

BXECURE LABS LEARNING MANUAL

Programming C++ with Arduino

A Practical Guide for Engineering Students and Beginners

Developed by Elipklim Nyahe

Published by BXecure Labs

Tools used Arduino IDE, Arduino Uno, USB cable and basic electronic components

Contents

1 Introduction to Embedded Systems and Arduino

2 C++ Fundamentals for Engineers

3 Arduino Program Structure

4 Variables Data Types and Operators

5 Control Flow Decisions and Loops

6 Functions in C++

7 Digital Input and Output

8 Analog Input and Output

9 Serial Communication

10 Arrays and Strings

11 Sensors and Actuators

12 Timing and Interrupts

13 Object Oriented Programming Basics

14 Debugging and Best Practices

15 Mini Projects and Practicals

Quick Reference Cheat Sheet

Glossary

Recommended Learning Schedule

Using this manual

This manual introduces C++ through practical Arduino programming and electronics examples. It is designed for engineering students, beginners and makers who want to understand both the code and the physical system it controls.

The chapters progress from board setup and program structure to inputs, outputs, sensors, actuators, serial communication, timing, debugging and small projects. Work through the examples in order if you are new to programming, or use the contents page to review a specific topic.

Type and test the examples yourself. Begin with one circuit or module, compare the observed result with the explanation, and record any error message or unexpected behaviour before changing several things at once.

Safety: Disconnect power before changing wiring. Check voltage, current requirements and polarity before powering a circuit. Do not connect motors directly to an Arduino input or output pin.

Some examples refer to components that may not be included in every BXecure Labs kit. Confirm the contents of your kit before beginning an exercise and consult the relevant component datasheet when specifications matter.

1. Introduction to Embedded Systems and Arduino

1.1 What is an Embedded System?

An embedded system is a small computer built inside a device to make it do one specific job. Think of it this way: your laptop is a general-purpose computer that can do many things. An embedded system is a tiny computer hidden inside something else — and it does only that one job, over and over.

Everyday examples:

- The controller inside a microwave oven (heats food for the right time)
- Traffic light systems (switches lights on a timer or on sensor input)
- Anti-lock brakes in a car (monitors wheel speed and releases brake pressure automatically)
- Digital thermometers (reads temperature and shows it on a display)
- Washing machine control panels

You will learn to build and program embedded systems using Arduino.

1.2 What is Arduino?

Arduino is two things at once:

1. **Hardware** — a small circuit board with a microcontroller chip at its heart
2. **Software** — a free program (the Arduino IDE) where you write and upload code

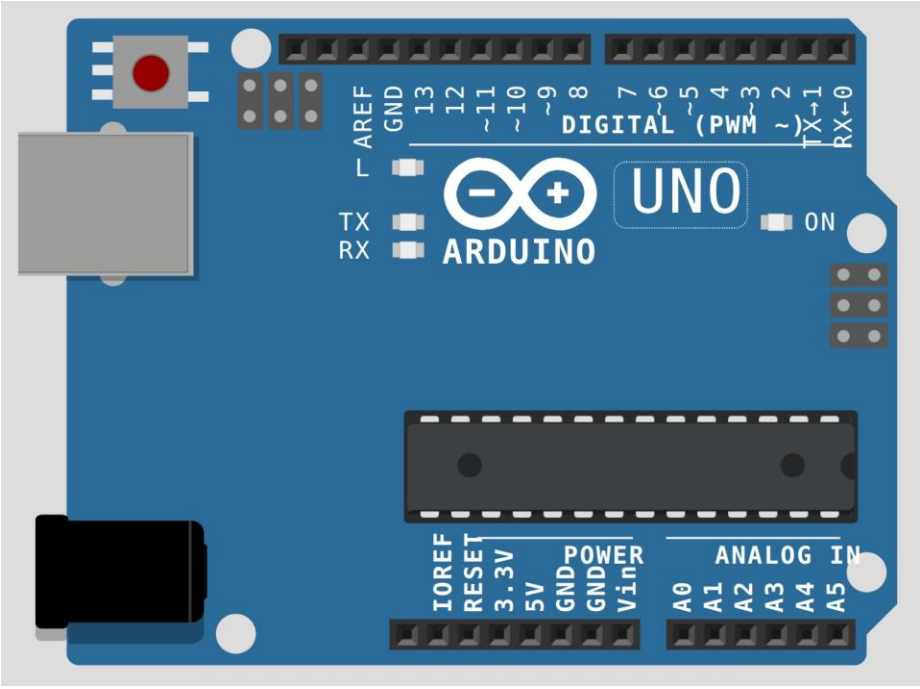


Figure 1.1 — The Arduino Uno board

Arduino was designed to make electronics programming accessible to everyone — artists, students, and beginners — while still being powerful enough for professional engineers.

1.3 The Arduino Uno Board — Every Part Explained

The Arduino Uno is the most popular board for beginners. Here is a labeled breakdown:

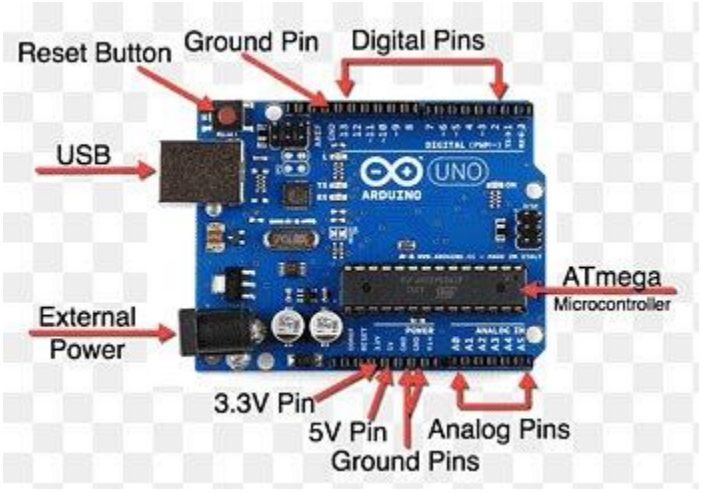


Figure 1.2 — Arduino Uno with key parts labeled

Part	What It Does
Microcontroller (ATmega328P)	The "brain" — executes your code
USB Port	Connects to your computer for programming and power
Power Jack	Connects an external power adapter (7–12V)
Digital Pins (0–13)	Can be set as input or output for signals (ON/OFF)
Analog Pins (A0–A5)	Read variable voltage values (sensors, potentiometers)
PWM Pins (~3,5,6,9,10,11)	Can produce variable-strength outputs
5V / 3.3V / GND Pins	Power supply pins for your components
Reset Button	Restarts your program from the beginning
TX/RX LEDs	Blink when data is being sent/received via USB
Built-in LED (Pin 13)	An LED on the board itself, great for testing

Microcontroller Specs:

Specification	Value
Chip	ATmega328P
Clock Speed	16 MHz
Flash Memory (program storage)	32 KB
SRAM (runtime variables)	2 KB
EEPROM (permanent data)	1 KB
Operating Voltage	5V
Digital I/O Pins	14

Specification	Value
Analog Input Pins	6

1.4 What is a Breadboard?

Before touching code, you need to understand the breadboard — the most important prototyping tool in electronics.

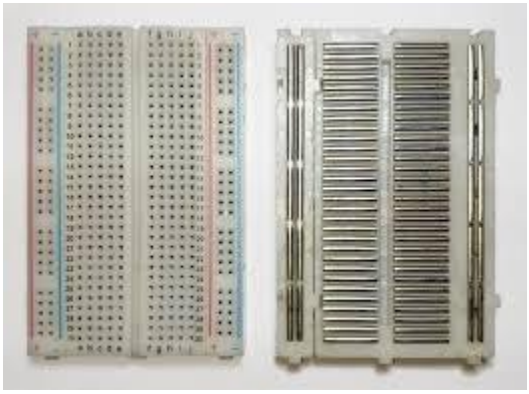


Figure 1.3 — A breadboard with its internal connections shown

Key points:

- Holes in the same horizontal row (in the middle section) are connected to each other
- The two vertical rails on each side (marked + and −) run the full length and are used for power (5V) and ground (GND)
- No soldering needed — just push component legs into the holes

1.5 Setting Up the Arduino IDE

The Arduino IDE (Integrated Development Environment) is where you write, verify, and upload your programs.

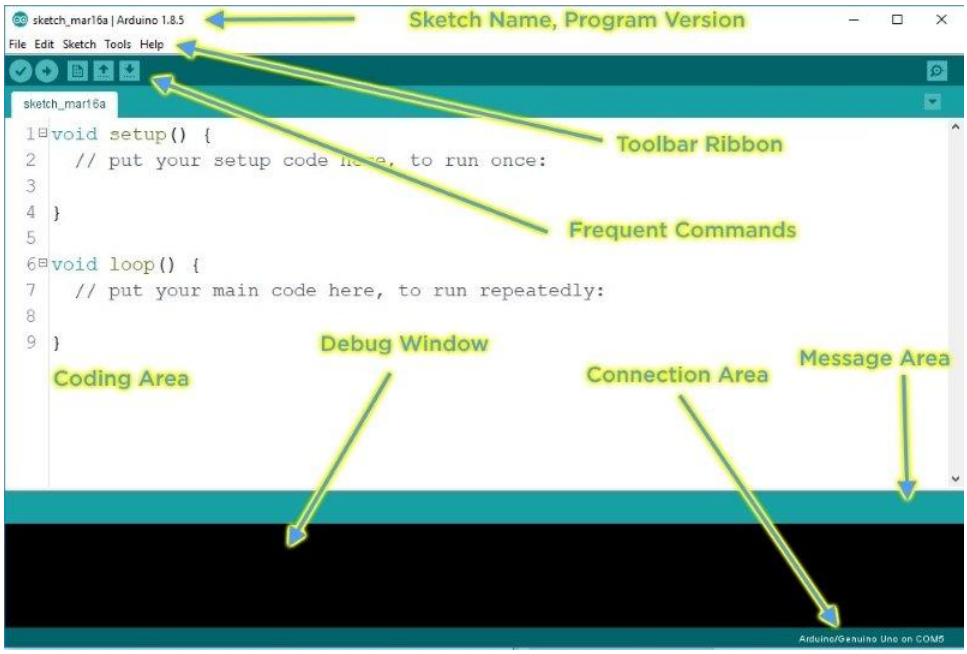


Figure 1.4 — The Arduino IDE with its main areas labeled

Setup steps:

3. Download Arduino IDE from https://www.arduino.cc/en/software
4. Install and launch the IDE
5. Connect your Arduino Uno via USB cable
6. Go to Tools → Board → Arduino AVR Boards → Arduino Uno
7. Go to Tools → Port and select the correct port (Windows: COM3/COM4; Mac/Linux: /dev/ttyUSB0 or /dev/cu.usbmodem...)
8. You are ready to write code!

> **TIP:** The checkmark (✓) button verifies (compiles) your code. The arrow (→) button uploads it to the Arduino. Always verify before uploading.

1.6 Your First Program — Blinking the Built-in LED

Every programmer starts with "Hello World." For hardware engineers, the equivalent is blinking an LED. The Arduino Uno has a built-in LED already connected to Pin 13.

```
// Blink: The "Hello World" of Arduino
// This program blinks the built-in LED on and off every second

void setup() {
    pinMode(13, OUTPUT);  // Tell Arduino: pin 13 is an OUTPUT
}

void loop() {
    digitalWrite(13, HIGH);  // Turn LED ON  (send 5V to pin 13)
    delay(1000);              // Wait 1000 milliseconds = 1 second
    digitalWrite(13, LOW);   // Turn LED OFF (send 0V to pin 13)
    delay(1000);              // Wait 1 second
}
```

To run this program:

9. Type the code into the IDE (or use File → Examples → Basics → Blink)
10. Click ✓ to verify — look for "Done compiling" at the bottom
11. Click → to upload — the TX/RX LEDs will flash briefly
12. Watch the LED on your board blink!

2. C++ Fundamentals for Engineers

2.1 What is C++?

C++ is the programming language you will use to give instructions to your Arduino. Think of it as the language Arduino "speaks."

C++ is one of the most widely used languages in the world, especially for:

- Embedded systems and microcontrollers
- Operating systems
- Game engines
- High-performance software

Arduino programs are written in C++ with a simplified structure — making it beginner-friendly while still teaching you real, industry-standard code.

2.2 How Your Code Gets to the Arduino

When you click "Upload," here is what happens behind the scenes:

```
Your Code (.ino file)
        |
  Compiler (avr-gcc)
  Translates C++ into
  machine instructions
        |
  Binary (.hex file)
  Just 1s and 0s the
  chip understands
        |
  Uploaded via USB
  into Flash Memory
        |
  Arduino runs the
  code immediately
```

You write human-readable C++. The compiler translates it. The Arduino executes it.

2.3 Comments — Notes in Your Code

Comments are messages left in code for humans to read. The compiler completely ignores them. Always comment your code — it helps you (and others) understand it later.

```
// This is a single-line comment. Everything after // is ignored.

/*
   This is a multi-line comment.
   It can span as many lines as you need.
   Useful for longer explanations.
*/

int x = 5;   // You can also put a comment after a line of code
```

TIP: Write comments before you write the code. It forces you to think through what you want to do first.

2.4 Statements and Semicolons

Every instruction (called a statement) in C++ must end with a semicolon (;).

Think of the semicolon like a period at the end of a sentence. Without it, C++ doesn't know where one instruction ends and the next begins.

```
int a = 10;          // Correct — semicolon at the end
int b = 20;          // Correct
int c = 30           // ERROR: missing semicolon — code won't compile!
digitalWrite(13, HIGH);  // Correct
```

2.5 Code Blocks and Curly Braces

Curly braces { } group multiple statements into a block that belongs together.

```
void setup() {
    // Everything between these braces runs as one group
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
    Serial.begin(9600);
}   // <-- closing brace ends the block
```

Every opening { must have a matching closing }. Mismatched braces are one of the most common beginner errors.

3. Arduino Program Structure

3.1 The Two Required Functions

Every Arduino program must have exactly these two functions:

```
void setup() {
    // This runs ONCE — when the board powers on or is reset
    // Use this section to:
    //   - Set pin modes (input or output)
    //   - Initialize Serial communication
    //   - Configure sensors and displays
}

void loop() {
    // This runs FOREVER — repeating over and over
    // Your main program logic goes here
    // When it reaches the closing }, it starts again from the top
}
```

3.2 Visualizing the Execution Flow

```
+--------------------------------------+
|         Power ON or Reset            |
+------------------+-------------------+
                   |
                   v
+--------------------------------------+
|     setup() — runs exactly ONCE      |
|  pinMode(13, OUTPUT);                |
|  Serial.begin(9600);                 |
+------------------+-------------------+
                   |
                   v
+--------------------------------------+
|     loop() — runs FOREVER            |
|   digitalWrite(13, HIGH);            |
|   delay(1000);                       |
|   digitalWrite(13, LOW);             |
|   delay(1000);                       |
|        (repeats forever)             |
+--------------------------------------+
```

3.3 Including Libraries

Libraries are collections of pre-written code that give you extra functionality. You include them at the very top of your program:

```
#include <Servo.h>          // Adds functions to control servo motors
#include <Wire.h>           // Adds I2C communication support
#include <LiquidCrystal.h>  // Adds LCD display support

void setup() { }
void loop()  { }
```

Think of libraries like apps — someone else wrote the hard parts, and you just use them.

3.4 Defining Constants

Constants are values that never change. Name them at the top of your file so they're easy to find and update:

```
// Method 1: Using #define (no data type, no semicolon)
#define LED_PIN 13
```

```
#define BAUD_RATE 9600

// Method 2: Using const (has a data type — preferred in C++)
const int BUTTON_PIN = 2;
const float REFERENCE_VOLTAGE = 5.0;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    Serial.begin(BAUD_RATE);
}
```

TIP: Why use constants? If you later change LED_PIN from 13 to 12, you only change it in one place — at the top — instead of hunting through your entire program.

4. Variables, Data Types, and Operators

4.1 What is a Variable?

A variable is a named box in the Arduino's memory that stores a value. You give the box a name, and you can put a value in it, read it, or change it.

```
+-----------------------------------------------+
|  Memory (SRAM)                                |
|                                               |
|   [ age ]      [ voltage ]     [ isOn ]       |
|   [  21 ]      [   3.3   ]     [ true ]       |
+-----------------------------------------------+
```

Declaring and using variables:

```
int age = 21;            // Create a variable called "age", store 21
float voltage = 3.3;     // Store the decimal number 3.3
char grade = 'A';        // Store the character A (note: single quotes)
bool isOn = true;        // Store true or false

age = 22;                // Change age to 22
voltage = voltage + 0.1; // voltage is now 3.4
```

4.2 Data Types — Choosing the Right Box

Different types of data need different types of boxes. Choosing the right type saves memory and prevents bugs:

Data Type	Size	Range	When to Use
bool	1 byte	true or false	ON/OFF states, flags
byte	1 byte	0 to 255	Small positive numbers
int	2 bytes	-32,768 to 32,767	General counting, pin numbers
unsigned int	2 bytes	0 to 65,535	Non-negative counts
long	4 bytes	+/- 2.1 billion	Large numbers (e.g., millis())
unsigned long	4 bytes	0 to ~4.3 billion	Timestamps
float	4 bytes	Decimals (~7 sig. figures)	Voltages, temperatures
char	1 byte	One character	Letters, symbols
String	Variable	Any text	Messages, labels

```
bool ledState = false;      // Is the LED on or off?
byte brightness = 200;      // 0-255 for PWM
int sensorValue = 512;      // Raw ADC reading
long timestamp = 0;         // Will store millis() values
float tempCelsius = 23.5;   // Temperature reading
char initial = 'E';         // First letter of a name
String name = "Arduino";    // A whole word or sentence
```

WARNING: Important for Uno users: On the Arduino Uno, double has the same precision as float (4 bytes). Do NOT assume extra decimal precision.

4.3 Local vs. Global Variables

Where you declare a variable determines where you can use it:

```
int globalCount = 0;  // Global: can be used in ANY function

void setup() {
    int localVal = 10;  // Local: only exists INSIDE setup()
    globalCount = 5;    // Accessing global is fine
}

void loop() {
    globalCount++;      // Fine — global is accessible everywhere
    // localVal++;      // ERROR — localVal doesn't exist here
}
```

TIP: Rule of thumb: Use local variables when possible. Use globals only when a value must be shared between setup() and loop().

4.4 Arithmetic Operators

```
int a = 10, b = 3;

int result1 = a + b;   // 13  — Addition
int result2 = a - b;   // 7   — Subtraction
int result3 = a * b;   // 30  — Multiplication
int result4 = a / b;   // 3   — Integer division (decimal part is dropped!)
int result5 = a % b;   // 1   — Modulo: the remainder after dividing
```

WARNING: Integer Division Trap: 10 / 3 gives 3, NOT 3.333. Both numbers are integers, so the result is also an integer. Fix: use float — (float)10 / 3 gives 3.333.

4.5 Comparison Operators

These compare two values and produce true or false:

```
int x = 10, y = 5;

x == y   // Is x equal to y?         -> false
x != y   // Is x NOT equal to y?     -> true
x > y    // Is x greater than y?     -> true
x < y    // Is x less than y?        -> false
x >= y   // Is x >= y?               -> true
x <= y   // Is x <= y?               -> false
```

4.6 Logical Operators

Combine multiple conditions:

```
bool hot = true;
bool rainy = false;

hot && rainy   // AND: both must be true -> false
hot || rainy   // OR:  at least one true -> true
!hot           // NOT: flips the value  -> false
```

Real example:

```
if (temperature > 30 && humidity > 70) {
    Serial.println("Heat alert! Both temp and humidity are high.");
}
```

4.7 Shorthand Assignment Operators

These are shortcuts that modify a variable and save it back:

```
int n = 10;
n += 5;    // n = n + 5  ->  n is now 15
n -= 3;    // n = n - 3  ->  n is now 12
n *= 2;    // n = n * 2  ->  n is now 24
n /= 4;    // n = n / 4  ->  n is now 6
n++;       // n = n + 1  ->  n is now 7
n--;       // n = n - 1  ->  n is now 6
```

5. Control Flow: Decisions and Loops

5.1 The if Statement

Makes a decision: "Do this ONLY if the condition is true."

```
int temperature = 35;

if (temperature > 30) {
    Serial.println("It is hot!");  // Only runs if temperature > 30
}
```

5.2 if-else Statement

Do one thing if true, a different thing if false:

```
int lightLevel = analogRead(A0);

if (lightLevel < 300) {
    digitalWrite(LED_PIN, HIGH);   // Dark -> turn on the light
} else {
    digitalWrite(LED_PIN, LOW);    // Bright -> turn off the light
}
```

5.3 if-else if-else Chain

Handle multiple conditions in sequence:

```
int score = 75;

if (score >= 90) {
    Serial.println("Grade: A – Excellent!");
} else if (score >= 80) {
    Serial.println("Grade: B – Very Good");
} else if (score >= 70) {
    Serial.println("Grade: C – Good");
} else if (score >= 60) {
    Serial.println("Grade: D – Pass");
} else {
    Serial.println("Grade: F – Fail");
}
```

Only one of these blocks will run — the first one whose condition is true.

5.4 The switch Statement

Use switch when comparing one variable against several known values. It's neater than a long if-else if chain for this purpose:

```
int mode = 2;

switch (mode) {
    case 1:
        Serial.println("Mode: IDLE");
        break;
    case 2:
        Serial.println("Mode: RUNNING");
        break;
    case 3:
        Serial.println("Mode: ERROR");
        break;
    default:
```

```
        Serial.println("Unknown mode!");
        break;
}
```

WARNING: Always include break at the end of each case. Without break, execution continues falling through into the next case — usually not what you want!

5.5 The for Loop

Repeat something a known number of times:

```
// Blink an LED exactly 5 times
for (int i = 0; i < 5; i++) {
    digitalWrite(13, HIGH);
    delay(200);
    digitalWrite(13, LOW);
    delay(200);
}
```

Anatomy of a for loop:

```
for ( int i = 0 ;  i < 5 ;  i++ )
      ---------    ------   ----
      Start:       Check:   Step:
      i begins     keep     add 1
      at 0         going    each time
                   while
                   i < 5
```

The loop runs when i = 0, 1, 2, 3, 4 — that's 5 total iterations. When i reaches 5, the condition i < 5 is false and the loop stops.

5.6 The while Loop

Repeat as long as a condition stays true — use when you don't know in advance how many times to loop:

```
int buttonState = digitalRead(BUTTON_PIN);

while (buttonState == HIGH) {
    // Keep looping until the button is pressed
    buttonState = digitalRead(BUTTON_PIN);  // Check again
}

Serial.println("Button was pressed!");
```

5.7 The do-while Loop

Like while, but always runs at least once before checking:

```
int reading;

do {
    reading = analogRead(A0);          // Read sensor at least once
    Serial.println(reading);
} while (reading < 100);              // Keep reading until value >= 100

Serial.println("Got a valid reading!");
```

5.8 Controlling Loops: break and continue

```
// break — immediately exit the loop
for (int i = 0; i < 100; i++) {
```

```
    if (i == 5) break;        // Stop when i reaches 5
    Serial.println(i);         // Prints: 0, 1, 2, 3, 4
}

// continue — skip the rest of this iteration, go to next
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) continue; // Skip even numbers
    Serial.println(i);          // Prints: 1, 3, 5, 7, 9
}
```

6. Functions in C++

6.1 Why Functions?

Imagine copying the same 10 lines of code in 5 different places. If you need to change it, you'd have to find and fix all 5 places — and probably miss one.

Functions solve this by giving a block of code a name. Write it once, call it anywhere. Instead of repeating four lines of blink code everywhere, you tuck the detail away inside blinkLED() and simply call it.

6.2 Anatomy of a Function

```
returnType  functionName  ( parameters )  {
    // body of the function
    return value;  // only needed if returnType is not void
}
```

- **returnType:** what kind of value the function gives back (void = nothing)
- **functionName:** any descriptive name you choose
- **parameters:** inputs the function needs (can be empty)
- **body:** the code that runs when the function is called
- **return:** sends a result back to the caller

6.3 void Functions — No Return Value

```
void blinkLED(int pin, int times) {
    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(300);
        digitalWrite(pin, LOW);
        delay(300);
    }
}

void setup() { pinMode(13, OUTPUT); }

void loop() {
    blinkLED(13, 3);   // Blink pin 13 three times
    delay(1000);
    blinkLED(13, 1);   // Blink once
    delay(1000);
}
```

6.4 Functions That Return a Value

```
// This function takes a Celsius temperature and RETURNS Fahrenheit
float celsiusToFahrenheit(float celsius) {
    float fahrenheit = (celsius * 9.0 / 5.0) + 32.0;
    return fahrenheit;   // Send the result back to whoever called this
}

void loop() {
    float tempC = 25.0;
    float tempF = celsiusToFahrenheit(tempC);  // tempF gets the returned value

    Serial.print("Celsius: ");    Serial.print(tempC);
    Serial.print("  Fahrenheit: "); Serial.println(tempF);
```

```
    delay(2000);
}
```

6.5 Function Prototypes

If a function is defined below the place it's first used, C++ needs a "heads-up" called a prototype:

```
// -- Prototypes (declarations at the top) --
float celsiusToFahrenheit(float c);
void blinkLED(int pin, int n);

// -- Required Arduino functions --
void setup() { Serial.begin(9600); }

void loop() {
    float t = celsiusToFahrenheit(30.0);  // Called before defined — OK because of prototype
    blinkLED(13, 2);
}

// -- Function definitions (can appear after loop) --
float celsiusToFahrenheit(float c) {
    return (c * 9.0 / 5.0) + 32.0;
}

void blinkLED(int pin, int n) {
    for (int i = 0; i < n; i++) {
        digitalWrite(pin, HIGH); delay(200);
        digitalWrite(pin, LOW);  delay(200);
    }
}
```

TIP: In the Arduino IDE, .ino files automatically handle basic prototypes — but for larger projects or .cpp files, prototypes are essential.

7. Digital Input and Output

7.1 What Does "Digital" Mean?

In electronics, digital means a signal has only two possible states:

```
HIGH  ------  ~5 Volts  --  Logic 1  --  ON
LOW   ------  ~0 Volts  --  Logic 0  --  OFF
```

You use digital signals to:

- Turn LEDs on and off
- Read button presses (pressed / not pressed)
- Control buzzers and relays

7.2 Configuring a Pin with pinMode()

Before using any pin, tell the Arduino whether it will send (OUTPUT) or receive (INPUT) a signal:

```
void setup() {
    pinMode(13, OUTPUT);          // Pin 13 will send signals
    pinMode(2, INPUT);            // Pin 2 will receive signals
    pinMode(4, INPUT_PULLUP);     // Pin 4 receives, with internal pull-up resistor
}
```

7.3 Digital Output — Controlling an LED

Circuit: Connect an LED and 220Ω resistor to Pin 13 (or use the built-in LED).

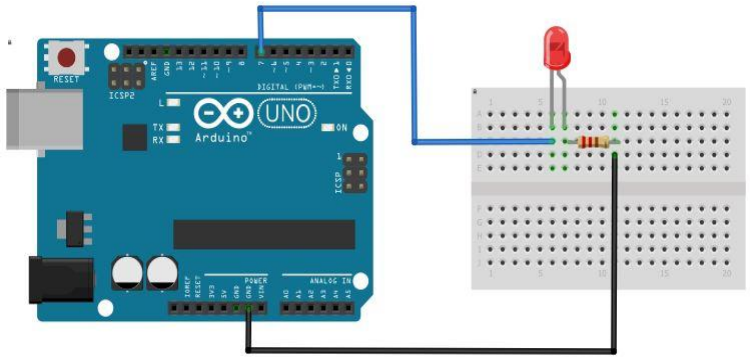


Figure 7.1 — Connecting an external LED to Arduino Pin 7 via a 220Ω resistor

```
Arduino Pin 13  ---- 220 Ohm Resistor ---- LED (+) ---- LED (-) ---- GND
```

```
const int LED_PIN = 7;

void setup() {
    pinMode(LED_PIN, OUTPUT);
}
```

```
void loop() {
    digitalWrite(LED_PIN, HIGH);  // Send 5V -> LED ON
    delay(500);
    digitalWrite(LED_PIN, LOW);   // Send 0V -> LED OFF
    delay(500);
}
```

WARNING: Always use a resistor with an LED! Without it, too much current flows and the LED (or the Arduino pin) burns out. 220Ω to 470Ω is safe.

7.4 Digital Input — Reading a Button

Circuit: Connect a push button between Pin 2 and GND.

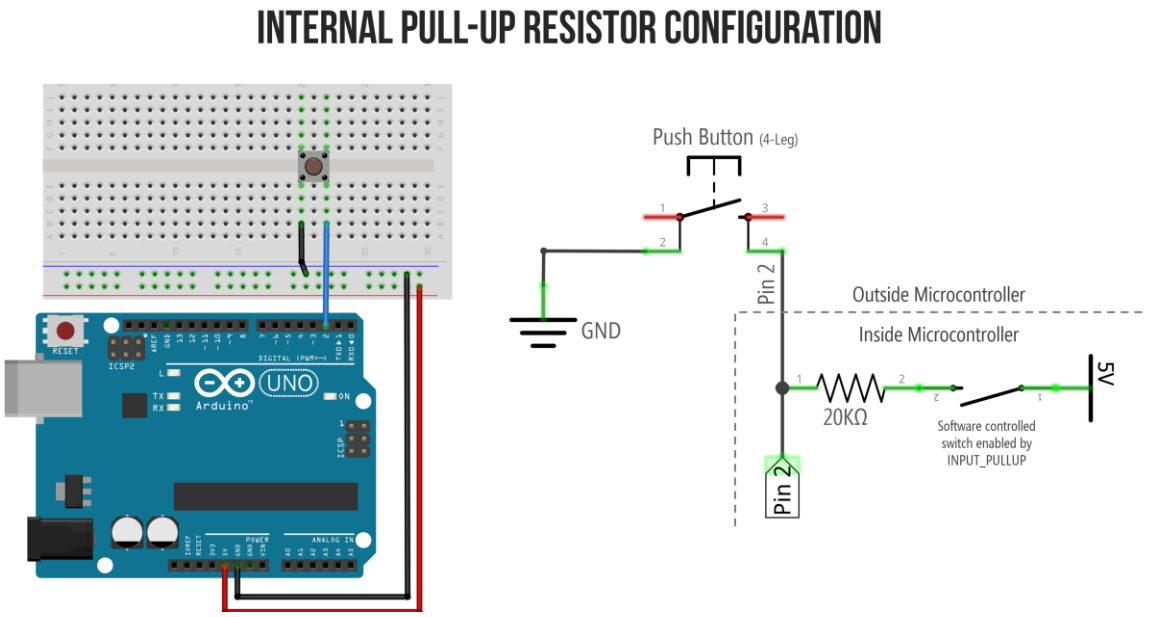


Figure 7.2 — Connecting a push button to Arduino using the internal pull-up resistor

```
const int BUTTON_PIN = 2;
const int LED_PIN = 13;

void setup() {
    pinMode(BUTTON_PIN, INPUT_PULLUP); // Internal pull-up enabled
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    int buttonState = digitalRead(BUTTON_PIN);

    // With INPUT_PULLUP:
    //   Not pressed -> pin reads HIGH (pulled to 5V internally)
    //   Pressed     -> pin reads LOW  (connected to GND via button)

    if (buttonState == LOW) {
        digitalWrite(LED_PIN, HIGH);  // Button pressed -> LED ON
    } else {
        digitalWrite(LED_PIN, LOW);   // Button released -> LED OFF
    }
}
```

TIP: What is a pull-up resistor? Without it, an unconnected input pin "floats" — randomly reading HIGH or LOW and causing erratic behaviour. INPUT_PULLUP internally connects the pin to 5V through a resistor, giving it a stable HIGH reading when nothing is connected.

7.5 Debouncing a Button

When you press a physical button, the metal contacts bounce against each other dozens of times in a few milliseconds. This looks like many rapid button presses to the Arduino. Debouncing filters out these false signals:

```
const int BUTTON_PIN = 2;
const int LED_PIN    = 13;

int   ledState        = LOW;
int   lastButtonState = HIGH;
unsigned long lastDebounceTime  = 0;
const unsigned long DEBOUNCE_MS = 50;  // 50 ms settling time

void setup() {
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    int reading = digitalRead(BUTTON_PIN);

    if (reading != lastButtonState) {
        lastDebounceTime = millis();  // Reset timer when state changes
    }

    if ((millis() - lastDebounceTime) > DEBOUNCE_MS) {
        // Signal has been stable for 50 ms -> it's a real press
        if (reading == LOW) {
            ledState = !ledState;         // Toggle LED
            digitalWrite(LED_PIN, ledState);
        }
    }

    lastButtonState = reading;
}
```

8. Analog Input and Output (PWM)

8.1 Analog vs. Digital Signals

	Digital	Analog
Possible values	Only 0 or 1	Any value in a continuous range
Example	Light switch (ON/OFF)	Volume knob (any level)
Reading on Arduino	digitalRead() -> 0 or 1	analogRead() -> 0 to 1023
Writing on Arduino	digitalWrite() -> HIGH/LOW	analogWrite() -> 0 to 255 (PWM)

8.2 Analog Input — analogRead()

The six analog pins (A0–A5) use a built-in ADC (Analog-to-Digital Converter) to measure voltage:

```
Input Voltage    ADC Output
-------------    ----------
    0V       ->      0
    2.5V     ->     511
    5V       ->    1023
```

The formula to convert back to voltage:

```
int rawValue = analogRead(A0);                  // 0 to 1023
float voltage = rawValue * (5.0 / 1023.0);      // 0.0 to 5.0 V
```

Reading a Potentiometer:

A potentiometer (variable resistor) is a great way to test analog input — turning the knob outputs a varying voltage.

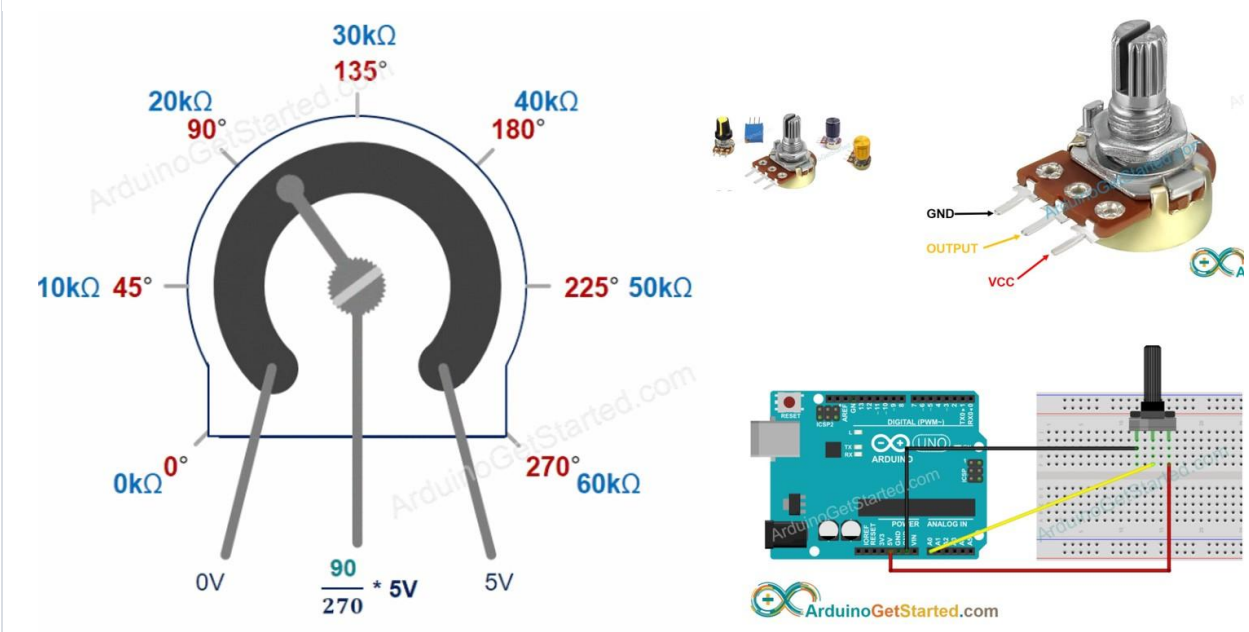


Figure 8.1 — Connecting a potentiometer to analog pin A0

```
5V  ---- Left leg of potentiometer
GND ---- Right leg of potentiometer
```

```
A0  ---- Middle leg (wiper) of potentiometer
```

```
const int POT_PIN = A0;

void setup() {
    Serial.begin(9600);
}

void loop() {
    int potValue = analogRead(POT_PIN);             // 0 to 1023
    float voltage = potValue * (5.0 / 1023.0);      // Convert to volts

    Serial.print("Raw: ");    Serial.print(potValue);
    Serial.print("  |  Voltage: "); Serial.print(voltage, 2);
    Serial.println(" V");
    delay(200);
}
```

8.3 Analog Output — PWM (Pulse Width Modulation)

The Arduino Uno cannot truly output variable voltages — its pins are either 5V or 0V. But it can fake variable voltages using PWM.

PWM works by switching the pin HIGH and LOW very rapidly. By varying the fraction of time spent HIGH (the duty cycle), the average voltage changes:

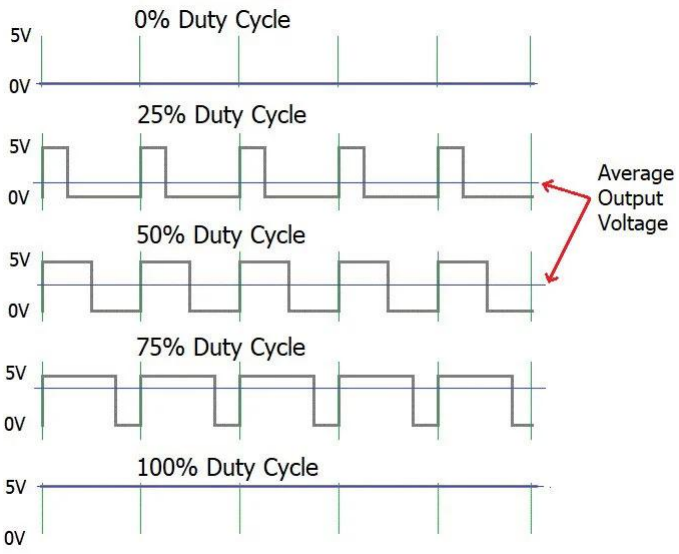


Figure 8.2 — PWM signals with different duty cycles producing different average voltages

```
Duty Cycle = Time spent HIGH / Total Period x 100%

0%   -> Always LOW  -> Average = 0V   (fully OFF)
25%  -> 1/4 time HIGH -> Average = 1.25V
50%  -> 1/2 time HIGH -> Average = 2.5V
75%  -> 3/4 time HIGH -> Average = 3.75V
100% -> Always HIGH -> Average = 5V   (fully ON)
```

PWM Pins on Arduino Uno: Pins 3, 5, 6, 9, 10, 11 (marked with ~ on the board)

```
// analogWrite(pin, value)
// value: 0 = 0% duty cycle (fully off)
//        127 = ~50% duty cycle (half power)
//        255 = 100% duty cycle (fully on)
```

```
analogWrite(9, 0);    // LED fully OFF
analogWrite(9, 64);   // LED at 25% brightness
analogWrite(9, 127);  // LED at ~50% brightness
analogWrite(9, 255);  // LED fully ON
```

8.4 Controlling LED Brightness with a Potentiometer

```
const int POT_PIN = A0;
const int LED_PIN = 9;   // MUST be a PWM pin (~)

void setup() {
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    int potValue   = analogRead(POT_PIN);    // 0 to 1023
    int brightness = potValue / 4;           // Scale to 0-255
    analogWrite(LED_PIN, brightness);
    delay(10);
}
```

8.5 The map() Function — Rescaling Values

map() converts a number from one range to another. Very useful for sensors:

```
// map(value, fromLow, fromHigh, toLow, toHigh)

int pwm     = map(analogRead(A0), 0, 1023, 0, 255);   // ADC -> PWM
int percent = map(analogRead(A1), 0, 1023, 0, 100);   // ADC -> percentage
int angle   = map(analogRead(A2), 0, 1023, 0, 180);   // ADC -> servo angle
```

8.6 The constrain() Function — Clamping Values

Keeps a value within a minimum and maximum:

```
int val = constrain(analogRead(A0), 100, 900);
// If analogRead returns 50  -> val = 100 (clamped to min)
// If analogRead returns 512 -> val = 512 (unchanged)
// If analogRead returns 980 -> val = 900 (clamped to max)
```

9. Serial Communication

9.1 What is Serial Communication?

The Serial port is a channel for your Arduino to talk to other devices — most commonly, your computer's Serial Monitor.

Think of it like a walkie-talkie:

- Arduino can send messages -> you read them in Serial Monitor
- You can type messages in Serial Monitor -> Arduino receives them

The Arduino Uno communicates via:

- TX pin (Pin 1): Transmits (sends) data
- RX pin (Pin 0): Receives data
- These pins are also shared with USB — don't use them as regular I/O while using Serial

9.2 Initializing Serial

```
void setup() {
    Serial.begin(9600);  // Start serial at 9600 baud (bits per second)
}
```

Then open Tools → Serial Monitor in the IDE and set the baud rate to 9600 (must match your code).

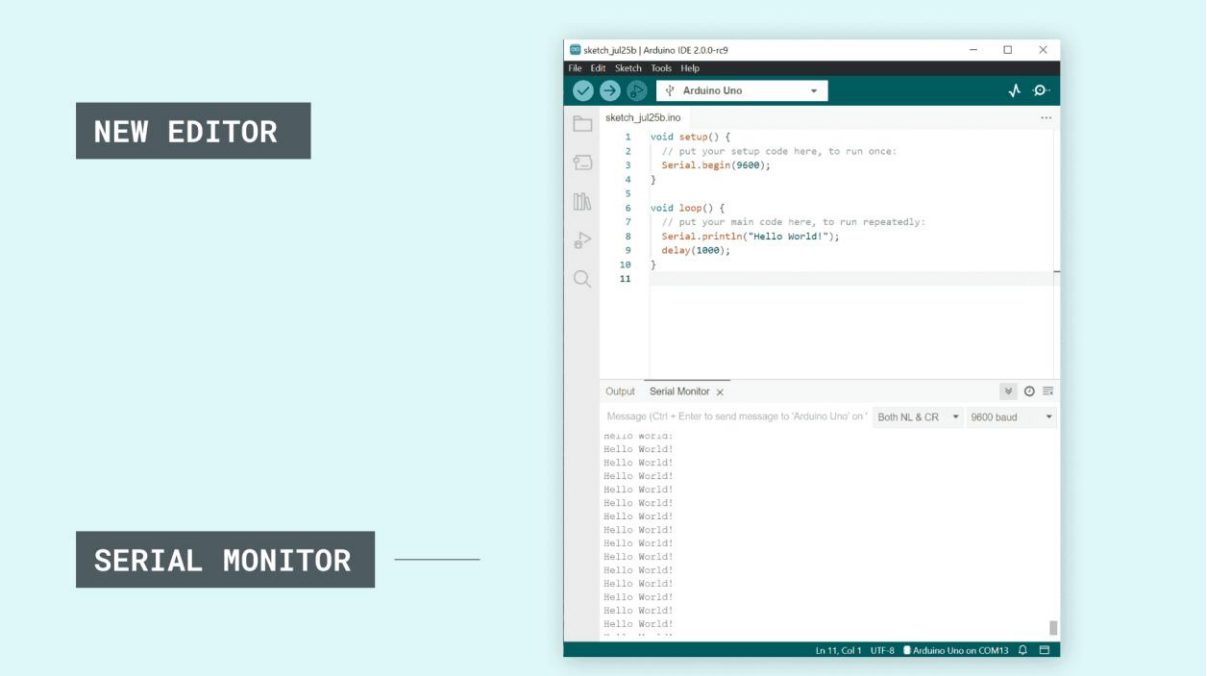


Figure 9.1 — The Arduino Serial Monitor showing output from the board

9.3 Sending Data to Serial Monitor

```
void loop() {
    int x = 42;
```

```
    float temp = 23.5;

    Serial.print("Value: ");      // Print without moving to next line
    Serial.println(x);            // Print AND move to next line -> "Value: 42"

    Serial.print("Temperature: ");
    Serial.print(temp);
    Serial.println(" degrees C"); // -> "Temperature: 23.50 degrees C"

    Serial.println();             // Print a blank line
    delay(1000);
}
```

Formatting numbers:

```
Serial.println(255, BIN);    // 11111111  (binary)
Serial.println(255, HEX);    // FF        (hexadecimal)
Serial.println(3.14159, 2);  // 3.14      (2 decimal places)
Serial.println(3.14159, 4);  // 3.1416    (4 decimal places)
```

9.4 Receiving Input from Serial Monitor

```
void setup() {
    Serial.begin(9600);
    Serial.println("Type a number and press Enter:");
}

void loop() {
    if (Serial.available() > 0) {       // Is there incoming data waiting?
        int received = Serial.parseInt(); // Read one integer
        Serial.print("You sent: ");
        Serial.println(received);
        Serial.print("Doubled: ");
        Serial.println(received * 2);
    }
}
```

9.5 Serial Plotter

The Serial Plotter (Tools → Serial Plotter) draws a live graph of values you print. Very useful for visualizing sensor readings:

```
void loop() {
    int sensorVal = analogRead(A0);
    Serial.println(sensorVal);  // Serial Plotter reads this as a data point
    delay(50);
}
```

10. Arrays and Strings

10.1 What is an Array?

An array is a list of values of the same type stored under a single name:

```
int scores[5]  ->  [ 85, 72, 91, 68, 78 ]
                    [0]  [1]  [2]  [3]  [4]
                     ^ Index starts at 0, not 1!
```

```
int scores[5] = {85, 72, 91, 68, 78};  // Declare and initialize

Serial.println(scores[0]);  // 85  (first element)
Serial.println(scores[2]);  // 91  (third element)
Serial.println(scores[4]);  // 78  (last element)

scores[1] = 95;             // Change second element to 95
```

WARNING: Off-by-one errors are common. An array of size 5 has indices 0, 1, 2, 3, 4. Accessing scores[5] reads memory outside the array — a serious bug!

10.2 Looping Through an Array

```
int ledPins[] = {2, 5, 8, 11, 13};   // 5 LEDs
int numLEDs = sizeof(ledPins) / sizeof(ledPins[0]);  // Get the count

// Set all as outputs
void setup() {
    for (int i = 0; i < numLEDs; i++) {
        pinMode(ledPins[i], OUTPUT);
    }
}

// Chase effect — one LED at a time
void loop() {
    for (int i = 0; i < numLEDs; i++) {
        digitalWrite(ledPins[i], HIGH);   // Turn this one ON
        delay(100);
        digitalWrite(ledPins[i], LOW);    // Turn it OFF before next
    }
}
```

10.3 2D Arrays

A 2D array is a table of values — rows and columns:

```
int matrix[2][3] = {
    {1, 2, 3},   // Row 0
    {4, 5, 6}    // Row 1
};

Serial.println(matrix[0][2]);  // Row 0, Column 2 -> 3
Serial.println(matrix[1][1]);  // Row 1, Column 1 -> 5
```

10.4 The Arduino String Class

The String class makes working with text easy:

```
String greeting = "Hello";
String name     = "Engineer";
String message  = greeting + ", " + name + "!"; // Concatenation
```

```
Serial.println(message);               // Hello, Engineer!
Serial.println(message.length());      // 17
Serial.println(message.toUpperCase()); // HELLO, ENGINEER!

// Check if the message contains a word
if (message.indexOf("Engineer") >= 0) {
    Serial.println("Found 'Engineer' in the message");
}

// Convert a number to string
int sensorVal = 512;
String output = "Sensor: " + String(sensorVal);
Serial.println(output);  // Sensor: 512
```

WARNING: Memory note: Heavy use of the String class can fragment Arduino's tiny 2KB SRAM. For advanced, memory-critical projects, use C-style char arrays instead.

11. Sensors and Actuators

11.1 The Sensor-Actuator Concept

Every embedded system follows this basic pattern:

```
Physical World
      |  (light, temperature, motion, etc.)
      v
  [ SENSOR ]  --> reads physical phenomenon --> produces electrical signal
      |
      v
  [ ARDUINO ] --> processes the signal --> makes a decision
      |
      v
  [ ACTUATOR ] --> produces a physical effect
      |
      v
Physical World (light, motion, sound, etc.)
```

11.2 Temperature Sensor — LM35

The LM35 is a simple 3-pin sensor that outputs 10 millivolts per degree Celsius. At 25°C its output is 0.25V, so analogRead() gives about 51.

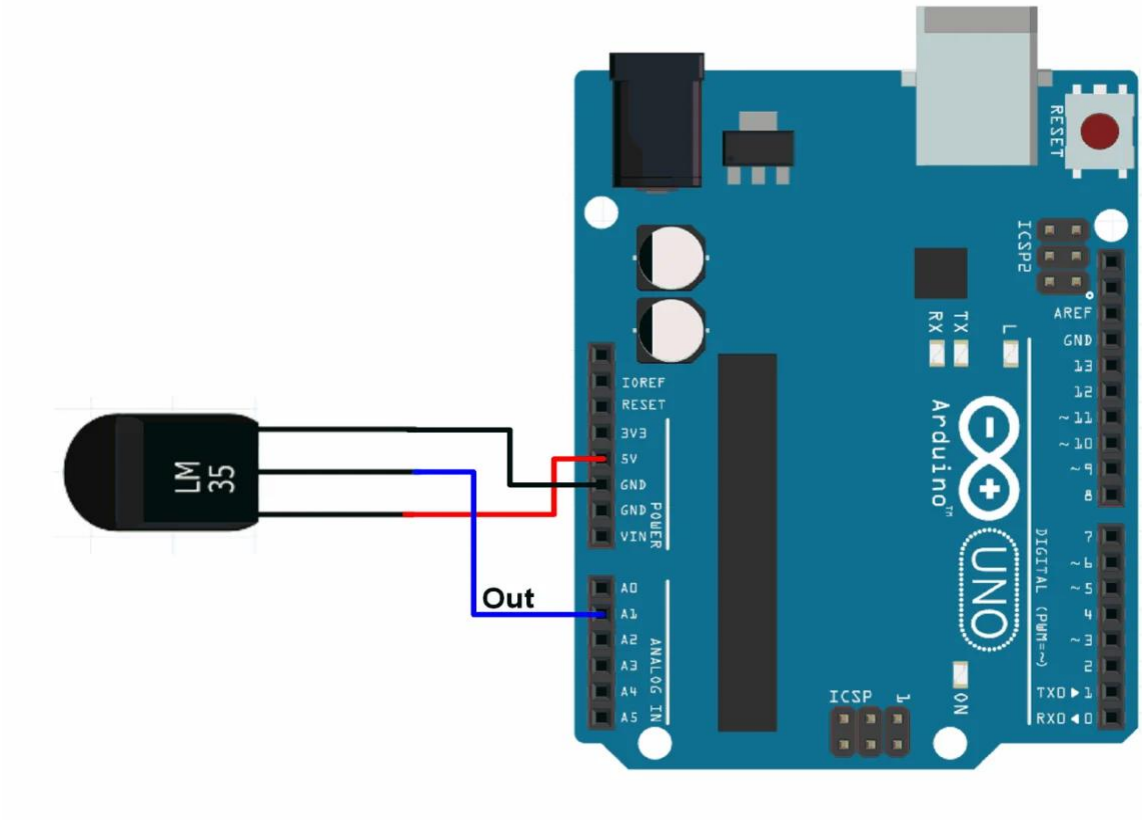


Figure 11.1 — LM35 pinout: +VS (power), VOUT (signal to A1), GND

```
LM35 Pin 1 (VS)   ---- 5V
LM35 Pin 2 (Vout) ---- A0
```

```
LM35 Pin 3 (GND)  ---- GND
```

```
const int TEMP_PIN = A1;

void setup() {
    Serial.begin(9600);
}

void loop() {
    int raw         = analogRead(TEMP_PIN);
    float voltage   = raw * (5.0 / 1023.0);    // Convert raw -> volts
    float tempC     = voltage * 100.0;          // 10mV/C -> x100
    float tempF     = (tempC * 9.0 / 5.0) + 32.0;

    Serial.print("Temp: ");
    Serial.print(tempC, 1);
    Serial.print(" C  |  ");
    Serial.print(tempF, 1);
    Serial.println(" F");
    delay(1000);
}
```

11.3 Ultrasonic Distance Sensor — HC-SR04

The HC-SR04 measures distance by sending a burst of ultrasound and timing the echo. The TRIG pin sends a pulse, the ultrasound travels until it hits an object, the echo returns, and the ECHO pin times how long the echo takes. Distance = (time × speed of sound) ÷ 2 — divided by 2 because the sound travels to the object AND back.

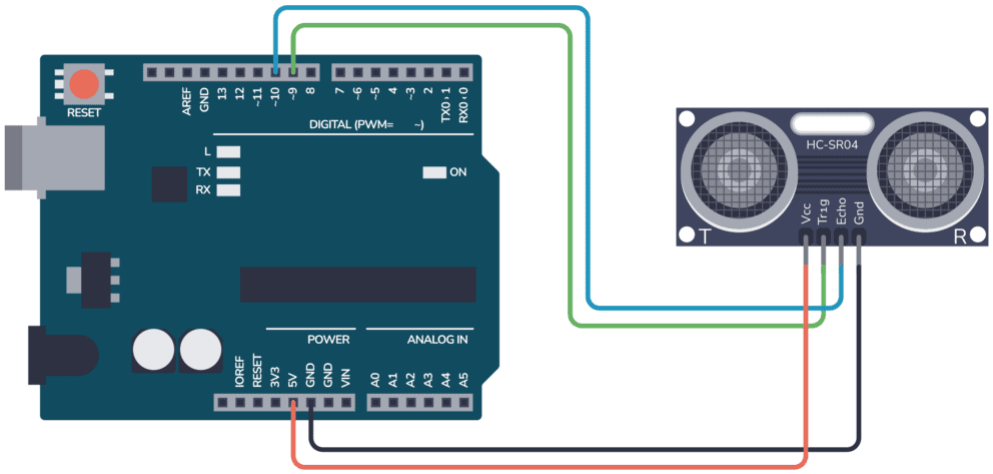


Figure 11.2 — HC-SR04 pinout and wiring to Arduino

```
HC-SR04 VCC  ---- 5V
HC-SR04 TRIG ---- Pin 9  (output from Arduino)
HC-SR04 ECHO ---- Pin 10 (input to Arduino)
HC-SR04 GND  ---- GND
```

```
const int TRIG_PIN = 9;
const int ECHO_PIN = 10;

void setup() {
    Serial.begin(9600);
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
}
```

```
long measureDistanceCm() {
    // Step 1: Send a 10-microsecond trigger pulse
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Step 2: Measure how long the echo pin stays HIGH
    long duration = pulseIn(ECHO_PIN, HIGH);  // time in microseconds

    // Step 3: Convert time to distance
    // Speed of sound ~ 0.034 cm/us
    // Distance = (duration x 0.034) / 2
    return duration * 0.034 / 2;
}

void loop() {
    long dist = measureDistanceCm();
    Serial.print("Distance: ");
    Serial.print(dist);
    Serial.println(" cm");
    delay(500);
}
```

11.4 Photoresistor (LDR) — Detecting Light

A Light Dependent Resistor (LDR) changes its resistance based on light. Bright light gives low resistance, more voltage at A0, and a high reading. Darkness gives high resistance, less voltage at A0, and a low reading.

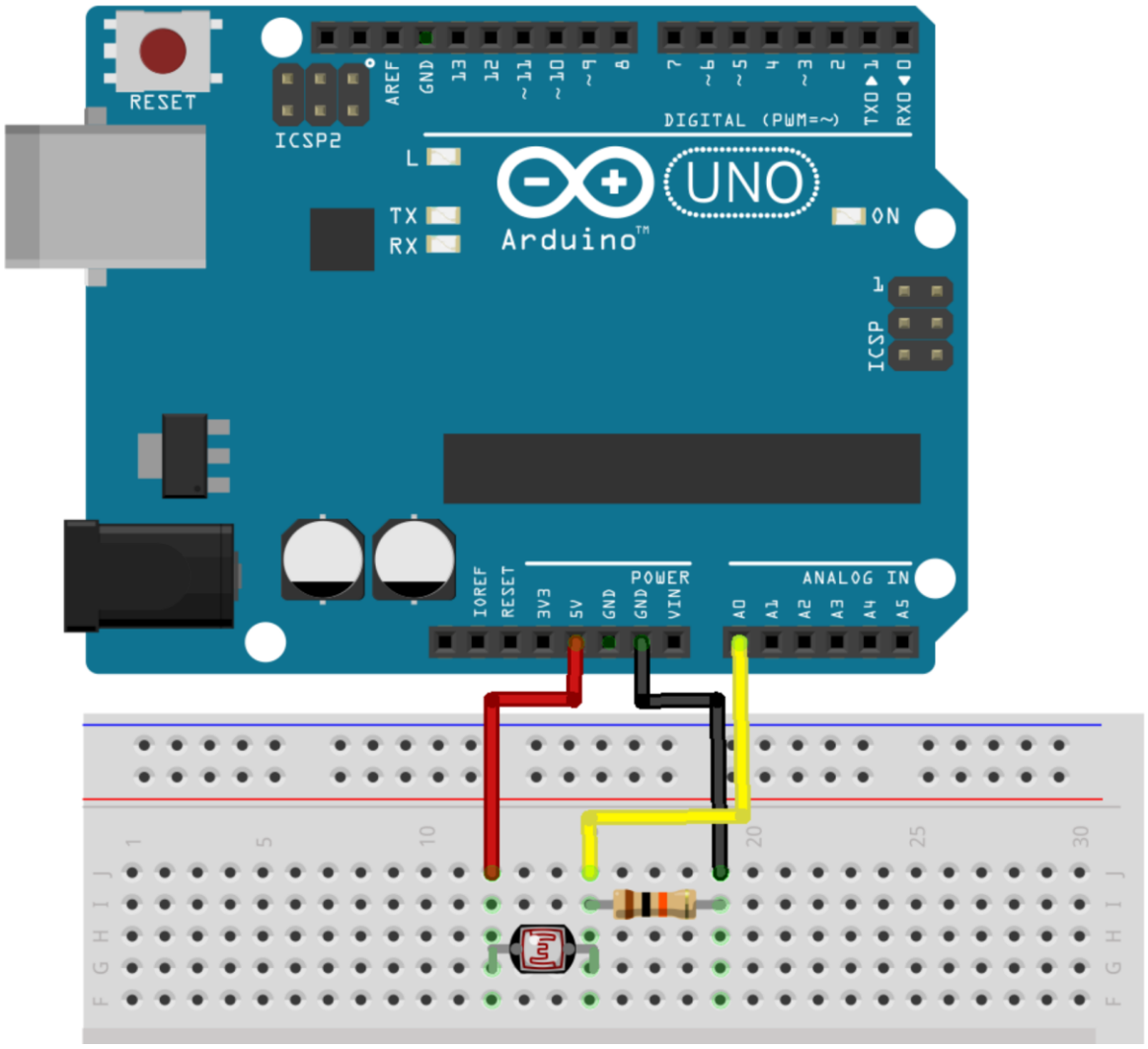


Figure 11.3 — LDR in a voltage divider circuit with a 10kΩ resistor

```
5V ---- 10k Ohm resistor --+-- LDR ---- GND
                           |
                           A0
```

```
const int LDR_PIN = A0;
const int LED_PIN = 9;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int light = analogRead(LDR_PIN);   // 0 (dark) to 1023 (bright)

    // Invert: when dark -> LED brighter
    int brightness = map(light, 0, 1023, 255, 0);
    analogWrite(LED_PIN, brightness);

    Serial.print("Light: "); Serial.print(light);
    Serial.print(" | LED brightness: "); Serial.println(brightness);
    delay(200);
}
```

11.5 Servo Motor

A servo motor rotates to a specific angle (0° to 180°). It has a built-in controller — you just tell it the angle.

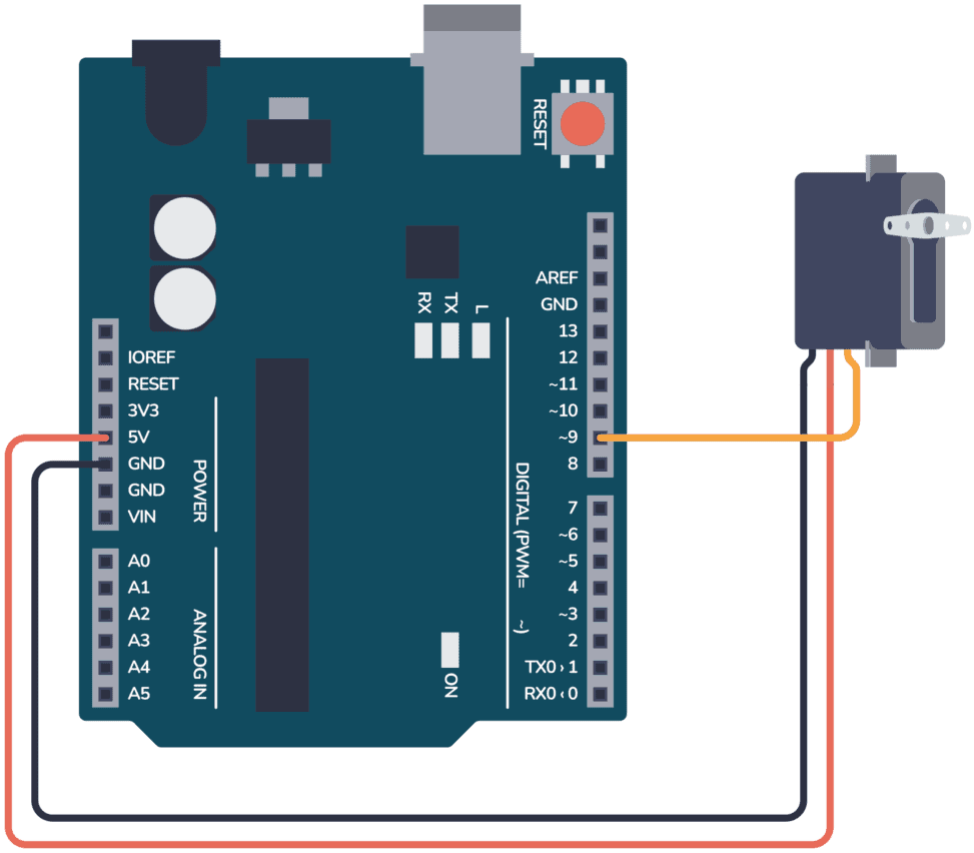


Figure 11.4 — SG90 servo motor wired to Arduino (Brown=GND, Red=5V, Orange=Signal)

```
Servo Brown wire  ---- GND
Servo Red wire    ---- 5V
Servo Orange wire ---- Pin 9
```

```
#include <Servo.h>

Servo myServo;

void setup() {
    myServo.attach(9);  // Connect servo to pin 9
}

void loop() {
    // Sweep from 0 to 180 degrees
    for (int angle = 0; angle <= 180; angle += 5) {
        myServo.write(angle);
        delay(50);
    }

    // Sweep back from 180 to 0 degrees
    for (int angle = 180; angle >= 0; angle -= 5) {
        myServo.write(angle);
        delay(50);
    }
}
```

11.6 DC Motor with L298N Driver

DC motors need more current than Arduino pins can provide. The L298N motor driver module acts as a middle man, using a separate power supply to drive the motor while Arduino controls the direction and speed:

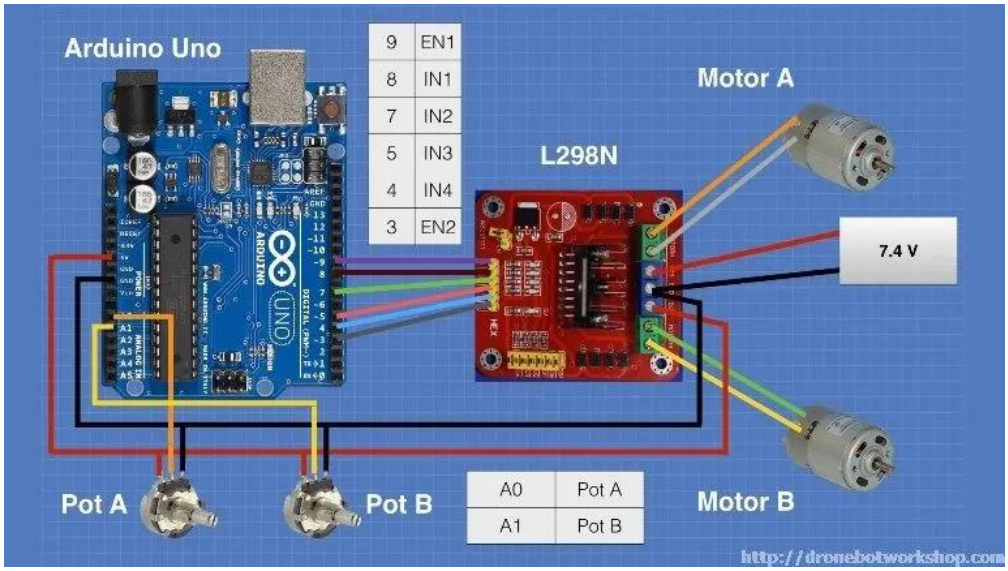


Figure 11.5 — L298N motor driver module connections to Arduino and motor

```
L298N ENA  ---- Arduino Pin 9 (PWM, controls speed)
L298N IN1  ---- Arduino Pin 7 (direction)
L298N IN2  ---- Arduino Pin 6 (direction)
L298N 12V  ---- External power supply (6-12V)
L298N GND  ---- Common GND (Arduino + battery)
L298N OUT1 & OUT2 ---- DC Motor terminals
```

```
const int ENA = 9;  // Speed (PWM)
const int IN1 = 7;  // Direction pin 1
const int IN2 = 6;  // Direction pin 2

void setup() {
    pinMode(ENA, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
}

void motorForward(int speed) {    // speed: 0-255
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
    analogWrite(ENA, speed);
}

void motorBackward(int speed) {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
    analogWrite(ENA, speed);
}

void motorStop() {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    analogWrite(ENA, 0);
}
```

```
void loop() {
    motorForward(200);  delay(2000);
    motorStop();        delay(500);
    motorBackward(150); delay(2000);
    motorStop();        delay(500);
}
```

12. Timing and Interrupts

12.1 The Problem with delay()

delay(1000) makes the Arduino pause for 1 second. But during that pause, the Arduino is completely frozen — it cannot read sensors, respond to buttons, or update anything.

This is fine for simple programs. For real projects, it creates problems:

```
With delay():                  Without delay() (using millis()):
----------------------------   ---------------------------------
LED blinks            OK       LED blinks            OK
Button NOT responding FAIL     Button responding     OK
Sensor NOT reading    FAIL     Sensor reading        OK
```

12.2 Non-Blocking Timing with millis()

millis() returns the number of milliseconds since the Arduino was last reset. Think of it as a clock that never stops. Instead of waiting, you check if enough time has passed:

```
unsigned long previousMillis = 0;    // When we last did something
const long INTERVAL = 1000;          // How often to do it (ms)
int ledState = LOW;

void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    unsigned long currentMillis = millis();  // What time is it now?

    // Has 1000ms passed since the last blink?
    if (currentMillis - previousMillis >= INTERVAL) {
        previousMillis = currentMillis;      // Save the current time
        ledState = !ledState;                // Toggle LED
        digitalWrite(13, ledState);
    }

    // This runs EVERY iteration of loop — not frozen!
    // You can read sensors, check buttons, update displays here.
}
```

12.3 Interrupts — Responding Instantly to Events

An interrupt is a signal that stops whatever the Arduino is doing and immediately runs a special function (called an ISR — Interrupt Service Routine). When the ISR finishes, Arduino continues from where it left off.

Think of it like a fire alarm: no matter what you're doing, when the alarm goes off, you stop and respond immediately.

External Interrupt Pins on Arduino Uno: Pin 2 and Pin 3 only

```
volatile int buttonCount = 0;  // 'volatile' is required for variables
                               // modified inside an ISR

void countPress() {
    // ISR — called instantly when the button is pressed
    // Keep ISRs SHORT — no delay(), no Serial.println()
    buttonCount++;
}
```

```
void setup() {
    Serial.begin(9600);
    pinMode(2, INPUT_PULLUP);

    // attachInterrupt(interrupt_number, function_to_call, trigger)
    attachInterrupt(digitalPinToInterrupt(2), countPress, FALLING);
    //  FALLING triggers on HIGH->LOW edge (button pressed)
}

void loop() {
    Serial.print("Button pressed ");
    Serial.print(buttonCount);
    Serial.println(" times");
    delay(500);
}
```

Interrupt trigger modes:

Mode	When it triggers
LOW	While pin is LOW
CHANGE	Any change (LOW->HIGH or HIGH->LOW)
RISING	LOW -> HIGH edge (button released with pullup)
FALLING	HIGH -> LOW edge (button pressed with pullup)

13. Object-Oriented Programming Basics

13.1 Why OOP?

As programs grow, keeping all the code in setup() and loop() becomes messy. Object-Oriented Programming (OOP) helps you organize code by grouping related data and behavior into objects.

Think of objects like real-world things:

Real World	OOP Equivalent
A blueprint for a car	Class
Your specific car	Object (instance)
Car's color, speed, fuel level	Attributes (member variables)
Accelerate, brake, honk	Methods (member functions)
Manufacturing the car	Constructor

13.2 Creating a Class — The LED Example

```
class LED {
  private:
    // Attributes — data this object stores
    int  pin;
    bool state;

  public:
    // Constructor — runs when the object is created
    LED(int p) {
        pin   = p;
        state = false;
        pinMode(pin, OUTPUT);
    }

    // Methods — things this object can do
    void on() {
        state = true;
        digitalWrite(pin, HIGH);
    }

    void off() {
        state = false;
        digitalWrite(pin, LOW);
    }

    void toggle() {
        state = !state;
        digitalWrite(pin, state ? HIGH : LOW);
    }

    bool isOn() {
        return state;
    }

    void blink(int times, int ms) {
        for (int i = 0; i < times; i++) {
            on();  delay(ms);
            off(); delay(ms);
        }
    }
```

```
};
```

13.3 Using the Class

```
// Create two LED objects — each manages its own pin independently
LED redLED(13);    // LED on pin 13
LED greenLED(12);  // LED on pin 12

void setup() {
    Serial.begin(9600);
}

void loop() {
    redLED.on();      greenLED.off();    delay(500);
    redLED.off();     greenLED.on();     delay(500);

    redLED.blink(3, 200);  // Blink red 3 times, 200ms each

    Serial.print("Red is: ");
    Serial.println(redLED.isOn() ? "ON" : "OFF");
}
```

Key benefits demonstrated:

- redLED and greenLED are independent — they have their own pin and state
- The internal details are hidden — you just call .on(), .off(), .blink()
- Reusable — want a third LED? Just write LED blueLED(11);

13.4 private vs public

```
class LED {
  private:
    int pin;     // Cannot access from outside: redLED.pin = 5; -> ERROR
    bool state;  // Same

  public:
    void on();   // Can call from outside: redLED.on();
    bool isOn(); // Can call: redLED.isOn();
};
```

- **private:** Data hidden inside the class. Only the class's own methods can access it.
- **public:** The interface the outside world uses.

This is called encapsulation — hiding internal details and exposing only what's needed.

14. Debugging and Best Practices

14.1 Common Errors and Their Fixes

Symptom	Likely Cause	What to Do
Code won't compile	Syntax error (missing ; }, misspelled name)	Read the error message — it tells you the line number
Unexpected behaviour	Logic error	Add Serial.println() to trace values at each step
LED not lighting	Missing resistor, wrong pin, reversed polarity	Check circuit, confirm pin number in code
Serial Monitor shows garbage	Wrong baud rate	Match Serial.begin(XXXX) with Monitor's rate setting
Motor doesn't move	Not enough current from Arduino pin	Use motor driver (L298N) with external power
Button doesn't respond	Floating input (no pull-up/down)	Use INPUT_PULLUP or add external resistor
programmer is not responding	Wrong port or board selected	Check Tools -> Board and Tools -> Port
Values jump erratically	Sensor wiring issue / noise	Check connections; add delay() between reads

14.2 Using Serial for Debugging

The most powerful debugging tool is Serial.println(). When something isn't working, print the values at key points:

```
void loop() {
    int raw  = analogRead(A0);
    float v  = raw * (5.0 / 1023.0);
    float t  = v * 100.0;

    // Debug prints — remove or comment out before final deployment
    Serial.print("DEBUG -> raw=");   Serial.print(raw);
    Serial.print("  v=");           Serial.print(v, 3);
    Serial.print("  tempC=");       Serial.println(t, 1);

    if (t > 35.0) {
        Serial.println("DEBUG -> Activating alarm!");
        // trigger alarm...
    }
    delay(500);
}
```

14.3 Code Quality Best Practices

```
// BAD code — hard to understand and maintain
int x = analogRead(A0);
if (x > 800) {
    digitalWrite(4, 1);
    digitalWrite(8, 1);
}

// GOOD code — self-documenting
const int TEMP_SENSOR_PIN = A0;
const int FAN_PIN         = 4;
const int ALARM_LED_PIN   = 8;
const int OVERHEAT_THRESHOLD = 800;  // Raw ADC value (~39C)
```

```
int temperatureRaw = analogRead(TEMP_SENSOR_PIN);

if (temperatureRaw > OVERHEAT_THRESHOLD) {
    digitalWrite(FAN_PIN, HIGH);       // Activate cooling fan
    digitalWrite(ALARM_LED_PIN, HIGH); // Turn on warning LED
}
```

Rules to live by:

13. Use descriptive names. temperatureSensor not ts. buttonPressed not b.
14. Name all magic numbers. DEBOUNCE_DELAY = 50 not just 50.
15. One function = one job. If your function does three things, split it into three functions.
16. Comment the why, not the what. Code shows what is happening — comments explain why.

14.4 Memory Management on Arduino Uno

The Uno has only 2KB of SRAM for runtime variables. Beginners often run out without realising:

```
// Store constant strings in Flash (program memory), not SRAM
Serial.println(F("This text lives in Flash, not SRAM!"));

// Use the smallest correct data type
byte pinNumber  = 13;         // Not int (which wastes 1 extra byte)
uint8_t counter = 0;          // 0-255, unsigned 8-bit

// Avoid many large String objects — they fragment SRAM
String s1 = "Hello";    // Risky in complex programs
String s2 = s1 + " World";

// Prefer character arrays for performance-critical code
char greeting[] = "Hello World";
```

14.5 Circuit Safety Rules

Before you build any circuit, memorise these rules:

17. Always use a resistor with an LED — 220Ω to 470Ω is typical. Without one, the LED and/or pin burns out immediately.
18. Max 40mA per pin — If a component needs more, use a transistor or driver module.
19. Check polarity before powering on — LEDs, electrolytic capacitors, and diodes will be damaged if connected backwards.
20. Disconnect USB before wiring — Make changes to circuits without power connected.
21. Never wire 5V to 3.3V components directly — Many sensors and modules run at 3.3V. Connecting them to 5V pins destroys them. Use a level shifter.
22. Never connect a motor directly to an Arduino pin — Always use a motor driver (L298N or similar).
23. Double-check GND connections — All components must share a common ground with the Arduino.

15. Mini Projects and Practicals

Project 1: Traffic Light System

What you'll learn: Digital output, sequencing, functions

Components: 3 LEDs (red, yellow, green), 3x 220Ω resistors, breadboard

```
Wiring:
Pin 11 ---- 220 Ohm ---- Red LED (+)    ---- GND
Pin 10 ---- 220 Ohm ---- Yellow LED (+) ---- GND
Pin  9 ---- 220 Ohm ---- Green LED (+)  ---- GND
```

```
const int RED    = 11;
const int YELLOW = 10;
const int GREEN  = 9;

void allOff() {
    digitalWrite(RED, LOW);
    digitalWrite(YELLOW, LOW);
    digitalWrite(GREEN, LOW);
}

void setup() {
    pinMode(RED, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(GREEN, OUTPUT);
}

void loop() {
    allOff(); digitalWrite(RED, HIGH);    delay(4000); // Stop:    4s
    allOff(); digitalWrite(YELLOW, HIGH); delay(1000); // Caution: 1s
    allOff(); digitalWrite(GREEN, HIGH);  delay(4000); // Go:      4s
    allOff(); digitalWrite(YELLOW, HIGH); delay(1000); // Caution: 1s
}
```

TIP: Extension: Add a pedestrian push button on Pin 2. When pressed, interrupt the green phase and go straight to yellow -> red.

Project 2: Smart Night Light (Auto-Dimming)

What you'll learn: Analog input, PWM output, map()

Components: LDR, 10kΩ resistor, LED, 220Ω resistor

```
5V ---- 10k Ohm --+-- LDR ---- GND
                  A0

Pin 9 ---- 220 Ohm ---- LED (+) ---- GND
```

```
const int LDR_PIN  = A0;
const int LED_PIN  = 9;   // PWM pin
const int DARK_THRESHOLD = 400;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int lightLevel = analogRead(LDR_PIN);
    int brightness;

    if (lightLevel < DARK_THRESHOLD) {
```

```
        // Darker room -> brighter LED (inversely proportional)
        brightness = map(lightLevel, 0, DARK_THRESHOLD, 255, 0);
    } else {
        brightness = 0;  // Room is bright -> LED off
    }

    analogWrite(LED_PIN, brightness);

    Serial.print("Light: "); Serial.print(lightLevel);
    Serial.print("  LED: "); Serial.println(brightness);
    delay(100);
}
```

Project 3: Digital Thermometer

What you'll learn: Analog sensors, math, rolling averages, conditional alerts

Components: LM35 sensor

```
const int TEMP_PIN   = A0;
const float HIGH_TEMP_ALERT = 35.0;  // C

float readings[5] = {0, 0, 0, 0, 0};  // Buffer for rolling average
int   readIndex   = 0;

void setup() {
    Serial.begin(9600);
    Serial.println("=== Digital Thermometer ===");
    Serial.println("Time(s) | Temp(C) | Temp(F) | Avg5(C)");
}

float readTempC() {
    int raw     = analogRead(TEMP_PIN);
    float volts = raw * (5.0 / 1023.0);
    return volts * 100.0;
}

float rollingAverage() {
    float sum = 0;
    for (int i = 0; i < 5; i++) sum += readings[i];
    return sum / 5.0;
}

void loop() {
    float tempC = readTempC();
    float tempF = (tempC * 9.0 / 5.0) + 32.0;

    readings[readIndex] = tempC;
    readIndex = (readIndex + 1) % 5;  // Wrap index 0->1->2->3->4->0

    float avg = rollingAverage();

    Serial.print(millis() / 1000); Serial.print("s  |  ");
    Serial.print(tempC, 1);        Serial.print("C  |  ");
    Serial.print(tempF, 1);        Serial.print("F  |  ");
    Serial.print(avg, 1);          Serial.println("C");

    if (avg > HIGH_TEMP_ALERT) {
        Serial.println("WARNING: Temperature too high!");
    }

    delay(1000);
}
```

Project 4: Ultrasonic Parking Sensor

What you'll learn: Ultrasonic sensor, buzzer, conditional beeping, functions

Components: HC-SR04, buzzer, LED

```
HC-SR04 TRIG ---- Pin 9     Buzzer (+) ---- Pin 8
HC-SR04 ECHO ---- Pin 10    Buzzer (-) ---- GND
HC-SR04 VCC  ---- 5V        LED (+) ---- 220 Ohm ---- Pin 13
HC-SR04 GND  ---- GND       LED (-) ---- GND
```

```
const int TRIG   = 9;
const int ECHO   = 10;
const int BUZZER = 8;
const int LED    = 13;

void setup() {
    Serial.begin(9600);
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);
    pinMode(BUZZER, OUTPUT);
    pinMode(LED, OUTPUT);
}

long getDistanceCm() {
    digitalWrite(TRIG, LOW);  delayMicroseconds(2);
    digitalWrite(TRIG, HIGH); delayMicroseconds(10);
    digitalWrite(TRIG, LOW);
    return pulseIn(ECHO, HIGH) * 0.034 / 2;
}

void beep(int count, int interval_ms) {
    for (int i = 0; i < count; i++) {
        digitalWrite(BUZZER, HIGH);
        digitalWrite(LED, HIGH);
        delay(60);
        digitalWrite(BUZZER, LOW);
        digitalWrite(LED, LOW);
        delay(interval_ms);
    }
}

void loop() {
    long d = getDistanceCm();
    Serial.print("Distance: "); Serial.print(d); Serial.println(" cm");

    if      (d < 10)  beep(3, 50);   // Very close  -> rapid triple beep
    else if (d < 20)  beep(2, 150);  // Close       -> double beep
    else if (d < 40)  beep(1, 400);  // Approaching -> single beep
    // else: no alert (object is far away)

    delay(300);
}
```

Quick Reference Cheat Sheet

Core Arduino Functions

```
// -- Pin Configuration --
pinMode(pin, OUTPUT);           // OUTPUT, INPUT, or INPUT_PULLUP

// -- Digital I/O --
digitalWrite(pin, HIGH);        // Set pin to 5V
digitalWrite(pin, LOW);         // Set pin to 0V
int val = digitalRead(pin);     // Read: returns HIGH (1) or LOW (0)

// -- Analog I/O --
int val = analogRead(A0);       // Read: returns 0 to 1023
analogWrite(pin, 0-255);        // PWM output (pins ~3,5,6,9,10,11 only)

// -- Timing --
delay(ms);                      // Pause for milliseconds (blocking)
delayMicroseconds(us);          // Pause for microseconds (blocking)
unsigned long t = millis();     // Milliseconds since power on
unsigned long t = micros();     // Microseconds since power on

// -- Math Utilities --
map(val, inLow, inHigh, outLow, outHigh);  // Rescale a value
constrain(val, min, max);                   // Clamp to range
abs(x);                                     // Absolute value
min(a, b);   max(a, b);                    // Minimum / maximum
sqrt(x);     pow(base, exp);               // Square root / power
random(min, max);   randomSeed(seed);      // Random numbers

// -- Serial Communication --
Serial.begin(9600);             // Initialize at baud rate
Serial.print("text");           // Print without newline
Serial.println("text");         // Print with newline
int n = Serial.available();     // Bytes waiting to be read
char c = Serial.read();         // Read one byte
int n = Serial.parseInt();      // Read an integer

// -- Pulses --
long dur = pulseIn(pin, HIGH);  // Time a pulse in microseconds

// -- Interrupts --
attachInterrupt(digitalPinToInterrupt(pin), ISR, FALLING);
detachInterrupt(digitalPinToInterrupt(pin));
```

Glossary

Term	Definition
ADC	Analog-to-Digital Converter — measures voltage and converts it to a number (0-1023 on Arduino)
ATmega328P	The microcontroller chip on the Arduino Uno
Baud Rate	Speed of serial communication, measured in bits per second
Breadboard	A solderless prototyping board for building temporary circuits
Debounce	Filtering out the rapid false signals caused by a button's mechanical bounce
Duty Cycle	In PWM, the fraction of time the signal is HIGH — determines average voltage
EEPROM	Electrically Erasable Programmable ROM — stores data that survives power-off
Flash	Program memory on the microcontroller — stores your compiled sketch
GND	Ground — the 0V reference point in a circuit
GPIO	General Purpose Input/Output — the digital pins
IDE	Integrated Development Environment — the Arduino software
ISR	Interrupt Service Routine — the function called when an interrupt fires
LDR	Light Dependent Resistor (photoresistor) — resistance decreases with more light
MCU	Microcontroller Unit — a single chip containing a processor, memory, and I/O
millis()	Arduino function returning milliseconds since last reset
PWM	Pulse Width Modulation — simulates analog output by rapidly switching a pin
Pull-up Resistor	Connects a pin to VCC so it reads HIGH by default; prevents floating
SRAM	Static RAM — volatile memory that holds variables during program execution
UART	Universal Asynchronous Receiver/Transmitter — the serial communication protocol
VCC / 5V	The positive supply voltage on the Arduino Uno
Volatile	A C++ keyword required for variables modified inside an ISR

Recommended Learning Schedule

Week	Lectures	Focus Area
1	1, 2, 3	Setup, Blink, Program Structure
2	4, 5	Variables, Data Types, Control Flow
3	6	Functions
4	7, 8	Digital & Analog I/O + PWM
5	9	Serial Communication
6	10, 11	Arrays, Strings, Sensors
7	12	Timing & Interrupts
8	13	OOP Basics
9	14	Debugging & Best Practices
10	15	Mini Projects

These notes were prepared for absolute beginners in Computer Engineering. For deeper study, consult the official Arduino Reference at Arduino Language Reference

— End of Lecture Notes —