What was built:

We developed a functional sound/music generating, editing, and visualization software. This program allows one to explore the realm of computational audio known as "bytebeat". Each function is built from a tree of mathematical and bitwise expression nodes. Audio is generated at 8-bit 11,025 Hz. Features include:

- Index of 2^{32} randomly generated functions to explore
- Waveform and spectrogram visualization
- Real time function editor with 10 binary operators, constants, and variables
- Save functions to and load from MAD file format
- Save audio to WAV file (scaled to 16-bit 44.1kHz)
- Playback speed adjuster

How it worked:

The project was built upon a GUI supported by PyQtGraph and PyQt5 with audio being supplied through an audio stream using PyAudio. Visuals such as the waveform and spectrogram were components used from the PyQtGraph library that were added as `items` to the overall layout of the interface.

The creation of audio came in the form of mathematical expressions which are randomly generated using classes that form an expression, (+, -, *, /, ^...) etc. An editor utilizing tree traversal algorithms was used to maneuver the fully developed function and change values/variables and operators by replacing nodes.

What doesn't work/features that could have been added:

- The function-saving functionality is file-based bare bones. A more fully featured preset browser would be useful.
- Finding compatibility between python libraries/versions and the team's different operating systems was problematic at first. It took some testing and a good bit of research before a solution worked for everyone. PyAudio still is not installable on Windows through pip with python 3.7
- Keyboard input handling is very rudimentary, and present the program is stateless. That is, every key can be pressed at any time and does at most one function. Ideally, different classes/windows could take over keyboard input and have different functionality at different times. For example: Press 'Enter' to begin editing a function, use arrow keys to navigate a menu, then press 'Enter' again to confirm. It was difficult to find Qt examples of this more complicated keyboard behavior.
- At first, we were expecting more low-level access to drawing functions via a library like pygame (allowing more creative visuals), but due to library compatibility we ended up using PyQtGraph to handle graphics and keyboard input, which guided the visual development of the application to be have a more structured look.

What lessons were learned:

- In an experimental project like this, where the effectiveness of the function-generating code was at first unknown, an iterative, agile-based approach to adding and testing functionality worked well.
- Prototyping basic functionality helped in testing the feasibility of some ideas.
- We were all new to GUI libraries, and now that we have built the project once with Qt, it could probably be redesigned with a better architecture in mind.
- Github experience using branching techniques and code reviews/check-ins.
- Collaboration in the form of pair programming is a great way to review and simultaneously debug code.
- Prototype annotation aids in documentation and helps in defining the scope of the methods.