Assignment 1

Programming and Music 1, November 13, 2025

This assignment concerns using SuperCollider as a tool to solve programming problems.

The first four exercises will be completed individually during class and handed in at the end of the session. These function as a short, exam-style check of your understanding of the basic programming tools covered so far.

The last four exercises must be completed independently outside of class. For these, you must submit both your code and a short written reflection (minimum 300 words) describing your approach, the strategies you used, and what you learned.

The final submission should include:

1. Code for exercises 1–4 (done in class).
2. Code for exercises 5–8 (done independently).
3. A 300-word reflection on your process.

Exercises:

---

1. Create a function named **printOverlap** that takes two arrays of numbers and finds all of those that exist **only in both** arrays and **prints** them the number of times indicated by each item to the *post window*.
   Input example: [1,2,3,4], [3,4,5,6]
   Output example: [3,3,3,4,4,4,4]

2. Write a function **sort** that takes as input an array of number. The function should **filter** the array and remove **duplicates** found among its items. It should print the duplicates found in the *post window*. It should then return an array without any duplicates that is **sorted** from the **highest** number to the **lowest** number.
   Input example: [12,4,6,6,4,8,8,2]
   Output example: [12,8,6,4,2]
3. Create a function that performs three **set operations** on a given input of two arrays. The output the program should return an array that contains three arrays. These are the **union** of the lists (all elements that are in A or in B (possibly both)) and their **intersection** (all elements that are both in A and B) and their **difference** (elements that are in A but not in B.). The input arrays could contain both numbers and letters.
   Input example: [1,2,3,4] [3,4,5,6]
   Output example: [[1,2,3,4,5,6], [3,4], [1,2]]
4. Create a function that **generates sentence** based on a given specification. The input that defines the output should be: **number of words** and **minimum** and **maximum length** of each word. The program should then be able to output formed sentences based on the words it generates. The length of each sentence should be variable.
   Input example: numberOfWords=4, min=3, max=6
   Output example: Rtuio oop afoli oiuytr.

---

5. Define a function called **generatePitches**. The function should be able to generate **different scales** (list of *pitches* or *intervals*). It should receive as input information about the *intervals*, *number of items* in a scale and any

other data needed for the process. The function should be able to return the scales but also allow for at least *two different* types of **filtering** (removing items from the list).

6. Implement a program that manages a collection of words. The words can either be generated or received as input. It should also be able to perform the following operations on them:
    a. *Sort* alphabetically
    b. *Sort* alphabetically in reverse order
    c. *Sort* by length
    d. *Sort* by length in reverse order
    e. Map the words into pitch numbers.
7. Implement a function named **RhymicalGenerator** that offers its user at least three different ways of generating arrays of durations based on a given input. These should then be played by a timed process (*Routine* or *Task*) with a simple *SynthDef*.
8. Implement a program called **MaterialGenerator**. It should be able to generate *pitches*, *durations* and *amplitudes* based on combining three different approaches for random and fixed numbers. At least two different generation methods need to be implemented for each of the lists. Finally the program should be able to play or audition the generated material to the composer using a timed process (*Routine* or *Task*) with a simple *SynthDef*.

The exam will take place on December 4th, 2025. The submission is on the same day.