General guidelines:

- These homeworks are tentatively due at the end of weeks 2, 4, 6, 8, 10. If the corresponding lecture material (module 1 for homework 1 etc.) has not been completed by the deadline, the deadline will be moved by one week. But the final deadlines are always **as posted on gradescope.**
- Welcome to start early, but homeworks should be considered in "draft form" until the submission page is active on gradescope
- **Each homework is worth 10% of the final grade, and one HW is dropped, so that all HWs are worth 40%**[1]
- Homeworks must be completed individually

Homework 1: Sine wave generation and binary classification

Homework 1 stubs and files:
https://drive.google.com/drive/folders/1CKMJ5mUGngTZ7eYrqDG9tpzt0r8sICkS?usp=sharing

Part A - Sine Wave Generation

1. Write a function that converts a musical note name to its corresponding frequency in Hertz (Hz) (1 mark)
2. Write a function that linearly decreases the amplitude of a given audio (1 mark)
3. Write a function that adds a delay effect to a given audio where the output is a combination of the original audio and a delayed audio (1 mark)
4. Write a function that generates a melody by concatenating sine waves for a sequence of notes and durations; write additional functions to concatenate a list of audio arrays sequentially, and to "mix" audio arrays by scaling and summing them (simulating simultaneous playback) (1 mark)
5. Modify your code so that your pipeline can generate sawtooth waves by adding 18 harmonics based on the equation in the stub (1 mark)

Part B - Binary Classification

6. Write functions (as per the stub) to compute simple statistics about the files (1 mark)
7. Implement a few simple feature functions, to compute the lowest and highest MIDI note numbers in a file, and the set of unique notes in a file (1 mark)
8. Implement an additional feature extraction function to compute the average MIDI note number in a file (1 mark)
9. The autograder will split your dataset into train and test sets using `scikit-learn`, and will train your model to classify whether a given file is intended for piano or drums; provide the feature function to do so (1 mark)[2]

---

[1] This year, since there's no midterm, this will be rescaled to be out of 50

[2] I ended up discarding a more complex version of this question since I didn't want to get into ML details yet, so this one should be a freebie if you've gotten the questions above it right.

10. Creatively incorporate additional features into your classifier to make your classification more accurate (1 mark)

Submission: you should submit your homework to gradescope as **`homework1.py`**. Note that if using jupyter notebooks, you'll need to export your notebook as .py (rather than e.g. changing the extension). The autograder will import your submission (`#import homework1`) and compare your function outputs to a reference solution.

Homework 2: Spectrogram-based classification

Homework 2 stubs and files:
https://drive.google.com/drive/folders/1gyue7bNliSwrg3NEhFP28Ug_qLUmUMEh?usp=sharing

For most of these questions you should use the "pipeline.ipynb" file to train a model using the feature functions you write. pipeline.ipynb will train the model on the training set, save the best model on the validation set, and compute its performance on the test set. The autograder will not run this training pipeline: instead, you will upload your saved model, which the autograder will run on the test set. *Generally speaking* you will not need to modify the training code, though you may need to make tiny modifications (e.g. change a random seed, set a learning rate) to get good performance. All models should have more than 90% accuracy on the test set.

1. Load the waveforms from the data folder and extract labels for each **(1 mark)**
2. Extract MFCC features, and train an MLP classifier
    a. Implement the feature function correctly **(1 mark)**
    b. Upload the weights of a trained model (should be > 90% accurate) **(0.5 marks)**
3. Extract spectrograms, and train a CNN classifier **(1.5 marks)**
4. Extract mel-spectrograms, and train a CNN classifier **(1.5 marks)**
5. Extract constant-Q transform features and train a CNN classifier **(1.5 mark)**
6. Augment the dataset by implementing a pitch shift function **(1 mark)**
7. Extend the dataset to include four classes (modify the training labels, and either the MLP or CNN model), and train a classifier on this model. Creatively improve this model to obtain accuracy > 93%.[3] Modifications could include changes to the feature function, the model architecture, dataset augmentations, etc.
    a. Modify the training labels **(0.5 marks)**
    b. Modify the classifier code **(0.5 marks)**
    c. Train a model that is > 93% accurate **(1 mark)**

Submission: you should submit your homework to gradescope as **`homework2.py`**. You should also upload your weight files (e.g. **`best_mlp_model.weights`**).

To minimize the time it takes to run the autograder, I'd recommend first implementing all the feature functions correctly before uploading any weight files. This way, the autograder won't need to run the classifier code and will be much faster.

[3] To keep the homework straightforward, I haven't hidden the test set; though if people seem to be solving this by deliberately training on the test data, I'll probably do so…

Homework 3: Symbolic music generation using Markov chains

Homework 3 stubs and files:
https://drive.google.com/drive/folders/1YXZUe3gl85FMka0IhbMfoZUmlZYcqHh-?usp=sharing

1. Download the (subset of) PDMX data from the homework folder and unzip. Write a function to extract note pitch events from a midi file; extract all note pitch events from the dataset and output a dictionary that maps note pitch events to the number of times they occur in the files. (e.g. {60: 120, 61: 58, …}) **(1 mark)**
2. Write a function to normalize the above dictionary to produce probability scores. (e.g. {60: 0.13, 61: 0.065, …}) **(1 mark)**
3. Generate a table of pairwise probabilities containing p(next_note | previous_note) for the dataset; write a function that randomly generates the next note based on the previous note based on this distribution. **(1 mark)**
4. Write a function to calculate the perplexity of your model on a midi file. **(1 mark)**

   The *perplexity* of this model is defined as $exp(-\frac{1}{N}\sum_{i=1}^{N} log(p(w_i|w_{i-1})))$, where $p(w_1|w_0) = p(w_1)$; $p(w_i|w_{i-1})$ $(i > 1)$ refers to the probability p(next_note | previous_note).
5. Implement a second-order Markov chain, i.e., one which estimates p(next_note | next_previous_note, previous_note); write a function to compute the perplexity of this new model on a midi file. **(1 mark)**

   The perplexity of this model is defined as $exp(-\frac{1}{N}\sum_{i=1}^{N} log(p(w_i|w_{i-2,}w_{i-1})))$, where $p(w_1|w_{-1}, w_0) = p(w_1)$; $p(w_2|w_0, w_1) = p(w_2|w_1)$; $p(w_i|w_{i-2}, w_{i-1})$ $(i > 2)$ refers to the probability p(next_note | next_previous_note, previous_note).
6. Our model currently doesn't have any knowledge of *beats*. Write a function that extracts beat lengths and outputs a list of [(beat position; beat length)] values. **(1 mark)**
7. Implement a Markov chain that computes p(beat_length | previous_beat_length) based on the above function. **(1 mark)**
8. Implement a function to compute p(beat length | beat position), and compute the perplexity of your models from Q7 and Q8. For both models, we only consider the probabilities of predicting the sequence of *beat length*. (hint: refer to Q3 to derive the perplexity equation) **(1 mark)**
9. Implement a Markov chain that computes p(beat_length | previous_beat_length, beat_position), and report its perplexity. **(1 mark)**
10. Use the model from Q5 to generate N notes, and the model from Q8 to generate beat lengths for each note. Save the generated music as a midi file (see code from workbook1) as q10.mid. **(1 mark)**