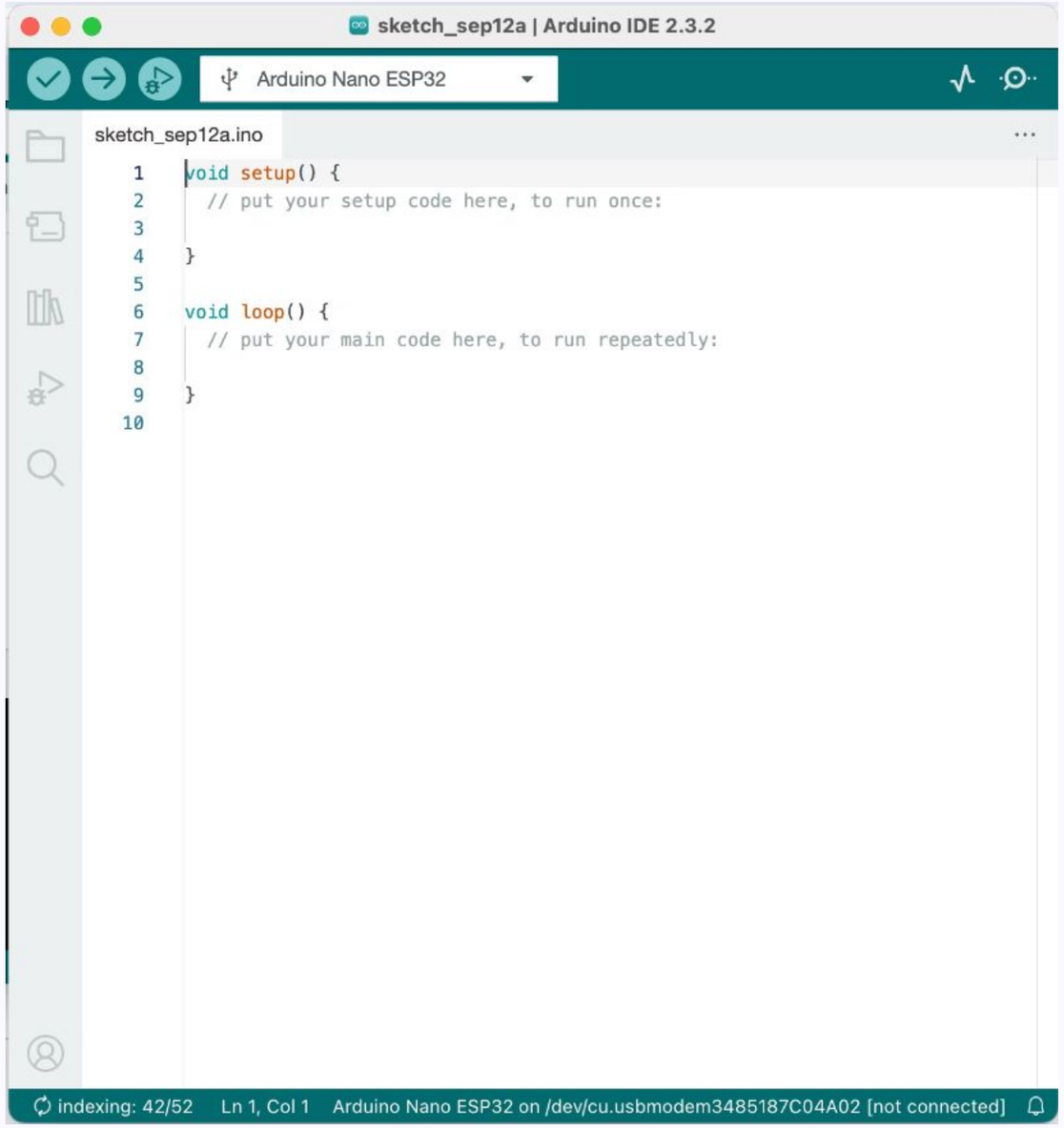
Lesson 5: Arduino Libraries for Sound



Arduino IDE (Integrated Development Environment)

Software application that provides everything needed to write, compile, and upload code to Arduino boards.

- Features code editor with syntax highlighting
- Includes compiler that converts your code into machine language
- Provides serial monitor for debugging
- Manages libraries for expanded functionality

Why Are Creative Coding Skills Still Important in the AI Era?

Understanding Fundamentals

AI can generate code, but understanding core principles of circuits, sensors, and programming enables you to customize instruments beyond templates and troubleshoot unexpected errors in your conductive ink projects.

Creative Problem-Solving

When building novel musical instruments like the Inkstrument, you'll encounter unique challenges that aren't in existing codebases or AI training data, requiring original solutions and adaptations.

Physical Computing Integration

Merging code with hardware like Arduino requires hands-on knowledge of how digital signals interact with analog materials like conductive ink, something AI assistance cannot fully replace.

Artistic Expression

Creative coding allows for unique artistic decisions in how gestures translate to sound, giving your instrument a signature character that generic AI-generated code cannot capture.

Balancing Fundamentals with Augmented Intelligence Tools for Creative

Troubleshooting Capability

Understanding circuit basics allows students to diagnose issues in their Inkstrument projects that AI might not recognize, especially when working with novel conductive materials.

Customization Beyond Templates

Knowledge of Arduino programming principles enables students to modify AI-generated code specifically for unique gesture-based sound control.

Material-Digital Integration

Grasping how conductive ink interacts with sensors allows students to create distinctive instruments while using AI to accelerate repetitive coding tasks.

Basic Arduino Programming

1

Structure

Sketches consist of setup() and loop().

2

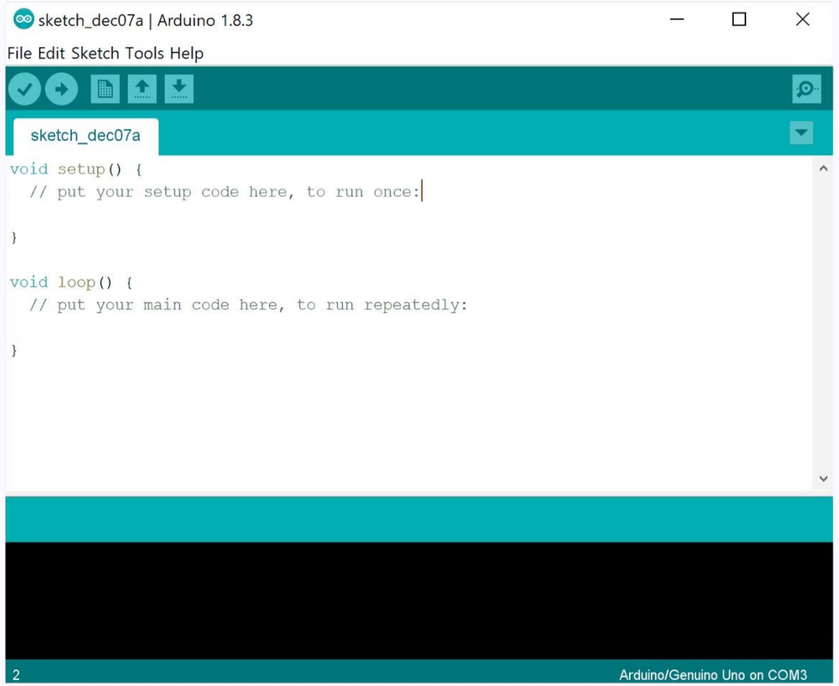
Setup

Initializes variables and pins.

3

Loop

Repeatedly executes the main code.



setup() and loop()

setup()

Runs once when your Arduino powers on or resets.

- Initialize pin modes with pinMode(pin, INPUT/OUTPUT)
- Start serial communication with Serial.begin(9600)
- Set initial values for variables

```
void setup() {
  pinMode(13, OUTPUT);  // LED pin
  Serial.begin(9600);   // Start serial monitor
  digitalWrite(13, LOW); // Initial state
}
```

loop()

Runs continuously after setup() completes.

- Contains your main program logic
- Reads sensors and controls outputs
- Executes repeatedly until power off

```
void loop() {
  digitalWrite(13, HIGH); // Turn LED on
  delay(1000);            // Wait 1 second
  digitalWrite(13, LOW);  // Turn LED off
  delay(1000);            // Wait 1 second
}
```

Hands-on: Blinking LED

Connect LED

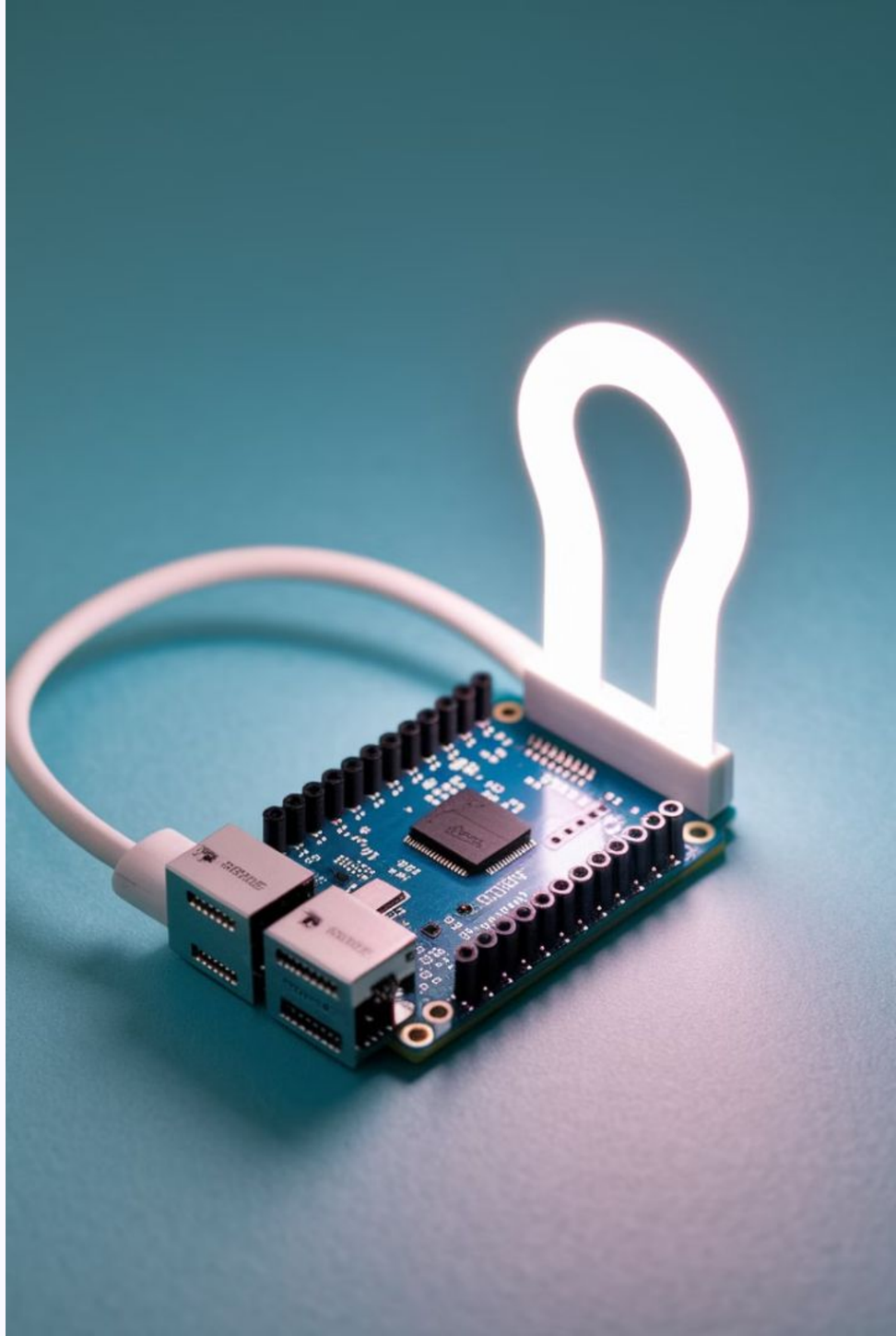
Wire an LED to an Arduino pin.

Write Code

Create a sketch to blink the LED.

Upload

Send the code to the Arduino board.



Blink

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}


// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

What is Serial Monitor in Arduino?

A debugging tool that allows communication between Arduino and computer.

Purpose

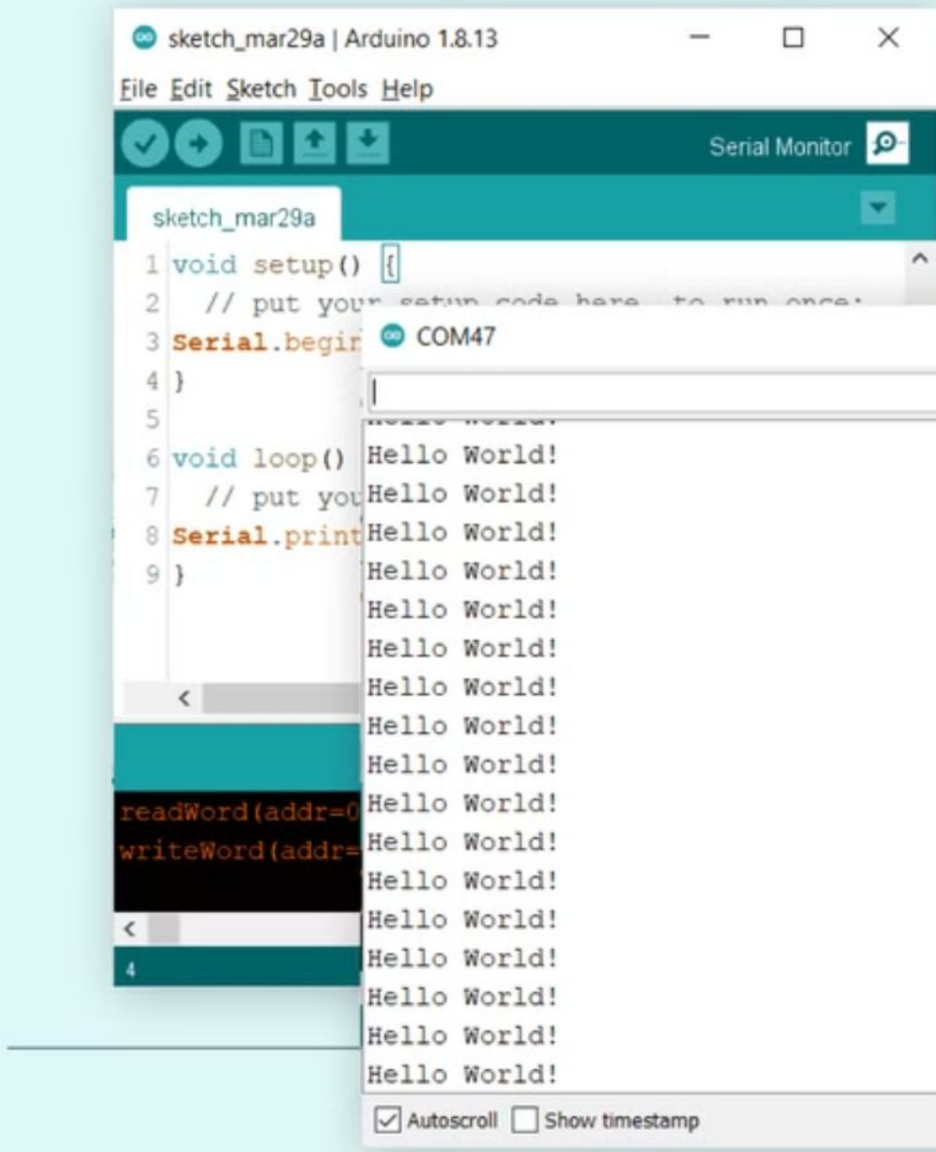
Displays data sent from Arduino to your computer, essential for troubleshooting and monitoring sensor values.

Usage

Accessed from Arduino IDE by clicking the magnifying glass icon or using Ctrl+Shift+M.

Configuration

Must match baud rate in code (e.g., Serial.begin(9600)) to properly display data.



Introduction to Arduino Libraries

1 **What Are They?**

Collections of pre-written code.

2 **Why Use Them?**

Simplify complex tasks and save time.

3 **Finding Them**

Available online and through the Arduino IDE.

What are Programming Library?

Reusable Code Packages

Collections of pre-written code that perform specific functions, saving developers development time.

Arduino Library Structure

Arduino libraries typically consist of C++ header (.h) and implementation (.cpp) files, organized to provide easy-to-use functions.

Arduino Libraries we use

Tone Library

Generates square wave tones on any Arduino pin, essential for creating musical notes in our Inkstrument project. Provides functions like tone() and noTone() for controlling pitch output.

Capacitive Sensor Library

Enables Arduino to detect touch through conductive materials. This library is critical for transforming our conductive ink drawings into interactive touch sensors.

SFEMP3Shield Library

Interfaces with the VS1053 MP3 decoder chip to play audio files stored on an SD card. Allows our Inkstrument to produce high-quality sounds beyond simple tones.

SdFat Library

Provides efficient and robust access to SD cards in FAT16 or FAT32 format. Essential for storing and retrieving sound samples and configuration files for our instrument.

Arduino Programming Fundamentals

Building our Inkstrument requires understanding key programming concepts in Arduino:

Variables & Data Types

Declaring variables with appropriate types (int, float, boolean) to store sensor readings from our conductive ink interfaces

Global Variables

Using global variables accessible across the entire program to manage state for our touch sensors and sound playback features

Library Imports

Including external .h library files for specialized functions like the Capacitive Sensor and SFEMP3Shield libraries we downloaded for touch detection and audio playback

touch_to_play_mp3

```
// Include necessary libraries for SD card, utility functions, and MP3 player shield.
#include <SdFat.h>
#include <SdFatUtil.h>
#include <SFE_MPHAT_Shield.h>

// Create instances of the SdFat (SD card) and SFEMP3Shield (MP3 player) classes.
SdFat sd;
SFEMP3Shield MP3player;

// Variables to store analog readings from pins A0 to A5.
int InData0 = 0; // Analog reading from pin A0
int InData1 = 0; // Analog reading from pin A1
int InData2 = 0; // Analog reading from pin A2
int InData3 = 0; // Analog reading from pin A3
int InData4 = 0; // Analog reading from pin A4
int InData5 = 0; // Analog reading from pin A5
int InData0 = 0;

// Sensitivity threshold for detecting touch inputs.
int TouchSensitivity = 280;
```

.h C++ library in Arduino

Header (.h) files in Arduino are C++ libraries that contain pre-written code, function definitions, and variable declarations that extend Arduino's functionality.

For our Inkstrument project, we're using two critical libraries:

The **CapacitiveSensor library** (.h) that enables our Arduino to detect touch through conductive ink patterns
The **SFEMP3Shield library** (.h) that provides functions for playing audio files from the VS1053 MP3 decoder

touch_to_play_mp3

```
void setup() {
    // Initialize serial communication at 115200 baud rate for debugging purposes.
    Serial.begin(115200);

    // Initialize the SD card using pin 9 as the chip select pin (CS).
    // If the SD card fails to initialize, halt the program and display an error.
    if (!sd.begin(9, SPI_HALF_SPEED)) sd.initErrorHalt();

    // Change directory to the root folder ("/") on the SD card.
    // If the directory change fails, halt the program and display an error.
    if (!sd.chdir("/")) sd.errorHalt("sd.chdir(\"/\")");

    // Initialize the MP3 player and set the volume level to maximum (10, 10).
    MP3player.begin();
    MP3player.setVolume(10, 10);

    // Configure pins A0 to A5 as input pins to read analog values for touch sensing.
    for (int i = A0; i <= A5; i++) {
        pinMode(i, INPUT);
    }

    // Disable Timer0 overflow interrupt to avoid conflicts with other operations.
    TIMSK0 &= !(1 << TOIE0);
}
```

Serial.begin 115200 vs 9600 ?

Serial communication in Arduino requires setting a baud rate - the speed at which data is transmitted between the Arduino and your computer.

115200 baud

- Faster data transmission (12x faster than 9600)
- Better for high-volume data like sensor readings
- Ideal for our Inkstrument's real-time responsiveness
- What we use in our touch_to_play_mp3 code

9600 baud

- Arduino's traditional default rate
- More stable for basic projects
- Sufficient for simple debugging
- Slower, but compatible with older hardware

MP3player.begin() & MP3player.setVolume()

MP3player.begin()

Initializes the VS1053 MP3 decoder shield to prepare it for audio playback. This function:

- Sets up communication between Arduino and the VS1053 chip
- Configures internal settings for optimal audio performance
- Must be called before any other MP3player functions

MP3player.setVolume(10, 10)

Adjusts the output volume level of both left and right audio channels:

- Parameters represent left and right channel volumes

touch_to_play_mp3

```
void loop() {
    // Read analog values from pins A0 to A5 and invert them (higher values mean touch detected).
    InData0 = 1024 - analogRead(A0); // Invert A0 reading
    InData1 = 1024 - analogRead(A1); // Invert A1 reading
    InData2 = 1024 - analogRead(A2); // Invert A2 reading
    InData3 = 1024 - analogRead(A3); // Invert A3 reading
    InData4 = 1024 - analogRead(A4); // Invert A4 reading
    InData5 = 1024 - analogRead(A5); // Invert A5 reading


    ...
}
```

What is analogRead()?

The analogRead() function is an Arduino command that reads the value from a specified analog pin, converting voltage (0-5V) into integer values (0-1023).

Reading sensor inputs

In our Inkstrument project, we use analogRead() to measure the conductive ink sensor values from pins A0-A5.

Value interpretation

Higher values (closer to 1023) indicate greater conductivity or touch intensity, which we invert (1024 - analogRead()) to make touch detection more intuitive.

Touch sensitivity threshold

We compare these analog readings against our TouchSensitivity value (280) to determine when a conductive ink pad is being touched.

This function is essential for converting the analog electrical signals from our conductive ink sensors into digital values our Arduino can use to trigger sounds.

Why 1024 - analogRead()?

In our Inkstrument project, we use the formula **1024 - analogRead()** to invert the sensor readings from our conductive ink touch pads. This inversion serves three critical purposes:

Intuitive Touch Response

Without inversion, a stronger touch would produce a lower value (closer to 0) because our conductive ink creates a path to ground when touched. By inverting with 1024 - analogRead(), higher values (closer to 1024) now represent stronger touches.

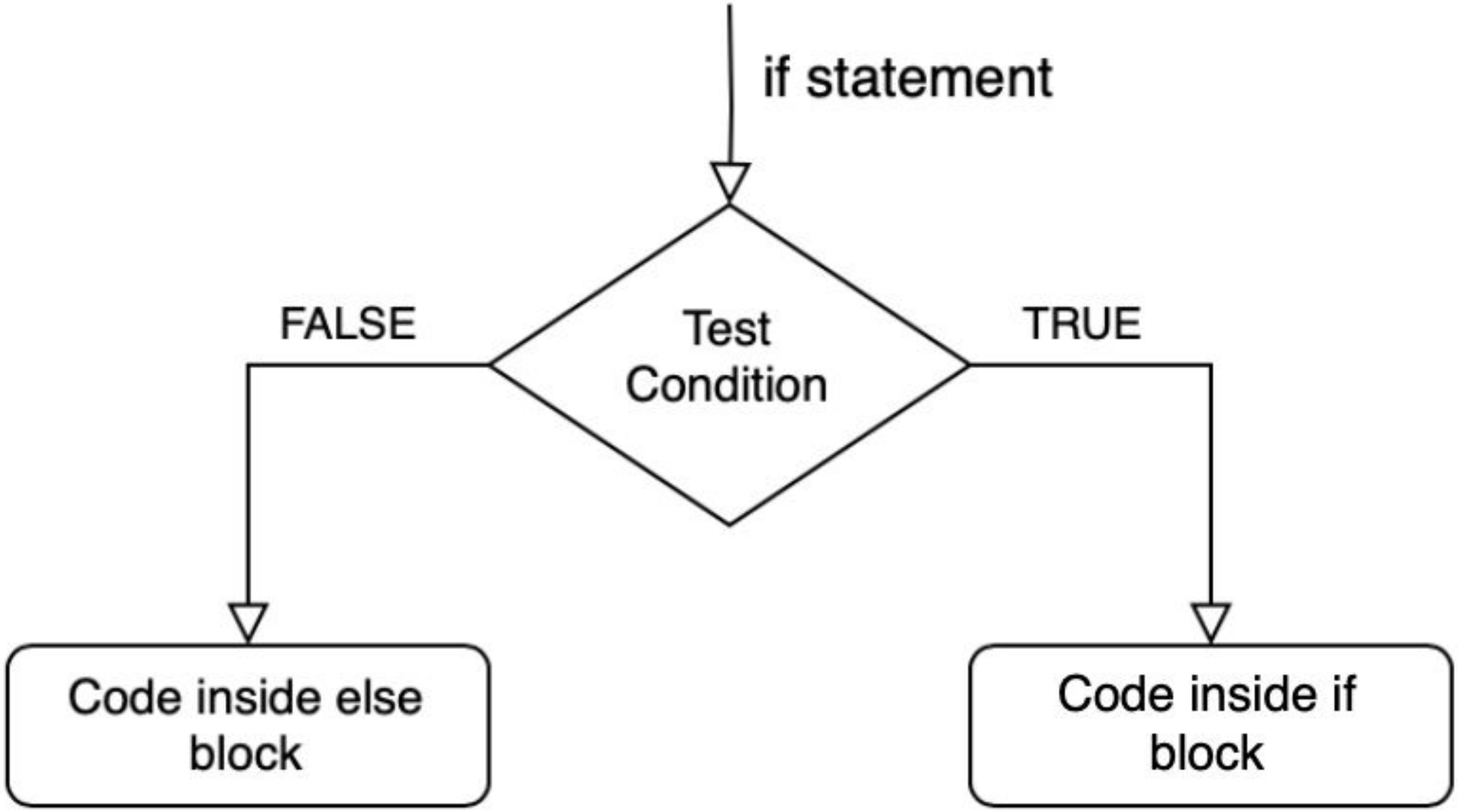
touch_to_play_mp3

```
void loop() {
    ...
    // Check which pin has been touched based on the sensitivity threshold.
    // Play the corresponding MP3 track if a touch is detected.
    if (InData0 >= TouchSensitivity) {
        MP3player.playTrack(0); // Play track 0 if A0 is touched
    } else if (InData1 >= TouchSensitivity) {
        MP3player.playTrack(1); // Play track 1 if A1 is touched
    } else if (InData2 >= TouchSensitivity) {
        MP3player.playTrack(2); // Play track 2 if A2 is touched
    } else if (InData3 >= TouchSensitivity) {
        MP3player.playTrack(3); // Play track 3 if A3 is touched
    } else if (InData4 >= TouchSensitivity) {
        MP3player.playTrack(4); // Play track 4 if A4 is touched
    } else if (InData5 >= TouchSensitivity) {
        MP3player.playTrack(5); // Play track 5 if A5 is touched
    } else {
        MP3player.stopTrack(); // Stop playback if no touch is detected
    }
    ...
}
```

Conditionals if(){}else{}

Conditional statements are crucial for our Inkstrument's interactivity. They allow the Arduino to make decisions based on sensor input values.

```
if (InData0 >= TouchSensitivity) {
    MP3player.playTrack(0);  // Play first sound if touched
} else if (InData1 >= TouchSensitivity) {
    MP3player.playTrack(1);  // Play second sound if touched
} else {
    MP3player.stopTrack();   // Otherwise stop all sounds
}
```


if statement
Test Condition
FALSE
TRUE
Code inside else block
Code inside if block

touch_to_play_mp3

```
void loop() {
    ...
    ...
    // Print the current state of the MP3 player and analog readings to the serial monitor.
    Serial.print(MP3player.isPlaying()); // Print whether the MP3 player is currently playing
    Serial.print(" ");
    Serial.print(InData0); // Print inverted value of A0
    Serial.print(" ");
    Serial.print(InData1); // Print inverted value of A1
    Serial.print(" ");
    Serial.print(InData2); // Print inverted value of A2
    Serial.print(" ");
    Serial.print(InData3); // Print inverted value of A3
    Serial.print(" ");
    Serial.print(InData4); // Print inverted value of A4
    Serial.print(" ");
    Serial.println(InData5); // Print inverted value of A5 followed by a newline

    // Add a delay to prevent rapid polling of the touch pins.
    delay(100);
}
```

Serial.print()

Serial.print() is essential for debugging our Inkstrument. It sends data from Arduino to your computer, allowing you to observe sensor readings in real-time.

```
// Output touch sensor values to Serial Monitor
Serial.print(InData0);  // First ink sensor reading
Serial.print(" ");      // Add space between values
Serial.print(InData1);  // Second ink sensor reading
Serial.println(InData2); // Add newline after last value
```

When designing your Inkstrument, use Serial.print() to:

- Calibrate touch sensitivity thresholds
- Verify that your conductive ink paths are functioning
- Debug issues when sounds aren't triggering as expected

Open the Serial Monitor (Tools → Serial Monitor) with baud rate set to 9600 to view the output.