore, it is easy to copy and paste that into Pencil Code. The final step is then to enclose the ABC notation in Pencil Code in double quotes and add code to pass it to a function that plays it. "Pure" Pencil Code using the built-in play function is shown in Figure 1(c), but a little additional code displayed a dynamic keyboard that could show which piano key was being played as each note sounded. We wrote a custom function called sing that built on this enhanced capability to allow students to play up to four parts simultaneously simply by specifying which phrase to play on which piano.

The code in Figure 2 (shown in text rather than block mode) calls our custom sing function to play *Frère Jacques* as a round (in the key of C). Note that line 10 rests part 2 for 8 beats before "singing" that part, thus creating the round. The graphic in Figure 2 shows the keyboards as they are displayed while the music is playing.

3.2 Song Choices

Songs were chosen by the school music teacher in conjunction with the two university professors. We began the program using popular songs such Rachel Platten's *Fight Song* [17], Taylor Swift's *Shake It Off* [23], and Shawn Mendes's *Stitches* [4]. We had an arranger create simple harmony parts for these songs, but that did not prove as popular with the students as we expected. The children had trouble with the harmony parts because the rhythms of these songs are complex.

The music teacher on our team suggested that we switch to "partner songs," which were sets of three simple, complementary melodies meant to be sung together. One such set consisted of *One Bottle of Pop*, *Don't Throw Your Trash in My Backyard*, and *Fish and Chips and Vinegar*. [Go to youtu.be/u-TdsmPHjo0 to hear these songs sung individually and then together, but not by our students.] It was much easier for the students to sing and code these simpler songs in multiple parts.

```
1  for [1..2]   # part 1, phrase 1
2    sing 1, “C D E C”
3  for [1..2]   # part 1, phrase 2
4    sing 1, “E F G2”
5  for [1..2]   # part 1, phrase 3
6    sing 1, “G3/4 A/4 G/2 F/2 E C”
7  for [1..2]   # part 1, phrase 4
8    sing 1, “C G, C2”
9
10 sing 2,”Z8”  # rest for 8 beats
11 for [1..2]   # part 2, phrase 1
12   sing 2,”C D E C”
13 for [1..2]   # part 2, phrase 2
14   sing 2,”E F G2”
15 for [1..2]   # part 2, phrase 3
16   sing 2, “G3/4 A/4 G/2 F/2 E C”
17 for [1..2]   # part 2, phrase 4
18   sing 2, “C G, C2”
```

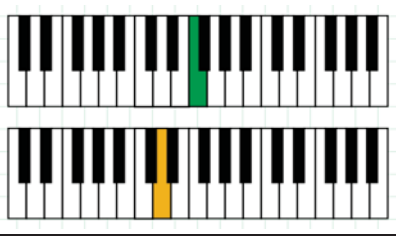


Figure 2. **Playing *Frère Jacques* as a round.**

3.3 Student Assistants

One of the program's key features was the relatively large number of university student assistants we employed. We had originally budgeted for two assistants, but it quickly became apparent that that was not enough. We really needed one university student for each pair of middle school students, so we increased the number of assistants to six.

To establish rapport, the university students (and professors) sang with the middle schoolers as well as assisted them both with reading music (which was necessary to translate scores into MIDI or ABC notation) and actual coding. About half the assistants were music majors, while the other half were computer science majors.

Unfortunately, however, we had only one female assistant: Nicole. With 12 female and 2 male children in the program, we feel that it is important that we work harder to recruit more female assistants next year, especially since Nicole provided us with invaluable insights about the complex issues that middle school girls deal with and thus helped us weather a number of storms.

As with any program that uses student assistants, some worked out well and some did not. Most were good at helping to keep the children on task and getting them past hurdles such as computer freeze-ups and simple programming issues. Some made excellent suggestions during our activities planning sessions, and some provided insightful observations when we reviewed each day's experiences.

Some assistants even established strong personal relationships with the middle schoolers and functioned as role models, which contributed significantly to the clubhouse atmosphere we were trying to maintain. When one assistant was absent, the children were always disappointed and asked if they would be there the next time we met.

3.4 Additional Performances

Another aspect of the program involved bringing in others involved in both music and computing to perform for the students. Our student assistants who were music majors occasionally performed, but we also brought in one of the *a cappella* groups on campus to demonstrate multipart singing.

Our arranger came by one day so that the children could meet the person who had created the arrangements they were singing. He taught them a new, simple multipart song, and they were simply enthralled by his ability to sing each of the different parts and not get confused!

4. RESEARCH QUESTIONS AND FIRST YEAR FINDINGS

Our research investigated two instructional questions and two supporting questions. We attempted to answer these questions both by documenting our own informal observations and by employing surveys and focus groups conducted by the external evaluation team. The former obviously suffer somewhat from observer bias, while the latter suffer somewhat from small sample size. (We started the year with 18 students and ended with 14.) Nonetheless, we feel that we can draw a number of conclusions from our first year experiences that can be reasonably supported by credible anecdotal evidence if not by hard numbers.

4.1 Instructional Questions

(1) *Can middle schoolers follow the connections from singing to digitized sound to MIDI or ABC notation and back to music to help them learn to program using the songs they like to sing?*

The answer to this question is an unqualified "yes," and we would go even further to say "yes, and sometimes with enthusiasm." Of course, not every student was "into" the coding part of the program, and some days it seemed like none of them wanted to do any coding at all. (As every parent knows, such is life when working with children.) But on other days a good number of students exhibited real excitement about their ability to "teach a computer to sing" and eagerly lined up for "show and tell" at the end of the day to demonstrate their accomplishments to others.

We began the year having students code music as a series of `play` blocks. As the year progressed, we introduced progressively more advanced computing concepts. The first of these was simple loops (like those in Figure 2), which allowed students to replay musical phrases that repeated in succession. Next we introduced loops with control variables, such as `for k in [1..3]`. This allowed students to make the connection between songs with larger repetitive structures that used first and second endings and conditional (`if`) statements in

code. We then introduced general variables, which allowed us to store and reuse musical phrases coded as ABC notation strings. A couple of students even got as far as indexed variables (one-dimensional arrays), which we used as two parallel structures to pair each note with its lyric.

One student in our program was a 7th grader, while all others were 5th and 6th graders. Only one had prior experience with computer programming, and 4 of the 14 did not have a computer at home. Given these demographics, the fact that the students had already been in school for more than eight hours by the time they began programming with us, and our own inexperience in teaching middle school students, we feel that the programming concepts we were able to introduce represent a reasonable level of learning.

(2) *Conversely, can programming their individual parts help students learn to sing in three- and four-part harmony?*

During initial discussions, the music teacher told the professors that she thought it would be difficult to get the children singing in more than two parts because they had never done it before and had no familiarity with it. By the end of the year, however, they were successfully singing the "partner songs" (Section 3.2) in three parts. They even cheered when they all finished at the same time! We can't give all the credit for that progress to the students' work in programming those songs, but the music teacher felt that at least some credit was due there.

One of the programming techniques that seemed to help students learn multipart songs was the use of variables to store and reuse musical phrases coded as ABC notation strings. This helped students see song structures, notice where phrases repeated, and understand how the melody lines went together. Again we do not want to overstate this result as it was impossible to measure objectively, we but have learned to trust the music teacher's instincts, as she is clearly highly attuned to the students' capabilities.

4.2 Supporting Questions

(3) *What resources, models, and tools (RMTs) are necessary to integrate STEM education into a middle school after-school choral program?*

We found the public school resources to be severely limited. The computer network and access to the Internet are so severely tied down that Windows systems could not access the network at all and sites such as YouTube were not accessible without teacher credentials. Luckily, access to all the music sites mentioned earlier was available to us.

We were also unable to install software on the school systems, and no one in the school had authority to do so, either. We had to make a request of the central school district office, and that took weeks to be fulfilled. To resolve this issue, we are planning to buy systems specifically for the project's exclusive use.

Our model for the after-school program was initially what we were used to: a laboratory class. This approach did not prove viable, as the students were simply unable to pay attention to instructions for more than a minute or two in the after-school environment. We therefore transitioned toward a clubhouse model, where students worked one-on-one or two-on-one with a professor, university student assistant, or another middle school student.

We prepared handouts with instructions and illustrations so that they could work on their own rather than listen to us explain how to accomplish the day's goals. We also hired three times the number of university student assistants than we had planned to, as it became evident that they were needed in the clubhouse model.

The tools we used have been discussed previously, but it is important to reemphasize that they changed and rotated throughout the year. The switch from Scratch to Pencil Code was the biggest unanticipated change at the beginning of the year, and the discovery of WebMusicScore for writing ABC notation proved a godsend.

(4) *Can the involvement of older students and teachers who match the students' racial and/or cultural backgrounds have a positive effect on the "people like me don't (or can't) do that" belief that so often stifles efforts to attract underrepresented groups to STEM?*

One of the issues that concerned us was our ability to "connect" with the middle school students. Almost all of the children had very different racial profiles from our own, and, as noted in Section 3.3, the vast majority were female (86% of 14 total). Numerous authors such as Kohl [11], Delpit [3], and Tatum [24] have written about issues related to race in the classroom, and we feared that at least some of those issues extended to gender, as well.

Looking back, it appears that we need not have been so concerned. While almost all of the children were black or of Asian or Hispanic or mixed descent and the professors and assistants were mostly white males, this did not appear to be a major stumbling block. The children did build relationships faster with some of the university students than with the professors, so age might also have been a factor here. We simply do not know whether the relationships would have developed more rapidly or deeply with mentors who more closely matched the children's racial profiles.

Gender seemed to play a larger role in relationship-building, as we observed at least some of the female children gravitating more freely toward Nicole, our lone female assistant, than toward the males. Then again, Nicole is of Hispanic descent, so that may have influenced the children's feelings toward her, as well. That said, the children also gravitated toward working with one of the white male assistants who exhibited exceptional teaching and interpersonal skills. With time, however, we feel that each of us was eventually able to "connect" to each child.

The bottom line is that although we can only report observational and anecdotal evidence, we feel that, in our case, the premise of this question appears to be unsupported. That said, we acknowledge that our project has a small sample size and that our results are dependent on the many specific personalities involved. Thus, it is difficult—if not impossible—to generalize these results.

4.3 Lessons Learned So Far

Based on these experiences, here are the main lessons we feel that we have learned so far:

- Working with children after they've been in school for 8 or more hours is hard. There are times when one has to just let them play. In addition, one must understand that some days they just won't want to code, and that pushing them to do so is futile. (There were even days when the beloved and experienced music teacher found it difficult to get them to sing.) "Go with the flow."
- Knowing how the children perceive a program such as this is also hard. Attitudes often cannot be seen, and one must be careful not to make assumptions about observed behaviors.
- There is a strong need for concrete, over-arching goals to tie sessions together. (This is one of the major issues we need to address in our second year.)
- Student assistants must be vetted carefully for their ability to work with children. (While we had no major problems with any of our assistants related to their interactions with the children, it is clear that some were far better at helping the children learn and remain on task than others.)
- There is simply no substitute for working with a quality teacher who has a strong rapport with the children and understands where they're coming from. On numerous occasions our music teacher partner pointed out where some of our assessments and impressions of how the program was going were wrong. For example, one student didn't seem to "engage" with the program, so we assumed that she simply didn't want to be in it. Our partner pointed out that all of this student's friends had dropped out of the program for one reason or another, and that the fact that she was still with us was a strong indicator of her desire to be there. We would never have known this, and we quickly learned to rely on our partner's perceptiveness and trust her instincts.
- Despite all the time and patience it takes to get children to focus on learning in an after-school program and the inevitable ups and downs of such an endeavor, there are numerous, priceless, unforgettable "ah-ha" moments that make all the effort worthwhile.

5. FORMAL EVALUATION

Our project was supported by an evaluation team centered in the UMass Lowell Center for Program Evaluation. The team observed after-school sessions, administered surveys to both students and faculty, and conducted focus groups with both students and faculty. The following statements are adapted from the Conclusion section of their Year 1 Evaluation Report:

> Faculty suggested that while end-of-year student programming was not as advanced as they had hoped, it clearly improved during the year. Overall, the students had more exposure to music than computer programming prior to starting the project. Therefore, the learning curve was steeper for computer programming. There were understandably individual differences in interest and skill levels; at least one student exceled in computer programming and was able to assist other students while pushing herself and the facilitators so that she could go further in her programming.
>
> The students' self-efficacy toward teaching someone else to program increased substantially over the course of the project. They seemed to like programming more at the end of the school year and reported an increase in interests in both music and computing programs outside of school.
>
> Students clearly enjoyed sharing their work with other students and the facilitators in the program. They showed confidence in their abilities to create the music on the computers and enjoyed the opportunity to display these abilities.
>
> The majority of the students were from underrepresented groups in STEM. While a shortcoming of the project may be the lack of facilitators from underrepresented groups, the ratio improved over time. Additionally, despite the lack of facilitator role models who "looked like" the student participants, the facilitators were able to connect well with the students.
>
> Most of the data pointed to the primary strength of the project being the relationships that were built between the students and facilitators that allowed the students to feel that they were in a safe place to learn and ask questions. Both the students and facilitators enjoyed working with each other.
>
> It is likely that this opportunity for middle school students, with more one-on-one interaction with students and faculty from the university, will a have long-lasting impact on their views toward education in general and, more specifically, toward the music and computer science fields.

6. ANTICIPATED CHANGES FOR YEAR 2

There are several changes that we plan to make to the program to improve it in its second year:

- We plan to recruit additional student assistants to increase the ratio of facilitators to students. We also hope to recruit more female and minority university students with whom the middle school students can identify. As discussed in Section 4.2, while the middle school students warmed to the white professors and university students over time, we observed that the girls had a stronger connection to our lone female student facilitator.
- We plan to buy new computers that will allow us to have more control over the school computing environment.
- We will attempt to produce a holiday album of both singing and computer-generated music that the students can share with their families and friends to further motivate their involvement in the program.
- We will attempt to have the students build a website that showcases their work and further motivates their involvement in the program.
- We will explore the possibility of partnering with another school also to further motivate involvement in the program.
- We will change some of the members of our advisory board to hopefully improve the feedback we are getting from that group.

We also intend to make better use of possible collaborations within the school itself. For example, the computer teacher has been on our advisory board, but we hope to tap her expertise more directly to gain insights on working with our students. The school also has an award-winning Social Studies teacher who is a technology wiz

and who has done collaborative work not only with the students in her own classroom, but also between her students and students in other schools. Both of these teachers may be able to help us with classroom management and possible activity development.

But perhaps most importantly, we intend to expand the "play" and/or "gaming" aspects of the one-on-one and social interactions of the program to improve the quality of teaching and student engagement. Students expressed the desire for such an approach in the focus groups conducted by our evaluation team, and we believe that incorporating a play- or game-based approach will give the students more chances to experiment, work collaboratively, and drive their own learning.

7. ACKNOWLEDGMENTS

This project is supported by Award No. 1515767 from the National Science Foundation Division of Research on Learning. Any opinions, findings, conclusions, or recommendations expressed in this paper are solely those of the authors and do not necessarily reflect the views of the National Science Foundation.

The authors acknowledge the contributions of the following people and thank them for their support of the project, particularly those at the Bartlett Community Partnership School (BCPS):

- Peter Holtz, Principal, BCPS
- Kara Haas, computing teacher, BCPS
- Kristin Weigold, after-school coordinator, BCPS
- Joe Meli, student asst., UMass Lowell CS major
- Jonathan Roche, student asst., UMass Lowell CS major
- Nicole Vasconcelos, student asst., UMass Lowell Music major
- Ben Miller, student asst., UMass Lowell Music major
- David Cherepov, student asst., UMass Lowell Math major
- Andrea Andzenge, evaluator, UMass Lowell Grad School of Ed
- Dan Washington, music arranger, Xerox Corporation and Barbershop Harmony Society

Note: The parents or guardians of all students pictured in this paper have signed IRB-approved photo release forms.

8. REFERENCES

[1] Audacity Open Source Development Team (2011). *Audacity: The Free, Cross-Platform Audio Editor and Recorder*. audacity.sourceforge.net *accessed* Oct. 25, 2014.

[2] Chambers, J. (2016). *An ABC Primer*. trillian.mit.edu/~jc/music/abc/doc/ABCprimer.html *accessed* Aug. 16, 2016.

[3] Delpit, L. (2006). *Other People's Children: Cultural Conflict in the Classroom*. New York: The New Press.

[4] Geiger, T., Parker, D., & Kyriakides, D. (2015). *Stitches*: Hal Leonard Music Publishing.

[5] Greher, G.R., & Heines, J.M. (2009). *Sound Thinking: Conceptualizing the Art and Science of Digital Audio for an Interdisciplinary General Education Course*. Assoc. for Technology in Music Instruction (ATMI) 2009 Conf. Portland, OR.

[6] Hatasuhut, R. (2016). *Web Music Score*. www.ronaldhutasuhut.com/wp-content/wms/WebMusicScore.html *accessed* Aug. 18, 2016.

[7] Heines, J.M., Jeffers, J., & Kuhn, S. (2008). *Performamatics: Experiences with Connecting a Computer Science Course to a Design Arts Course.* Int'l. Jrnl. of Learning **15**(2):9-16.

[8] Heines, J.M., Greher, G.R., & Kuhn, S. (2009). *Music Performamatics: Interdisciplinary Interaction*. Proc. of the 40th ACM Tech. Symposium on CS Education, pp. 478-482. Chattanooga, TN: ACM.

[9] Heines, J.M., Greher, G.R., Ruthmann, S.A., & Reilly, B. (2011). *Two Approaches to Interdisciplinary Computing+ Music Courses.* IEEE Computer **44**(12):25-32.

[10] Ho, W.-C. (2004). *Use of information technology and music learning in the search for quality education.* British Jrnl. of Educational Technology **35**(1):57-67.

[11] Kohl, H. (1994). *"I Won't Learn from You" and Other Thoughts on Creative Maladjustment*. NY: The New Press.

[12] Magerko, B., Freeman, J., McKlin, T., McCoid, S., Jenkins, T., & Livingston, E. (2013). *Tackling engagement in computing with computational music remixing*. Proc. of the 44th ACM Tech. Symposium on CS Education, pp. 657-662.

[13] Martin, F., Greher, G.R., Heines, J.M., Jeffers, J., Kim, H.-J., Kuhn, S., Roehr, K., Selleck, N., Silka, L., & Yanco, H. (2009). *Joining Computing and the Arts at a Mid-Size University.* Jrnl. of Computing Sciences in Colleges **24**(6):87-94.

[14] National Science Foundation (2007). *NSF Award #0722161 - CPATH CB: Performamatics: Connecting Computer Science to the Performing, Fine, and Design Arts*. CNS: Division of Computer and Network Systems, www.nsf.gov/awardsearch/showAward?AWD_ID=0722161 *accessed* Nov. 4, 2013.

[15] National Science Foundation (2011). *NSF Award #1118435 - Computational Thinking through Computing and Music*. DUE: Division of Undergrad. Ed., www.nsf.gov/awardsearch/ showAward?AWD_ID=1118435 *accessed* Nov. 4, 2013.

[16] Pencil Code (2016). *Dream it. Code it*. pencilcode.net *accessed* 8/16/2016.

[17] Platten, R., & Bassett, D. (2015). *Fight Song*: Columbia Records.

[18] Ruthmann, S.A., & Heines, J.M. (2009). *Designing Music Composing Software with and for Middle School Students: A Collaborative Project among Senior Computer Science and Music Education Majors*. Association for Technology in Music Instruction (ATMI) 2009 Conference. Portland, OR.

[19] Ruthmann, S.A., Greher, G.R., & Heines, J.M. (2012). *Real World Projects for Developing Musical and Computational Thinking*. 30th Int'l Society for Music Education (ISME) World Conf. on Music Ed. Thessaloniki, Greece.

[20] Scratch (2016). *Create stories, games, and animations; Share with others around the world*. scratch.mit.edu *accessed* 8/16/2016.

[21] Stinson, L. (2013). *Google and apple alums invent adorable robots that teach kids to code. Wired*.

[22] Studer, K. (2005). *Maximum Technology in the Music Classroom: Minimum Requirements.* Teaching Music **13**(3):44-47.

[23] Swift, T., Martin, M., & Shellback (2014). *Shake It Off*: Big Machine Records.

[24] Tatun, B. (1997). *"Why Are All the Black Kids Sitting Together in the Cafeteria" and Other Conversations About Race*. New York: Basic Books.

[25] Trueman, D. (2011, as quoted by Jacqui Cheng). *Musicians, Tune Your Keyboards: Playing in a Laptop Orchestra*. arstechnica.com/gadgets/news/2011/07/laptop-orchestras-what-are-they-and-where-did-they-come-from.ars *accessed* Nov. 14, 2011.

[26] Walshaw, C. (2016). *ABC Notation*. abcnotation.com *accessed* Aug. 16, 2016.

Two Approaches to Interdisciplinary Computing+Music Courses

Jesse M. Heines, Gena R. Greher, S. Alex Ruthmann, and Brendan L. Reilly
University of Massachusetts Lowell

The developers of a university curriculum designed to bridge the gaps between the two disciplines have found that there are numerous ways to introduce arts majors to computing, and science and engineering majors to the arts.

The intersection of computing and music can enrich pedagogy in numerous ways, from low-level courses that use music to illustrate practical applications of computing concepts to high-level ones that use sophisticated computer algorithms to process audio signals.

We explore the ground between these extremes by describing our experiences with two types of interdisciplinary courses. In the first, arts and computing students worked together to tackle a joint project, even though they were taking independent courses. In the second, all students enrolled in the same course, but every class was taught by two professors: one from music and the other from computer science. This course was designed to teach computing and music *together*, rather than as one in service to the other.

WHO'S THE MORE CREATIVE?

It is the first day of a new semester. Two students walk into your class. You have never seen them before, and you know nothing about them. When you identify them and check their primary fields of study, you see that one is a computer science major, the other a music major. Which do you assume is the more creative?

Surely, most of us would say it is the music major. The general perception is that people in the arts are more creative than those in the sciences, particularly those in computing. But is this truly the case?

Consider the types of learning experiences that characterize each field. In music, students mainly focus on the *re-creation* of art. They learn to master their instruments by studying someone else's original creations. The *composition* of original works is advanced study, typically pursued by only a handful of music majors and usually at the graduate level.

In CS, students might initially re-create programs that implement known algorithms, but they quickly progress to writing original programs to solve problems. Those problems might be carefully bounded, but good students tend to devise solutions that exhibit a wide range of approaches.

It is interesting to note that music students also must learn concepts and syntax. Think of staves, notes, key signatures, accidentals, fingerings, and so on. The difference is what they do with these. In general, they apply what they have learned to try to play a piece exactly as their teachers say it should be played. CS students try to apply what they have learned to *solve a problem* outlined by their teachers.

So, on reconsideration, which do you now judge to be the more creative?

0018-9162/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

Table 1. Computing+Music course offerings.

Listing department	Number	Percent
Music	40	77
Computer science	9	17
Co-listed	3	6
Type of instruction	**Number**	**Percent**
Single instructor	28	54
Team teaching	8	15
Not identified	16	31
Student level targeted	**Number**	**Percent**
1st- and 2nd-year undergraduates	21	40
3rd- and 4th-year undergraduates	19	37
Graduate students	5	10
Multiple levels	7	13

INTERDISCIPLINARY LEARNING

It is not our purpose, of course, to instigate an argument over who is more creative than whom. But it certainly is our purpose to break stereotypes and to stress that when we look at science and engineering majors versus their peers in the arts, business, and other supposedly nontechnical majors, it is clear that they have much to learn *from each other*. It is not much of a stretch to assert that the technologies most of our CS graduates will be working on 5 to 10 years after they graduate probably have not been invented yet. This can make it a bit hard to decide what or how we should teach them. We have therefore based our work on the following postulates.

Once our CS students graduate, it is very likely that they will never again write a program of any significant size by themselves. Instead, they will work in teams, and those teams will undoubtedly be interdisciplinary. Even if certain members of the team do not write a single line of code, they will have a say not only in what a program does, but also in how it is implemented.

Basic skills will remain basic. An array will always be an array, and a linked list will always be a linked list. With all the buzz about students seeking CS programs with concentrations in game development, programmers who succeed in that subfield will be those who understand that interesting games are built on the fundamentals of algorithms and data structures, just as musicians understand that interesting music is built on the fundamentals of melody, rhythm, and harmony. As Michael Zyda stated, "The game industry … wants graduates with a strong background in computer science. It does not want graduates with watered-down computer science degrees, but rather an enhanced set of skills."[1]

The need for everyone to have basic computer skills will only increase. Jeanette Wing stated that the basic skill in problem solving is "computational thinking," which "involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science."[2] According to Wing, this "is a fundamental skill for everyone, not just for computer scientists." We strongly agree, and we feel that exposing arts students to computational thinking *within their own field* has huge potential for enhancing their education.

Everyone has something to learn from everyone else. Virtually all jobs today involve interdisciplinary teams, and working in such teams usually requires abandoning assumptions about our coworkers' fields. Reflecting on one of the assignments in our interdisciplinary course, a CS major wrote, "It was great to work with someone as musically (and graphically) inclined as Maria [a music major]. I lack a lot of knowledge about both of those, and her ideas made very notable improvements in the programming as well as the music and graphics." Note that the CS major specifically mentions improvements to the *programming* based on ideas from the music major.

COMPUTING+MUSIC COURSES

To address these issues, we developed two interdisciplinary course models that our colleague Fred Martin dubbed *synchronized* and *hybrid*.[3] The synchronized model pairs two independent, upper-level courses in different disciplines and requires interdisciplinary teams of students to complete a project collaboratively. The hybrid model is a single course taught by two professors from different disciplines, with both in the classroom throughout the semester.

These are, of course, but two of myriad models employed in interdisciplinary computing+music courses. To put our work in perspective, we took an *informal* look at 52 courses at 40 colleges and universities that cover computing through music or music through computing. Some of these were identified by attendees at a March 2011 workshop on this topic under the auspices of the ACM SIGCSE Music Committee[4] and sponsored by the NSF-funded LIKES project[5] (www.likes.org.vt.edu). Additional courses were found by the student researcher on our team, who searched the Web for syllabi that combined computing and music in interdisciplinary courses.

Our search criteria specifically *excluded* audio recording and production courses that have the shaping of sound through electronics and signal processing as their primary objectives. Although these courses fall at the intersection of computing and music, they focus on *using* technology to achieve desired sounds rather than *teaching* computational and musical concepts together. Table 1 presents general information about the courses we discovered and gives an overall picture of the landscape.

Table 2 presents the content of the 52 courses, as gleaned from their posted syllabi. This is an inexact measure, to be sure, but it still gives a somewhat reasonable view of the field. (The numbers in each section do not add

up to 52 and the percentages do not total 100 percent because some entries fall into more than one category.)

There is indeed a large range of courses offered, subjects covered, perspectives taken, teaching styles employed, and software systems used. Based on a review of this data and reflections on our familiarity with some of the teachers of these courses, the following overall picture emerges:

- At the upper end of the curriculum, virtually all courses that cover computing+music are advanced offerings by music departments. We know of no upper-level CS courses dedicated to addressing issues faced by musicians (although of course there may be some unknown to us).
- Courses and research at the upper end require deep understanding of *both* computation and music. Examples include the algorithmic composition work by Michael Edwards[6] and by Andrew Brown and Andrew Sorenson.[7]
- At the lower end of the curriculum, music is typically used to demonstrate or to introduce concepts. This is music *in service to* computing, not music *integrated with* computing. An example is the media computation work by Mark Guzdial and Barbara Ericson.[8]

Our work attempts to fill some of the gaps between these types of courses by integrating computing and music at a high conceptual level. The synchronized course targets mid- to upper-level music and CS majors with the intent of furthering students' knowledge of both. The hybrid course is a general education offering open to all students in the university. It attempts to provide an understanding of where computing and music interact, at a level that is accessible to students without deep knowledge of one or the other. Thus, our work is at both ends of the instructional spectrum.

COMBINED GUI PROGRAMMING AND MUSIC METHODS

One way to get started in interdisciplinary teaching and learning is to connect the students in two existing courses through a joint project. Administratively, this is a "low-hanging fruit" approach because it does not involve getting a new course approved or making any changes to the course catalog. All that is needed are professors who agree to collaborate with each other to build an interdisciplinary project into their courses.

In our case, the CS professor teaches a project-based course in graphical user interface programming, which fit nicely with a project-based course on teaching methods taught by the music professors. After reviewing the projects that we assign in our respective courses, we decided to make our initial foray into interdisciplinary teaching using a "found instruments" project that has been used in music for years.

Table 2. Computing+Music course content.

Disciplines covered	Number	Percent
Sound/audio	37	71
Computer science	36	69
Music (composition)	22	42
Music (theoretical)	12	25
Media	5	5
Primary focus	**Number**	**Percent**
Composition	31	60
Sound symbols	27	52
CS (introductory)	18	35
Sound processing	17	33
CS (specialized)	12	23
Music theory	7	13
Interactive media	1	2
Software used	**Number**	**Percent**
Max/MSP	11	21
Audacity	4	8
Processing	4	8
SuperCollider	3	6
ChucK, Disklavier, Pro Tools, Reason	2 each	4 each
Audition, Garage Band, Matlab, Peak, PureData, Reaktor, Scratch, Sibelius	1 each	2 each

The music assignment

For the musicians, the purpose of our assignment is similar to Andrew Hugill's description of a project intended "to strip away previous ideas of 'musicianship,' [by] reevaluating the sounding properties of objects, how they may be made into instruments, how playing techniques might be developed, and how music may be created as a result."[9] Music students are asked to do the following:

- Using only household object(s), create a musical "instrument" that can produce several different pitches or timbres. Your instrument must be able to produce several different types of sounds, or sounds with several different characteristics.
- Create a composition for your instrument that employs a specific musical form of your choice. It need not be long. A 2-3 minute piece is sufficient, but it must include distinct sections that give it form. That is, your composition must include distinctive opening, middle, and closing sections.
- Devise a system of creative notation that others will be able to understand well enough to perform your composition. Your notational system should not resemble traditional musical notation in any way.

Figure 1. Mike playing his jacket as a found instrument.

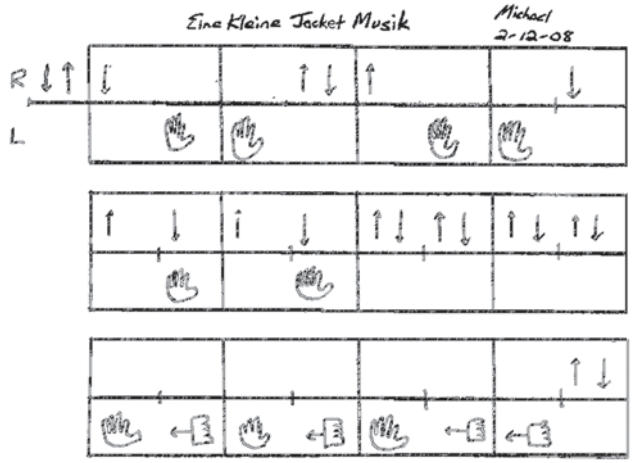


Figure 2. Mike's notation for his composition.

- Bring your instrument and notated composition to class. Come prepared to explain your work and to perform your piece.

To achieve camaraderie and pique interest, the CS majors are also given this assignment. Our experience is that the CS students "find" instruments that exhibit just as much novelty as those of their music counterparts. When we get the students from the two courses together, we do several things to build community, including having them jam on their instruments in mini-ensembles. Again, the CS students "get into" this project just as much as the music students, and the resultant "music" is, well, "interesting."

In another class activity, students try to play each other's found instruments from the notations created for those instruments. We have them do this without first hearing the original composer play the piece and without any verbal explanation of the notational system. This is a good test of the communicability of the notation by itself, and it opens up several avenues for discussion of human factors. As an example of this activity, see www.youtube.com/watch?v=IJuGoYnCxSs.

The computing assignment

The found instruments project connected to computing through the creative notation. In this project, we

- introduced CS students to standard music notation software using Finale NotePad (www.finalemusic.com/NotePad) and Noteflight (www.noteflight.com);
- assigned CS and music teams and charged the CS students with creating a music notation program for the notation devised by their music partners; and
- scheduled several joint classes in which the music students could work with the CS students on the programs' designs, review the CS students' works in progress and offer comments and suggestions for improving the programs, and finally act as usability test subjects on the finished products.

Some of the programs produced as a result of these collaborations and the lessons learned from them were truly astounding.

As Figure 1 shows, in one of the best of these projects, Mike, a music student, used his jacket as a found instrument, creating sounds by slapping it, rubbing it, working the zipper, and so on. He then created a piece satirically named *Eine Kleine Jacket Musik*. Figure 2 shows an excerpt from Mike's creative notation. Performances of Mike's piece first by Chase, a CS student, and then by Mike himself are posted at www.youtube.com/watch?v=iD4dEZOTiIg.

Figure 3 shows part of Mike's partner Chris's composition to demonstrate the CS concepts and skills involved in developing such a program.

In Figure 3a, a few icons from the tool palette on the left in the composition program have been placed onto the right- (R) and left-hand (L) staves in the composing area by either dragging and dropping them or double-clicking on them in the tool palette. In Figure 3b, the insertion cursor is positioned between the sixth and seventh icons on the left-hand staff, as indicated by the thick vertical bar. At this point, double-clicking on an icon in the tool palette would insert the icon to the right of the insertion cursor, which is to the left of the last icon on staff L.

In Figure 3c, the backspace key has just been pressed, and the blank (or "rest") icon that the arrow cursor in Figure 3b pointed to has disappeared. The issue is that the thick vertical bar insertion cursor has also disappeared,

Figure 3. Chris's music composition program for Mike's jacket notation. (a) State 1: a few icons from the tool palette on the left in the composition program have been placed onto the right- (R) and left-hand (L) staves in the composing area. (b) State 2: the insertion cursor is positioned between the sixth and seventh icons on the left-hand staff. (c) State 3: the icon pointed to by the arrow cursor in Figure 3b has been deleted. (d) State 4: the scratch icon in the tool palette (indicated by the arrow cursor) has been double-clicked to insert it into the composition.

leaving users to wonder where the insertion point is. In most editors, the insertion point would not change—that is, if the user double-clicked an icon in the tool palette at this point, that icon would still be inserted to the left of the right-most hand icon on staff L.

Unfortunately, this is not what happens. Instead, when the "scratch" icon is double-clicked, it is inserted at the beginning of the staff, as Figure 3d shows. This may be fully logical to a programmer who has implemented the composition area as a pair of linked lists, but it is not at all logical to someone used to working with any sort of text editor.

When the anomaly was pointed out to Chris, he immediately recognized the problem and said, "I can't believe I didn't notice that." But that's exactly why usability tests are needed. Programmers are often "too close" to their work to see even the most obvious user interface issues. Teaching this point in a lecture setting requires students to mentally connect theory and practice. When it is learned from a peer while testing the student's own software, the connection is far more concrete, and the lesson is learned at a deeper level that is more personal and, therefore, more effective.

Thus, the fresh views of students in other disciplines can teach valuable lessons to our computing students. Likewise, for music majors, helping nonmusicians translate their musical concepts into computer programs can shed light on the clarity of their thinking—or lack thereof. Such reciprocal learning,[10] in which students learn from each other instead of just from their professors, exemplifies one of the best characteristics of interdisciplinary courses.

SOUND THINKING

Our synchronized courses worked well at the upper end of our curricula, but we also wanted to work at the lower end so that we could introduce more students to the benefits of interdisciplinary courses. Following the pioneering work of Holly Yanco and colleagues in combining art and robotics at our university,[11] we developed Sound Thinking, a new hybrid course that could be offered to all students in the university (http://soundthinking.uml.edu).

Two characteristics about the way in which Sound Thinking was put into the course catalog contributed significantly to its success. First, it was co-listed in both the music and CS departments. Second, we applied for and were granted general education status for the course. Arts students who take it register using the CS department number and receive science and technology general education credit. Science students register using the music department number and

Figure 4. Top view of the lever drumitar.

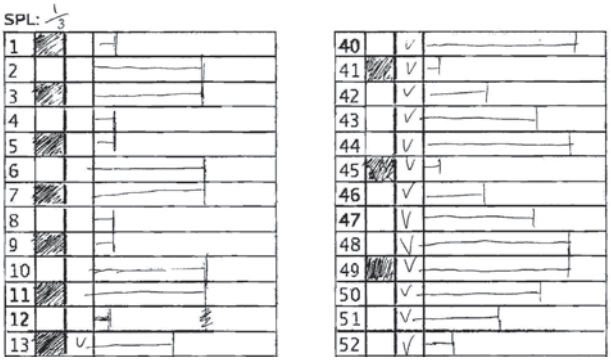


Figure 5. Notation for playing the lever drumitar.

receive arts and humanities credit. These characteristics were essential to achieving the critical number of registrations needed for the course to run, especially with two professors present at all class meetings.

Revisiting found instruments

We also used the found instruments project at the beginning of Sound Thinking, but we took it in another direction. After students created their instruments and notations, we had them record the sounds their instruments could make and then used those as an introduction to sound editing.

Eric, a CS student, created what he called a "lever drumitar," shown in Figure 4. He strung a guitar string across the opening of a cup, secured it with strong tape, and rigged up a carabiner to use as a lever for changing the cable's tension. This allowed him to produce different sounds when he strummed the cable with a soda can tab.

Figure 5 shows the original notation that Eric created for his instrument. Each row represents an action. If the square in the second column is filled in, the string is to be strummed. A V in the third column indicates that the time duration is to be shortened. The length of the line in the fourth column indicates the carabiner's position.

For the next assignment, students recorded the various sounds their found instruments could generate and loaded them into Audacity. They then created original compositions by looping and combining those sounds. To hear Eric's original lever drumitar sounds and his remixed composition, go to www.youtube.com/watch?v=_zA_hn_4T8k.

Extending found instruments

For the next assignment, students loaded their sounds using the Scratch programming language[12] and sequenced those sounds by chaining "play sound until done" blocks together. Initially, they just created linear chains like that shown in Figure 6a. When they wanted to repeat a sound or just use it again, they simply dragged in another block and selected the sound they wanted it to play.

With a bit of experimentation, all the students succeeded in creating Scratch programs that used looping as shown in Figure 6b. With a bit more instruction and encouragement, most students were able to incorporate variables, nested loops, and conditional structures as shown in Figure 6c, as well.

Finally, with help from each other rather than from the professors—which indicates true student involvement in the course and is the best way for them to learn: by teaching others—some students figured out how to do more advanced things, such as playing two or more sounds simultaneously using the "play sound" and "broadcast" and its complementary "when I receive" block, leading to interesting and sometimes relatively complex discussions about synchronization.

Many CS concepts are at play here, and we use the word "play" intentionally. The Scratch development group at the MIT Media Lab is called the Lifelong Kindergarten Group for good reason. The ability to learn through *thoughtful* play that involves the use of creativity is at the heart of what we are trying to achieve. The music and arts students learn about computing, to be sure, but so do the CS and engineering students.

Using a visual programming environment like Scratch forces CS majors—who have been "brought up" on languages like C/C++ and Java and on text-based coding environments—out of their comfort zone. It is amazing how many of them stumble when they discover that a Scratch loop does not provide access to its index (counter) variable. It is pretty easy for them to implement a counter, but solving this problem requires a bit of creative thinking. In addition, explaining to their nontechnical peers what they are doing not only increases their partners' understanding, but solidifies their own as well. As the saying goes, "If you really want to learn something, teach it to someone else."

Sound Thinking builds on the found instruments project and its related assignments by introducing MIDI concepts and generating music using Scratch's various

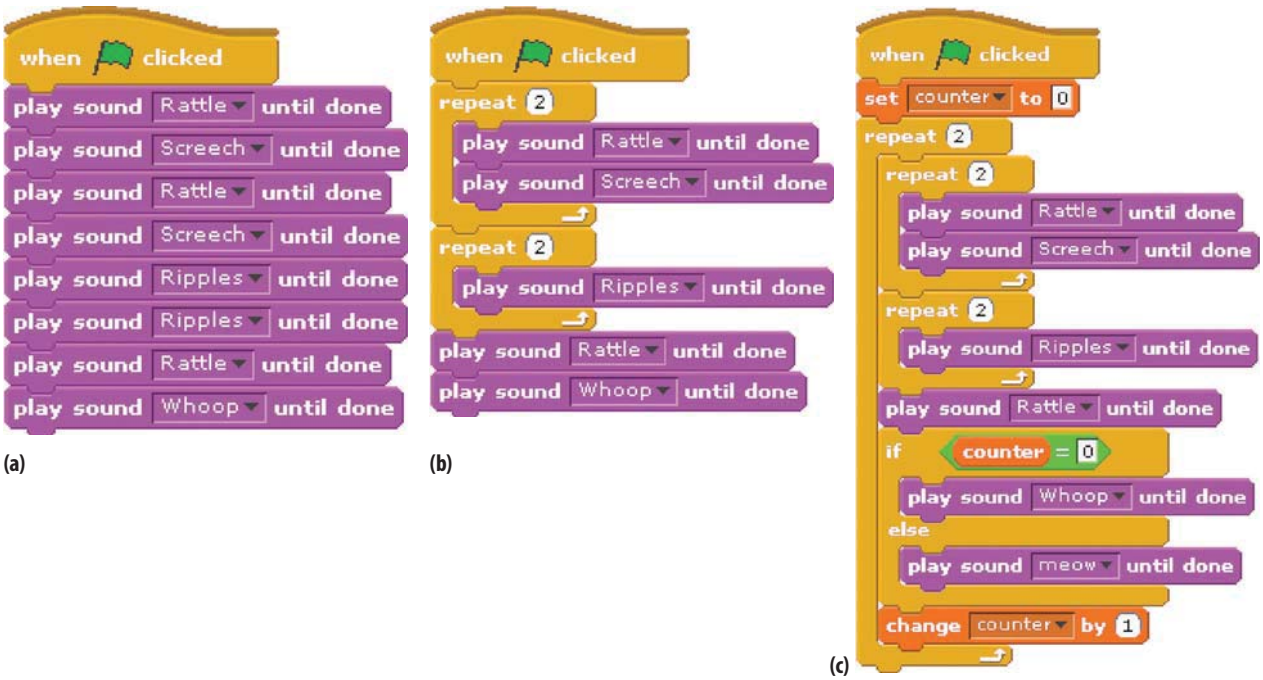


Figure 6. Scratch programs that chain sound blocks together to play (a) a straight sequence of sounds, (b) a sequence of sounds using loops, and (c) a looped sequence with conditionals.

"sound" blocks, shown in Figure 7. We have created several different types of assignments using these blocks, including having students write a composition based on only major 2nds and perfect 5ths (to take music majors out of their Western music comfort zone), writing algorithms to transpose lists as either MIDI values or interval deltas into different keys, and coding multiple parts that must be carefully synchronized.

These and other assignments are described in detail at soundthinking.uml.edu. Through these assignments, music majors learn about computing, and CS students learn about music.

INTERDISCIPLINARY TEACHING

One measure of the success of our work is the lasting effect it has on students. This is difficult to assess, but the number of students who return semesters later to tell us how they applied the concepts they learned in a different context gives us confidence that at least some of the activities we developed have generated good results. We are currently working to devise more rigorous evaluations to substantiate this belief.

In addition, the effects of our interdisciplinary experiences were not limited to the students. The professors also learned from each other, not only about discipline-specific content, but also about teaching and pedagogy. As the NSF evaluator of our Performamatics project wrote in her final report:

Figure 7. Blocks available from the Scratch sound panel.

One CS faculty member … changed his approach to teaching significantly in some situations, assigning more open-ended projects, a change well received by students. … Change in faculty is an essential but often overlooked element of institutional and curricular change.

The professors' experiences in teaching with each other were so positive that they continued to do so even after the original NSF funding expired. Then in 2011, we were awarded a grant from the NSF TUES program to disseminate our work in a series of workshops for interdisciplinary pairs of professors. The first of these free workshops will be offered 21-22 June 2012. Faculty interested in attending are invited to visit www.performamatics.org for further information and to apply.

Our explorations of ways to bridge the gaps in computing+music education are really just beginning. We believe that there are many more ways to introduce arts majors to computing and science and engineering majors to the arts, and that our approaches offer effective ways to work toward that goal in an undergraduate institution. We are constantly working to improve our current methods and to extend our work into more advanced offerings that move into live coding[13-15] and text-based music coding environments such as SuperCollider, Impromptu, Processing, and Max/MSP.

Acknowledgments

This work is supported by National Science Foundation Awards 0722161 and 1118435. In addition to the authors, Fred Martin and Sarah Kuhn of the University of Massachusetts Lowell and Scott Lipscomb of the University of Minnesota are members of these project teams. Lipscomb thoroughly reviewed an early draft of this article and provided significant suggestions for improvement. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Please see www.performamatics.org for further information, including how to apply to attend our NSF-sponsored workshop.

References

1. M. Zyda, "Computer Science in the Conceptual Age," *Comm. ACM*, vol. 52, no. 12, 2009, pp. 66-72.
2. J.M. Wing, "Computational Thinking," *Comm. ACM*, vol. 49, no. 3, 2006, pp. 33-35.
3. F. Martin et al., "Joining Computing and the Arts at a Mid-size University," *J. Computing Sciences in Colleges*, vol. 24, no. 6, 2009, pp. 87-94.
4. R. Beck and J. Burg, "Report on the LIKES Workshop on Computing and Music," ACM SIGCSE Music Committee, to be published in 2012.
5. W. Chung et al., "LIKES: Educating the Next Generation of Knowledge Society Builders," *Proc. 15th Americas Conf. Information Systems* (AMCIS 09), Assoc. for Information Systems, 2009; www.likes.org.vt.edu/files/LIKES_AMCIS09_pub.pdf.
6. M. Edwards, "Algorithmic Composition: Computational Thinking in Music," *Comm. ACM*, vol. 54, no. 7, 2011, pp. 58-67.
7. A.R. Brown and A. Sorensen, "Interacting with Generative Music through Live Coding," *Contemporary Music Rev.*, vol. 28, no. 1, 2009, pp. 17-29.
8. M. Guzdial and B. Ericson, *Introduction to Computing and Programming in Java: A Multimedia Approach*, Prentice Hall, 2005.
9. A. Hugill, *The Digital Musician*, Routledge, 2008.
10. H.F. Silver, R.W. Strong, and M.J. Perini, *Strategic Teacher: Selecting the Right Research-Based Strategy for Every Lesson*, chapt. 13, Assoc. for Supervision & Curriculum Development, 2008.
11. H.A. Yanco et al., "Artbotics: Combining Art and Robotics to Broaden Participation in Computing," *Proc. AAAI Spring Symp. Robots and Robot Venues: Resources for AI Education*, 2007; www.cs.hmc.edu/roboteducation/papers2007/c39_yancoArtbotics.pdf.
12. M. Resnick et al., "Scratch Programming for All," *Comm. ACM*, vol. 52, no. 11, 2009, pp. 60-67.
13. A. Brown, "Generative Structures Performance," 2011; http://vimeo.com/26193440.
14. A. Ruthmann, "Live Coding & Ichiboard-Enhanced Performance," 2011; www.youtube.com/watch?v=qehSEroHn4E.
15. A. Sorenson and A. Brown, "aa-cell Live Coding at the Loft 2," 2007; www.youtube.com/watch?v=tj74-q_Mxrg.

Jesse M. Heines *is a professor of computer science at the University of Massachusetts Lowell, with a strong interest in music and its power to interest students in computing. Heines received an EdD in educational media and technology from Boston University. Heines and Gena Greher are writing a book on interdisciplinary teaching that is currently under contract with Oxford University Press. Contact him at jesse_heines@uml.edu.*

Gena R. Greher *is an associate professor of music education at the University of Massachusetts Lowell. Her research focuses on creativity and listening skill development in children and on examining the influence of integrating multimedia technology in urban music classrooms. Greher received an EdD in music and music education from the Teachers College of Columbia University. Contact her at gena_greher@uml.edu.*

S. Alex Ruthmann *is an assistant professor of music education at the University of Massachusetts Lowell, where he teaches courses at the intersection of music education and arts computing. His research explores social/digital media musicianship and creativity, as well as the development of technologies for music learning. Ruthmann received a PhD in music education from Oakland University, Michigan. Contact him at alex_ruthmann@uml.edu.*

Brendan L. Reilly *is an undergraduate computer science major at the University of Massachusetts Lowell. He has played bass since grade school, participating in every musical group available to him. Contact him at brendan_reilly@student.uml.edu.*

Teaching Computational Thinking through Musical Live Coding in Scratch

Alex Ruthmann
Dept. of Music
Univ. of Massachusetts Lowell
Lowell, MA 01854
1-978-934-3879
Alex_Ruthmann@uml.edu

Jesse M. Heines
Dept. of Computer Science
Univ. of Massachusetts Lowell
Lowell, MA 01854
1-978-934-3634
heines@cs.uml.edu

Gena R. Greher
Dept. of Music
Univ. of Massachusetts Lowell
Lowell, MA 01854
1-978-934-3893
Gena_Greher@uml.edu

Paul Laidler
Student, Dept. of Computer Science
Univ. of Massachusetts Lowell
Lowell, MA 01854
1-978-934-3634
plaidler@gmail.com

Charles Saulters II
Student, Dept. of Music
Univ. of Massachusetts Lowell
Lowell, MA 01854
1-978-934-3634
kingd615@hotmail.com

ABSTRACT
This paper discusses our ongoing experiences in developing an interdisciplinary general education course called *Sound Thinking* that is offered jointly by our Dept. of Computer Science and Dept. of Music. It focuses on the student outcomes we are trying to achieve and the projects we are using to help students realize those outcomes. It explains why we are moving from a web-based environment using HTML and JavaScript to Scratch and discusses the potential for Scratch's "musical live coding" capability to reinforce those concepts even more strongly.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education — *computer science education, curriculum.*

General Terms
Design, Languages

Keywords
Performamatics, Scratch, computer science education, interdisciplinary courses, musical live coding, generative music, curriculum design.

1. PERFORMAMATICS BACKGROUND
Performamatics is a series of courses intended to attract students to computer science (CS) by tapping their inherent interest in performance and the arts. Toward that end, two CS professors have teamed with five Music, Theater, and Art professors to offer both introductory and advanced courses where assignments designed to reinforce CS concepts center around applications in the Arts. These courses are in the spirit of pioneering work done by Cooper, Dann, & Pausch [3], Guzdial [6], and Yanco et al. [15], and have been described in many other papers and presentations [5, 7, 8, 9, 12]. Readers are also referred to www.performamatics.com for links to online materials.

The most successful Performamatics courses to date (based on enrollment and student feedback) have clearly been the introductory ones. These are general education (GenEd) courses co-listed in two departments, allowing CS majors to earn Arts & Humanities GenEd credit while Arts majors earn Science & Technology GenEd credit. *Tangible Interaction Design* is a collaboration between CS and Art, while *Sound Thinking*, the course on which this paper focuses, is a collaboration between CS and Music.

2. SOUND THINKING
One of the hurdles in getting our first offering of *Sound Thinking* approved for dual GenEd credit was to convince the GenEd committee that Music majors would learn something about technology and CS majors would learn something about music. The committee feared that if project teams had both CS and Music majors, each group would naturally navigate to its own discipline and there might be relatively little true cross-over. We therefore established the following behavioral objectives for all students.

Upon completion of this course, students should be able to:
1. Identify properties of sound and describe the organization of sound into music.
2. Design a simple notation system and describe the differences between formal and informal notation.
3. Distinguish between analog and digital audio.
4. Discuss the basic differences between various audio file formats and sound compression techniques.
5. Create a web-based computer program that plays a music file.
6. Create a web-based computer program that plays a user-definable sequence of music files.

In the first half of the semester, students created compositions for "found" instruments, invented notations for those instruments, recorded the instruments' sounds, manipulated those sounds with

SIGCSE'10, March 10-3, 2010, Milwaukee, Wisconsin, USA.

audio editors, and remixed and recomposed the sounds into original compositions. In the second half, they created webpages that incorporated music, used a JavaScript Application Programmer Interface (API), and developed interactive web applications in which music played an integral part. Looking back, we see that objectives 1, 2, 5, and 6 held the most interest for students, and that's where we spent most of our class time.

Before the course was taught, there was much discussion among the Performamatics faculty about the development platform to be used for programming assignments. A CS professor said, "The Music majors will cringe if you make them code. You've got to use a visual programming environment." But a Music professor strongly disagreed, saying "One of our goals is to have them overcome any fear they have of code. We *want* them to see real code." Given that we thought students would enjoy creating webpages that they could share with their friends, we therefore chose Dreamweaver as our development platform, because it allowed viewing the page layout and its underlying code simultaneously. This helped students easily see the cause-and-effect relationship between the code and its result. We taught the basics of underlying HTML and JavaScript (with a custom API to play sounds and sound files), and only one of the 13 students had any real trouble doing the webpage development assignments.

On the contrary, most of the Music majors were very technically savvy and the post course student evaluations revealed that they wanted to know *more* about what was "under the hood." They enjoyed referring to the CS professor on the faculty team as a "magician" (because he could almost always make their pages do what they wanted when their CS partners were stumped), and they suggested that when the next the course is taught, the programming should be spread throughout the semester rather than confined to the second half.

We are now revising *Sound Thinking* for its next offering. With the help of students funded through a Research Experience for Undergraduates supplement to our NSF CPATH grant, we investigated a number of other platforms for use in the course. We have settled on Scratch [10, 11]. The remainder of this paper discusses why we feel that Scratch is an appropriate platform for teaching computational thinking through music.

3. MAKING MUSIC WITH SCRATCH

Scratch has the ability to generate and play sounds using various components in its Sound category. But if one wants to begin making *music* with more than one sound line in Scratch, one needs to address the issue of synchronization.

This issue is best addressed once students are familiar with looping, an essential concept in the implementation of many musical models. As students begin working with models where multiple voices are layered, it becomes necessary to maintain the tempo using the Scratch timer. Figure 1 shows a loop that will play a hand clap (MIDI drum instrument #39) every quarter of a beat. However, this example will not have a steady tempo, and if it was used to layer multiple voices, each would eventually fall out of synchronization.

The Scratch timer offers a way to address this problem. We first determine how many seconds our hand claps should be apart and then use the Scratch timer to control when each clap should occur. Figure 2 shows a more complex loop that will remain synchronized with the tempo regardless of the number of iterations performed. Each iteration waits until the Scratch timer reaches the value stored in variable now. This variable holds the time when a hand clap should sound. A message is then broadcast to play the hand clap. This ensures that the loop will complete and return to the "wait until" statement before the next hand clap needs to be played by delegating the actual playing to a "when I receive" event handler. The value of variable now is then changed to contain the time at which the next hand clap should be played. The Scratch timer ensures that the hand claps remain in tempo.

Note that in addition to synchronization, many other computational thinking concepts are touched upon by this example.

- looping
- initialization
- use of variables
- changing variables algorithmically
- modularization
- event processing

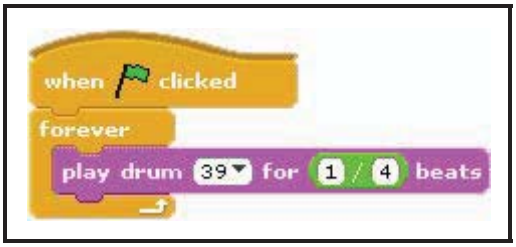


Figure 1. A hand clap loop.

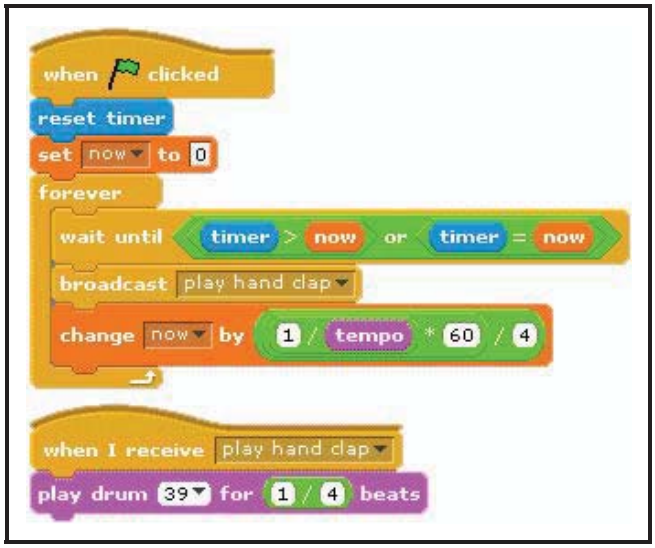


Figure 2. Hand clap loop synchronized via the Scratch timer.

4. FROM CODE TO MUSIC

Once students understand basic note and sound generation in Scratch and can implement synchronization, more musical, generative algorithms for creating and manipulating sequences of notes can be explored.

One possible starting point is to use the "forever" loop to generate random melodies constrained by lower and upper boundaries as shown in Figure 3. In this example, random musical pitches from

middle C (MIDI note #60) and the C above that (MIDI note #72) are chosen and played for half a beat. Because the "play note" function is surrounded by a "forever" loop, Scratch continues to generate notes until the Scratch stop button (●) is pressed.

Figure 3. Code for a random melody by boundary constraints.

The fact that Scratch also functions as a live interpreter/compiler makes things more interesting. This feature allows the boundaries of the random pitches as well as the duration of the sound played to be manipulated in real time through "musical live coding" [2, 13, 14] without disrupting the sounds being generated. That is, the resultant melody can be changed in real time by adjusting the upper and lower bounds of the random function and changing the duration value for the beat without stopping program execution.

The code in Figure 3 can be expanded to generate a random melody from notes provided in a pitch set as shown in Figure 4. This code implements a Scratch "list" that contains a Pentatonic (five note) pitch set. It then selects a random note from that pitch set and adds a pitch offset (MIDI note 38). In a real-time performance, the pitch could be changed by manipulating the offset to move the randomly chosen notes higher and lower through the pitch register. Additionally, through the use of the "pick random" function, the bounds of the notes chosen from the Pentatonic pitch set could be further constrained. For example, if we wanted to choose only the 2nd, 3rd, or 4th note of the set, we could change the function to "pick random 2 to 4." This technique enables live coders to create more variety in the resultant musical output.

In most music, melodies do not move by random intervals. If one has a large pitch set, random intervals could result in very large leaps from one pitch to the next. A more natural sounding melody can be generated by implementing a "random walk" algorithm to change subsequent pitches as shown in Figures 5a and 5b. This results in a more musical melody by constraining the interval movement between -4 and +4 of the prior pitch. This approach also enables the melody to move freely across the MIDI pitch spectrum rather than to be constrained by the length of the pitch set as was the case in the prior examples.

Eventually, these techniques can be expanded to model the musical styles of various composers. Our initial explorations have centered on generating music in the style of Arvo Pärt and Philip Glass. Figure 6 shows a small part of a larger algorithm inspired by Arvo Pärt's *Stabat Mater* [1]. This example iterates through the AeolianPitchSet list (organized as a descending minor scale) to select pitches and then chooses random rhythm values from the RhythmSet list. The values of the RhythmSet list were derived from an aural analysis of the *Stabat Mater* and weight the probability of selecting a whole note twice that of selecting a half note. In the full algorithm, this code is duplicated twice to create three multithreaded musical parts that are triggered via keystrokes. The addition of human control to starting and stopping threads enables the performer to create dynamic variations in musical form and texture by starting and stopping sections of the overall code.

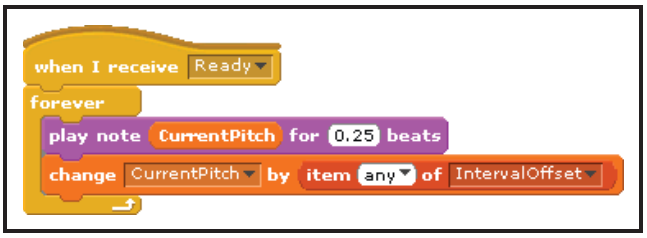


Figure 5a. Code for a melody via a "random walk" algorithm.

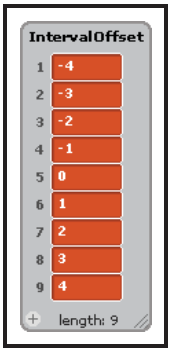


Figure 5b. Interval list for use with the "random walk" algorithm.

In addition to the live manipulation of lists, variables, and offsets shown in prior examples, Figure 6 also enables the selection of multiple pitch set lists, modification of the direction in which the list is iterated, and the ability to choose randomly or to isolate and repeat pitch values. Additional functions from the Scratch Sound and Numbers menus can be added, removed, and manipulated in real time to generate more musical control and expression. For example, a "change volume by x" function could be added to create changes in musical dynamics or to realize dynamic fading in or fading out of sections of the code. Additionally, a "change tempo by x" function could be inserted at various points to slow down or speed up the tempo. This could be set to a discrete value or by a mathematical function as shown in Figure 7.

Performing effective real-time manipulation of code (musical live coding) to create and shape generated music requires both musical and computational understanding. From a musical perspective, one needs to understand how the ongoing, generative music should sound. From a computational perspective, one needs to understand how the code can be adjusted and manipulated in real time to achieve the aural and musical changes and outcomes one

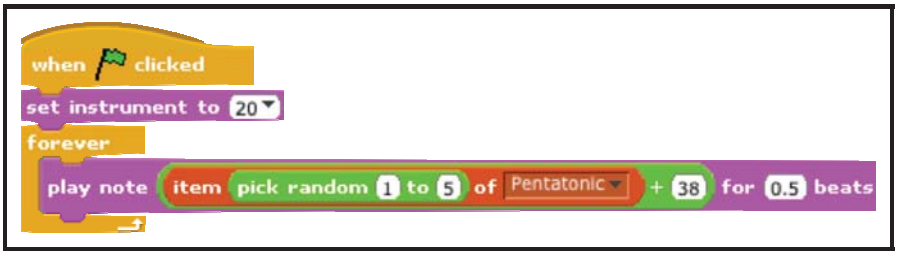


Figure 4. Code for a melody from pitch and rhythm set lists.

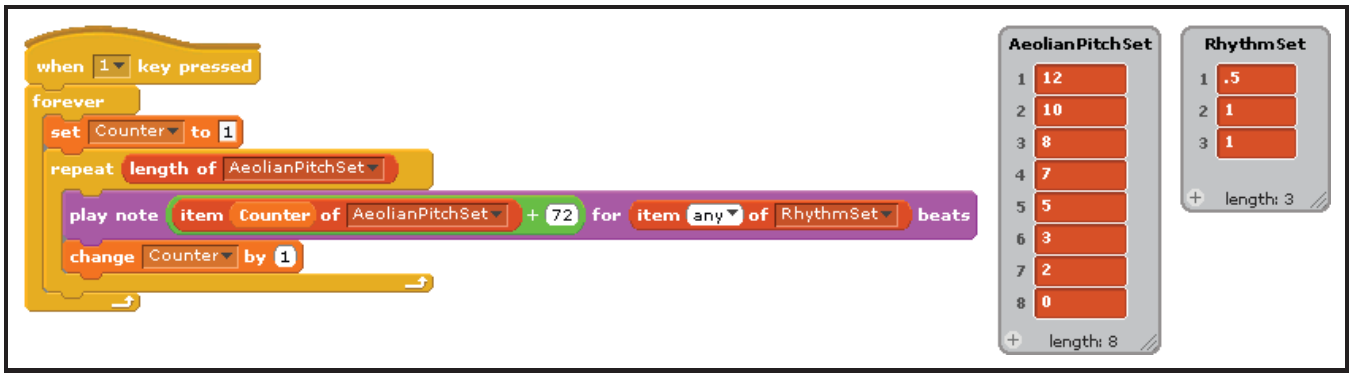


Figure 6. Melodic code inspired by Arvo Pärt's *Stabat Mater*.

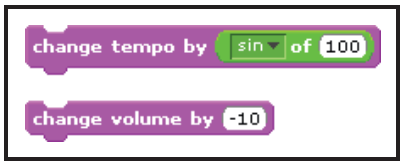


Figure 7. Examples of built-in Scratch functions well suited to musical live coding.

desires. These are advanced skills, but they can be learned through experimentation and exploration that is both educational and fun. Scratch provides a unique, easy-to-learn platform that enables musical live coding by allowing nearly all aspects of the code to be adjusted in real time. Students can then share and showcase their work in live or pre-coded performances, which is the essence of Performamatics.

5. ADDING A TANGIBLE INTERFACE

As the course develops, we plan to integrate tangible computing using IchiBoards [4] (Figure 8). Using these devices' live sensing capabilities, we can implement gestural musical input and design new instruments to perform musical algorithms implemented in Scratch.

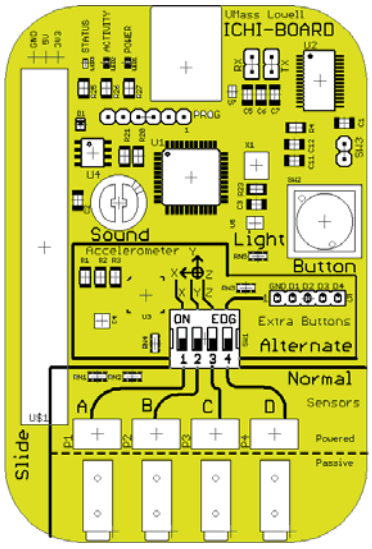


Figure 8. IchiBoard [4].

Figure 9 shows a simple program that converts an IchiBoard into a musical instrument. A "forever" loop is used to enable continuous live sensing of the board's button and slider sensors. When the button is pressed, the slider value is read and a note is played whose pitch corresponds to that value. Computational thinking comes into play because the IchiBoard's slider returns values between 0 and 100. To convert those values to a 7-note whole tone musical scale in which each interval is two equal half-steps apart, the value returned by the slider is combined with a copy of itself on which a modulus 2 operation has been applied. This ensures that when the slider is moved, the pitch of the note being played always jumps by a whole step rather than a half step. With the beat value set to 0.01, a continuous stream of pitches sounds when the button is pressed. The result is a surprisingly expressive instrument with which the user can establish a rhythm through interaction with the button and play gestural pitch sweeps through manipulation of the slider.

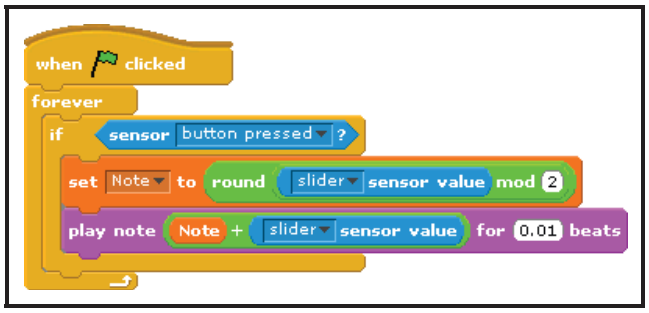


Figure 9. Code for a simple IchiBoard musical instrument.

The previous example only takes advantage of two of the eight possible sensor inputs on the IchiBoard. More complex interface configurations and Scratch code are currently in development that will enable more interesting musical performances and live coding demonstrations of computational thinking.

The integration of IchiBoards as an interface for tangible computing enables discussion of CS hardware concepts such as:

- What is a device?
- What is a sensor?
- What is a signal, and how is it detected in software?
- What is an event, and how is it detected in software?
- What different types of events are triggered by various devices (real and virtual)?

Integrating physical computing with Scratch's graphical coding environment provides a unique platform for expressive computing in real time. Programs can not only be written to create music, but they can be written to model musical environments that are performed through musical live coding or the design and interfacing of tangible computing devices such as the IchiBoard.

6. INTERDISCIPLINARY BENEFITS

After taking *Sound Thinking* in the Spring 2009 semester, Music major Charles Saulters developed a strong interest in using computational thinking as a means of developing more expressive gestural music controllers. He pursued a Research Experience for Undergraduates with us over the summer, exploring ways to apply these Performamatics concepts in even more exciting ways. He describes his work as follows.

> I am interested in enabling others to achieve more than they ever thought possible through the use of computational thinking in real world situations that are relevant and interesting to students. One "hook" that I found particularly interesting is the manipulation of virtual instruments, composing for and performing using nontraditional devices such as the iPod Touch. Now more than ever, we musicians find ourselves in an age where technologically almost anything is possible. It is therefore crucial that we understand what makes computers function and acquire a strong working knowledge of programs and the coding behind them.
>
> Interdisciplinary collaboration helps cultivate new and exciting innovations that can bring about the revitalization of CS education for which Performamatics was conceived. Using music as a hook, we can create innovative live performances and interesting visuals in conjunction with "musical live coding" to tap the imagination of people who might never have considered CS as a possible major. People (like myself) tend to be intimidated by the mystifying technical jargon. However, with more exposure to interesting multi-disciplinary projects, students can start thinking computationally and actively using that new way of thinking in a hands-on way without even realizing they are doing so. At that point, the fear is gone.
>
> Devices such as the iPod Touch and iPhone are ideal tools for exploring computational thinking. They are easy to use, have simple, intuitive user interfaces, and have a wide range of functionality: file transfer, web browsing, MIDI control through accelerometers, light sensors, microphones, and touch sensors.

While no Scratch interface to iPhones or iPods yet exists, these sensor-rich input devices have tremendous potential as expressive interfaces to musical live coding and performance. We see our work in integrating IchiBoards into *Sound Thinking* Version 2 as an initial step in providing these benefits.

7. ACKNOWLEDGMENTS

The work described in this paper is based upon work supported by the National Science Foundation under Grant No. 0722161, "CPATH CB: Performamatics: Connecting Computer Science to the Performing, Fine, and Design Arts" and a complementary Research Experience for Undergraduates (REU) supplement. *Principal Investigator:* Jesse M. Heines. *Co-Principal Investigators:* Fred G. Martin, Gena Greher, Jim Jeffers, and Karen Roehr. *Senior Personnel:* Sarah Kuhn and Nancy Selleck. *Student Researchers:* Paul Laidler and Charles Saulters II. Additional information on the Performamatics project can be found at www.performamatics.org.

Alex Ruthmann's adaptation of musical live coding in Scratch is based on his collaboration with Andrew R. Brown and Andrew C. Sorensen at the Queensland Univ. of Tech., Brisbane, Australia.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Brown, A.R. (2005). *Making Music with Java*. Brisbane, Australia: Lulu.com.

[2] Brown, A.R., & Sorensen, A.C. (2009). Interacting with generative music through live coding. *Contemporary Music Review* **28**(1):17-29.

[3] Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Jrnl. of Computing Sciences in Colleges* **15**(5):107-116.

[4] Engaging Computing Group (2009). *IchiBoard*. www.cs.uml.edu/ecg/index.php/IchiBoard/IchiBoard *accessed* Nov. 16, 2009.

[5] Greher, G.R., & Heines, J.M. (2008). Connecting Computer Science and Music Students to the Benefit of Both. *Assoc. for Technology in Music Instruction (ATMI) 2008 Conf.* Atlanta, GA.

[6] Guzdial, M. (2003). A media computation course for non-majors. *SIGCSE Bulletin* **35**(3):104-108.

[7] Heines, J.M., Goldman, K.J., Jeffers, J., Fox, E.A., & Beck, R. (2008). Interdisciplinary approaches to revitalizing undergraduate computing education. *Jrnl. of Computing in Small Colleges* **23**(5):68-72.

[8] Heines, J.M., Jeffers, J., & Kuhn, S. (2008). Performamatics: Experiences With Connecting a Computer Science Course to a Design Arts Course. *The Int'l. Jrnl. of Learning* **15**(2):9-16.

[9] Heines, J.M., Greher, G.R., & Kuhn, S. (2009). Music Performamatics: Interdisciplinary Interaction. *Proc. of the 40th ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 478-482. Chattanooga, TN: ACM.

[10] Malan, D.J., & Leitner, H.H. (2007). Scratch for budding computer scientists. *Proc. of the 38th ACM SIGCSE Technical Symposium on Computer Science Education.* Covington, Kentucky, USA: ACM.

[11] Maloney, J., *et al.* (2004). Scratch: A Sneak Preview. *Second Int'l. Conf. on Creating, Connecting and Collaborating through Computing (C5'04)*, pp. 104-109. Kyoto, Japan.

[12] Martin, F., *et al.* (2009). Joining Computing and the Arts at a Mid-Size University. *Jrnl. of Computing Sciences in Colleges* **24**(6):87-94.

[13] Sorensen, A.C., & Brown, A.R. (2007). aa-cell in practice: An approach to musical live coding. *Proc. of the Int'l. Computer Music Conf.* Copenhagen, Denmark.

[14] Wang, G., & Cook, P.R. (2004). On-the-fly programming: using code as an expressive musical instrument. *Proc. of the 2004 Conf. on New Interfaces for Musical Expression*, pp. 138-143. Hamamatsu, Shizuoka, Japan: National Univ. of Singapore.

[15] Yanco, H.A., Kim, H.J., Martin, F., & Silka, L. (2007). Artbotics: Combining Art and Robotics to Broaden Participation in Computing. *Proc. of the AAAI Spring Symposium on Robots & Robot Venues*. Stanford Univ., CA.

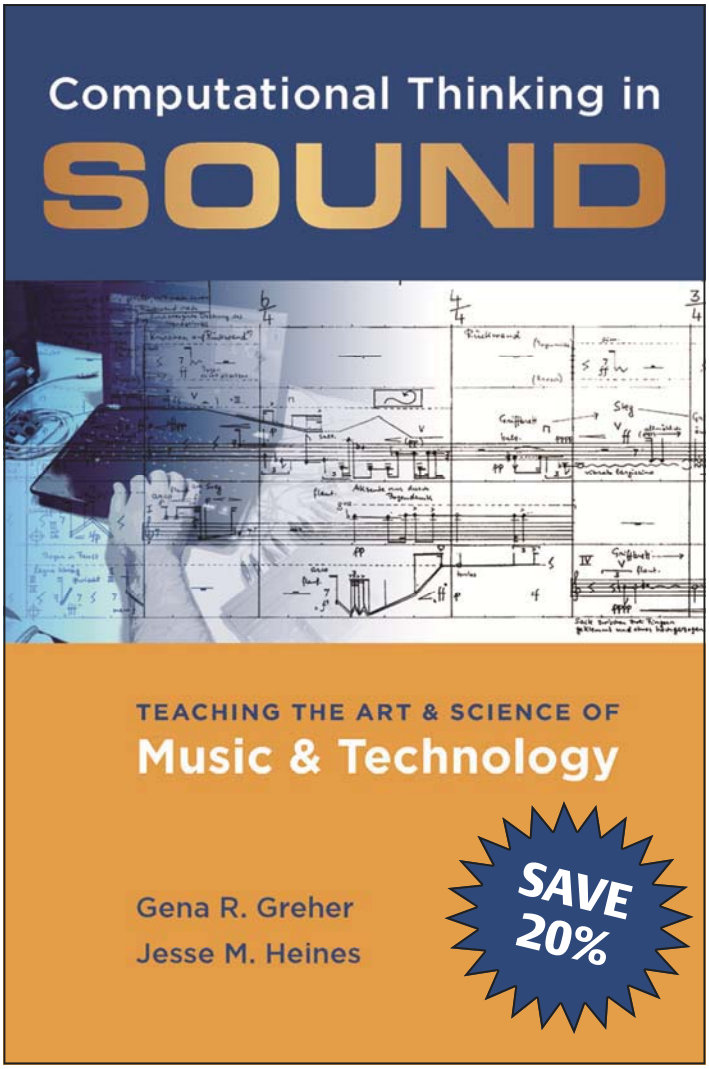
NEW FROM OXFORD

COMPUTATIONAL THINKING IN SOUND

TEACHING THE ART AND SCIENCE OF MUSIC AND TECHNOLOGY

By Gena R. Greher and Jesse M. Heines

With *Computational Thinking in Sound*, veteran educators Gena R. Greher and Jesse M. Heines provide the first book ever written for music fundamentals educators that is devoted specifically to music, sound, and technology. Using a student-centered approach that emphasizes project-based experiences, the book provides music educators with multiple strategies to explore, create, and solve problems with music and technology in equal parts. It also provides examples of hands-on activities that encourage students, alone and in groups, to explore the basic principles that underlie today's music technology and freely available multimedia creation tools. *Computational Thinking in Sound* is an effective tool for educators to introduce students to the complex process of computational thinking in the context of the creative arts through the more accessible medium of music.



March 2014

Hardcover | 272 pp. | ~~$35.00~~ **$28.00**

978-0-199-826190-3

Gena R. Greher is a Professor of Music Education at UMass Lowell. Her research focuses on creativity and listening skill development in children and examining the influence of integrating multimedia technology in urban music classrooms and music teacher education through School-University partnerships.

OXFORD UNIVERSITY PRESS

Jesse M. Heines is a Professor of Computer Science at UMass Lowell. His primary teaching responsibilities include object-oriented programming and graphical user interfaces. His research focuses on computer science education, particularly interdisciplinary approaches that blend computer science with music and other fields to enhance instructional effectiveness in both.

Order online at **www.oup.com/us**. Enter promotion code **28862** to save **20%**

CREATE YOUR WORLD
SCRATCH AT MIT 2012

Making Music with Scratch

a workshop presented at

Scratch@MIT 2012

Friday, July 27, 2012

Jesse M. Heines

Dept. of Computer Science
Jesse_Heines@uml.edu

Gena R. Greher

Dept. of Music
Gena_Greher@uml.edu

University of Massachusetts Lowell

www.performamatics.org

PERFORMAMATICS

SCRATCH

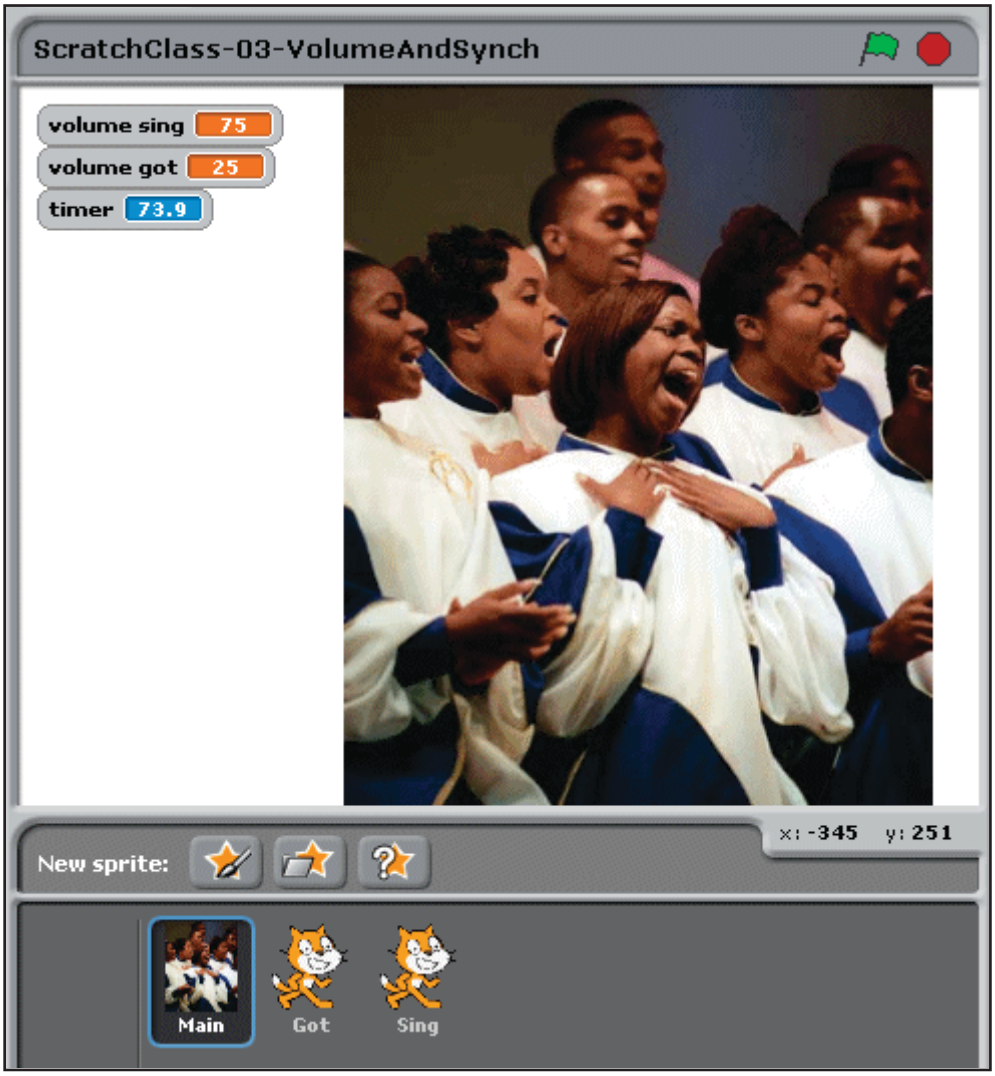
Copyright Notice

A full color version of this handout in PDF format may be downloaded from:

http://bit.ly/scratch2012music

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Playing and Synchronizing MIDI Files



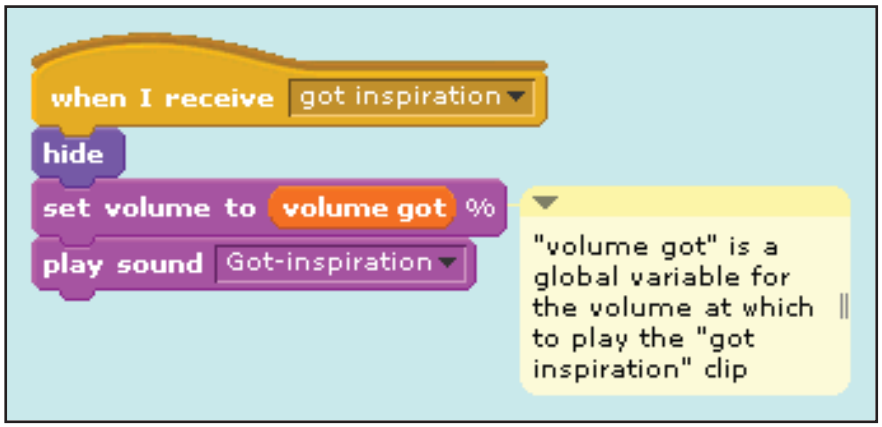
Volume and Synchronization Concepts

- use of variables when setting the volume
- local vs. global attributes, specifically volume
- use of a control script and broadcasts
- use of the Scratch timer for synchronization

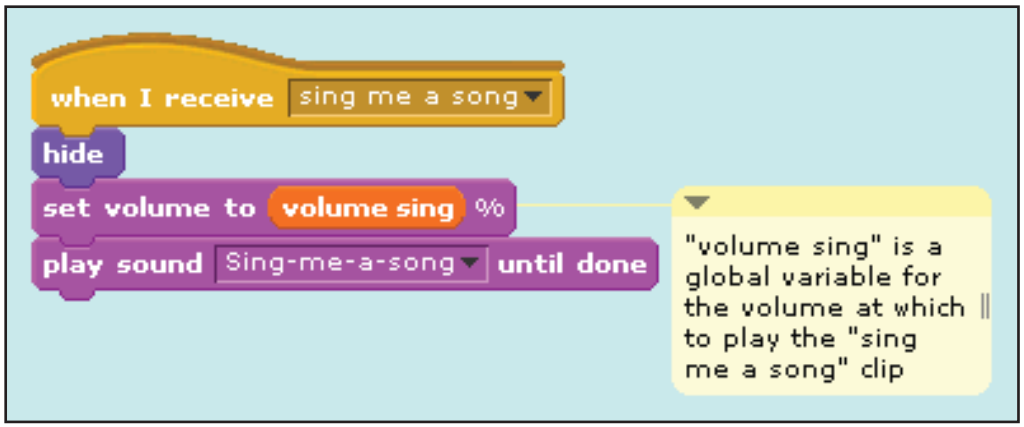
Playing and Synching MIDI Files (cont'd)

MP3 Player Scripts

Script in Sprite "Got"



Script in Sprite "Sing"



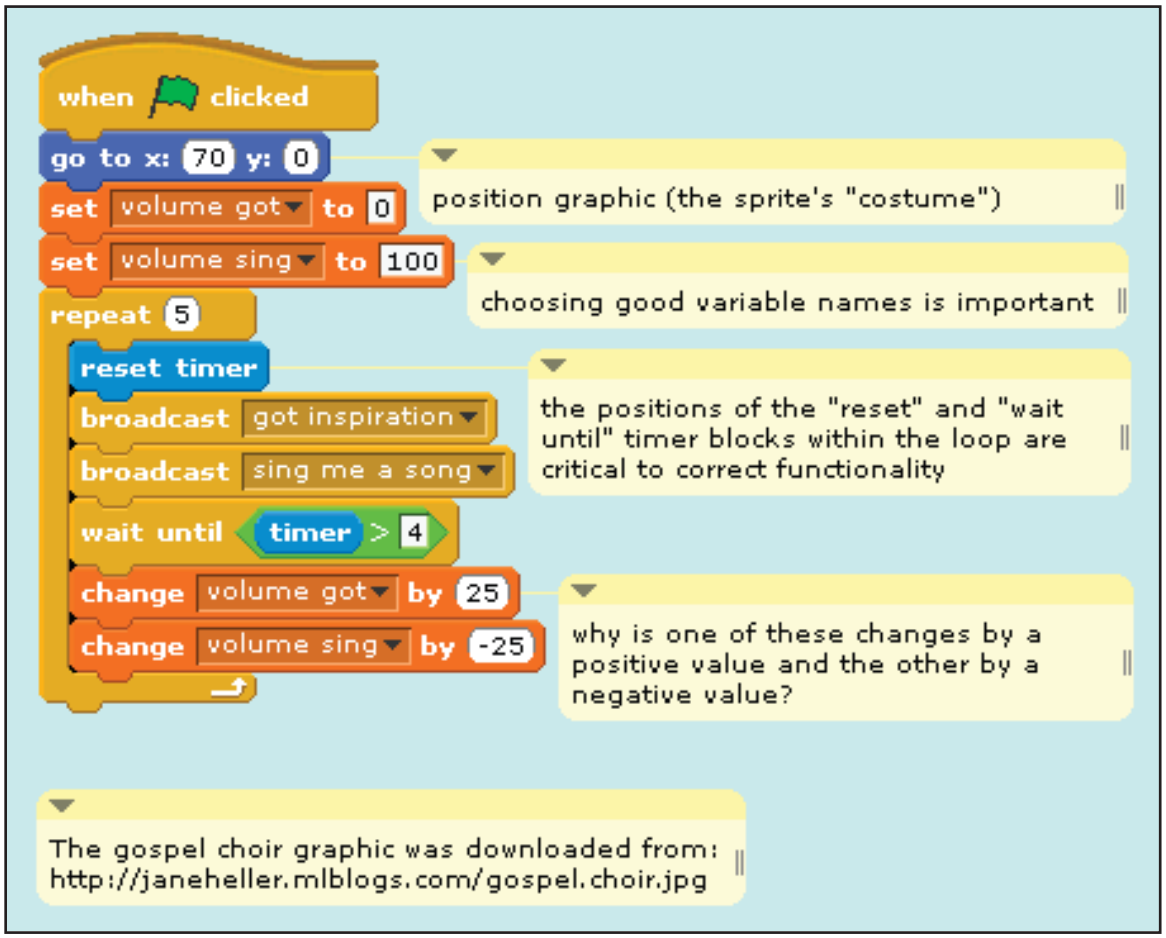
Each script must be in its own sprite to allow volume to be controlled independently.

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Playing and Synching MIDI Files (cont'd)

Control Script

Script in Sprite "Main"

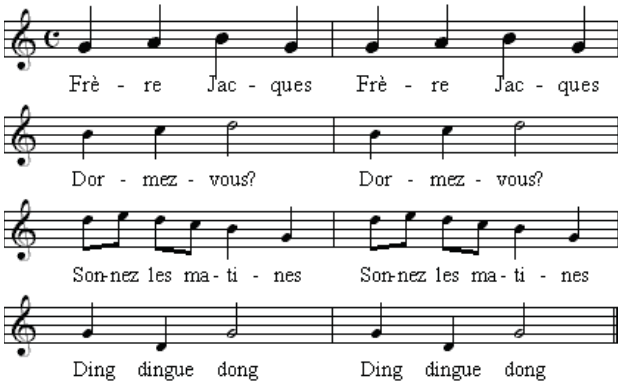


Note the order of the blocks and the critical position of the **change** blocks. Changing the volume parameter before the **wait until** block will cause the volume to be changed while the MP3 is playing. Such behavior may be desirable in other programs, but not this one.

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Frère Jacques

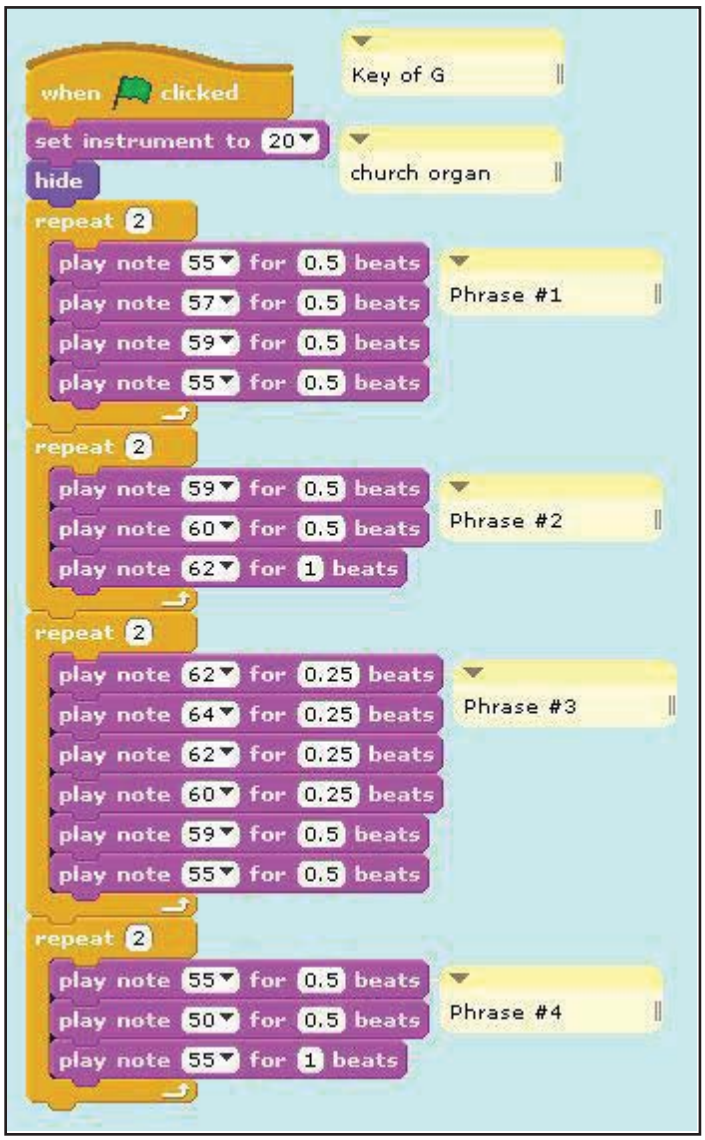
Version 1: Playing Notes

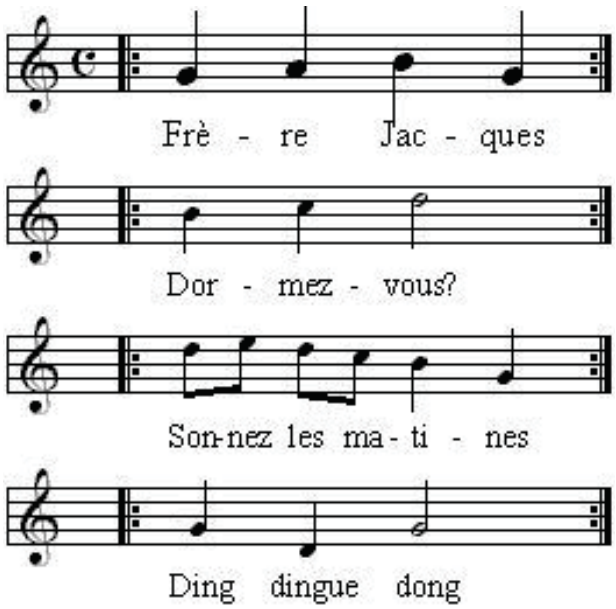


Remember Turbo Speed!

Frère Jacques

Version 2: Using Loops





Remember to set Turbo Speed to improve performance.

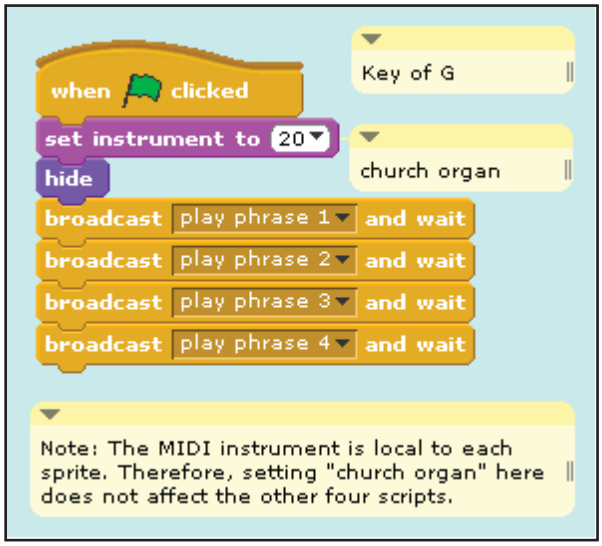
Acknowledgement: The scores on this and the previous page were adapted from www.csdraveurs.qc.ca/musique/flutalors/images/frere.gif and www.mamalisa.com/images/scores/frerejacques.jpg, respectively, but as of July 12, 2012, these URLs no longer appear to be valid.

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Frère Jacques

Version 3: Separating Phrases

Main Script



Phrases Scripts (4, cont'd on next page)

#1

Thought: We could set the instrument in each script, but that would contradict the **DRY** programming principle: "Don't Repeat Yourself."

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

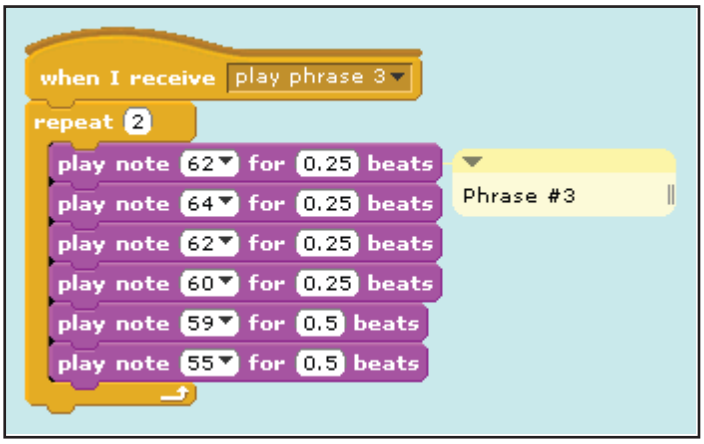
Frère Jacques

Version 3: Separating Phrases (cont'd)

Phrases Scripts (cont'd)

#2

#3



#4

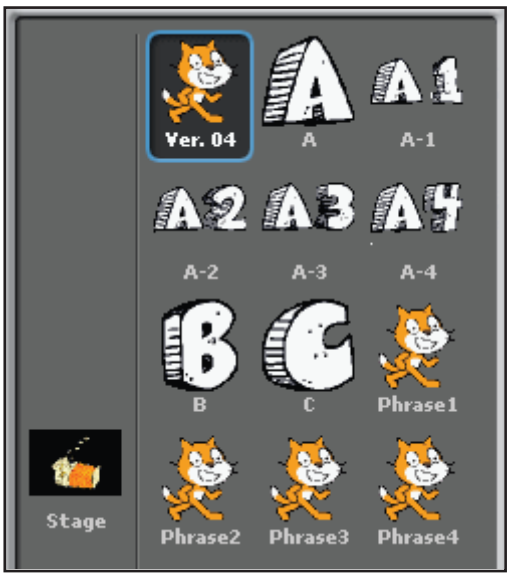
Challenge: How can we set the instrument **JUST ONCE** and have that setting apply to all scripts?

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

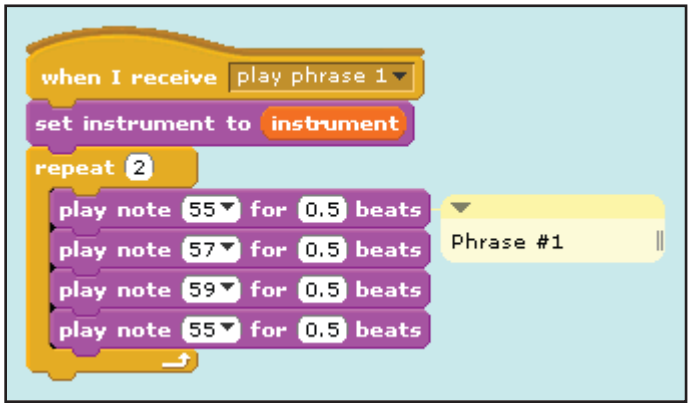
Frère Jacques

Version 4: Playing a Round

Main Script



Phrases Scripts



Note the addition of the **set instrument** block and the use of the **instrument** variable (set in the Main script) as the value to set. Other phrase scripts similarly contain this one revision.

continued on next page

Frère Jacques

Version 4: Playing a Round (cont'd)

Scripts A-1 through A-4

. . .

Others scripts are similar, differing only in **when I receive**.

Control Script A – single threaded

continued on next page

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Frère Jacques

Version 4: Playing a Round (cont'd)

Control Script B – multi-threaded

Control Script C – multi-threaded repeat

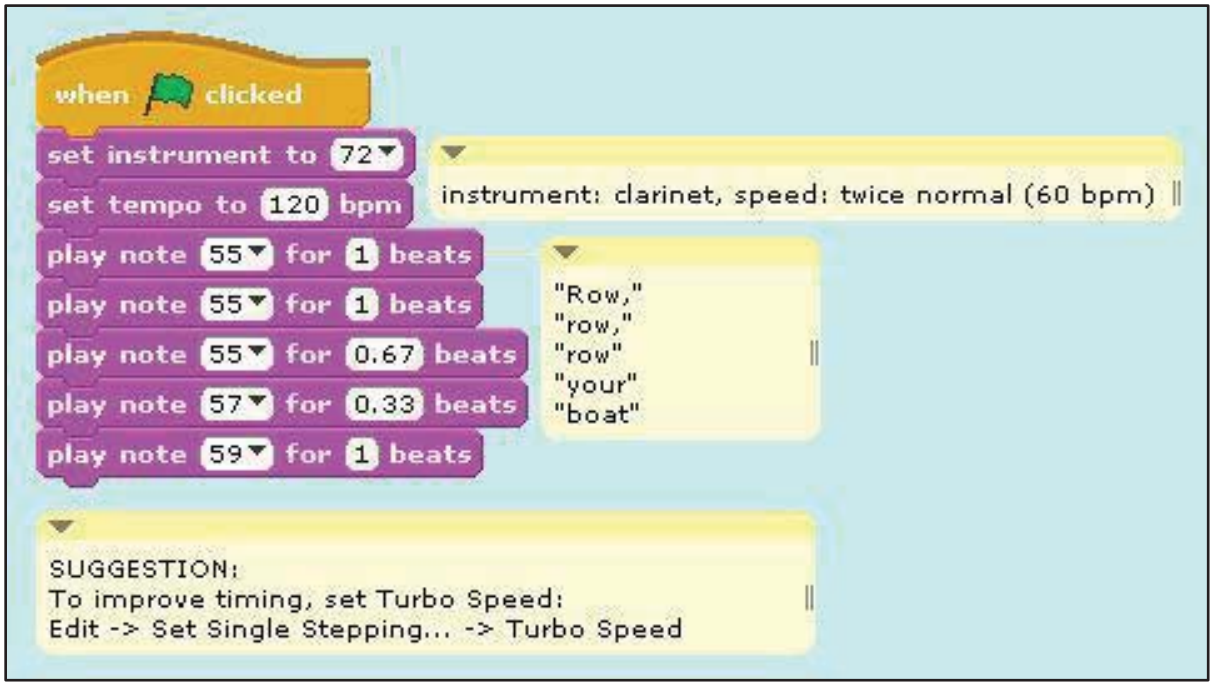
end of Version 4

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

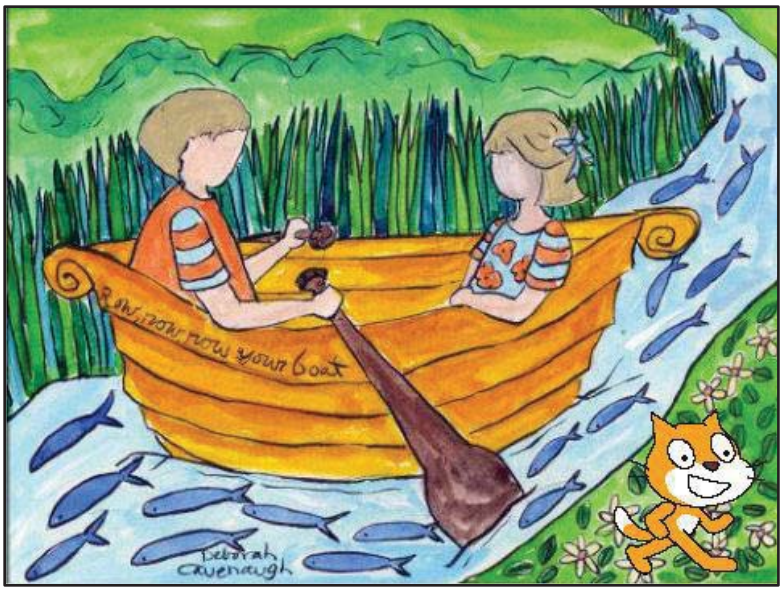
Row, Row, Row Your Boat

Version 1: Playing Notes

Single Script



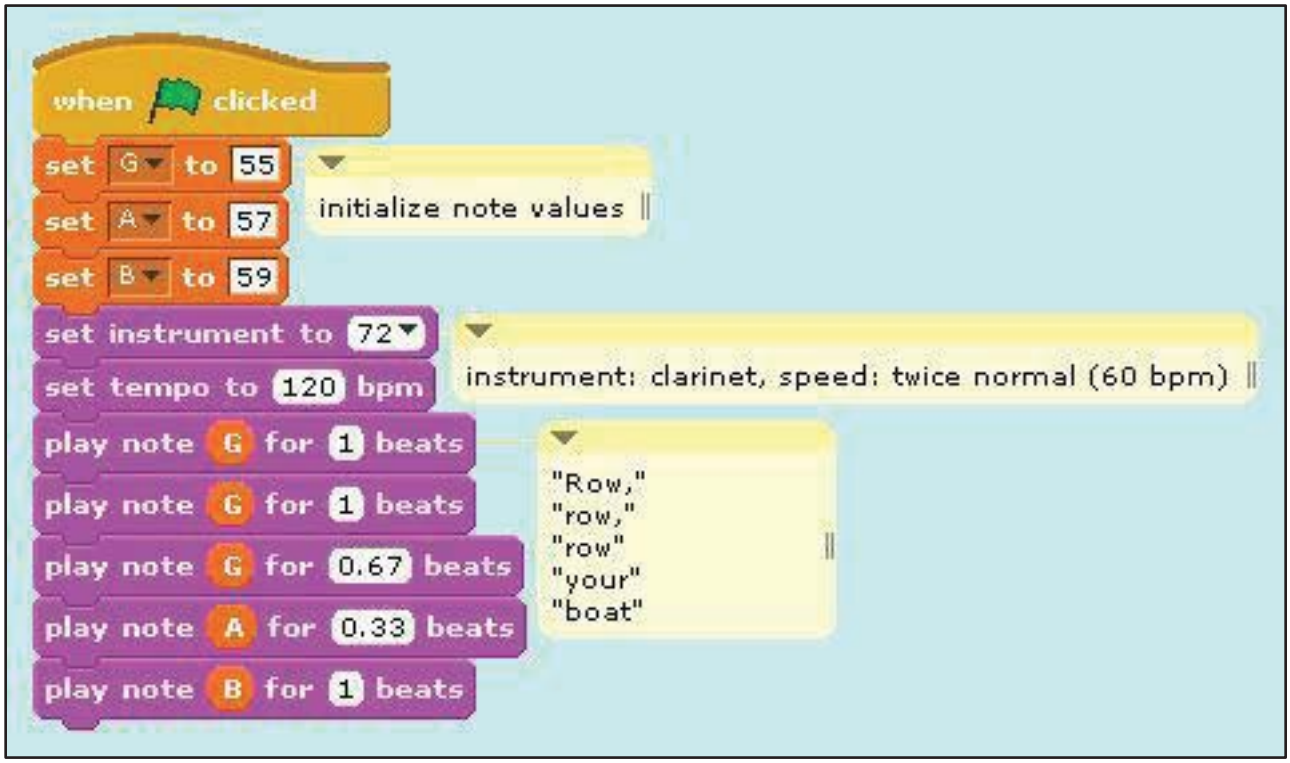
Output Window



Row, Row, Row Your Boat

Version 2: Playing Notes Using Variables

Single Script



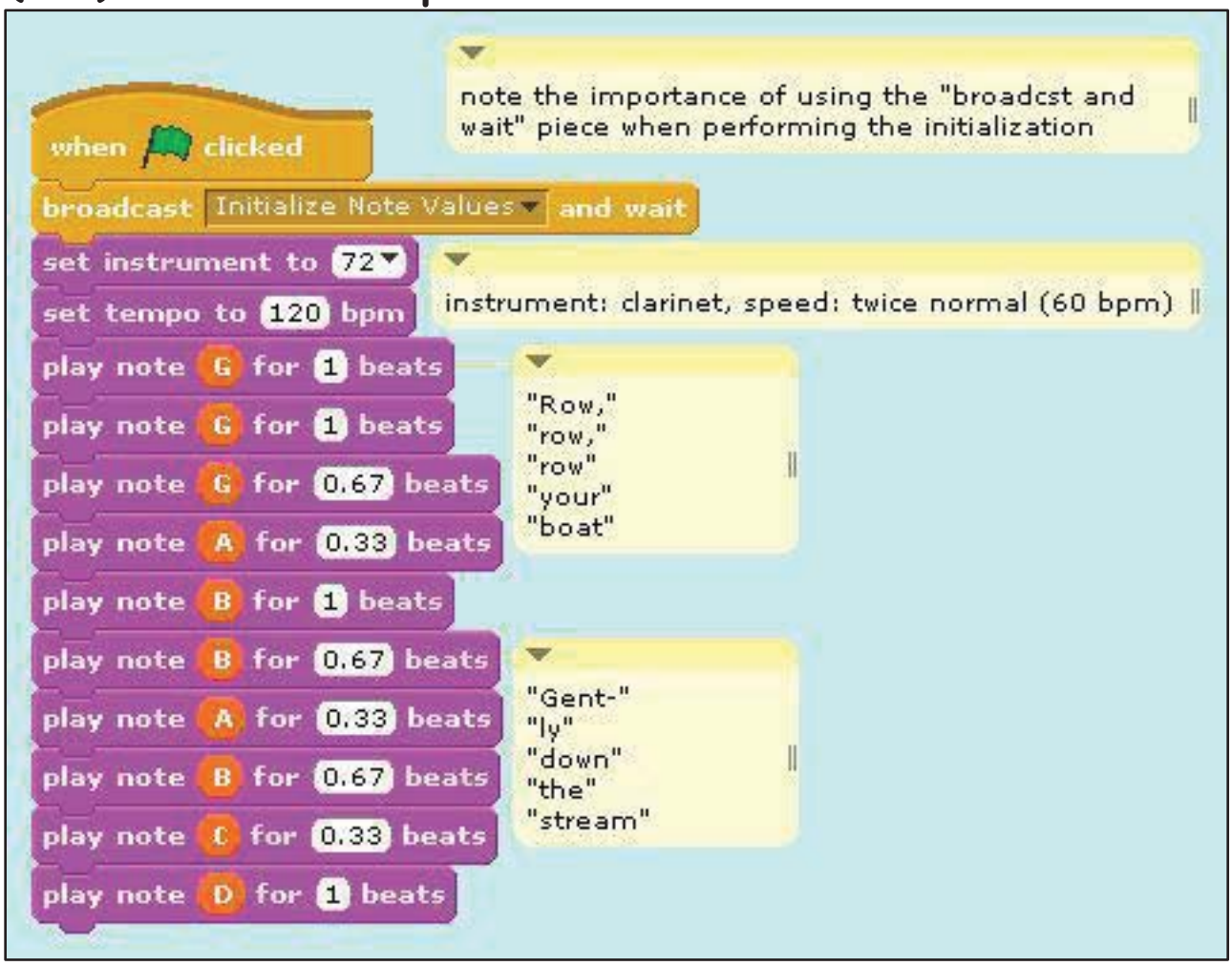
Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 3: Separating Initialization

Two Scripts

(3a) **Main** Script

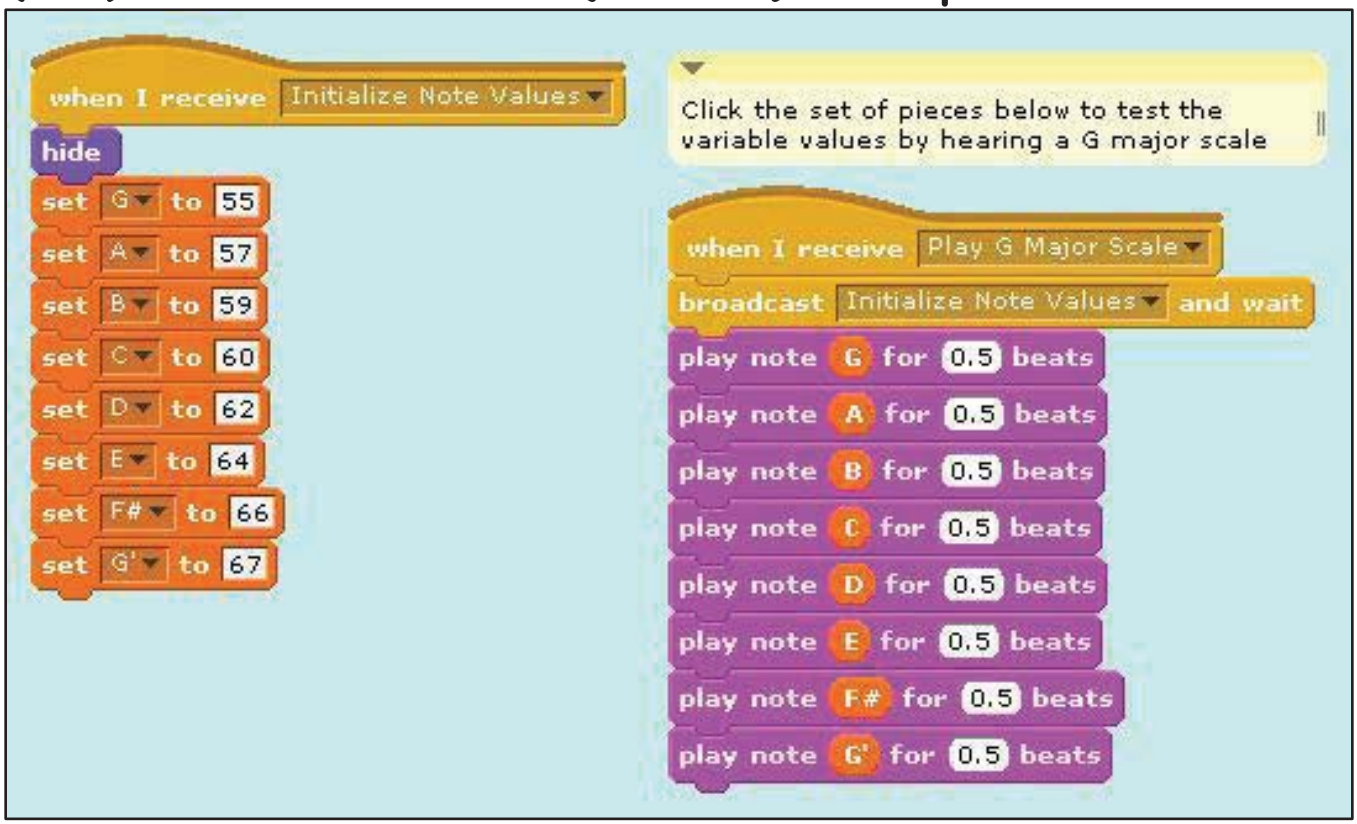


continued on next page

Row, Row, Row Your Boat

Version 3: Separating Initialization (cont'd)

(3b) **Initialization ("Init")** Script



end of Version 3

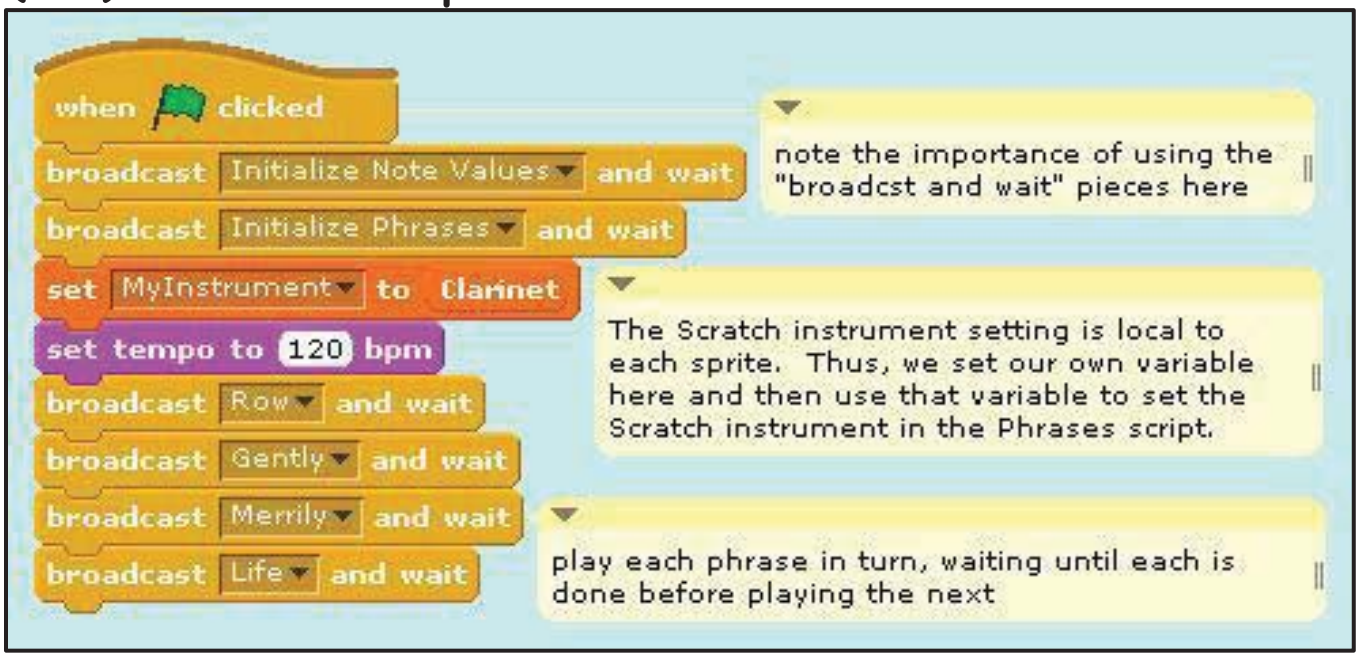
Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 4: Separating Phrases

Three Scripts

(4a) **Main** Script

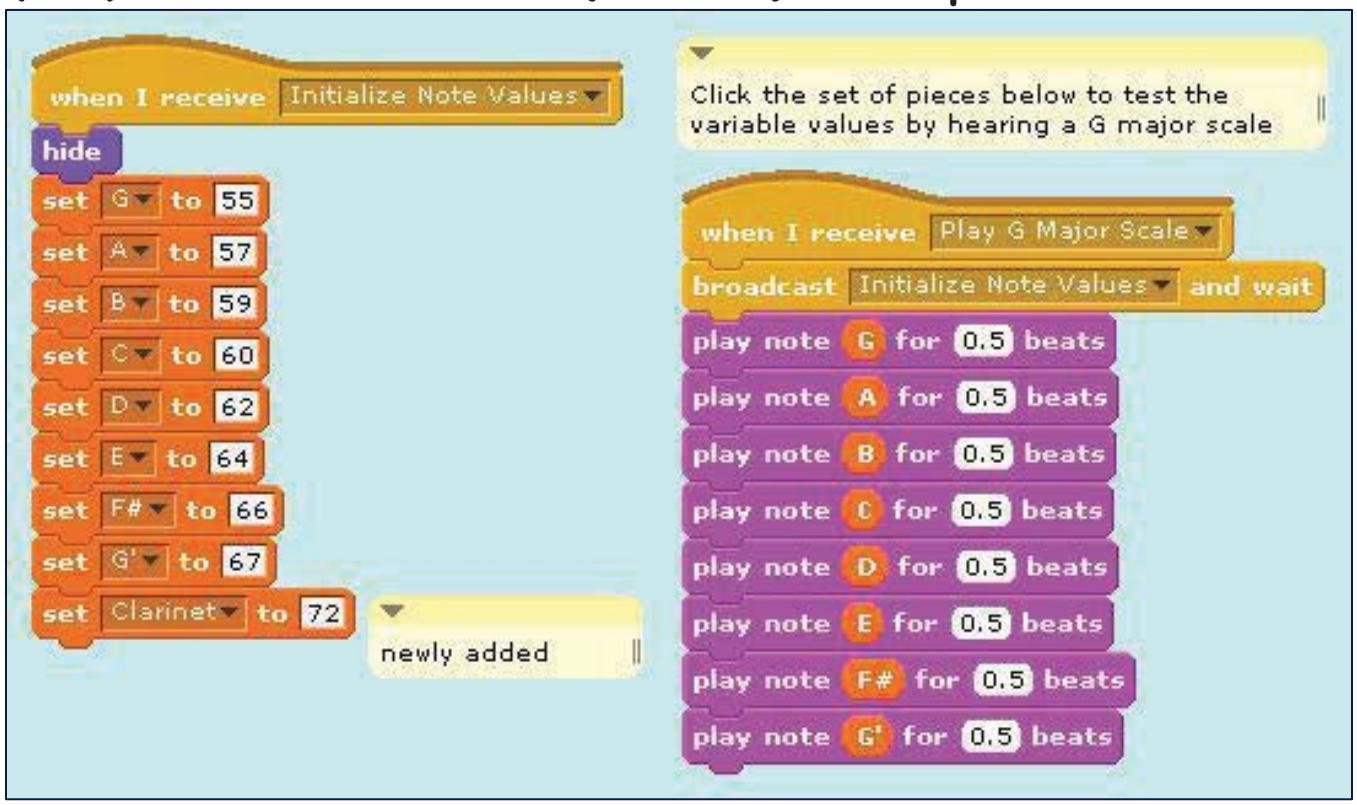


continued on next page

Row, Row, Row Your Boat

Version 4: Separating Phrases (cont'd)

(4b) **Initialization ("Init")** Script



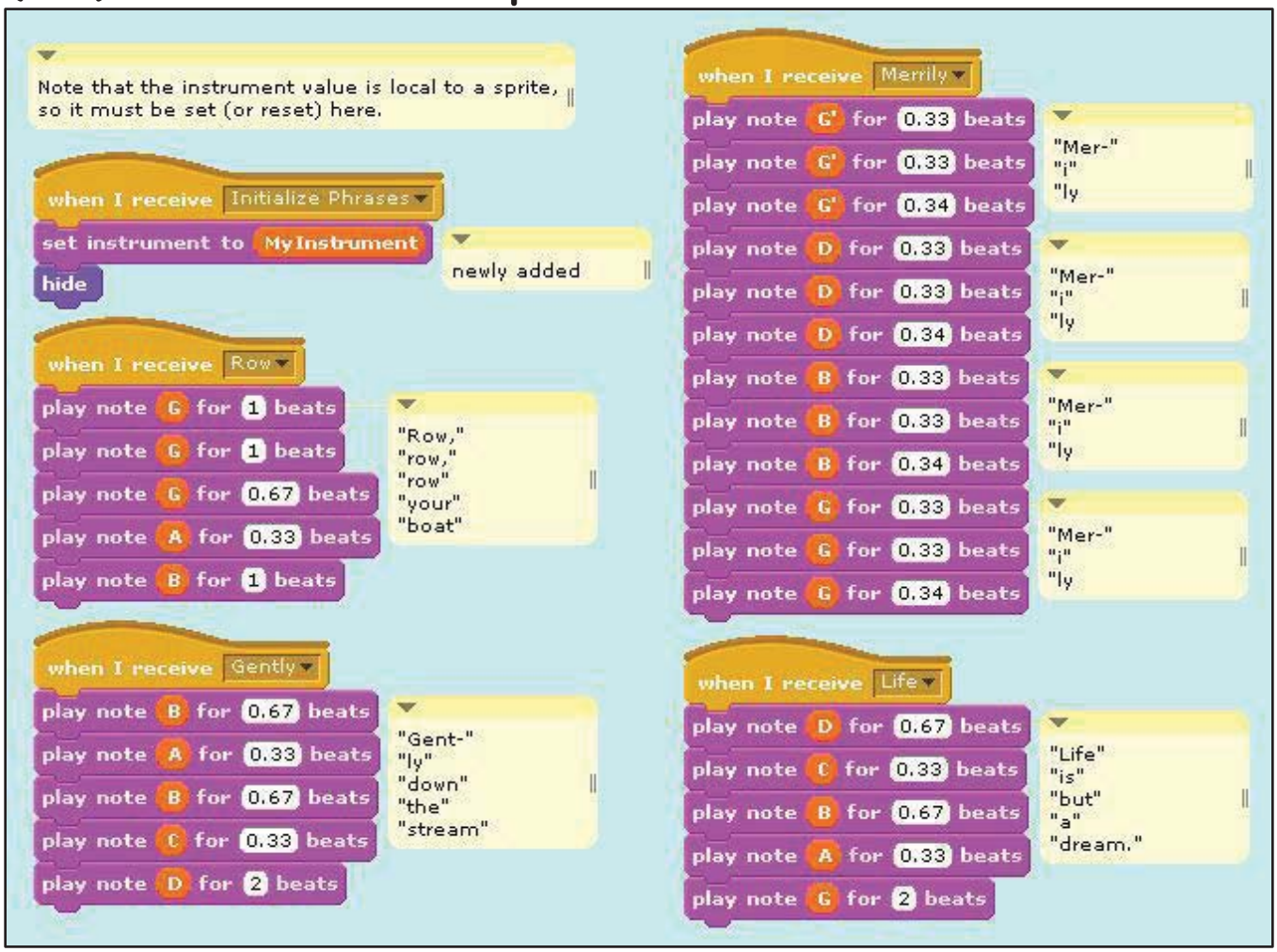
continued on next page

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 4: Separating Phrases (cont'd)

(4c) **Phrases** Script



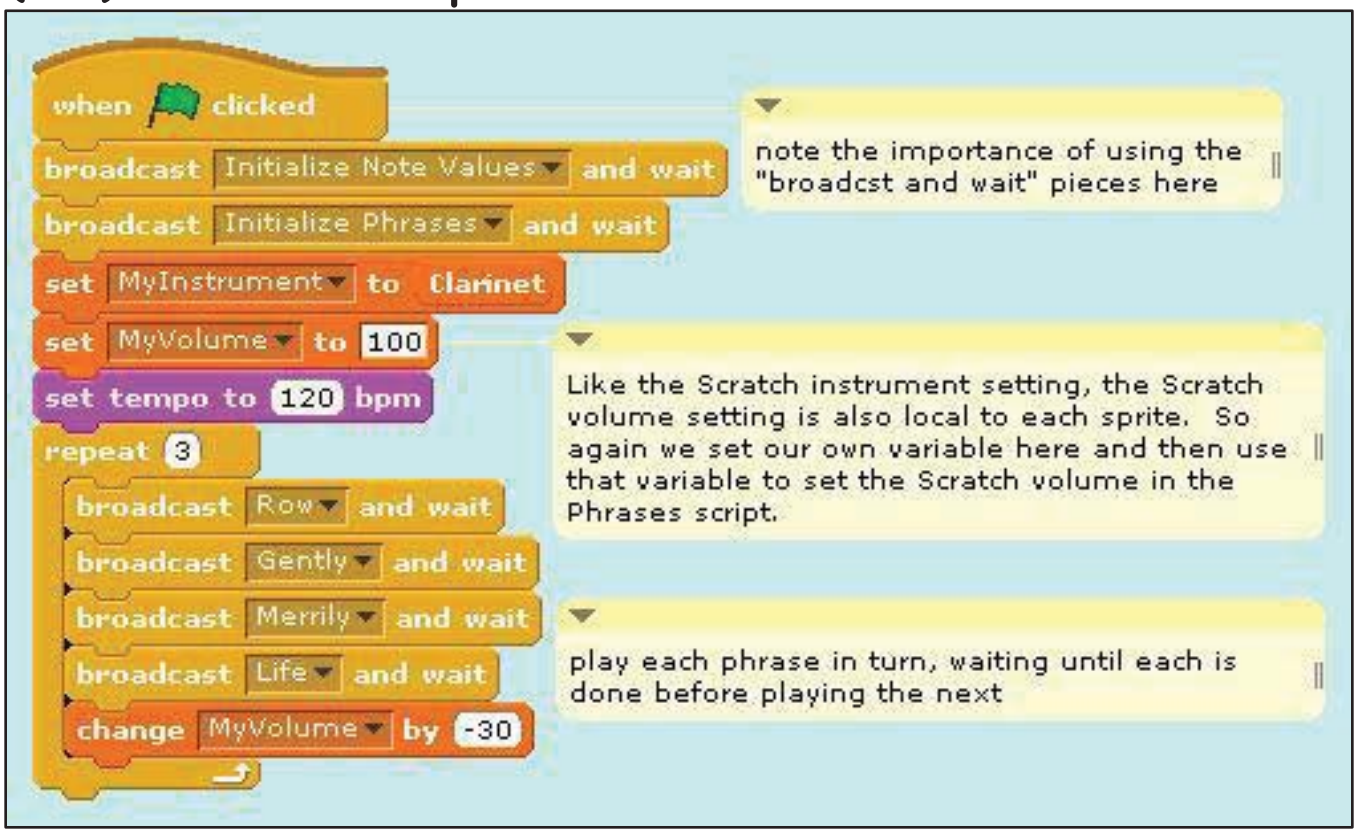
end of Version 4

Row, Row, Row Your Boat

Version 5: Looping and Fading

Three Scripts

(5a) **Main** Script



(5b) **Initialization ("Init")** Script

(*same as on page 34*)

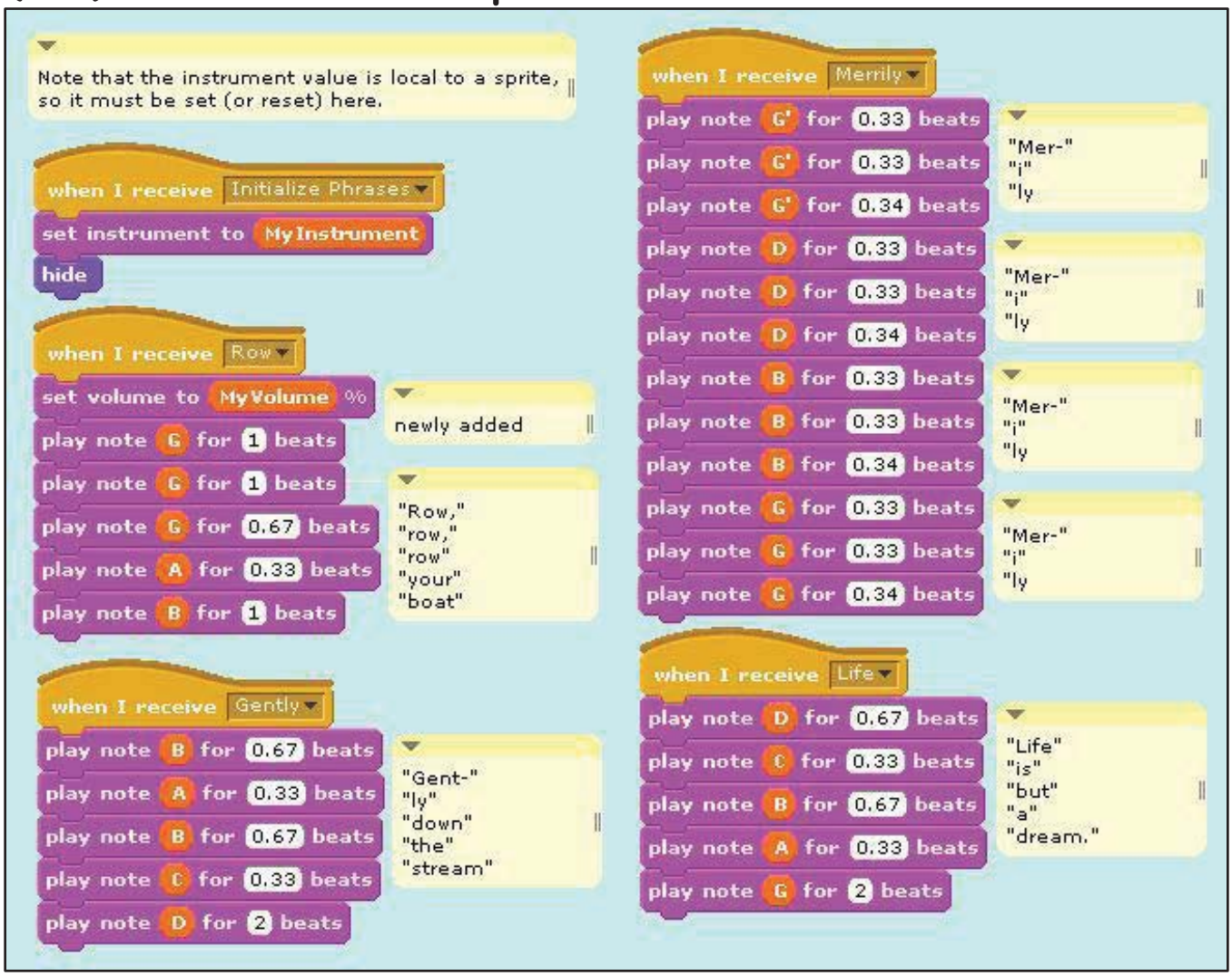
continued on next page

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 5: Looping and Fading (cont'd)

(5c) **Phrases** Script



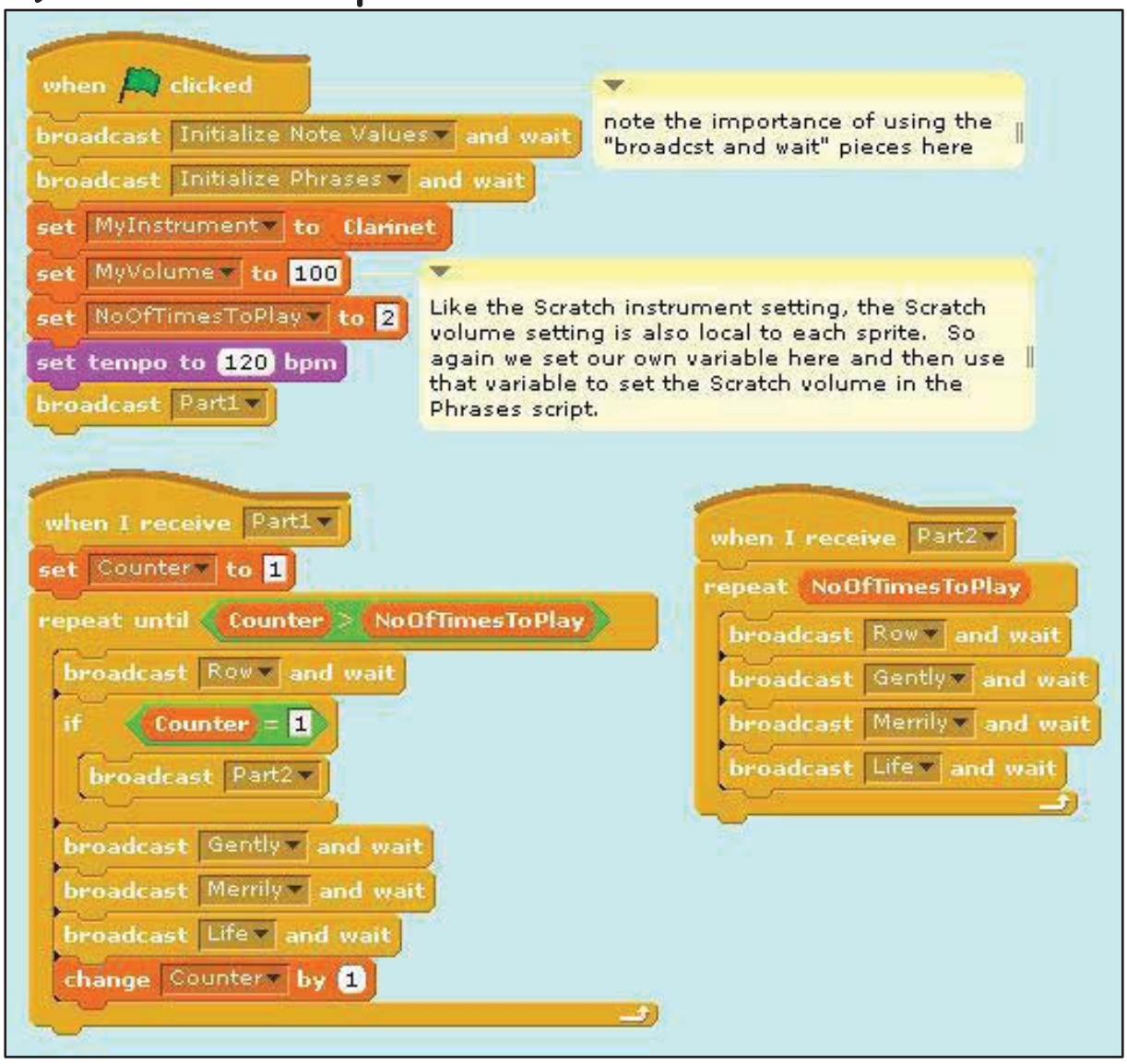
end of Version 5

Row, Row, Row Your Boat

Version 6: Playing a Round with One Instrument

Three Scripts

(6a) **Main** Script

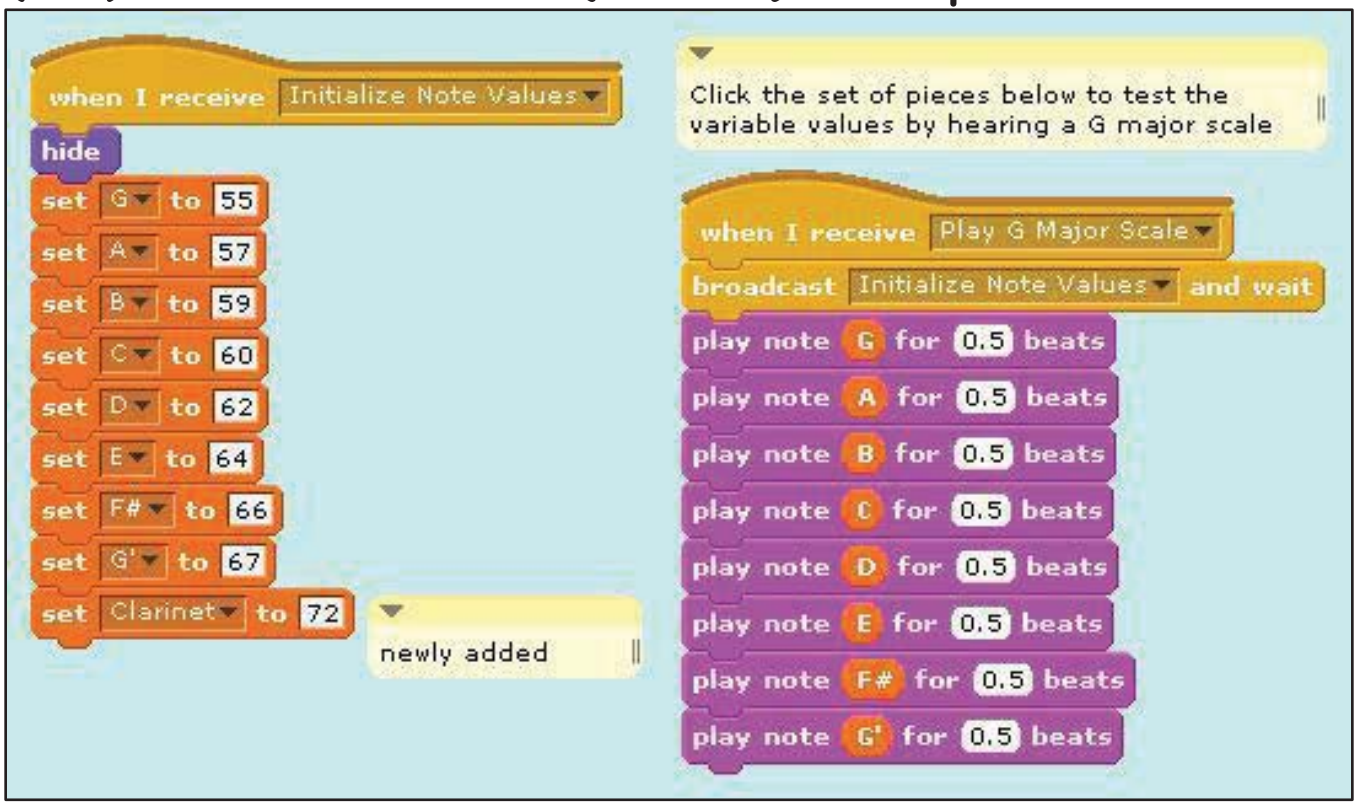


Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 6: Playing a Round with One Instrument (cont'd)

(6b) **Initialization ("Init")** Script



(6c) **Phrases** Script

(*same as on page 37*)

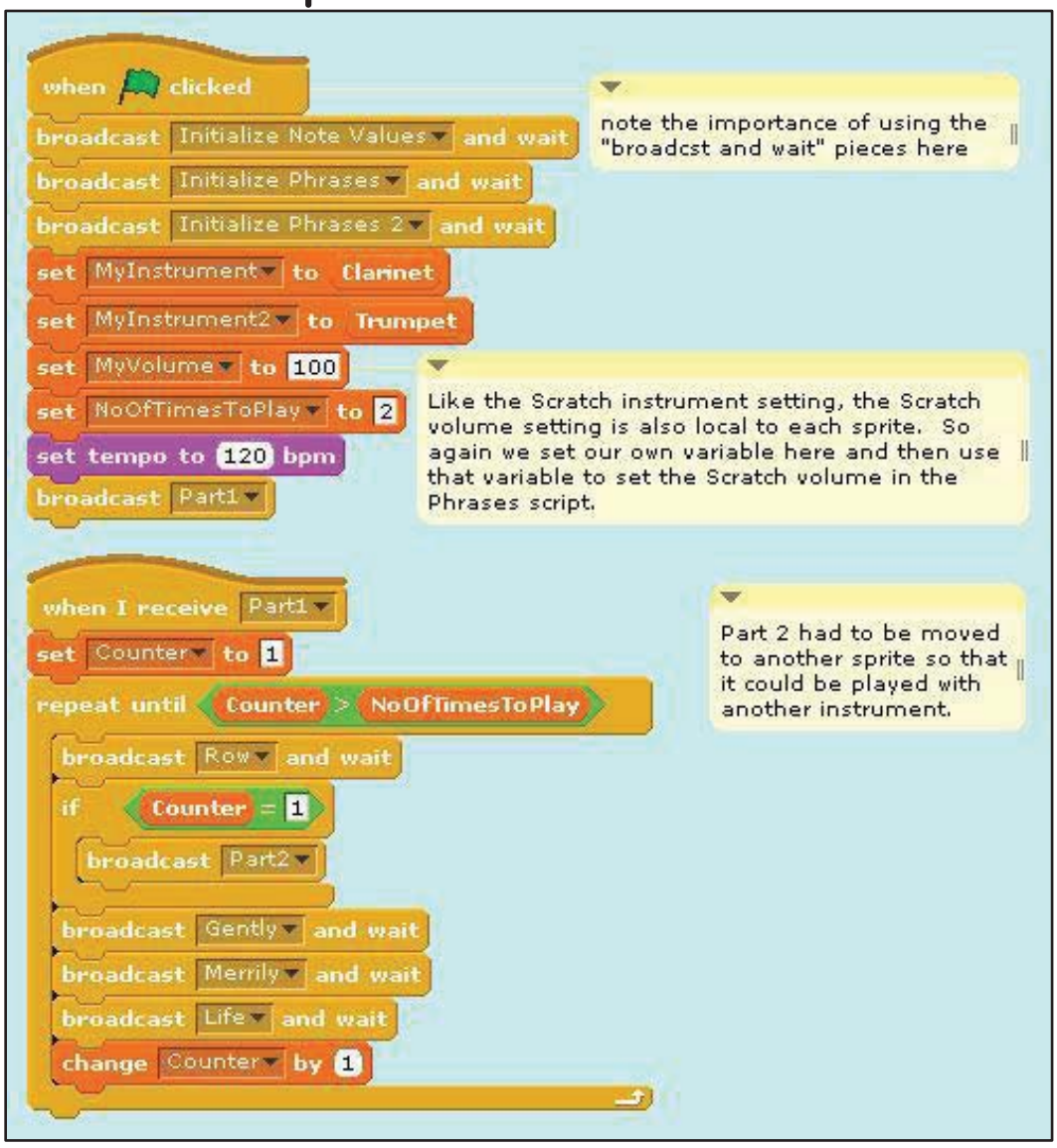
end of Version 6

Row, Row, Row Your Boat

Version 7: Playing a Round with Two Instruments

Five Scripts

(7a) **Main** Script



Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 7: Playing a Round with Two Instruments (cont'd)

(7b) **Initialization ("Init")** Script

(7c) **Phrases** Script

(*same as on page 37*)

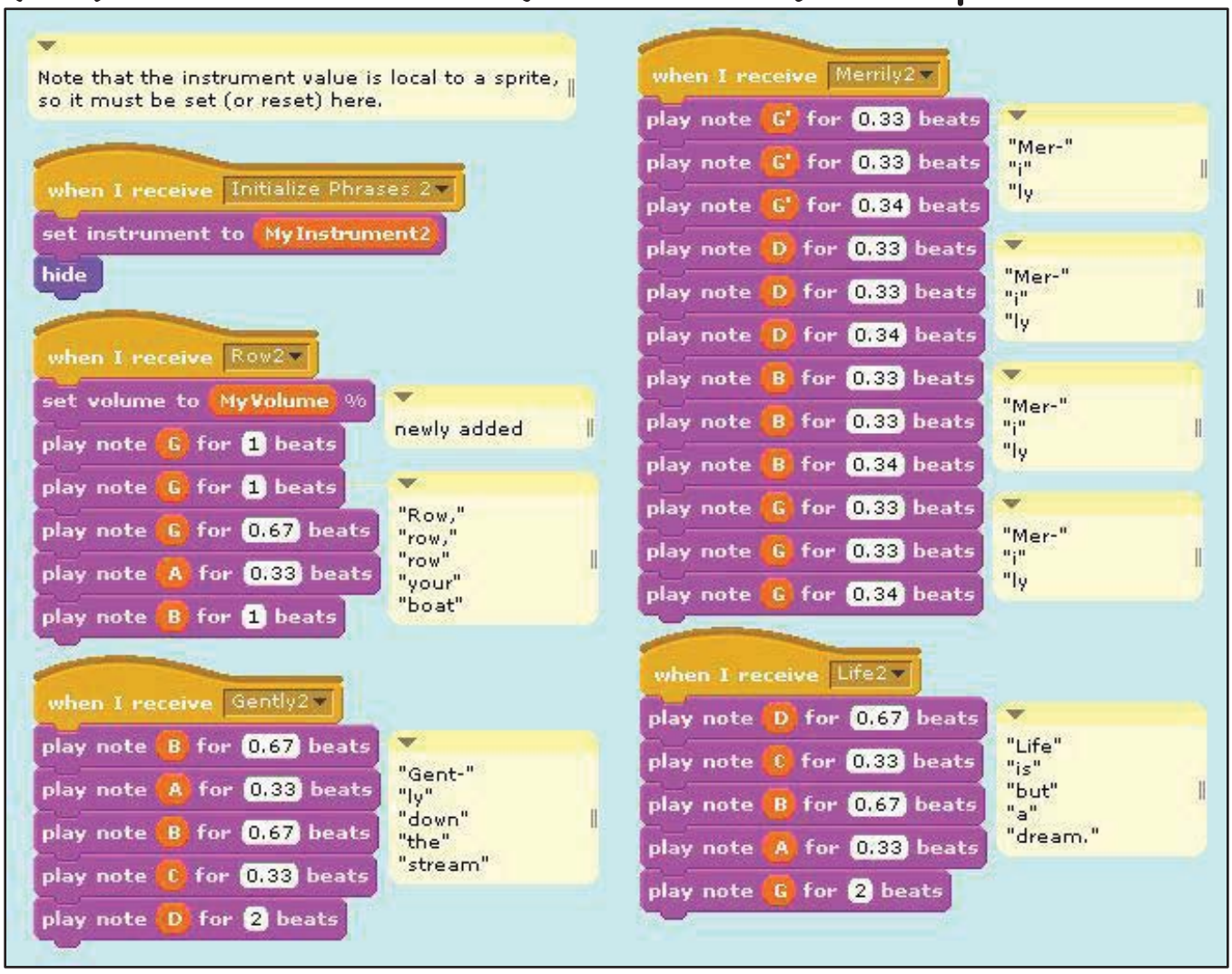
(7d) **Part2** Script ➔

continued on next page

Row, Row, Row Your Boat

Version 7: Playing a Round with Two Instruments (cont'd)

(7e) Instrument2 ("Instru2") Script



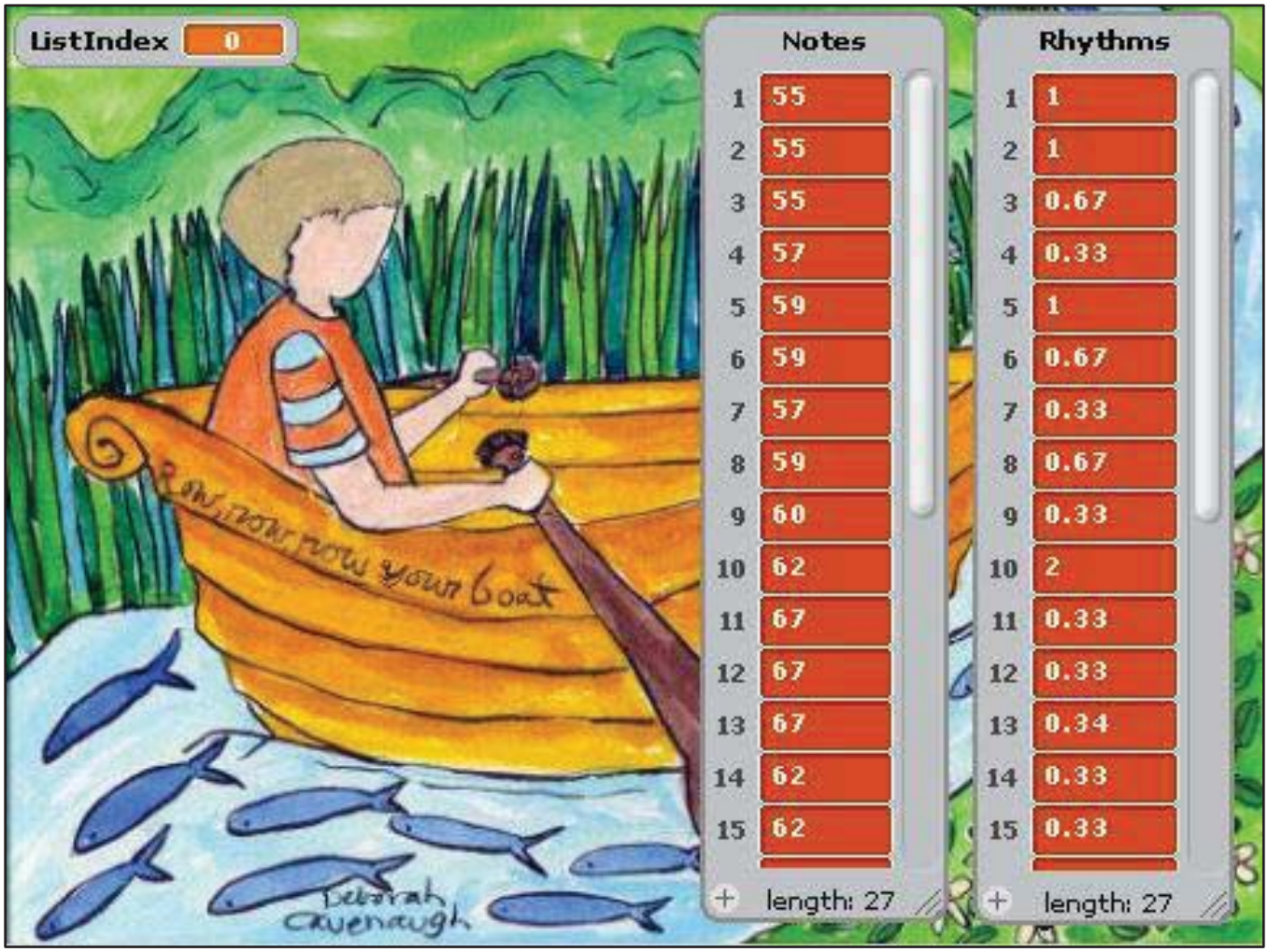
end of Version 7

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 8: Storing Notes and Rhythms in Lists

Output Window



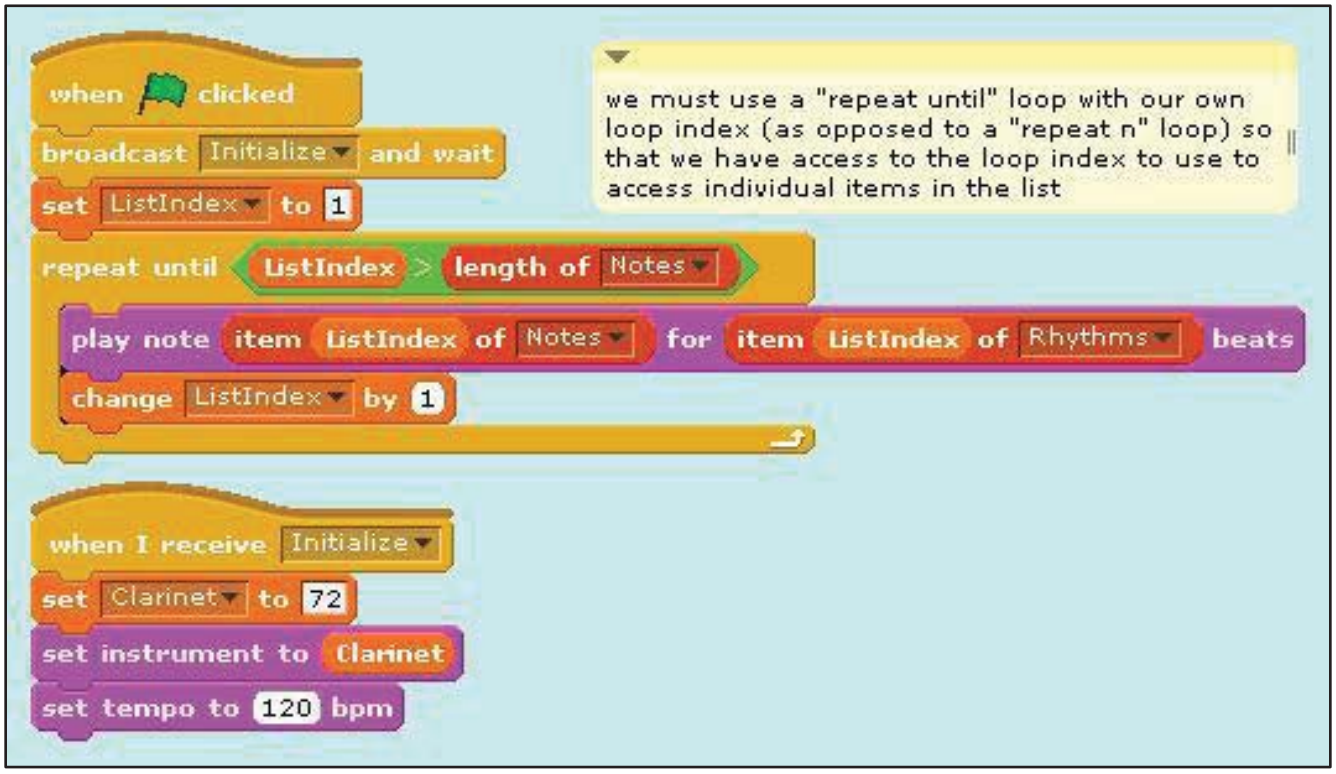
continued on next page

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 8: Storing Notes and Rhythms in Lists (cont'd)

Single Script



end of Version 8

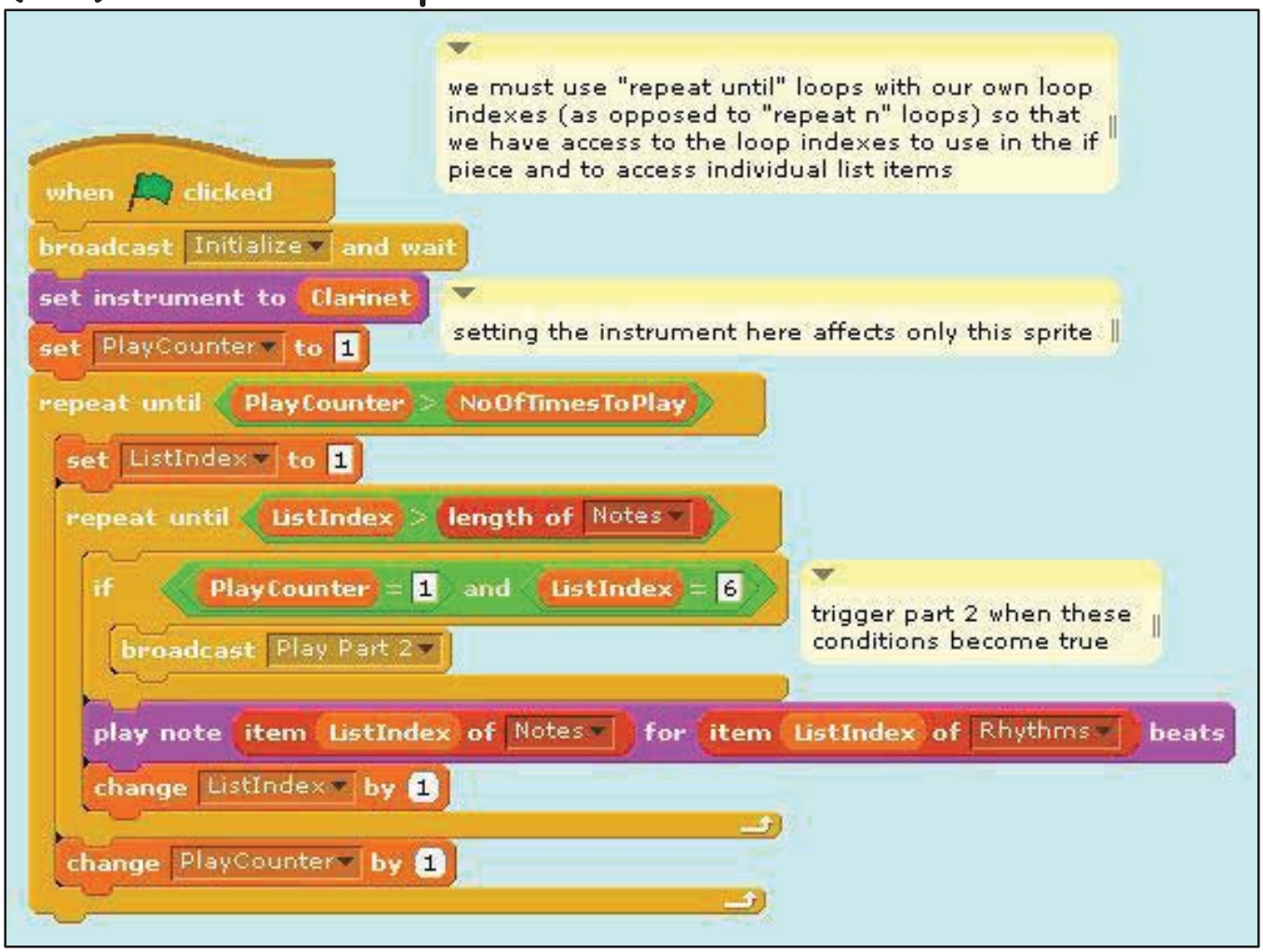
Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 9: Playing a Round Using Lists

Three Scripts

(9a) **Main** Script



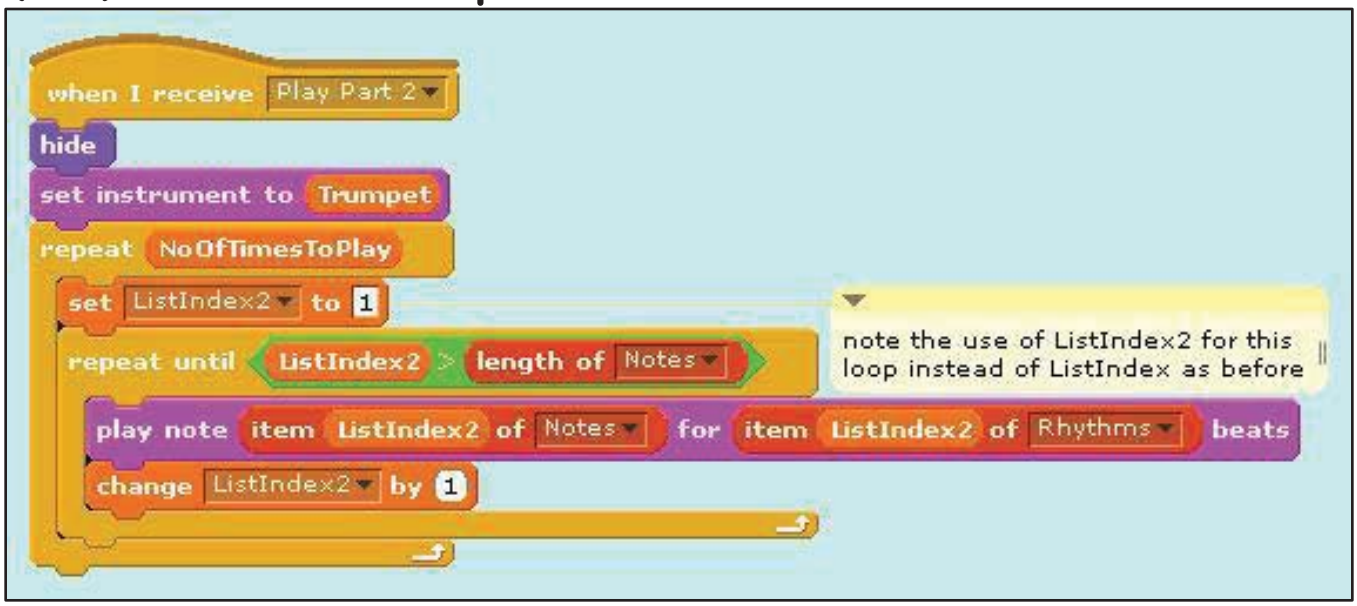
continued on next page

Row, Row, Row Your Boat

Version 9: Playing a Round Using Lists (cont'd)

(9b) **Initialization ("Init")** Script

(9c) **Part2** Script



end of Version 9

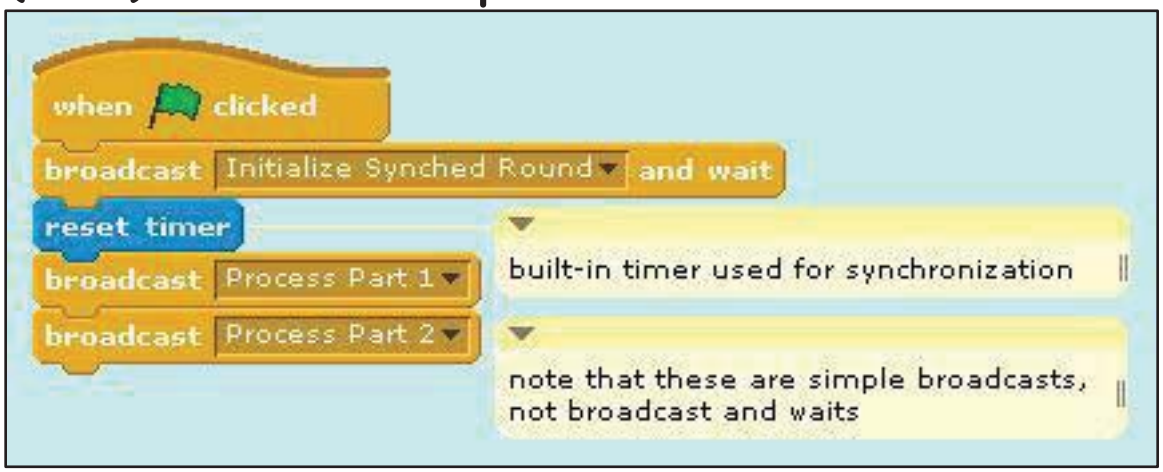
Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

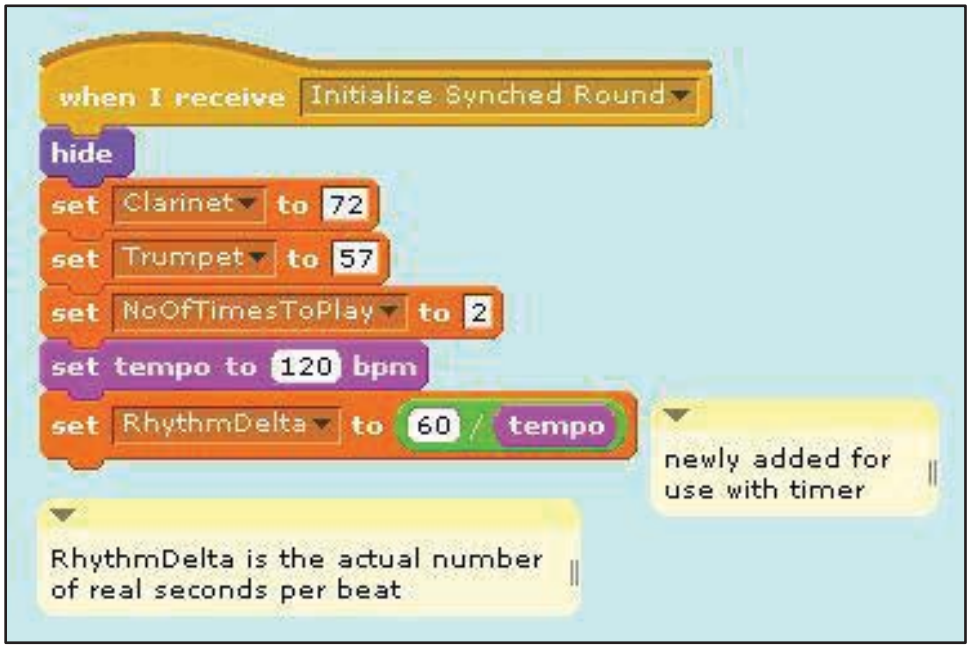
Version 10: Synchronizing Play from Lists

Four Scripts

(10a) **Main** Script



(10b) **Initialization ("Init")** Script



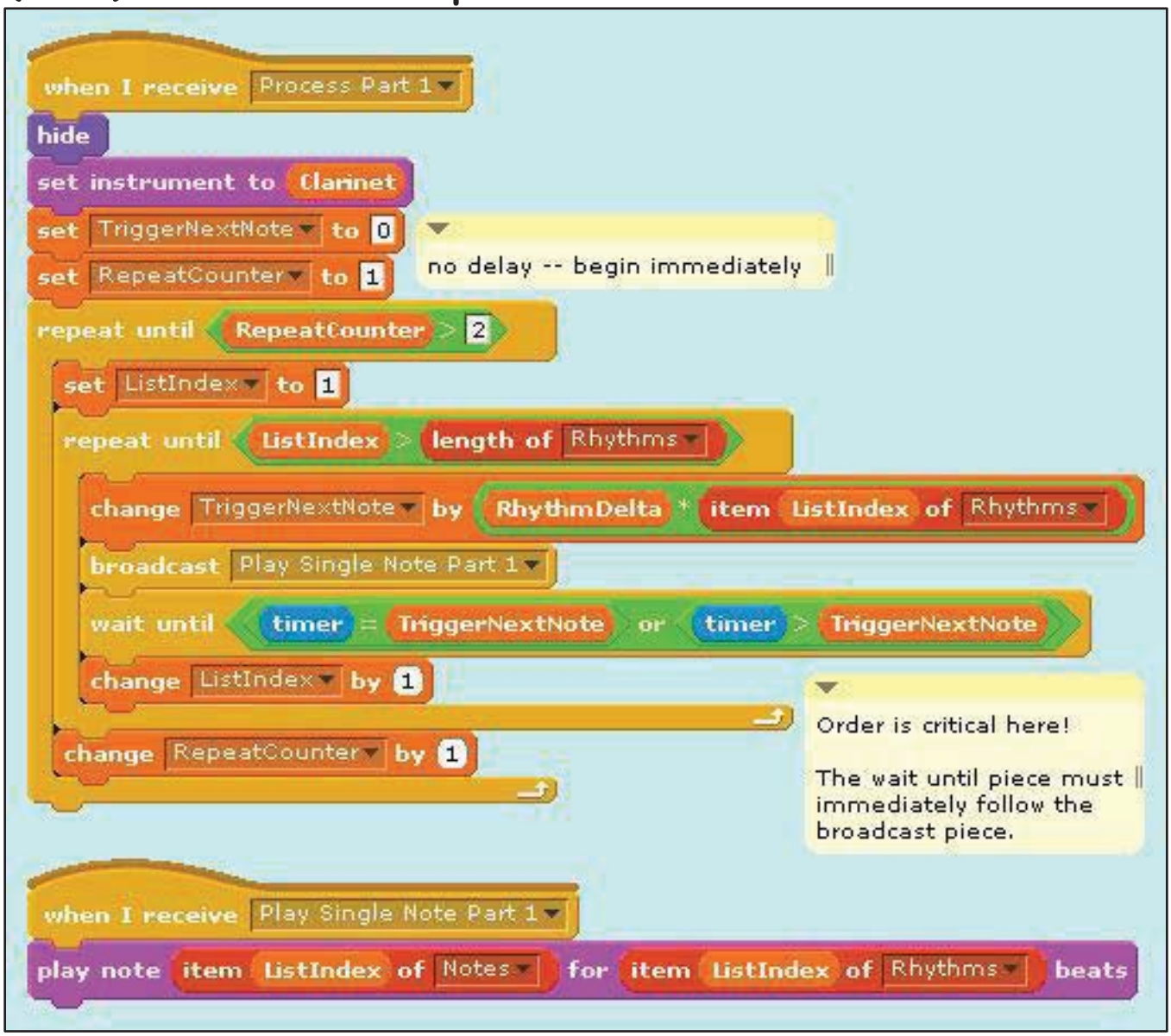
continued on next page

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 10: Synchronizing Play from Lists (cont'd)

(10c) Part 1 Script



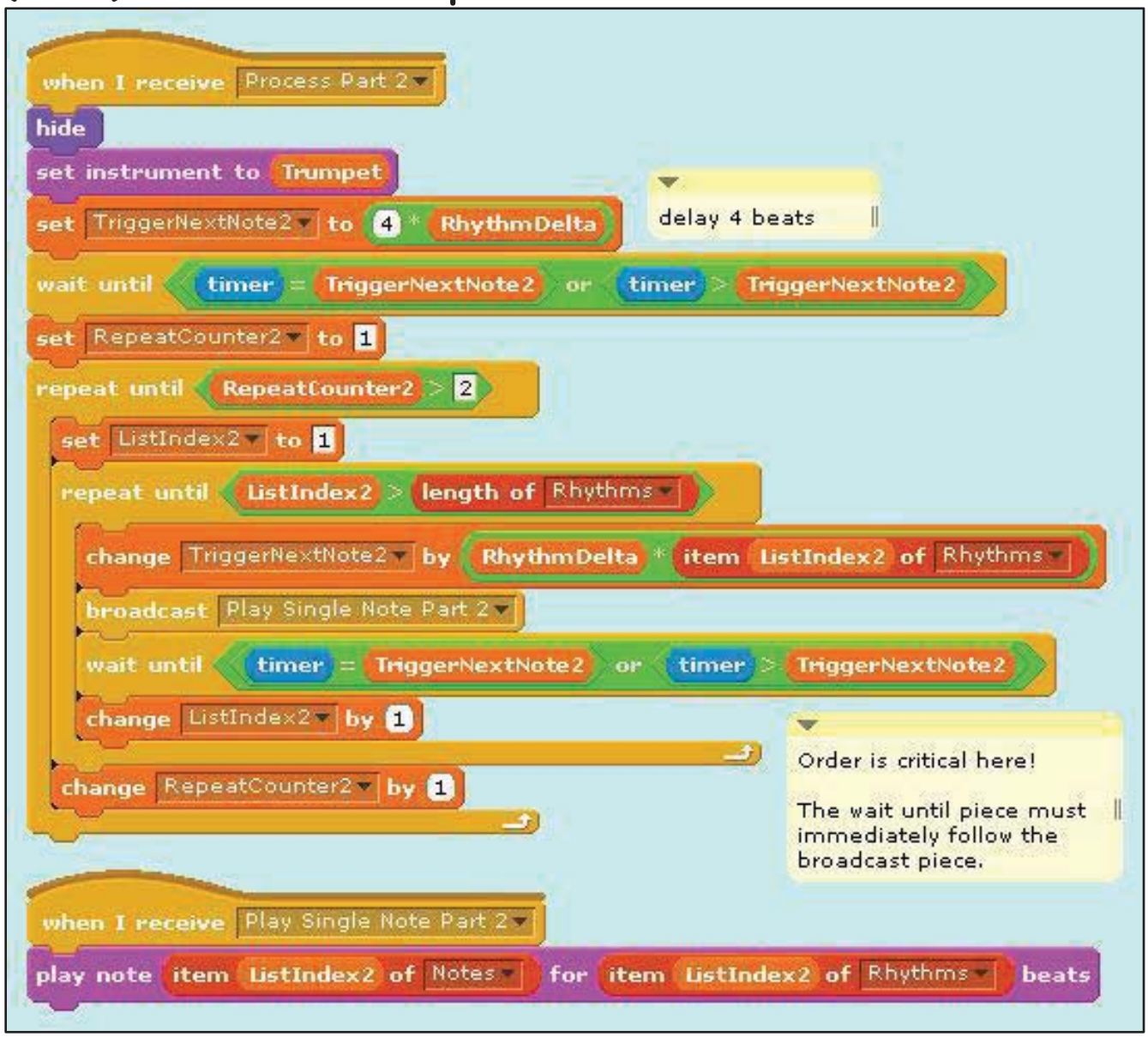
continued on next page

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Row, Row, Row Your Boat

Version 10: Synchronizing Play from Lists (cont'd)

(10d) **Part 2** Script



end of Version 10

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Extending the Examples

1. **Use a variable to set the tempo.**
   - Add a slider to the variable so that you can change the tempo in real time.
   - Find all the places you need to use the variable to reset the tempo when you change it in real time.
   - Which version of playing the round best stays synchronized when you change the tempo?

2. **Transpose the melody to another key.**
   - Create a variable to hold a pitch offset.
   - Find all the places you need to use that variable to play the melody in the new key.

3. **Increase the number of times that the round repeats.**
   - Do the parts stay in synch?

4. **Increase the number of parts that play simultaneously.** (Be sure to set Turbo Speed to do this!)
   - When should each part "come in"?
   - How much should the first beat of each part be offset?

Extending the Examples (cont'd)

5. **Play the melody backwards.**
   - Can you play multiple parts backwards, too?

6. **Increase the number of times that the round repeats.**
   - Do the parts stay in synch?

7. **Increase the number of parts that play simultaneously.** (Be sure to set Turbo Speed before you try this!)
   - When should each part "come in"?
   - How much should the first beat of each part be offset?

8. **Make a round using the G-major scale.**
   - Put the note values for a G-major scale into a list. See page 32 for code that initializes and plays a G-major scale, but remember that you must use the integer values, not the variable names, to play notes from a list.
   - Start Part 2 when Part 1 plays its third note (B, MIDI note #59).
   - Add Part 3, starting when Part 1 plays its fifth not (D, #62).

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.

Extending the Examples (cont'd)

9. **Play random notes in the G-major scale.**
   - Start with the list created for the previous exercise.
   - Use the "pick random" piece in the Operators group to pick a random note from the list.
   - Play each note for 0.25, 0.50, 0.75, or 1.00 beats, also selected randomly.
   - Does the result sound musical?

10. **Create a program that can play any <u>major</u> scale given any starting note.**
    - Store the starting note in a variable.
    - For a major scale, the number of half-tones between each note is:
      2, 2, 1, 2, 2, 2, 1
    - Another way to think about this is:
      Do + 2 ➔ Re + 2 ➔ Mi + 1 ➔ Fa + 2 ➔
      Sol + 2 ➔ La + 2 ➔ Ti + 1 ➔ Do
    - Create a list containing the changes between the notes, and then use a loop to process the list and play the scale.

Extending the Examples (cont'd)

11. **Create a program that can play any harmonic minor scale given any starting note.**
    - For a harmonic minor scale, the number of half-tones between each note is:
      2, 1, 2, 2, 1, 3, 1
    - Create a new list containing these changes, but use the same loop that you created for the previous exercise to play this scale.

12. **Create a program to play a major chord.**
    - A major chord is the 1st, 3rd, and 5th notes of the scale, usually complemented by the octave above the 1st note. Thus, a G-major scale has notes G (#55), B (#59), D (#62), and G' (#67).
    - Another way to think about this is to compute the half-tone difference from the starting note: 0, 4, 7, 12.
    - Set a starting note and then use a "broadcast" to play the four notes simultaneously.

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu. Performamatics is an interdisciplinary project at UMass Lowell funded by the National Science Foundation. See http://performamatics.org.