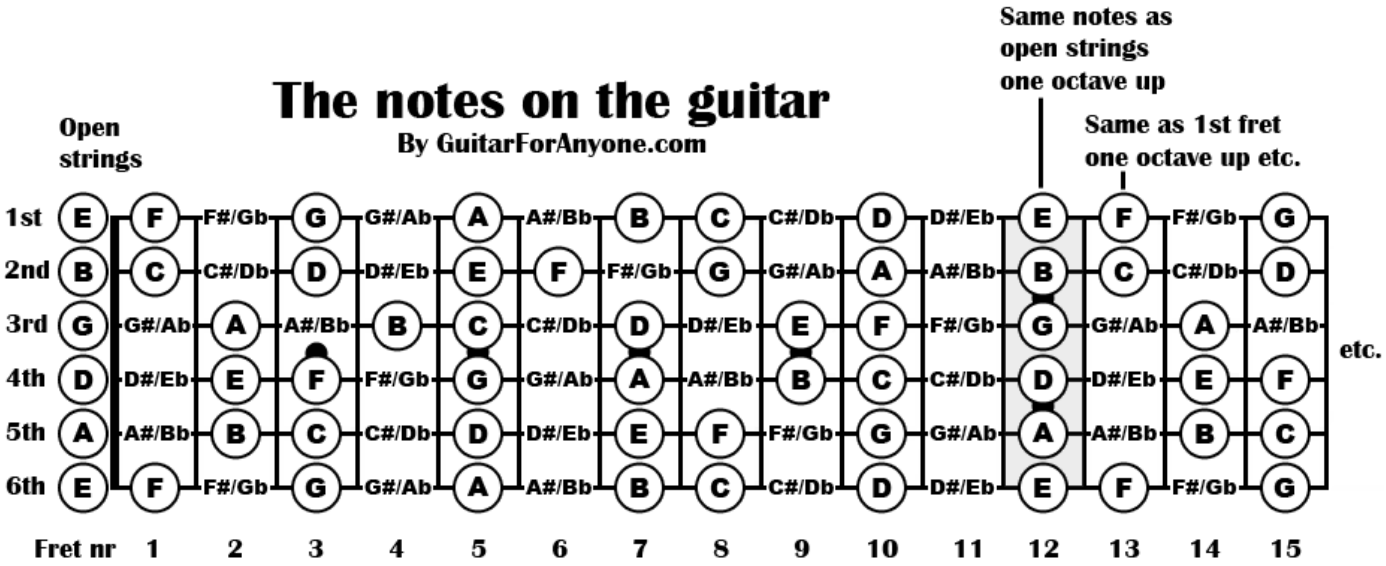
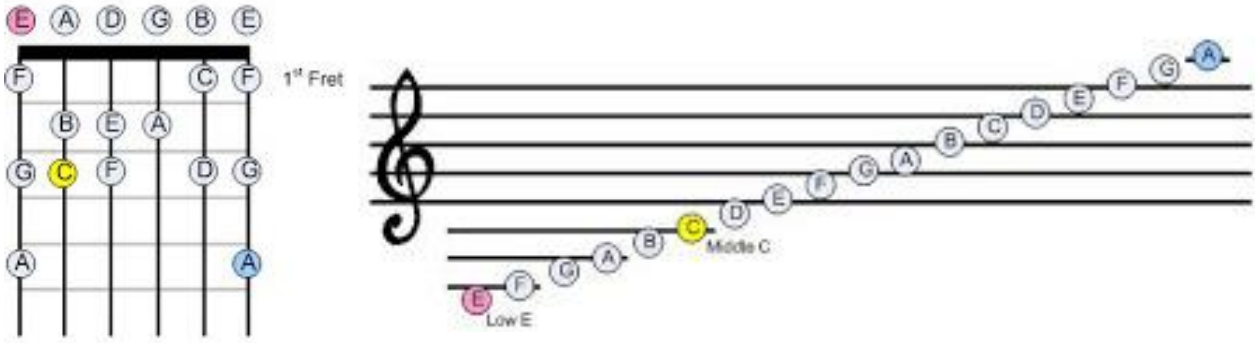
Project 4 - Sound Player

Due Date: Thursday 5/3/18 at 11:59pm

For this project, you are to write a python program that will take a string containing "musical notes" as the input and create a .wav file that plays those notes. We will be using notes as played on a guitar as shown below (we could have used the notes on a piano, but then we would have 88 notes instead of 30 notes).



We won't be using all of these notes, just the basic 18 notes for the guitar plus the flat/sharp notes around these 18 (for a total of 30 notes). These basic 18 notes shown with the notes marked on sheet music are as follows. (Note that one A note is shown twice so the graphic shows 19 notes instead of 18 notes.)



For this program, we will need to be able to translate from the notes to the frequencies of sound each note creates when played on a guitar. Please note that the frequencies for guitar notes and piano notes differ by one octave.

Frequency	Note	String	Fret	Comments
82.4	E	6th string	open	
87.3	F	6th string	1st	

92.5	F#	6th string	2nd	
98.0	G	6th string	3rd	
103.8	G#	6th string	4th	
110.0	A	5th string	open	The above images shows the fingering for this note twice (once as the open fret on the 5th string and twice as the 5th fret on the 6th string).
116.5	A#	5th string	1st	
123.5	B	5th string	2nd	
130.8	C	5th string	3rd	Note position of "middle C" on a piano
138.6	C#	5th string	4th	
146.8	D	4th string	open	
155.6	D#	4th string	1st	
164.8	E	4th string	2nd	
174.6	F	4th string	3rd	
185.0	F#	4th string	4th	
196.0	G	3rd string	open	
207.6	G#	3rd string	1st	
220.0	A	3rd string	2nd	
233.1	A#	3rd string	3rd	
246.9	B	2nd string	open	
261.6	C	2nd string	1st	Frequency of "middle C" on a piano
277.2	C#	2nd string	2nd	
293.6	D	2nd string	3rd	
311.1	D#	2nd string	4th	
329.6	E	1st string	open	
349.2	F	1st string	1st	
370.0	F#	1st string	2nd	
392.0	G	1st string	3rd	
415.3	G#	1st string	4th	
440.0	A	1st string	5th	

Since the input to the program will be a string of notes. Each note needs to be specified by a unique character. Each basic note on the guitar occurs 2 or 3 times, so we need to be able to distinguish between the three different A notes: the A note at 110 hz, the A note at 220 hz, and the A note at 440 hz.

- We will use the digits of 7,8,9 to represent the 3 lowest notes of E, F, G that are played on the 6th string of the guitar.
- We will use the lower case letters a through g to represent the notes of A on the 5th string to G on the 3rd string.
- We will use the upper case letters A through G to represent the notes of A on the 3rd string to G on the 1st string.
- We will use the upper case letter of H to represent the note of A on the 1st string.

The above representation could allow us to extend to higher notes by using additional upper case letters, but this project will NOT require that.

The one thing that this project will require is the representation of the sharp notes. This will be needed to mathematically translate from the notes to proper frequencies. The sharp notes will use the shifted keys of the digits.

So to relist the notes with frequencies and their corresponding character representation:

Frequency	Note	Character Representation
82.4	E	7
87.3	F	8
92.5	F#	!
98.0	G	9
103.8	G#	@
110.0	A	a
116.5	A#	#
123.5	B	b
130.8	C	c
138.6	C#	$
146.8	D	d
155.6	D#	%
164.8	E	e
174.6	F	f
185.0	F#	^ - the "up arrow" character at "Shift-6"
196.0	G	g
207.6	G#	&
220.0	A	A
233.1	A#	*
246.9	B	B
261.6	C	C
277.2	C#	(
293.6	D	D
311.1	D#	)
329.6	E	E
349.2	F	F
370.0	F#	_
392.0	G	G
415.3	G#	+
440.0	A	H

To list all of the characters in order in a single Python String, we would use that following code:

```
noteRepresentation = "78!9@a#bc$d%ef^g&A*BC(D)EF_G+H"
```

Once we have the above string, we can use it and a mathematical formula to calculate the frequency of each note. (Note: the exponentiation operation symbol in Python is **).

```
freq = 440.0 * 2.0 ** ((x - y)/12.0)
where:
   x = position in the String of the desired note
   y = position in the String of the A note with frequency of 440 hz
```

With the use of the Python String Library method find( ), we can easily find the position of our desired note and the position of the A note with frequency of 440 hz. For example, if we wanted to calculate the frequency of the G note played on the 3rd string. The "note character" is 'g'. The "note character" of the A note with frequency 440 hz is 'H'. We then have the following Python code:

```
noteRepresentation = "78!9@a#bc$d%ef^g&A*BC(D)EF_G+H"
x = noteRepresentation.find('g')   # x becomes 15
y = noteRepresentation.find('H')   # y becomes 29
exponent = (x - y)/12.0
freq = 440.0 *  (2.0  ** exponent)     # freq becomes 196.0 (if rounded)
```

One more item that we will need to deal with is the length of the note being played and the amount of time between each note. To keep this as simple as possible, we will assume 2 beat per second. Thus each "beat" will last for 1/2 second. So:

- A quarter note will take up 1/2 second in the resulting .wav file.

However, since we have to separate each note, we will have a "half beat" of silence between notes. This restriction will only allow us to play notes of a quarter note or longer. To distinguish between two quarter notes and one half note, our input will need to specify what is being played during every 1/4 of a second.

- A quarter note will be 1/4 of a second of sound followed by 1/4 of a second of silence.
- A half note will be 3/4 of a second of sound followed by 1/4 of a second of silence.
- A whole note will be 7/4 of a second of sound followed by 1/4 of a second of silence.

The input will then need a character to represent silence. The default character will be the period/dot character. However, if we encounter any "non-note" character, we are to treat it as "silence".

In order to keep the input as simple as possible, we will leave the length of each note up to how the user gives the input. Thus to have a quarter note of the 110.0 hz a note that will be "1/4 of a second of sound followed by 1/4 of a second of silence", the input would be the following string:

- a. Where the a character in the string specifies the 1/4 of a second of sound and the . character in the string specifies the 1/4 of a second of silence.

The following string will be 4 quarter notes separated by an 1/4 of a second of silence (with an addition 1/4 of a second of silence at the end).

- a.a.a.a.

While the following string will be 2 half notes separated by an 1/4 of a second of silence (with an addtional 1/4 of a second of silence at the end).

- aaa.aaa.

While the following string will be 1 whole note (with an addtional 1/4 of a second of silence at the end).

- aaaaaaa.

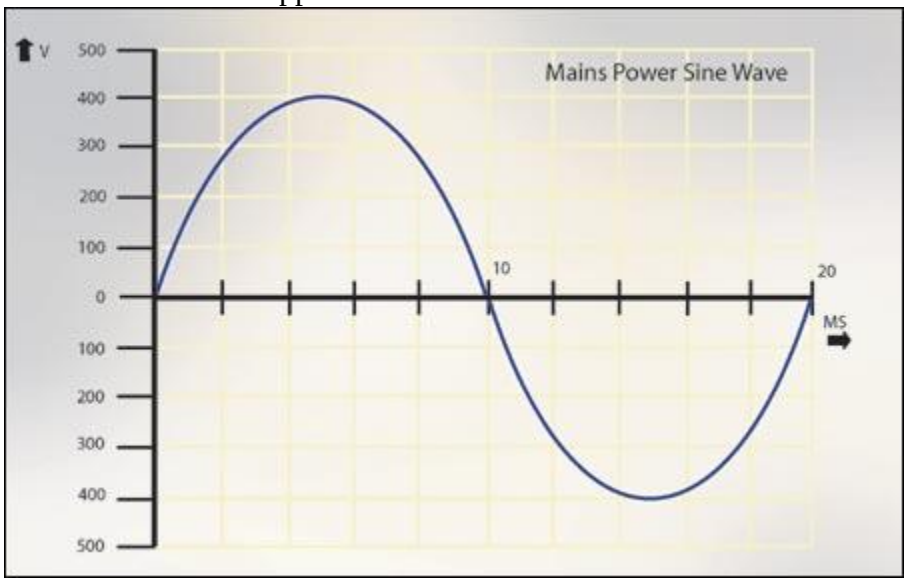
Note that each character in the program input will specify 1/4 of a second of the created sound file. So a string of 8 characters will always create a sound file that is two seconds long. The notes played will be determined by the actual characters listed in the string.

Making Sine Waves

The next thing we need to do is to create the proper sine wave for each frequency. Our text book does discuss how to do this on page 267 from the Guzdial/Ericson text. Note that this code creates a sine wave that lasts for N seconds, where N is an integer number. However, we only want each sine wave to last for 1/4 of a second, so we will need to modify that for our program.

The following is a discussion of the creation of a sine wave for those that do not have access to the code and discussion given in the text book. The following is not exactly the same as the text's discussion. Thus if you don't follow what the book is stating, perhaps this discussion works better for you (or perhaps the other way around). Note that using the code properly is the important item. Just as using the formula for luminance properly was more important that understanding how/why those precise values were chosen.

Note that sine wave appears as:



First we need to determine how many sound samples are needed for one cycle of the wave at frequency f. If we want to create a sound at a certain frequency (like 440 Hz), we have to fit the sine wave into 1/440 of a second. If we have a sample file with 22050 sample per second, that means we have about put the since wave in about 50 samples. To calculate the number of samples needed we can take the total number of samples per second (i.e. the sampling rate of a .wav file) and divide that by the frequency.

```
samplesPerCycle = samplingRate / freq
```

Second we want to fill in that number of samples with a sine wave. To do this we need to figure out at what point in the wave each sample is located. This can be done by taking the Index number for each sample and dividing this by the number of samples per cycle.

```
wavePoint = sampleIndex / samplesPerCycle
```

For this assignment, each frequency will be maintained for 1/4 of a second. So to determine the length of the sound and the range of the sampleIndex values, we divide the sampleRate by 4.

Now, we need to find the sine value of the wavePoint. Since the sin( ) function of the Python library takes input in radians, we need to first multiple the wavePoint value by 2 * pi before calling the sine() function. Fortunately, the Python library also gives us the value if pi.

```
rawValue = sin (wavePoint * 2.0 * pi)
```

The last thing that needs to be done is to determine the amplitude of our sine wave. The result from sin() give us a value from -1 to 1. We need to multiple this value by some constant to give our sine wave some volume. Multiplying by a value of 5000 seems to work well.

User Input

Our input to the program will be a string that we will get from the user. It is up to the user to give us a properly formatted input. Example of this input are:

- The beginning of Jingle Bells:
  E.E.EEE.E.E.EEE.E.G.C.D.EEEEEEE.
- The beginning of Twinkle Twinkle Little Star (Or Mozart's Twelve Variations on "Ah vous dirai-je, Maman"):
  g.g.D.D.E.E.DDD.C.C.B.B.A.A.ggg.
- London Bridges:
  D.E.D.C.B.C.DDD.A.B.CCC.B.C.DDD.D.E.D.C.B.C.DDD.AAA.DDD.B.ggg.
- Old MacDonald :
  C.C.C.g.A.A.g...E.E.D.D.C.....g.C.C.C.g.A.A.g...E.E.D.D.C.
- Go Team Go (traditional school fight song):
  ggg.BBB.C.(((.D.ggg.BBB.C.(((.D.ggg.BBB.C.(((.D.G.G.G...g.g.g.

To get the input from the user, prompt the user for an input string using the requestString( ) function from the JES library.

While testing your program, you can simply copy and paste the above strings into this prompt. You job is to take the input and create the .wav file containing those notes.

Output of the Program

After the .wav file has been created, show the sound object using the explore() function.

Also prompt the user for a file and save the .wav file to a name given by the user.

Program Writing Style

Your program must be written using good programming style which includes:

- Use of multiple functions
- Good variable names
- In-line commenting
- Proper indentation of program statements
- Use of blank lines to separate blocks of code.
- Header block commenting for the program and each function written

  Your header block comment for the program must include the following:

  - Your Name,
  - Net-ID,
  - Course Name,
  - Assignment Name and
  - Day and time of your CS 111 lab section (i.e. Monday at 9:00)
  - A short description of the assignment.

  Header block comments for each function must include the following:

  - A description of the purpose of the function
  - A listing of the name, type and purpose of every parameter
  - A description of the return value and its type

Project Collaboration

You are allowed to receive help on this project from other students who are also taking CS 111. Each student must still complete and submit his/her own project. You will be required to include a **Collaboration Statement** somewhere on your project if you receive help. This statement can simply be something like the following:

For this project, I received help from the following member of CS 111.

- Devang Jariwala
- Shun Liang
- Hongwei Zhu

This statement should list each helping student's name in a comment in the "header comment" of your Python file that includes the main() function.

Submission of your Project

Submit your code via Blackboard. Use the assignment link for Project 4 to submit your python code.