EE 105 Fall 2025

Homework 1 – upload to grade scope

(Due September 11, before class, late submission will incur 20% points/day penalty)

Instructions: Perform the following tasks based on the circuits and concepts discussed in class. Be sure to show all work where applicable.

Problem 1: Proximity Alarm

Use the onboard proximity sensor to detect whether any object is within 3cm of the Arduino board. If anything gets within 3cm of the board, increase the blinking frequency of the built-in LED.

The Nano 33 BLE Sense includes a proximity sensor (APDS9960) that works by emitting infrared light and measuring the reflection from nearby objects. Inside the sensor, a photodiode converts the reflected light into an electrical signal, which is then digitized by an **Analog-to-Digital Converter (ADC)** built into the chip. An ADC takes a continuous analog signal (like a voltage that can smoothly vary with distance or light intensity) and converts it into a discrete number that a digital processor can understand. *For example, instead of "1.73 volts," the ADC might output a value like 542 on a 10-bit scale (0–1023).* Since the Arduino is a digital system that processes numbers rather than voltages, it cannot directly interpret analog signals without this conversion. The proximity sensor therefore delivers already-digitized values to the Arduino, allowing the board to easily use them in logic, plotting, or control tasks.

Checkpoint-0: Use Arduino blink code as baseline. Your system, by default, should have a blinking LED at 1Hz. (LED ON for 0.5s - OFF for 0.5s)

Answer (attach screenshots)

Checkpoint-1: Use APDS9960 library, continuously read proximity values, and print to Serial Monitor. (Include serial monitor screenshot in your report)

Answer (attach screenshots)

Checkpoint-2: Open Serial Plotter to see the values change in real time when moving your hand closer/farther. (Include serial plotter screenshot in your report)

Answer (attach screenshots)

Checkpoint-3: Get the threshold value of the sensor for an object (or, your hand) distance of 3cm from the board.

Answer (attach photos)

Checkpoint-4: Write a conditional (decision logic) so that whenever an object is within 3cm of the board, the built-in LED blinks at 4Hz frequency.

Answer (attach photos)

Answer (Please provide your entire Arduino code in your report.)

Problem 2: Sound Level Indicator

Use the PDM microphone to read sound levels. Write code that toggles the built-in LED ON/OFF when the sound amplitude exceeds a set threshold (e.g., clapping near the board).

The microphone in Nano 33 BLE Sense is a MEMS PDM mic. Pulse Density Modulation (PDM) is a way to represent an analog signal (like sound) as a 1-bit digital stream. Instead of storing multi-bit samples, PDM encodes the signal in the *density of ones and zeros*. High amplitude means more 1s than 0s in a short window, whereas a Low amplitude means more 0s than 1s. The microcontroller in Nano 33 BLE includes a hardware PDM interface that converts this bitstream into standard PCM (Pulse Code Modulation) samples (16-bit integers at audio rates like 16 kHz).

Checkpoint-1: Use Arduino PDM library, configure the microphone and verify using *if(PDM.available())* that the raw data is actually coming in.

Answer (attach screenshots)

Checkpoint-2: Arduino automatically decimates PDM → PCM (16-bit signed samples). Read samples into a buffer and print them to the Serial Monitor. (Show the printed values in your report)

Answer (attach screenshots)

Checkpoint-3: Measure sound levels and find threshold for a clap. Utilize serial plotter of the Arduino IDE.

Answer (Include these plots)

Checkpoint-4: Add a decision logic for toggling ON/OFF the built-in LED, when it hears a clap.

Answer (Please provide your entire Arduino code in your report.)

Problem 3: NumPy Array Creation and Row-wise Sorting

Write a function called create_and_sort_array(n) that performs the following operations:

1. Takes an integer n as input parameter
2. Creates a 2xn NumPy array where all entries are random integers between 1 and 50 (inclusive)
3. Sorts the array based on the values in the second row (index 1) in ascending order
4. Returns the sorted array

Function Signature:

```
def create_and_sort_array(n):
    """
    Input:
    n - integer, number of columns for the 2×n array

    Output:
    Returns a 2×n NumPy array with random integers (1-50),
    sorted by the second row values in ascending order.

    Example:
    n = 4
    Original array might be:
    [[23, 45, 12, 38],
     [17, 8,  41, 29]]

    Sorted array (by second row):
    [[45, 23, 38, 12],
     [8,  17, 29, 41]]
    """
    # YOUR CODE HERE
    return sorted_array
```

Requirements:

- Sort the entire array based on the second row values
- Both rows should be rearranged according to the sorting of the second row
- Return the final sorted NumPy array

Answer (Please provide your entire python code)

Problem 4: Dictionary and List Comprehension with Circuit Components

Write a function filter_high_power_resistors(resistors_dict) that takes a dictionary of resistor specifications and returns a sorted list of resistance values for resistors that can handle at least 0.5 watts.

Requirements:

- Use list comprehension (no for-loops)
- Filter resistors with power ⩾ 0.5 watts
- Return only resistance values, sorted in ascending order

Input Format:

```
resistors_dict = {
    'R1': {'value': 1000, 'power': 0.25},
    'R2': {'value': 220, 'power': 0.5},
    # ... more resistors
}
```

Example:

```
resistors = {
    'R1': {'value': 1000, 'power': 0.25},
    'R2': {'value': 220, 'power': 0.5},
    'R3': {'value': 470, 'power': 1.0}
}
# Expected output: [220, 470]
```

Function Signature:

```
def filter_high_power_resistors(resistors_dict):
    # YOUR CODE HERE
    return
```

Answer (Please provide your entire python code)

Problem 5: NumPy Array Slicing and Manipulation

Write a function extract_resistor_data(rows, cols) that creates a resistor value matrix and extracts specific data using slicing operations.

Requirements:

1. Create a 2D NumPy array of shape (rows, cols) where:
    - First row contains resistor values in order of 1s: [1, 2, 3, 4, ...] ohms
    - Second row contains values in order of 10s: [10, 20, 30, 40, ...] ohms
    - Third row contains values in order of 100s: [100, 200, 300, 400, ...] ohms
    - And so on...
2. Extract every 2nd row starting from row 0 (rows 0, 2, 4, ...)
3. From those rows, extract every 3rd column starting from column 1 (col. 1, 4, 7, ...)
4. Return the sliced array

Constraints:

- rows ⩽ 9
- cols ⩽ 9

Example:

```
# For rows=3, cols=8
# Created resistor matrix should be:
# [[   1    2    3    4    5    6    7    8]
# [   10   20   30   40   50   60   70   80]
# [  100  200  300  400  500  600  700  800]]

# Expected output after slicing (rows 0,2,4 and columns 1,4,7):
# [[   2    5    8]
# [  200  500  800]
```

Function Signature:

```
def extract_resistor_data(rows, cols):
    # Step 1: Create the resistor value matrix (rows ⩽ 20, cols ⩽ 9)
    # YOUR CODE HERE

    # Step 2: Apply slicing
    # YOUR CODE HERE

    return sliced_array
```

Answer (Please provide your entire python code)