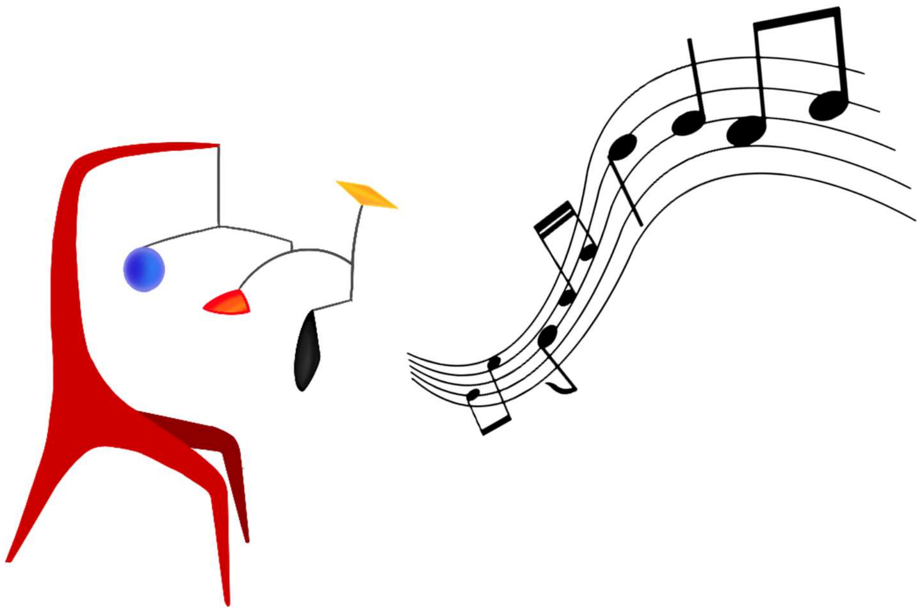
TuneScope
Creating Digital Music in Snap!



Glen Bull, Rachel Gibson,
Jo Watts, and N. Rich Nguyen

TUNESCOPE

Creating Digital Music in Snap!

Editors

Glen Bull, Rachel Gibson, Jo Watts,
N. Rich Nguyen, and Luke Dahl

Technical Editors

Gina Bull and Emily Liu

Art Editor

Alexis Kellam

TuneScope Code Development

Eric Stein and Harsh Padhye

Published by

Association for Advancement of Computers in Education (AACE)

Preface

I love *TuneScope*. If it weren't for computers, I'd be fooling around with musical instruments every free minute. After all, what else besides music delivers direct emotional access? *TuneScope* lets me combine my favorite pastimes - programming and music making - all in the Snap*!* platform. It's a brilliant testament to a vibrant world-wide community of lifelong learners and a bold statement that education ought to be multi-modal and holistic. – *Jens Möenig*

TuneScope music blocks are an extension of the educational programming language, Snap! This language was developed by Jens Möenig and Brian Harvey with support from the National Science Foundation. Snap! is designed to make advanced computational concepts accessible to nonprogrammers. It is currently used by more than a thousand high school and college computer science teachers. It is also used for personal enrichment and enjoyment by countless others.

The TuneScope music library in Snap*!* extends the types of music that can be created in Snap*!* TuneScope was developed through collaboration among the School of Education and Human Development, the Department of Computer Science, and the Department of Music at the University of Virginia. A companion course is approved as an elective in the Bachelor of Arts in Computer Science program at the University of Virginia. The primary objective is to facilitate creativity and exploration of the arts through coding.

Brian Harvey notes, "Languages in the *Logo* family, including *Scratch* and *Snap!*, take the position that our mission is to bring programming to the masses." We share that goal.

Acknowledgements

TuneScope extends the music creation capabilities of *Snap!* through addition of sampled musical instruments such as pianos, guitars, and drums. Eric Stein and Harsh Padhye, computer science students working under the direction of Rich Nguyen, developed *TuneScope*. Luke Dahl, professor of music technology at the University of Virginia, served as a consultant for elements related to music theory and pedagogy.

Modules with instructional videos and resources that support topics in the book were developed with support from a National Science Foundation grant (DRL-1842342, Glen Bull, P.I.). These resources are available on the *Make to Learn* web site at:

https://www.maketolearn.org/music

We would like to acknowledge an enormous debt to Brian Harvey and Jens Möenig for creation of the computing environment, *Snap!,* that made our work possible. We appreciate their support and encouragement throughout this effort and thank them for the many questions that they answered. Brian and Jens created a world and a community that has facilitated and enabled creative work by many others, including the TuneScope working group.

Table of Contents

Chapter 1

Introduction

This initial module provides an orientation to the Snap*!* programming language. If you are already familiar with Snap!, you can skip ahead to the next chapter.

Snap*!* was developed at the University of California, Berkeley, and is used in computer science (CS) for non-CS majors at that university. Because of its widespread use, a user community has developed around this programming language. The capabilities of this language make it well-suited to explorations in art and music.

Topic 1.1 Securing a Snap! Account

A free Snap*!* account can be obtained from the University of California, Berkeley web site:

https://snap.berkeley.edu/

Note: At the time this is written, the blocks and music extensions described are available on the Snap! development site:

https://snap.berkeley.edu/versions/dev/snap.html

They will be incorporated into the main Snap! build after its is next update (from Ver 8.0 to 8.1).

Secure a Snap*!* account before continuing. The Snap*!* reference manual is available here:

https://Snap.berkeley.edu/Snap/help/SnapManual.pdf

Section II of the reference manual provides information about securing a Snap*!* account and saving projects. Review this section before continuing to the explorations that follow below.

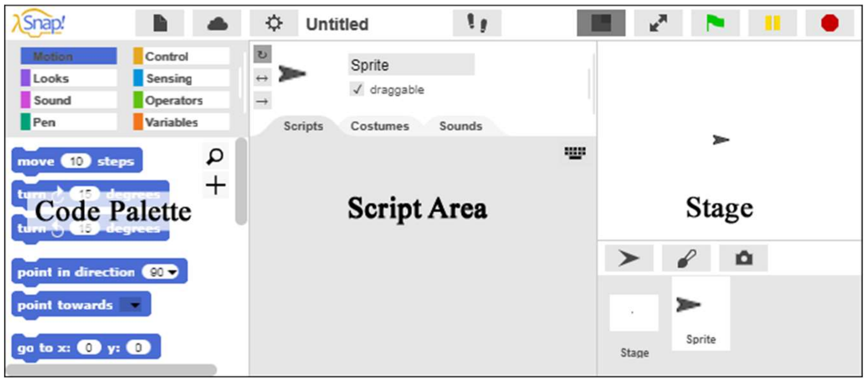
Snap*!* help forums maintained at the University of California, Berkeley, are available here:

https://forum.Snap.berkeley.edu/

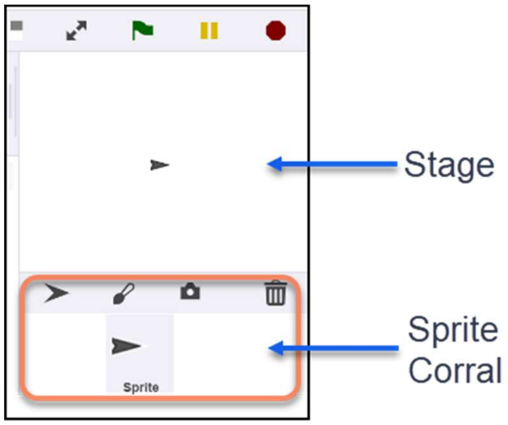
Assistance with questions that are not addressed in the reference manual can be obtained through the Snap*!* forum. In order to gain trusted status on the forum, you must read at least five different posts and spend at least fifteen minutes on the forum. This is to prevent spam posts by bots. For the same reason, if you are extremely fast typist, the forum software may assume that you are a bot.

Topic 1.2 The Snap! Workspace

In a typical Snap*!* session, blocks of code are dragged from the *Code Block Palette* on the left-hand side of the screen to a *Script Area* in the middle of the screen. Blocks of code are snapped together to create scripts. Clicking a group of code blocks causes the script to run, peforming an action. Often these actions involve movement of sprites on a *Stage* at the right-hand side of the screen.



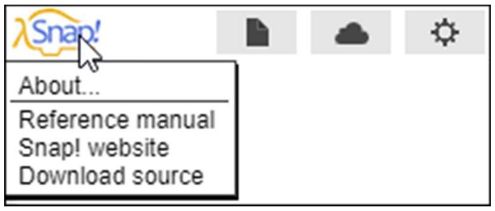
A *sprite corral* beneath the stage indicates which sprite is currently selected.



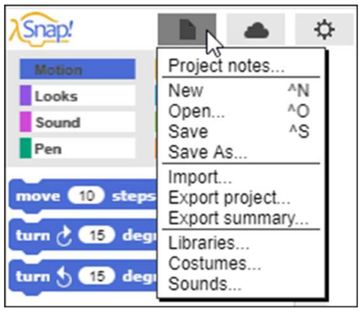
Each sprite has its own separate script space.

Topic 1.3 Snap*!* Menus

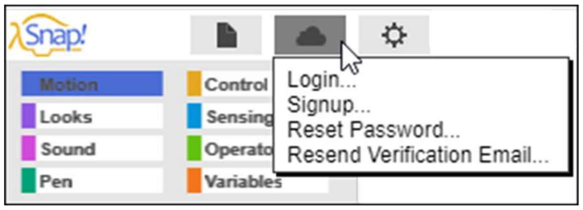
There are several menus that can be accessed in the top left-hand corner of the Snap*!* screen. The Snap*!* icon in the top left-hand corner can be used to access the Snap*!* reference manual.



The *File* menu to the right of the Snap*!* icon can be used to save projects and open projects that have been previously saved. This menu can also be used to access costumes for sprites, sounds, and libraries of additional Snap*!* code blocks.



The *Login* menu is to the right of the *File* menu.



A user must be logged into Snap*!* in order to save projects. Therefore it is a good idea to log in to Snap*!* at the beginning of every session.

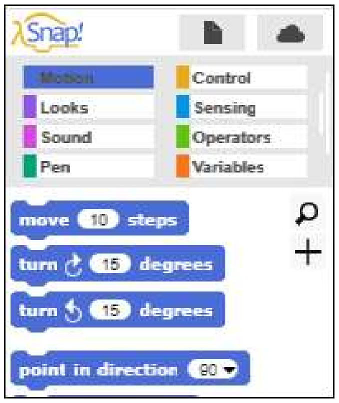
A *Settings* menu is to the right of the *Login* menu. This menu provides a number of options for customizing Snap*!* For example, the *Zoom Blocks* option can be used to increase the size of the code blocks so that they can be more easily viewed in presentations.

Exploration 1.3

If you have not already, log into Snap*!* Explore some of the options available under the different menu settings.

Topic 1.4 The Code Block Palette

Scripts are created in Snap*!* by snapping blocks of code together (such as the **Move 10 Steps** code block in the illustration). The types of *code blocks* available are displayed in a *Code Block Palette* at the top left-hand side of the screen. For example, the *Motion* code blocks are currently highlighted in the palette below. Other categories of code blocks include *Looks*, *Sound*, *Pen*, *Control*, *Sensing*, *Operators*, and *Variables*. Each category is a different color (e.g.. *Motion* code blocks are blue). Click on the different categories (Motion, Looks, etc.) to access the code blocks associated with that category.



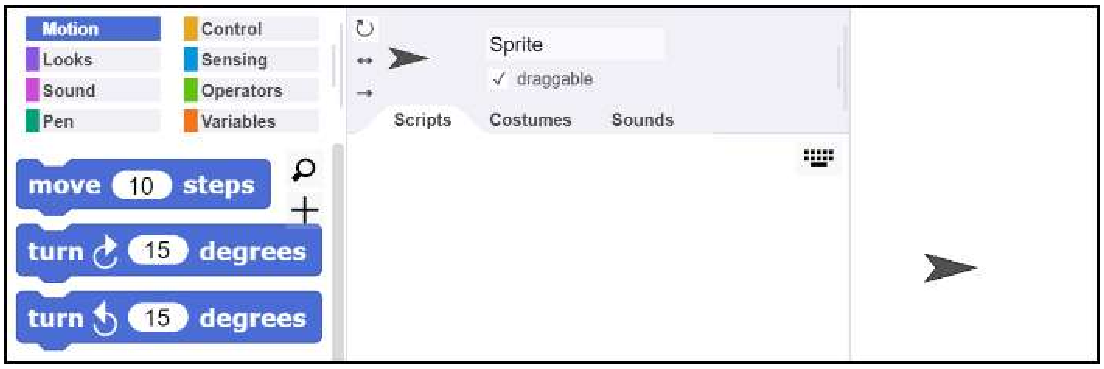
The *Motion* code blocks direct the movement of sprites (actors that can move about the stage on the right-hand side of the screen.) The *Looks* code blocks control the appearance of sprites. The *Sound* code blocks are used to play sounds. The *Pen* code blocks control the color and thickness of a pen that the sprite uses to draw lines on the stage. The *Control* code blocks provide control structures such as the **Repeat** command. The *Sensing* code blocks are used to sense the status of Snap*!* objects and monitor external inputs such as the keyboard and the microphone. The *Operators* code blocks provide mathematical and logical functions. The *Variables* palette is used to create and modify variables.

Exploration 1.4 The Code Block Palette

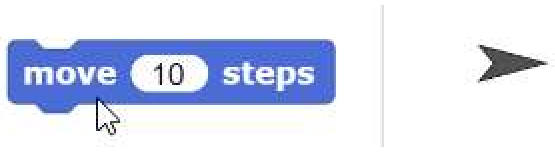
Click on each of the categories in the *Code Block Palette* to get a sense of the types of commands that are found under each category.

Topic 1.5 Motion and Pen Palettes

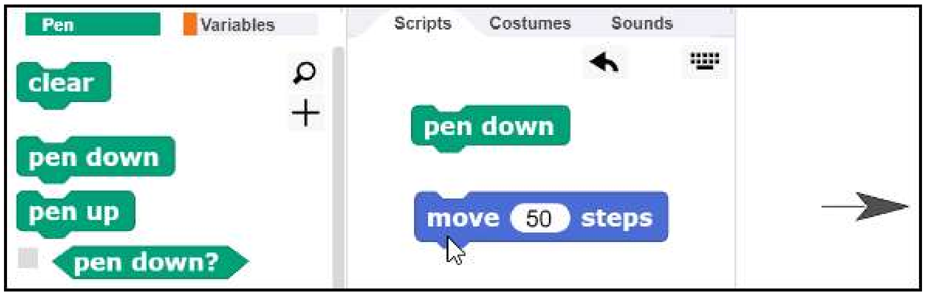
The different types of code blocks are arranged in palettes on the left-hand side. Each category of code block is a different color. The motion commands are blue code blocks.



If a motion block is moved into the script area in the middle of the screen and clicked, it moves a sprite (the black triangle on the right).

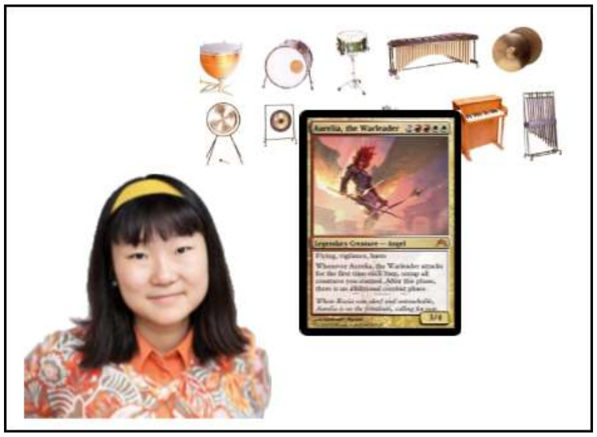


A group of commands in the *Pen Palette* control a pen that can be raised or lowered as the sprite moves.



Topic 1.6: Creating an Animated Biography

Snap*!* can be used as a scripted storytelling tool. In the following example, this capability will be used to create a short, animated biography. The example that follows, one of the authors describes her background as an orchestral percussionist and her interest in the game *Magic, the Gathering*.



Two videos describing the way in which this animated biography was created can be accessed here:

Part 1: [Animated Biography – Video 1]

Part 2: [Animated Biography – Video 2]

Once you have watched the video overview, open the Snap! biography program that is available here:

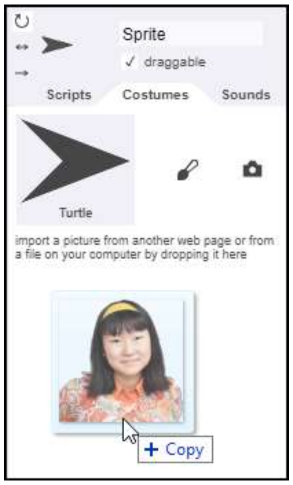
[Snap! Biography Program]

Note: Log into Snap! before accessing the link above. The Snap*!* file will enable you to inspect the code blocks in the program to see how the blocks were used to create the animated biography. Once you understand how everything works, create a similar animated biography for yourself.

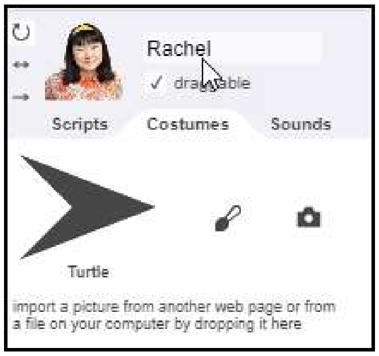
The remainder of this chapter describes the process in greater detail. This will enable you to refer to a specific action or method if you need more information about the way in which a result was obtained.

Topic 1.7: Creating an Animated Biography

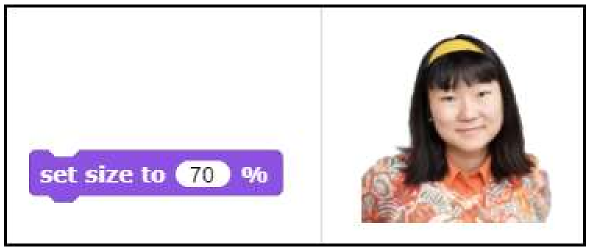
Begin by dragging a picture of yourself from the desktop into the *Costumes* tab.


Sprite
✓ draggable
Scripts
Costumes
Sounds
Turtle
import a picture from another web page or from a file on your computer by dropping it here
+ Copy

The costume of the sprite will change from its default, a black triangle, to a thumbnail image with your picture. Change the name in the textbox beside the thumbnail image from the default name of "sprite" to your name.


Rachel
✓ draggable
Scripts
Costumes
Sounds
Turtle
import a picture from another web page or from a file on your computer by dropping it here

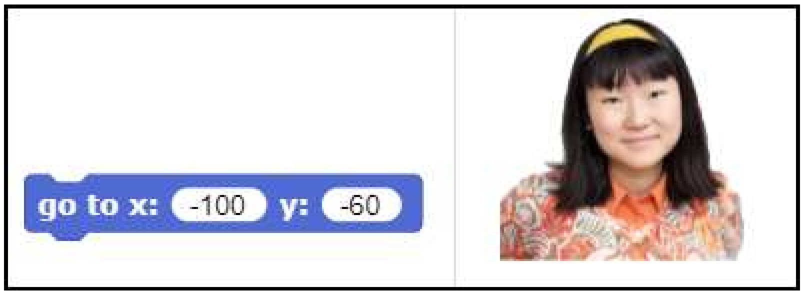
The image of the sprite will also appear on the stage. Attributes such as size can be adjusted using commands in the *Looks* palette.



set size to 70 %

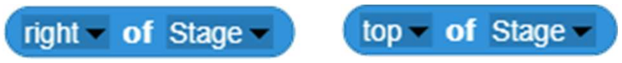
Topic 1.8: Controlling the Position of a Sprite

The default size of the stage is 480 steps wide by 360 steps wide, with the coordinate of [0 0] representing the center of the stage. The Go To block can be used to send the sprite to any part of the stage. The coordinates of [– 100 - 60], for example, would send the sprite to the lower left-hand corner of the stage.



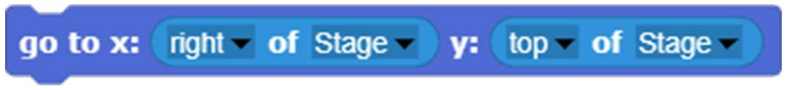
```
go to x: -100 y: -60
```

Blocks such as Right of Stage and Top of Stage (found in the Sensing palette) can be used to send the sprite to a specific corner of the stage.



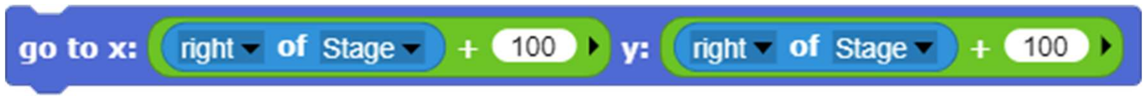
```
right of Stage    top of Stage
```

For example, this command would send the sprite to the top, right corner of the stage.



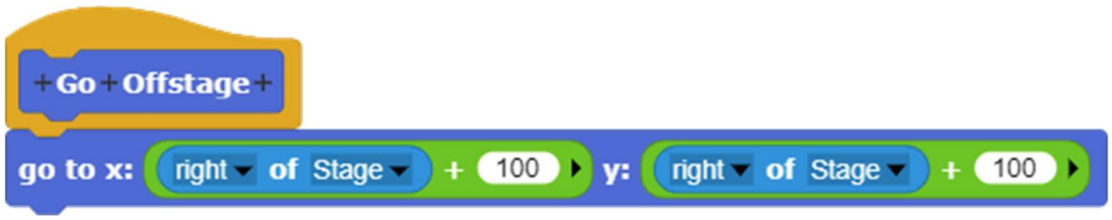
```
go to x: right of Stage y: top of Stage
```

The command below would send the sprite beyond the boundaries of the stage.



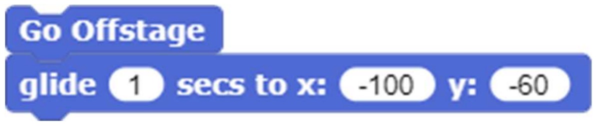
```
go to x: right of Stage + 100 y: right of Stage + 100
```

Since several different sprites will be asked to go offstage, this block will be used frequently. For that reason, creation of a custom block named Go Offstage will document the reason that these coordinates are used.



```
+Go+Offstage+
go to x: right of Stage + 100 y: right of Stage + 100
```

Once the Go Offstage block is created, the following sequence would ask the sprite to start off stage and then glide to the lower left-hand corner of the stage.

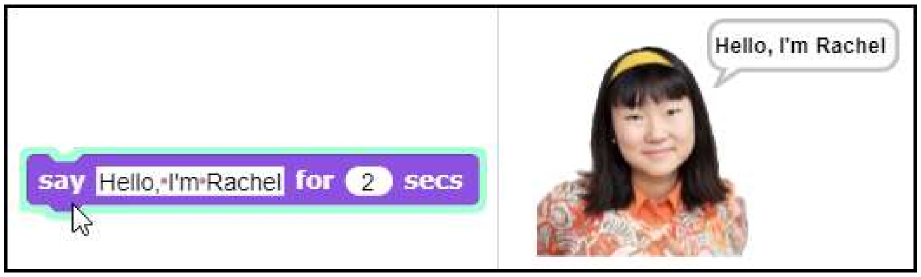


```
Go Offstage
glide 1 secs to x: -100 y: -60
```

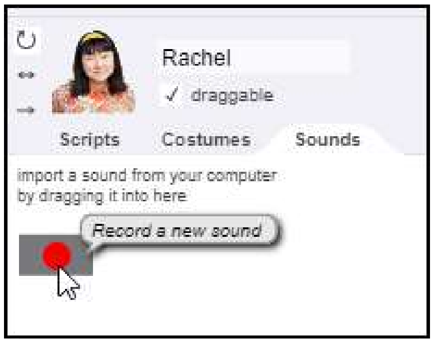
The method described provides the foundation of a method for creating objects in Snap! and animating them to tell a story.

Topic 1.8: Creating Dialog

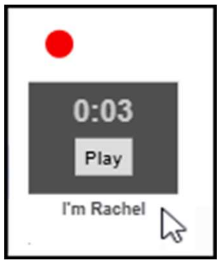
The **Say** command (found under the *Looks* palette) causes a speech bubble to appear beside the object.



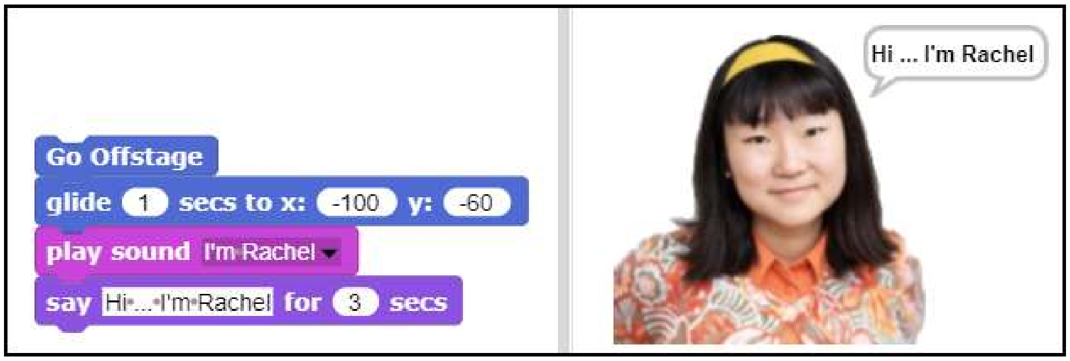
The *Sounds* tab above the script area can be used to record speech using the microphone of the computer. (Note: the *Record* function does not work in the Safari web browser.)



Once speech has been recorded, the caption of the sound can be changed in the text box below the sound.



Putting it all together, the following sequence of blocks would cause the sprite to go offstage, glide to the lower left-hand corner, play the recording, "I'm Rachel" and place the same phrase in a speech bubble.

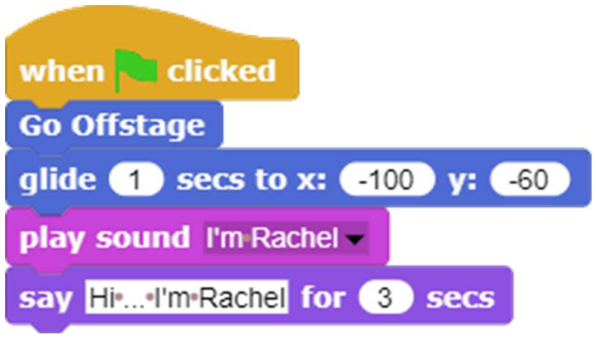


```
Go Offstage
glide 1 secs to x: -100 y: -60
play sound I'm Rachel
say Hi ... I'm Rachel for 3 secs
```

In this manner, a character can be created using a costume of a sprite and used to develop an animated biography.

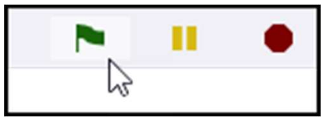
Topic 1.9: Initiating the Sequence

A *Green Hat* block (founded in the *Control* palette) can be used to initiate the sequence.

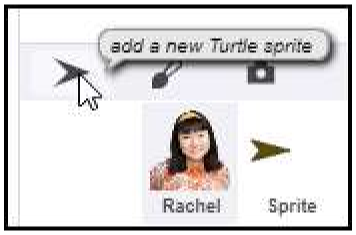


```
when [flag] clicked
Go Offstage
glide 1 secs to x: -100 y: -60
play sound I'm Rachel
say Hi ... I'm Rachel for 3 secs
```

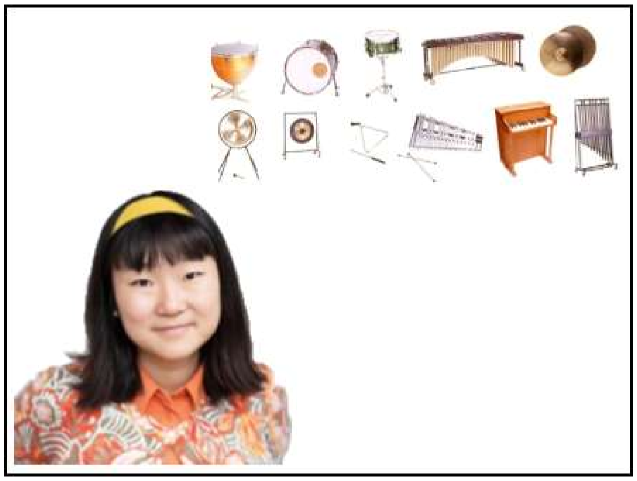
When the Green Hat block is added, clicking the green flag in the upper right-hand corner of the Snap! window will initiate the sequence of commands found under this block.



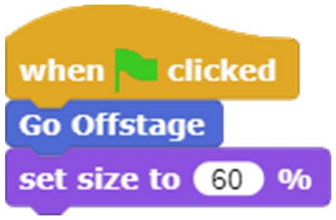
Additional sprites can be created to add other objects to illustrate the story. An additional sprite can be added by clicking the black triangle in the sprite corral (found just below the stage).



A new costume can be added to the second sprite by dragging an image from the desktop into the costume tab of the second sprite. In this instance, an image of percussion instruments has been added to the stage.

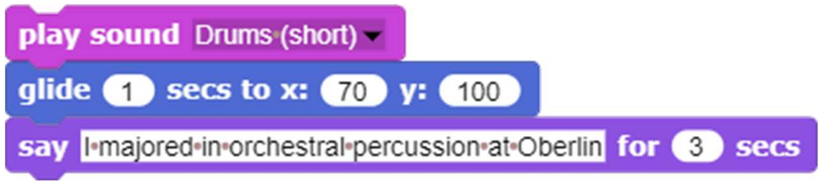


The following script in the script area of the second sprite (which has been named "Percussion" to reflect the image of percussion instruments) positions the sprite and adjusts its size when the green flag is clicked.



```
when [flag] clicked
Go Offstage
set size to (60) %
```

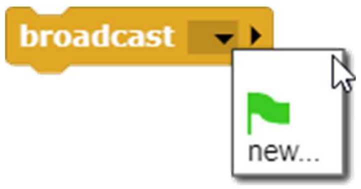
One of advantage of using the *Green Flag* to begin a program is that the scripts of several sprites can be initiated simultaneously, taking place in parallel. The script for the *Percussion* sprite employs a format similar to that used for the first sprite. It plays the sound of drums as the sprite glides to a point in the upper right-hand corner of the stage. A speech bubble then appears that says, "I majored in orchestral percussion at Oberlin."



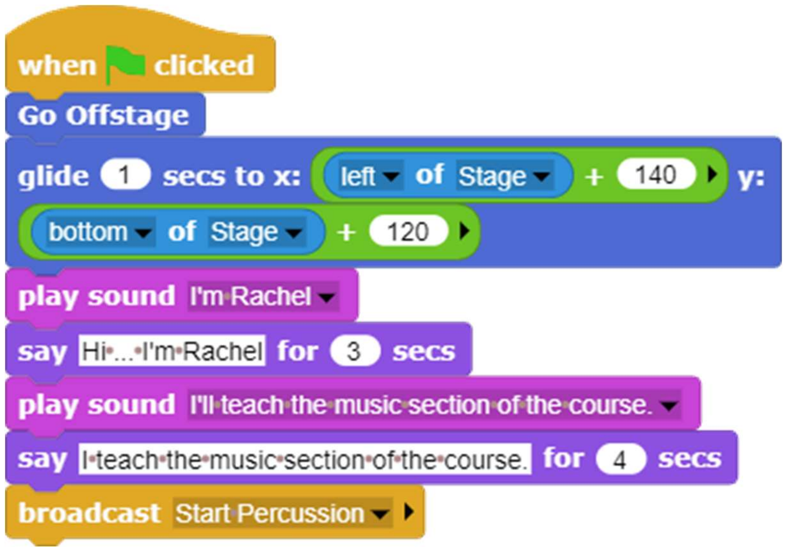
```
play sound [Drums (short) v]
glide (1) secs to x: (70) y: (100)
say [I majored in orchestral percussion at Oberlin] for (3) secs
```

Topic 1.10. Completing the Animated Biography

The **Broadcast** block (found in the Control palette) enables the first sprite to send a message to the second sprite.

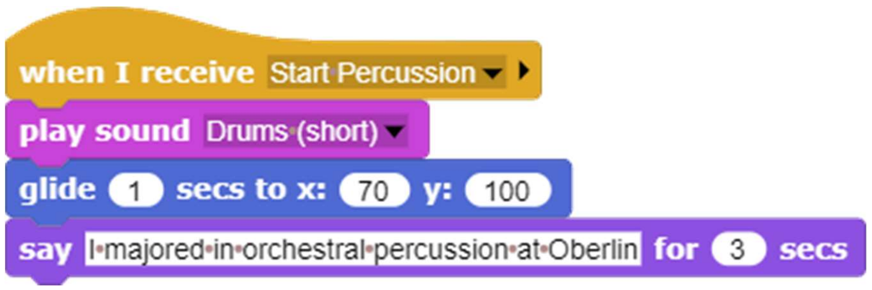


This enables the first sprite to broadcast the message “Start Percussion” to initiate the percussion sequence.



```
when [flag] clicked
Go Offstage
glide 1 secs to x: (left of Stage) + 140 y: (bottom of Stage) + 120
play sound I'm Rachel
say Hi... I'm Rachel for 3 secs
play sound I'll teach the music section of the course.
say I teach the music section of the course. for 4 secs
broadcast Start Percussion
```

The **When I Receive [message]** block (found under the *Control* palette) can then be used to enable the second sprite to begin its sequence of commands when it receives the message “Start Percussion.”



```
when I receive Start Percussion
play sound Drums (short)
glide 1 secs to x: 70 y: 100
say I majored in orchestral percussion at Oberlin for 3 secs
```

In this manner, sprites can send messages to one another to coordinate the order in which events occur.

Conclusion

This introduction to scripts and sprites in Snap*!* provides an orientation to a programming and development tool that will be used in the subsequent chapters. If you would like to know more about this tool, the Joy and Beauty of Computing is an introduction to general computing concepts developed at the University of California, Berkeley:

Beauty and Joy of Computing

In the next chapter, this tool will be used to introduce digital sound and sound sampling.

Chapter 2

Introduction to Digital Audio and Sampling

This chapter introduces digital audio and sampling. The following topics were covered in the preceding chapter: (1) Securing a Snap*!* Account, (2) The Snap! Workspace, (3) Snap*!* Menu**s,** (4) The Code Block Palette, (5) Motion and Pen Palettes. If you are not already familiar with these topics, review this prerequisite information before continuing.

This chapter continues with foundational concepts in Snap*!* It also introduces two basic methods for generating sound with a computer:

1. Digitized Sound
2. Synthesized Sound

Digitized sound converts a sound in the physical world into a table of numbers inside the computer. Digitizing a sound in this manner makes it possible to manipulate the sound using a digital computer. The process of converting a sound into a digital format inside a computer is also known as *sampling*. For example, an artist might refer to the process of digitizing a drum on a recording as "sampling a drum beat."

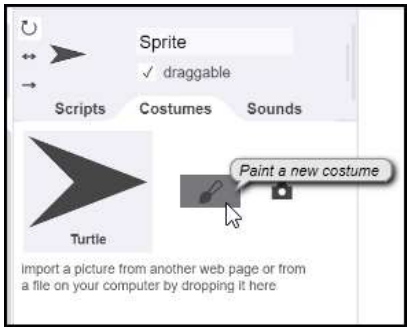
Synthesized sound generates a tone using an algorithm. This method creates an artificial sound inside the computer and plays it through the computer's speaker.

In the sections that follow, the process of constructing a circle will be used to construct a *sine wave*. Any regularly recurring sound, such as a note from a violin or a piano, can be constructed by combining multiple sine ways. The process of constructing a musical note in this manner is known as *music synthesis*.

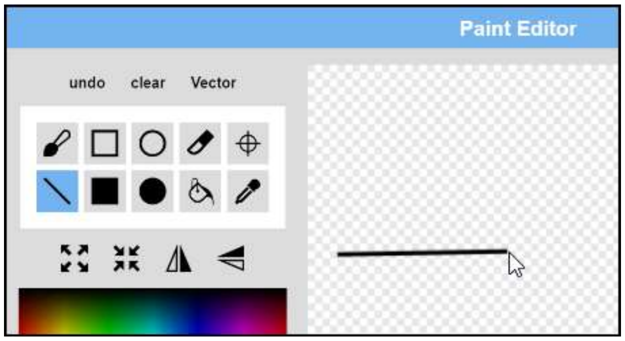
Topic 2.1 The Costume Editor

The costume editor can be used to edit the costume of the sprite. Ultimately this will result in in a visual representation of a soundwave. This graphic representation of a sound can then be used to create a tone that can be played through a computer speaker.

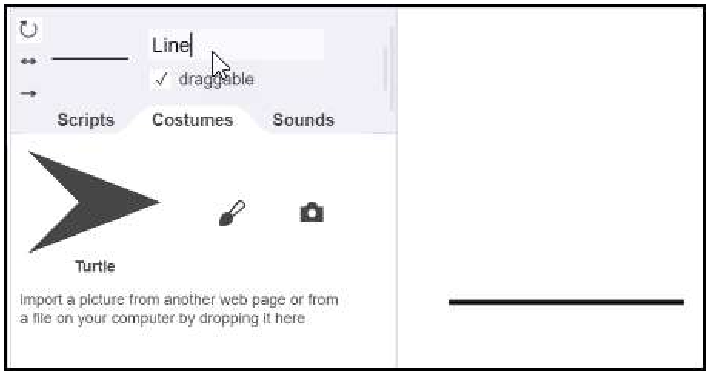
The center tab above the script editor can be used to access a paint program that can be used to edit the sprite's costume.


Sprite
✓ draggable
Scripts
Costumes
Sounds
Paint a new costume
Turtle
Import a picture from another web page or from a file on your computer by dropping it here

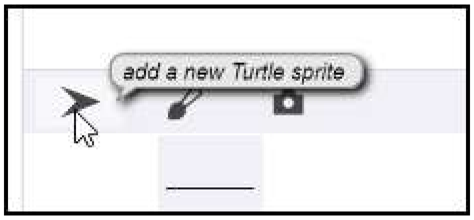
In the example below, the paint editor has been used to create a costume that is a line.



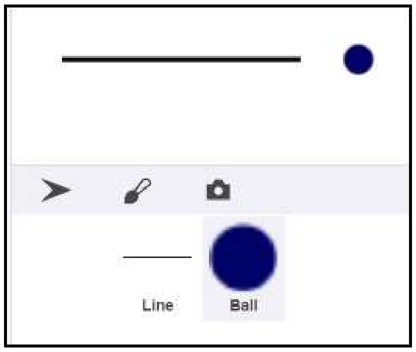
The name of the sprite can be changed to reflect the shape of its new costume.


Line
draggable
Scripts
Costumes
Sounds
Turtle
Import a picture from another web page or from a file on your computer by dropping it here

Clicking on the sprite icon at the bottom of the stage provides the option of creating additional sprites.

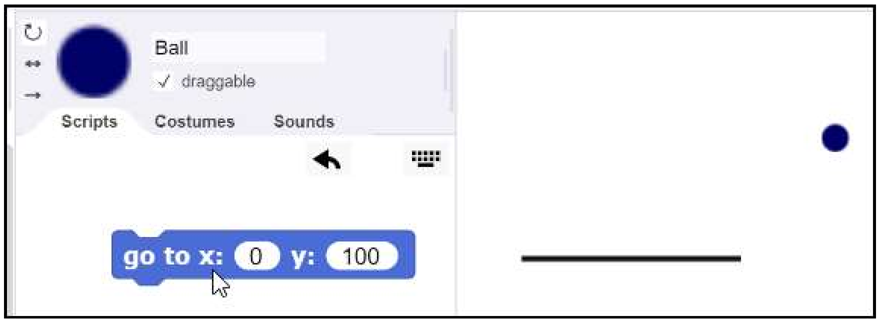


In the example below, a second sprite has been given the costume of a ball.

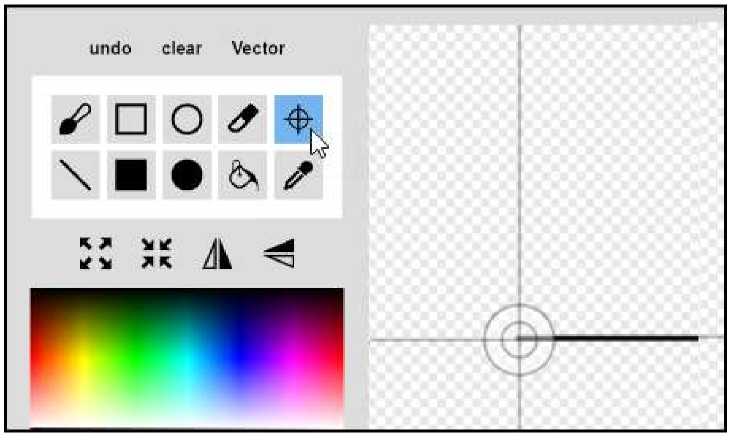


Topic 2.2 Drawing a Circle

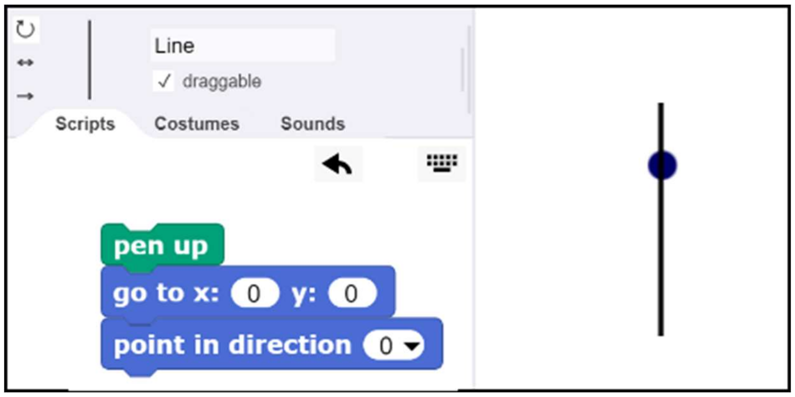
Each sprite has its own script area that is used to control the actions of that sprite. (If all of your scripts seem to suddenly disappear, you may have inadvertently clicked on the script area of a different sprite.) The name of the sprite associated with each script is shown at the top of the script area. In the example below, the sprite named *Ball* has been sent to position 0 on the X Axis and position 100 on the Y Axis.



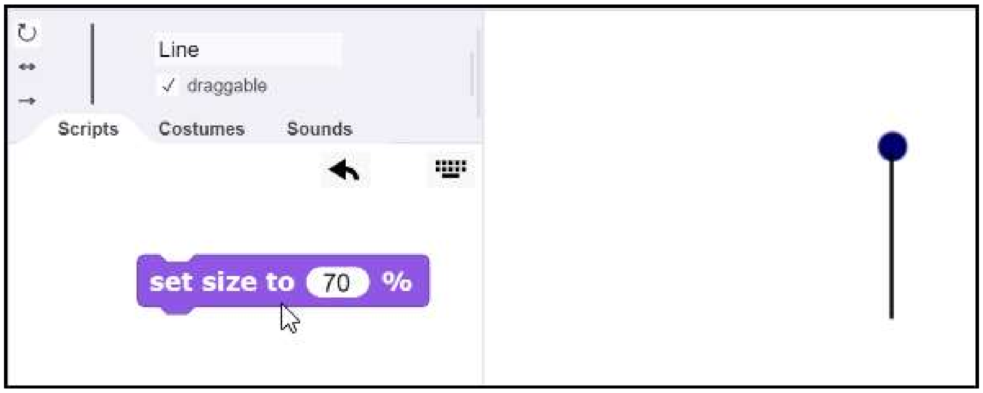
The point on a sprite that is used as the target for coordinate commands can be adjusted by using the scope tool (an icon that looks like cross hairs in a scope) in the paint editor. In the example below, the coordinate target for the sprite named *Line* has been placed at the left-hand side of the line.



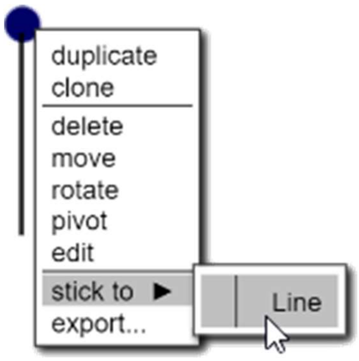
In the example below, the sprite named line has been told to go to the coordinates of "0 0" and point in the direction of "0" (i.e., pointing up).



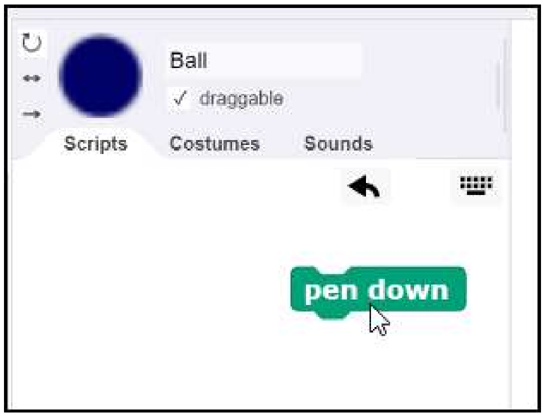
The **Set Size** block (found under the *Looks Palette*) can then be used to adjust the size of the line so that its tip intersects with the sprite named *Ball*.



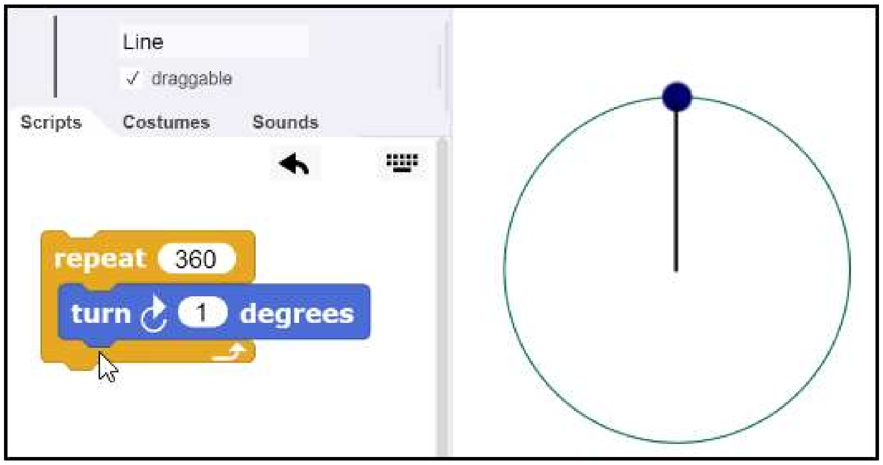
If the *Ball* sprite is right-clicked, a menu will appear that will make it possible to attach the ball to the top of the line.



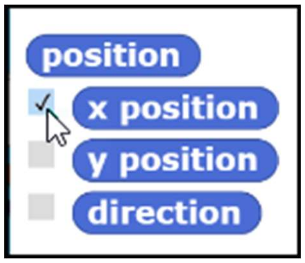
First the **Pen Down** block is applied in the script area of the *Ball* sprite.



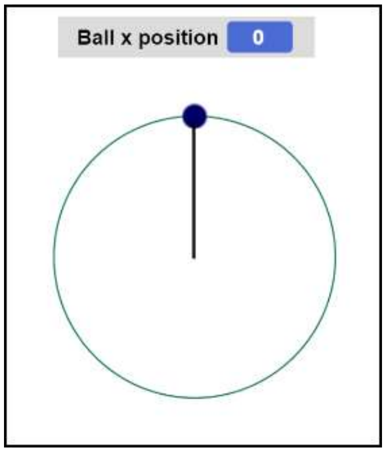
The **Repeat Block** (found under the *Control* palette) can be combined with the **Turn** block to rotate the line through a 360-degree arc. A perfect circle results.



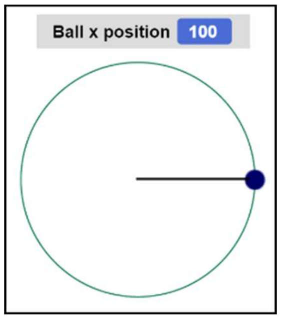
As the ball moves around the circle, you can view the X coordinate of the ball's location by enabling a checkbox beside the **X Position** reporter block



Once this is done, the position of the ball on the X axis will be reported.

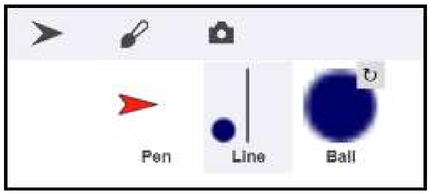


When the line is pointing in a direction of 90 degrees, the position of the ball on the X axis is 100. When the line is pointing a direction of – 90 degrees, the position of the ball on the X axis is – 100. When the ball is at the top or bottom of the circle, its position on the X axis is 0 (i.e., in the center of the horizontal axis).

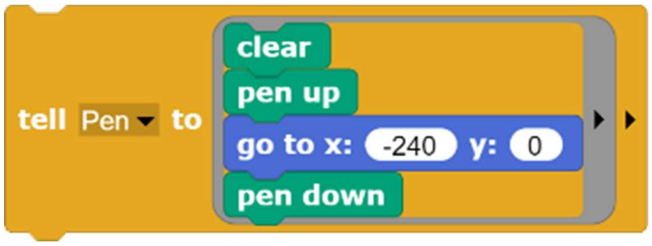


Topic 2.3 Graphing a Waveform

It can be difficult to visualize the intermediate positions. However, computers are adept at graphing data. A third sprite named Pen can be created to draw a graph of the position of the ball as it moves around the circle.

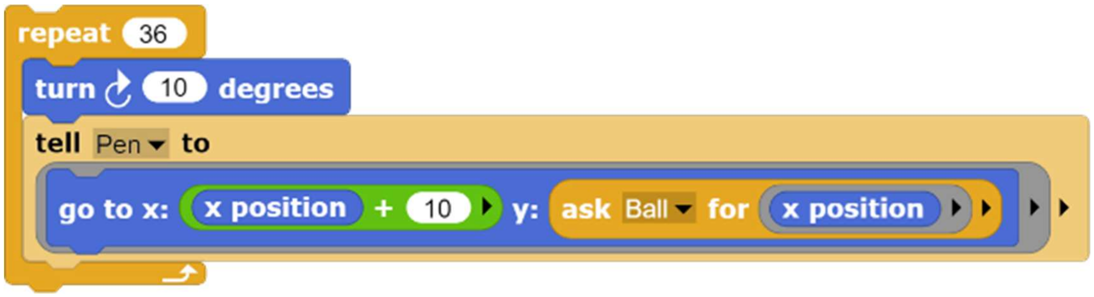


A setup procedure can be created to tell the Pen sprite to (1) clear the screen, (2) put the pen up, (3) go to the left side of the stage (i.e., coordinates of “-240 0”), and (4) put the pen down. (The **Tell** command is found under the *Control* palette.)



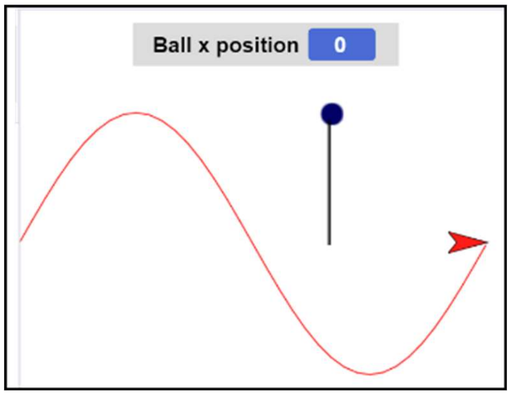
```
tell Pen to
    clear
    pen up
    go to x: -240 y: 0
    pen down
```

The position of the ball on the X axis can then be graphed in ten-degree increments by telling the pen to plot the X position of the ball on the vertical axis, and move over 10 degrees each time.



```
repeat 36
    turn ↻ 10 degrees
    tell Pen to
        go to x: (x position + 10) y: (ask Ball for (x position))
```

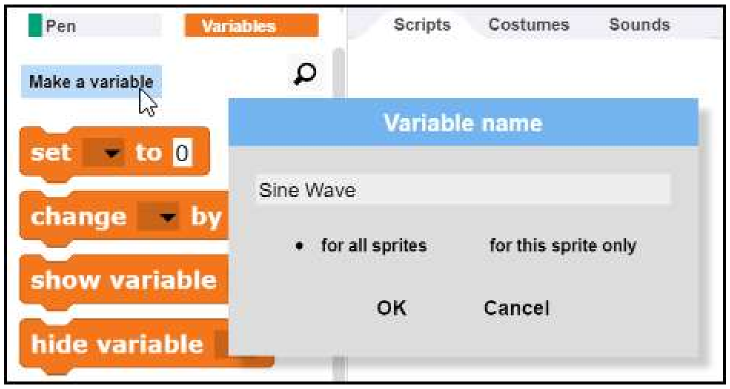
A graph of a figure that is known in trigonometry as a *sine wave* results.



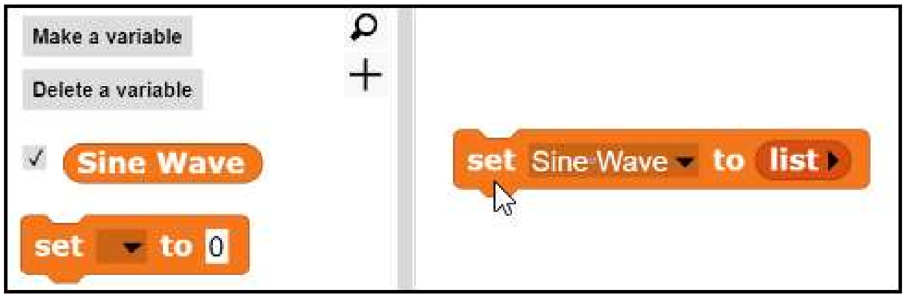
Topic 2.4 Sound Synthesis

The preceding section served two purposes. It provided an overview of some of the commands available in the different code block palettes. It also established the foundation for introduction of synthesized sound. If the coordinates used to generate the sine wave are recorded into a list, the sine wave can be used to generate a tone.

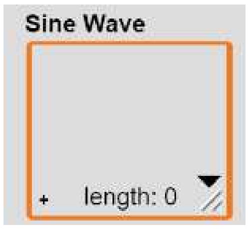
The values can be recorded in a list by making a variable. This option is found at the top of the *Variables* palette. In the example below, the variable created has been given the name of *Sine Wave*.



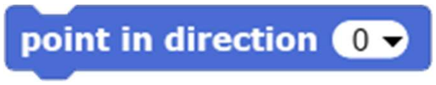
Once a variable has been created, it should be initialized by setting it to an empty list, as shown in the illustration below.



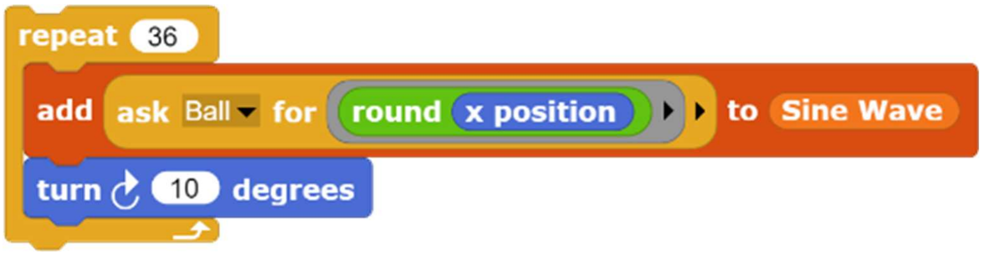
After the variable is initialized in this manner, an empty list will be shown on the stage.



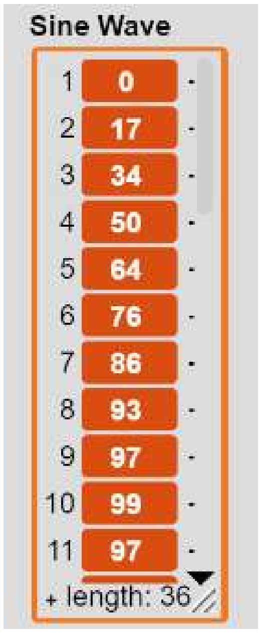
Begin with the sprite named "Line" pointing upwards (i.e., pointing at 0 degrees).


```
point in direction (0 ▾)
```

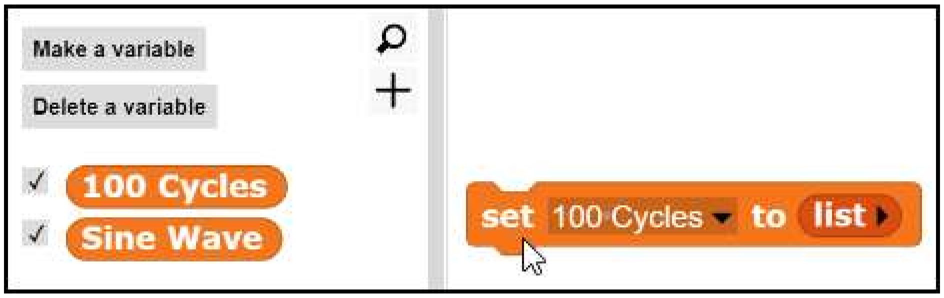
Then repeat the procedure developed in the preceding section, adding the position of the ball at ten-degree intervals to the list.


```
repeat (36)
    add (ask [Ball ▾] for (round (x position)) ▸) ▸ to (Sine Wave)
    turn ↻ (10) degrees
```

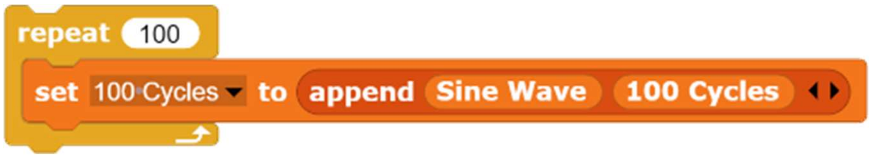
As we would anticipate from previous explorations, the position of the ball on the X axis increases from 0 to nearly 100 as it moves from 0 degrees to 90 degrees. Then the value begins to decrease again as the ball begins to move through the second quadrant of the circle.


```
Sine Wave
1   0
2   17
3   34
4   50
5   64
6   76
7   86
8   93
9   97
10  99
11  97
+ length: 36
```

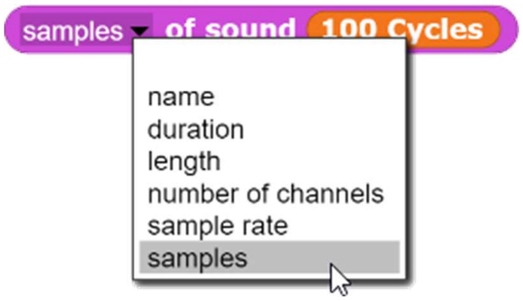
To hear the tone clearly, several cycles of the sine wave are needed. A second variable 100 Cycles can be created to store the additional cycles of the sine wave and initialized.



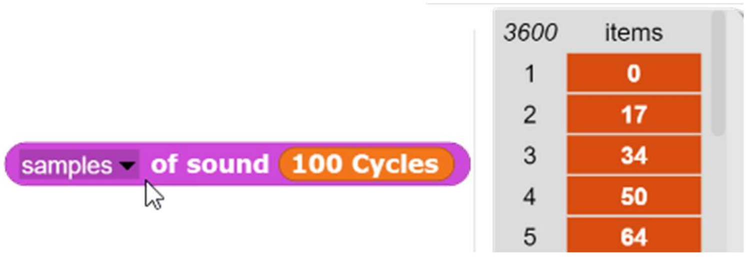
The following procedure stores the addition cycles of the sine wave in the new variable.


```
repeat 100
    set 100 Cycles to append Sine Wave 100 Cycles
```

One of the options found under the **Durations of Sound** reporter block is *Samples of Sound*.

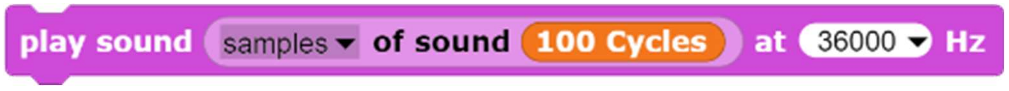


Clicking the reporter block displays a list with the 3600 samples that have been generated.



3600	items
1	0
2	17
3	34
4	50
5	64

The **Play Sound** block can be used to play the samples at a rate of 36,000 samples per second. When this is done, a tone is heard through the computer speaker.



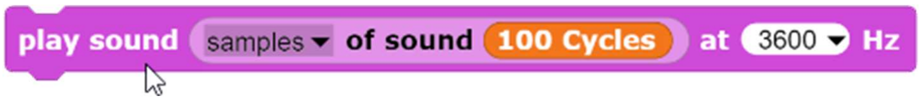
A digital-to-analog converter (DAC) in the computer translates these values into an electrical current that creates a magnetic field in an electromagnet. The magnetic coil, in turn, moves the cone of a loud speaker back and forth to move air molecules back and forth, producing sound.

This process illustrates several characteristics related to the nature of sound. First, sound is generated by a back-and-forth movement. When the values are positive, the cone of the speaker moves forward. When the values are negative, the cone of the speaker moves in the opposite direction. Second, the rate of the back-and-forth movement is the *frequency* of the sound. (The term *frequency* refers to how frequently an event occurs – in this case, the number of back-and-forth movements per second.) The terms *cycles-per-second* and *Hertz* are often used to describe the number of back-and-forth movements per second. In this context, these terms can be used interchangeably.

At a rate of 36,000 samples per second, the 3600 samples in the list will be played back in one-tenth of a second. By extrapolating, we can determine that if 100 cycles can be played back in one-tenth of a second, 1000 cycles would be played back in a full second. In other words, this process generated a 1000 cycles per second tone. (The frequency could also be expressed as a 1000 Hz tone.)

Topic 2.5 Frequency and Intensity

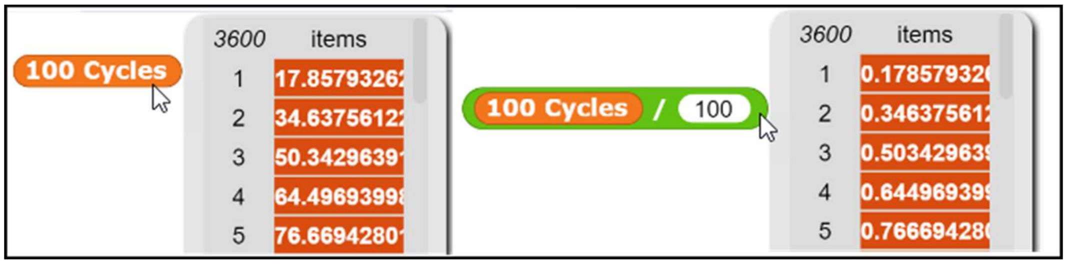
The effect of changing the rate at which the samples are played back can be demonstrated by selecting a playback rate of 3600 Hz instead of 36,000 Hz. This generates a 100 Hz tone.



```
play sound samples of sound 100 Cycles at 3600 Hz
```

This illustrates another aspect of sound. Pitch is the perceptual correlate of frequency. The pitch of higher frequencies is also perceived as higher, but the relationship is not linear. Frequency is a physical phenomenon. Pitch is a perceptual phenomenon. They are related, but they are not the same thing, and should not be used interchangeably.

If frequency is determined by the number of cycles of the sound that are played in a given time period, the intensity of the sound is determined by the amplitude of the waveform. The amplitude of the waveform refers to the height of the waveform. The amplitude of the values can be adjusted using an arithmetic operator. For example, if the values are divided by one hundred, the values will range from 0 to 1 rather than from 0 to 100. (For the sake of comparison, the samples below have not been rounded.)



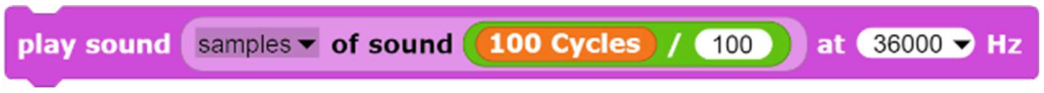
100 Cycles

3600	items
1	17.8579326
2	34.6375612
3	50.3429639
4	64.4969399
5	76.6694280

100 Cycles / 100

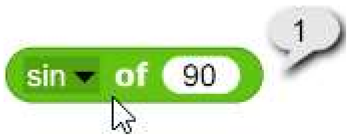
3600	items
1	0.17857932
2	0.34637561
3	0.50342963
4	0.64496939
5	0.76669428

When the lower amplitudes are played back, the magnetic field generated by the speaker coil will be weaker, and the distance traversed by the speaker cone will not be as far. Consequently, the intensity of the sound will be reduced.



Just as pitch is the perceptual correlate of frequency, loudness is the perceptual correlate of intensity. A sound with a lower intensity does not sound as loud.

Note: Snap*!* has a built-in **Sin** reporter block:

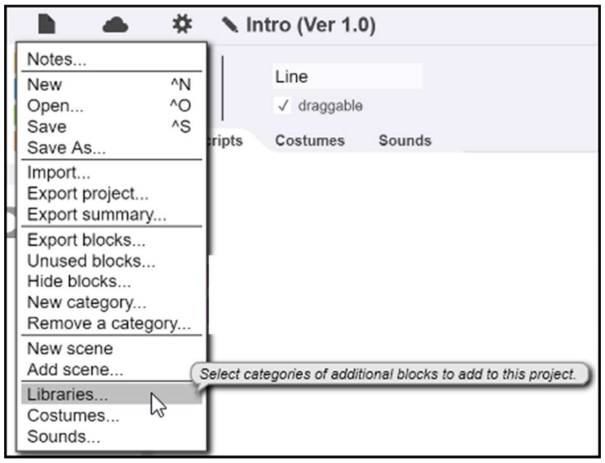


Constructing these values (rather than using the built-in function) illustrates that higher math such as trigonometry or calculus is not required to understand their origins. By following the distance of a point on a circle from the midline of the circle as it moves around the circumference of the circle, it is possible to see how the values are generated.

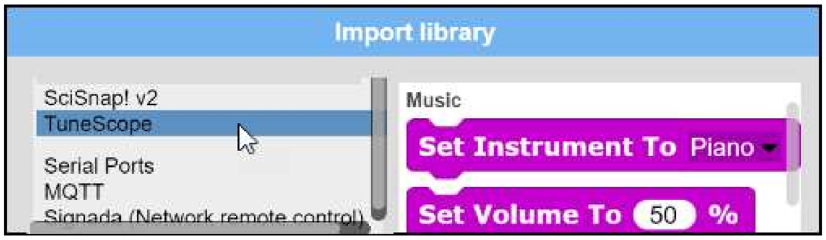
If the motion of a freely moving object in nature is graphed, such as the motion of a pendulum or a vibrating violin string, a sine wave is produced. The example of the circle is just one instance of the more general phenomenon of a back-and-forth oscillation. The sounds produced by most musical instruments are the result of a similar back-and-forth movement, such as vibrating string of a violin or the vibrating reed of a clarinet.

Topic 2.6 The TuneScope Tone Command

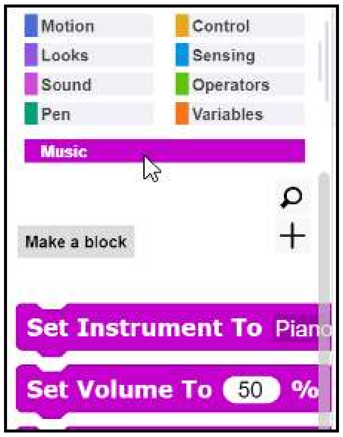
The TuneScope library of music blocks can be imported by selecting the *Libraries* option (found under the *File* menu).



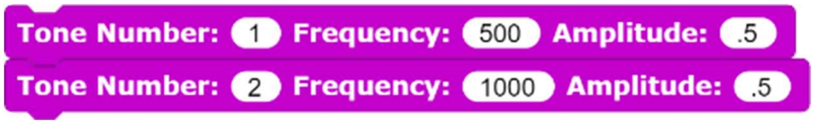
If the *TuneScope* library is selected, additional music blocks will be imported.



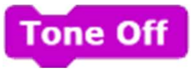
Importing the TuneScope library adds another palette of blocks to Snap*!*



The TuneScope **Tone** block can be used to play multiple tones that combine different frequencies and amplitudes.


Tone Number: 1 Frequency: 500 Amplitude: .5
Tone Number: 2 Frequency: 1000 Amplitude: .5

The **Tone Off** block can be used to turn off the tones.

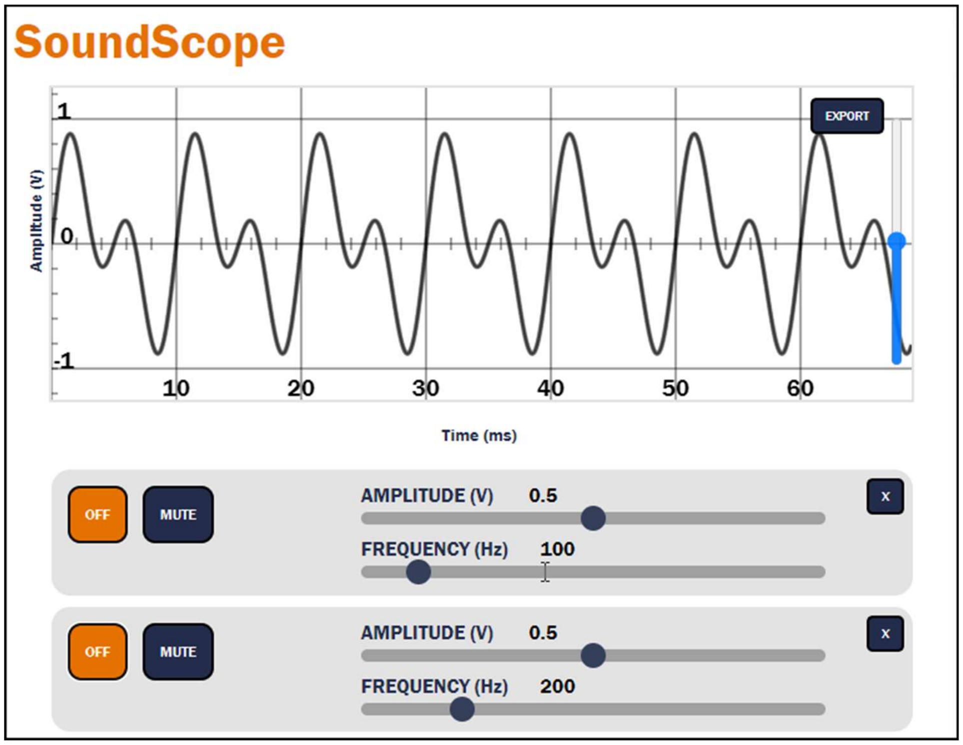


Topic 2.7 SoundScope

Any periodic sound, in which the same waveform recurs cycle after cycle, can be synthesized by combining tones of different frequencies and amplitudes. The predecessor of TuneScope is a tool called SoundScope that can be used to combine tones of different frequencies and amplitudes. SoundScope can be accessed at:

https://maketolearn.org/soundscope/

The results can be visualized on a display, illustrating the waveform that is generated.

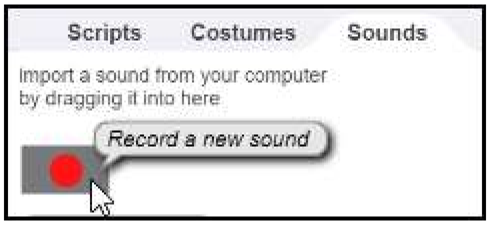


It is not necessary to understand the underlying nature of sound in order to create music with a computer, in the same way that it is not necessary to understand music theory to enjoy a concert. However, this understanding can enhance the appreciation of the music.

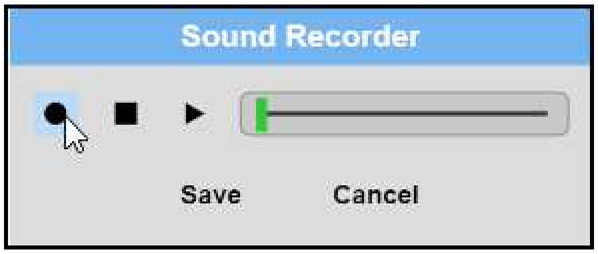
Topic 2.8 Digitized Sound

Synthesized sound generates sound artificially by using an algorithm to generate a series of numbers that become a waveform. Synthesized digital sound originates inside the computer. Digitized sound reverses the process, recording a natural sound occurring in the physical world outside the computer, converting electrical voltages into numbers inside the computer using an analog-to-digital (A / D) converter.

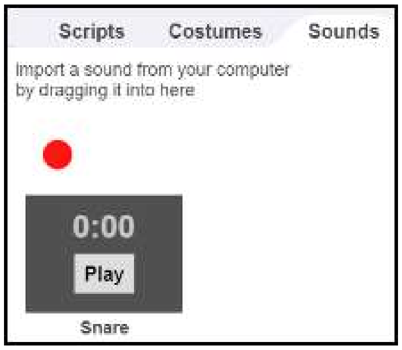
This can be done in Snap*!* by using the red *Record* button in the *Sounds* tab.



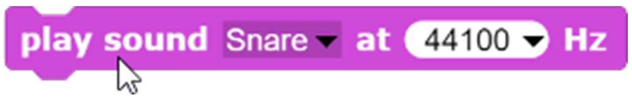
Clicking the red *Record* button accesses a *Sound Recorder* interface that can be used to record sounds.



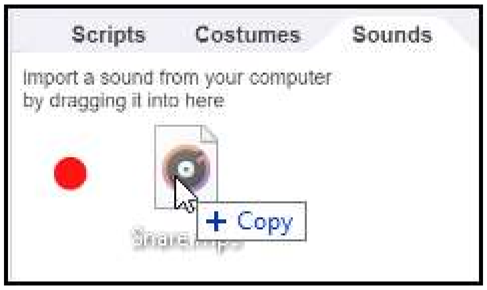
A recording of a musical instrument is known as a *sampled sound*. In the example below, a snare drum has been recorded.



Sounds recorded in this way can be replayed using the **Play Sound** block.



Sounds in ".wav" or ".mp3" format that are saved on the local computer can also be imported by dragging the file into the sound tab of Snap*!*



Conclusion

This chapter served as a further introduction to Snap*!* in the context of digitized and synthesized sound. The next chapter introduces TuneScope blocks that extend the music capabilities of Snap*!*, beginning with drums and rhythm.

Chapter 3. Musical Rhythms

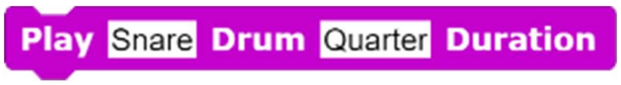
Glen Bull, Jo Watts, and Rachel Gibson

A drum track establishes the rhythm or beat for the music. For that reason, the drummer is often a key member of a band, establishing the tempo or rhythm that other musicians follow. A rotating object offers mechanical way of creating a pattern that repeats.

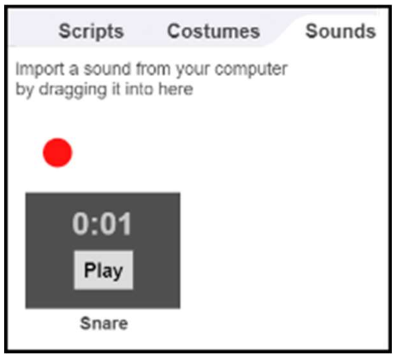
The San Francisco Exploratorium collaborated with the Lifelong Kindergarten group at M.I.T. to create a series of mechanical sound machines that could be programmed to produce repeating patterns in this way. A digital version of these rotating machines can be created in TuneScope.

Topic 3.1 Creating a Play Drum Block

In the previous chapter, sampled snare drum and bass drum sounds were played using the **Play Sound** block. These sampled sounds are reused in this chapter to create a **Play Drum** block.

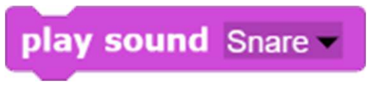


The **Play Drum** block uses the **Play Sound** block, found under the Snap*! Sound* palette to play a sampled drum sound. The ability to record and play a sampled sound was introduced in Chapter 1.

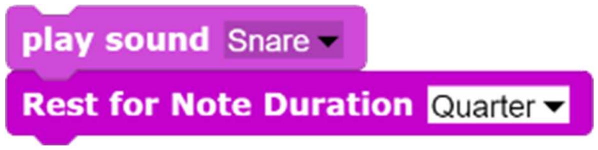


This capability has been used to import two sampled drum sounds, a snare drum and a bass drum, into Snap*!*.

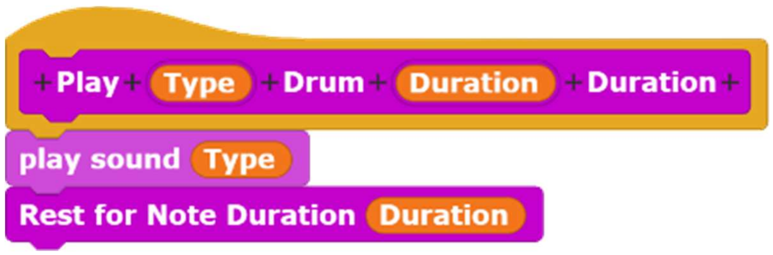
The **Play Sound** block can be used to play sampled sounds that have been imported into Snap*!* In the example below, it is being used to play sampled snare drum sound.



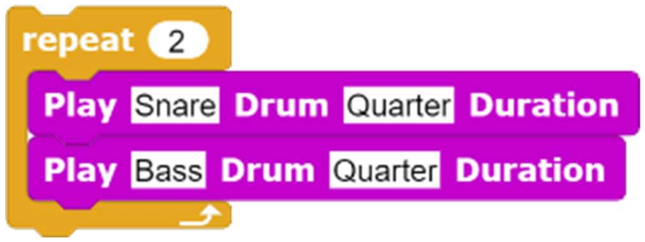
The Play Sound block can be combined with a block from the TuneScope music library. The TuneScope music library was introduced in the previous chapter. After the TuneScope library (accessed through the *Library* option under the Snap*!* file menu) is imported, this library of music extensions is found under a new palette titled *Music*. The **Rest for Note Duration** block can be combined with the **Play Sound** block to cause Snap*!* to wait for a specified number of beats before continuing to the next command.



These two blocks can them be used to create a new block, **Play Drum**. The inputs *Type* and *Duration* are dragged into the input slots for **Play Sound** and **Rest for Note Duration**.


+Play+ Type +Drum+ Duration +Duration+
play sound Type
Rest for Note Duration Duration

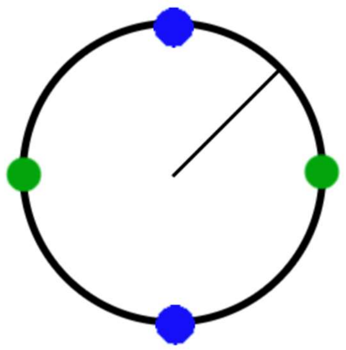
A drum rhythm can be created by using a **Repeat** block to create alternating snare drum and bass drum hits. Note that the inputs to Play Drum must be typed exactly or the block will not work. For example, if the sound is named “Snare” (with a capital “S”) but entered into the input as “snare” (with a lowercase “s”), nothing will happen. (Inputs for the duration can include Whole, Half, Quarter, Eighth, and Sixteenth).


repeat 2
Play Snare Drum Quarter Duration
Play Bass Drum Quarter Duration

The capability for creation of drum patterns can be used to create a display for visualizing rhythms graphically.

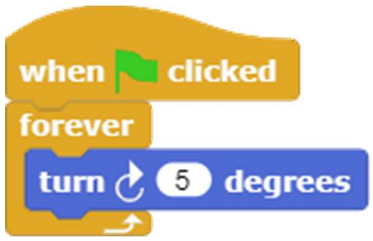
Topic 3.2 Visualizing Drum Patterns

The circle below can be visualized as the face of a clock. As the second Ahand sweeps around the circle, it plays a bass drum each time it strikes a blue dot and plays a snare drum sound each time it strikes a green dot.



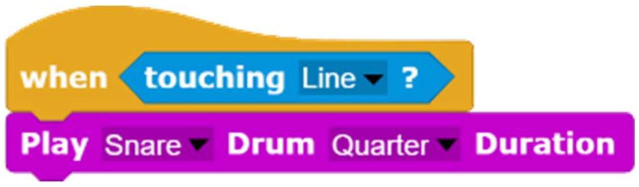
In music with a 4/4 time signature (i.e., four beats per measure) the drum sounds generated by the blue dots occur in time with notes that have the stronger emphasis in the music. That is, the first and third beats are accented.

In the previous chapter, a line that rotates around a circle was created. This line and circle are repurposed in this chapter to create a circular drum machine. The rotation of the line is accomplished by placing a **Turn** block inside a **Forever** loop.



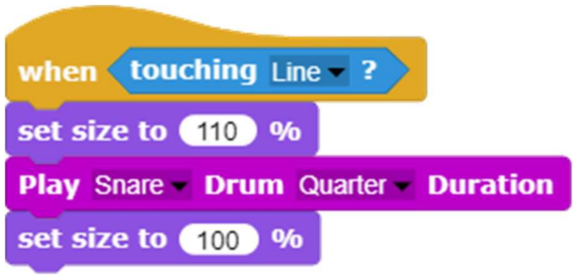
```
when [flag] clicked
forever
    turn ↻ (5) degrees
```

The **Point in Direction** block can be used to precisely position the line in a specified direction. Another sprite, such as the blue dot shown in the earlier illustration, can then be placed on the tip of the line. The code for the *Blue Dot* sprite plays a snare drum sound when it is touching the line.



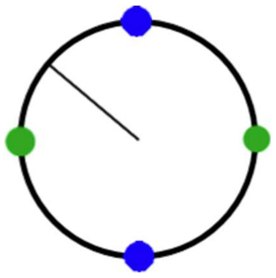
```
when <touching [Line ▾] ?>
Play [Snare ▾] Drum [Quarter ▾] Duration
```

The size of the blue dot can be increased slightly while the drum sound is playing to provide visual as well as auditory feedback.

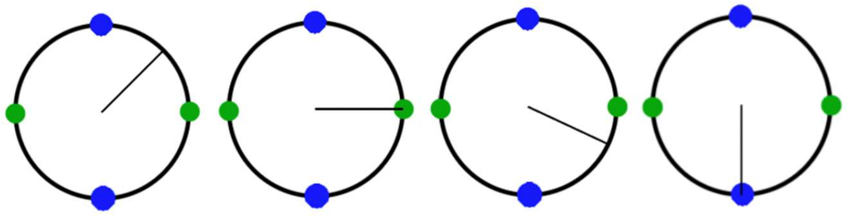


```
when <touching [Line v] ?>
set size to (110) %
Play [Snare v] Drum [Quarter v] Duration
set size to (100) %
```

A circle drawn through the dots can reinforce the concept of repeating circular motion. The circle does not affect sound production; rather, it is added as a visual aid. The blue sprites have been placed at 0 and 180 degrees in the illustration below and the green dots have been placed at 90 and 270 degrees. This placement produces the common four-beat drum pattern.



As the line sweeps around the circle, the specified drum sound will play as it touches each dot.



Exploration 3.2 Visualizing Drum Patterns

Continue to visualize patterns by adding additional circles. Try adding a larger circle that encapsulates the current one. Make the line longer so it reaches the new circle and plot more colored dots onto it, with each corresponding to a different drum sound.

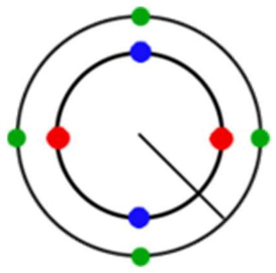
Topic 3.3 Exploring Drum Patterns with Circles

In music with a 4/4 time signature, the first beat of each measure usually coincides with the strongest accent in the melody. Therefore, a chord change often occurs at this point. Some genres, such as dance hall, reggae, and ska, reverse this pattern.

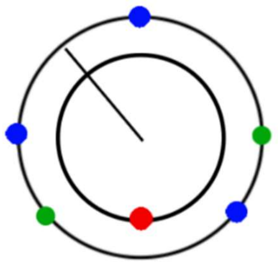
Within a rhythmic pattern, shifting the regular accents associated with the pattern is called "syncopation." Syncopated music can disrupt the listeners expectations and create a pleasing sensation when the rhythm resolves to its original pattern. This technique spread in America from gospel choirs, eventually finding its way into rock and roll, rockabilly, and funk.

The colored disks can be moved around the larger circle to create a variety of drum patterns. More circles can be added to make increasingly complex patterns that explore rhythms from around the world.

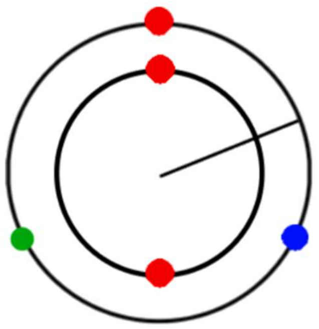
In this example, there are three different colored dots, each with a corresponding instrument. This visualization is a rhythm often found in Greek Hasapiko music.



This visualization is a Cuban cinquillo rhythm. Note that while most of the drums fall on the beat or off-beat, there are two that fall between beats.



This visualization displays a rhythm that is often found in traditional Persian music. This rhythm is played in 6/8 time. In 6/8 time, there are six eighth notes per measure. This allows the rhythm to be divided into both two and three beats.



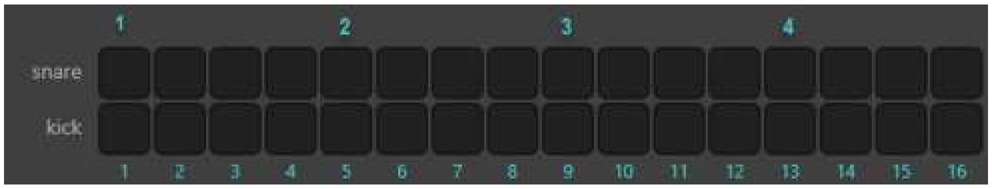
Exploration 3.3 Exploring Drum Patterns with the Rhythm Visualizer

Implement the different rhythms shown above. After watching John Varney's TED Ed Talk, pick out a few more rhythms from the video to implement, hear, and explore. Try to find other rhythms from around the world to try out on the musical circle.

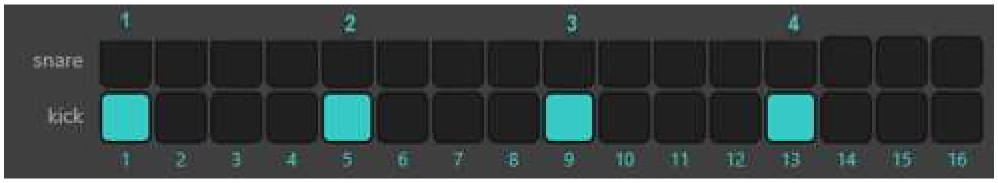
Topic 3.4 Creating a Drum Track

Many professional musicians and DJs use drum machines to build drum tracks. A drum machine is sometimes known as a *drum sequencer*. One type of interface for a drum machine resembles the illustration below.

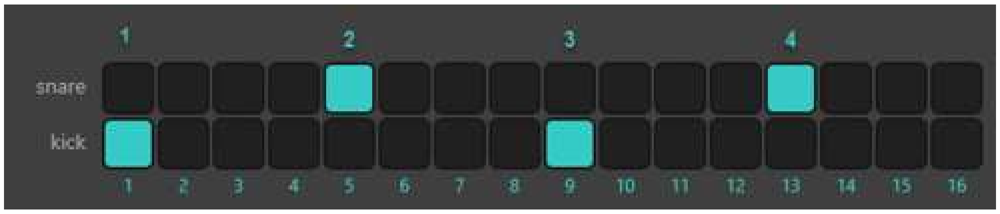
There are two rows of 16 squares. This particular drum machine is set to *sixteenth notes*, so there are 16 possible beats to be played. The top of the grid indicates where the *quarter notes* land within the measure. Drum machines are set up in a linear way, which means that it reads the grid from left to right.



To play the kick drum *on each quarter note* the squares at the first, second, third, and fourth quarter note location are turned on. This is shown in the example below. In this example, the kick drum plays on the first, second, third, and fourth quarter note of a measure.

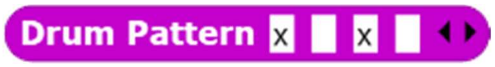


The following example illustrates a simple blues rhythm in which the kick drum is played on the *first* and *third* quarter notes, while the snare drum is played on the *second* and *fourth* quarter notes.

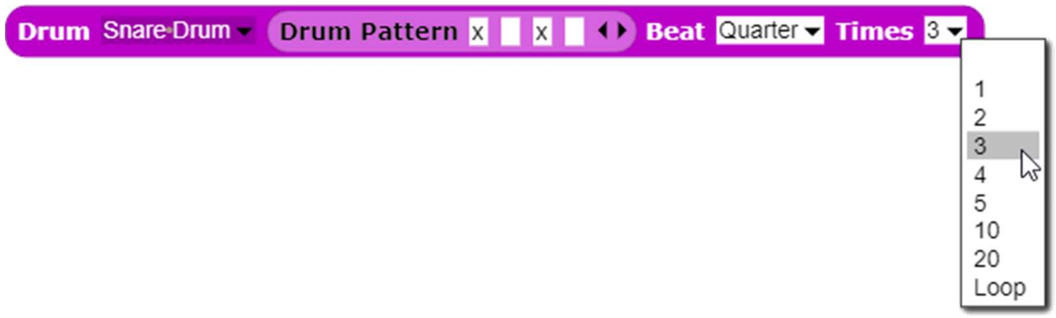


This is one of the most basic, yet foundational rhythms that nearly all other blues and rock rhythms are constructed from. In the topics that follow, a similar interface for a drum machine will be constructed using TuneScope music blocks in Snap*!*.

To simplify construction, a drum track with only four rather than sixteen slots is created in the example that follows. However, once the foundational principles are understood, a drum machine of any type or design can be constructed. In the block shown below, each of the four slots represents a quarter note. An "X" is placed in the slots where a drum beat will occur.

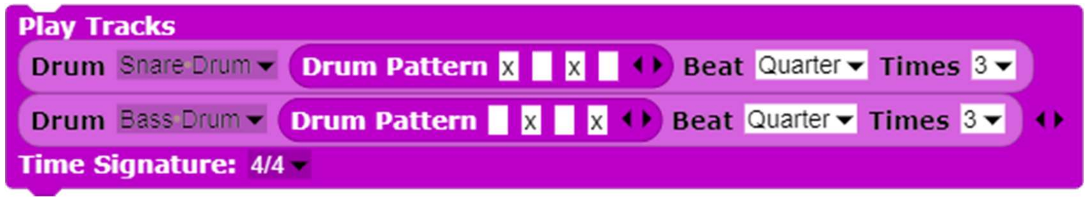


This list is supplied as an input to the **Drum** reporter block. Options allow the type of drum (Snare) and the duration (quarter note) to be selected. In addition, the number of times that the pattern is repeated can be selected.



If there is an additional music track for another instrument, such as a piano or guitar, a *looping* option can be selected that causes the drum pattern to repeat until the melody is completed. Other types of music tracks will be described in subsequent chapters. In addition to the options on the dropdown menu, a number can be manually typed into the *times* input slot.

The **Play Tracks** block can then be used to play the various drum tracks that have been constructed, in much the same way that a drum machine is used to construct similar patterns.


Play Tracks
Drum Snare Drum Drum Pattern x x Beat Quarter Times 3
Drum Bass Drum Drum Pattern x x Beat Quarter Times 3
Time Signature: 4/4

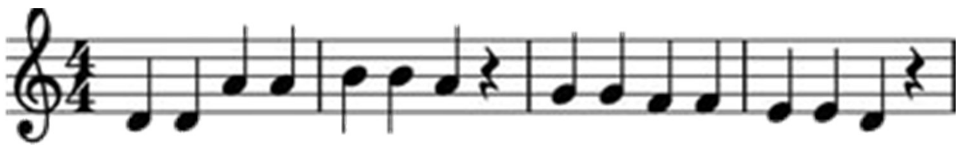
As noted above, the **Play Tracks** block can also be used to play tracks with other instruments such as piano tracks and guitar tracks (described in subsequent chapters).

Exploration 3.4 Creating Parallel Drum Tracks

Create a **Play Tracks** sequence that combines at least two drum patterns.

Topic 3.5 Musical Measures

Music consists of a series of notes played over a period of time. On a musical staff, the vertical axis represents pitch and the horizontal axis represents time. In this notation, notes are grouped together in *measures*. One measure consists of the notes between two vertical bars.

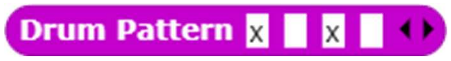


The number of notes in a measure is determined by the time signature. A time signature is written in the form of two numbers stacked on top of each other: $\frac{4}{4}$, $\frac{3}{4}$, etc. (pronounced "four four." "three four", etc.). The first number specifies the number of beats in a measure and the second number indicates the duration of the note that is counted as a beat. For example, a 4/4 time signature indicates that there are four beats per measure and that each beat is a quarter note.

Consequently, one measure in 4/4 time must equal a whole note. This can be satisfied by any combination of the following:

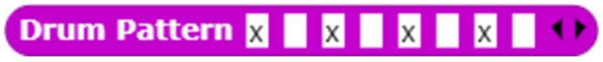
8 eighth notes
4 quarter notes
2 half notes
1 whole note

The **Drum Pattern** block can be used to represent four quarter notes in this manner.

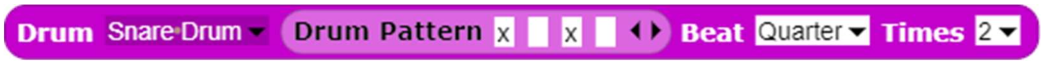


An "x" in a checkbox indicates that the drum is struck. An empty checkbox indicates that there is a wait. (A wait in which an instrument is silent is known as a *rest* in musical terms.)

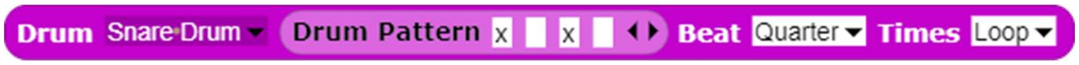
Similarly, eight eighth notes could be represented in this manner.



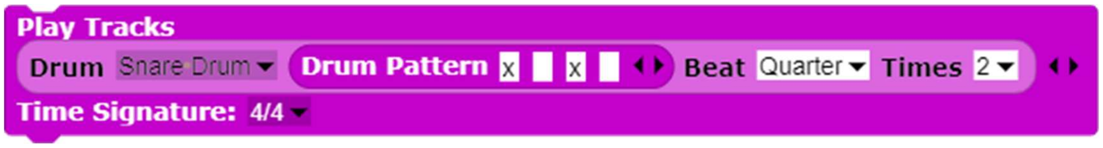
A pattern created in this manner can be placed in a Drum block that specifies the length of each note (quarter note in the example below) and the number of times that the pattern is repeated.



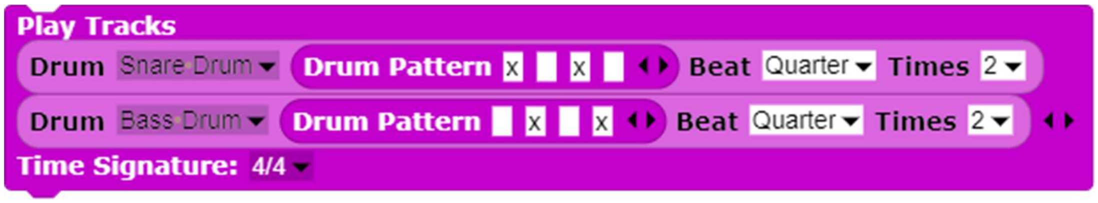
One option in the dropdown menu is to have the drum pattern loop indefinitely.



The drum pattern can be then played using the **Play Tracks** block. (Note: the first time this command is run, there will be an initial delay of several seconds while TuneScope's drum samples are loaded into Snap*!*

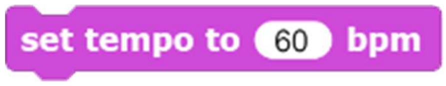


In this example, the snare drum and the bass drum are alternating.


Play Tracks
Drum Snare Drum Drum Pattern x x Beat Quarter Times 2
Drum Bass Drum Drum Pattern x x Beat Quarter Times 2
Time Signature: 4/4

The beat established by the drum track enables the other members of the band to play in time so that the notes played by the various instruments are aligned.

Tempo is the other major factor that controls timing. Tempo is expressed in *beats per minute*. For example, a tempo of 60 beats per minute is equivalent to one beat per second. In Snap*!*, the tempo is controlled by the **Set Tempo** block (found under the *Sound* palette).



Topic 3.6 Exploring Rhythms across Cultures

The pattern of *Bass Drum – Snare – Bass Drum – Snare* is one of the most basic drum patterns novices learn. This pattern forms the foundation for other more advanced patterns. Try doubling a drum beat using eighth notes instead of quarter notes. Try shifting a beat slightly or leaving it out completely. How does this change the feel of the rhythm? How does the experience of creating patterns with a linear drum track differ from the experience of creating a circular drum pattern?

Drum tracks can be used to explore rhythms from different genres of music, such as the Cuban cinquillo drum pattern and Persian drum pattern described above. Try exploring drum patterns from other cultures and create variants that you develop. Once you have created a basic pattern with two drum tracks, try adding a cymbal or a hi-hat as a third track. Cymbals and hi-hats are often used to provide additional ornamentation to a drum pattern. Explore placement at different places in the pattern to see what sounds best.

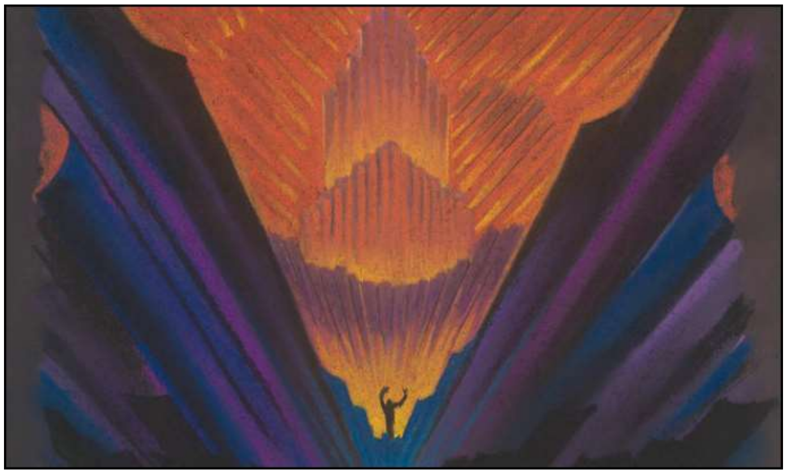
Exploration 3.6 Exploring Rhythms across Cultures

Construct and play the rhythms described above. Then identify other rhythms from around the world and create drum tracks to play them.

Chapter 4. Combining Art and Music

Glen Bull, Jo Watts, and Rachel Gibson

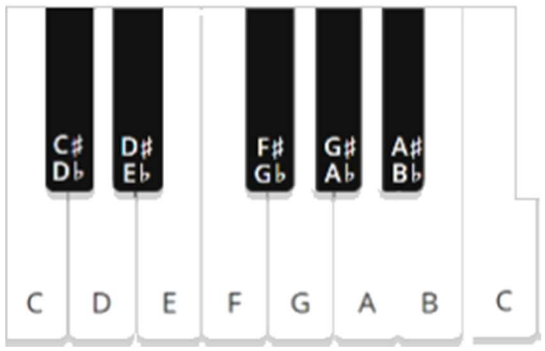
The film *Fantasia* begins with a series of abstract swirling images with a conductor leading an orchestra playing Bach's *Toccata and Fugue in D Minor*. Animated shapes and abstract images move in time to the music.



Music blocks in the TuneScope library extend the capabilities of Snap! that makes it possible to create the equivalent of a code-based Digital Audio Workstation (DAW) that can be used to compose and play music. For those interested in the technical details, TuneScope music blocks uses the W3C Web Audio application program interface (API) to generate musical notes and instruments.

Topic 4.1 The Building Blocks of Music

An understanding of the basic building blocks of music is helpful in creating music in TuneScope. The Western chromatic musical scale consists of twelve notes. On a piano, these notes consist of seven white keys and five black keys. Pressing a piano key causes a piano string to vibrate. For example, the note *C* in the middle of the piano keyboard vibrates at a rate of approximately 261 times per second. Because this note is near the middle of the piano keyboard, it is called *Middle C*.

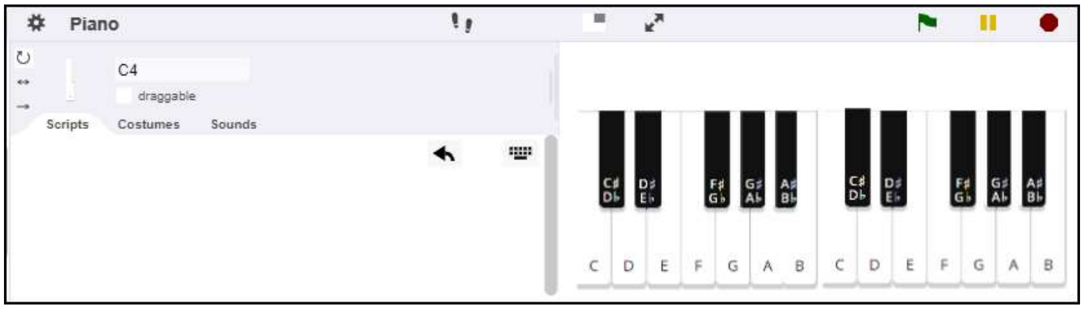


Higher rates of vibration are perceived as higher pitches. A span of notes on the piano keyboard that begins with one note (such as the note "C" in the illustration above) and ends in the same note name is known as an octave. The octave that begins with middle C is in the fourth octave (counting from the left) on the keyboard. This can also be written "C4" (where "C" refers to the note and "4" refers to the fourth octave).

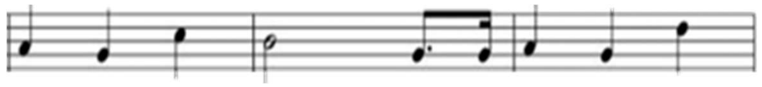
A piano keyboard that can be used to explore octaves can be accessed at:

https://tunescope.org/index.html#project:Username=maketolearn&ProjectName=Piano

Pressing a piano key on the computer screen plays the note associated with that key.



When written, notes usually appear on a series of five parallel horizontal lines, known as the *staff*. Notes are grouped into *measures*. A measure consists of the notes between two vertical lines on the musical staff.



As noted in the previous chapter, the *time signature* is the convention used in Western musical notation to signify the number of beats in a measure (also referred to as a musical *bar*) and to describe which note value is counted as a beat. The time signature is written in the form of two numbers stacked on top of each other: $\frac{4}{4}$, $\frac{3}{4}$, etc. (These are pronounced “four four.” “three four”, etc.) The first number specifies the number of beats in a measure and the second number indicates the duration of the note that is counted as a beat. For example, a 4/4 time signature indicates that there are four beats per measure and that each beat is a quarter note. A 3/4 time signature indicates that there are three beats per measure and that each beat is a quarter note. A 6/8 time signature indicates that there are six beats per measure and that each beat is an eighth note. A 4/4 time signature is sometimes described as *common time* because it is the most commonly used time signature.

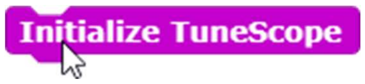
Exploration 4.1 The Building Blocks of Music

Use the piano keyboard created with sprites to explore the relationship of notes within the fourth and fifth octaves of the chromatic scale.

4.2 Musical Notes

In the previous chapter, sound samples for a snare drum and a bass drum were imported into Snap*!* TuneScope uses sound samples in a similar way. However, TuneScope loads sound samples for a much broader range of instruments into Snap*!*

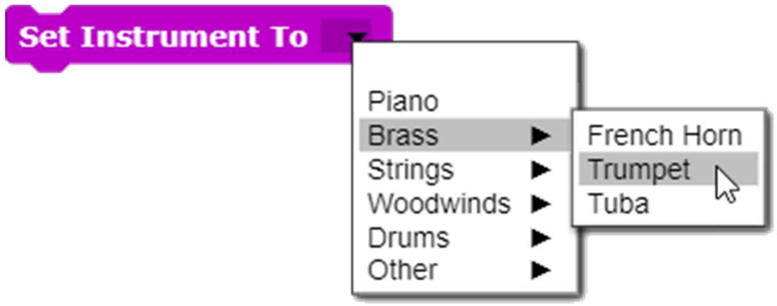
Important Note: Before the sound samples for musical instruments can be used, TuneScope should be initialized.



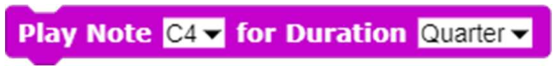
The process of initialization loads the sound samples for the instruments. This typically takes several seconds. After the sound samples for all of the musical instruments have been loaded into Snap*!*, a message will appear that confirms that TuneScope has been initialized:

It is only necessary to initialize TuneScope and load the sound samples once for each Snap*!* session.

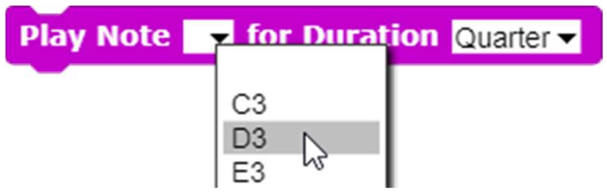
TuneScope music blocks access sampled instruments that can be used to play musical notes. These musical instruments are selected through the **Set Instrument** code block. This code block enables a musical instrument to be selected from a dropdown menu.



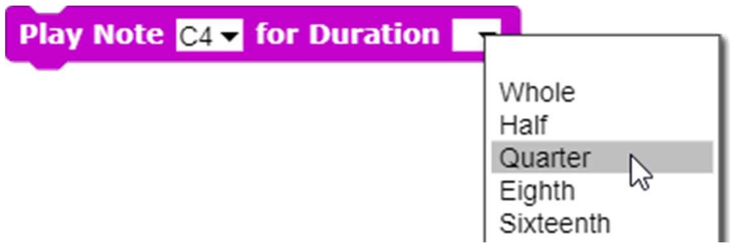
Once an instrument has been selected, the **Play Note** code block can be used to play a musical note.



A dropdown menu in the **Play Note** code block can be used to select musical notes. (Note designations can also be typed directly into this input slot.)



A second dropdown menu is used to select the duration of the note. The note duration determines the length of time that the note is played.



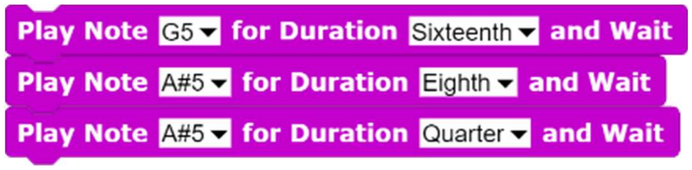
In much of western music, a *quarter note* is considered to be one beat. (Why is one beat not a *whole note* rather than a quarter note? Because in 4/4 time, there are four beats per measure; a whole note is one *measure.)* Moving up the menu doubles the number of beats in the note. Moving down the menu halves the length. For example, a half note is two beats, while a whole note is four beats. In the other direction, an eighth note is half a beat. Dotted notes equal the duration of the named note plus the duration of the next shorter note. For example, a dotted half note would be equal in duration to a half note plus a quarter note, or three beats. Triplet notes divide the duration of two of the named notes into three equal durations. For example, a quarter note triplet would be three evenly timed notes played within the normal duration of two quarter notes.

Exploration 4.2 Musical Notes

Explore the voices of different musical instruments that can be accessed through the **Set Instrument** code block. Use the **Play Note** code block to play back notes using different instruments.

Topic 4.3 Combining Notes

The **Play Note and Wait** code block waits until one note is completed before beginning the next note. Several **Play Note and Wait** code blocks can be combined to play a series of notes. For example, the opening notes of the blues song *Crossroads* are G5, A#5, and A#5



```
Play Note G5 for Duration Sixteenth and Wait
Play Note A#5 for Duration Eighth and Wait
Play Note A#5 for Duration Quarter and Wait
```

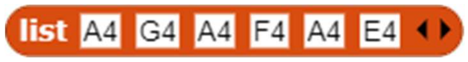
The first note (G5) lasts for a sixteenth of a measure, the second note (A#5) lasts for one-eighth measure, and the third note (A#5) lasts for a quarter measure.

Exploration 4.3 Combining Notes

Create your own three-note sequence in TuneScope using the **Play** code block. You can either use notes from a favorite song or create your own combination of notes.

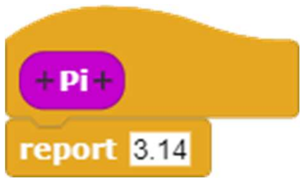
Topic 4.4 Playing a List of Musical Notes

Since the music visualization in *Fantasia* begins with *Toccata and Fugue in D Minor*, this seems like an appropriate place to begin with musical visualization in Snap*!* A list of notes can be stored in a list:



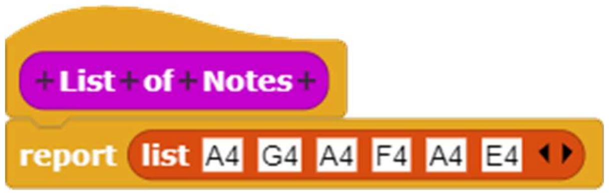
```
list A4 G4 A4 F4 A4 E4
```

Most computing languages have a mechanism for storing both variables and constants. *Variables* are program elements that may need to change under program control – the length of a side in a polygon, for example. *Constants* are elements that will not change – the value of pi for example. In Snap*!* a reporter block can be used to store elements that will not need to change under program control.



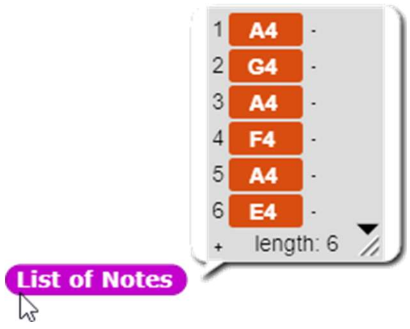
```
+Pi+
report 3.14
```

A reporter block can also be used to store a list of notes.

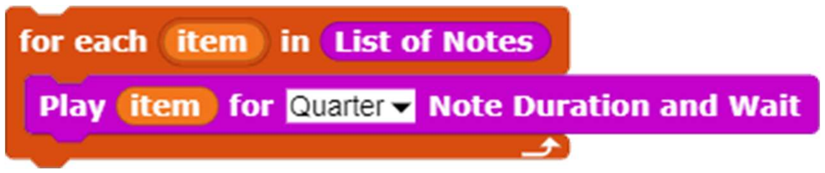


```
+List+of+Notes+
report list A4 G4 A4 F4 A4 E4
```

The reporter block can then be used to retrieve the list of notes when needed:

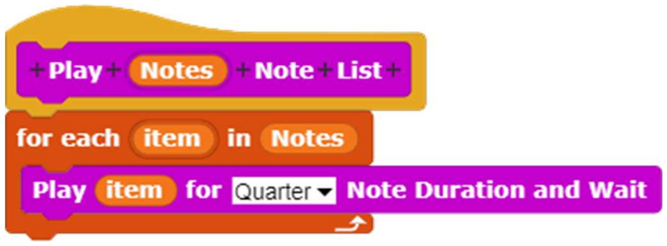


A loop can then be used to play each note in the list.



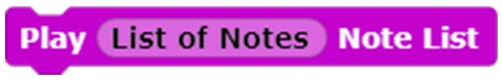
```
for each item in List of Notes
    Play item for Quarter Note Duration and Wait
```

Storing a list of notes has two benefits: (1) The notes for all of the songs created in this way can be stored in a separate user-created category such as Songs in the blocks palette. (2) The reporter blocks containing the songs can be exported and shared with others. This method could be encapsulated in the form of a custom procedure.



```
+Play+ Notes +Note+List+
for each item in Notes
    Play item for Quarter Note Duration and Wait
```

This custom block can then be used to play any list of notes.



```
Play List of Notes Note List
```

Exploration 4.4 Playing a List of Musical Notes

Create a list of notes. Then use the **Play Note List** code block to play the list of notes.

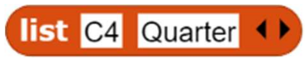
Topic 4.5 Playing a Musical Motif

A musical motif is a recurring musical theme. For example, in the movie *Jaws* the alternating notes of E and F create a sense of unease. Characters in a play or film are sometimes given a musical motif that becomes their musical identity. The opening notes of the Imperial March are played when Darth Vader appears in *Star Wars*.

Each note in a sequence of notes has a specific duration associated with it. For example, the durations associated with the opening notes of *Over the Rainbow* are listed in the table below.

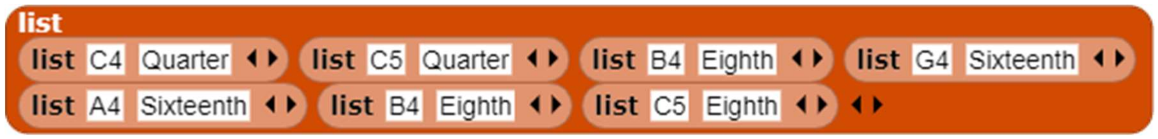
Over the Rainbow	
Note	*Duration*
C4	Quarter
C5	Quarter
B4	Eighth
G4	Sixteenth
A4	Sixteenth
B4	Eighth
C5	Eighth

In the previous section, a series of notes were stored in a list. However, a duration also needs to be associated with each note. Each note and duration can be described in a list consisting of two items: the pitch of the note (*C4*) and the duration of the note (*Quarter*).



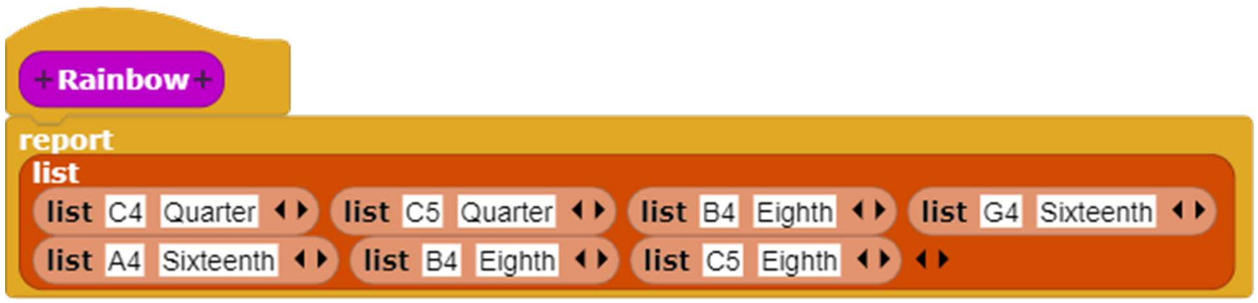
```
list C4 Quarter ◂▸
```

Each *note* consisting of a *pitch* and a *duration* can then be stored in a list of lists in the following manner.



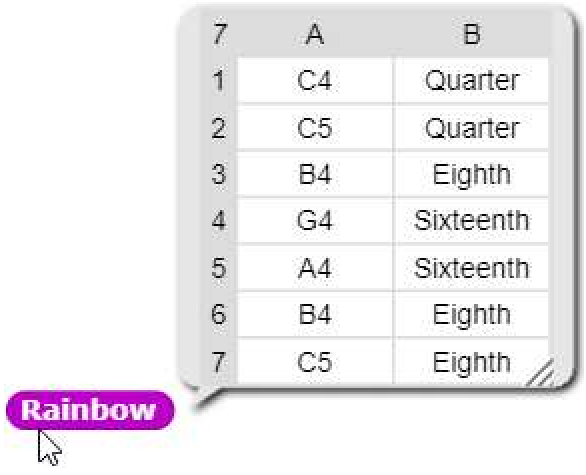
```
list
  list C4 Quarter ◂▸  list C5 Quarter ◂▸  list B4 Eighth ◂▸  list G4 Sixteenth ◂▸
  list A4 Sixteenth ◂▸  list B4 Eighth ◂▸  list C5 Eighth ◂▸  ◂▸
```

The opening notes of *Over the Rainbow* can then be placed in a reporter block.



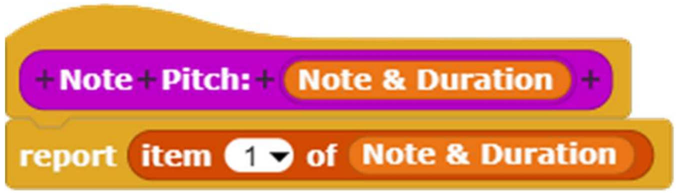
```
+Rainbow+
report
  list
    list C4 Quarter ◂▸  list C5 Quarter ◂▸  list B4 Eighth ◂▸  list G4 Sixteenth ◂▸
    list A4 Sixteenth ◂▸  list B4 Eighth ◂▸  list C5 Eighth ◂▸  ◂▸
```

The reporter will return the lists of notes and durations.



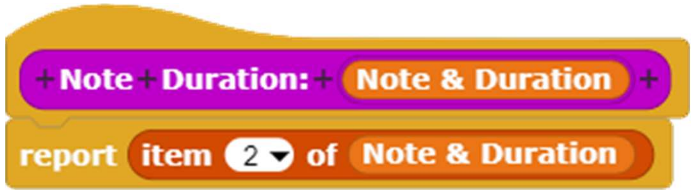
7	A	B
1	C4	Quarter
2	C5	Quarter
3	B4	Eighth
4	G4	Sixteenth
5	A4	Sixteenth
6	B4	Eighth
7	C5	Eighth

After developing a structure for recording the notes and durations of a musical motif, some tools for retrieving this information will be useful. The reporter block shown below retrieves the first item (i.e., the note pitch) of a combined "Note & Duration" pair.



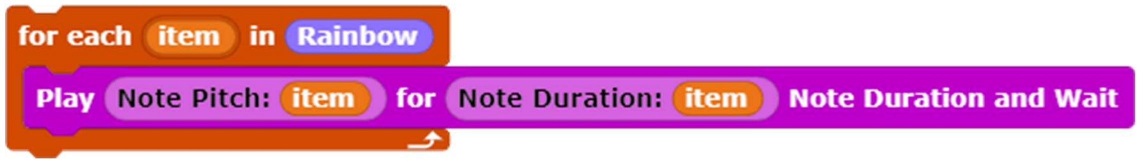
```
+Note+Pitch:+ (Note & Duration) +
report item (1 ▾) of (Note & Duration)
```

A parallel reporter block retrieves the second item (i.e., the note duration) of a combined "Note & Duration" pair. Associating details such as *Item 1* or *Item 2* with meaningful names reduces the programmer's cognitive load.



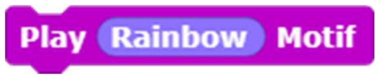
```
+Note+Duration:+ (Note & Duration) +
report item (2 ▾) of (Note & Duration)
```

These tools can be used to retrieve the Note Pitch and Note Duration for each item in the list of opening notes of *Over the Rainbow*.



```
for each (item) in (Rainbow)
    Play (Note Pitch: (item)) for (Note Duration: (item)) Note Duration and Wait
```

The **Play Motif** block works through the list of notes in the motif until it reaches the end of the list.



```
Play (Rainbow) Motif
```

The list of notes and durations could have been stored in two separate lists, such as a list of "Rainbow Pitches" and a list of "Rainbow Durations" rather than as a combined list of notes and durations. This approach is certainly viable, but has two potential drawbacks:

1. One potential drawback is that the number of lists needed is doubled. That is not an issue for a single song. However, we have provided sample opening note sequences for a dozen songs (one for each of twelve musical intervals). For this many songs, it is more convenient to represent each song with a single list.

2. We have also found through experience that when there are separate note and duration lists for each song, inevitably the notes and durations become misaligned or mismatched. When the two elements are combined in a single list, it reduces the potential for errors of this kind.

The drawback of a combined list is that there is slightly more overhead in creating tools to retrieve the pitches and durations. However, these tools only need to be created one time (as shown above).

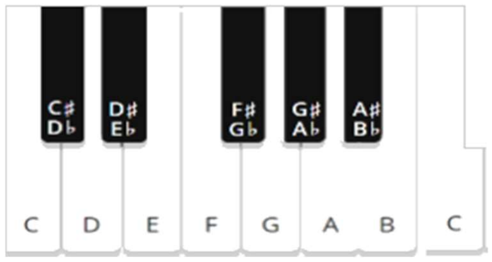
Exploration 4.5 Playing Musical Motifs

Play a musical motif from the list of four note sequences provided. Experiment with adapting or revising one of the sequences.

Topic 4.6 Creating a Musical Motif

Some background on sound and music theory is useful for music composition. In this case, a four-note sequence will be created for use as a musical motif. However, the concepts described are applicable to composition of music of any length.

When a guitar string is plucked, the string moves back and forth as it vibrates. The rate of vibration is known as the frequency. Each note has a characteristic frequency. When two notes are played, the ratio of the frequency of the first note to the frequency of the second note determines whether the combination is perceived as harmonious. Combinations of notes whose ratios of frequencies can be approximated by low whole number digits such as 3:2, 4:3, and 5:4 are generally perceived as more harmonious. This relationship was first observed in ancient times but contemporary songwriters make use of this relationship today.



The fundamental frequency of the note C4 is approximately 264 Hz. The note C5, which begins the next octave, is double the frequency of C4. The ratio of C5 to C4 is 2:1 and therefore is perceived as harmonious.

The span between C4 and C5 is divided into twelve intervals with seven white keys (C, D, E, F, G, A, and B) and five black keys (known as sharps and flats). An interval of four piano keys between musical notes yields a ratio of 5:4 and therefore is perceived as harmonious. Similarly an interval of five piano keys between musical notes has a ratio of 4:3 and also is perceived as harmonious.

However, an interval of one piano key beween notes has a ratio of 25:24 and is generally perceived as inharmonious.

The table below lists songs that begin with each of the twelve musical intervals that are possible in the Western chromatic scale. As long as the intervals between respective piano keys remain the same, the first note can begin on any piano key. To simplify illustration of these relationships, the notes in the songs below have been shifted so that the opening note always begins with “C”.

Songs that Begin with Each of Twelve Intervals						
Interval			*Song*	*Opening Notes*	*Harmonious?*	
1	C	C#	Jaws	C C# C C#	No	
2	C	D	It’s My Life	C D C D C D C A G A	-	
3	C	Eb	Smoke on the Water	C Eb F C Eb F# F	-	
4	C	E	Saints Go Marching In	C E F G C E F G	Yes	5:4
5	C	F	Here Comes the Bride	C F F F	Yes	4:3
6	C	F#	Maria (West Side Story)	C F# G	No	
7	C	G	Star Wars Theme	C G F E D C G	Yes	3:2
8	C	G#	Go Down Moses	C A A G G A A F	No	
9	C	A	My Way	C A C A G A	Yes	5:3
10	C	Bb	Somewhere	C Bb B F# D#	No	
11	C	B	Star Trek Theme	C B A# G# F# F E	No	
12	C	C	Over the Rainbow	C4 C5 B G A B C	Yes	2:1

In constructing this table, we found it easier to find well-known examples of songs that begin with intervals that are perceived as harmonious. The composer Leonard Bernstein liked the intervals of 6 and 10, creating the songs *Maria* (which begins with an interval of 6) and the song *Somewhere* (which begins with an interval of 10) for the musical play *West Side Story*. However, other examples of songs with these intervals often perceived as less harmonious did not immediately come to mind as we considered possible examples for inclusion in the table.

To create a harmonious motif, combine sequences of notes that have harmonious intervals between the notes. These are notes sequences that have frequencies in ratios of low whole numbers. To create a dissonant motif (for example, for a villain) combine sequences of notes that do not have harmonious intervals. Engaging music shifts between harmonious and dissonant movements. The tension between harmonious and dissonant sounds animates many songs.

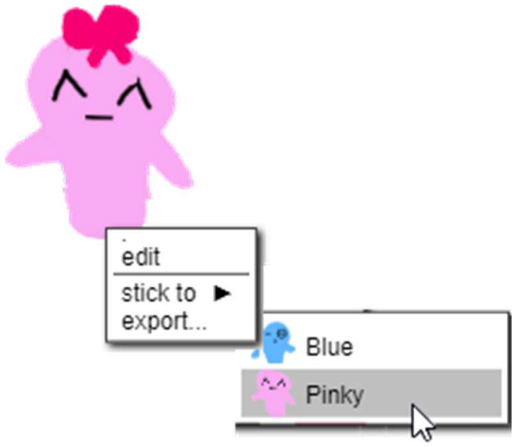
Exploration 4.6 Creating Musical Motifs

Create a harmonious musical motif (i.e., a four or five-note sequence). Then create a dissonant motif. Experiment with the effect of assigning different durations to the notes selected.

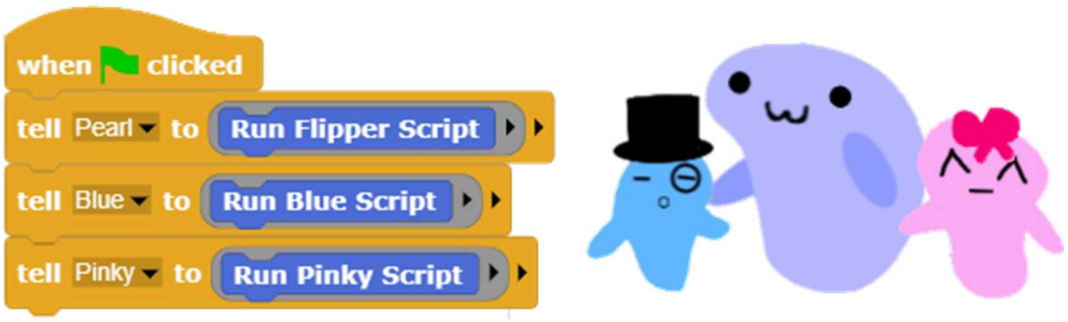
Topic 4.7 Creating an Animated Character

Rachel Gibson created the trio of characters below in the paint editor. The top hat of the first character, eyes of the middle character, and the red bow of the third character were constructed as separate sprites.

A drop-down menu allows a selected item (such as the red bow) to be attached to the character. The item can then be animated separately, but can move with the character as it shifts about the stage.

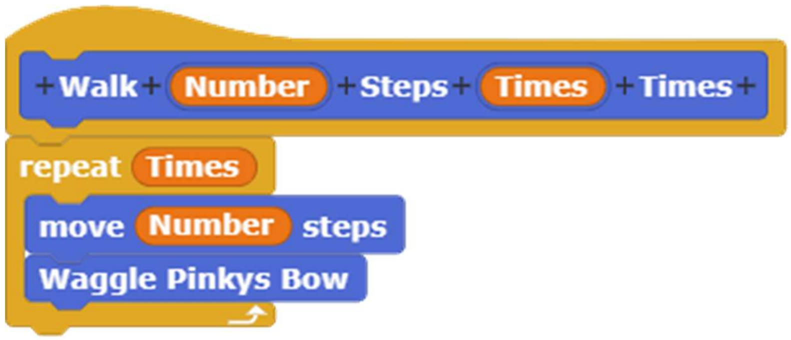


The directions for all three characters are stored in the script area associated with the stage.

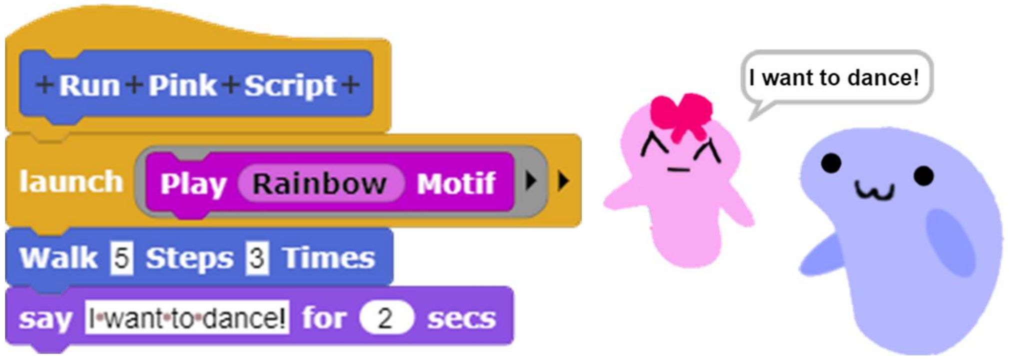


```
when [flag] clicked
tell Pearl ▾ to (Run Flipper Script) ▸
tell Blue ▾ to (Run Blue Script) ▸
tell Pinky ▾ to (Run Pinky Script) ▸
```

The **Tell** command is then used to tell each character to run its script. Pink's script directs Pinky to walk a specified distance across the stage. Pinky's block repeats a **Move** command for a specified number of times. Each time Pinky moves, her bow waggles. This is accomplished by rocking the bow clockwise and then back counter-clockwise after a short delay.



Pinky's script plays her musical motif and then directs her to walk across the stage. A **Launch** block is used to execute the **Play Motif** block. This causes the notes to be played in parallel with the subsequent actions. After walking across the stage, the **Say** code block causes Pinky to say, "I want to dance!" in words that appear in a speech bubble beside the character.

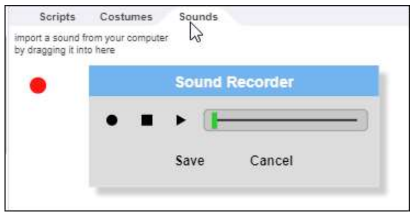


Exploration 4.7 Creating an Animated Character

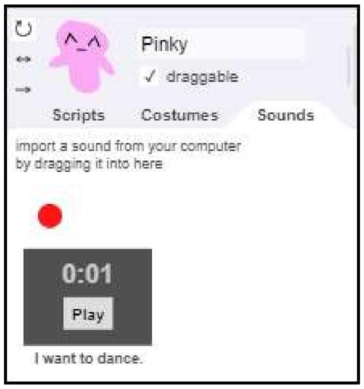
> Create an animated character with an attached element that is animated separately as the character moves across the stage. Play an associated musical motif at the beginning of the character's action.

Topic 4.8 Recording Speech

The computer's microphone can be used to capture a sample of a naturally occurring sound. A sound recorder in Snap*!* is accessed through the *Sounds* tab to the right of the *Scripts* tab. (Note: on a Macintosh computer, you will be asked to give permission to access the computer's microphone the first time the sound recording capability is accessed.)



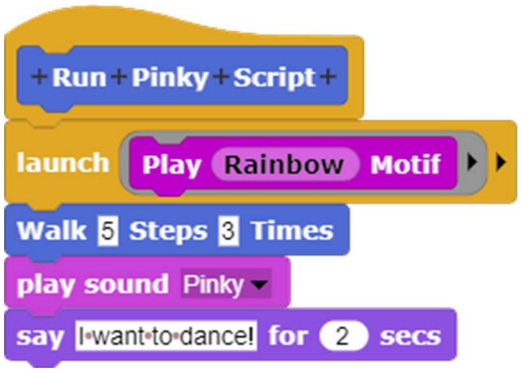
Once a sound has been recorded, it can be assigned any name that is appropriate. In this instance, the words "I want to dance." were recorded.



Pinky's script can now can be updated to include audio of the words "I want to dance!" to accompany the speech bubble that appears on the screen. The words appear to be presented in the speech bubble at the same time as the audio because once the audio is initiated the script does not wait before going on to present the text in the speech bubble beside the character. The Play Note and Play Sound blocks have two forms:

1. One form in which the block is initiated and then continues to the next block in the script without waiting and
2. A second form in which the block is initiated but then waits until the sound or note is completed before continuing.

The **Play Motif** block uses the **Play Note and Wait** form so that the notes are played sequentially (rather than in parallel, to create a chord). Consequently, the **Launch** block is needed so that once the **Play Motif** block is initiated, the script will continue to the next command (**Walk**) in the script. However, since **Play Sound** is used (rather than **Play Sound and Wait**), the **Launch** block is not needed; the script continues to the next block without waiting once the sound is initiated.



Exploration 6.8 Recording Speech

Use the Snap*! Sound Recorder* to record a phrase that is added to a character's script.

Chapter 5. Musical Chords

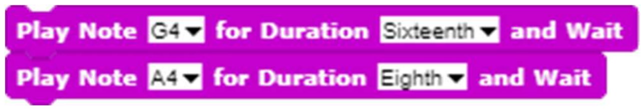
Glen Bull, Jo Watts, Rachel Gibson, and Luke Dahl

The melody of a song consists of a sequential series of musical notes played one after another. A musical chord consists of several notes played together. A repeated progression of chords often is used as a backing track that repeats as the melody is played over it. This pattern, known as a chord progression, creates a sense of movement.

Some instruments, such as the piano, allow the musician playing it to play chords with one hand and melody with another. With other instruments, such as the guitar, it can be difficult to play both chords and melody at the same time. Therefore, many bands employ two guitarists, one to play chords and another to play melodies. Other instruments, like brass and woodwinds, are physically incapable of playing more than one note at a time, making playing chords on these instruments impossible. A computer can be programmed to play chords and melodies, including ones that would not be possible on a physical instrument.

Topic 5.1 Parallel and Sequential Notes

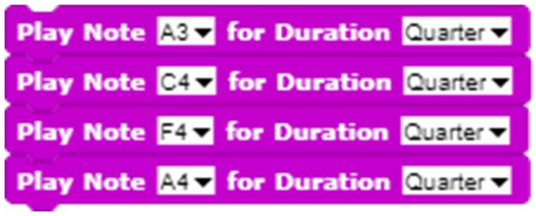
The **Play Note and Wait** block waits until one note ends before beginning the next.



```
Play Note G4 for Duration Sixteenth and Wait
Play Note A4 for Duration Eighth and Wait
```

In the illustration above, the note G4 is played for one-sixteenth of a beat, followed by A4 for an eighth of a beat. On a piano, the first key would be struck, and then the musician would wait for one-sixteenth of a beat before striking the next piano key.

In contrast, a chord consists of several notes played together at the same time. The **Play Note** code block does not wait until one note ends before beginning the next note.



```
Play Note A3 for Duration Quarter
Play Note C4 for Duration Quarter
Play Note F4 for Duration Quarter
Play Note A4 for Duration Quarter
```

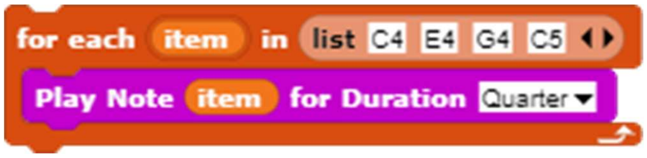
On a piano, the musician would strike all four piano keys corresponding to the notes A3, C3, F3, and A4 at the same time. In previous modules, notes that sound good in combination with another were identified. These notes are often combined into subgroups within the twelve-note chromatic scale, forming subsets of the chromatic scales such as the major scales and the pentatonic scales. Combinations of notes that sound pleasing when played together are also often combined to form chords.

Exploration 5.1 Play Notes

Use the **Play Note** code block to create combinations of notes that sound pleasing. Identify two or three combinations of notes that could be used to form chords.

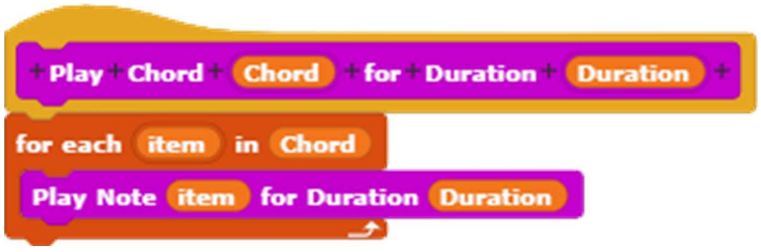
Topic 5.2 Play Chord Block

In a previous module, the **Play Motif** block was developed to play a sequence of notes in a list. This same method can be adapted to play a chord by replacing **Play Note and Wait** with **Play Note**.



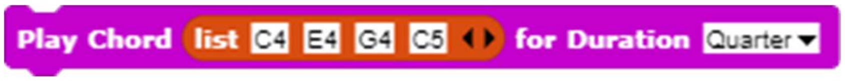
```
for each item in list C4 E4 G4 C5
    Play Note item for Duration Quarter
```

This method can be used as the basis for a **Play Chord** block.



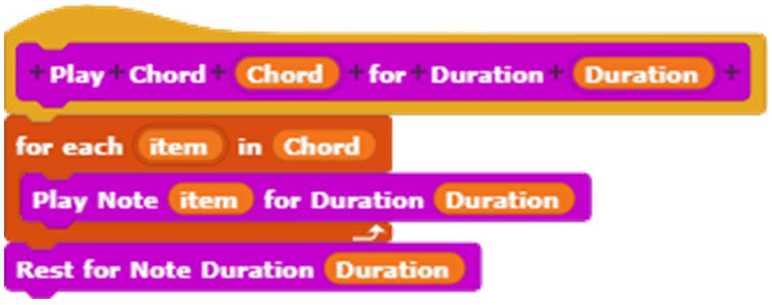
```
+Play+Chord+ Chord +for+Duration+ Duration +
for each item in Chord
    Play Note item for Duration Duration
```

The list of notes that form the chord are provided as an input to the **Play Chord** code block and play the chord for the duration specified.



```
Play Chord list C4 E4 G4 C5 for Duration Quarter
```

The **Play Chord** block should wait until the chord is completed before continuing to the next chord. This can be accomplished by adding a **Rest** block at the end of the procedure.



```
+Play+Chord+ Chord +for+Duration+ Duration +
for each item in Chord
    Play Note item for Duration Duration
Rest for Note Duration Duration
```

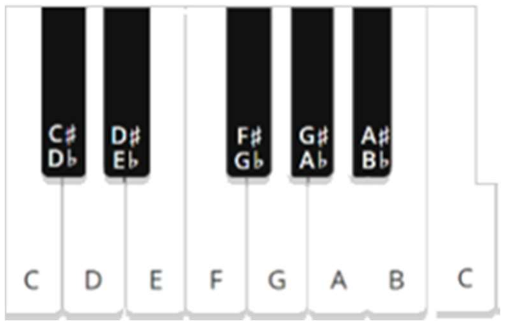
Exploration 5.2 Play Chord Block

Identify a sequence of notes that sounds pleasing when played in combination to form a chord.

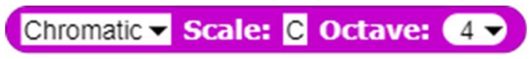
Topic 5.3 Musical Scales

The term *musical scale* refers to a sequence of notes ordered by frequency or pitch. Many different musical scales have been developed throughout history. The Western chromatic musical scale consists of twelve notes. Other scales have been developed that employ different numbers of notes.

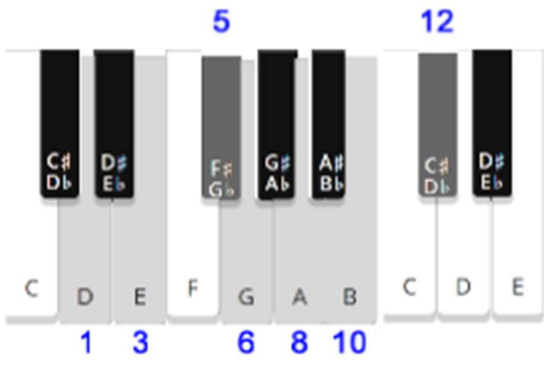
The concept of an octave was introduced in a previous chapter, *Combining Art and Music*. An octave is a span of notes on the piano keyboard that begins with one note (such as the note "C" in the illustration below) and ends in the same note one octave higher (e.g., from C4 to C5).



A custom **Chromatic Scale** code block reports the twelve notes of the chromatic scale.

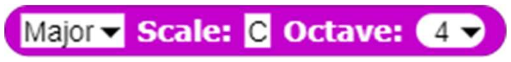


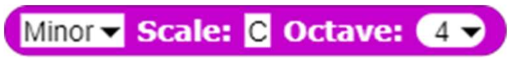
Many other scales have been formed that are subsets of the full twelve notes of the chromatic scale. For example, the major scale is formed using the seven most harmonious notes of the chromatic scale. There are twelve major scales, one for each note in the chromatic scale. The notes for each major scale are always drawn from the same relative position in the chromatic scale. For example, the notes that form the D Major scale are drawn from the same relative positions of the chromatic scale. This scale consists of the notes numbered 1, 3, 5, 6, 8, 10, 12 (beginning with D) in the illustration below.



Much of the music in medieval times was designed to create music that sounds harmonious (i.e., heavenly). However, dissonance is also an important tool in composition of music. The blues, for example, make use of dissonance. The *minor scales* are perceived as more dissonant than the major scales. Minor scales are formed from the first, third, fourth, sixth, eighth, nineth, and eleventh notes of the chromatic scale.

The drop-down menu on the Scale reporter block can be used to find the notes in major (and minor) scales beginning with any note.





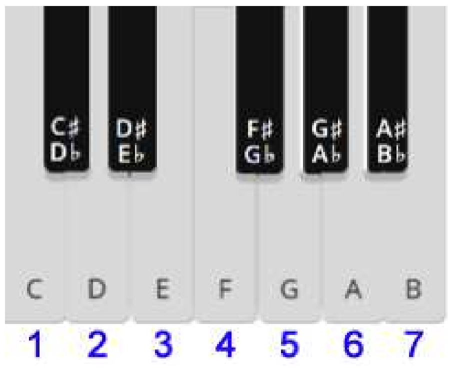
For those who are interested, details of how the **Scale** code block is constructed are provided in the appendix of this module, along with instructions on how to create other scales.

Exploration 5.3 Major Scales

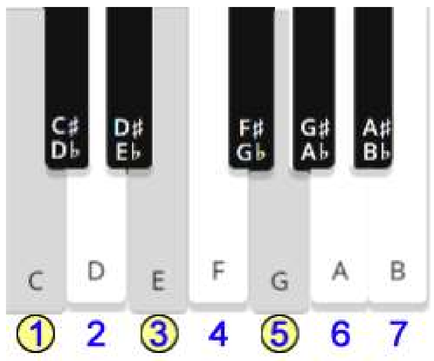
Create a custom code block that plays the first, third, and fifth note of a major scale. For example, in the C Major scale, the notes C, E, and G would be played. Try playing three note sequences from several scales one after the other; for example, C, E, and G followed by F, A, and C followed by G, B, and D.

Topic 5.4 Major Chords

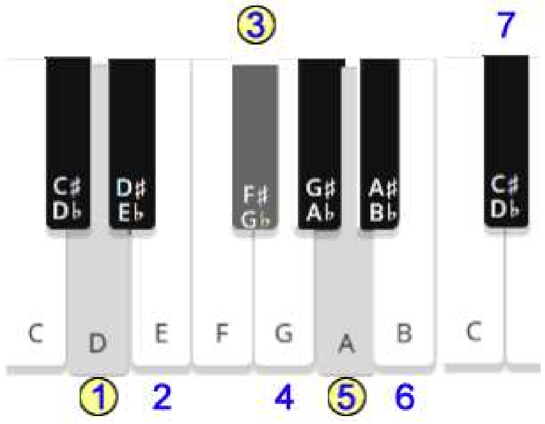
Major chords are three-note combinations of notes within a seven-note major scale. For example, the C Major scale consists of the seven white keys, numbered one through seven in the illustration below.



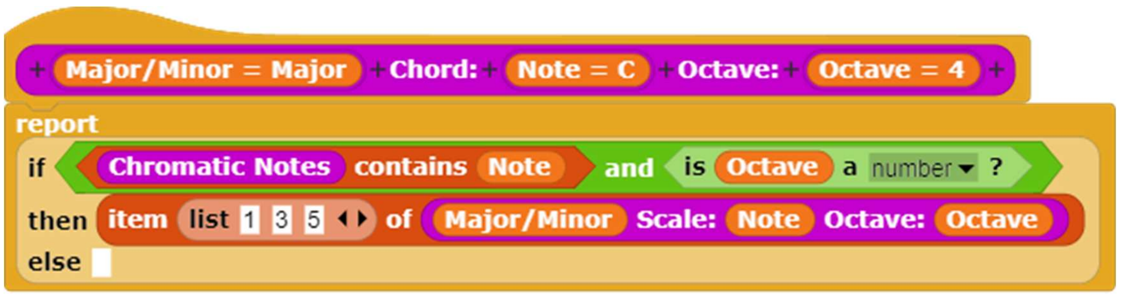
The C Major chord consists of the first, third, and fifth notes of the C Major scale (i.e., the notes C, E, and G).



This pattern is repeated for all of the other major scales. For example, the D Major chord is composed of the first, third, and fifth notes of the D Major scale (i.e., the notes D, F#, and A).

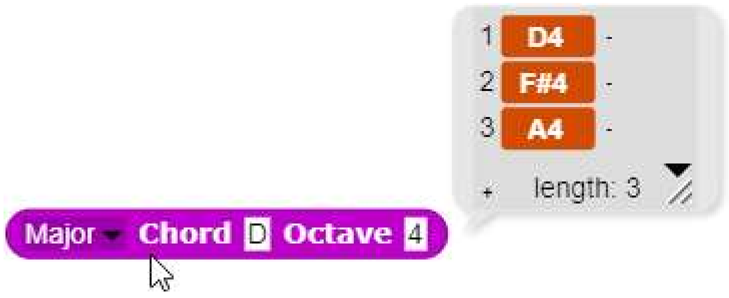


The **Chord** reporter block checks to ensure that the entered note and octave are part of the chromatic scale. If they are, the block reports the first, third, and fifth note of the corresponding scale.

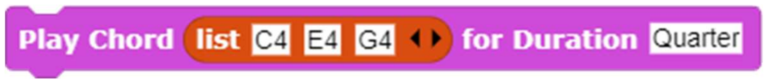


```
+ Major/Minor = Major +Chord:+ Note = C +Octave:+ Octave = 4 +
report
  if < Chromatic Notes contains Note and is Octave a number ? >
  then item list 1 3 5 ◂▸ of Major/Minor Scale: Note Octave: Octave
  else
```

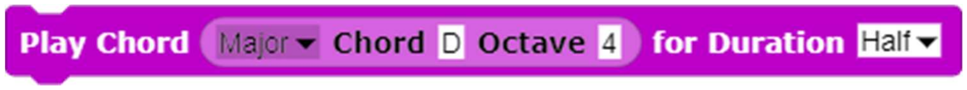
For example, the D Major chord consists of D, F#, and A. In the example below, the fourth octave has been specified for this chord.



Recall that the **Play Chord** code block plays a group of notes that form a chord.



The **Major Chord** reporter block can now be used in combination with the **Play Chord** code block to play any of the twelve major chords in any octave.



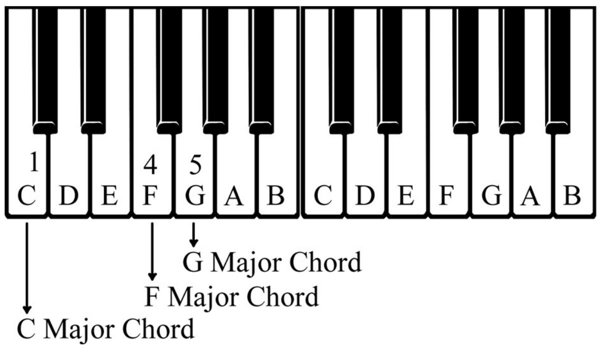
Further information on minor scales and instruction on how to form chords with more than three notes are provided in the appendix of this module.

Exploration 5.4 Major Chords

Try combining the note C from one octave with the notes E and G from different octaves. How does that affect the quality of the chord produced? Try assembling several sequences of chords. For example, try combining a C Major chord, followed by a F Major chord, followed by a G Major chord. What other sequences of chords sound good in combination with one another?

Topic 5.5 Chord Progressions in the Major Scale

A chord progression consists of a series of chords, often used as a backing track that accompanies a melody consisting of individual notes. A common chord progression consists of the chords associated with the first, fourth, and fifth note in a major scale.



For example, the notes C, F, and G are the first, fourth, and fifth notes in the C Major scale. Therefore, the C Major chord progression would consist of the C Major chord, the F Major chord, and the G Major chord.

Because a *Major Scale* chord progression is established by the first, fourth, and fifth notes in the scale, the corresponding chords are often referenced as the *I Chord*, the *IV Chord*, and the *V Chord*. Roman numerals are used to reference chords to differentiate chords from notes. The I-IV-V chord progression for the C Major scale is summarized in the table below.

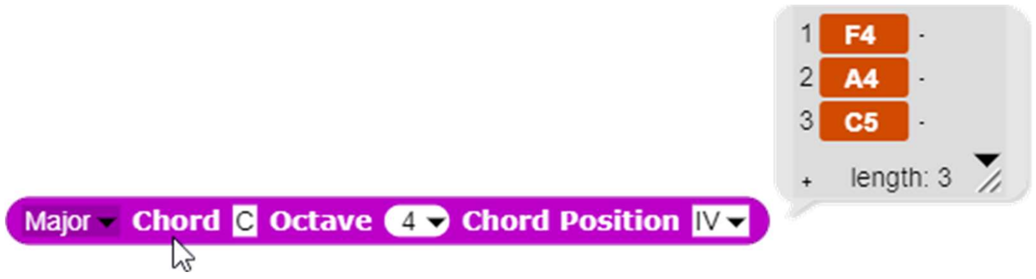
Chord Numbers	I	IV	V
C Major Chord Progression	C Chord	F Chord	G Chord

A chord progression beginning with a D Major chord would consist of the D Major Chord, the G Major Chord, and the A Major chord. Since these chords maintain the same relative positions in the D Major scale, the chord numbers remain the same.

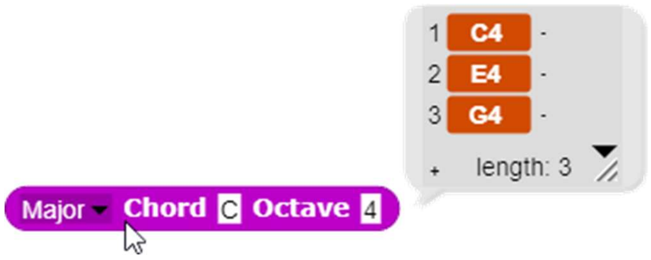
Chord Numbers	I	IV	V
D Major Chord Progression	D Chord	G Chord	A Chord

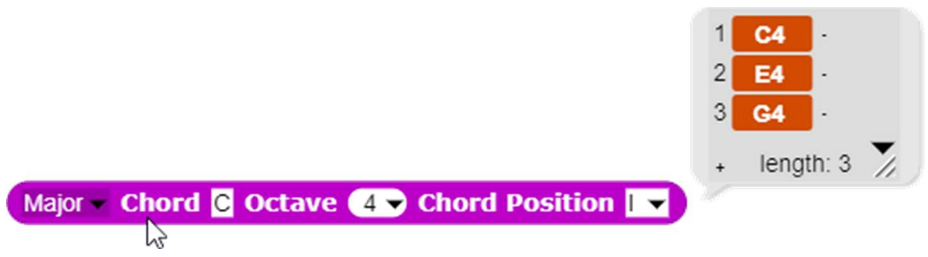
There are many other chord progressions, but the I-IV-V chord progression is the basis of much of the popular music that employs the Western chromatic scale.

The **Major / Minor Chord Position** block can be used to identify the chords along a major scale.



Chords can be described by using either the name of the chord (C Major chord) or the number of the chords (the I chord). Both chords consist of the same notes regardless of whether the name of the chord or the number of the chord is used.

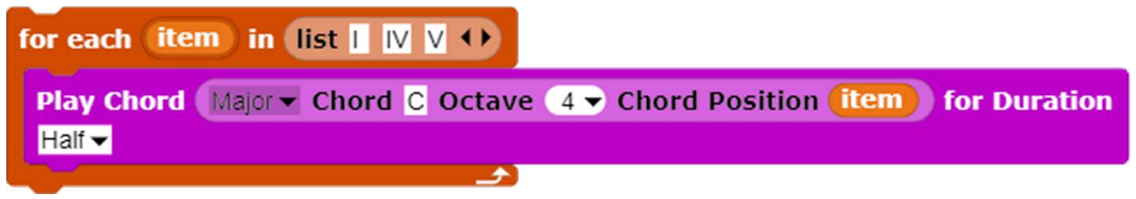




```
Major Chord C Octave 4 Chord Position I
```

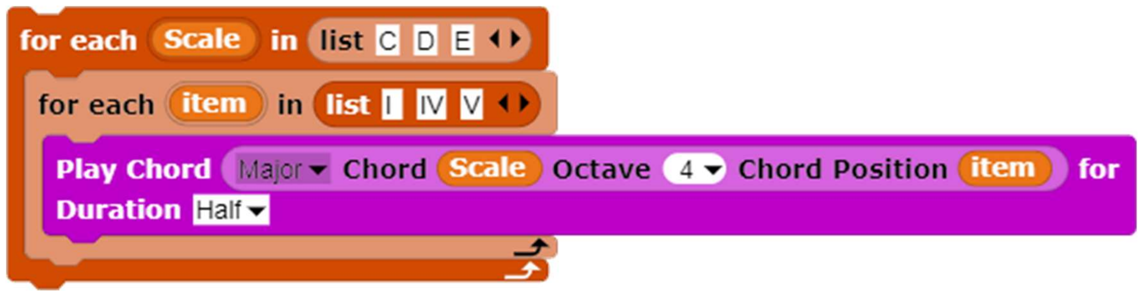
```
1 C4
2 E4
3 G4
length: 3
```

However, there is one advantage to referring to chords by number. A loop can be constructed to play a sequence of numbered chords.



```
for each item in list I IV V
    Play Chord (Major Chord C Octave 4 Chord Position item) for Duration Half
```

Once code has been constructed that can play a progression of numbered chords, the chord progression can be readily shifted from one scale to another. The outer loop in the example below shifts from one scale to the next (C Major, D Major, E Major) and the inner loop plays a numbered chord sequence (I, IV, V) within each scale.



```
for each Scale in list C D E
    for each item in list I IV V
        Play Chord (Major Chord Scale Octave 4 Chord Position item) for Duration Half
```

Because of this ease in shifting from one scale to another, musicians will sometimes refer to the "I Chord" or the "IV Chord". Using this notation, the chords for the first verse of "Over the Rainbow" would be represented in the following way:

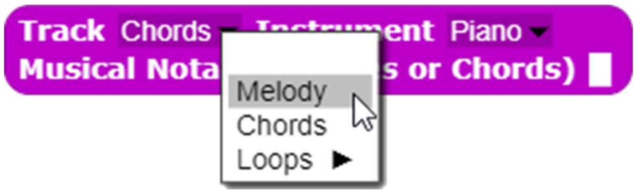
I iii IV I IV I vii ii vii i

Exploration 5.6 Chord Progressions in the Major Scale

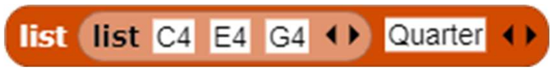
Try experimenting with chord progressions composed of various major chords. For example, create a custom code block that plays a V-IV-I chord progression in any scale.

Topic 5.7 Chord Tracks

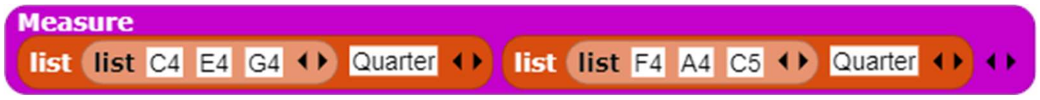
The **Play Tracks** block was introduced in a previous module. The **Track** block can be used to create either a melody track (consisting of individual notes) or a chord track. Loops can be created for both melody and chord tracks; when a loop is created, the track loops until the main music track is completed.


Track Chords Instrument Piano
Musical Notation (Notes or Chords)
Melody
Chords
Loops

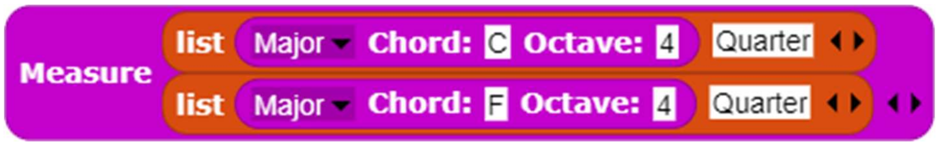
The format for a chord consists of two or more notes and a duration.



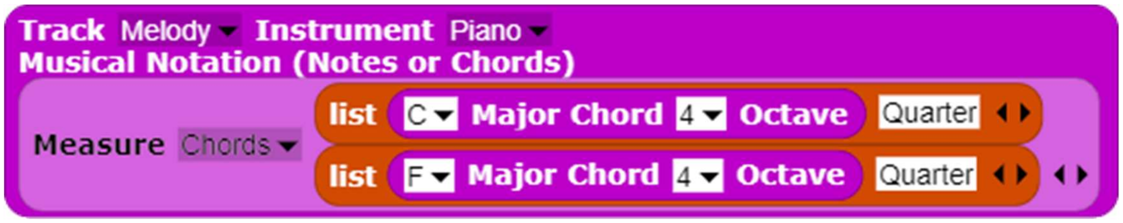
A series of chords formatted in this manner are then placed in a measure. In music, a measure is used to group notes together. In TuneScope, the **Measure** block is also used as a mechanism to ensure that the music remains synchronized across tracks. Chords can be grouped together using the **Measure** block.



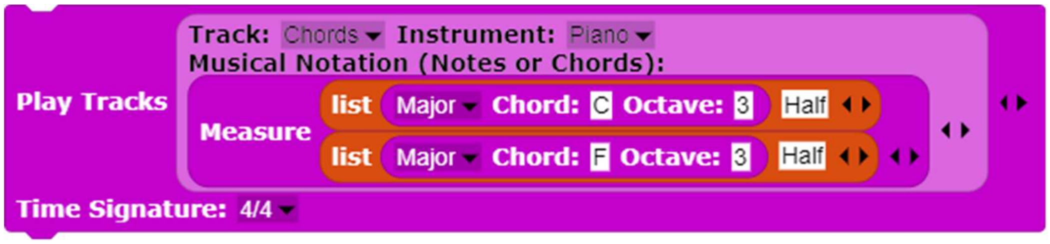
A **Chords** reporter block could also be used in place of individually listed notes (as shown above) since the two forms are functionallly equivalent.


Measure
list Major Chord: C Octave: 4 Quarter
list Major Chord: F Octave: 4 Quarter

A list of chords formatted in this manner can then be placed in the input to a **Track** block (with the *Chords* option selected).


Track Melody Instrument Piano
Musical Notation (Notes or Chords)
Measure Chords
list C Major Chord 4 Octave Quarter
list F Major Chord 4 Octave Quarter

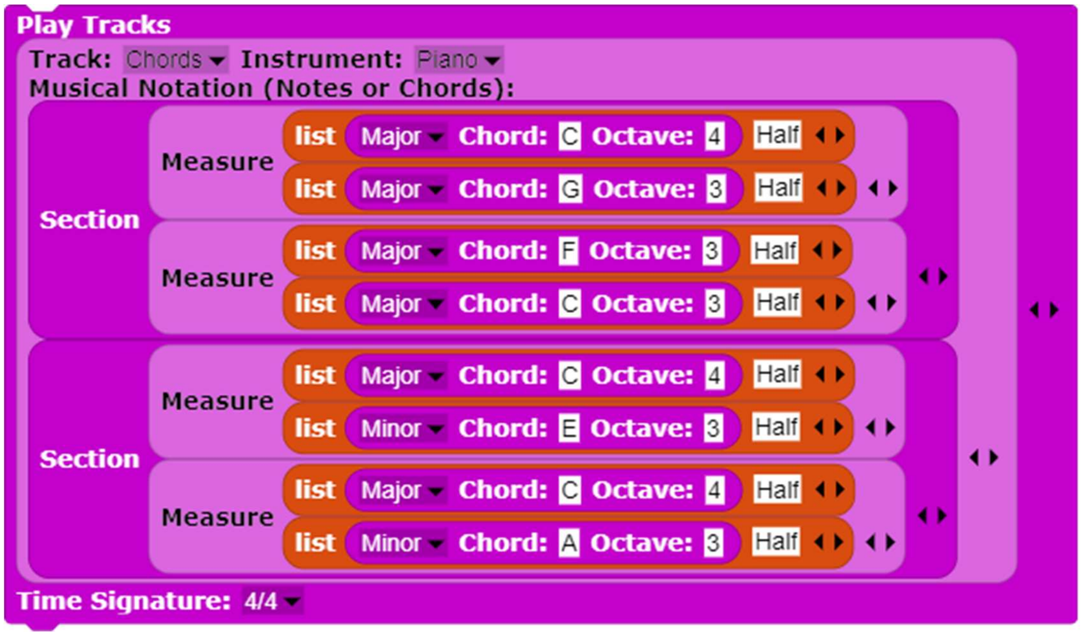
Music tracks formed in this manner can then be played with the **Play Tracks** block.



```
Play Tracks
    Track: Chords  Instrument: Piano
    Musical Notation (Notes or Chords):
        Measure
            list Major Chord: C Octave: 3  Half
            list Major Chord: F Octave: 3  Half
Time Signature: 4/4
```

The **Play Tracks** block can then be used to combine melody, chord, and drum tracks to compose and play a tune.

Multiple measures can be combined into sections, as shown below. This can be helpful when there are groups of measures that repeat throughout a song. For example, most popular songs have a chorus that repeats between verses.



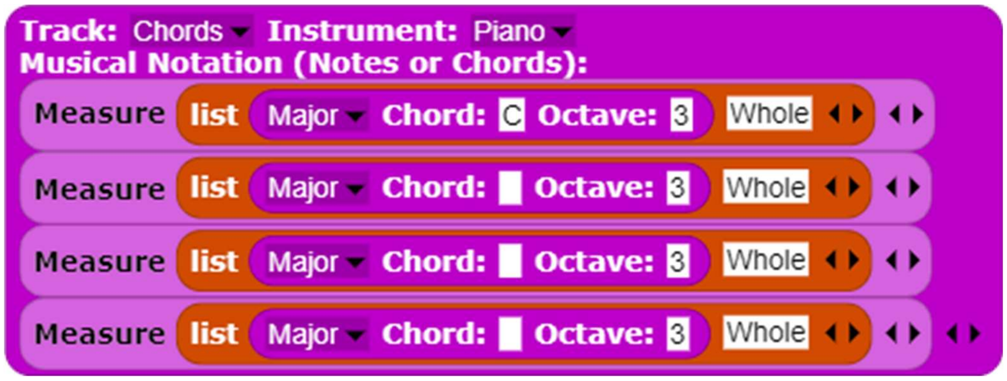
```
Play Tracks
    Track: Chords  Instrument: Piano
    Musical Notation (Notes or Chords):
        Section
            Measure
                list Major Chord: C Octave: 4  Half
                list Major Chord: G Octave: 3  Half
            Measure
                list Major Chord: F Octave: 3  Half
                list Major Chord: C Octave: 3  Half
        Section
            Measure
                list Major Chord: C Octave: 4  Half
                list Minor Chord: E Octave: 3  Half
            Measure
                list Major Chord: C Octave: 4  Half
                list Minor Chord: A Octave: 3  Half
Time Signature: 4/4
```

Topic 5.8 Creating a Chord Progression with a Melody

A chord progression can be used as the basis for development of an accompanying melody. Here are several popular chord progressions that begin with the C Major chord.

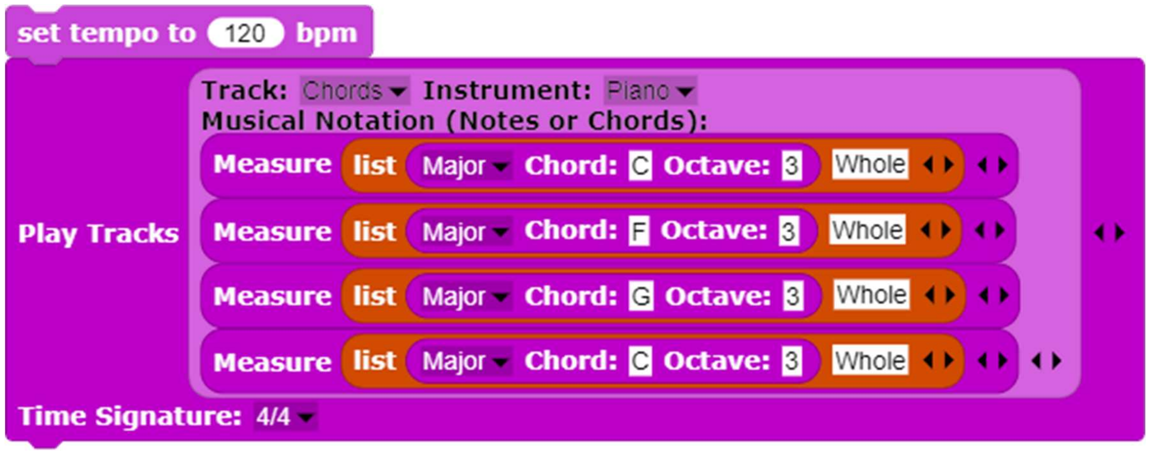
Chord Progressions
C Major - F Major - G Major - C Major
C Major - G Major - A Minor - F Major
C Major - F Major - G Major - F Major
C Major - A Minor - F Major - G Major

Use a template like the one shown below to complete a chord progression using an example in the table above.



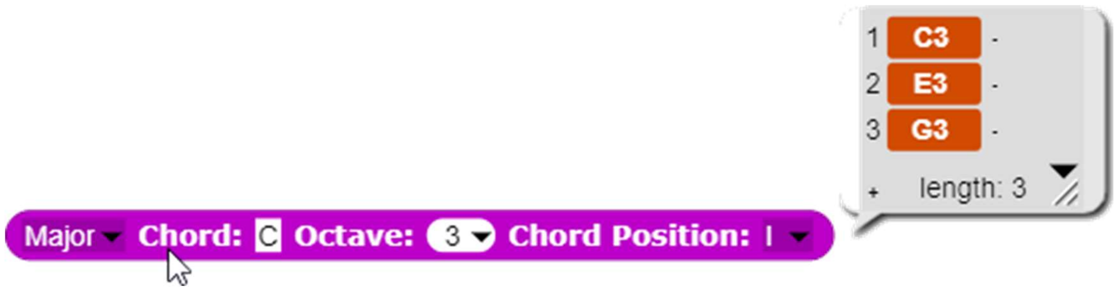
```
Track: Chords  Instrument: Piano
Musical Notation (Notes or Chords):
  Measure  list  Major  Chord: C  Octave: 3  Whole
  Measure  list  Major  Chord:    Octave: 3  Whole
  Measure  list  Major  Chord:    Octave: 3  Whole
  Measure  list  Major  Chord:    Octave: 3  Whole
```

The completed chord progression can then be placed in a **Play Tracks** block to hear the result.



```
set tempo to 120 bpm
Play Tracks
  Track: Chords  Instrument: Piano
  Musical Notation (Notes or Chords):
    Measure  list  Major  Chord: C  Octave: 3  Whole
    Measure  list  Major  Chord: F  Octave: 3  Whole
    Measure  list  Major  Chord: G  Octave: 3  Whole
    Measure  list  Major  Chord: C  Octave: 3  Whole
Time Signature: 4/4
```

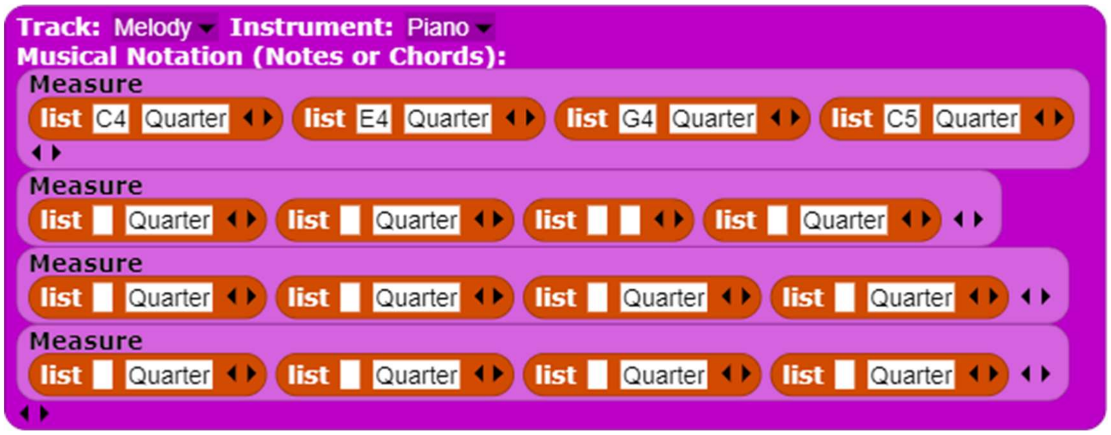
Each chord in the progression can then be deconstructed to identify the individual notes.



```
Major  Chord: C  Octave: 3  Chord Position: I

1  C3
2  E3
3  G3
length: 3
```

The notes in the chord – C3, E3, and G3 in the case of the C Major chord – are then used as the basis of notes in an accompanying melody. The melody should be in a different octave than the chords. In this example, the C Major chord that begins the chord progression is in the third octave. The melody, therefore, has been moved up by an octave. Since four quarter notes are needed to complete the measure, the beginning note (C4) has been repeated at the end, but in a different octave (C5). By this means, the starting point for a melody has been completed.



The same process should then be repeated for the other chords in the chord progression to complete the template. The **Melody** track can then be added to the **Play Tracks** block so that the chords and melody can be played together.

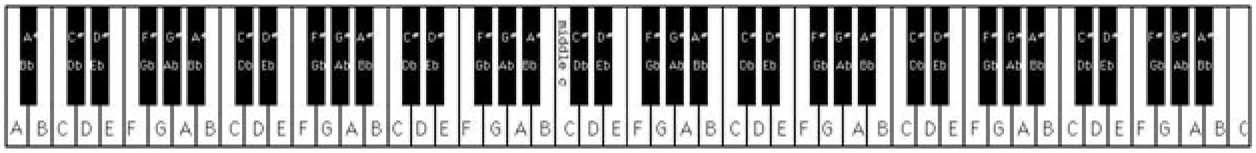
The result is generally a melody to accomany the chord progresion that will be listenable. This can then be used as a starting point for variation in the melody. For example, the same notes can be rearranged in a different order. Other variations are explored in the next chapter.

Chapter 6. Creating a Melody

Glen Bull, Jo Watts, Rachel Gibson, and Luke Dahl

The term musical scale refers to a sequence of notes ordered by pitch. In this context, the word *scale* is derived from the Latin word for ladder, meaning literally a *ladder of notes*. (The phase "to scale a fortress" is also related to the same root word.)

Many different musical scales have been developed throughout history. The Western chromatic musical scale consists of twelve notes. Other scales have been developed that employ different numbers of notes.


A# Bb
C# Db D# Eb F# Gb G# Ab A# Bb
C# Db D# Eb F# Gb G# Ab A# Bb
C# Db D# Eb F# Gb G# Ab A# Bb
middle c
C# Db D# Eb F# Gb G# Ab A# Bb
C# Db D# Eb F# Gb G# Ab A# Bb
C# Db D# Eb F# Gb G# Ab A# Bb
C# Db D# Eb F# Gb G# Ab A# Bb
A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C

The overwhelming majority of popular music today is composed using groups of notes drawn from within the notes of the chromatic scale. The keys of today's pianos are tuned using this scale.

Most songs are written using subsets of notes within the larger 12-note scale. Somewhat confusingly, these subsets of the larger scale are also known as *scales*. Creation of a tune primarily using notes from a subset of the larger scale yields a more melodious result.

Topic 6.1 Creating Harmonious Melodies

Any freely vibrating object, such as a violin or guitar string, vibrates at a specific rate that is affected by its length, mass, and tension on the string. This rate of vibration is known as its frequency. Freely vibrating objects also generate overtones that are multiples of the base frequency. For example, a string moving back and forth at a rate of 100 times per second would also generate overtones at 200 times per second, 300 times per second, 400 times per second, etc. An overtone that is a multiple of the base frequency (also known as the *fundamental frequency*) is known as a *harmonic*.

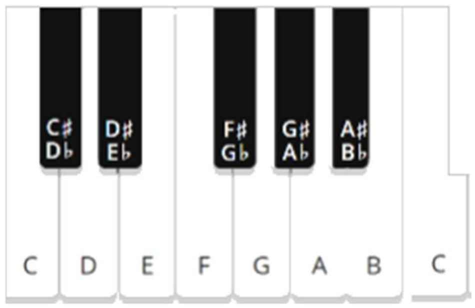
When the harmonics of two tones are closely aligned, the resulting blend is also perceived as *harmonious*. For example, if Tone 1 has a fundamental frequency of 100 cycles per second, its harmonics will be 100, 200, 300, 400, etc. If Tone 2 has a fundamental frequency of 200 cycles per second, its harmonics will be 400, 600, 800, etc. The closest possible alignment of harmonics occurs when the fundamental frequency of one tone is double the frequency of another tone.

In general, harmonics of tone are in better alignment when the ratio of the frequency of one tone to a second tone is a low whole-number ratio. For example, when the frequency of one tone is double the frequency of another tone, the result is a 2:1 ratio. Other harmonious combinations include a 3:2 ratio, a 4:3 ratio, and a 5:4 ratio. On the other hand, combinations such as 15:8 or 45:32 are perceived as less harmonious.

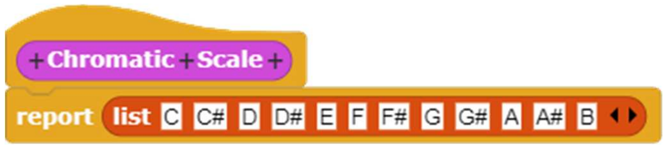
Scales that are subsets of the full 12-note Chromatic scale are, not surprisingly, drawn from combinations of notes that are perceived as harmonious. Use of these subsets of notes results in tunes that are more likely to sound pleasing in combination with one another. Music theorists have written libraries of books about this topic, so in many respects this explanation is a simplification. However, for anyone who does not have time to read dense tomes on the topic, this overview provides context for the reason that subsets of notes are organized in this way.

Topic 6.2 The Twelve-Note Chromatic Scale

The concept of an octave was introduced in a previous chapter, *Combining Art and Music*. An octave is a span of notes on the piano keyboard that begins with one note (such as the note "C" in the illustration below) and ends in the same note.



A custom **Chromatic Scale** code block reports the twelve notes of the chromatic scale.



```
+Chromatic+Scale+
report list C C# D D# E F F# G G# A A# B
```

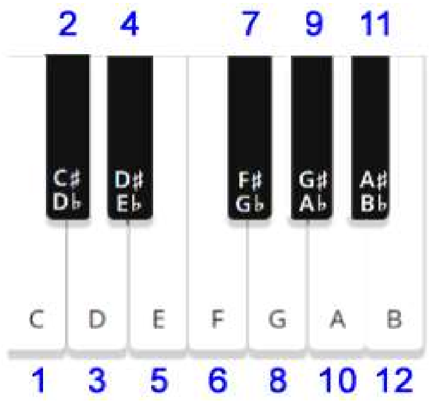
The **Chromatic Scale** block can be used to construct major and minor scales that are subsets of the chromatic scale

Exploration 6.1 The Twelve-Note Western Chromatic Scale

Create a custom code block that uses the **Chromatic Scale** code block described in this section as an input to play the notes of the scale in order. Modify the code block so that it plays the notes of the scale in order beginning and ending with the note "E".

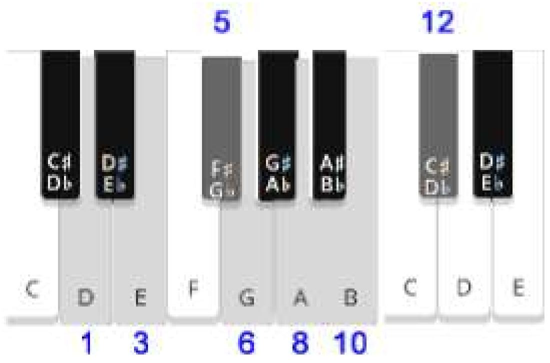
Topic 6.2 Major Scales

Within the 12-note Chromatic scale, the subsets of notes known as the Major scales are composed of groups of seven notes within the larger scale. For example, the C Major scale consists of following notes within the Chromatic scale beginning with the note "C" - notes 1, 3, 5, 6, 8, 10, and 12:

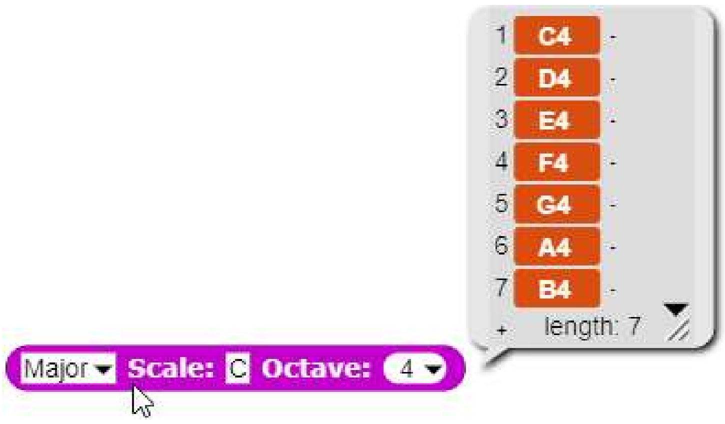


As the illustration above shows, these are the white notes on the keyboard. Consequently, the C Major scale does not have any sharps or flats (i.e, black keys) in it.

There are twelve major scales, one for each note in the chromatic scale. The notes for each major scale are always drawn from the same relative position in the chromatic scale. For example, the D Major scale consists of the notes numbered 1, 3, 5, 6, 8, 10, 12 (beginning with D) in the illustration below. Each of the other major scales is based on this same pattern.



Using the same pattern of notes means that while the *absolute pitch* shifts, the relative intervals of one note to another within a scale remain the same. This makes it easy for musicians to transpose tunes from one scale to another, and facilitates musical exploration and invention. This regularity also facilitates creation of a custom code block that reports the seven notes in any major scale. For example, the notes of the C Major scale in the fourth octave are shown in the illustration below.



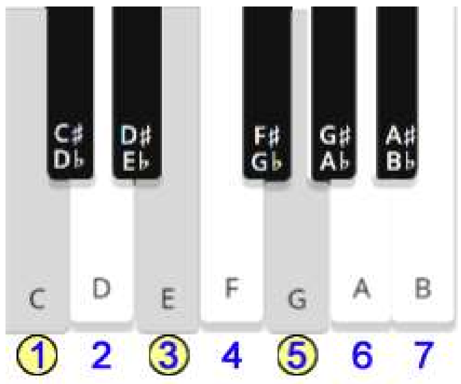
Much of the music in medieval times was designed to create music that sounds harmonious (i.e., heavenly). However, dissonance is also an important tool in composition of music. The blues, for example, make use of dissonance. The *minor scales* are perceived as more dissonant than the major scales. Minor scales are formed from the first, third, fourth, sixth, eighth, nineth, and eleventh notes of the chromatic scale.

Exploration 6.2 Major Scales

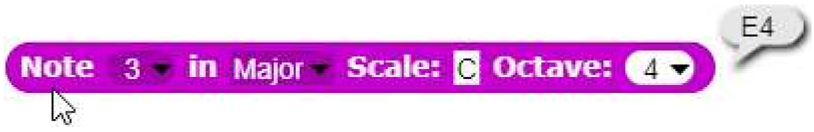
Create a custom code block that plays the notes of a major scale. Then create a generalized version that will play the notes of any major scale.

Topic 6.3 Harmonious Combinations within Major and Minor Scales

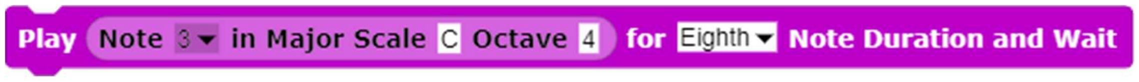
The first, third, and fifth notes within any major or minor musical scale sound particularly harmonious when played in combination with one another. For example, C, E, and G in the C Major scale sound harmonious when played in combination with one another.



The **Note # in Major Scale** block reports the note associated with any position in a major scale. For example, E4 is the third note in the C major scale.



This block can be used as an input to the **Play Note and Wait** block.


Play Note 3 in Major Scale C Octave 4 for Eighth Note Duration and Wait

Musicians also refer to musical intervals within a scale by name. These names are assigned with reference to the position of notes within the major and minor scales. A *major second* refers to the interval between the first and second notes in a major scale, a *major third* refers to the interval between the first and third notes in a major scale, etc.

Names of Musical Intervals			
Note	*Number*	*Name*	*Ratio*
C	1		
D	2	Major 2nd	
E	3	Major 3rd	5:4
F	4	Perfect 4th	4:3
G	5	Perfect 5th	3:2
A	6	Major 6th	5:3
B	7	Major 7th	

The two intervals with the lowest whole-number ratios (4:3 and 3:2) in the scale are known as a *perfect* fourth and a *perfect* fifth respectively. These are designated as perfect because they are judged to sound the most harmonious.

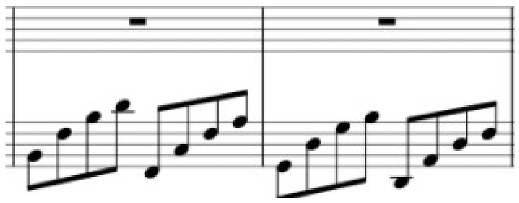
Exploration 6.3 Musical Intervals within a Scale

Create a custom block that randomly selects notes from different positions in a major scale. Play several different combinations of notes and judge which sequences sound best to you.

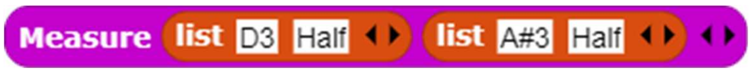
6.4 Musical Measures

As outlined above, melodies constructed from notes within a scale are more likely to be perceived as pleasing. Often composers create ascending or descending combinations of notes within a scale. When a melody is created, notes are traditionally grouped within measures.

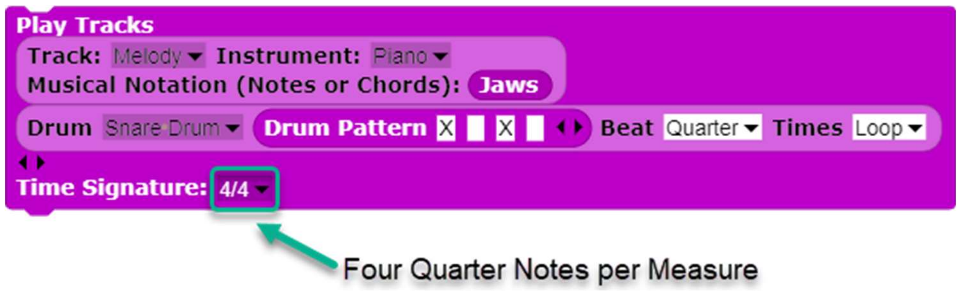
The concept of measures was discussed in previous chapters. A measure in music is a unit that holds a specified number of beats played at a specified tempo. Notes within a measure are placed between vertical bars in music notation.



The **Measure** block is used to group sequences of notes that fall within a measure.



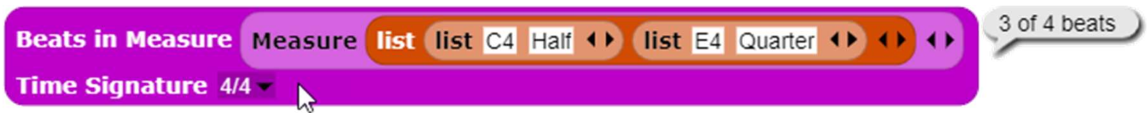
The notes in 4/4 time must add up to one whole note per measure.


Play Tracks
Track: Melody Instrument: Piano
Musical Notation (Notes or Chords): Jaws
Drum Snare Drum Drum Pattern X X Beat Quarter Times Loop
Time Signature: 4/4
Four Quarter Notes per Measure

Any combination that add up to a whole note satisfies the requirements for a measure in 4/4 time.

Durations that Add Up to a Whole Note								
Number	*Notes*							
1	Whole							
2	Half	Half						
4	Quarter	Quarter	Quarter	Quarter				
8	Eighth	Eighth	Eighth	Eighth	Eighth	Eighth	Eighth	Eighth

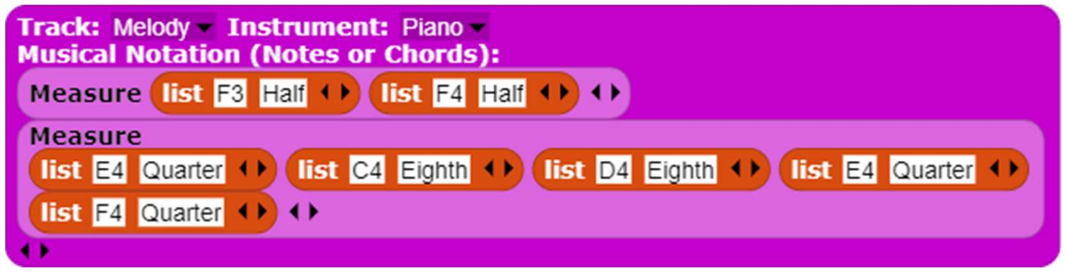
The **Beats in Measure** block reports the number of beats associated with the notes currently in a measure. For example, the notes C4 (half a beat) and E4 (a quarter beat) add up to a total of 3 beats.



```
Beats in Measure  Measure  list ( list C4 Half ◂▸ ) ( list E4 Quarter ◂▸ ) ◂▸ ◂▸
Time Signature 4/4 ▾
```

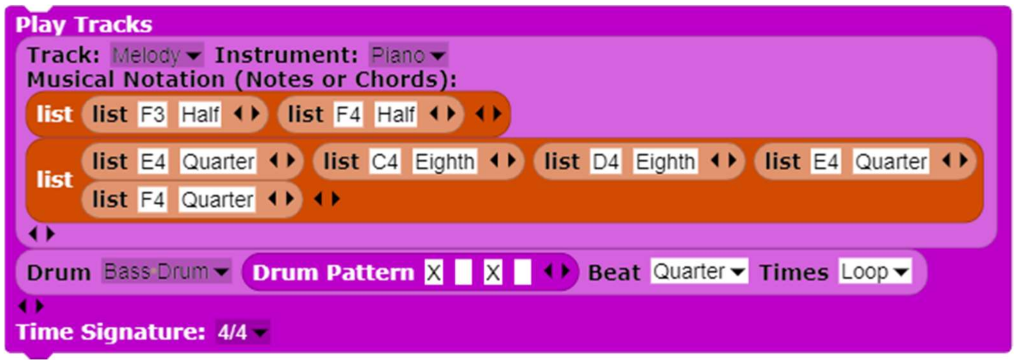
Since there are 4 beats per measure in 4 / 4 time, one more beat would be needed to complete the measure. The duration of the second note, E4, could be increased from a quarter note to a half note to accomplish this. Alternatively, a third note that is a quarter note in duration could be added to the measure.

The example below is a piano track. Note that each measure in this track adds up to a whole note.



```
Track: Melody ▾  Instrument: Piano ▾
Musical Notation (Notes or Chords):
  Measure ( list F3 Half ◂▸ ) ( list F4 Half ◂▸ ) ◂▸
  Measure
    ( list E4 Quarter ◂▸ ) ( list C4 Eighth ◂▸ ) ( list D4 Eighth ◂▸ ) ( list E4 Quarter ◂▸ )
    ( list F4 Quarter ◂▸ ) ◂▸
◂▸
```

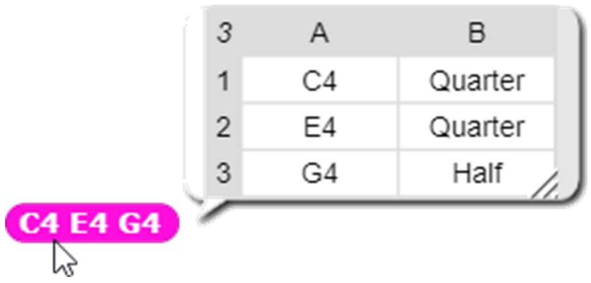
Completed measures are assembled into instrumental tracks. The **Play Tracks** block was introduced in previous chapters. The example below consists of a piano track combined with a drum track.



```
Play Tracks
  Track: Melody ▾  Instrument: Piano ▾
  Musical Notation (Notes or Chords):
    list ( list F3 Half ◂▸ ) ( list F4 Half ◂▸ ) ◂▸
    list ( list E4 Quarter ◂▸ ) ( list C4 Eighth ◂▸ ) ( list D4 Eighth ◂▸ ) ( list E4 Quarter ◂▸ )
         ( list F4 Quarter ◂▸ ) ◂▸
  ◂▸
  Drum Bass Drum ▾  ( Drum Pattern X _ X _ ◂▸ )  Beat Quarter ▾  Times Loop ▾
◂▸
Time Signature: 4/4 ▾
```

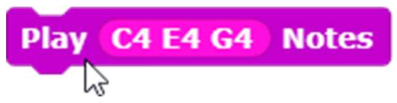
Topic 6.5 Musical Variations

A pleasing combination of notes can be extended through musical variations. This will be illustrated through use of the musical sequence "C4 E4 G4".

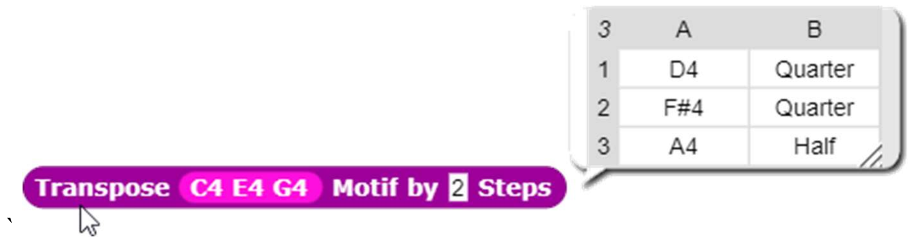


3	A	B
1	C4	Quarter
2	E4	Quarter
3	G4	Half

The **Play Notes** block can be used to play the sequences of notes generated:

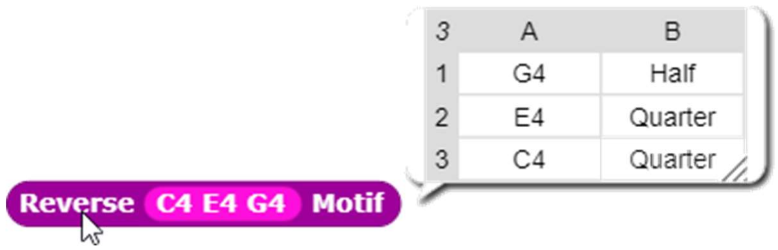


If this sequence is transposed by shifting it to a different point on the piano keyboard, the sequence of "C4 E4 G4" can be transformed to "D4 F#4 A4".



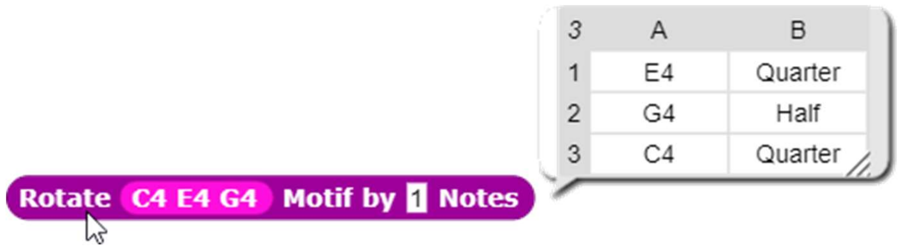
3	A	B
1	D4	Quarter
2	F#4	Quarter
3	A4	Half

The sequence can be reversed, changing "C4 E4 G4" into "G4 E4 C4".



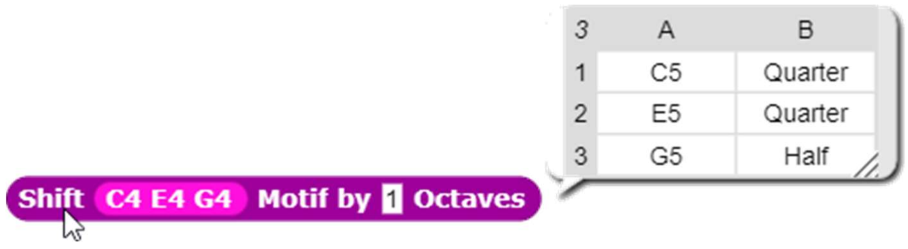
3	A	B
1	G4	Half
2	E4	Quarter
3	C4	Quarter

If the sequency of notes is rotated, moving the first note to the end of the sequence, "C4 E4 G4" becomes transformed to "E4 G4 C4".



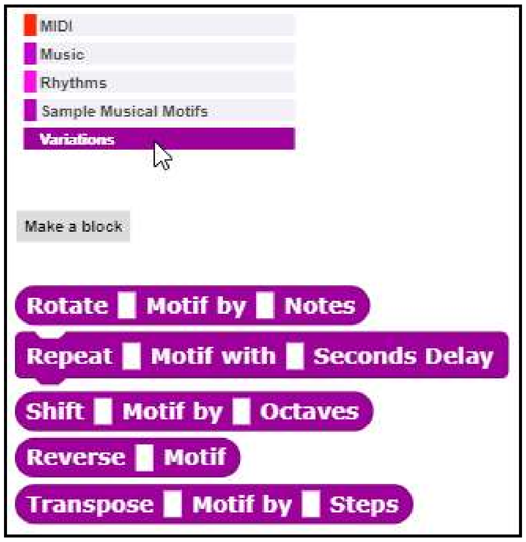
3	A	B
1	E4	Quarter
2	G4	Half
3	C4	Quarter

The sequence can be shifted into a different octave, so that "C4 E4 G4" becomes "C5 E5 G5".



3	A	B
1	C5	Quarter
2	E5	Quarter
3	G5	Half

Reporter blocks for generating several common musical variations have been provided in the program accompanying the MIDI beta site in a palette titled *Variations*:



By exploring musical patterns and variations in this manner, combinations of notes can be created to produce a melody.

Chapter 7. MIDI Tunes

Glen Bull, Jo Watts, and Rachel Gibson

The *Musical Instrument Digital Interface* (MIDI) is a technical standard that describes ways in which electronic musical instruments can connect to one another. It was extended to include communications between a MIDI instrument and a computer. Consequently, thousands of digital songs are now available in MIDI format.

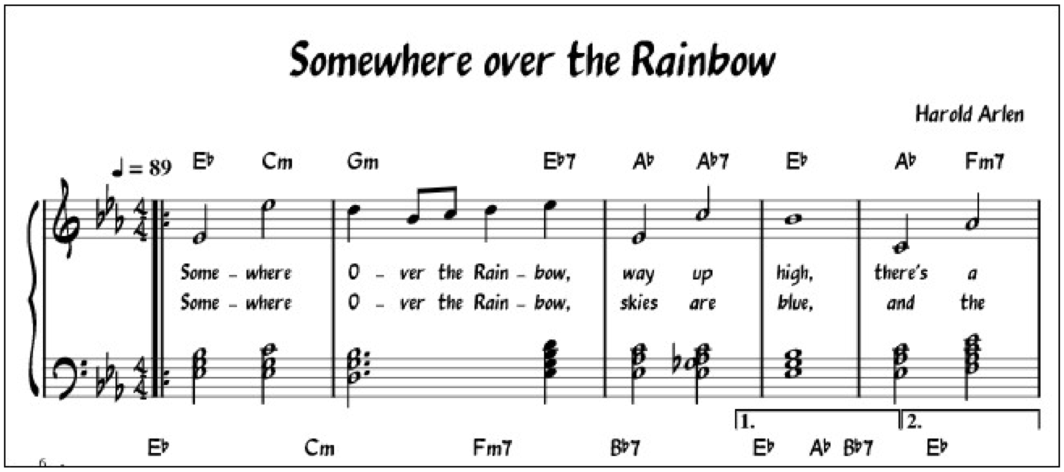
The MIDI standard can extend the capabilities of TuneScope in two ways:

1. The melody of a MIDI file can be extracted and imported into TuneScope.

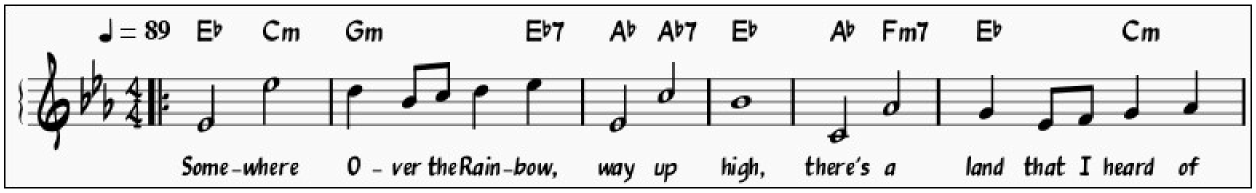
2. An external MIDI keyboard can be used to record a list of notes in TuneScope.

7.1 Extracting a Melody from a MIDI File

The example below illustrates the score for a song. The melody is shown in the top musical staff (the treble clef) while accompanying chords are shown in the bottom staff (the bass clef).


Somewhere over the Rainbow
Harold Arlen
♩ = 89
E♭ Cm Gm E♭7 A♭ A♭7 E♭ A♭ Fm7
Some - where O - ver the Rain - bow, way up high, there's a
Some - where O - ver the Rain - bow, skies are blue, and the
1. 2.
E♭ Cm Fm7 B♭7 E♭ A♭ B♭7 E♭

The process of translating the melody into a list of TuneScope notes is simplified if all of the tracks except for the melody are removed from the score. This can be done in a number of different music notation programs such as NoteFlight and MuseScore. A description of how this might be done in MuseScore is provided in the appendix. The resulting score should now consist of just the melody.


♩ = 89
E♭ Cm Gm E♭7 A♭ A♭7 E♭ A♭ Fm7 E♭ Cm
Some-where O - ver the Rain-bow, way up high, there's a land that I heard of

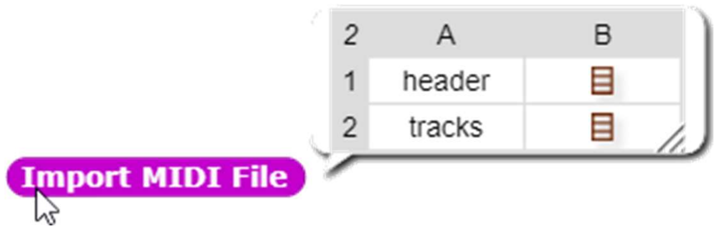
The melody track can then be examined by converting it to a human readable form. JavaScript is a language frequently used for web development. (Snap*!*, for example, is written in JavaScript.) JavaScript Object Notation (JSON) files are a human readable data format. There are a number of tools for converting MIDI files to JSON format. A description of how this can be done using a Tone.js conversion tool is provided in the appendix.

When the MIDI file is converted to JSON format, the MIDI information is displayed in a text format. The file has a header with information such as the time signature and the beats per minute, and tracks of notes. Each note in a track has information such as the name of the note (D#4) and its duration.

```
{
  "duration": 1.2794976020833333,
  "durationTicks": 911,
  "midi": 63,
  "name": "D#4",
  "ticks": 0,
  "time": 0,
  "velocity": 0.6299212598425197
},
```

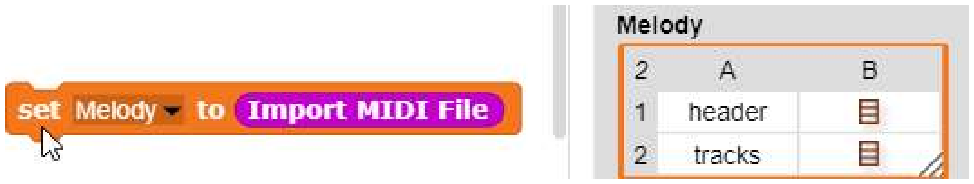
7.2 Importing a MIDI File into Snap*!*

If a MIDI file is converted to JSON format with a tool such as the Tone.js MIDI converter, the resulting JSON file can be imported into Snap*!* by dragging the JSON file from the desktop to the script area. The **Import MIDI File** block saves a step by converting a selected MIDI file into JSON format. Clicking this block produces a dialog box that requests a MIDI file. Once a MIDI file is selected, the output of the reporter block displays a table in JSON format. The header and the tracks described in the preceding section appear in this table.



2	A	B
1	header	▤
2	tracks	▤

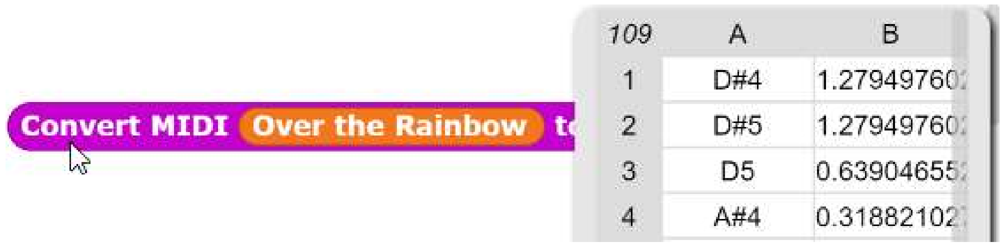
The output of the reporter block can be assigned to a variable.



2	A	B
1	header	▤
2	tracks	▤

7.3 Converting a MIDI File to TuneScope Format

The **Convert MIDI to TuneScope** block converts the imported MIDI file into TuneScope format consisting a of a list of notes and durations (in seconds)

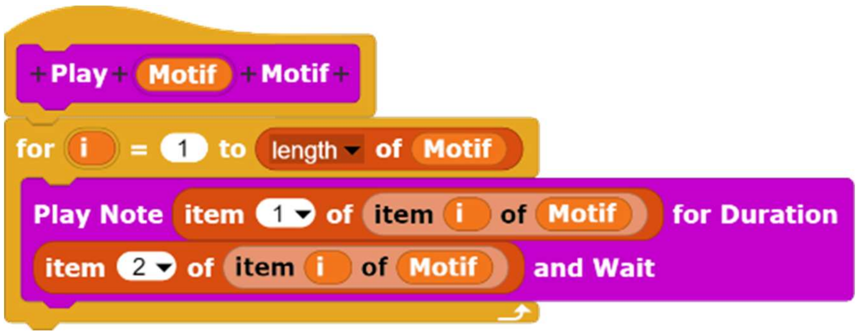


109	A	B
1	D#4	1.27949760
2	D#5	1.27949760
3	D5	0.63904655
4	A#4	0.31882102

The current version of the **Convert MIDI** block can only process a melody consisting of individual notes. It cannot process chords and cannot process multiple tracks. Therefore, preprocessing to remove extraneous elements such as chords and multiple tracks must be completed before the **Convert MIDI** block is used.

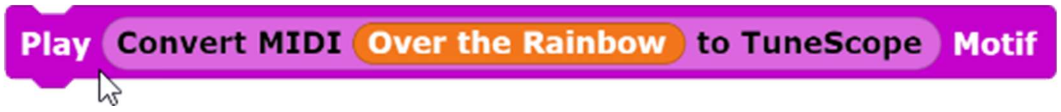
7.4 Playing a Converted Melody

Once a melody in MIDI format has been converted to TuneScope format, the **Play Motif** block (introduced in a previous chapter) can be used to play the resulting list of notes.



```
+Play+ (Motif) +Motif+
for (i) = (1) to (length of (Motif))
    Play Note (item (1) of (item (i) of (Motif))) for Duration
    (item (2) of (item (i) of (Motif))) and Wait
```

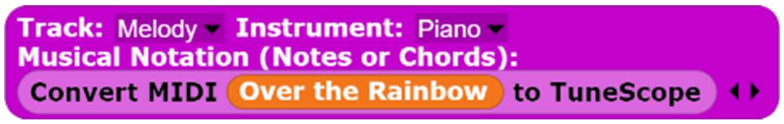
This block will play the converted MIDI track just as it would any other list of notes.



```
Play (Convert MIDI (Over the Rainbow) to TuneScope) Motif
```

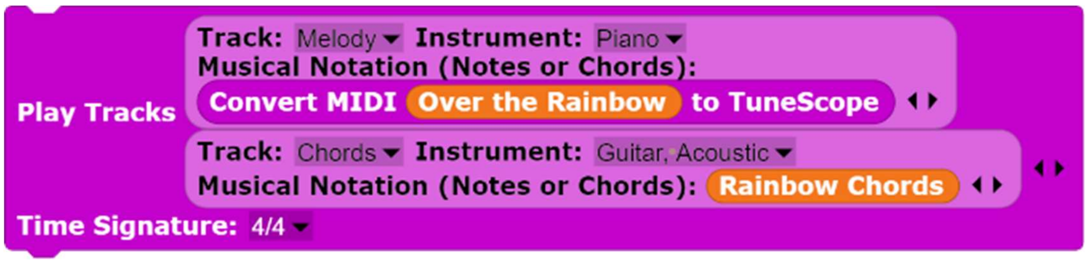
7.5 Playing TuneScope Tracks

The original TuneScope track blocks only process durations in the format of *quarter note*, *half note*, etc. An experimental version of the track block in the alpha TuneScope site can also process durations in seconds (the format used in MIDI files).



```
Track: Melody Instrument: Piano
Musical Notation (Notes or Chords):
(Convert MIDI (Over the Rainbow) to TuneScope)
```

This makes it possible to import a melody in MIDI format, and combine the converted track in TuneScope format with a chord track or a drum loop.



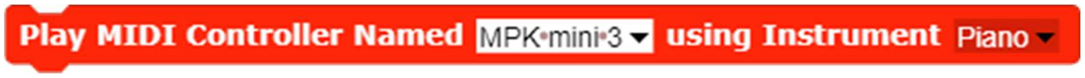
```
Play Tracks
    Track: Melody Instrument: Piano
    Musical Notation (Notes or Chords):
    (Convert MIDI (Over the Rainbow) to TuneScope)

    Track: Chords Instrument: Guitar, Acoustic
    Musical Notation (Notes or Chords): (Rainbow Chords)
Time Signature: 4/4
```

Once this feature has completed its review and testing on the Snap*!* development site, it will be added to the TuneScope extensions in Snap*!*

7.6 MIDI Keyboards

The **Play MIDI Controller** block can be used to tell TuneScope that a MIDI keboard is connected and available for use. A small number of MIDI keyboard models used to test this feature are available in the dropdown menu. Other MIDI controllers can be specified by typing the name into the input slot. (If the exact name of the MIDI controller connected to the computer is not typed correctly, the MIDI connection to TuneScope will not work.)



Once the musical instrument (*Piano* in the example) and the name of the MIDI controller (MPK mini 3 in this example) are specified, the **Play MIDI Controller** block is clicked once. The MIDI initialization will then be active for the remainder of the session.

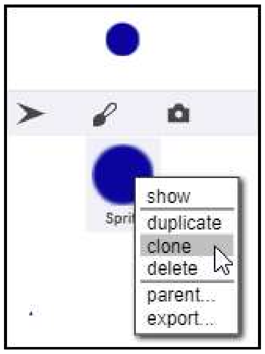
When a key is pressed on the MIDI keyboard, the key pressed is stored in the *Current Note* variable.

In addition to displaying the current note when a key on the MIDI keyboard is pressed, the corresponding MIDI number is displayed in a variable named *Current Note MIDI*. Each note has a corresponding MIDI number. C4 corresponds to the MIDI number 60. C#4 corresponds to the MIDI number 61, D4 corresponds to the MIDI number 62, and so on. MIDI numbers can be useful for transposing a series of notes to a different scale.

The ability to detect when a MIDI key is pressed and take an action in TuneScope based on the keypress can be used to create interactive music programs.

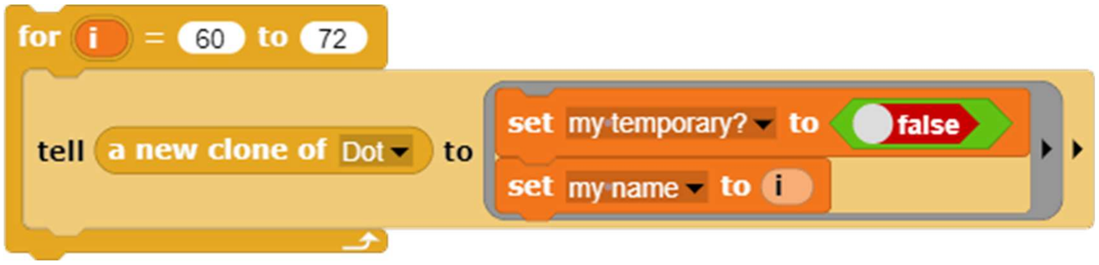
7.7 Interactive Music Visualization with MIDI Instruments

TuneScope music blocks lend themselves to creation of music visualizations. Many different ways of visualizing music have been developed. Light shows often use color to represent different elements of the musical scale. Visualizations could be created that highlight different major or minor scales. In Snap*!* / TuneScope, many types of visualizations can be constructed using a separate sprite to represent each note. In Snap*!*, clones provide a way to create multiple instances of a sprite. A clone of a sprite can be created by right-clicking the sprite icon in the sprite corral below the stage.

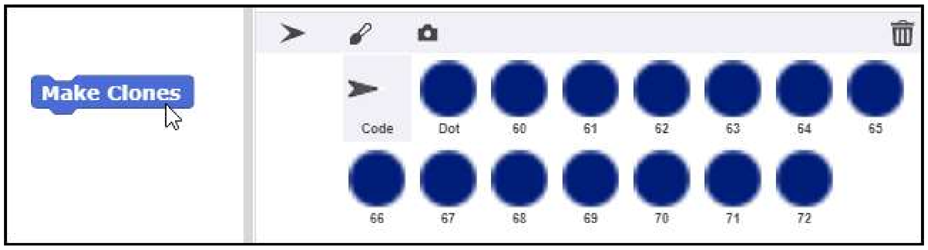


Two types of clones can be created with a script. Temporary clones are ones that disappear after they are no longer needed. For example, a shower of shooting stars might streak across the sky and disappear after they burn up. A permanent clone is one that persists, like a regular sprite, but shares attributes of its template with the parent sprite.

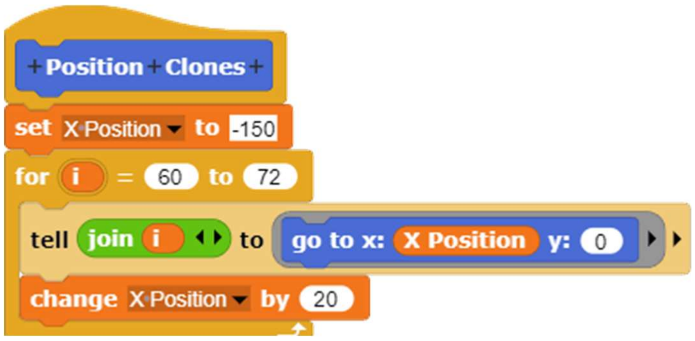
The following script creates a permanent clone for each item from 60 to 72. (These are the MIDI numbers that correspond to the octave beginning with Middle C (i.e., C4). The default state of a new program-generated clone is “temporary”; this script sets the *temporary* status to false, making the clone permanent. It then assigns each clone with a name corresponding to the current indexed number (“i”).


```
for i = 60 to 72
    tell (a new clone of Dot) to
        set my temporary? to false
        set my name to i
```

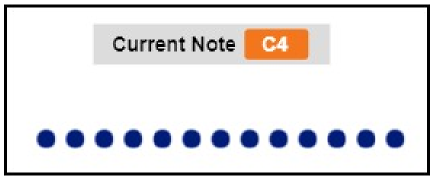
A code block **Make Clones** incorporates this script. When the **Make Clones** block is run, the numbered clones of the sprite named “Dot” appear in the sprite corral.



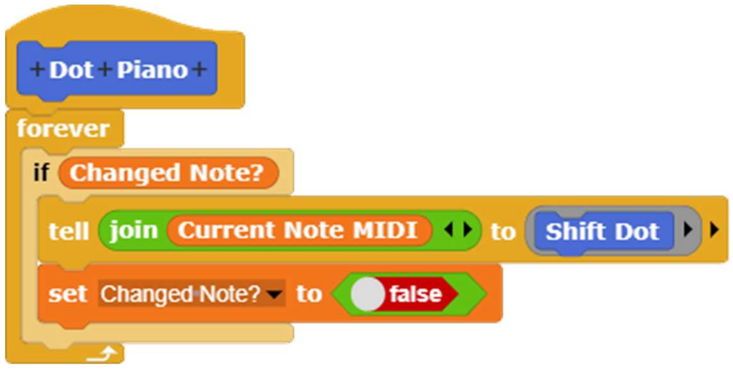
Once the clones have been created, the following procedure can be used to position the clones in a row across the screen. The variable “X Position” is set to an initial point at the left side of the stage. Each successive clone is then moved to a position to the right of the clone before. (The **Join** block is a string operator that is used to ensure that numbers such as “60” are treated as the name of a sprite rather than as a number that may be used in an arithmetic operation.)


```
+Position+Clones+
set X Position to -150
for i = 60 to 72
    tell (join i) to (go to x: (X Position) y: 0)
    change X Position by 20
```

After the **Position Clones** block is run, the clones will appear in a row across the screen.

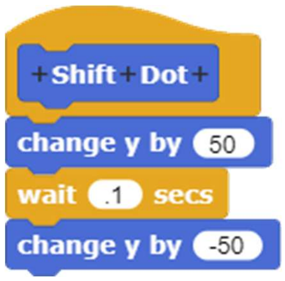


After this setup, a Dot Piano block can be created to shift each dot upward when the corresponding note on the external MIDI keyboard is played. The *Changed Note* variable can be used to determine when a new note has been played.



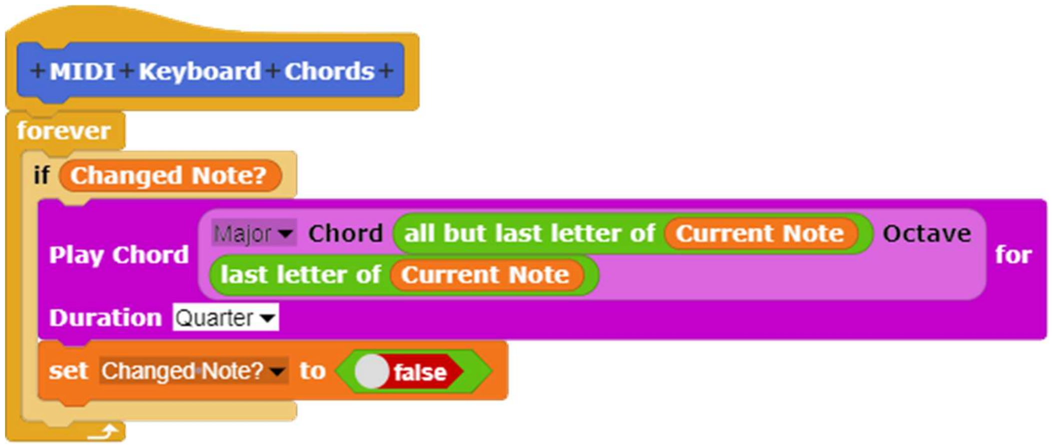
```
+Dot+Piano+
forever
  if Changed Note?
    tell join Current Note MIDI ◂▸ to Shift Dot ▸ ▸
    set Changed Note? ▾ to false
```

The **Shift Dot** block moves the dot upward for a moment and then back downward again.



```
+Shift+Dot+
change y by 50
wait .1 secs
change y by -50
```

Many other musical variations are possible. For example, the following procedure plays the chord corresponding to each note played.

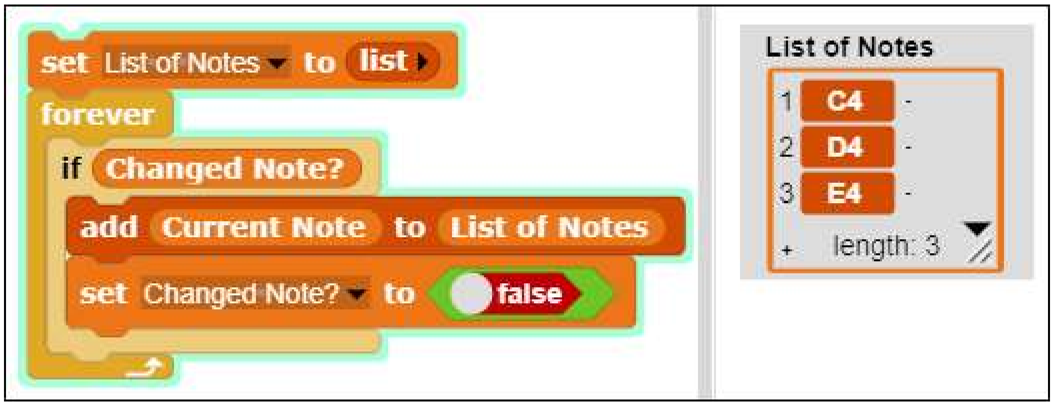


```
+MIDI+Keyboard+Chords+
forever
  if Changed Note?
    Play Chord  Major ▾ Chord all but last letter of Current Note  Octave   for
                last letter of Current Note
    Duration Quarter ▾
    set Changed Note? ▾ to false
```

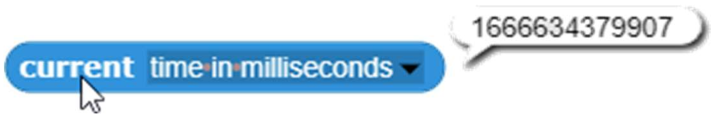
If a note such as “C#4” is played, the string operator **All But Last Letter** separates the note “C#” from the octave “4”. Similarly, the **Last Letter** block extracts the octave.

7.8 Recording Notes with a MIDI Keyboard

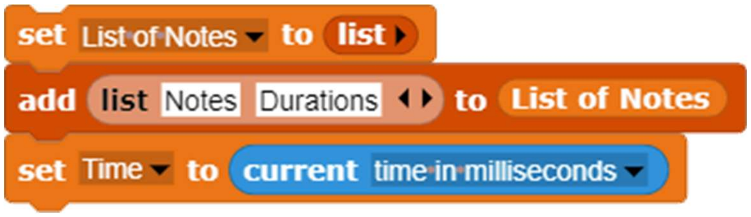
A MIDI keyboard can also be used as an input device to play and record a list of notes. If the current note changes, the new note is added to the variable *List of Notes*. The *Changed Note* variable is reset to *False* after the note is recorded. The loop then waits until the next note is played.


```
set List of Notes to list
forever
  if Changed Note?
    add Current Note to List of Notes
    set Changed Note? to false
```

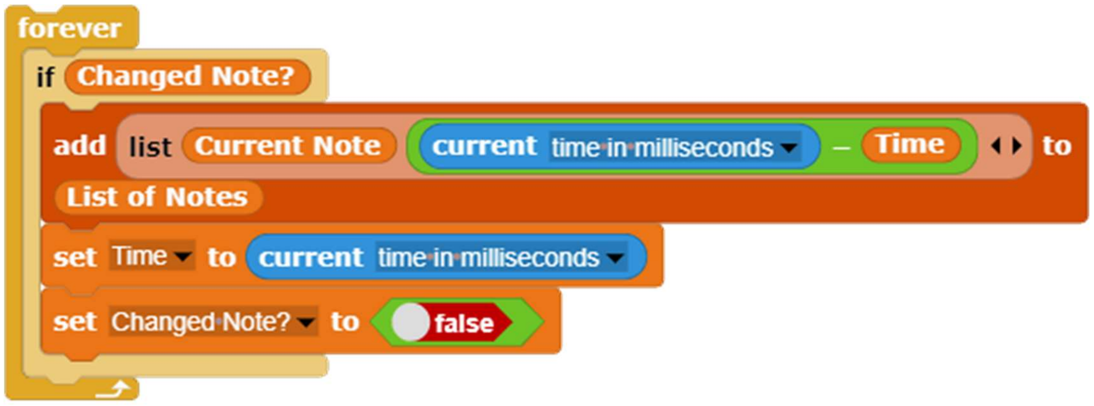
The **Current Time in Milliseconds** reporter block can also be used to record the duration between notes in milliseconds.



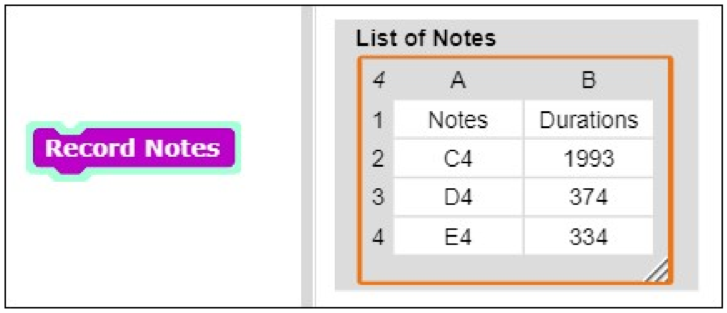
The **Record Note** procedure is initialized by creating a table with the heading “Notes” for the first column of the table and the heading “Durations” for the second column of the table. The current time in milliseconds is also recorded.


```
set List of Notes to list
add list Notes Durations to List of Notes
set Time to current time in milliseconds
```

In the main loop of the procedure, the time between the keypresses is recorded by subtracting the current time that a new key is pressed from the previously recorded time. The current note and duration are then added to the table.


forever
if Changed Note?
add list Current Note current time in milliseconds – Time to List of Notes
set Time to current time in milliseconds
set Changed Note? to false

Any key on the MIDI keyboard is pressed to begin. The duration in milliseconds between keypresses is then recorded for each subsequent note.



4	A	B
1	Notes	Durations
2	C4	1993
3	D4	374
4	E4	334

Since the previously recorded time is indeterminate for the first note, the duration for the first note is not meaningful and can be discarded.

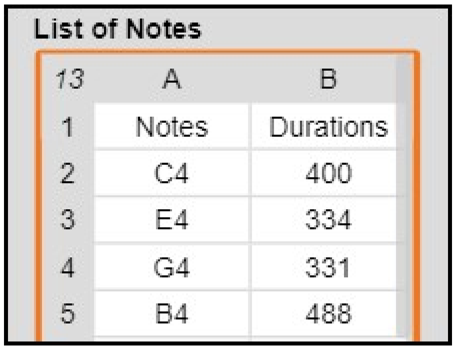
7.9 Musical Patterns

Once a sequence of notes has been created, either by capturing notes played on an external MIDI keyboard or by manually typing notes into a list, a corresponding duration must be provided for each note. A series of utilities for applying musical patterns to a list of notes. These Musical Patterns utilities can be accessed through the link below:

Musical Patterns

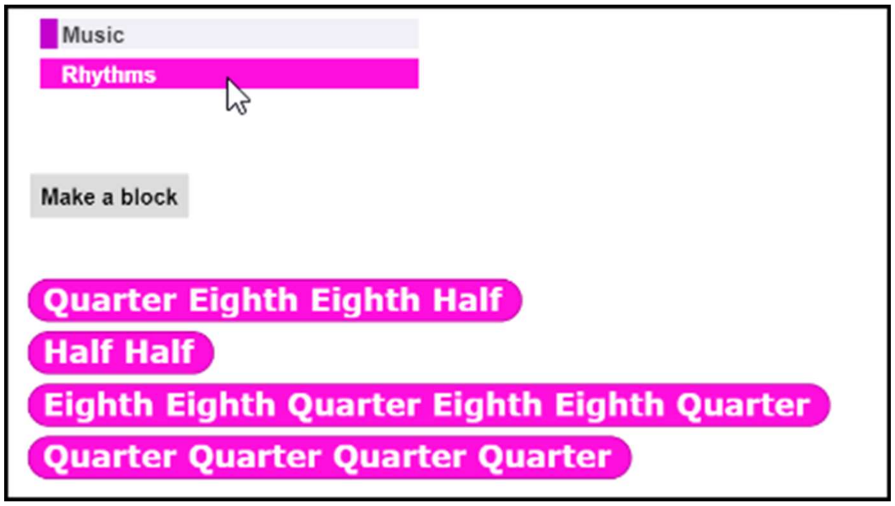
These utilities include the following blocks: a **Record Notes** block to record notes and durations from an external MIDI keyboard, pattern blocks, an **Apply Rhythm to Note List** block to apply the patterns to a recorded list of notes, a **Segment of Note List** block that displays a selected segment of a list of notes, and an **Add Segment to Melody** block that adds a selected segment to a list of notes.

The **Record Notes** block records the time elapsed between each note played on the MIDI keyboard. That provides an approximation of note durations that can be converted into musical terms such as quarter note, half notes, etc.

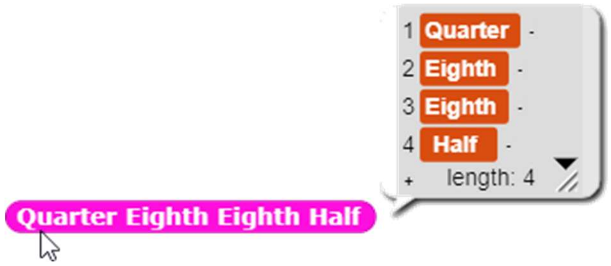

List of Notes

13	A	B
1	Notes	Durations
2	C4	400
3	E4	334
4	G4	331
5	B4	488

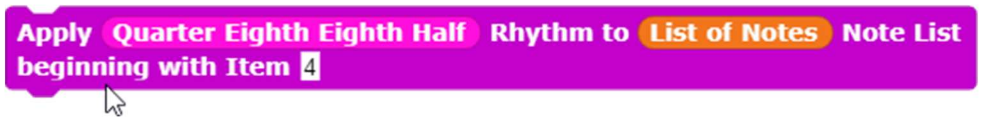
These utilities for construction of musical patterns include a *Rhythms* palette with a number of musical patterns.


Music
Rhythms
Make a block
Quarter Eighth Eighth Half
Half Half
Eighth Eighth Quarter Eighth Eighth Quarter
Quarter Quarter Quarter Quarter

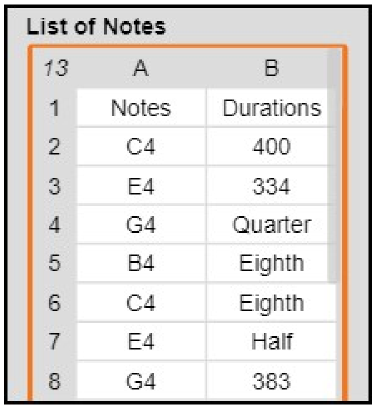
For example, the **Quarter Eighth Eighth Half** reporter block reports that sequence of durations:



The **Apply Rhythm to Note List** block can be used to apply the pattern of durations to a list of notes.


Apply Quarter Eighth Eighth Half Rhythm to List of Notes Note List beginning with Item 4

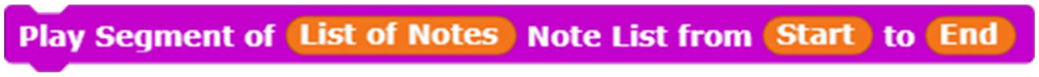
The selected pattern of durations is then applied to the list of notes beginning with the selected item in the list (Item 4, in this example).



List of Notes

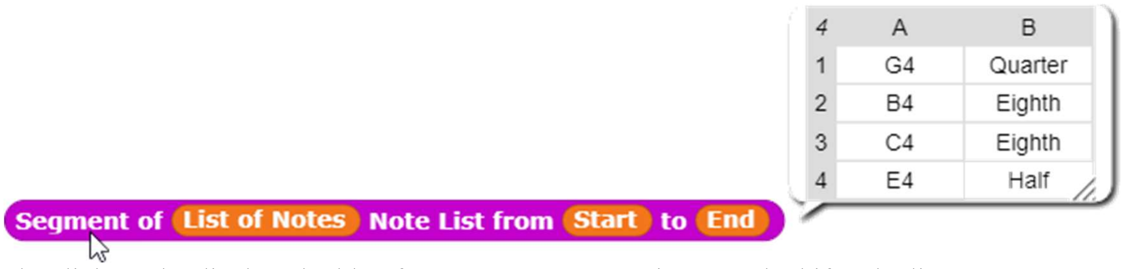
13	A	B
1	Notes	Durations
2	C4	400
3	E4	334
4	G4	Quarter
5	B4	Eighth
6	C4	Eighth
7	E4	Half
8	G4	383

The starting and end points of the segment are recorded as global variables *Start* and *End*. These variables can be used with the **Play Segment** block to play the pattern.



```
Play Segment of (List of Notes) Note List from (Start) to (End)
```

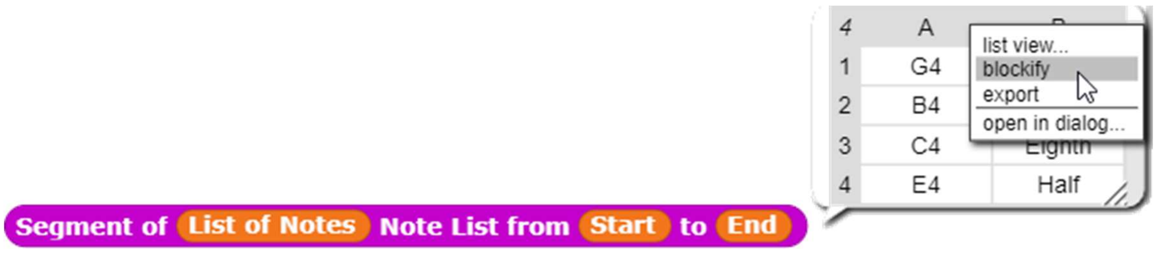
The **Segment of Note List** reporter block can be used to display the selected segment.



```
Segment of (List of Notes) Note List from (Start) to (End)
```

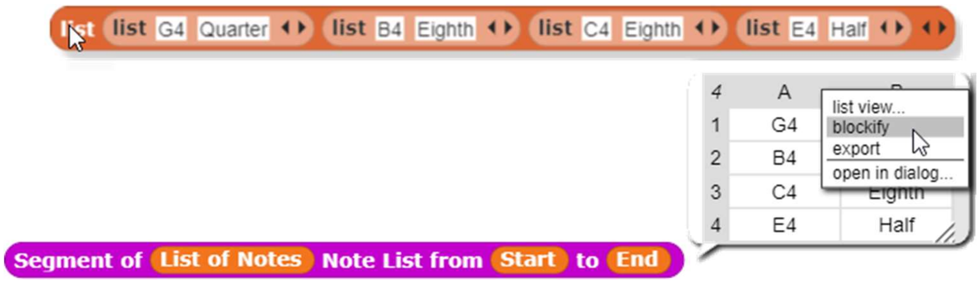
4	A	B
1	G4	Quarter
2	B4	Eighth
3	C4	Eighth
4	E4	Half

A right-click on the displayed table of notes presents an option to "Blockify" the list.



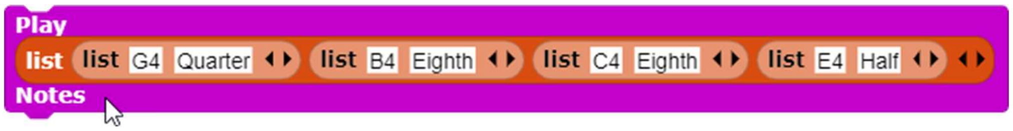
```
Segment of (List of Notes) Note List from (Start) to (End)
```

The *blockify* option creates a list of the notes and durations:

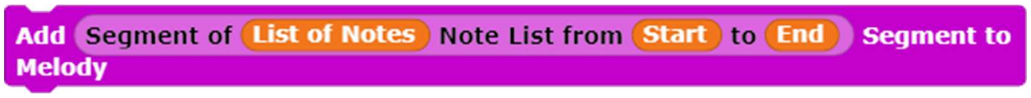


```
list (list G4 Quarter) (list B4 Eighth) (list C4 Eighth) (list E4 Half)
Segment of (List of Notes) Note List from (Start) to (End)
```

The list of the notes can then be played in the same manner as any other list of notes:



The **Add Segment to Melody** block adds the selected segment to a list named *Melody*.

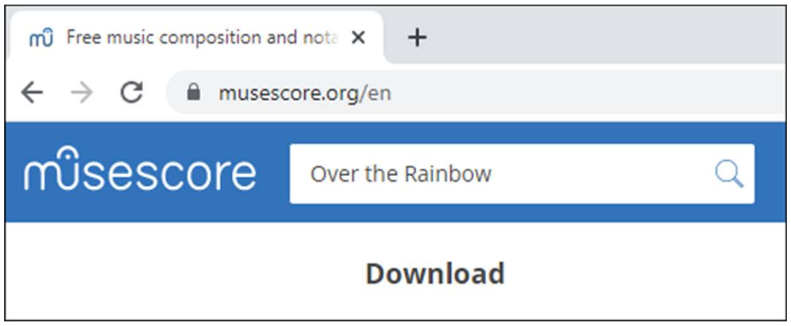


Through a process of exploration, sequences of notes can be generated and combined with patterns of durations. The segments of notes created in this way can then be combined to form a melody.

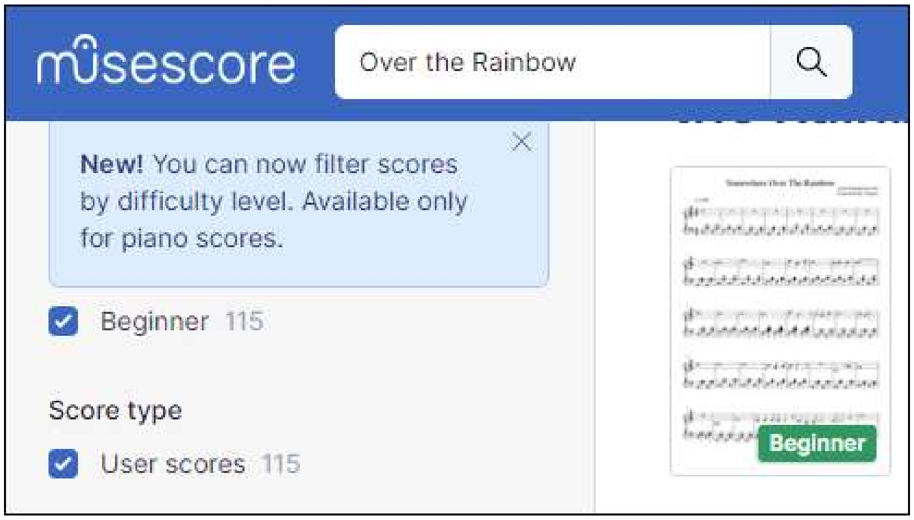
MIDI Appendix

A. Editing MIDI Files with Music Notation Programs

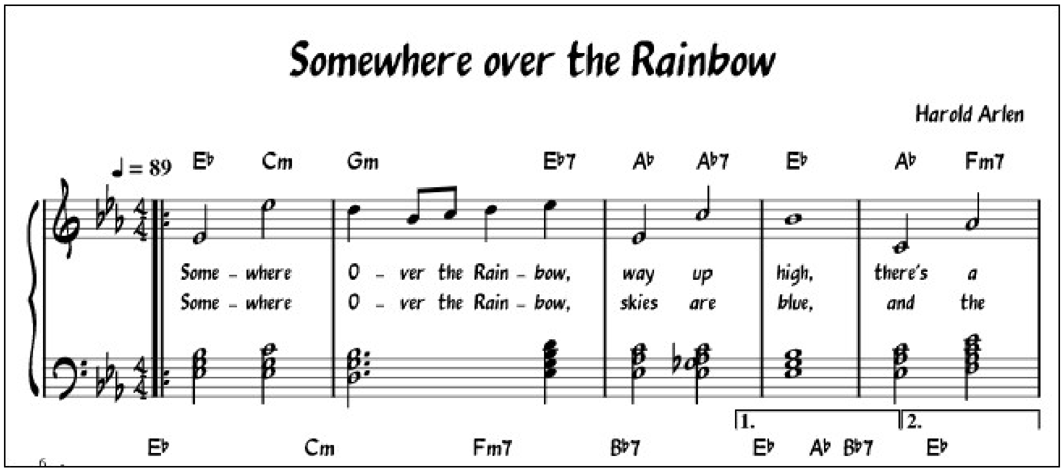
Music notation programs such as *NoteFlight* and *MuseScore* can be used to import and edit MIDI files. These programs can also include search engines that can be used to search for songs.



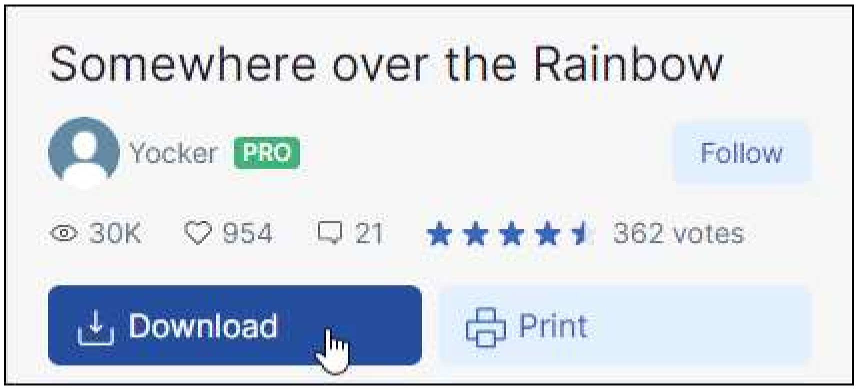
Music scores can be filtered by categories such as "Beginner" and "User Scores". Filtering search results for scores that have been created by individual users (rather than by a corporation) is more likely to yield music scores that can be edited without a fee.


musescore
Over the Rainbow
New! You can now filter scores by difficulty level. Available only for piano scores.
Beginner 115
Score type
User scores 115
Beginner

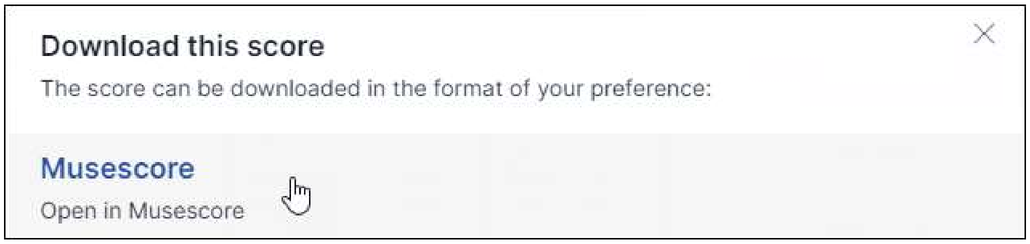
Once a music score for a song is identified, the score looks like this:


Somewhere over the Rainbow
Harold Arlen
♩= 89
E♭ Cm Gm E♭7 A♭ A♭7 E♭ A♭ Fm7
Some - where O - ver the Rain - bow, way up high, there's a
Some - where O - ver the Rain - bow, skies are blue, and the
1. 2.
E♭ Cm Fm7 B♭7 E♭ A♭ B♭7 E♭

The score can then be downloaded:



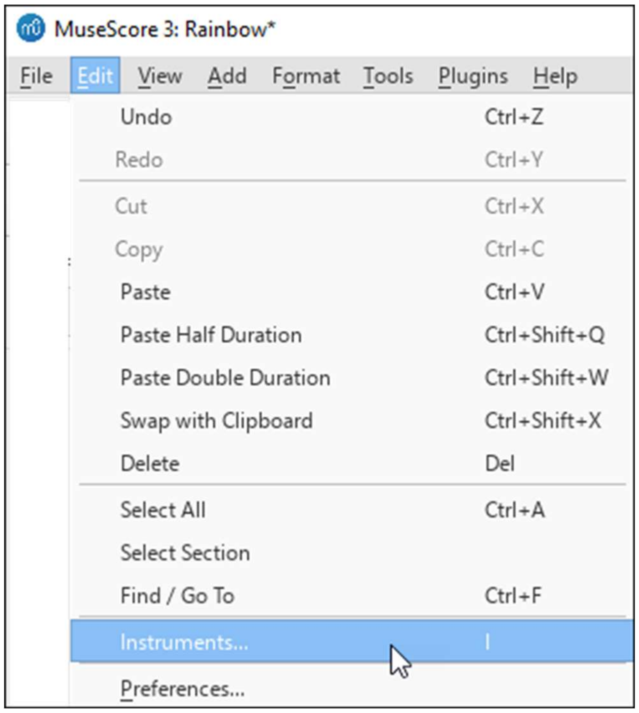
Options for several formats, including Musescore, are available:



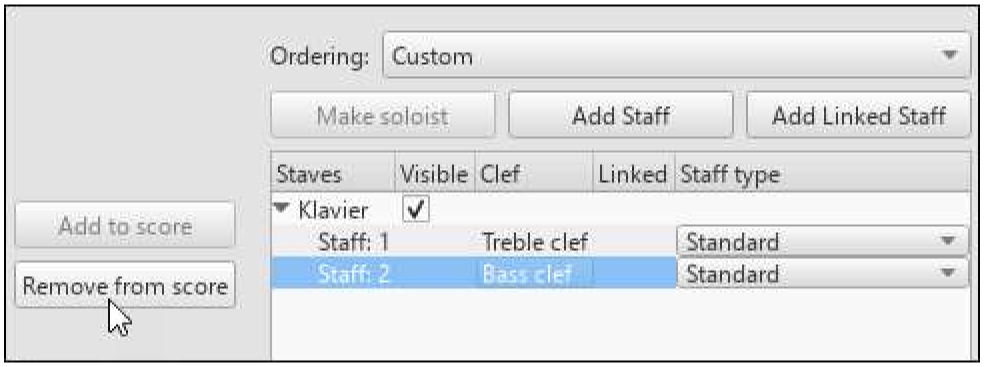
Download this score

The score can be downloaded in the format of your preference:

Once the file is exported and saved in native Musescore format, it can be re-opened in Musescore and edited. The native Musescore format includes the option of editing the musical instruments included in the score:

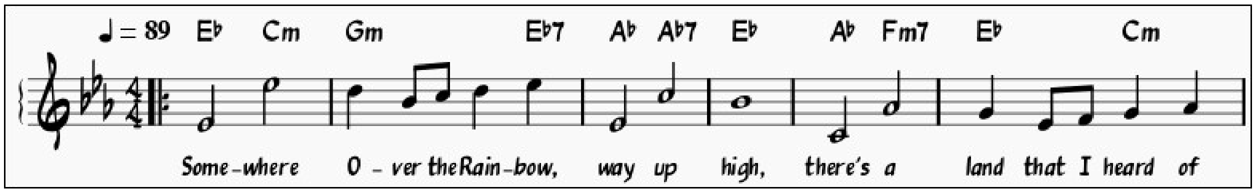


In the example below, the individual notes of the melody are in the treble clef (the top staff of the score) and the chords are in the bass clef (in the bottom staff). To extract the melody, the bass cleff can be removed from the score.

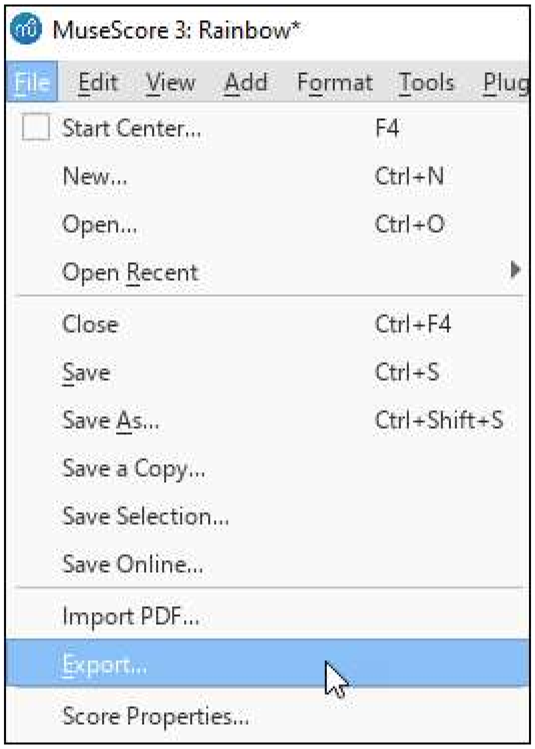


Staves	Visible	Clef	Linked	Staff type
Klavier	✓			
Staff: 1		Treble clef		Standard
Staff: 2		Bass clef		Standard

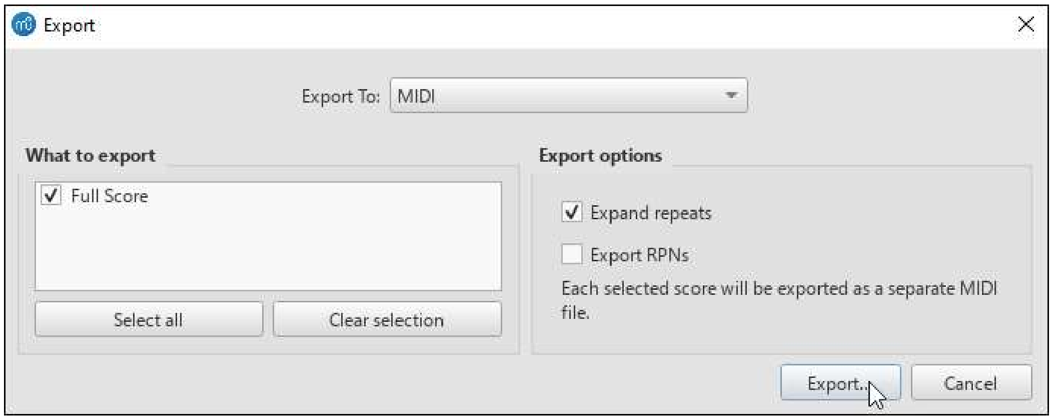
The resulting score now consists of just the melody.


♩ = 89
E♭ Cm Gm E♭7 A♭ A♭7 E♭ A♭ Fm7 E♭ Cm
Some-where O - ver the Rain-bow, way up high, there's a land that I heard of

The melody obtained in this way can now be exported for use in TuneScope.



The score should be exported in MIDI format.



Export To: MIDI

What to export

- [✓] Full Score

Export options

- [✓] Expand repeats
- [ ] Export RPNs

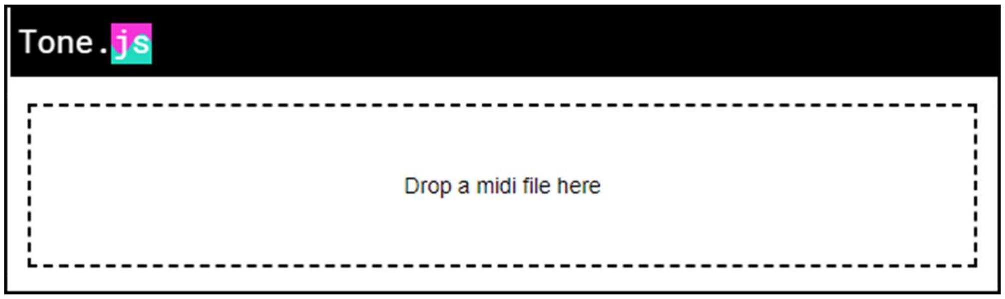
Each selected score will be exported as a separate MIDI file.

B. Converting MIDI Files to JSON Format

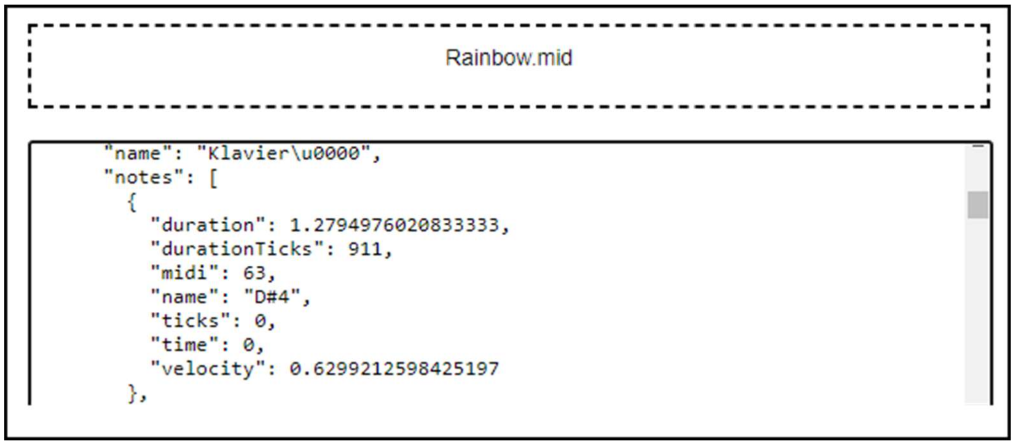
There are a number of programs that can convert a MIDI file into a human-readable JSON format. This example uses the Tone.js MIDI conversion program:

https://tonejs.github.io/Midi/

The Tone.js MIDI conversion interface provides an area into which a MIDI file can be dropped.



The converted file appears in a text window below.



```
      "name": "Klavier\u0000",
      "notes": [
        {
          "duration": 1.2794976020833333,
          "durationTicks": 911,
          "midi": 63,
          "name": "D#4",
          "ticks": 0,
          "time": 0,
          "velocity": 0.6299212598425197
        },
```

The file in this window can then be copied to the clipboard and pasted into a text editor such as NotePad or NotePad++. The file can then be saved with a ".json" extension. The ".json" file obtained in this way can then be imported into Snap*!* by dragging the file into the script area of Snap*!*

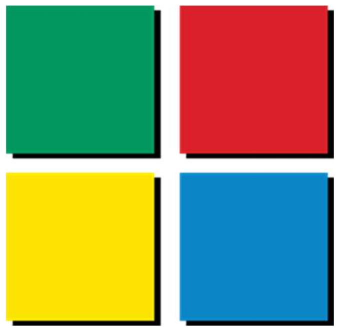
Chapter 8. Musical Games

Glen Bull and Jo Watts

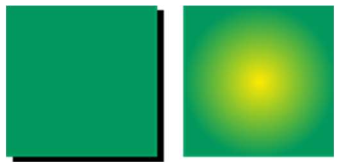
The musical game *Simon*, introduced in 1978, initiated an era of electronic games (Smithsonian Magazine). A series of tones light up panels on the game. The player must repeat the sequence by touching each panel in the correct order. The musical sequence becomes progressively longer and more complex. The game was an immediate hit and is still sold today.

Electronic games were made possible by small, affordable computer chips. The computer chip in the *Simon* game synthesized musical tones just as *TuneScope* does.

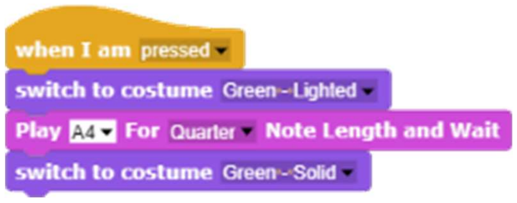
The computing power underlying applications like *TuneScope* makes it possible for anyone with the interest to create musical games. The TuneScope version of the musical sequencing game is recreated through creation of a green, red, yellow, and blue sprite.



Each sprite has two costumes: a solid-colored costume, and a second costume with a radial gradient that begins with a solid color at the perimeter and fades to a yellow shade in the center. The sprite switches to the second costume when a tone is played.



The **Switch to Costume** code block is used to switch to the lighted costume when the right button of the mouse is pressed while the mouse is over the sprite. When the mouse button is released, the sprite plays a note and then switches back to the solid costume.

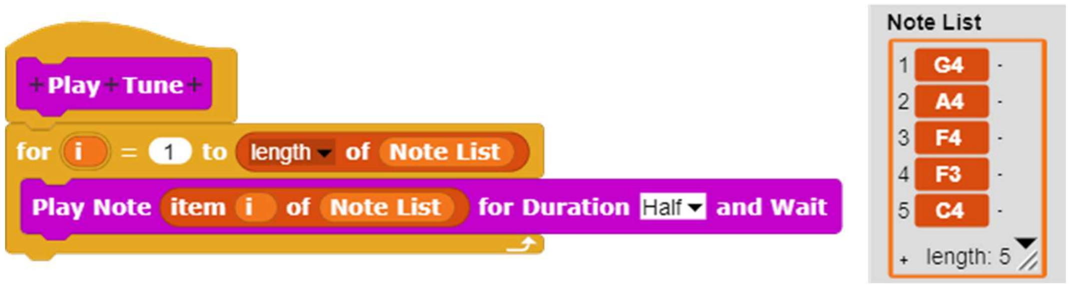


```
when I am pressed
switch to costume Green-Lighted
Play A4 For Quarter Note Length and Wait
switch to costume Green-Solid
```

In the movie *Close Encounters of the Third Kind* scientists use a music synthesizer to communicate with the aliens' space ship:

- Start with the note - "G"
- Up a full note - "A"
- Down a major third – "F"
- Now drop an octave – "F" (an octave lower)
- Up a perfect fifth – "C"

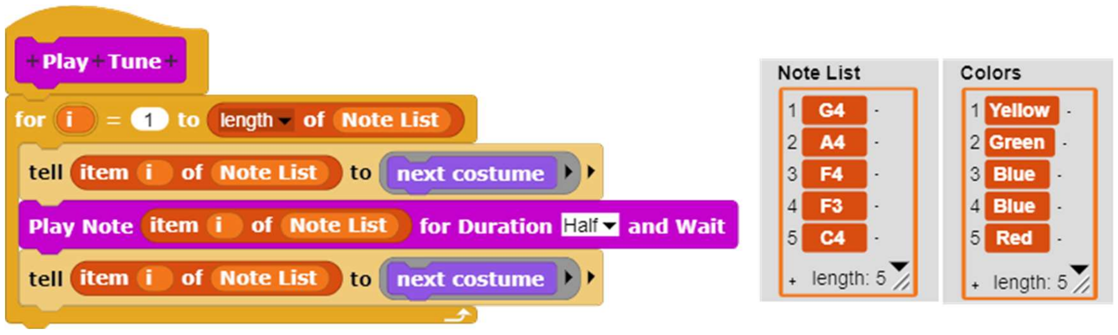
Because the space ship in the movie resembled the musical game, the movie contributed to the popularity of the game. Therefore, it seems appropriate to use this series of notes in this illustration. The now-familiar **For** loop is used to play a list of notes contained in the variable *Note List*.



```
+Play+Tune+
for i = 1 to length of Note List
    Play Note item i of Note List for Duration Half and Wait
```

Note List	
1	G4
2	A4
3	F4
4	F3
5	C4
length: 5	

Each note in the *Note List* has a color associated with it in a parallel list of *Colors*. The **Play Tune** procedure is extended to make use of the colors. The sprite of the corresponding color is told to switch to the next costume before the note is played. For example, before the first note in the *Note List* (G4) is played, the *Yellow* sprite is told to switch to the next (highlighted) costume. After the note is played, the *Yellow* sprite is told to switch back to the solid costume.

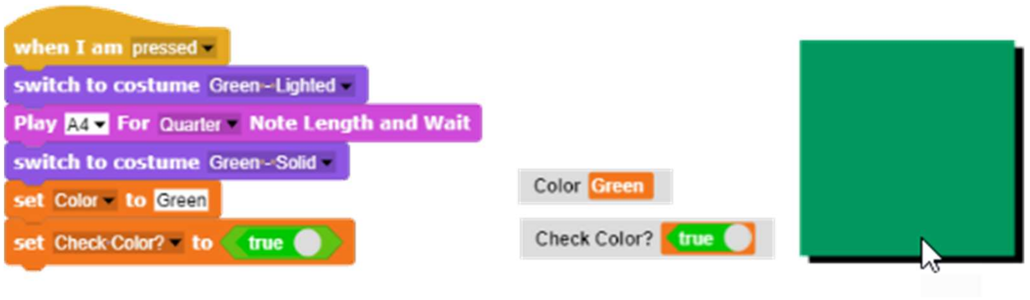


```
+Play+Tune+
for i = 1 to length of Note List
    tell item i of Note List to next costume
    Play Note item i of Note List for Duration Half and Wait
    tell item i of Note List to next costume
```

Note List		Colors	
1	G4	1	Yellow
2	A4	2	Green
3	F4	3	Blue
4	F3	4	Blue
5	C4	5	Red
length: 5		length: 5	

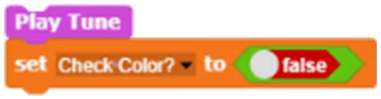
As a result, the sprite associated with each note is highlighted when the note is played.
A **Play Game** procedure is used to play the tune, lighting up the associated squares as each note is played. The player must click each square in the same order as the notes in the original tune.

Two additional blocks are added to each sprite's script (i.e., the scripts associated with the green, red, yellow and blue squares). The **Set Color** code block sets a variable named *Color* to the color of the block. The **Set Check Color?** block then sets a second variable named *Check Color* to a value of *True*. (The **True / False** block is found in the *Operators* section of the *Code Block Palette*.)



```
when I am pressed
switch to costume Green--Lighted
Play A4 For Quarter Note Length and Wait
switch to costume Green--Solid
set Color to Green
set Check Color? to true
```

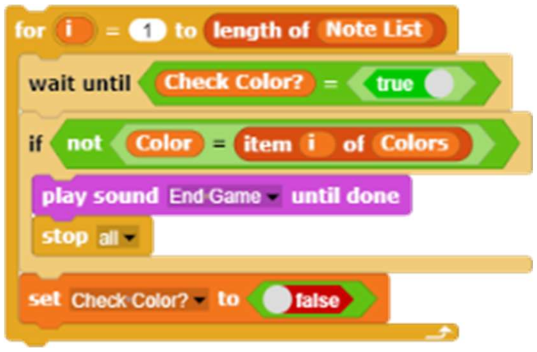
The **Play Game** procedure first plays the tune, lighting up the corresponding squares, and then sets the value of the *Check Color?* variable to *False*. This will cause the procedure to pause at this point, waiting until a square is clicked.



```
Play Tune
set Check Color? to false
```

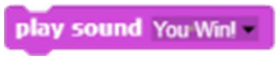
Clicking one of the squares then changes the value of the *Check Color?* variable to *True*, enabling the **Play Game** procedure to advance to the next group of code blocks.

This group of blocks consists of a *For* loop that waits until the value of the *Check Color?* variable is changed to *True*. When a square is clicked, an *If* statement is used to check whether the color of the square clicked is the same color as the corresponding item in the list of colors. If the colors do not match, an *End Game* sound is played and the **Stop All** code block is used to end the game.



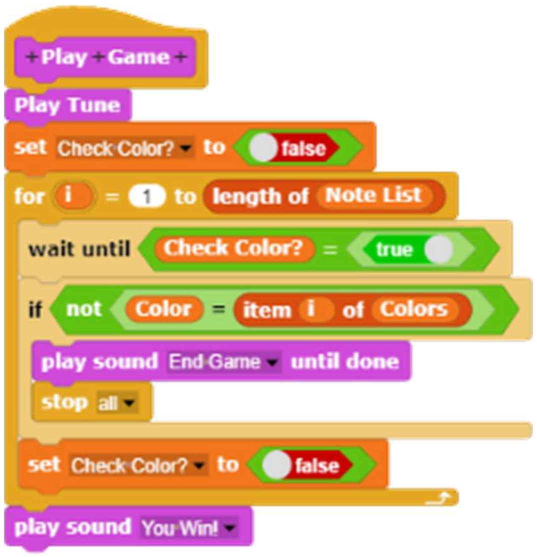
```
for i = 1 to length of Note List
  wait until Check Color? = true
  if not Color = item i of Colors
    play sound End Game until done
    stop all
  set Check Color? to false
```

If the colors do match, the *Check Color?* variable is reset to *False* again. The **For** loop then waits until the next square is clicked, setting the *Check Color?* variable to *True*.If all the squares are selected in the correct order, a "You win!" sound (a trumpet fanfare) is played.



```
play sound You Win!
```

The complete **Play Game** procedure looks like this. The **Play Game** procedure starts the game and then waits until a square is clicked. If the color of the square clicked is correct, the procedure waits until the next square is clicked. The game ends either when an incorrect square is clicked, or when the player successfully completes the entire sequence in the correct order.

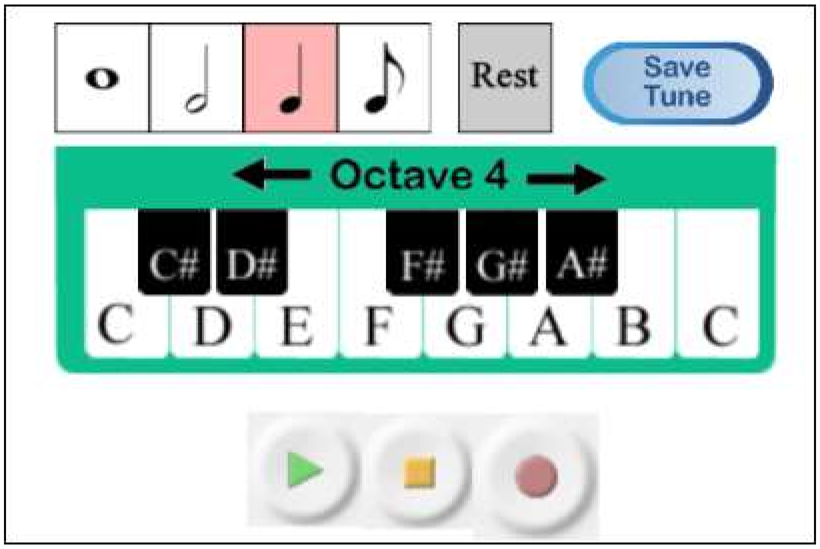


```
+Play+Game+
Play Tune
set Check Color? to false
for i = 1 to length of Note List
    wait until Check Color? = true
    if not Color = item i of Colors
        play sound End Game until done
        stop all
    set Check Color? to false
play sound You Win!
```

This basic musical game procedure could be adapted and refined. For example, the game could be modified so that it initially plays a single note, and then adds an additional note each time the player is successful.

One limitation of the game is that it can only be used with four notes. (The notes “F4” and “F3” are both assigned to the blue square in this illustration.) The number of notes could be extended through variants of the game such as creation of a musical version of the “Whack a Mole” game. In this variant, moles could pop up as each note was played. (In another variation, memorizing a tune might allow a player to anticipate where the next mole is scheduled to pop up.)

A similar technique could be used to create a player piano such as the one shown below.



Once a tune was recorded, the player could be challenged to play the same sequence of notes using the keys on the piano keyboard.

Chapter 9. Acoustic Tools

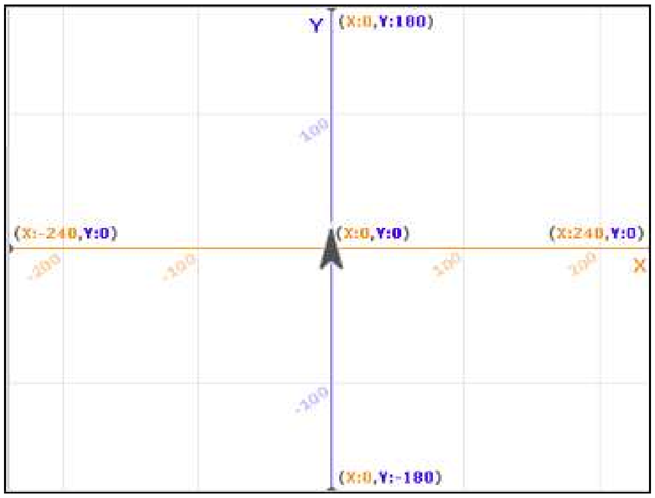
Glen Bull, Jo Watts, and Joe Garofalo

The process of recording a sound was described in the previous chapter. The *Record* function is found under the *Sounds* tab. The process of recording a sound converts the back-and-forth movements of a computer's microphone into an electrical signal. The amplitude of the varying electrical signal is repeatedly measured and converted into a series of digital numbers. This process is known as analog-to-digital (A/D) conversion. The process indirectly measures minute movements of the microphone and converts that movement into a series of digital numbers.

Even a short segment of digitized sound may consist of thousands of sound samples. If the digitization rate is 48,000 samples per second (for example), a single second of speech will result in a list of 48,000 numbers. These numbers represent the voltage levels generated by the back-and-forth movement of the diaphragm of the microphone used to capture the digitized sound.

In article "The Magic Number Seven, Plus or Minus Two" the psychologist George Miller reports that the average individual can only hold about seven numbers in short term memory at one time. (This is one of the most cited scientific papers of all time.) The limits may vary by one or two, but under no circumstance is it possible for a person to process or understand a list of 48,000 numbers.

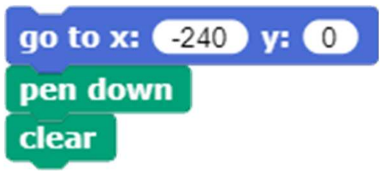
This list of numbers representing a second of digitized speech, however, is more comprehensible if it is presented as a graphical representation rather than in number form. The coordinate system for the default *Snap!* stage spans the range from -240 to +240 units on the horizontal (X) axis, and from -180 to +180 units on the vertical (Y) axis.



These coordinates provide the framework for graphing any type of data in *Snap!,* including a segment of sound.

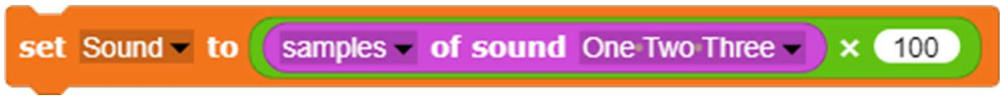
Topic 9.1 Graphing Sound Samples

Prior to graphing the waveform, the turtle is positioned on the left side of the screen. The screen is cleared, and the turtle's pen is lowered.



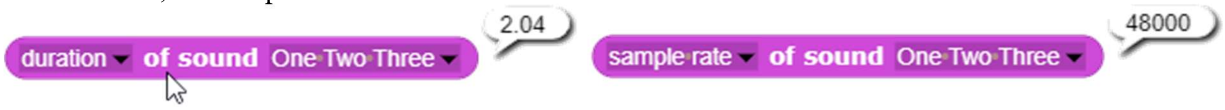
```
go to x: -240 y: 0
pen down
clear
```

Depending on the computer, the digitized sound samples may consist of numbers such as 0.27 and 0.42. To be clearly seen, the positive peaks of the waveform should ideally fall between 20 and 100 on the vertical axis of the stage. The sound samples can be multiplied by an appropriate number to increase the scale if necessary. In the example below, the samples of the sound utterance "One Two Three" have been multiplied by 100 and assigned to a variable named *Sound*.



```
set Sound to (samples of sound One Two Three) × 100
```

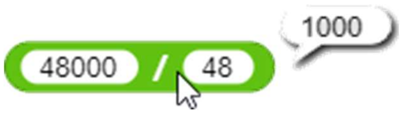
Once these setup and scaling activities are completed, a loop can be used to plot the sound samples. The utterance is slightly more than 2 seconds long. At a sample rate of 48,000 samples per second, more than 96,000 samples were collected.



```
duration of sound One Two Three
2.04

sample rate of sound One Two Three
48000
```

Dividing 48 into 48,000 yields the result of 1000. Consequently, if every 48th point is plotted, each point will represent a one millisecond time increment (since there are 1000 milliseconds in a second).

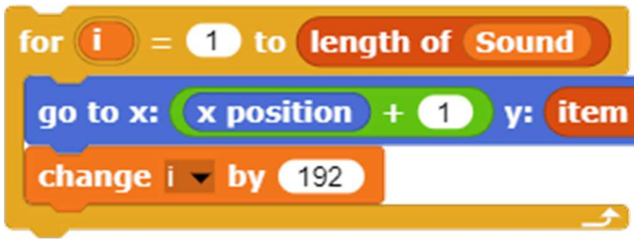


```
48000 / 48
1000
```

The default width of the stage is 480 steps. If each step in the graph represents a one millisecond time increment, it is still only possible to graph approximately a half-second of the utterance (0.48 seconds).

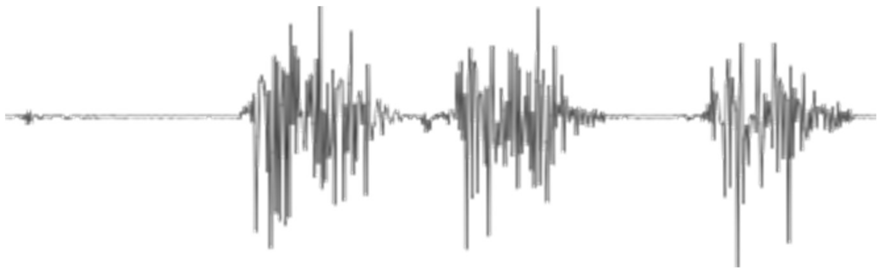
This time scale can be adjusted by calibrating the number of sound samples that are graphed. If every 96th sound sample is graphed, each point will represent two milliseconds, and it becomes possible to graph nearly a full second of sound on the stage. If every 192nd sound sample is graphed, each point will represent four milliseconds, and it becomes possible to graph nearly two seconds of sound on the stage.

Since the recording of the words "one, two, three" is approximately two seconds long, graphing every 192nd point results in a time scale that makes it possible to display the entire utterance on the stage. This can be accomplished with a For loop that graphs every 192nd sound sample in the utterance. The default increment for a **For** loop is 1; in this instance, the increment has been increased to every 192nd point. After each point is plotted, the turtle moves over one step. By this means, the amplitude of each sound sample is graphed.

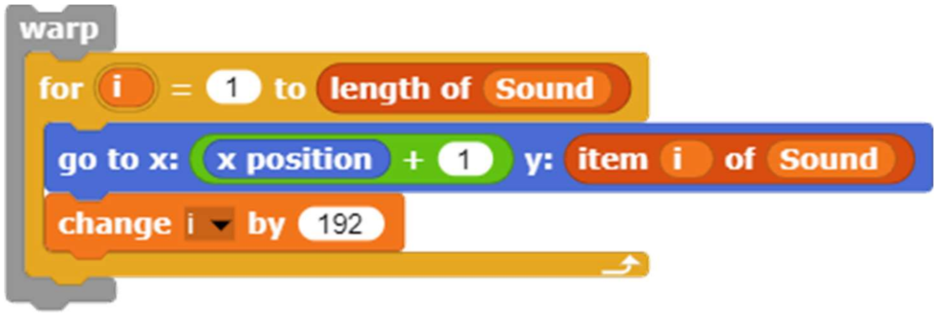


```
for (i) = 1 to length of Sound
    go to x: (x position + 1) y: item i of Sound
    change i by 192
```

The graph shown in the illustration below results. The words “one”, “two”, and “three” can be clearly seen. The up and down variations in amplitude correspond to minute variations in the back-and-forth movements of the diaphragm of the microphone that captured the sound.

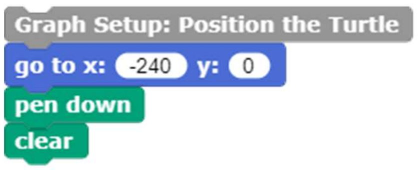


The graph can be displayed much faster if the procedure is placed in a **Warp** block. The **Warp** code block suspends all background operations until the plotting loop is completed.

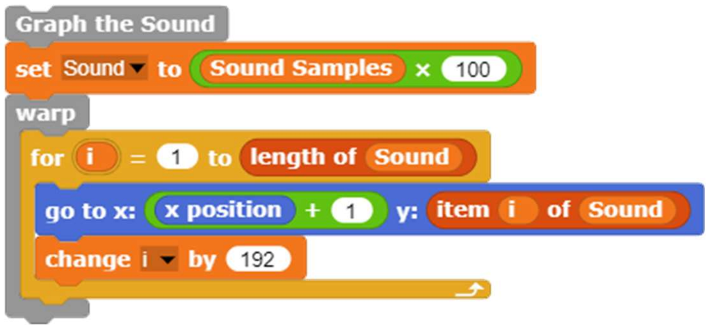


```
warp
    for (i) = 1 to length of Sound
        go to x: (x position + 1) y: item i of Sound
        change i by 192
```

The script to graph the sound has two components: (1) a graph setup procedure that positions the turtle and clears the screen, and (2) a loop that moves through the table of sound samples and graphs these values on the stage.



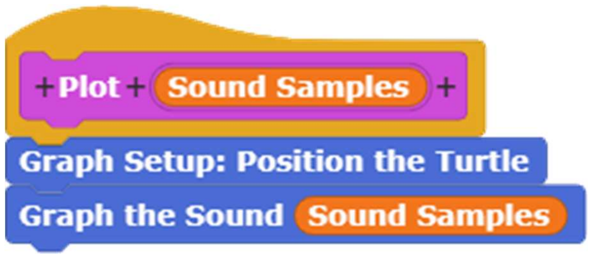
```
Graph Setup: Position the Turtle
go to x: -240 y: 0
pen down
clear
```



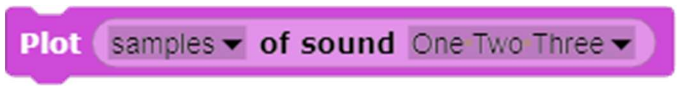
```
Graph the Sound
set Sound to (Sound Samples × 100)
warp
    for (i) = 1 to length of Sound
        go to x: (x position + 1) y: item i of Sound
        change i by 192
```

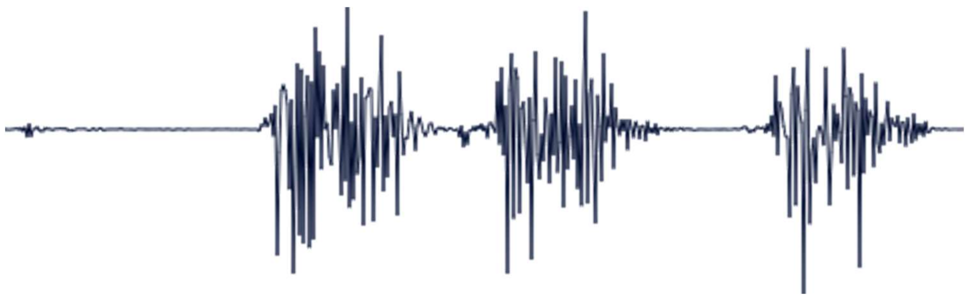
These two components can be used to build the code blocks **Graph Setup: Position the Turtle** and **Graph the Sound**. These two blocks, in turn, can be used to create the custom code block,

Plot Sound Samples, that plots the values of the sound samples to create a graph of the sound wave.


+Plot+ Sound Samples +
Graph Setup: Position the Turtle
Graph the Sound Sound Samples

The **Plot Sound Samples** code block accepts the samples of a recorded sound as an input and plots the results.





Displaying the sound samples graphically makes it possible to gain a better understanding of the characteristics of the waveform.

Exploration 9.1 Graphing Sound Samples

Use the **Plot Sound Samples** code block to graph different types of recorded sounds: speech, music, noise, etc. How does the display of the different types of sounds differ? Change the time scale of the graph displayed on the stage.

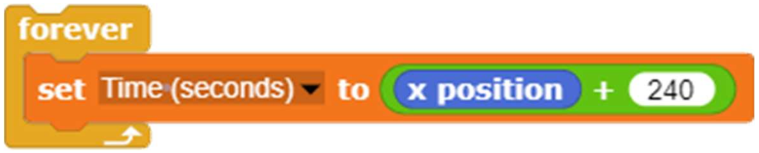
Topic 9.2 Time Scale

A sense of the time scale is often useful. For example, if 20 cycles of a waveform are displayed, determination of the frequency of the waveform requires calculation of the duration of the sound segment, which makes knowledge of the time scale a prerequisite for obtaining this information. Other characteristics of sound such as onset of the sound (the rate at which the amplitude of the waveform increases at the beginning of an utterance) require a known time scale.

In the previous section, every 192nd sound sample was graphed. Since 48,000 samples per second were acquired when the sound was digitized, each sound sample plotted represents a 4-millisecond time increment. Calculation of the timeline involves two steps:

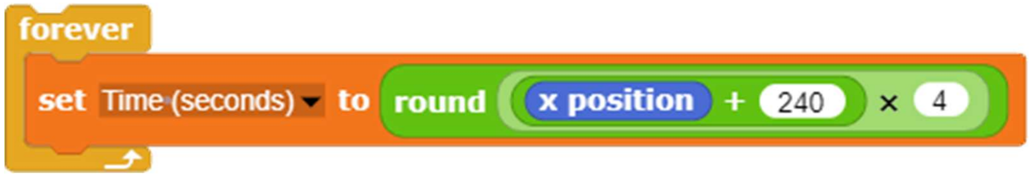
1. First, the absolute position of the turtle on the *X-axis* (i.e., the horizontal axis) must be determined. The **X-Position** code block can be used to obtain this value. Since the coordinates

of the default stage range from -240 on the left to +240 on the right, the position of the turtle ranging from 1 to 480 can be determined by adding 240 to the turtle's current location. The result of this calculation can be assigned to a variable named *Time (seconds)*.



```
forever
  set Time (seconds) to (x position) + 240
```

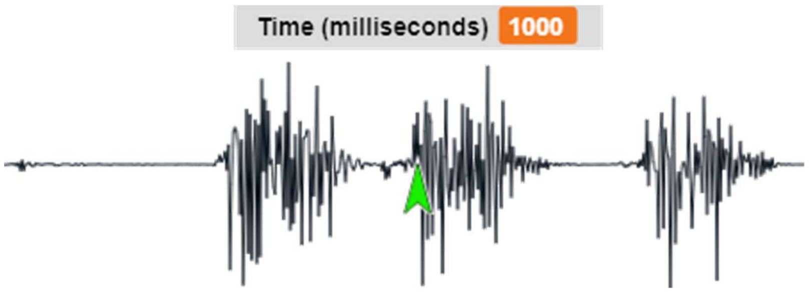
2. Since each sound sample represents a time increment of 4 milliseconds (when every 192nd sound sample is graphed, given a sampling rate of 48,000 samples per second), this result must be multiplied by four to obtain the location in milliseconds. For simplicity, this result is rounded to the nearest millisecond.



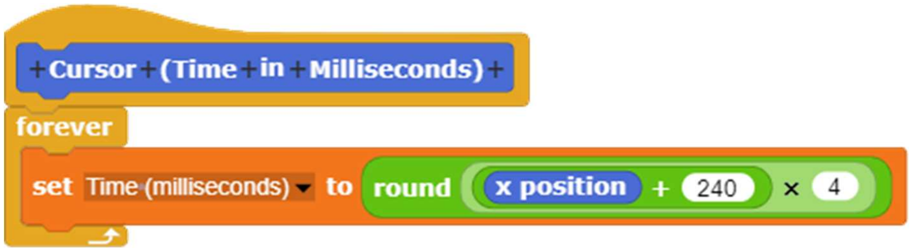
```
forever
  set Time (seconds) to round (((x position) + 240) × 4)
```

To obtain time in seconds, this result can be divided by 1000 (since there are 1000 milliseconds in a second). In subsequent calculations below, results will be displayed in milliseconds.

When this loop is executed, positioning the turtle (recast in the role of a time cursor) slightly to the right of the center of the stage yields a value of 1000 milliseconds (i.e., one second). Since the duration of the utterance "One, Two, Three" is about two seconds in length, this result confirms that the calculation of the time scale is accurate.



This code can be encapsulated in a custom code block that uses the turtle as a cursor to indicate the time in milliseconds based on its position on the horizontal axis.



```
+Cursor+(Time+in+Milliseconds)+
forever
  set Time (milliseconds) to round (((x position) + 240) × 4)
```

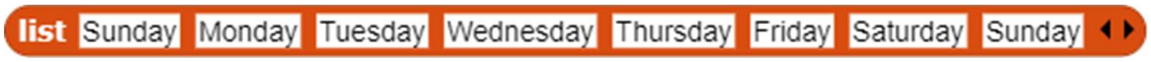
The ability to use the turtle as a marker that indicates the time scale is useful in a number of different analyses of the acoustic waveform.

Exploration 9.2 Time Scale

Use the turtle as a time marker to determine the start of the graph of the waveform of the word "One." Then determine the time in milliseconds at which the word ends. How long is the word "One"? How does this duration compare with the length of the words "Two" and "Three"?

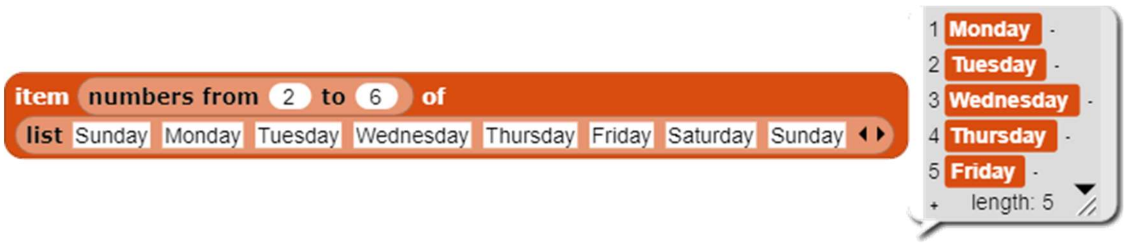
Topic 9.3 Sound Segments

There are times when it may be useful to isolate a portion of an utterance or a specific sound segment for analysis. The **Numbers from __ to __** code block can be used to isolate a portion of a list. For example, if only the workdays in a list of the seven days of the week are desired:



```
list Sunday Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

The list of workdays can be obtained by requesting Items 2 through 6 (Mon through Fri) in the list:

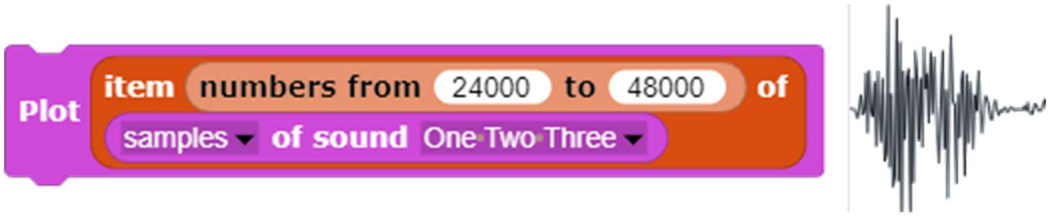


```
item (numbers from 2 to 6) of
  list Sunday Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

```
1 Monday
2 Tuesday
3 Wednesday
4 Thursday
5 Friday
length: 5
```

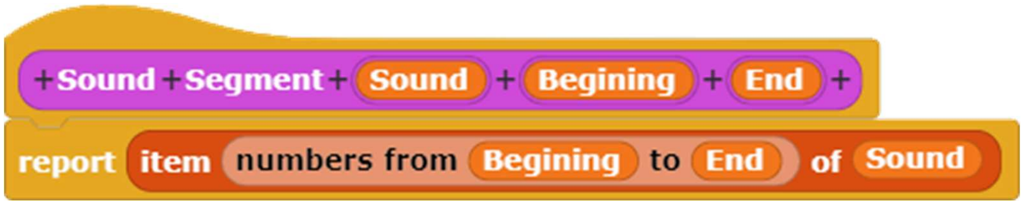
The same method can be used to obtain a portion of a sound recording. If there are 48,0000 sound samples in a second (for sound digitized at that rate), then *Items 24,000 to 48,000* will yield the portion of the utterance from 0.5 seconds to 1.0 seconds.

In fact, plotting *Items 24,000 through 48,000* of the utterance "One, Two, Three" yields the word "One" just as we would expect (since this word begins about a half-second into the recording).



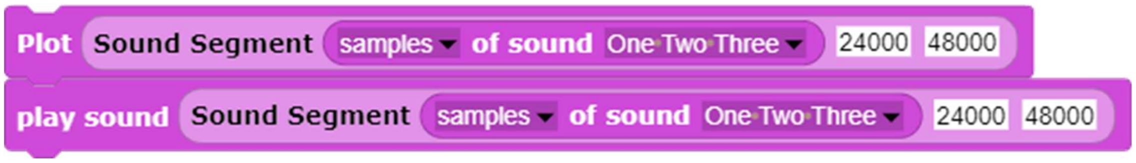
```
Plot item (numbers from 24000 to 48000) of
       samples of sound One·Two·Three
```

This capability can be incorporated into a custom **Sound Segment** code block.



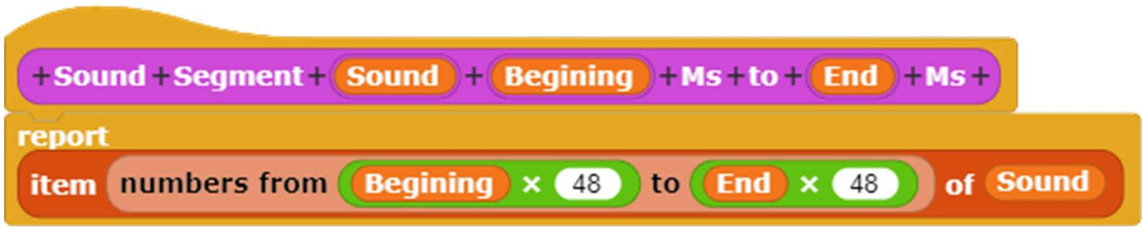
```
+Sound+Segment+(Sound)+(Begining)+(End)+
report item (numbers from (Begining) to (End)) of (Sound)
```

One benefit of creating a custom **Sound Segment** code block is that this code block can not only be used to graph a sound segment but also to play the sound segment to verify that the portion of the utterance obtained is the desired segment.



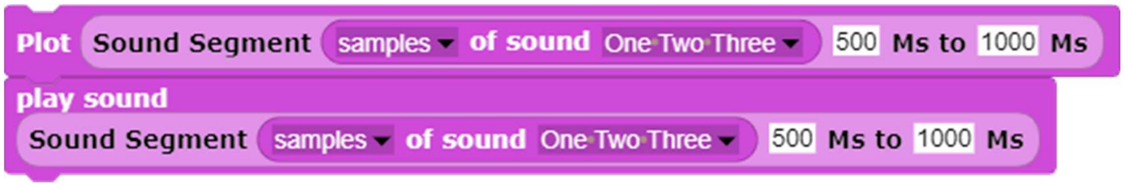
```
Plot Sound Segment (samples of sound One Two Three) 24000 48000
play sound Sound Segment (samples of sound One Two Three) 24000 48000
```

It may be more convenient to enter the start and end of the sound segment in milliseconds rather than in sound samples. This conversion can be accomplished by multiplying milliseconds by 48 (since 48 times 1000 milliseconds yields 48,000 sound samples). The addition of the label *Ms* after the input slots to indicate the units of measurement employed improves clarity.



```
+Sound+Segment+(Sound)+(Begining)+Ms+to+(End)+Ms+
report
  item numbers from (Begining × 48) to (End × 48) of Sound
```

The **Sound Segment** code block can now be used with the start and end of the desired sound segment indicated in milliseconds.



```
Plot Sound Segment (samples of sound One Two Three) 500 Ms to 1000 Ms
play sound
  Sound Segment (samples of sound One Two Three) 500 Ms to 1000 Ms
```

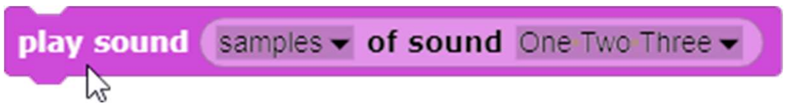
The time cursor developed in the previous section can be used to indentify the beginning and end of a desired sound segment. Once these values are obtained, they can be used as inputs to the Sound Segment code block to isolate portions of the utterance.

Exploration 9.3 Sound Segments

Use the **Sound Segment** code block to identify and isolate consonants and the vowels in the words “Two” and “Three”.

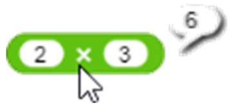
Topic 9.4 Amplification

When a digitized sound is played, the numbers in the list of sound samples control the displacement of the speaker cone. The larger the number, the greater the displacement. Greater displacement of the speaker cone, in turn, results in a sound that is perceived as louder.

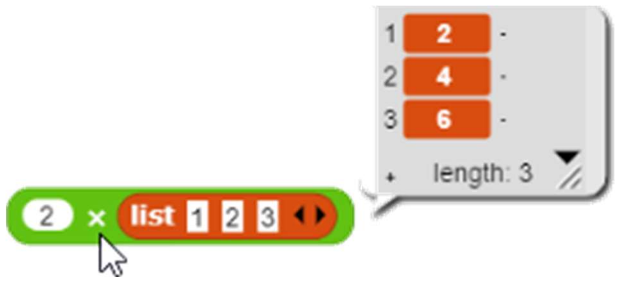


```
play sound (samples of sound One Two Three)
```

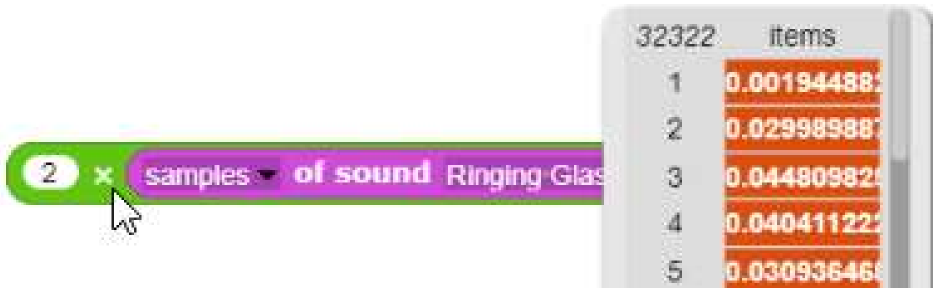
Therefore, the loudness of the sound can be increased by increasing the amplitude of the sound samples. The *multiplication operator* is used to multiply the value of a number.



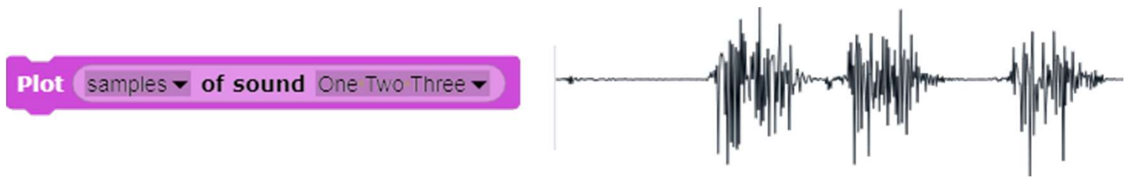
The multiplication operator can be used to multiply all of the numbers in a list.



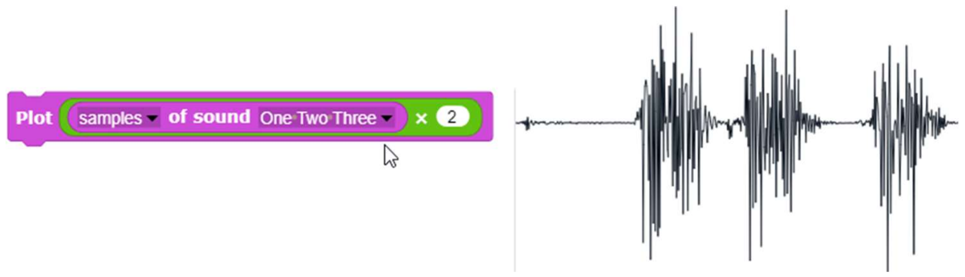
For example, each item in the list of sound samples can be doubled by multiplying by two.



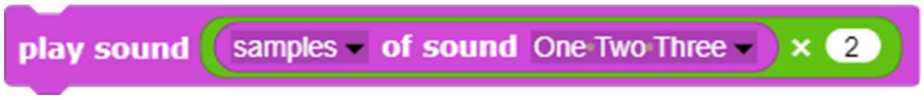
If the graph of the waveform results in the following plot,



then multiplying the values in the list of sound samples by two results in a graph that is double the amplitude of the previous one.

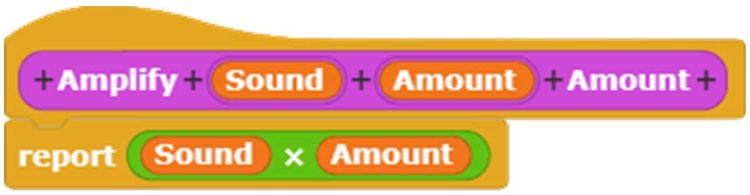


If the sound samples that have been increased in amplitude are played, the sound is perceived as louder.



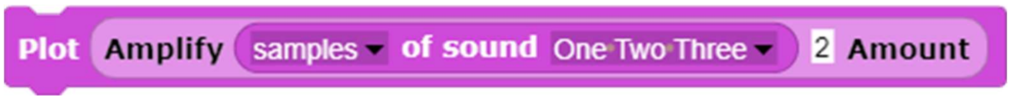
```
play sound (samples of sound One Two Three) × 2
```

This method can be used to create a custom **Amplify** code block to control the amplification of the sound.



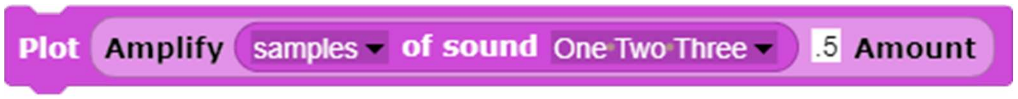
```
+Amplify+ Sound + Amount +Amount+
report Sound × Amount
```

If the amount of amplification is greater than one, the amplitude of the sound is increased.

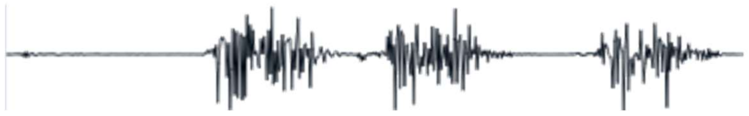


```
Plot Amplify (samples of sound One Two Three) 2 Amount
```

If the amount of amplification is less than one, the amplitude of the sound is decreased.



```
Plot Amplify (samples of sound One Two Three) .5 Amount
```



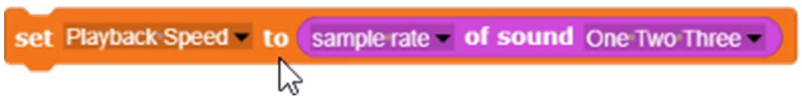
Exploration 9.4 Amplification

Use the **Amplify** code block to increase and decrease the amplitude of a digitized sound. Graph each result and play the resulting sound. How does the amplitude of the graph of the waveform correspond to the perceived loudness of the sound?

Topic 9.5 Adjusting Frequency

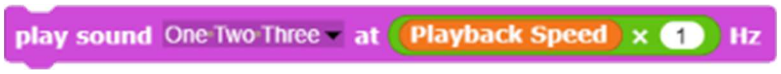
Previously, we adjusted the *amplitude* of a sound (perceived as loudness) by multiplying each number in the list of sound samples. Similarly, we can adjust the *frequency* (perceived as pitch) of the recording by varying the playback speed.

This can be accomplished by first setting a variable, *Playback Speed*, to the sampling rate. The default rate at which sound samples in TuneScope are recorded is 48,000 samples per second. Therefore, in most cases the sample rate will be 48,000 samples per second.



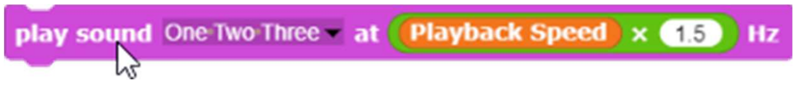
```
set Playback Speed to (sample rate of sound One Two Three)
```

When the sound is played back at the same speed at which it was recorded, a normal pitch is heard. In other words, the recorded frequency and the frequency played back are the same.



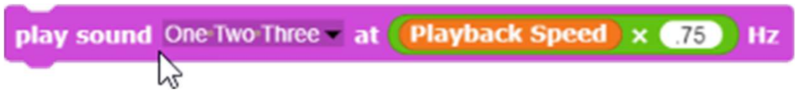
```
play sound One Two Three at (Playback Speed × 1) Hz
```

However, increasing the playback speed increases the frequency played back in comparison with the frequency at which the sound was recorded.



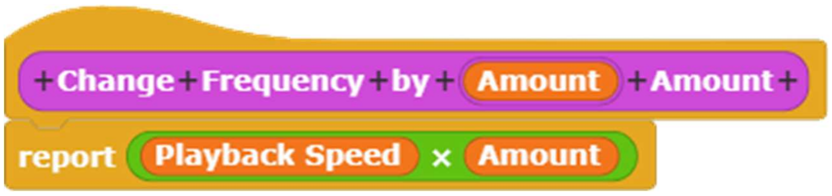
```
play sound One Two Three at (Playback Speed × 1.5) Hz
```

Similarly, decreasing the playback speed decreases the frequency.



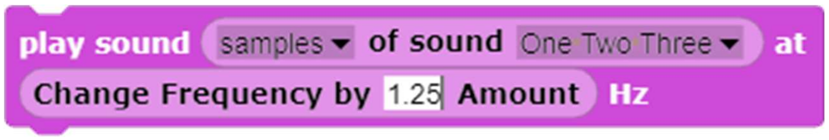
```
play sound One Two Three at (Playback Speed × .75) Hz
```

This method can be used as the basis for development of a custom code block to adjust the frequency of a sound.



```
+Change+Frequency+by+(Amount)+Amount+
report (Playback Speed × Amount)
```

This code block can then be used to adjust the frequency of a recorded sound when it is played back.



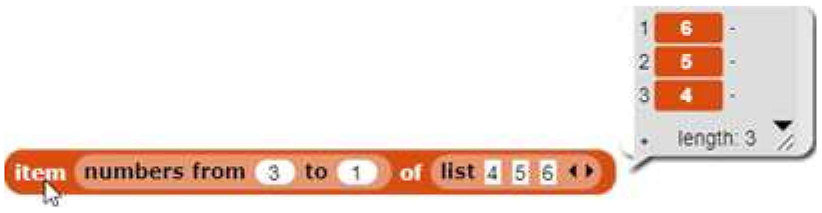
```
play sound (samples of sound One Two Three) at
Change Frequency by 1.25 Amount Hz
```

Exploration 9.5 Adjusting Frequency

Use the **Change Frequency** code block to increase and decrease the frequency of a digitized sound when it is played back. How does the frequency of the sound correspond to its perceived pitch? What happens to the length of the sound?

Topic 9.6 Reversing a Sound

If the items in a list are displayed from the end of the list to the beginning, the numbers are reversed. In the example below, the numbers “4, 5, and 6” are reversed.

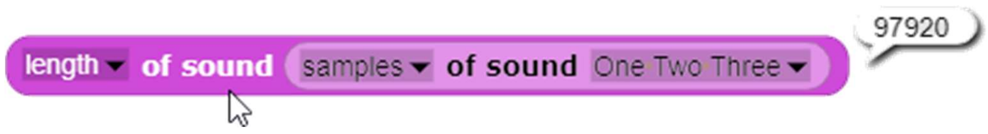


```
item (numbers from 3 to 1) of (list 4 5 6)
```

1	6
2	5
3	4

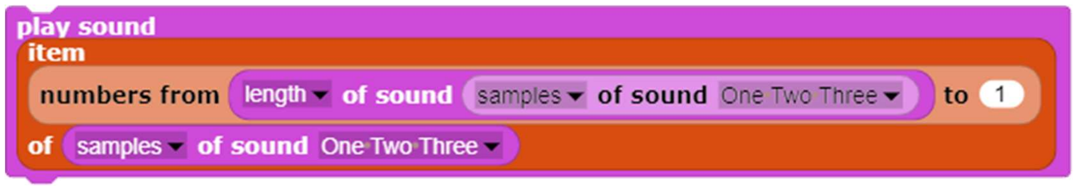
length: 3

The **Length of Sound** code block can be used to identify the total number of sound samples in a list. In this example, there are a total of 97,920 sound samples in the list.



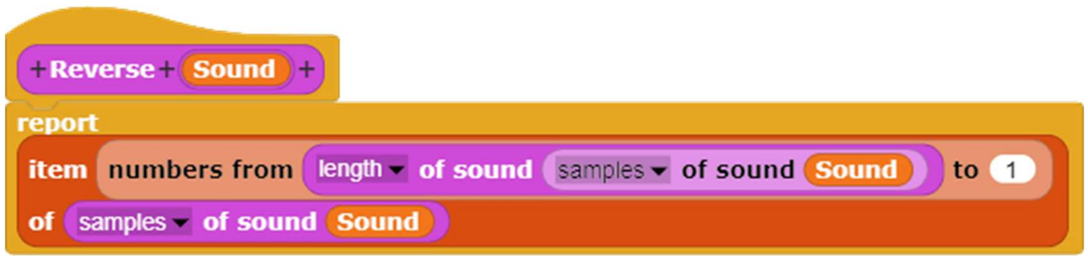
```
length of sound (samples of sound One Two Three)
97920
```

By playing the list of sound samples from the **Length of Sound** (97,920) to Item 1 in the list, the sound is played in reverse.



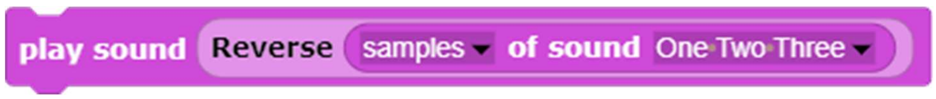
```
play sound
  item (numbers from (length of sound (samples of sound One Two Three)) to 1)
  of (samples of sound One Two Three)
```

This method can be used to create a custom code block, **Reverse**, to reverse the sound.



```
+Reverse+ (Sound) +
report
  item (numbers from (length of sound (samples of sound Sound)) to 1)
  of (samples of sound Sound)
```

The **Reverse** code block reverses a sound, starting with the end of the list of sound samples and continuing to the first item in the list of sound samples.



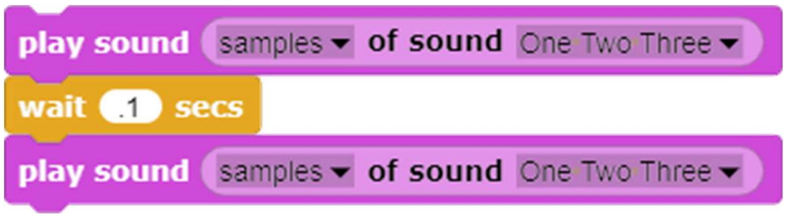
```
play sound Reverse (samples of sound One Two Three)
```

Exploration 9.6 Reversing a Sound

> Use the **Reverse** code block to play a list of digitized sound samples in reverse. Play several vowels in reverse to examine the effect that this has upon perception. Are vowels still recognizable when they are reversed? Then play several types of consonants in reverse. What effect does this have upon perception?

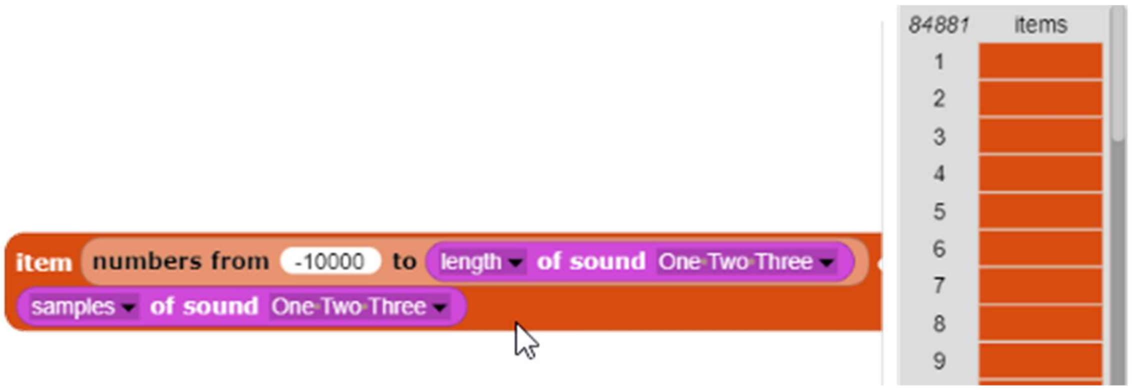
Topic 9.7 Echo

If a sound begins to play and then is played a second time with a slight delay, the result is perceived as an echo.



```
play sound (samples of sound One Two Three)
wait .1 secs
play sound (samples of sound One Two Three)
```

Musicians often use this method to create reverberation. A permanent list that combines one list of sound samples with a second, delayed list of samples can be created by placing a series of empty cells at the beginning of the second list. If the items from -10,000 to the **Length of the List** are displayed, the first 10,000 items in the list will be empty.

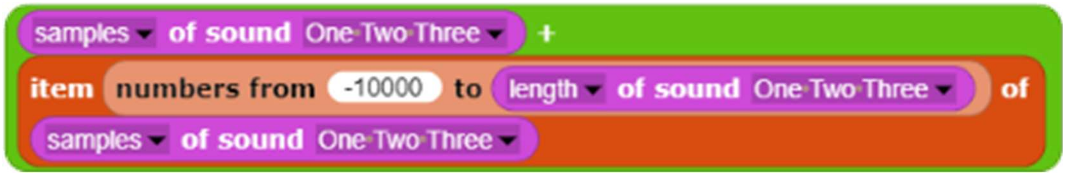


```
item (numbers from -10000 to (length of sound One Two Three))
samples of sound One Two Three
```

```
84881 items
1
2
3
4
5
6
7
8
9
```

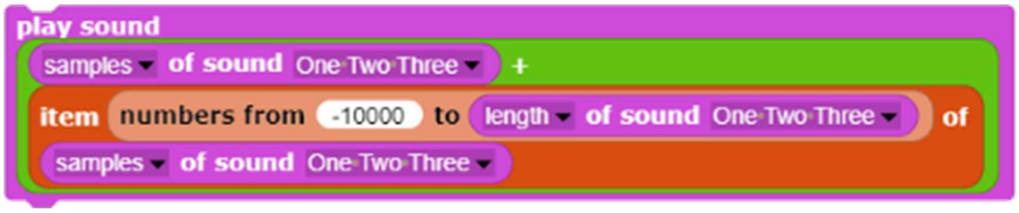
When the empty cells in the list of sound samples are played back, the sound will be silent during this portion of the list. This, in effect, delays the playback for that amount of time.

The list with the silence at the beginning can be combined with the original list by using the plus ("+") operator.



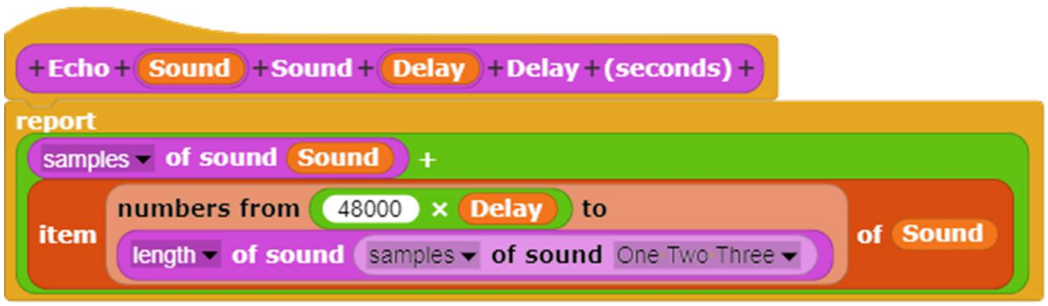
```
(samples of sound One Two Three) +
(item (numbers from -10000 to (length of sound One Two Three)) of
 (samples of sound One Two Three))
```

When the combined lists created in this manner are played back, there is a slight delay before the second list of sound samples begins, creating the effect of an echo. Increasing or decreasing the number of empty cells in the second list controls the length of the delay.



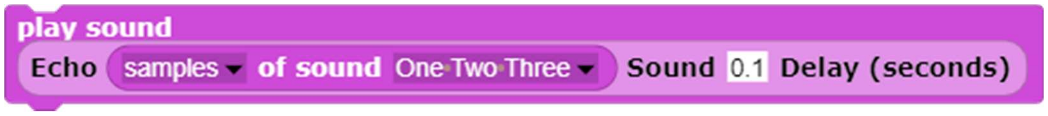
```
play sound
 (samples of sound One Two Three) +
 (item (numbers from -10000 to (length of sound One Two Three)) of
  (samples of sound One Two Three))
```

This technique can be used to create a custom **Echo** code block to create an echo with a specified delay that controls the timing of when the second sound begins playing.



```
+Echo+(Sound)+Sound+(Delay)+Delay+(seconds)+
report
  (samples ▾ of sound (Sound)) +
  item (numbers from (48000 × (Delay)) to
        (length ▾ of sound (samples ▾ of sound One Two Three ▾))) of (Sound)
```

The custom **Echo** code block can be used to specify the amount of echo that is incorporated into the playback.



```
play sound
  Echo (samples ▾ of sound One Two Three ▾) Sound 0.1 Delay (seconds)
```

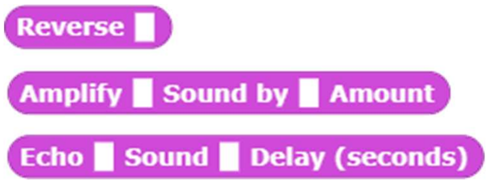
In a live performance setting, such as an auditorium, the amount of delay is related to the amount of time that it takes for the sound to travel to the walls and reflective surfaces of the auditorium and return. These reverberations affect the quality of musical performances. Listener's perceptions of recorded performances are also affected by these factors.

Exploration 9.7 Echo

Use the **Echo** code block to explore the effect on different lists of digitized sound samples. Attempt to create the perception of a large auditorium by controlling the amount of echo employed in the playback.

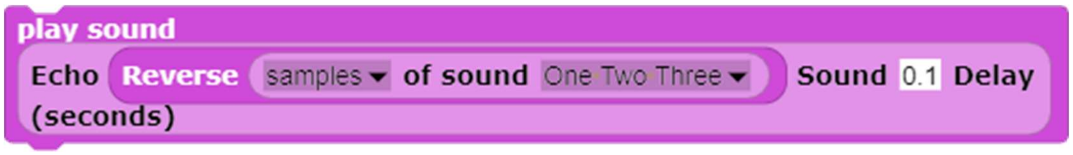
Topic 9.8 Combining Sound Effects

The **Reverse**, **Amplify**, and **Echo** code blocks can be combined with one another.



```
Reverse [ ]
Amplify [ ] Sound by [ ] Amount
Echo [ ] Sound [ ] Delay (seconds)
```

For example, a reversed sound with echo added can be played by combining those two code blocks.



```
play sound
  Echo (Reverse (samples ▾ of sound One Two Three ▾)) Sound 0.1 Delay
  (seconds)
```

Exploration 9.8 Combining Sound Effects.

Explore the effect of combining the **Reverse**, **Amplify**, and **Echo** code blocks in various combinations.

Summary

The acoustic tools developed for manipulating and digitizing sound samples can perform operations on the list of sound samples to increase or decrease the loudness, increase or decrease the pitch, play the sound backwards, add an echo, or isolate specific words or sounds within a longer recording.

The sound effects created through custom code blocks such as Amplify and Echo are illustrative of many other types of operations that can be performed to create other sound effects. For example, a high-pass filter removes some of the lower frequencies. Conversely, a low-pass filter removes some of the higher frequencies. These methods are the basis for many popular effects now used to create and record music. In various combinations, these methods are also the basis for creating and manipulating synthesized speech.

10. Lyrics and Vocal Tracks

Glen Bull, Jo Watts, and Rachel Gibson

Topic 10.1 Lyrics

The process of creating a melody, accompanying chords, and a drum track in TuneScope has been described in previous modules. Lyrics can be combined with music to create a song. For example, Judy Garland's classic theme song in the Wizard of Oz ("Somewhere Over the Rainbow") begins with the following lyrics.

Somewhere over the rainbow
Way up high
There's a land that I heard of
Once in a lullaby

The following table illustrates the way in which lyrics are combined with the melody, chords, and drum track.

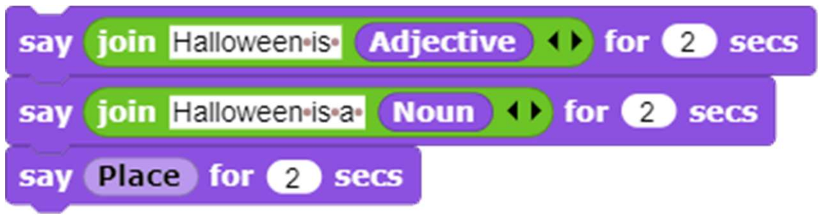
Somewhere over the Rainbow							
	Measure 1		**Measure 2**				
Lyrics	Some	where	O -	ver	the	Rain -	bow
Melody	F3	F4	E4	C4	D4	E4	F4
	Half	Half	Quarter	Eighth	Eighth	Quarter	Quarter
Drum	x	x	x			x	
Chords	F Major (F A C)		A Minor (A C E)				
	Whole Note		Whole Note				

Exploration 10.1

Enter the first two measures of a tune created in previous modules in a table similar to the one above. Then add a line of lyrics to accompany the tune.

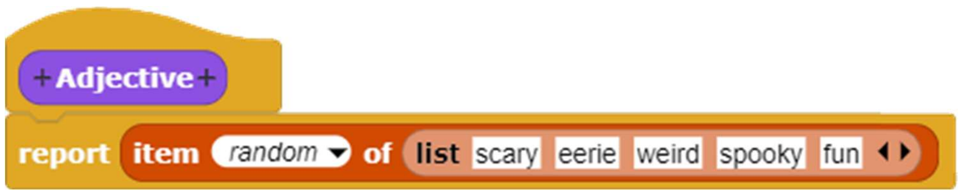
Topic 10.2 Computer Poetry

Creation of original lyrics to accompany a tune can be daunting for the novice. Creation of a poem that can be read with the accompaniment of music offers a useful entry point. Computer-generated poetry provides one strategy for stimulating the imagination. One strategy involves random selection of parts of speech such as nouns and adjectives to complete phrases.

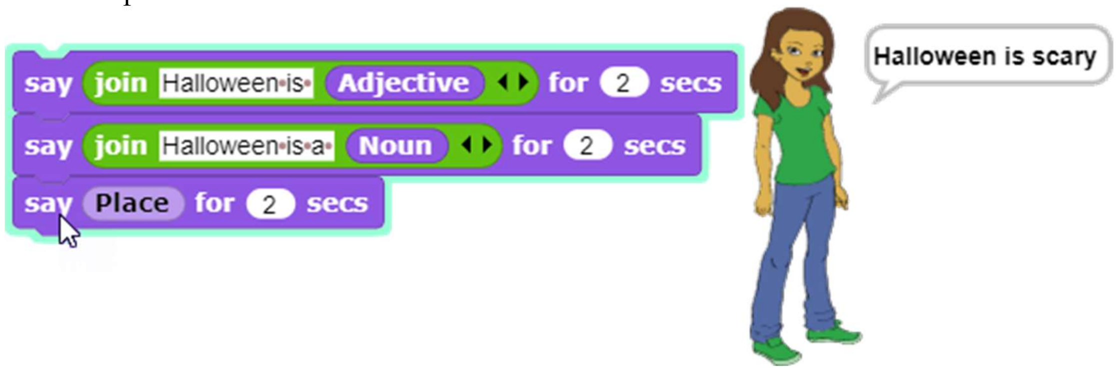


```
say join Halloween is  Adjective ◂▸ for 2 secs
say join Halloween is a  Noun ◂▸ for 2 secs
say Place for 2 secs
```

The **Adjective** reporter block, for example, generates a randomly selected adjective as output.


```
+Adjective+
report item (random) of list scary eerie weird spooky fun
```

The **Say** code block causes the currently selected sprite (in this example, Abbey) to say the words provided as inputs.


```
say join Halloween is  Adjective for 2 secs
say join Halloween is a  Noun for 2 secs
say Place for 2 secs
```

This method can generate metaphor poems such as:

Halloween is scary.
Halloween is a ghost
in the moonlight.

Exploration 10.2 Computer Poetry

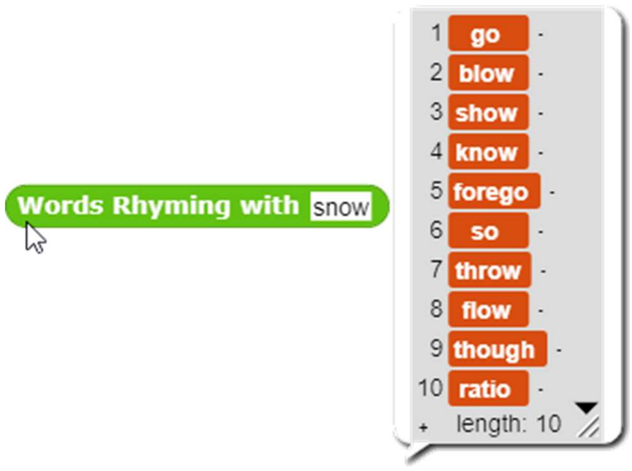
Create a metaphor poem similar to the one above. Ask friends to provide items for each category in the metaphor poem. Then run the program several times and pick the poem that you like the best.

Topic 10.3 Rhyming Dictionaries

A metaphor poem does not require words to rhyme. A rhyming dictionary can provide a useful way to identity words that rhyme. For example, the first line of a winter poem might be:

I love the winter snow

The **Words Rhyming with** ____ reporter block can generate a list of words that rhyme with "snow":



This might lead to a second line that rhymes with the first line:

I love the winter snow,
Hearing the wind blow

There are a number of online dictionaries that can be accessed to generate lists of rhyming words. An application interface is needed to access rhyming words from within Snap*!* or TuneScope. One of these services can be accessed through: "datamuse.com/api/". The first part of the web address for the api consists of the following element:

https://api.datamuse.com

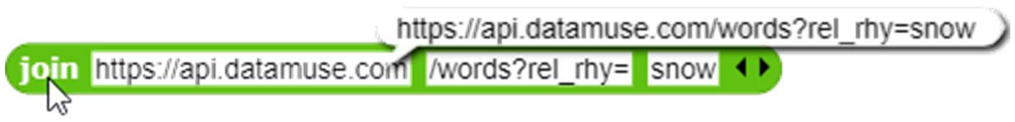
This element is combined with

/words?rel_rhy=

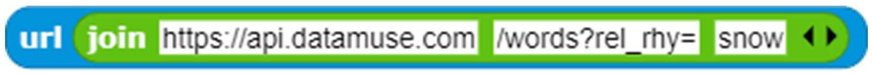
Followed by a word such as "snow". The final construct would consist of the following:

https://api.datamuse.com/words?rel_rhy=snow

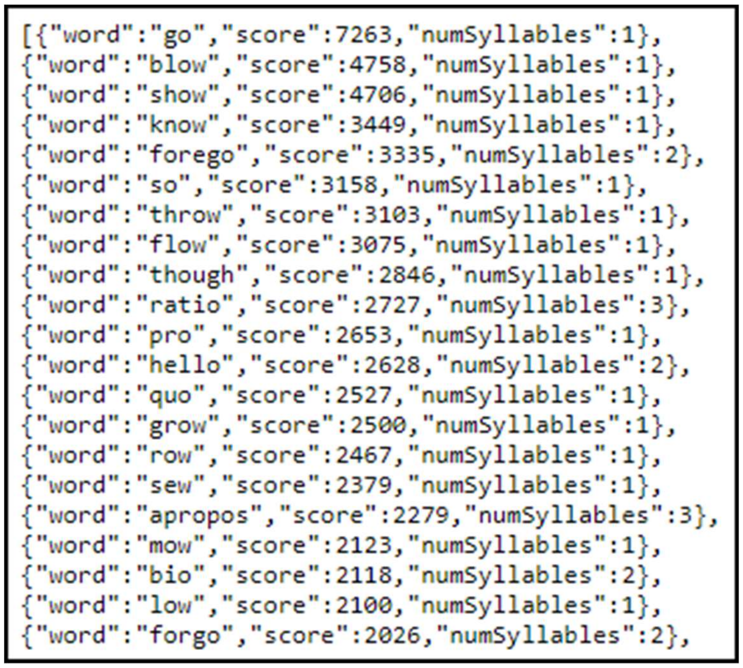
The **Join** block can join these elements together in Snap*!*:


https://api.datamuse.com/words?rel_rhy=snow
join https://api.datamuse.com /words?rel_rhy= snow

The web address constructed in this manner is then placed in the Snap*!* **URL** block:

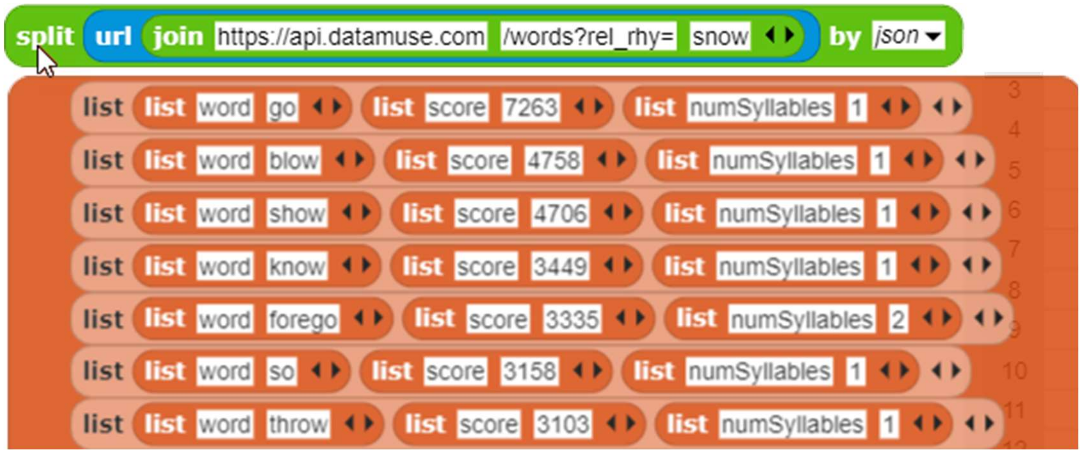


The output is returned in a format that looks like this:



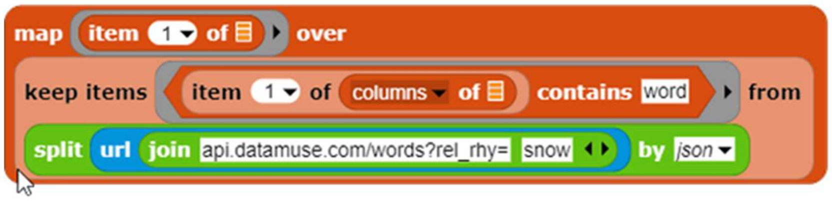
```
[{"word":"go","score":7263,"numSyllables":1},
{"word":"blow","score":4758,"numSyllables":1},
{"word":"show","score":4706,"numSyllables":1},
{"word":"know","score":3449,"numSyllables":1},
{"word":"forego","score":3335,"numSyllables":2},
{"word":"so","score":3158,"numSyllables":1},
{"word":"throw","score":3103,"numSyllables":1},
{"word":"flow","score":3075,"numSyllables":1},
{"word":"though","score":2846,"numSyllables":1},
{"word":"ratio","score":2727,"numSyllables":3},
{"word":"pro","score":2653,"numSyllables":1},
{"word":"hello","score":2628,"numSyllables":2},
{"word":"quo","score":2527,"numSyllables":1},
{"word":"grow","score":2500,"numSyllables":1},
{"word":"row","score":2467,"numSyllables":1},
{"word":"sew","score":2379,"numSyllables":1},
{"word":"apropos","score":2279,"numSyllables":3},
{"word":"mow","score":2123,"numSyllables":1},
{"word":"bio","score":2118,"numSyllables":2},
{"word":"low","score":2100,"numSyllables":1},
{"word":"forgo","score":2026,"numSyllables":2},
```

This output includes the number of syllables in each rhyming word and a score used to rank the rhyming words. The data that accompanies the rhyming words is in a format known as JavaScript Object Notation (JSON). The **Split** block can be used to split JSON data into lists of data.

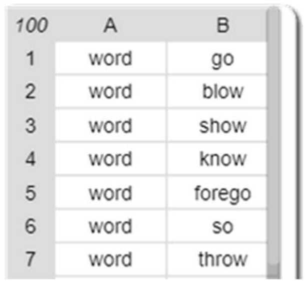


```
split url join https://api.datamuse.com /words?rel_rhy= snow by json
list list word go  list score 7263  list numSyllables 1
list list word blow  list score 4758  list numSyllables 1
list list word show  list score 4706  list numSyllables 1
list list word know  list score 3449  list numSyllables 1
list list word forego  list score 3335  list numSyllables 2
list list word so  list score 3158  list numSyllables 1
list list word throw  list score 3103  list numSyllables 1
```

The **Map** block can be used to apply an operation (i.e., map the operation) to all of the items in a list simultaneously. This method can be more efficient that using a **For** loop to apply an operation to each item in a list one at a time. In this example, the **Map** block has been used to keep Item 1 of the columns that contain the item “word”:

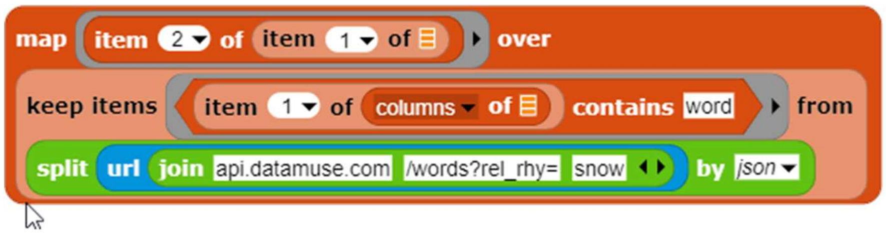


```
map (item (1) of [ ]) over
    keep items (item (1) of (columns of [ ]) contains [word]) from
        split (url (join [api.datamuse.com/words?rel_rhy=] [snow])) by [json]
```

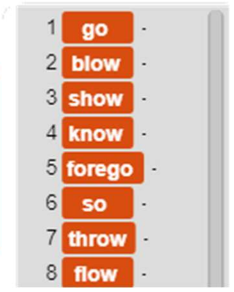


100	A	B
1	word	go
2	word	blow
3	word	show
4	word	know
5	word	forego
6	word	so
7	word	throw

Once this result has been obtained, Item 2 of each remaining list (i.e., the rhyming word) can be selected:

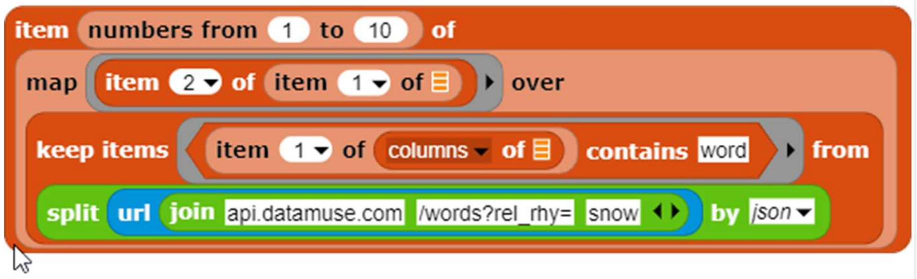


```
map (item (2) of (item (1) of [ ])) over
    keep items (item (1) of (columns of [ ]) contains [word]) from
        split (url (join [api.datamuse.com] [/words?rel_rhy=] [snow])) by [json]
```

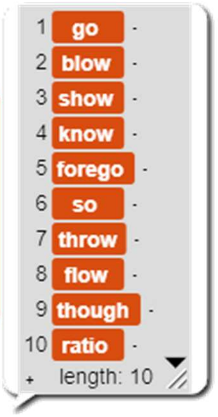


1	go
2	blow
3	show
4	know
5	forego
6	so
7	throw
8	flow

If only the ten most relevant results are wanted, the **Item** block can be combined with the **Numbers 1 to 10** block to select the first ten items in the list:



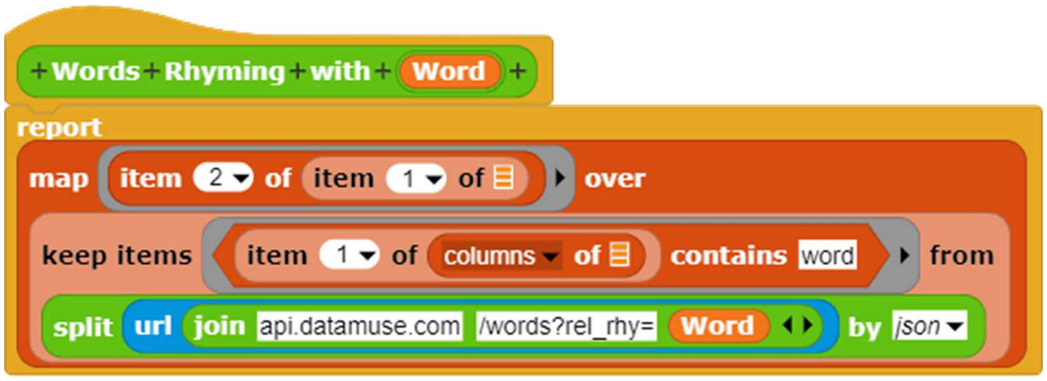
```
item (numbers from (1) to (10)) of
    map (item (2) of (item (1) of [ ])) over
        keep items (item (1) of (columns of [ ]) contains [word]) from
            split (url (join [api.datamuse.com] [/words?rel_rhy=] [snow])) by [json]
```



1	go
2	blow
3	show
4	know
5	forego
6	so
7	throw
8	flow
9	though
10	ratio

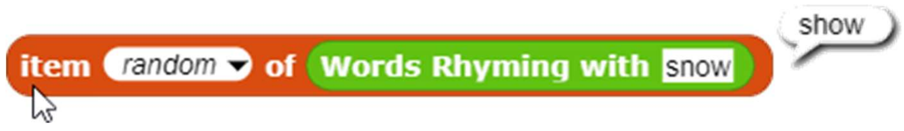
length: 10

By this means, the Data Muse api can be used to create a **Words Rhyming with ___ code block**.



```
+Words+Rhyming+with+ Word +
report
  map ( item 2 of item 1 of ⌸ ) over
    keep items ( item 1 of columns of ⌸ contains word ) from
      split url join api.datamuse.com /words?rel_rhy= Word by json
```

The **Item Random** block can then be combined with the **Words Rhyming with** _ block to generate lines of poetry in the same manner as the Metaphor Poems described in the preceding section.

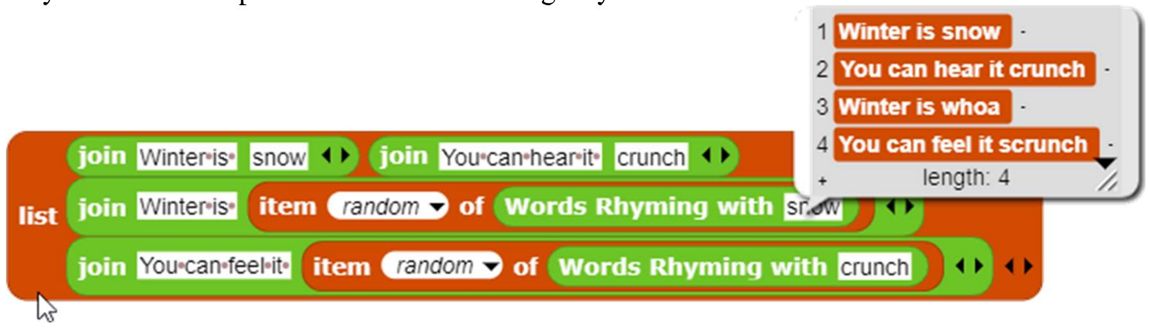


```
item random of Words Rhyming with snow
```

The rhyming dictionary can be used to generate poems that rhyme. For example, the following is an example of an A-B A-B rhyming pattern.

Winter is snow.
You can hear it crunch.
Winter is _____.
You can feel it _____.

This rhyme could be implemented in the following way:



```
list
  join Winter is snow
  join You can hear it crunch
  join Winter is item random of Words Rhyming with snow
  join You can feel it item random of Words Rhyming with crunch
```

```
1 Winter is snow
2 You can hear it crunch
3 Winter is whoa
4 You can feel it scrunch
length: 4
```

These results can be edited to create a poem that is the most satisfactory by the user.

Winter is snow.
You can hear it crunch.
Winter is whoa!
You can feel it scrunch.

In this manner, the computer can be used to stimulate creation of poems.

Exploration 10.3 Rhyming Dictionaries

Use the rhyming dictionary to generate a rhyming poem. After running the procedure several times, pick the version of the of the poem that you like the best.

Topic 10.4 Other Options for Generation of Words

The Data Muse API can be used to access several other options such as *words that follow another word* or *words that are associated with another word*. The table below lists options that can be accessed by appending them to the URL for the Data Muse API: "api.datamuse.com"

Examples of Data Muse Options	
Option	*Suffix*
Words that rhyme with	**/words?ml=**
Words with a meaning similar to	/words?rel_rhy=
Words that follow	/words?lc=
Words associated with	/words?reltrg=

There are many other options that can be accessed in a similar way to generate poetry.

Exploration 10.4 Other Options for Generation of Words

Incorporate some of the other options such as "Words that Follow" or "Words Associated With" to generate poetry.

Topic 10.5 Creating Lyrics for an Existing Melody

In the United States, the copyright for a musical composer expires 70 years after the death of the composer. Pachelbel died in 1706, so this musical composition has been in the public domain since 1776.

This means that anyone has been free to re-purpose this work and incorporate it into their own tunes and compositions. Consequently, elements of Pachelbel's Canon have been incorporated into the following songs and many others.

Hook, Blues Traveler
Go West, Pet Shop Boys
Cryin', Aerosmith
Tunnel of Love, Dire Straits
No Woman No Cry, Bob Marley and the Wailers
Streets of London, Ralph McTell
Memories, Maroon Five
Let It Be, The Beatles

The form also lends itself to parody. The following are lyrics created at the beginning of allergy season to accompany Pachelbel's Canon:

Pachelbel's Allergy

Flowers blooming
Pollen looming

Spring in C'ville
Sneezin' season

Buy your Kleenex
In profusion
Claritin is part of mealtime

The following lyrics in a more serious vein were created by Mary Peng, a University of Virginia student.

Faith, Hope, Life

You are my faith, my hope, my life
You gave me the chance to breathe
The fresh air of the world
And walk around
With a heart of wonder
Eyes for splendor

Time goes by so fast
My love for you
Grows deeper
Every passing day

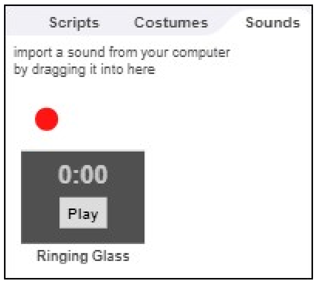
Exploration 10.5 Lyrics

Create lyrics to accompany Pachelbel's Canon or a similar classic work.

Topic 10.6 Recording Vocals

Once lyrics have been created, the next step is to record a vocal track to accompany the other music tracks. A sound recording program such as Audacity (https://www.audacityteam.org/) can be used to record vocals. The vocals can then be exported as an ".mp3" or ".wav" file and imported into TuneScope.

Although the audio editing capabilities of TuneScope are not as advanced as the editing features of a dedicated audio recording program such as Audacity, it is possible to edit audio waveforms directly in TuneScope. (The process of recording a sound was described in an earlier chapter.)



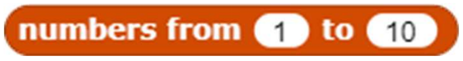
This process converts the back-and-forth movements of a computer's microphone into an electrical signal. The amplitude of the varying electrical signal is repeatedly measured and converted into a series of digital numbers. This process is known as analog-to-digital (A/D) conversion. The process indirectly measures minute movements of the microphone and converts that movement into a series of digital numbers.

Exploration 10.6

Use the record feature to record and play back a short utterance.

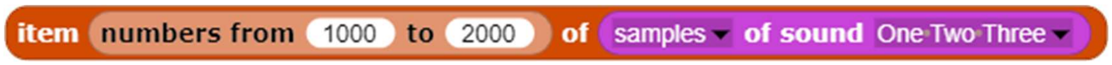
Topic 10.7 Editing Sound Segments

Previously the **Numbers** block was used to select the ten most relevant rhyming words.



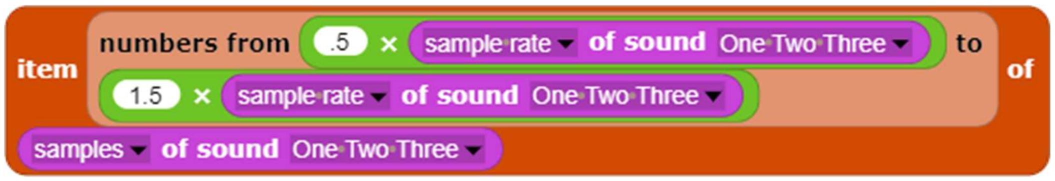
```
numbers from 1 to 10
```

The **Numbers** reporter block can also be used to select a specific range of sound samples.



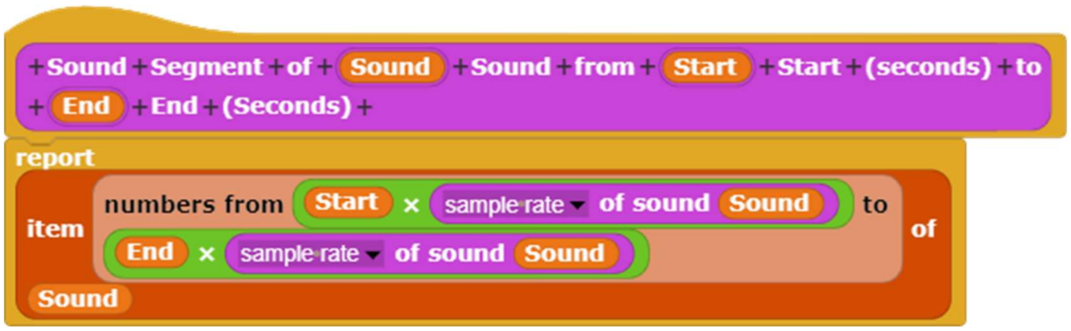
```
item (numbers from 1000 to 2000) of (samples of sound One Two Three)
```

The starting point and end point can be expressed in seconds if these numbers are multiplied by the sample rate of the sound.



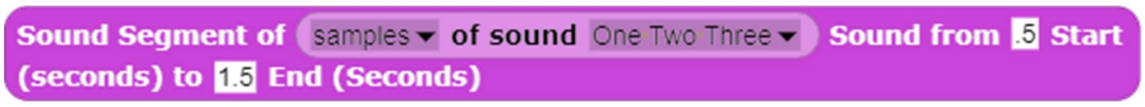
```
item (numbers from (.5 × (sample rate of sound One Two Three)) to (1.5 × (sample rate of sound One Two Three))) of
    (samples of sound One Two Three)
```

This sound sample selection procedure can be encapsulated as a **Sound Segment** block.



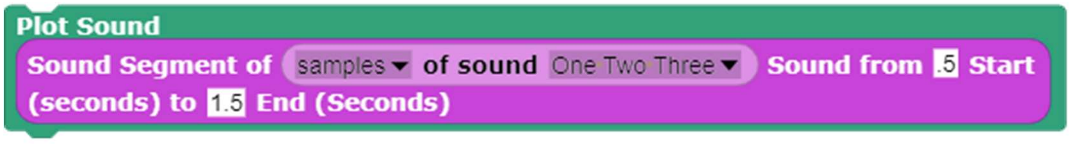
```
+Sound+Segment+of+(Sound)+Sound+from+(Start)+Start+(seconds)+to
+(End)+End+(Seconds)+
report
    item (numbers from ((Start) × (sample rate of sound (Sound))) to ((End) × (sample rate of sound (Sound)))) of
        (Sound)
```

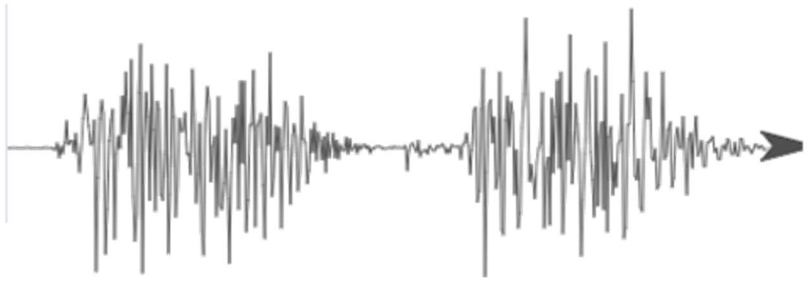
The **Sound Segment** block can, for example, be used to select the sound samples that fall between a half-second and one and one-half seconds.



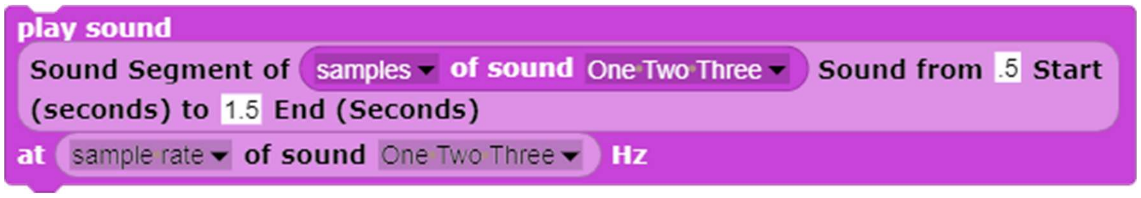
```
Sound Segment of (samples of sound One Two Three) Sound from .5 Start
(seconds) to 1.5 End (Seconds)
```

The sound samples selected in this manner can be graphed.


Plot Sound
Sound Segment of samples of sound One Two Three Sound from .5 Start (seconds) to 1.5 End (Seconds)



They also can be used to play specific segments of the recorded sound.


play sound
Sound Segment of samples of sound One Two Three Sound from .5 Start (seconds) to 1.5 End (Seconds)
at sample rate of sound One Two Three Hz

In this manner, specific segments of a sound recorded in TuneScope can be identified and played. Sound segments, in turn, can be combined to form a vocal track to accompany other music tracks created in TuneScope.

Exploration 10.7 Editing Vocal Tracks

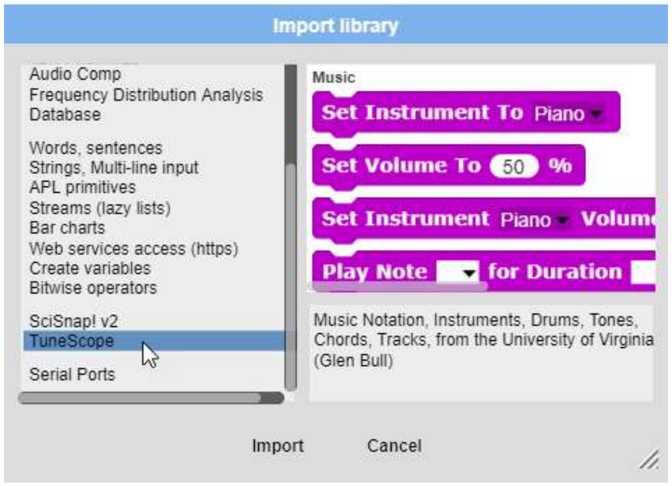
Create and record lyrics to accompany one verse of a tune. Then use the sound editing capabilities described above to trim the beginning of the recording so that it starts at the correct time when played synchronously with the accompanying music.

TuneScope Reference

The TuneScope music library is a new feature incorporated into Snap*!* 8.0. TuneScope music blocks extend the music capabilities of Snap!. This project was undertaken through a collaboration among the Department of Music, the Department of Computer Science, and the School of Education at the University of Virginia. This reference is a guide to the TuneScope music library in Snap*!*

I. *TuneScope* Setup

The *TuneScope* library can be imported into an empty Snap*!* project by selecting the *File* icon and selecting *Libraries*. The *Import Library* menu will appear, allowing the *TuneScope* library to be imported into a new project.



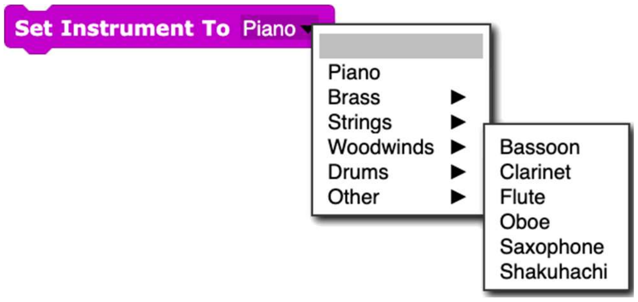
Music Notation, Instruments, Drums, Tones, Chords, Tracks, from the University of Virginia (Glen Bull)

Initialize *TuneScope*

The **Initialize TuneScope** block loads the sound samples used by *TuneScope* . This should occur at the start of any *TuneScope* session. When a *Play Note* block (**Play Note, Play Note and Wait**, etc.) is run, it checks to be sure that the required sound samples have been loaded. If the sound samples have not yet been downloaded, the **Initialize TuneScope** block is run. Loading the sound samples can take approximately five to ten seconds.

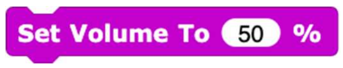
Set Instrument

The **Set Instrument** block selects a global musical instrument for playing notes.



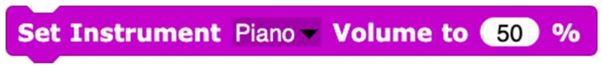
Set Volume

The **Set Volume** code block sets the overall sound level of all instruments.



Set Instrument Volume

The **Set Volume of Instrument** code block adjusts the volume level of a specific instrument.



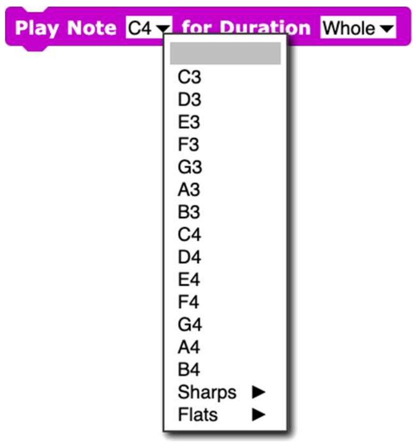
II. Musical Notes

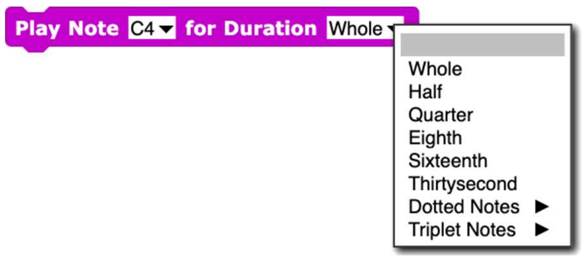
Play Note

The **Play Note** code block plays a musical note. Several **Play Note** blocks can be combined to create a musical chord.

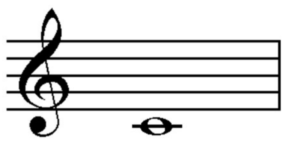
Notes consist of a note name (e.g., C, C#, etc.) combined with an octave number (e.g., 3, 4, etc.). The term *scientific pitch notation* describes this method of defining musical notes.

A note duration (whole, half, quarter, etc.) can be selected from the drop-down menu.



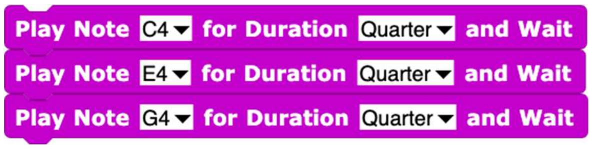


The equivalent representation on a musical staff is shown below.

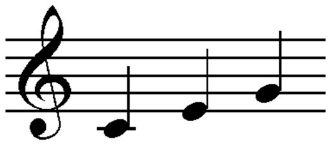


Play Note and Wait

The **Play Note and Wait** code block waits until one note is completed before beginning the next note. Several **Play Note and Wait** code blocks can be combined to play a series of notes. A note duration (whole, half, quarter, etc.) can be selected from the drop-down menu.

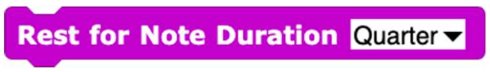


```
Play Note C4 for Duration Quarter and Wait
Play Note E4 for Duration Quarter and Wait
Play Note G4 for Duration Quarter and Wait
```

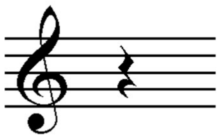


Rest

The **Rest** code block inserts a delay of a specified note duration in a sequence of notes.

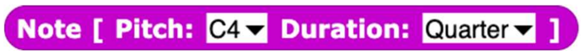


```
Rest for Note Duration Quarter
```

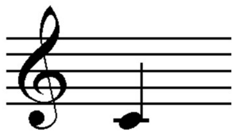


Note

The **Note** code block reports a list that consists of a specified note name and duration. These can be selected using the drop-down menu or entered directly.



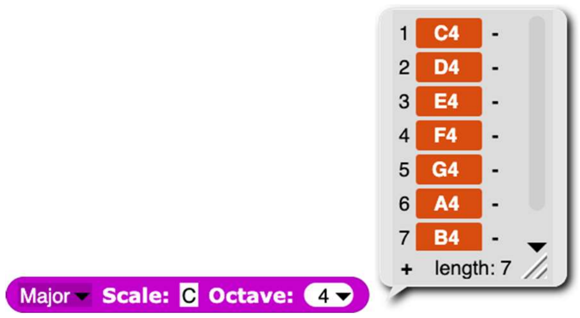
```
Note [ Pitch: C4 Duration: Quarter ]
```



III. Musical Scales

Scale

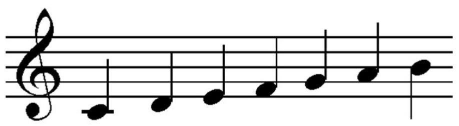
The **Scale** block reports the notes in the specified scale using the provided note and octave number. The leading drop-down menu allows the user to select the scale type (major, minor, or chromatic).



```
Major Scale: C Octave: 4
```

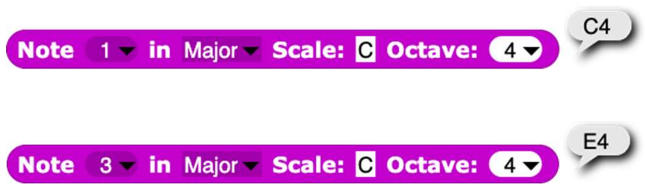
1	C4
2	D4
3	E4
4	F4
5	G4
6	A4
7	B4

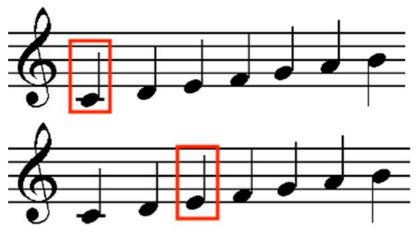
+ length: 7



Note Position in Scale

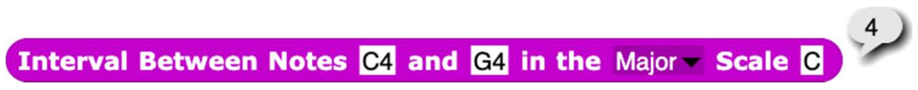
The **Note Position in Scale** block reports the note in the selected position of a specified scale. A drop-down menu allows the user to select major or minor scale. In the example below, the first note in a C Major scale starting in the fourth octave is C4, and the third note is E4.


Note 1 in Major Scale: C Octave: 4 → C4
Note 3 in Major Scale: C Octave: 4 → E4



Interval Between Notes

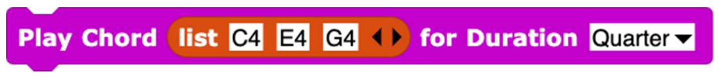
The **Interval Between Notes** block reports the distance (in steps) between two notes in a scale. A drop-down menu allows the user to select major or minor scales. In the example below, within the C major scale, there are four steps in between C4 and G4.


Interval Between Notes C4 and G4 in the Major Scale C → 4

IV. Musical Chords

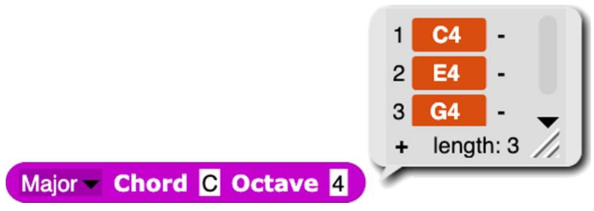
Play Chord

The **Play Chord** block plays all notes in a list simultaneously. The duration of the chord may be selected from a drop-down menu or entered directly.


Play Chord list C4 E4 G4 for Duration Quarter

Major/Minor Chord

The **Major/Minor Chord** block reports the notes in a major or minor chord as a list. This block uses the specified note and octave as the lowest note of the chord. A drop-down menu allows the user to select major or minor chords. Both the note and the octave should be entered directly.

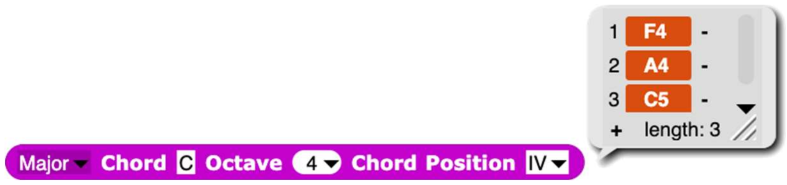

Major Chord C Octave 4 →

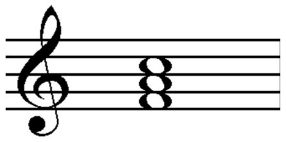
1	C4
2	E4
3	G4

length: 3

Major/Minor Chord Position

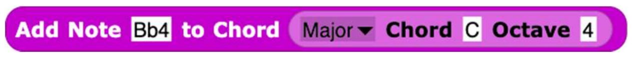
The **Major/Minor Chord Position** block extends the **Major/Minor Chord** block by allowing the user to find the notes in chords along the selected scale. Roman numerals differentiate chord positions from octave numbers (e.g., the Roman numeral "I" reports the notes of the first chord in the sequence, the Roman numeral "iii" reports the notes in the third chord, and so on). Upper case Roman numerals represent major chords within the scale. Lower case Roman numerals represent minor chords within the scale.

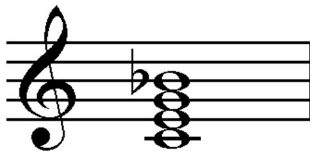




Add Note to Chord

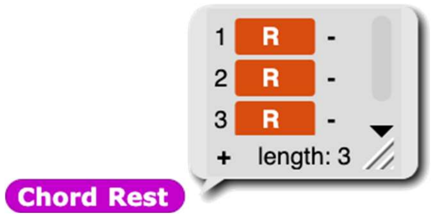
The **Add Note to Chord** code block reports a chord with an additional note appended. In the example below, the **Major/Minor Chord** code block is used as the input, and a B-flat note is appended to the resulting chord.





Chord Rest

The **Chord Rest** code block reports a rest ("R") for use with chords.

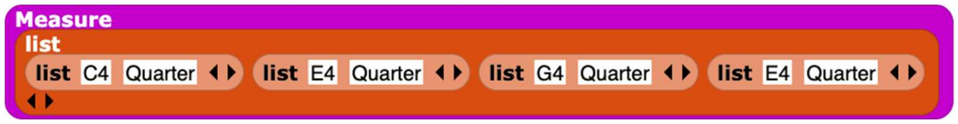


V. Musical Tracks

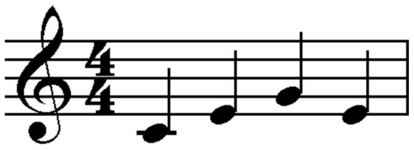
Musical tracks can be used to combine sequences of notes to create a song. Notes can be combined into measures. Measures, in turn, are combined to form musical tracks.

Measure

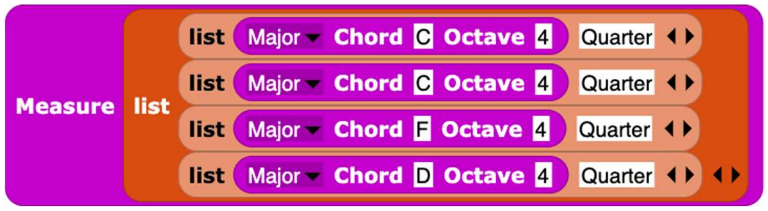
The **Measure** block is used to group sequences of notes into measures. Each note in the measure is paired with a note duration. The pairs are entered into the **Measure** block using a list.
Note: A musical measure consists of the notes within one bar of music.



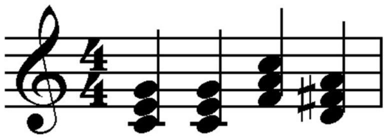
The notated equivalent of the block's result is shown below.



The **Measure** block also groups chords and chord durations into measures.

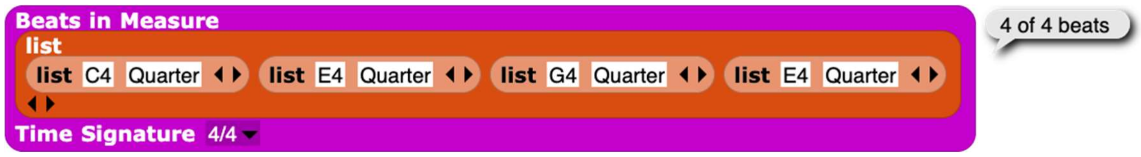

Measure list
list Major Chord C Octave 4 Quarter
list Major Chord C Octave 4 Quarter
list Major Chord F Octave 4 Quarter
list Major Chord D Octave 4 Quarter

The notated equivalent of the block's result is shown below.

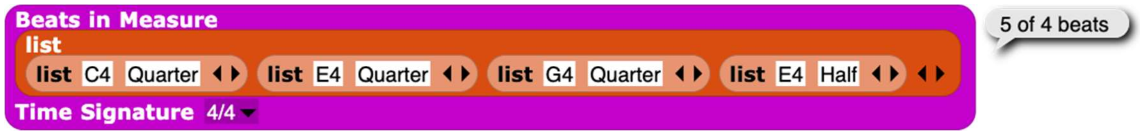


Beats in Measure

The **Beats in Measure** block can be used to determine if the combined durations of notes or chords within a measure are correct. The combined duration of notes within a measure must equal the number of beats specified by the time signature. For example, in 4/4 time, there can be four quarter notes within a measure. The **Beats in Measure** block reports the difference between the number of actual beats in a measure versus the required number of beats.

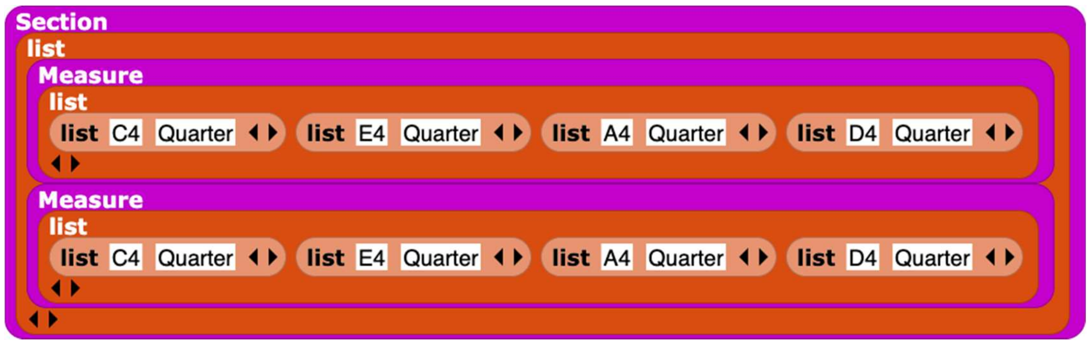

```
Beats in Measure
list
  list C4 Quarter  list E4 Quarter  list G4 Quarter  list E4 Quarter
Time Signature 4/4
```
4 of 4 beats

If the durations are not correct, the **Beats in Measure** will report the discrepancy.

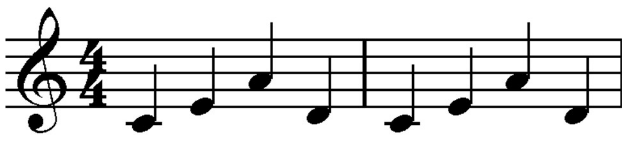

```
Beats in Measure
list
  list C4 Quarter  list E4 Quarter  list G4 Quarter  list E4 Half
Time Signature 4/4
```
5 of 4 beats

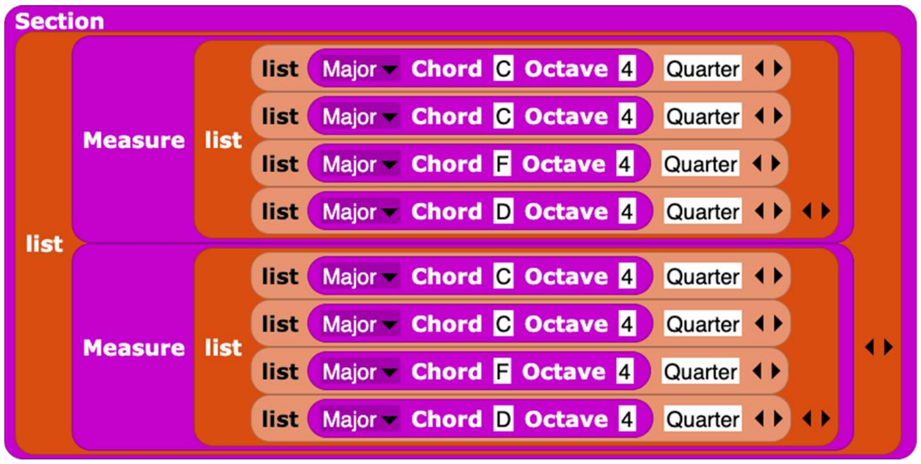
Section

The **Section** code block groups measures together. This can be helpful for separating groups of measures that repeat in various places throughout a song, like verses and choruses. This block works with both notes and chords. To add measures to a section, put the measures into a list.


```
Section
list
  Measure
  list
    list C4 Quarter  list E4 Quarter  list A4 Quarter  list D4 Quarter
  Measure
  list
    list C4 Quarter  list E4 Quarter  list A4 Quarter  list D4 Quarter
```

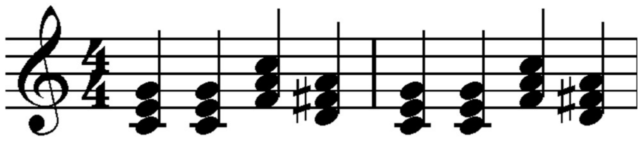
The notated equivalent of the block's result is shown below.





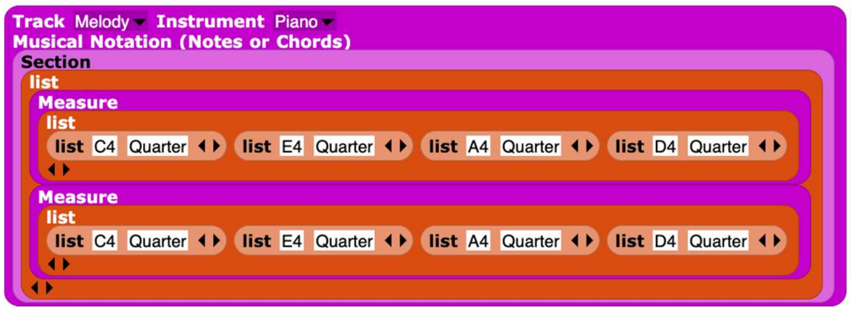
```
Section
  list
    Measure list
      list Major Chord C Octave 4 Quarter
      list Major Chord C Octave 4 Quarter
      list Major Chord F Octave 4 Quarter
      list Major Chord D Octave 4 Quarter
    Measure list
      list Major Chord C Octave 4 Quarter
      list Major Chord C Octave 4 Quarter
      list Major Chord F Octave 4 Quarter
      list Major Chord D Octave 4 Quarter
```

The notated equivalent of the block's result is shown below.

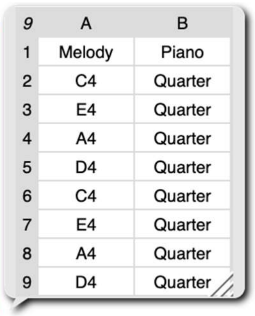


Track

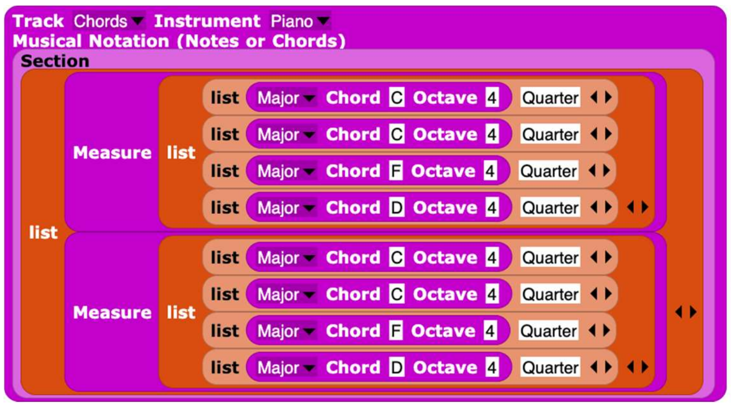
The **Track** code block specifies the type of track to be played and assigns it an instrument. Both the track type and instrument are set using drop-down menus. The musical notation input accepts notes or chords paired with corresponding durations. For notes and chords, the musical notation should be entered as shown below.



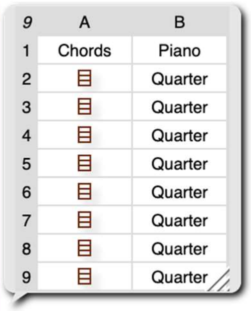
```
Track Melody Instrument Piano
Musical Notation (Notes or Chords)
Section
  list
    Measure
      list
        list C4 Quarter   list E4 Quarter   list A4 Quarter   list D4 Quarter
    Measure
      list
        list C4 Quarter   list E4 Quarter   list A4 Quarter   list D4 Quarter
```



9	A	B
1	Melody	Piano
2	C4	Quarter
3	E4	Quarter
4	A4	Quarter
5	D4	Quarter
6	C4	Quarter
7	E4	Quarter
8	A4	Quarter
9	D4	Quarter



```
Track Chords Instrument Piano
Musical Notation (Notes or Chords)
Section
  list
    Measure list
      list Major Chord C Octave 4 Quarter
      list Major Chord C Octave 4 Quarter
      list Major Chord F Octave 4 Quarter
      list Major Chord D Octave 4 Quarter
    Measure list
      list Major Chord C Octave 4 Quarter
      list Major Chord C Octave 4 Quarter
      list Major Chord F Octave 4 Quarter
      list Major Chord D Octave 4 Quarter
```

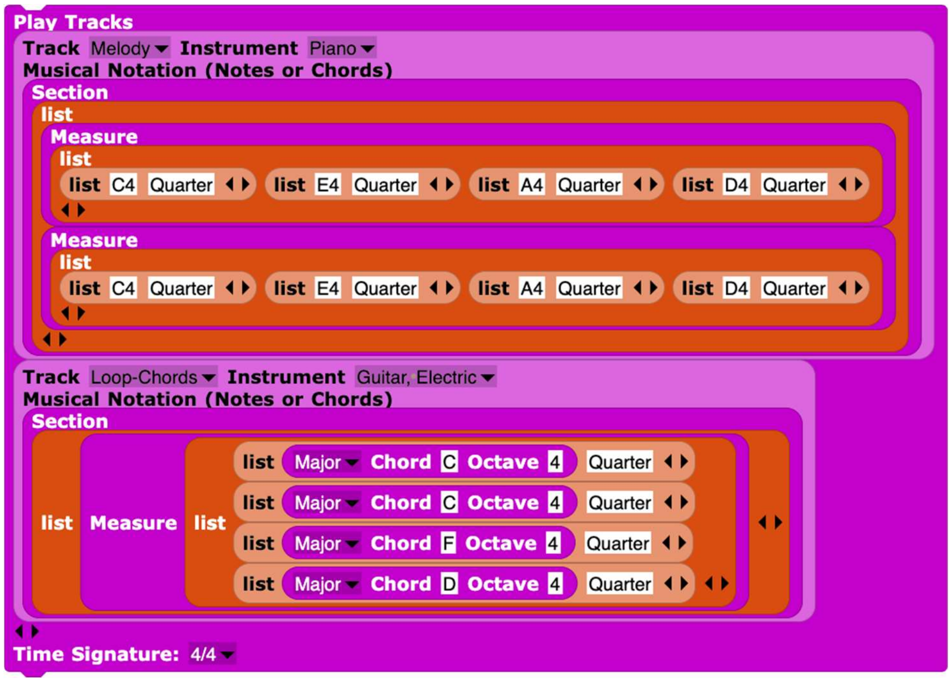


9	A	B
1	Chords	Piano
2	☰	Quarter
3	☰	Quarter
4	☰	Quarter
5	☰	Quarter
6	☰	Quarter
7	☰	Quarter
8	☰	Quarter
9	☰	Quarter

The **Track** block also allows for the creation of loops. A loop is a section of notes or chords that will repeat for the duration of the song. Both types of loops may be selected from the track drop-down menu. *For a loop to play, it must have an accompanying track that is not a loop.*

Play Tracks

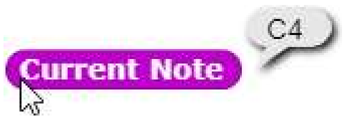
The **Play Tracks** block is used to play music tracks. In most cases several tracks with musical instruments are played in parallel. For example, a piano track and a guitar track are played in parallel in the example below.



```
Play Tracks
  Track Melody Instrument Piano
  Musical Notation (Notes or Chords)
    Section
      list
        Measure
          list
            list C4 Quarter  list E4 Quarter  list A4 Quarter  list D4 Quarter
        Measure
          list
            list C4 Quarter  list E4 Quarter  list A4 Quarter  list D4 Quarter
  Track Loop-Chords Instrument Guitar, Electric
  Musical Notation (Notes or Chords)
    Section
      list Measure list
        list Major Chord C Octave 4 Quarter
        list Major Chord C Octave 4 Quarter
        list Major Chord F Octave 4 Quarter
        list Major Chord D Octave 4 Quarter
Time Signature: 4/4
```

Current Note

When **Play Tracks** is used to play a music track, the **Current Note** block reports the note that is currently playing in the first track. This capability can be used to synchronize a music event with other events in Snap*!*

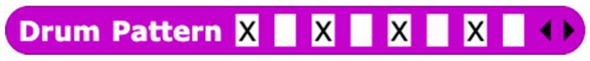


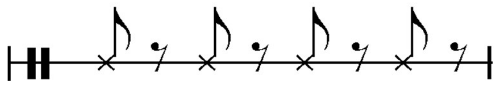
At present, the **Current Note** block can only be used to report notes in a melody track. Therefore, the first track must be a melody track rather than chord track or a drum track in order for **Current Note** to yield accurate results.

VI. Drum Tracks

Drum Pattern

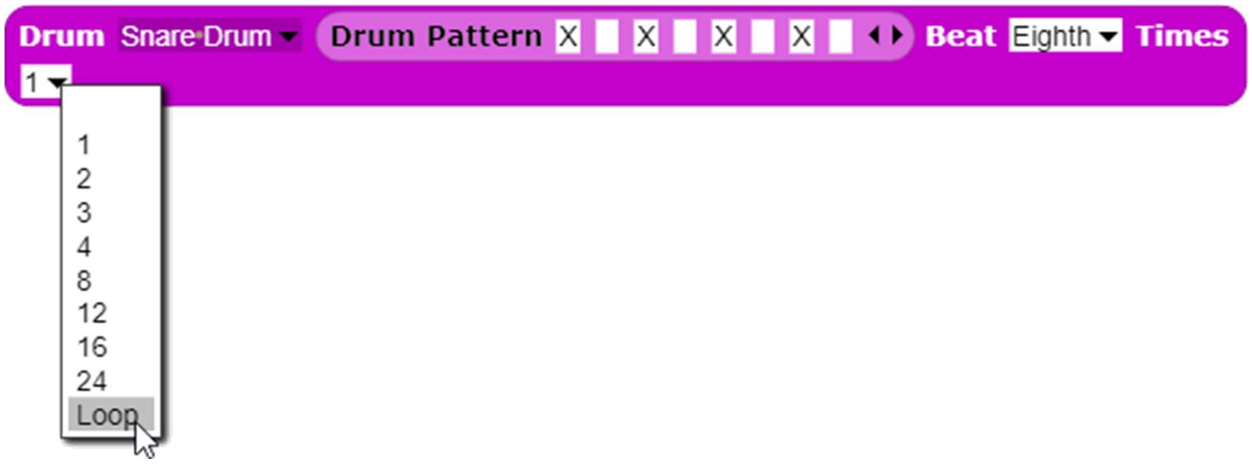
The **Drum Pattern** block reports a series of beats that tell the drum when to play. Beats can be added or subtracted from the track using the arrows on the right of the block. A list block can also be used to report a drum pattern.





Drum

The **Drum** block reports a drum pattern, assigns a drum to it, and sets the duration value of each item in the pattern. Different drums can be selected from the drop-down menu. Beat durations can be selected from the drop-down menu or entered manually. The number of times that the pattern repeats can also be selected from a drop-down menu.

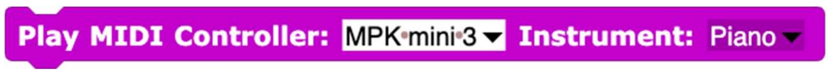


VII. MIDI Tools

MIDI is an acronym for *Musical Instrument Digital Interface*. The MIDI tools can be used to play notes from a MIDI keyboard.

Play MIDI

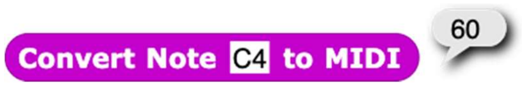
When the **Play MIDI** block is run, the notes played on an attached MIDI keyboard are played through the computer speakers.



When the **Play MIDI** block is enabled, the **Current Note** block (described under Section V "Musical Tracks") reports the current note played on the MIDI keyboard at the time that the key on the keyboard is pressed. When a second key is pressed, the note reported by **Current Note** changes to the new note played.

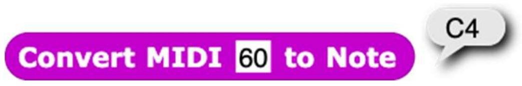
Note to MIDI

The **Note to MIDI** block reports the MIDI number that corresponds to the note name entered.



MIDI to Note

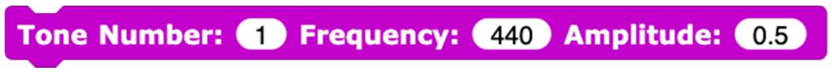
The **MIDI to Note** block reports the note name that corresponds to the MIDI number entered.



VIII. Acoustic Tools

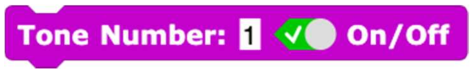
Tone

The **Tone** block allows the user to assign a number to a tone and enter a frequency and amplitude.



Tone Number On/Off

The **Tone Number On/Off** block allows the user to turn the specified tone on or off. This block will play a sine wave.



Tone Off

The **Tone Off** block turns off all the tones currently playing.