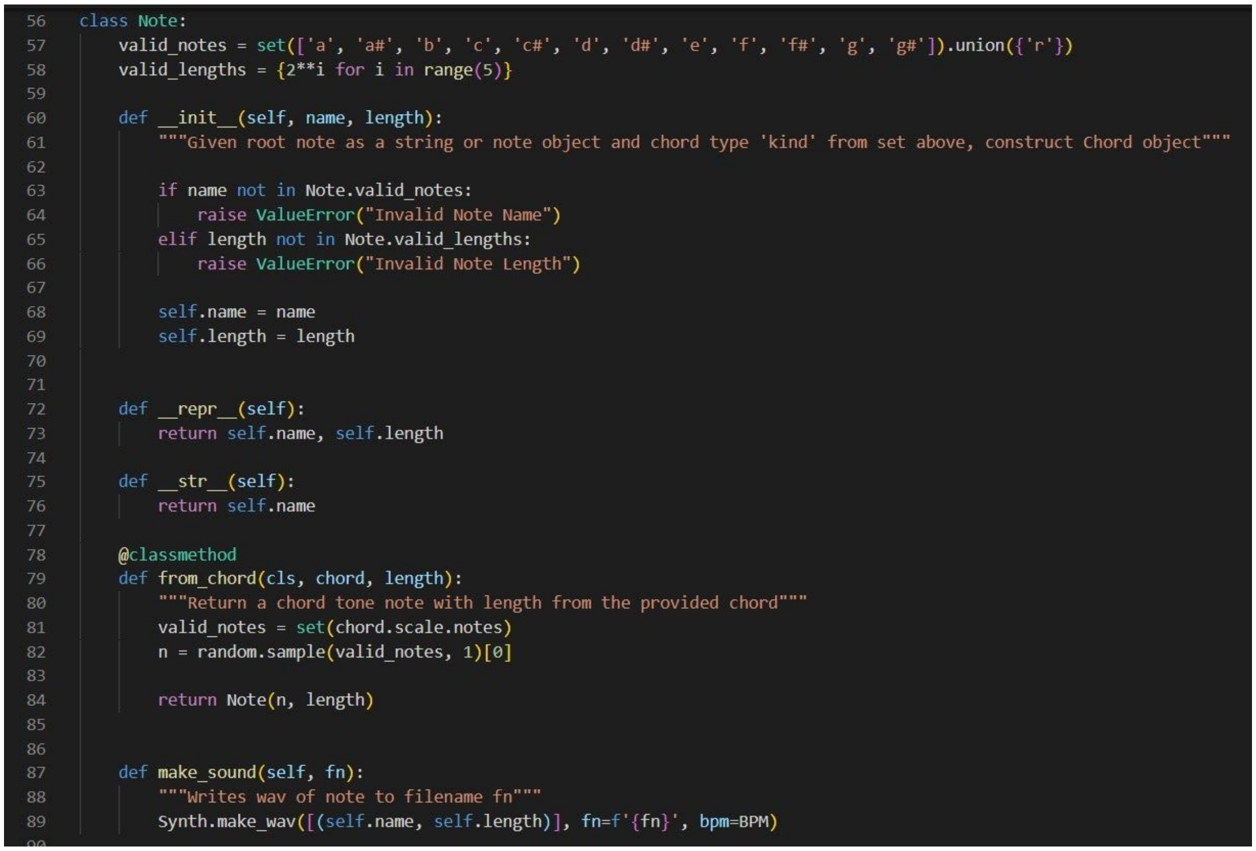
Project Writeup

The main idea for our project was to create an app that was able to generate digital musical output randomly using python code. Originally, we had a few different ideas on how we could potentially implement this. Some of the ideas were to take user input to determine what type of music would be created, but we decided that would have been far too complicated for us to execute. Ultimately, we created a rule-based system based on music theory that produces unique sound in 10 second clips. We decided to use these smaller clips so the app didn't create music indefinitely. Using this method, we were able to create a queue that accepts a list of these clips. If someone is on the app, it will play music until they leave the website, but the app will only produce more sound if the length of the list that contains the clips is under a certain length. This allows us to create music for a user anytime they are in the app, but not create music when there are no users on the site.

An example use case for our project is really any website that is looking to add music or some music feature. Similar to our simple web app, music would only be produced when a user was actually present on the web page that included the code/music feature. Another more obscure example use case could be using this type of system in an elevator or something similar that generally offers some sort of music for guests. This system would require the technology to monitor whether a user is in the physical space, if a guest is present the music would play and use the same queue system that is found in our web app, if no guests are present, music will not play. This technology and implementation could result in cost savings as music would no longer be played constantly in the elevator.

Our project originated as a system that would take some user input and produce music that corresponds to that input. However, after some brainstorming, we decided that this level of complexity would require a robust knowledge of music and music theory. It would have also required a lot more variation within the project. Had we decided to continue with this style of project, we likely would have been forced to create sounds that correspond to each possible input a user could choose. So for instance, if we gave users five different options for their current mood, we would have had to make five unique sound profiles. We ultimately decided against this because of the added complexity that it would require, and this format would increase the likelihood of our group encountering errors.

Instead we decided to simplify our project to avoid running into these errors. As was mentioned above, we decided to develop a rule-based system that creates unique digital music in a web app using flask. The rules were developed by Liam, using his knowledge of music and music theory. With these rules in mind, Liam created classes such as Notes, Chords, Bars, and Scales using OOP. Each class contains various methods, including methods that allow the object to be synthesized into sound and into a wav file.



```python
class Note:
    valid_notes = set(['a', 'a#', 'b', 'c', 'c#', 'd', 'd#', 'e', 'f', 'f#', 'g', 'g#']).union({'r'})
    valid_lengths = {2**i for i in range(5)}

    def __init__(self, name, length):
        """Given root note as a string or note object and chord type 'kind' from set above, construct Chord object"""

        if name not in Note.valid_notes:
            raise ValueError("Invalid Note Name")
        elif length not in Note.valid_lengths:
            raise ValueError("Invalid Note Length")

        self.name = name
        self.length = length


    def __repr__(self):
        return self.name, self.length

    def __str__(self):
        return self.name

    @classmethod
    def from_chord(cls, chord, length):
        """Return a chord tone note with length from the provided chord"""
        valid_notes = set(chord.scale.notes)
        n = random.sample(valid_notes, 1)[0]

        return Note(n, length)


    def make_sound(self, fn):
        """Writes wav of note to filename fn"""
        Synth.make_wav([(self.name, self.length)], fn=f'{fn}', bpm=BPM)
```

Using these new objects, we developed a queuing system so that wav files would not be created indefinitely when users are not present on the site. This does a few things for us. One, it saves storage space because we are not creating music unnecessarily while users are not present. Two, and related to the first, it is just much more efficient. If this site was like an elevator, music would be created whether or not a guest is present, this uses energy and storage that do not need to be used. Admittedly the costs are very minimal in the short term, however, over long periods of time these costs can add up. This system plays a bar if a user is present on the web app, when that bar is played, if the length of the list of bars that have been created becomes shorter than a predetermined number, 10 in our case, it will create another bar so the queue will not become empty while a user is on the site. One issue with this system is harmonic continuity: in plain english, do the chords sound good together? If multiple users are using the site at once, the queue will send each user the most recently generated bar; this can lead to somewhat ugly jumps at the end of each bar if the harmony deviates sufficiently during the bar sent to another user. This is something we hadn't considered in our initial design, but it isn't a dealbreaker. Our model's strength is intra-bar harmony and intra-chord melody, not long-form structure, so not much is lost by occasional harmonic jumps. It would be much more of an issue if we had implemented a more complex system for composing the harmony. In the future, if we were to revise this projectm

Our deployed app can be found at https://musictermproject.herokuapp.com/.