F# Project - Sound Synthesis

The aim of this project is to create a sound synthesizer that can be used to create programmable music. This project was inspired by Sonic Pi (https://sonic-pi.net/) and other live coding music packages. The ultimate aim of this project is to be able to play music from code that looks something like the code given below.

```
let note length note octave =
    Creation.makeNote Creation.sine length note octave
    |> Envelope.apply 0.9 0.5 0.1 0.4
    |> Filter.flatten 0.4
    |> Filter.echo 0.4

let halfNote = note 0.5
let quaterNote = note 0.25
let eightNote = note 0.125


let tune =
    seq { yield! quaterNote Note.B 4
          yield! quaterNote Note.B 4
          yield! eightNote Note.B 4
          yield! eightNote Note.B 4
          yield! eightNote Note.C 4
          yield! eightNote Note.D 4
          yield! halfNote Note.D 4
          yield! quaterNote Note.D 4
          yield! eightNote Note.D 4
          yield! eightNote Note.C 4
          yield! eightNote Note.C 4
          yield! eightNote Note.B 4 }
    |> Filter.applyFunction (Fliter.lfo 0.6)


// play the tune
let player = Player.Play(tune, Repeat = true)
```

The Deliverables

The project has been broken into 4 parts each with its own delivery date. At each for the delivery dates a version of the library should be delivered with a public API described in the relevant part. Alongside this API the following items are required:

- Unit tests that will automatically show that the logic implemented by the API works correctly
- Documentation explaining how to use the API
- Scripts demonstrate how to use each of the provided functions and make it easy to play and visualizes the resulting souls
- A script that combines a number of different functions implemented to create a pleasing tune

Part 1 - Create and Play a Basic Note

Delivery date: 14 January 2022

The first step in this project is to create and play a simple note. To complete this module, the sound synthesiser's public API should have the following functionalities:

- An oscillator function or functions that can generate the four basic waveforms at variable audible frequencies. The four basic wave forms are sine, square, triangle and sawtooth.
- A function to save waveform to disk, so it can be played back through a standard audio application
- A function to read a section of an audio file from disk
- A function to play the waveform directly without saving it to disk

Hints and tips:

A good way of checking that wave forms are correct is to use a chart package such as XPlot (https://fslab.org/XPlot/) to check they look correct.

It's suggested to save the waveform in the WAV format, because it's a format that's simple and should not require a third part library.

To play back the sound it will be necessary to call the operating system's audio functions directly or use a library that does this.

Part 2 - Basic Filters, Envelopes, and Chords

Delivery date: 21 January 2022

A filter is a function that takes a waveform and modifies it someway to produce a modified sound. An envelope is a kind of filter used to vary a sound's amplitude over time, helping to give it an interesting and more realistic sound.

A chord is combining two or more waveforms to create a new sound. A chord is similar to a filter in that it can be seen as one waveform applying a transformation to another.

A filters module should be created to provide the following filters:

- Modify the wave's amplitude by a fixed amount
- Cut off the wave at specific amplitude to given the "overdriven" often used in rock songs
- Add echo to the sound
- A flange effect filter, for a description of this effect see wikipedia: https://en.wikipedia.org/wiki/Flanging
- A reverb effect filter, wikipedia has a description of reverberation: https://en.wikipedia.org/wiki/Reverberation

A four stage envelope should be provided with the following parameters: attack, decay, sustain and release. Wikipedia provides the standard definitions for these parameters: https://en.wikipedia.org/wiki/Envelope_(music)#ADSR. A basic version of the envelope should have a linear gradient between each stage, while more advanced envelope implementations will provide a way to specify nonlinear gradients.

A combine function should also be provided to combine two or more waves into a chord.

Part 3 - Frequency Analysis and Advanced filters

Delivery date: 28 January 2022

A spectroscope analyses the frequency of a sound sample. This can then be used to create a low pass or high pass filter. A low pass filter removes low frequency sounds while a high pass filter removes high frequency sounds. A module with parameterized low and high pass filters should be provided.

An LFO, low frequency oscillator, is a type of filter used to change a sound over time. The waveform of the LFO is not applied directly to the target waveform, but the waveform of the LFO is used to modify the amplitude or the frequency of the target sound. A module should be provided with both an AM and FM LFO.

Part 4 - MP3 Compression

Delivery date: 11 February 2022

The WAV format is popular because it's easy to use, but the resulting files can be very large. This is a disadvantage for both storage and network transfer. The MP3 format became popular because it can store the same sounds in a file that's about a factor of 10 smaller than the equivalent WAV. The format meant that songs could be much more easily shared over networks and played back on small portable devices, so the format became a catalyst behind the digital music revolution.

A module to load, save, playback MP3 files should be provided. It should also be possible to use all the filters and other sound tools with the MP3s once they are loaded.

Just creating a wrapper around an existing MP3 library should be avoided in favour of implementing the MP3 compression algorithm.