

PROJECT DOCUMENT • STUDENT HANDBOOK • 2026 EDITION

Arduino + Processing

Learn by Example

Basic exercises for high-school students — from the first blinking LED to an automatic irrigation system. Updated edition with learning objectives, vocabulary, 42 new exercises, 36 quiz questions and a full answer key.

7

MODULES

42

EXERCISES

36

QUIZ QUESTIONS

10CODE
CORRECTIONS

PROJECT NUMBER

2024-2-RS01-KA210-SCH-000271542

Erasmus+ KA210 — Small-scale partnerships in school education

How to use this handbook

Each module starts with **learning objectives** and **key vocabulary**, then walks you through theory and hands-on labs with complete, tested code. At the end you'll find the original **student tasks**, new **exercises**, a **self-check quiz** and **reflection questions**. Solutions are in the **answer key** at the back — try first, check later!

Exercise levels

Basic

Intermediate

Advanced

Challenge

Basic — everyone · Intermediate — consolidate the lesson · Advanced — go further · Challenge — open-ended projects.

Exercise types

TYPE	WHAT YOU DO
Predict	Read code and predict what happens before running it
Calculate	Use physics and maths formulas with real numbers
Build	Wire a circuit and make it work
Code	Write or extend a sketch
Debug	Find and fix mistakes in code or circuits
Explain	Explain a concept in your own words
Design	Plan and carry out your own mini-project or investigation

Lab safety rules

 **POWER**
Unplug USB before changing wiring.
Never connect 5V directly to GND. Only low-voltage supplies ($\leq 12\text{ V}$) — never mains.

 **HEAT & WATER**
If something gets hot or smells, disconnect and tell your teacher. Keep electronics away from water; dry your hands.

Contents

F1 First Steps with Arduino

6 exercises · 5 quiz questions

F2	Processing — Real-Time Visualization	6 exercises · 6 quiz questions
P1	Temperature & Humidity — DHT11	6 exercises · 5 quiz questions
P2	Light Sensor — Photoresistor (LDR)	6 exercises · 5 quiz questions
P3	Sonar — Ultrasonic Sensor HC-SR04	6 exercises · 5 quiz questions
P4	Arduino Ohm Meter with Processing Display	6 exercises · 5 quiz questions
P5	Automatic Irrigation System	6 exercises · 5 quiz questions
A	Answer key	
B	Teacher notes & errata	

First Steps with Arduino

Make a microcontroller do something real — in under ten minutes.

Meet the Arduino Uno, learn how a breadboard works, upload your first program and wire an external LED. This is the foundation for every project in the Digital Challenge series.

BY THE END OF THIS MODULE YOU CAN...

1. Identify the main parts of the Arduino Uno board (USB, digital pins, analog pins, GND, 5V, reset button, ATmega328P).
2. Explain how the rows and rails of a breadboard are connected inside.
3. Write, verify and upload a sketch using `setup()`, `loop()`, `pinMode()`, `digitalWrite()` and `delay()`.
4. Wire an LED safely with a current-limiting resistor and calculate a suitable resistor value.
5. Follow a systematic workflow: schematic → simulation → real wiring → upload → test.

KEY VOCABULARY

Microcontroller: A tiny computer on a single chip that reads inputs and controls outputs. On the Uno it is the ATmega328P.

Sketch: The name for an Arduino (or Processing) program.

Digital pin: A pin that is either HIGH (5 V) or LOW (0 V). The Uno has pins 0–13.

Analog pin: A pin (A0–A5) that can measure a voltage between 0 and 5 V as a number 0–1023.

GND (ground): The 0 V reference point. Every circuit needs a path back to GND.

Anode / Cathode: The + (longer leg) and – (shorter leg, flat side) of an LED.

Resistor: A component that limits current. Measured in ohms (Ω).

Breadboard: A reusable board for building circuits without soldering.

Required equipment

COMPONENT	QTY	PURPOSE
Arduino Uno	1	Main development board
USB cable	1	Connection to computer and program upload
Breadboard	1	Quick connection without soldering
LED diode	1-3	Visual indicator of program execution
220 Ω resistor	1-3	Current limiting for the LED
Push button	1	Input for the advanced tasks
Jumper wires	~6	Connecting components
Arduino IDE	1	Writing and uploading programs
Tinkercad or Circuit Designer	1	Circuit simulation and verification

THEORY What is Arduino?

Arduino is a hardware **and** software platform built around a microcontroller. A small board can read signals from sensors, process them, and then control LEDs, motors, relays, displays or other devices.

Thanks to simple wiring, a huge number of examples and extensive library support, Arduino is ideal for beginners as well as for rapid prototyping in school and project work.



MENTAL MODEL

Think of Arduino as the **brain** of your project: it reads **inputs** (sensors, buttons) and controls **outputs** (LEDs, motors, displays).

01

Input

sensor / button

02

Process

your sketch

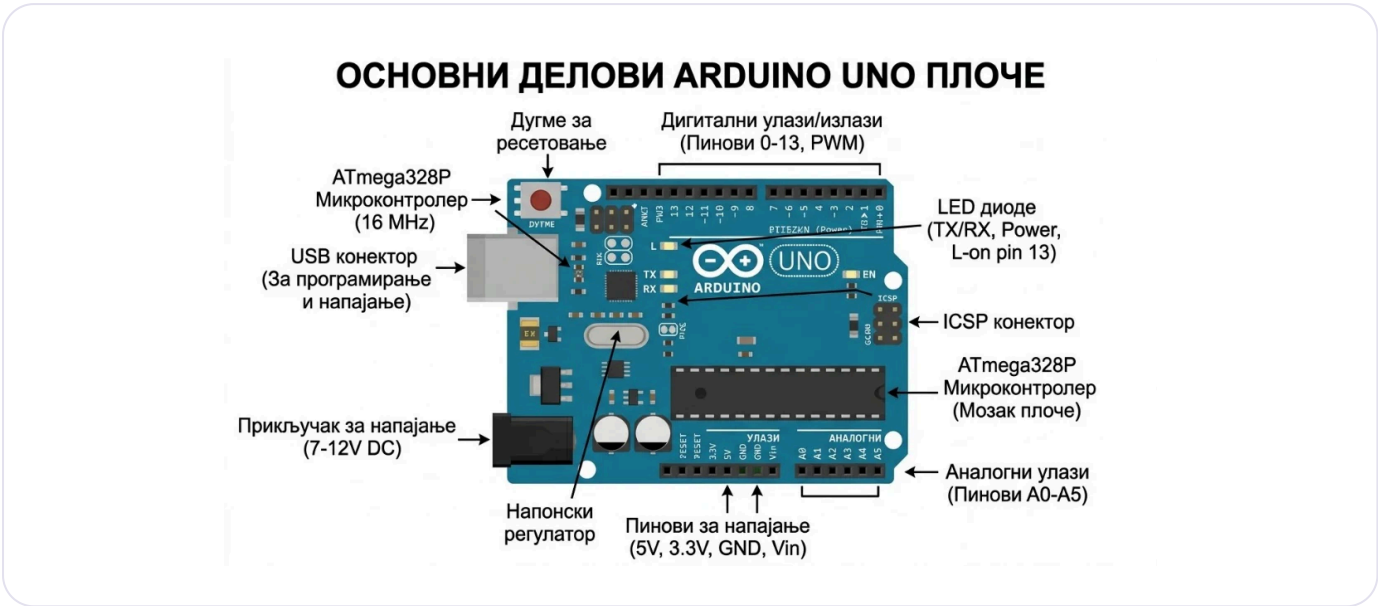
03

Output

LED / motor / screen

Why the Arduino Uno?

The Uno is the most common first board: stable, well documented and easy to use. It has enough input/output pins for school experiments without unnecessary complexity. Once you know the Uno, moving to a Nano, Mega or ESP32 is easy.



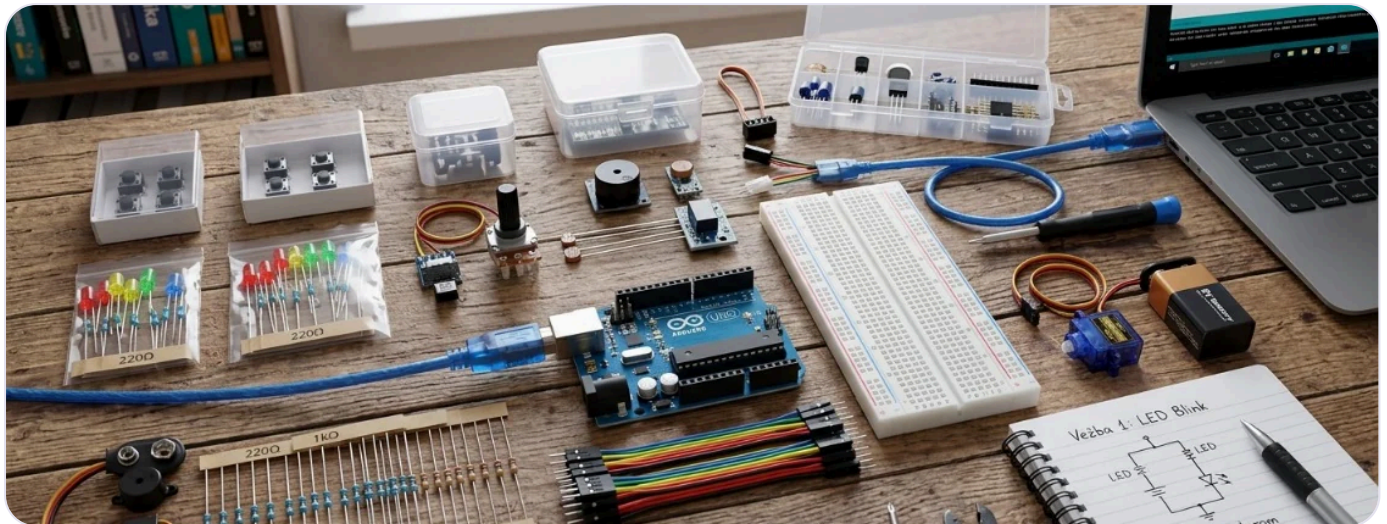
— Main parts of the Arduino Uno board (original handbook figure).

PART	WHAT IT DOES
USB connector	Connects to the computer — power + program upload
Digital pins 0–13	Digital input and output (HIGH / LOW)
Analog pins A0–A5	Read analog sensor values (0–1023)
GND	Ground — the negative reference
5V and 3.3V	Power supply pins for components
Built-in LED	Connected to pin 13 — perfect for the first test
Reset button	Restarts your program from the beginning
ATmega328P	The microcontroller chip — the processor of the board



TIP

Always identify **GND** and the **power pins** before connecting any component.



— A basic electronics starter kit: board, breadboard, wires, resistors, LEDs, buttons and sensors.

A **breadboard** lets you connect components without soldering, so you can change connections, test ideas and re-verify circuits quickly. Inside, each short row of 5 holes is connected; the long **rails** along the edges are usually used for 5V and GND.

A **resistor** limits current. An LED has almost no resistance of its own, so without a resistor it would draw too much current and burn out — or damage the Arduino pin.

LED POLARITY

The **anode** (longer leg) connects to the signal pin through the resistor. The **cathode** (shorter leg, flat side of the rim) connects to GND.

How big should the resistor be?

Use Ohm's law. The resistor "uses up" the voltage the LED doesn't need. With a 5 V pin, a red LED that drops about 2 V, and a safe current of about 15 mA:

FORMULA

$$R = (V_{\text{supply}} - V_{\text{LED}}) / I = (5 \text{ V} - 2 \text{ V}) / 0.015 \text{ A} = 200 \text{ } \Omega$$

Choose the next standard value up: 220 Ω .



SAFETY FIRST

Unplug the USB cable before changing wiring. Never connect 5V directly to GND (a short circuit). If anything gets hot or smells, disconnect immediately and tell your teacher.

TOOLS **Arduino IDE, Tinkercad & Circuit Designer**

Programs for Arduino are usually written in the **Arduino IDE**: this is where code is written, checked for errors and uploaded to the board. **Tinkercad** and **Circuit Designer** let you plan and simulate a circuit before building it for real.

- writing and editing programs (sketches),
- uploading programs to the Arduino board,
- monitoring program execution via the **Serial Monitor**,
- adding libraries for sensors and other components.

The IDE is free on the official Arduino website. After installing, select the correct **board type** and **COM port**, then upload.



BEST PRACTICE

Simulate the circuit in Tinkercad first, then build it physically on the breadboard.

Sensors & actuators you will meet later

SENSOR / ACTUATOR	TYPE	USED IN
DHT11	Temperature & humidity sensor	Project 1
LDR (photoresistor)	Light sensor	Project 2
HC-SR04	Ultrasonic distance sensor	Project 3
SG90 servo motor	Position actuator	Project 3
Relay module	Electrical switch (actuator)	Project 5
Soil moisture sensor	Capacitive sensor	Project 5

HANDS - ON **Lab 1 — First Upload: Blink**

This lab checks that the board is connected, the right COM port is selected and a program uploads successfully. We use the **built-in LED on pin 13**, so no external circuit is needed — the best possible

first test.

Blink.ino

● ARDUINO · C++

```
1 const int ledPin = 13;    // built-in LED
2
3 void setup() {
4   pinMode(ledPin, OUTPUT); // runs once
5 }
6
7 void loop() {             // repeats forever
8   digitalWrite(ledPin, HIGH);
9   delay(1000);
10  digitalWrite(ledPin, LOW);
11  delay(1000);
12 }
```

COMMAND	EXPLANATION
<code>const int ledPin = 13;</code>	Defines the pin number where the LED is connected.
<code>setup()</code>	Runs once at the start of the program.
<code>pinMode(ledPin, OUTPUT);</code>	Sets the pin as an output.
<code>loop()</code>	Repeats indefinitely while the board is powered.
<code>digitalWrite(ledPin, HIGH/LOW);</code>	Turns the LED on (HIGH = 5 V) or off (LOW = 0 V).
<code>delay(1000);</code>	Pauses execution for 1000 milliseconds = 1 second.

How to upload

- 01 Open the Arduino IDE.
- 02 Go to **Tools** → **Board** → **Arduino Uno**.
- 03 Go to **Tools** → **Port** and select the correct COM port.
- 04 Click **Upload** (the → arrow icon).
- 05 The built-in LED should start blinking once per second.

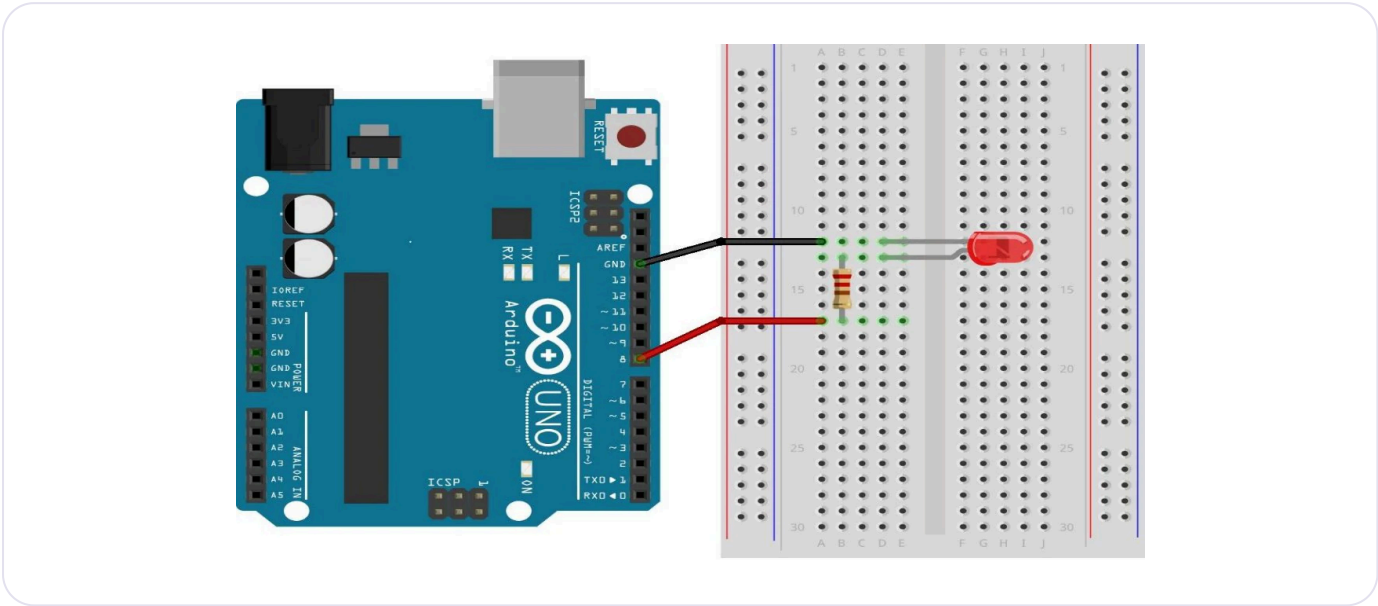


TIP

If the LED blinks — success! The board is connected and working correctly.

HANDS - ON **Lab 2 — An External LED on a Breadboard**

Now move from the built-in LED to an external one and build a real circuit for the first time. Notice how the schematic, the physical wiring and the program connect.



— Physical breadboard wiring: pin 8 → 220 Ω → LED → GND.

ARDUINO UNO	COMPONENT	CONNECTION
Pin 8	LED anode (+)	Through a 220 Ω resistor
GND	LED cathode (–)	Directly to ground

ExternalLED.ino● ARDUINO · C++

```
1 const int ledPin = 8;
2
3 void setup() {
4   pinMode(ledPin, OUTPUT);
5 }
6
7 void loop() {
8   digitalWrite(ledPin, HIGH);
9   delay(500);
10  digitalWrite(ledPin, LOW);
11  delay(500);
12 }
```

The program turns the LED on pin 8 on, waits half a second (500 ms), turns it off, waits again — and repeats forever.



A SYSTEMATIC APPROACH

Schematic → Simulation (Tinkercad) → Real wiring → Upload code → Test. If it doesn't work, check the **physical connections first**, then the code.

Student tasks

LEVEL	TASK	GOAL
Basic	Change the delay value	Understand the effect of pause duration
Intermediate	Move the LED to a different pin	Practise pin assignment in code
Advanced	Add a push button to control the LED	Learn INPUT_PULLUP and digitalRead
Challenge	Create a 3-LED sequential light effect	Multiple outputs, timing logic

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
Upload fails / "port not found"	Wrong COM port or board selected	Tools → Port: pick the port that appears when you plug the board in.
LED never lights	LED inserted backwards	Turn the LED around: long leg towards the resistor/pin.
LED lights but is very dim	Resistor too large (e.g. 10 kΩ)	Check the colour bands: 220 Ω is red-red-brown.
Nothing changes after editing	Code verified but not uploaded	Click Upload, not only Verify.

F1 • Exercises

F1.1 BASIC PREDICT • 5 MIN

Predict the rhythm

Without running it, describe exactly what the LED does with this `loop()`. How many times does it flash per minute?

starter

• ARDUINO • C++

```
1 void loop() {  
2   digitalWrite(8, HIGH);  
3   delay(200);  
4   digitalWrite(8, LOW);  
5   delay(800);  
6 }
```

Hint: Add the two delays to get the length of one full cycle.

F1.2 BASIC CALCULATE • 10 MIN

Choose the resistor

A blue LED needs about **3.0 V** and should run at **10 mA**. The Arduino pin gives **5 V**. Calculate the ideal resistor, then choose a real value from this list: 100 Ω , 150 Ω , 220 Ω , 330 Ω , 1 k Ω .

Hint: $R = (V_{\text{supply}} - V_{\text{LED}}) / I$. Remember 10 mA = 0.010 A. Always round **up** to the next available value.

F1.3 INTERMEDIATE CODE • 15 MIN

SOS beacon

Program the external LED to blink **SOS** in Morse code: three short (200 ms), three long (600 ms), three short — then a 2-second pause. Use a helper function so you don't repeat yourself.

Hint: Write `void flash(int ms)` that turns the LED on for `ms`, then off for 200 ms. Call it inside `for` loops.

F1.4 INTERMEDIATE DEBUG · 10 MIN

Find three bugs

This sketch should blink an LED on pin 8 but it does not even compile — and even when it compiles, the LED stays dark. Find all **three** problems.

starter

● ARDUINO · C++

```
1 const int ledPin = 8
2
3 void setup() {
4   pinMode(ledPin, INPUT);
5 }
6
7 void loop() {
8   digitalWrite(ledPin, HIGH);
9   delay(500);
10  digitalWrite(ledpin, LOW);
11  delay(500);
12 }
```

Hint: Look for a missing character, a wrong mode, and a spelling difference (C++ is case-sensitive).

F1.5 ADVANCED BUILD · 20 MIN

Push-button lamp

Add a push button between **pin 2** and **GND**. The LED on pin 8 should be ON only while the button is held down. Use the internal pull-up resistor.

1. Place the button across the middle gap of the breadboard.
2. Connect one side to pin 2 and the other side to GND.
3. In `setup()` use `pinMode(2, INPUT_PULLUP);`.
4. Remember: with a pull-up, the pin reads **LOW** when pressed.

F1.6 CHALLENGE DESIGN · 30 MIN

Traffic light controller

Build a traffic light with red (pin 10), yellow (pin 9) and green (pin 8) LEDs, each with its own 220 Ω resistor. Sequence: green 4 s → yellow 1 s → red 4 s → red + yellow 1 s → repeat. **Extension:** add a pedestrian button that shortens the green phase.

Hint: Write a helper `void lights(bool r, bool y, bool g, int ms)` that sets all three LEDs and then waits.

Self-check quiz

1. Which function runs only once when the Arduino starts?

- ☐ A loop()
- ☐ B setup()
- ☐ C main()
- ☐ D start()

2. Which leg of an LED connects to GND?

- ☐ A The longer leg (anode)
- ☐ B The shorter leg (cathode)
- ☐ C Either — LEDs work both ways
- ☐ D Neither — it connects to 5V

3. What does `delay(250)` do?

- ☐ A Waits 250 seconds
- ☐ B Waits 0.25 seconds
- ☐ C Blinks 250 times
- ☐ D Sets the brightness to 250

4. Why do we always put a resistor in series with an LED?

- ☐ A To make it brighter
- ☐ B To limit the current so the LED and pin are not damaged
- ☐ C To change its colour
- ☐ D Because the code needs it

5. Which pin is connected to the built-in LED on an Arduino Uno?

- ☐ A Pin 0
- ☐ B Pin 8
- ☐ C Pin 13
- ☐ D A0

Reflection

- Where in your home or school is there a device that probably contains a microcontroller? What are its inputs and outputs?
- Why is it smart to simulate a circuit before building it?
- What was the hardest part of today — the wiring or the code? Why?

CONCLUSION

You took the first step with the Arduino platform. Through Blink and the external LED you saw how a program, electronic components and a physical circuit come together as one system. Reading a schematic, writing `setup()` and `loop()`, using `digitalWrite()` and `delay()` and wiring a breadboard are skills you will use in every following project — from temperature sensors to sonar and automatic irrigation.

Processing — Real-Time Visualization

Turn raw sensor numbers into live graphics on your screen.

Processing is the visual front-end for every project: it receives data from Arduino over USB and draws it as text, bars, colours and animations. Learn the sketch structure, the drawing API, `map()` and the serial pattern used everywhere.

BY THE END OF THIS MODULE YOU CAN...

1. Describe how Arduino and Processing communicate over a serial (USB) connection at the same baud rate.
2. Write active Processing sketches using `setup()` and `draw()`, shapes, colours and text.
3. Use the coordinate system (origin top-left, Y grows downward) to place elements precisely.
4. Scale sensor values to screen sizes with `map()` and format numbers with `nf()`.
5. Receive, validate and parse single values and comma-separated values (CSV) in `serialEvent()`.
6. Send commands from Processing back to Arduino with mouse clicks and key presses.

KEY VOCABULARY

Processing: A free, Java-based language and environment for visual and interactive programs, created at the MIT Media Lab in 2001.

PDE: Processing Development Environment — the editor, console and toolbar where you write sketches.

Serial communication: Sending data one bit after another over a wire — here, the USB cable between Arduino and computer.

Baud rate: Speed of serial communication in bits per second (e.g. 9600). Both sides must match.

COM port: The name your computer gives the Arduino's serial connection (e.g. COM3 or `/dev/ttyUSB0`).

CSV: Comma-Separated Values, e.g. `23.5,61.0` — several numbers in one line of text.

Parsing: Turning text (a String) into usable data (numbers).

Frame: One execution of `draw()`. Processing draws ~60 frames per second.

Required equipment

COMPONENT	QTY	PURPOSE
Processing IDE 4.x	1	Writing and running visualizations — processing.org
Arduino IDE 2.x	1	Programming the Arduino board — arduino.cc
Arduino Uno	1	Sending sensor data over serial
USB cable	1	Connecting Arduino to the computer
Any sensor	1	DHT11, LDR, HC-SR04 ... to generate data

THEORY What is Processing?

Processing is a free, open-source programming environment designed for visual output, animation and interaction. In Arduino projects it is the **visual dashboard**: it cannot control hardware directly, but it is excellent at receiving data and turning numbers into compelling visuals.

FEATURE	ARDUINO IDE	PROCESSING
Primary purpose	Control hardware (sensors, motors)	Create visual output (graphics, charts)
Runs on	Microcontroller (Arduino board)	Computer (Windows, macOS, Linux)
Output	Signals to components	Graphics on screen
Data flow	Sends data via Serial	Receives data via Serial
Language base	C / C++	Java (also Python & JavaScript modes)
Sketch structure	<code>setup()</code> + <code>loop()</code>	<code>setup()</code> + <code>draw()</code>



TIP

Processing is free and runs on Windows, macOS and Linux. Download it from processing.org/download.

ARCHITECTURE How Arduino and Processing Work Together

The two programs talk through **serial communication** over the USB cable. Every project in the series follows the same pipeline:

- 01 A **sensor** reads a physical value (temperature, distance, light ...).
- 02 **Arduino** processes the value and sends it with `Serial.println()`.
- 03 **Processing** receives the text, parses it and converts it to a number.

04 Processing **draws** text, bars, colours or shapes on the screen.



SAME SPEED ON BOTH SIDES

Both programs must use the same baud rate, e.g. 9600. Arduino: `Serial.begin(9600)` —
Processing: `new Serial(this, port, 9600)`.

CORE CONCEPTS **Sketch Structure & Events**

Every Processing program (a **sketch**) has two main functions — just like Arduino — plus optional event functions that run when something happens.

FUNCTION	WHEN IT RUNS	TYPICAL USE
<code>setup()</code>	Once, at program start	<code>size()</code> , serial setup, initial variables
<code>draw()</code>	Continuously (~60 fps)	<code>background()</code> , <code>text()</code> , <code>rect()</code> , shapes
<code>serialEvent()</code>	When new serial data arrives	Read and parse incoming Arduino data
<code>mousePressed()</code>	When a mouse button is clicked	UI buttons, interactions
<code>keyPressed()</code>	When a key is pressed	Send commands back to Arduino

Skeleton.pde

PROCESSING · JAVA

```
1 void setup() {
2   // Runs ONCE at the start
3   // Set window size, initialize serial port
4 }
5
6 void draw() {
7   // Runs CONTINUOUSLY – like loop() in Arduino
8   // Draw shapes, text, update the display
9 }
```

The Serial library

SerialSetup.pde

PROCESSING • JAVA

```
1 import processing.serial.*; // 1. Import the library
2
3 Serial myPort; // 2. Declare the serial port object
4
5 void setup() {
6   size(400, 300);
7   printArray(Serial.list()); // 3. List available ports
8   // 4. Open the port – index [0] is usually the first available
9   myPort = new Serial(this, Serial.list()[0], 9600);
10  myPort.bufferUntil('\n'); // 5. Call serialEvent() once per full line
11 }
```



TIP

If `Serial.list()[0]` is the wrong port, look at the list printed by `printArray(Serial.list())` in the console and change the index.



WARNING

Close the Arduino IDE **Serial Monitor** before running Processing — both cannot use the same port at the same time.

FUNCTION	DESCRIPTION	EXAMPLE
<code>size(w, h)</code>	Set window width and height	<code>size(600, 400)</code>
<code>background(r,g,b)</code>	Fill background with a colour	<code>background(255)</code> — white
<code>fill(r,g,b)</code>	Set fill colour for shapes	<code>fill(255, 0, 0)</code> — red
<code>stroke(r,g,b)</code>	Set outline colour	<code>stroke(0)</code> — black
<code>text(str, x, y)</code>	Draw text at a position	<code>text("Hello", 50, 100)</code>
<code>textSize(n)</code>	Set font size	<code>textSize(20)</code>
<code>rect(x, y, w, h)</code>	Draw a rectangle	<code>rect(50, 100, 200, 30)</code>
<code>ellipse(x, y, w, h)</code>	Draw an ellipse / circle	<code>ellipse(200, 200, 50, 50)</code>
<code>line(x1,y1,x2,y2)</code>	Draw a line	<code>line(0, 0, 200, 200)</code>
<code>map(v,lo1,hi1,lo2,hi2)</code>	Scale a value to a new range	<code>map(temp, 0, 100, 0, 300)</code>
<code>nf(n, d1, d2)</code>	Format a number as text	<code>nf(3.14159, 1, 2)</code> → "3.14"

Colour in Processing

Processing uses **RGB** values from 0 to 255. A fourth value (alpha) sets transparency.

One number means greyscale: `fill(0)` is black, `fill(255)` is white.

```

sketch                                ● PROCESSING · JAVA

1 fill(255, 0, 0);                    // Red
2 fill(0, 255, 0);                    // Green
3 fill(0, 0, 255);                    // Blue
4 fill(255, 255, 0);                  // Yellow
5 fill(128, 0, 128);                  // Purple
6 fill(0);                            // Black
(shorthand)
7 fill(255);                          // White
(shorthand)
8 fill(255, 0, 0, 150);               // Semi-transparent red

```

The coordinate system

The origin (0, 0) is in the **upper-left** corner. X grows to the right; **Y grows downward**. In a 320 × 240 window, (319, 239) is the last visible pixel in the lower-right corner.



NOTE

`rect(50, 100, 200, 30)` starts 100 pixels from the **top** — not the bottom. To draw a bar that grows upward, start at `height - barHeight`.

MATHEMATICS `map()` — the Most Important Function

`map()` rescales a value from one range to another. Sensor values (0–1023) almost never fit your screen directly, so you will use `map()` in **every** project.

FORMULA

$$\text{result} = \text{lo2} + (\text{value} - \text{lo1}) \times (\text{hi2} - \text{lo2}) / (\text{hi1} - \text{lo1})$$

This is a linear function — the same “ $y = kx + n$ ” you know from maths class.

sketch

• PROCESSING • JAVA

```
1 // Sensor value 0-1023 mapped to bar width 0-300 pixels:
2 float barWidth = map(sensorValue, 0, 1023, 0, 300);
3 rect(50, 100, barWidth, 30);
4
5 // Temperature 0-50 °C mapped to the red colour component:
6 float r = map(temperature, 0, 50, 0, 255);
7 fill(r, 0, 0);
```



TIP

`map()` does **not** clamp. If the value goes outside the input range, the result goes outside the output range. Wrap it in `constrain(x, min, max)` when you need limits.

TOOLS The Processing Development Environment

The PDE contains a text editor, a compiler and a display window. It has a message area, a text console, tabs for files, a toolbar and menus. Full reference: processing.org/environment.

COMPONENT	PURPOSE
Text editor	Write and edit sketch code
Message area	Feedback while saving, exporting and errors
Console	Text output from <code>print()</code> and <code>println()</code>
Toolbar	► Run (compile + open window) and ■ Stop
Tabs	Manage multiple files within one sketch
Menus	File, Edit, Sketch, Debug, Tools, Help

Most-used menu commands

MENU	COMMAND	WHAT IT DOES
File	New / Open / Save As...	Create, open, or save a copy of a sketch
File	Examples...	Browse built-in example sketches — great for learning
File	Export Application...	Create a standalone program from your sketch
Edit	Auto Format	Tidy indentation (Ctrl+T)
Edit	Comment / Uncomment	Toggle <code>//</code> on the selected lines
Sketch	Present	Run full-screen (Esc to exit)
Sketch	Tweak	Change numbers and colours live while running
Sketch	Import Library...	Add <code>import</code> lines or install new libraries
Sketch	Add File...	Copy images/fonts into the sketch's <code>data/</code> folder
Debug	Enable Debugger	Breakpoints, Step, Step Into, Step Out
Tools	Color Selector...	Pick colours and copy RGB/HSB/Hex values
Tools	Archive Sketch	Save a .zip copy of the sketch

Sketches and the Sketchbook

Each sketch lives in its own folder; the main file has the same name as the folder with a `.pde` extension. Images, fonts and sounds go into a `data/` sub-folder. Find the Sketchbook location in **File** → **Preferences**.

```
1 MyProject/           ← sketch folder
2 └─ MyProject.pde     ← main file (same name as folder)
3 └─ data/             ← images, fonts, sounds
```

Renderers

RENDERER	DESCRIPTION	BEST FOR
Default	Standard 2D renderer	Most 2D sketches and sensor displays
P2D	Faster OpenGL 2D	High-speed 2D graphics
P3D	3D with camera, lighting, materials	3D visualizations
FX2D	High-quality JavaFX 2D	Large / high-resolution displays

Choose a renderer as the third argument of `size()`, e.g. `size(400, 300, P3D);`.

Static vs. active sketches

Static — no `setup()` / `draw()`; runs once. Good for learning coordinates, shapes and colour.

Active — uses `setup()` + `draw()` for animation, sensors and interaction.

```
1 size(200, 200);
2 background(255);
3 noStroke();
4 fill(255, 204, 0);
5 rect(30, 20, 50, 50);
```

```
1 void setup() {
2   size(200, 200);
3   noStroke();
4   fill(0, 102, 153, 204);
5 }
6 void draw() {
7   background(255);
8   rect(width - mouseX, height - mouseY,
9     50, 50);
10  rect(mouseX, mouseY, 50, 50);
11 }
```



LIBRARIES, TOOLS & MODES

Library: **Sketch** → **Import Library** → **Add Library...** · Tool: **Tools** → **Manage Tools...** · Mode: **Mode menu** → **Add Mode...**. The Serial library is built in — start every project sketch with `import processing.serial.*;`.

HANDS-ON · NO HARDWARE

Lab 1 — Your First Processing Sketch

Before connecting Arduino, get comfortable with the environment: draw a rounded rectangle, some text, and a circle whose colour changes over time.

FirstSketch.pde

PROCESSING · JAVA

```
1 float t = 0;
2
3 void setup() {
4   size(500, 300);
5   textAlign(CENTER, CENTER);
6 }
7
8 void draw() {
9   background(30);           // dark grey background
10  t++;
11
12  // Animated colour using sine waves
13  float r = 127 + 127 * sin(t * 0.05);
14  float g = 127 + 127 * sin(t * 0.03);
15
16  // Rounded rectangle (last argument = corner radius)
17  fill(r, g, 100);
18  rect(50, 80, 400, 60, 10);
19
20  // White text centred in the rectangle
21  fill(255);
22  textSize(22);
23  text("Processing + Arduino = Power!", 250, 110);
24
25  // Animated circle
26  fill(r, 50, 200);
27  ellipse(250, 220, 60, 60);
28 }
```

UPDATED Uses a new variable `t` instead of redeclaring Processing's built-in `frameCount`.

CONCEPT	WHERE IN THE CODE
Window size	<code>size(500, 300)</code> in <code>setup()</code>
Continuous animation	<code>draw()</code> runs ~60 times per second
Dynamic colour using maths	<code>sin()</code> of a growing number
Rectangle	<code>rect(x, y, width, height, radius)</code>
Text	<code>text()</code> , <code>textSize()</code> , <code>textAlign()</code>
Circle	<code>ellipse(cx, cy, width, height)</code>

**TIP**

Press **Ctrl+R** (Cmd+R on Mac) to run, **Ctrl+.** to stop — or use the ► / ■ buttons.

HANDS-ON Lab 2 — Receiving Data from Arduino

Arduino sends a counter (0–100) every 500 ms; Processing shows it as text and as a bar.

Step 1 – Counter.ino (upload first)

● ARDUINO · C++

```
1 int counter = 0;
2
3 void setup() {
4   Serial.begin(9600);
5 }
6
7 void loop() {
8   Serial.println(counter); // send number to Processing
9   counter++;
10  if (counter > 100) counter = 0;
11  delay(500);
12 }
```

**WARNING**

Upload to Arduino, then **close the Serial Monitor** before running Processing.

```
1 import processing.serial.*;
2
3 Serial myPort;
4 int receivedValue = 0;
5
6 void setup() {
7     size(400, 250);
8     printArray(Serial.list());           // check available ports
9     myPort = new Serial(this, Serial.list()[0], 9600);
10    myPort.bufferUntil('\n');
11 }
12
13 void draw() {
14     background(240);
15
16     fill(50);
17     textSize(20);
18     text("Received from Arduino: " + receivedValue, 50, 80);
19
20     fill(100, 150, 255);
21     float barW = map(receivedValue, 0, 100, 0, 300);
22     rect(50, 120, barW, 40);
23
24     fill(0);
25     textSize(16);
26     text(receivedValue + " / 100", 50 + barW/2, 145);
27 }
28
29 void serialEvent(Serial p) {
30     String incoming = p.readStringUntil('\n');
31     if (incoming != null) {
32         incoming = trim(incoming);
33         if (incoming.matches("\\d+")) {           // only digits? then it is safe
34             receivedValue = int(incoming);
35         }
36     }
37 }
```

PART	EXPLANATION
<code>import processing.serial.*</code>	Imports the serial communication library
<code>Serial myPort</code>	Object that represents the serial port connection
<code>Serial.list()[0]</code>	Selects the first available COM port
<code>serialEvent(Serial p)</code>	Called automatically when new data arrives
<code>readStringUntil('\n')</code>	Reads one line of text up to the newline
<code>trim()</code>	Removes spaces and newline characters
<code>matches("\\d+")</code>	True only if the text contains digits only (safe parsing)
<code>map(value, 0, 100, 0, 300)</code>	Scales 0–100 to a pixel width of 0–300

HANDS-ON Lab 3 — Parsing Multiple Values (CSV)

In Projects 1, 3 and 5 Arduino sends **several values in one line**, separated by commas. This is the pattern you'll reuse again and again.

SendTwoValues.ino

● ARDUINO · C++

```

1 void setup() {
2   Serial.begin(9600);
3 }
4
5 void loop() {
6   float temperature = 23.5; // in the real project: a sensor reading
7   float humidity    = 61.0;
8
9   Serial.print(temperature);
10  Serial.print(",");          // comma separator
11  Serial.println(humidity);   // newline at the end
12  delay(1000);
13 }
```

```
1 import processing.serial.*;
2
3 Serial myPort;
4 float temperature = 0;
5 float humidity = 0;
6
7 void setup() {
8   size(500, 300);
9   myPort = new Serial(this, Serial.list()[0], 9600);
10  myPort.bufferUntil('\n');
11 }
12
13 void draw() {
14   background(255);
15   fill(0);
16   textSize(22);
17   text("Temperature: " + nf(temperature, 1, 1) + " \u00B0C", 50, 100);
18   text("Humidity:      " + nf(humidity, 1, 1) + " %", 50, 150);
19
20   fill(220, 50, 50); // temperature bar
21   rect(50, 170, map(temperature, 0, 50, 0, 380), 25);
22
23   fill(50, 100, 220); // humidity bar
24   rect(50, 210, map(humidity, 0, 100, 0, 380), 25);
25 }
26
27 void serialEvent(Serial p) {
28   String data = p.readStringUntil('\n');
29   if (data != null && data.contains(",")) {
30     String[] values = split(trim(data), ",");
31     if (values.length == 2) {
32       temperature = float(values[0]);
33       humidity    = float(values[1]);
34     }
35   }
36 }
```

THE CSV PARSING PATTERN – USED IN ALL PROJECTS

STEP	CODE	WHY
1. Read line	<code>readStringUntil('\n')</code>	Wait for a complete line
2. Check not null	<code>if (data != null)</code>	Avoid errors on empty reads
3. Check format	<code>data.contains(",")</code>	Confirm it is CSV
4. Split	<code>split(trim(data), ",")</code>	Divide into an array at each comma
5. Check length	<code>values.length == 2</code>	Ensure the right number of values
6. Convert	<code>float(values[0])</code>	Turn text into a number

HANDS-ON Lab 4 — Sending Commands Back to Arduino

Communication works both ways. Processing can send a character to switch a pump (Project 5) or start a servo (Project 3). Here, mouse clicks and keys switch the built-in LED.

ReceiveCommands.ino

● ARDUINO · C++

```
1 void setup() {
2   Serial.begin(9600);
3   pinMode(13, OUTPUT);           // built-in LED
4 }
5
6 void loop() {
7   if (Serial.available() > 0) {
8     char cmd = Serial.read();
9     if (cmd == '1')      digitalWrite(13, HIGH); // LED ON
10    else if (cmd == '0') digitalWrite(13, LOW);  // LED OFF
11  }
12 }
```

```
1 import processing.serial.*;
2
3 Serial myPort;
4 boolean ledOn = false;
5
6 void setup() {
7   size(400, 220);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   textAlign(CENTER, CENTER);
10 }
11
12 void draw() {
13   background(240);
14   fill(30);
15   textSize(18);
16   text("Arduino LED Control", 200, 30);
17   text("Click buttons or press 1 / 0 on keyboard", 200, 60);
18
19   fill(ledOn ? color(0, 200, 0) : color(180)); // ON button
20   rect(60, 100, 120, 50, 8);
21   fill(255);
22   text("ON", 120, 125);
23
24   fill(!ledOn ? color(200, 50, 50) : color(180)); // OFF button
25   rect(220, 100, 120, 50, 8);
26   fill(255);
27   text("OFF", 280, 125);
28
29   fill(30);
30   textSize(16);
31   text("Status: LED is " + (ledOn ? "ON" : "OFF"), 200, 190);
32 }
33
34 void setLed(boolean on) {
35   myPort.write(on ? '1' : '0');
36   ledOn = on;
37 }
38
39 void mousePressed() {
40   if (mouseX > 60 && mouseX < 180 && mouseY > 100 && mouseY < 150) setLed(true);
41   if (mouseX > 220 && mouseX < 340 && mouseY > 100 && mouseY < 150) setLed(false);
42 }
43
44 void keyPressed() {
45   if (key == '1') setLed(true);
46   if (key == '0') setLed(false);
47 }
```



LEVEL UP

All five projects use the same core pattern: **serialEvent()** → **parse** → **draw()**. Master it here, apply it everywhere.

HOW THIS INTRODUCTION CONNECTS TO THE FIVE PROJECTS

PROJECT	SENSOR	VISUALIZATION	KEY TECHNIQUE
1 — DHT11	Temperature & humidity	Two-value text + bars	CSV parsing, 2 values
2 — LDR	Light intensity	Single bar + colour change	<code>map()</code> , background colour
3 — HC-SR04	Distance (sonar)	Radar sweep display	Trigonometry, <code>ellipse</code> , <code>line</code>
4 — Ohm meter	Resistance	Numeric display + gauge	<code>nf()</code> , <code>arc()</code> , <code>text</code>
5 — Irrigation	Soil moisture	Status display + bar	Threshold logic, fill colour

CHEAT SHEET **Arduino vs. Processing — Quick Reference**

CONCEPT	ARDUINO IDE	PROCESSING
Program entry point	<code>setup()</code> + <code>loop()</code>	<code>setup()</code> + <code>draw()</code>
Print to console	<code>Serial.println()</code>	<code>println()</code>
Read input	<code>Serial.read()</code>	<code>myPort.readStringUntil()</code>
Delay / pause	<code>delay(ms)</code>	<code>delay(ms)</code> — avoid in <code>draw()</code>
Variable types	<code>int</code> , <code>float</code> , <code>char</code> , <code>bool</code>	<code>int</code> , <code>float</code> , <code>String</code> , <code>boolean</code>
Draw rectangle / text	— (no screen)	<code>rect()</code> , <code>text()</code>
Colour	— (no screen)	<code>fill()</code> , <code>stroke()</code>
Event handling	—	<code>mousePressed()</code> , <code>keyPressed()</code>
Import library	<code>#include <Lib.h></code>	<code>import library.*;</code>

Student tasks

LEVEL	TASK	GOAL
Basic	Display a single number from Arduino as text	<code>Serial</code> , <code>int()</code> , <code>text()</code>
Intermediate	Draw a coloured bar proportional to the sensor value	<code>map()</code> , <code>rect()</code> , <code>fill()</code>
Advanced	Parse two values (CSV) and display both with bars	<code>split()</code> , multiple visual elements
Challenge	Add ON/OFF buttons that send commands to Arduino	<code>mousePressed()</code> , <code>Serial.write()</code>
Challenge	Create a real-time graph of the last 100 readings	Arrays, shifting values, <code>line()</code>

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
Port not found / empty <code>Serial.list()</code>	Arduino not connected or driver missing	Check USB; install the CH340 driver for clone boards.
Port busy / in use	Arduino Serial Monitor still open	Close the Serial Monitor in the Arduino IDE first.
No data received	Wrong index in <code>Serial.list()</code> [0]	Run <code>printArray(Serial.list())</code> and choose the right index.
Garbage characters	Baud rate mismatch	Use 9600 on both sides.
<code>NullPointerException</code>	data is null (incomplete line)	Always check if <code>(data != null)</code> .
Wrong / garbage values on screen	Parsing non-numeric text	Validate with <code>data.matches("\\d+")</code> or check length after <code>split()</code> .
Sketch freezes	Heavy work or <code>delay()</code> inside <code>draw()</code>	Read serial in <code>serialEvent()</code> ; never block <code>draw()</code> .

F2 • Exercises

F2.1 BASIC CALCULATE • 10 MIN

map() by hand

Calculate each result **without a computer**, then check with the playground above.

- a) `map(512, 0, 1023, 0, 300)`
- b) `map(25, 0, 50, 0, 255)`
- c) `map(750, 1023, 300, 0, 100)`

Hint: $result = lo2 + (value - lo1) \times (hi2 - lo2) / (hi1 - lo1)$. In (c) the input range is reversed — that's fine.

F2.2 BASIC PREDICT • 5 MIN

Where will it appear?

In a `size(400, 300)` window, describe where each shape appears: `ellipse(200, 150, 40, 40)`, `rect(0, 280, 400, 20)`, `ellipse(width, 0, 60, 60)`.

F2.3 INTERMEDIATE CODE • 15 MIN

Bouncing ball

No hardware: make a ball move across the window and bounce off all four walls. Change its colour every time it hits a wall.

Hint: Store position (x, y) and speed (dx, dy). Each frame add the speed. If $x < 0$ or $x > width$, flip the sign of dx.

F2.4 INTERMEDIATE DEBUG · 10 MIN

Why is the screen blank?

Arduino is sending numbers, but this sketch shows **nothing** and sometimes crashes with a `NullPointerException`. Find the problems.

starter

● PROCESSING · JAVA

```
1 import processing.serial.*;
2 Serial myPort;
3 int value;
4
5 void setup() {
6   size(400, 200);
7   myPort = new Serial(this, Serial.list()[0], 115200);
8 }
9
10 void draw() {
11   fill(0);
12   text(value, 50, 100);
13 }
14
15 void serialEvent(Serial p) {
16   value = int(trim(p.readStringUntil('\n')));
17 }
```

Hint: Compare with Lab 2: what speed does Arduino use? What happens to old text when there is no `background()`? What if the line is not complete yet?

F2.5 ADVANCED CODE · 25 MIN

Real-time line graph

Extend Lab 2: keep the **last 100** received values in an array and draw them as a scrolling line graph across the window.

Hint: Shift every element one place left (`vals[i] = vals[i+1]`) and put the new value at the end. Use `map()` for both X (index → width) and Y (value → height, reversed!).

F2.6 CHALLENGE DESIGN · 30 MIN

Three-value protocol

Design your own message format for **three** values (e.g. `x,y,button` from a joystick). Write the Arduino side with fake values from `random()`, and a Processing sketch that moves a circle with x/y and changes its colour when `button = 1`. Your parser must **ignore** malformed lines.

Hint: Check `values.length == 3` and wrap each conversion. Test robustness by sending a broken line such as `12,,` or `hello`.

Self-check quiz

1. What is the Processing equivalent of Arduino's `loop()`?

- ☐ A `setup()`
- ☐ B `draw()`
- ☐ C `serialEvent()`
- ☐ D `repeat()`

2. In Processing, where is the point (0, 0)?

- ☐ A Centre of the window
- ☐ B Bottom-left corner
- ☐ C Top-left corner
- ☐ D Bottom-right corner

3. Arduino uses `Serial.begin(9600)`. What must Processing use?

- ☐ A `new Serial(this, port, 115200)`
- ☐ B `new Serial(this, port, 9600)`
- ☐ C Any speed works
- ☐ D `Serial.begin(9600)`

4. What does `map(50, 0, 100, 0, 400)` return?

- ☐ A 50
- ☐ B 100
- ☐ C 200
- ☐ D 400

5. Arduino sends "24.5,60". What does `split(trim(data), ",")` give?

- ☐ A A single number 84.5
- ☐ B An array ["24.5", "60"]
- ☐ C The text "24.560"
- ☐ D An error

6. Why must the Serial Monitor be closed before running Processing?

- ☐ A It slows the computer down
- ☐ B Only one program can open a serial port at a time
- ☐ C It changes the baud rate
- ☐ D It resets the Arduino

Reflection

- Explain the full journey of one sensor reading — from the physical world to a pixel on your screen — in your own words.
- Why is it a good idea to validate incoming data before using it?
- Where have you seen live data visualizations in everyday life (weather apps, fitness trackers, ...)?

CONCLUSION

Processing is the visual bridge between the physical world and the screen. Once you understand how data flows from a sensor through Arduino's serial port into Processing's `draw()`, you can build any real-time visualization you can imagine. Serial setup, parsing, drawing with `map()`, and sending commands back to Arduino are a complete toolkit for every project that follows.

Temperature & Humidity — DHT11

Build a mini weather station that streams live climate data.

Measure air temperature and relative humidity with a DHT11 sensor, send both values as CSV over serial, and visualize them in real time with Processing — your first IoT-style system.

BY THE END OF THIS MODULE YOU CAN...

1. Install and use an Arduino library (Adafruit DHT sensor library).
2. Wire a digital sensor and explain the role of a pull-up resistor.
3. Explain what relative humidity means physically.
4. Send two measurements in one CSV line and parse them in Processing.
5. Detect sensor errors with `isnan()` and add a high-temperature alarm.

KEY VOCABULARY

DHT11: A low-cost digital sensor for temperature (0–50 °C, ± 2 °C) and humidity (20–90 % RH, ± 5 %).

Relative humidity (RH): How much water vapour the air holds compared with the maximum it could hold at that temperature, in %.

Library: Ready-made code you include to talk to a component without writing everything yourself.

Pull-up resistor: A resistor that keeps a signal line at HIGH when nothing is pulling it LOW.

NaN: “Not a Number” — the value returned when a reading failed. Test with `isnan()`.

IoT: Internet of Things — everyday objects with sensors that share data.

Required equipment

COMPONENT	QTY	PURPOSE
Arduino Uno	1	Microcontroller
DHT11 sensor	1	Temperature and humidity measurement
10 kΩ resistor	1	Pull-up resistor (only for the bare 3/4-pin sensor)
Breadboard	1	Component connection
Jumper wires	~5	Component connection
USB cable	1	Connection to computer
Arduino IDE + Processing	1	Programming and visualization

What is the DHT11?

The DHT11 is a **digital** sensor: it measures temperature and relative humidity and sends the result as a digital signal. No analog-to-digital conversion is needed, it is easy to connect, and ready-made libraries exist — perfect for beginners.

What is relative humidity?

Relative humidity is the percentage of water vapour in the air compared with the **maximum** amount the air can hold at that temperature and normal pressure. Warm air can hold more water than cold air — that's why a cold bottle from the fridge "sweats" on a warm day.

FORMULA

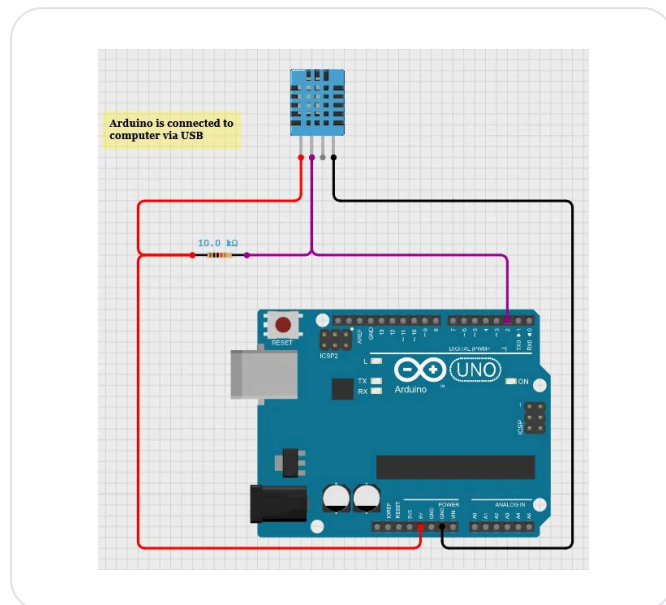
$$RH = (\text{actual water vapour} \div \text{maximum possible at this temperature}) \times 100 \%$$

Serial communication

Arduino sends both readings in one line, separated by a comma:

```
1 Serial.println("24.5,60"); // temperature = 24.5 °C, humidity = 60 %
```

HANDS-ON Exercise 1 — Connecting the DHT11



— DHT11 wiring to the Arduino Uno.

DHT11 PIN	ARDUINO PIN
VCC	5V
GND	GND
DATA	Digital pin 2

Using the bare 3/4-pin sensor? Place a **10 kΩ resistor** between **VCC** and **DATA**.



TIP

The 3-pin DHT11 **module** (on a small PCB) already has a built-in pull-up resistor.

Step 1 — Install the library

- 01 In the Arduino IDE open **Sketch → Include Library → Manage Libraries**.
- 02 Search for **"DHT sensor library" by Adafruit**.
- 03 Click **Install** (accept the "Adafruit Unified Sensor" dependency).

Step 2 — Arduino code

DHT11_Serial.ino

● ARDUINO · C++

```
1 #include "DHT.h"
2
3 #define DHTPIN 2
4 #define DHTTYPE DHT11
5
6 DHT dht(DHTPIN, DHTTYPE);
7
8 void setup() {
9   Serial.begin(9600);
10  dht.begin();
11 }
12
13 void loop() {
14   float humidity = dht.readHumidity();
15   float temperature = dht.readTemperature(); // Celsius
16
17   if (isnan(humidity) || isnan(temperature)) {
18     Serial.println("Failed to read from DHT sensor!");
19     delay(2000);
20     return;
21   }
22
23   Serial.print(temperature);
24   Serial.print(",");
25   Serial.println(humidity);
26   delay(2000); // DHT11: max. 1 reading per second
27 }
```

COMMAND	EXPLANATION
<code>dht.readTemperature()</code>	Reads the temperature in °C
<code>dht.readHumidity()</code>	Reads the relative humidity in %
<code>isnan(x)</code>	True if the reading failed (Not a Number)
<code>Serial.print()</code>	Sends data to the computer
<code>delay(2000)</code>	Waits 2 seconds between readings



TIP

Data is sent as **comma-separated values** because Processing can split them easily.

HANDS-ON Exercise 2 — Data Visualization in Processing

The sketch reads the data from Arduino, shows temperature and humidity as text, and visualizes both with bars.

WeatherStation.pde

PROCESSING • JAVA

```
1 import processing.serial.*;
2
3 Serial myPort;
4 float temp = 0, hum = 0;
5
6 void setup() {
7   size(500, 260);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13   background(255);
14   fill(0);
15   textSize(20);
16   text("Temperature: " + nf(temp, 1, 1) + " \u00B0C", 50, 70);
17   text("Humidity: " + nf(hum, 1, 1) + " %", 50, 150);
18
19   fill(220, 50, 50); // temperature bar 0-50 °C
20   rect(50, 85, map(temp, 0, 50, 0, 400), 25);
21   fill(50, 100, 220); // humidity bar 0-100 %
22   rect(50, 165, map(hum, 0, 100, 0, 400), 25);
23
24   if (temp > 30) { // alert system
25     fill(255, 0, 0);
26     text("WARNING: HIGH TEMPERATURE!", 50, 235);
27   }
28 }
29
30 void serialEvent(Serial p) {
31   String data = p.readStringUntil('\n');
32   if (data != null && data.contains(",")) {
33     String[] values = split(trim(data), ",");
34     if (values.length == 2) {
35       temp = float(values[0]);
36       hum = float(values[1]);
37     }
38   }
39 }
```

UPDATED Added the missing `import processing.serial.*`, moved reading into `serialEvent()` and added bars and the alarm.

Student tasks

LEVEL	TASK	GOAL
Basic	Display temperature and humidity	
Intermediate	Add graphical progress bars	
Advanced	Add a high-temperature alarm	
Challenge	Save data to a CSV file	
Challenge	Draw a real-time graph	

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
No data received	Incorrect COM port	Use <code>printArray(Serial.list())</code> to find the correct port.
"Failed to read" message	Poorly connected sensor or wrong pin	Check DATA → pin 2 and that DHTTYPE is DHT11 (not DHT22).
Unstable values	Poor contact on the breadboard	Press wires in firmly; avoid long loose jumpers.
Processing not receiving data	Serial speed mismatch	Both sides 9600 baud.

P1 • Exercises

P1.1 BASIC EXPLAIN • 10 MIN

Read the datasheet

The DHT11 datasheet says: *range 0–50 °C, accuracy ±2 °C; humidity 20–90 % RH, accuracy ±5 %*. Your sensor shows **22 °C** and **48 %**. What are the possible real values? Could you use this sensor in a freezer?

P1.2 BASIC PREDICT • 5 MIN

Parse it in your head

For each line Arduino might send, write what `temp` and `hum` become in the Processing sketch (or "unchanged"): `25.0,40.0` • `Failed to read from DHT sensor!` • `26.1,` • `27.3,55.0,1`

Hint: Check each guard: `contains(",")` and `values.length == 2`.

P1.3 INTERMEDIATE CODE • 15 MIN

Comfort zone indicator

Classrooms are comfortable between **20–24 °C** and **40–60 % RH**. Add a large circle to the Processing sketch that is **green** inside the comfort zone, **yellow** if only one value is outside, and **red** if both are outside. Write the status text next to it.

P1.4 INTERMEDIATE CALCULATE • 15 MIN

Dew point estimate

A simple approximation for the **dew point** (the temperature at which water condenses) is $T_d \approx T - (100 - RH) / 5$. Calculate T_d for (a) 25 °C / 60 % and (b) 18 °C / 90 %. Then add the calculation to Processing and display it.

P1.5 ADVANCED CODE • 20 MIN

Log to a CSV file

Save every reading with a timestamp to `weather.csv` so you can open it later in a spreadsheet and draw a chart.

Hint: Use `PrintWriter output = createWriter("weather.csv");` in `setup()`, `output.println(...)` in `serialEvent()`, and `output.flush(); output.close();` when a key is pressed.

Classroom climate study

Run a real investigation: log the classroom climate for one full lesson **with windows closed** and one **with windows open**. Plot both datasets, compare the averages, and write a short recommendation for your school (how often should you ventilate?).

1. Formulate a hypothesis.
2. Collect data with your logger (P1.5).
3. Compute min, max and average for each condition.
4. Present a chart and a 5-sentence conclusion.

Self-check quiz

1. What does the DHT11 measure?

- ☐ A Light and sound
- ☐ B Temperature and relative humidity
- ☐ C Distance and speed
- ☐ D Air pressure

2. What does `isnan(temperature)` check?

- ☐ A Whether the temperature is negative
- ☐ B Whether the reading failed (Not a Number)
- ☐ C Whether it is too hot
- ☐ D Whether the sensor is DHT22

3. Why does the code wait 2 seconds between readings?

- ☐ A To save battery
- ☐ B The DHT11 can only give a new reading about once per second
- ☐ C Processing is slow
- ☐ D To make the LED blink

4. Warm air can hold ... water vapour than cold air.

- ☐ A less
- ☐ B the same amount of
- ☐ C more
- ☐ D no

5. The 4-pin bare DHT11 needs a 10 kΩ resistor between...

- ☐ A DATA and GND
- ☐ B VCC and DATA
- ☐ C VCC and GND
- ☐ D Nothing — it never needs one

Reflection

- Where would you place the sensor in a room to get a fair measurement? Why not next to the radiator or window?
- Our sensor is ± 2 °C accurate. Is that good enough for a weather station? For a hospital incubator?

- How could this project help save energy at school?

CONCLUSION

This project introduced working with digital sensors, libraries, and two-value data streams. By combining the Arduino Uno with Processing you built a real-time climate monitor — the same principle used in smart homes, greenhouses and weather stations around the world.

Light Sensor — Photoresistor (LDR)

Read the brightness of the room and paint it on screen.

Use a photoresistor in a voltage divider, read it with `analogRead()` (0–1023), and build a Processing display whose bar and background react to light in real time.

BY THE END OF THIS MODULE YOU CAN...

1. Explain how the resistance of an LDR changes with light.
2. Build a voltage divider and calculate its output voltage.
3. Convert between voltage and the 10-bit ADC value (0–1023).
4. Visualize a single sensor value as text, a bar and a background colour.
5. Calibrate a sensor by measuring its minimum and maximum in real conditions.

KEY VOCABULARY

LDR / photoresistor: A resistor whose resistance falls when light shines on it ($\approx 1\text{ M}\Omega$ in darkness, $\approx 1\text{ k}\Omega$ in bright light).

Voltage divider: Two resistors in series; the voltage in the middle depends on their ratio.

ADC: Analog-to-Digital Converter: turns 0–5 V into a number 0–1023 (10-bit) on the Uno.

`analogRead()`: Arduino function that reads an analog pin and returns 0–1023.

Calibration: Measuring real minimum and maximum values so your scale fits reality.

Required equipment

COMPONENT	QTY	PURPOSE
Arduino Uno	1	Microcontroller
Photoresistor (LDR)	1	Light sensing
10 k Ω resistor	1	Voltage divider
Breadboard	1	Component connection
Jumper wires	~5	Component connection
USB cable	1	Connection to computer
Processing IDE	1	Data visualization

THEORY Theoretical Background

How does a photoresistor work?

An LDR (Light Dependent Resistor) is made of a semiconductor. Light gives its electrons enough energy to conduct, so **in bright light the resistance decreases; in darkness it increases**.

The voltage divider

Arduino can't measure resistance directly — only voltage. So we put the LDR in series with a fixed 10 kΩ resistor. The voltage at the junction (connected to A0) changes with light.

FORMULA

$$V_{A0} = 5\text{ V} \times R_{\text{fixed}} / (R_{\text{LDR}} + R_{\text{fixed}})$$

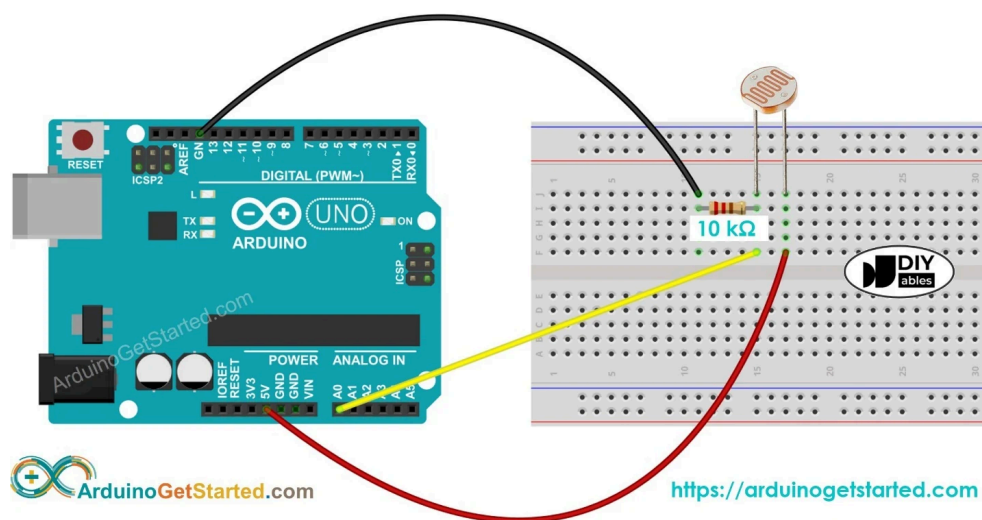
In our wiring the LDR is on the 5V side and the 10 kΩ resistor on the GND side. More light → smaller R_{LDR} → higher voltage → higher reading.

FORMULA

$$\text{reading} = V_{A0} / 5\text{ V} \times 1023$$

The ADC converts the voltage into a number from 0 to 1023.

HANDS-ON Exercise 1 — Circuit Setup & Arduino Code



— LDR + 10 kΩ voltage divider connected to A0.

COMPONENT	CONNECTS TO
LDR (leg 1)	5V
LDR (leg 2)	A0 and 10 kΩ resistor (leg 1)
10 kΩ resistor (leg 2)	GND



TIP

The junction between the LDR and the resistor connects to **A0**.

LightSensor.ino

● ARDUINO · C++

```

1  const int sensorPin = A0;
2
3  void setup() {
4    Serial.begin(9600);
5  }
6
7  void loop() {
8    int lightValue = analogRead(sensorPin);    // 0 (dark) ... 1023 (bright)
9    Serial.println(lightValue);
10   delay(200);
11 }
```

HANDS-ON Exercise 2 — Processing Visualization

Start simple with a text display, then upgrade to the robust `serialEvent()` version with a bar.

```
1 import processing.serial.*;
2
3 Serial myPort;
4 int lightLevel = 0;
5
6 void setup() {
7   size(400, 200);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9 }
10
11 void draw() {
12   background(255);
13   fill(0);
14   textSize(20);
15   text("Light Intensity: " + lightLevel, 50, 100);
16
17   if (myPort.available() > 0) {
18     String val = myPort.readStringUntil('\n');
19     if (val != null) lightLevel = int(trim(val));
20   }
21 }
```

```
1 import processing.serial.*;
2
3 Serial myPort;
4 int lightLevel = 0;
5
6 void setup() {
7   size(400, 200);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13   background(255);
14   fill(0);
15   textSize(20);
16   text("Light Intensity: " + lightLevel, 50, 50);
17
18   fill(0, 100, 255);
19   rect(50, 100, map(lightLevel, 0, 1023, 0, 300), 30);
20 }
21
22 void serialEvent(Serial p) {
23   String val = p.readStringUntil('\n');
24   if (val != null) {
25     val = trim(val);
26     if (val.matches("\\d+")) lightLevel = int(val);
27   }
28 }
```



MAKE IT REACT

Replace `background(255)` with `background(map(lightLevel, 0, 1023, 0, 255));` — the window now gets darker when the room gets darker.

Student tasks

LEVEL	TASK	GOAL
Basic	Display the numeric light value	
Intermediate	Draw a coloured bar proportional to the light level	
Advanced	Change the background colour based on light intensity	
Challenge	Log values with timestamps to a text file	

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
Value stays at 0	LDR not connected or open circuit	Check the LDR leg goes to 5V and the junction to A0.
Value stays at 1023	Resistor missing or GND not connected	Check the 10 kΩ resistor goes from A0 to GND.
Unstable readings	Poor breadboard contact or flickering lights	Press components in firmly; average several readings.
Range is small (e.g. 400–700)	Fixed resistor doesn't match the LDR	Try 4.7 kΩ or 22 kΩ, or calibrate with <code>map()</code> .

P2 • Exercises

P2.1 BASIC CALCULATE • 10 MIN

Voltage divider practice

With $R_{\text{fixed}} = 10\text{ k}\Omega$ and 5 V supply, calculate **V_A0** and the expected **analogRead()** value when the LDR is (a) $1\text{ k}\Omega$ (bright), (b) $10\text{ k}\Omega$ (normal room), (c) $100\text{ k}\Omega$ (dark).

Hint: $V_{A0} = 5 \times 10 / (R_{\text{LDR}} + 10)$ with resistances in $\text{k}\Omega$; $\text{reading} = V_{A0} / 5 \times 1023$.

P2.2 BASIC PREDICT • 5 MIN

Swap the parts

What happens to the readings if you **swap** the LDR and the $10\text{ k}\Omega$ resistor (LDR now between A0 and GND)?

P2.3 INTERMEDIATE BUILD • 20 MIN

Automatic night-light

Add an LED (pin 9 + $220\text{ }\Omega$). Turn it **on** when the reading drops below a threshold of your choice and **off** when it is brighter. Print the state to the Serial Monitor.

P2.4 INTERMEDIATE CODE • 15 MIN

Smooth dimming with PWM

Instead of on/off, make the LED on pin 9 get **brighter as the room gets darker**, using `analogWrite()` (0–255).

Hint: Pin 9 supports PWM (~). Use `map(light, 0, 1023, 255, 0)` — note the reversed output range.

P2.5 ADVANCED CODE • 20 MIN

Calibrate with min/max

Your sensor never reaches 0 or 1023. Make Processing **remember the smallest and largest** values it has seen and map the bar between those, so it always uses the full width. Press **R** to reset calibration.

Sunrise logger

Design an experiment that measures how light levels change near a window during a whole school day. Log a value every minute with a timestamp, then plot a graph. At what time is your classroom brightest? Does it depend on the weather?

Self-check quiz

1. In bright light, an LDR's resistance...

- ☐ A increases
- ☐ B decreases
- ☐ C stays the same
- ☐ D becomes zero

2. What range does `analogRead()` return on the Arduino Uno?

- ☐ A 0–255
- ☐ B 0–1023
- ☐ C 0–5
- ☐ D –512–512

3. Why do we need the fixed 10 k Ω resistor?

- ☐ A To protect the LDR from light
- ☐ B To form a voltage divider so Arduino can measure a changing voltage
- ☐ C To make the LED brighter
- ☐ D It is optional

4. An analog reading of 512 corresponds to roughly...

- ☐ A 0.5 V
- ☐ B 1 V
- ☐ C 2.5 V
- ☐ D 5 V

5. The reading is always 1023. The most likely cause is:

- ☐ A Too much light
- ☐ B Fixed resistor or GND not connected
- ☐ C Wrong baud rate
- ☐ D The LDR is reversed

Reflection

- Which devices you own adjust themselves to light? How do you think they do it?
- The LDR response is not linear. Why might that matter when you build a “lux meter”?
- How could automatic lighting reduce your school's energy use?

CONCLUSION

You learned how analog sensors and voltage dividers work, and how to turn a changing resistance into a live visualization. By combining the Arduino Uno with Processing you built an interactive light meter — the same idea behind auto-brightness screens and smart street lighting.

Sonar — Ultrasonic Sensor HC-SR04

Measure distance with sound and build a working radar screen.

Use the HC-SR04 to measure distance with ultrasound, control an SG90 servo from Processing, make the servo react to obstacles, and finally combine everything into a sweeping radar display.

BY THE END OF THIS MODULE YOU CAN...

1. Explain how an ultrasonic sensor measures distance using the echo time.
2. Calculate distance from time with $d = v \cdot t / 2$ and explain the factor 2.
3. Control a servo motor with the Servo library and commands from Processing.
4. Write non-blocking code so Arduino can react to new commands at any time.
5. Convert polar coordinates (angle, distance) to screen coordinates with $\sin()$ and $\cos()$.

KEY VOCABULARY

Ultrasound: Sound above human hearing (> 20 kHz). The HC-SR04 uses 40 kHz.

Echo: A reflected sound wave that returns to the sensor.

pulseIn(): Arduino function that measures how long a pin stays HIGH, in microseconds.

Servo motor: A motor that turns to a precise angle (0–180°) controlled by a PWM signal.

PWM: Pulse-Width Modulation — switching a signal on and off quickly; the pulse width carries the information.

Polar coordinates: Describing a point by an angle and a distance from the centre, like a radar does.

Required equipment

COMPONENT	QTY	PURPOSE
Arduino Uno	1	Microcontroller
HC-SR04 ultrasonic sensor	1	Distance measurement (2–400 cm)
SG90 servo motor	1	Radar sweep (Ex. 2–4)
Plastic holder for sensor	1	Mount the sensor on the servo
Breadboard + jumper wires	1 set	Component connection
USB cable	1	Connection to computer
Processing IDE	1	Visualization

THEORY **Physics of Ultrasonic Sensing**

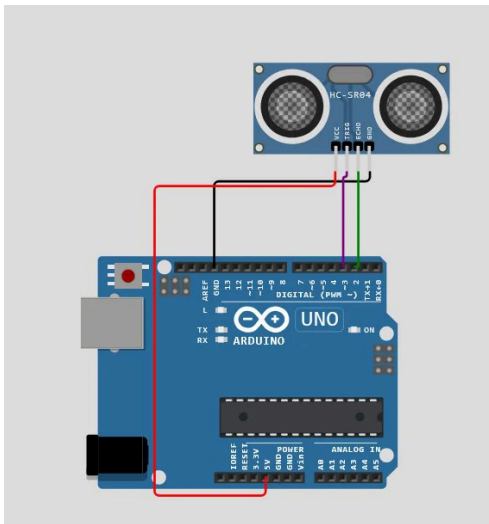
The sensor emits a short burst of ultrasound (~40 kHz). When the wave hits an object it bounces back. The sensor measures the time until the echo returns — just like bats and dolphins do.

FORMULA

$$\text{distance} = (\text{speed of sound} \times \text{time}) / 2$$

Speed of sound $\approx 343 \text{ m/s} = 0.0343 \text{ cm}/\mu\text{s}$ in air at 20 °C. We divide by 2 because the sound travels there **and back**.

HANDS-ON **Exercise 1 — Measure Distance & Display in Processing**



— HC-SR04 connected to the Arduino Uno.

HC-SR04 PIN	ARDUINO PIN
VCC	5V
GND	GND
TRIG	Pin 3
ECHO	Pin 2



NOTE

TRIG starts a measurement with a 10 μs pulse. ECHO stays HIGH for as long as the sound was travelling.

```

1 #define trigPin 3
2 #define echoPin 2
3
4 void setup() {
5   Serial.begin(9600);
6   pinMode(trigPin, OUTPUT);
7   pinMode(echoPin, INPUT);
8 }
9
10 void loop() {
11   digitalWrite(trigPin, LOW);
12   delayMicroseconds(2);
13   digitalWrite(trigPin, HIGH);           // 10 µs trigger pulse
14   delayMicroseconds(10);
15   digitalWrite(trigPin, LOW);
16
17   long duration = pulseIn(echoPin, HIGH, 30000); // timeout 30 ms ≈ 5 m
18   float distance = duration * 0.0343 / 2;       // cm
19   Serial.println(distance);
20   delay(200);
21 }

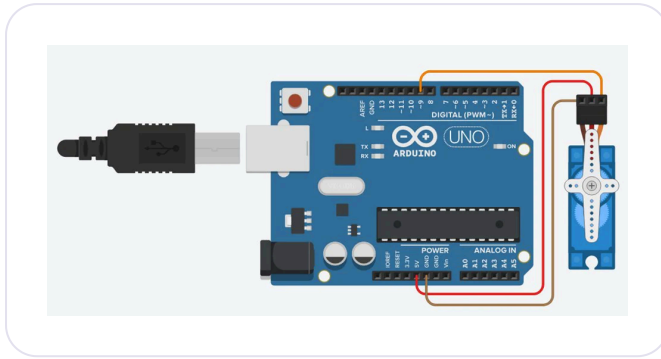
```

```

1 import processing.serial.*;
2
3 Serial myPort;
4 float distance;
5
6 void setup() {
7   size(600, 200);
8   printArray(Serial.list());
9   myPort = new Serial(this, Serial.list()[0], 9600);
10  myPort.bufferUntil('\n');
11 }
12
13 void draw() {
14   background(255);
15   fill(0);
16   textSize(24);
17   text("Distance: " + nf(distance, 0, 2) + " cm", 50, 50);
18
19   fill(100, 200, 255);           // horizontal bar
20   rect(50, 100, map(constrain(distance, 0, 200), 0, 200, 0, width - 100), 30);
21 }
22
23 void serialEvent(Serial p) {
24   String data = p.readStringUntil('\n');
25   if (data != null) distance = float(trim(data));
26 }

```

HANDS - ON Exercise 2 — Control the SG90 Servo from Processing



— SG90 servo wiring.

SERVO WIRE	ARDUINO PIN
Signal (orange)	Pin 9
VCC (red)	5V
GND (brown)	GND

Background reading: *The Beginner's Guide to Micro Servos* —

docs.arduino.cc/learn/electronics/servo-motors

ServoSweep.ino

● ARDUINO · C++

```
1 #include <Servo.h>
2
3 Servo servo;
4 bool sweeping = false;
5 int angle = 0;
6 int step = 1;
7
8 void setup() {
9   Serial.begin(9600);
10  servo.attach(9);
11 }
12
13 void loop() {
14   if (Serial.available()) {
15     String command = Serial.readStringUntil('\n');
16     command.trim();
17     sweeping = (command == "1");    // "1" = START, anything else = STOP
18   }
19
20   if (sweeping) {                  // one small step per loop()
21     angle += step;
22     if (angle >= 180 || angle <= 0) step = -step;
23     servo.write(angle);
24     delay(10);
25   }
26 }
```

UPDATED Rewritten as a non-blocking sweep: the original nested for-loops could not react to STOP until a full sweep had finished.

```

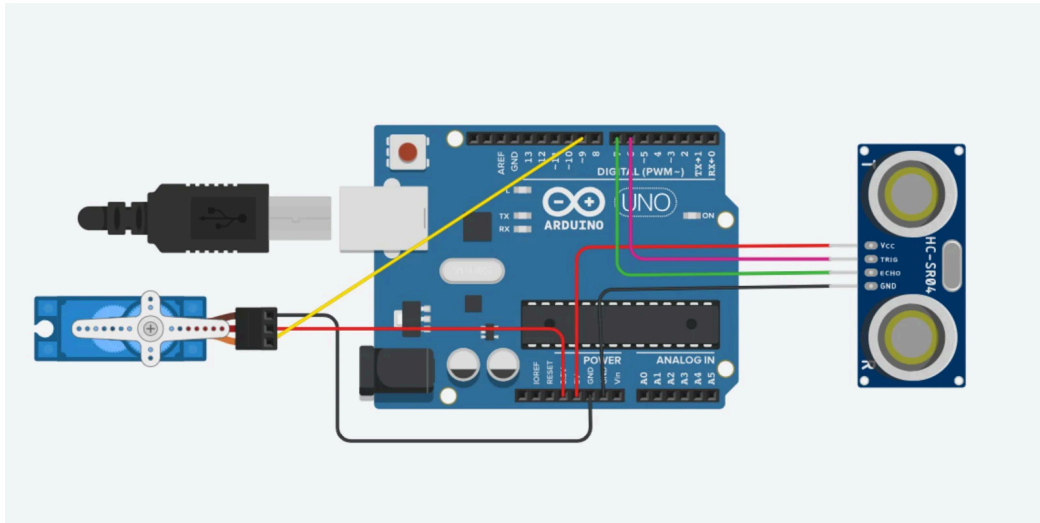
1  import processing.serial.*;
2
3  Serial myPort;
4  boolean isSweeping = false;
5
6  void setup() {
7      size(400, 200);
8      myPort = new Serial(this, Serial.list()[0], 9600);
9      textAlign(CENTER, CENTER);
10 }
11
12 void draw() {
13     background(240);
14     textSize(16);
15     fill(0);
16     text("Servo Control", width/2, 30);
17     drawButton("START", 100, 100, isSweeping);
18     drawButton("STOP", 250, 100, !isSweeping);
19 }
20
21 void drawButton(String label, int x, int y, boolean active) {
22     fill(active ? color(0, 200, 0) : color(180));
23     rect(x, y, 100, 40, 7);
24     fill(0);
25     text(label, x + 50, y + 20);
26 }
27
28 void mousePressed() {
29     if (mouseX > 100 && mouseX < 200 && mouseY > 100 && mouseY < 140) {
30         myPort.write("1\n");
31         isSweeping = true;
32     } else if (mouseX > 250 && mouseX < 350 && mouseY > 100 && mouseY < 140) {
33         myPort.write("0\n");
34         isSweeping = false;
35     }
36 }

```

- ▶ **START** sends "1" → Arduino begins sweeping the servo.
- ▶ **STOP** sends "0" → Arduino halts the movement immediately.
- ▶ Expand it with an angle slider, speed control or real-time feedback.

HANDS-ON Exercise 3 — Ultrasonic Sensor Controlling the Servo

The servo turns to **90°** when an object is closer than 50 cm and returns to **0°** otherwise — like an automatic barrier.



— HC-SR04 (TRIG 6, ECHO 7) and servo (pin 9) together.

```
1 #include <Servo.h>
2
3 const int TRIG_PIN = 6;
4 const int ECHO_PIN = 7;
5 const int SERVO_PIN = 9;
6 const int DISTANCE_THRESHOLD = 50; // centimetres
7
8 Servo servo;
9 float duration_us, distance_cm;
10
11 void setup() {
12   Serial.begin(9600);
13   pinMode(TRIG_PIN, OUTPUT);
14   pinMode(ECHO_PIN, INPUT);
15   servo.attach(SERVO_PIN);
16   servo.write(0);
17 }
18
19 void loop() {
20   digitalWrite(TRIG_PIN, HIGH); // 10 µs pulse
21   delayMicroseconds(10);
22   digitalWrite(TRIG_PIN, LOW);
23
24   duration_us = pulseIn(ECHO_PIN, HIGH);
25   distance_cm = 0.017 * duration_us; // = 0.0343 / 2
26
27   if (distance_cm < DISTANCE_THRESHOLD) servo.write(90);
28   else servo.write(0);
29
30   Serial.print("distance: ");
31   Serial.print(distance_cm);
32   Serial.println(" cm");
33   delay(500);
34 }
```

**STUDENT TASK**

Write a Processing program that shows the distance (a) as a number with `text()` and (b) as a horizontal bar. Careful: this sketch sends `distance: 23.4 cm`, not just a number — see exercise P3.4.

PROJECT Exercise 4 — Build a Radar Scanner**LEVEL UP**

This exercise is suitable as a full project topic.

The servo sweeps from 0° to 180°; at each step Arduino measures the distance and sends `angle,distance` . Processing converts each pair to an (x, y) point.

RadarSweep.ino

● ARDUINO · C++

```
1 #include <Servo.h>
2
3 Servo myServo;
4 #define trigPin 9
5 #define echoPin 10
6
7 float measure() {
8     digitalWrite(trigPin, LOW);
9     delayMicroseconds(2);
10    digitalWrite(trigPin, HIGH);
11    delayMicroseconds(10);
12    digitalWrite(trigPin, LOW);
13    long duration = pulseIn(echoPin, HIGH, 30000);
14    return duration * 0.0343 / 2;
15 }
16
17 void setup() {
18     Serial.begin(9600);
19     myServo.attach(6);
20     pinMode(trigPin, OUTPUT);
21     pinMode(echoPin, INPUT);
22 }
23
24 void loop() {
25     for (int angle = 0; angle <= 180; angle += 2) {
26         myServo.write(angle);
27         delay(50);
28         Serial.print(angle);
29         Serial.print(",");
30         Serial.println(measure());
31     }
32     for (int angle = 180; angle >= 0; angle -= 2) { // sweep back too
33         myServo.write(angle);
34         delay(50);
35         Serial.print(angle);
36         Serial.print(",");
37         Serial.println(measure());
38     }
39 }
```

```

1  import processing.serial.*;
2
3  Serial myPort;
4  int angle;
5  float distance;
6
7  void setup() {
8      size(600, 600);
9      myPort = new Serial(this, Serial.list()[0], 9600);
10     background(0);
11 }
12
13 void draw() {
14     if (myPort.available() > 0) {
15         String data = myPort.readStringUntil('\n');
16         if (data != null && data.contains(",")) {
17             String[] parts = split(trim(data), ",");
18             if (parts.length == 2) {
19                 angle = int(parts[0]);
20                 distance = float(parts[1]);
21                 float r = map(distance, 0, 100, 0, 250);
22                 float x = width/2 + r * cos(radians(angle));
23                 float y = height - r * sin(radians(angle));
24                 stroke(0, 255, 0);
25                 fill(0, 255, 0);
26                 ellipse(x, y, 5, 5);
27             }
28         }
29     }
30 }

```

FORMULA

$$x = cx + r \cdot \cos(\theta) \quad y = cy - r \cdot \sin(\theta)$$

Minus, because Y grows downward on screen.

LEVEL	TASK
Basic	Display distance as text on the radar screen
Intermediate	Draw radar grid lines and circles
Advanced	Colour-code dots by distance (green / yellow / red)
Challenge	Add sweep-line animation and a sound alert

Student tasks

LEVEL	TASK	GOAL
Basic	Display distance as text	Serial + text()
Intermediate	Draw radar grid lines and circles	arc(), line(), loops
Advanced	Colour-code dots by distance (green/yellow/red)	Conditionals, colour
Challenge	Add sweep-line animation and a sound alert	Trigonometry, fading, Minim

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
Distance always 0	No echo (timeout) or TRIG/ECHO swapped	Check pins; aim at a flat object 10–100 cm away.
Random big jumps	Soft or angled surfaces absorb/deflect sound	Use flat, hard targets; average 3 readings.
Servo jitters or Arduino resets	Servo draws too much current from USB	Power the servo from a separate 5 V supply (common GND).
Radar points drawn upside down	Y-axis points down on screen	Use $y = cy - r \cdot \sin(\theta)$.

P3 • Exercises

P3.1 BASIC CALCULATE • 10 MIN

Echo arithmetic

`pulseIn()` returns **1166 μ s**. (a) How far away is the object? (b) What echo time do you expect for an object **2 m** away?

Hint: $d = t \times 0.0343 / 2$ (cm, with t in μ s). For (b) rearrange: $t = 2d / 0.0343$.

P3.2 BASIC EXPLAIN • 5 MIN

Why divide by two?

A classmate forgets the `/ 2` in the distance formula. What will their readings look like? Explain using a sketch of the sound path.

P3.3 INTERMEDIATE CALCULATE • 10 MIN

Temperature matters

The speed of sound depends on air temperature: $v \approx 331.3 + 0.606 \times T$ (m/s, T in $^{\circ}\text{C}$). At **0 $^{\circ}\text{C}$** and **35 $^{\circ}\text{C}$** , what distance would the sensor report for a real distance of 100 cm if the code always assumes 343 m/s?

Hint: First find the real echo time $t = 2 \times 1 \text{ m} / v_{\text{real}}$, then compute the reported distance with $v = 343 \text{ m/s}$.

P3.4 INTERMEDIATE DEBUG • 15 MIN

Text in the numbers

Exercise 3 sends lines like `distance: 23.4 cm`. A student uses `distance = float(trim(data));` in Processing and the bar never moves. Why? Fix it **two** ways: once in Arduino, once in Processing.

P3.5 ADVANCED CALCULATE • 15 MIN

Polar to screen

The radar centre is at (300, 600) and 100 cm maps to 250 px. Calculate the screen position of an object at **angle 30°, distance 60 cm**, and at **angle 135°, distance 100 cm**.

Hint: $r = 60 \times 2.5 = 150 \text{ px}$. $x = 300 + r \cdot \cos \theta$, $y = 600 - r \cdot \sin \theta$.

Full radar display

Build the complete radar: green grid arcs every 25 cm, angle lines every 30°, a sweep line following the current angle, a fading trail, and dots colour-coded by distance (red < 30 cm, yellow < 60 cm, green otherwise).

Hint: Instead of `background(0)` each frame, draw a translucent black rectangle (`fill(0, 20); rect(0, 0, width, height);`) — older drawings slowly fade away.

Self-check quiz

1. What frequency does the HC-SR04 use?

- ☐ A 40 Hz
- ☐ B 440 Hz
- ☐ C 40 kHz
- ☐ D 4 MHz

2. Why is the measured time divided by 2?

- ☐ A To convert μs to ms
- ☐ B The sound travels to the object and back
- ☐ C The sensor has two eyes
- ☐ D To round the value

3. Which library controls the SG90?

- ☐ A Wire.h
- ☐ B Servo.h
- ☐ C DHT.h
- ☐ D Motor.h

4. A radar point at 90° is drawn...

- ☐ A to the right of the centre
- ☐ B straight above the centre
- ☐ C to the left of the centre
- ☐ D below the centre

5. Why was the servo sweep rewritten as "one step per loop()"?

- ☐ A To make it faster
- ☐ B So Arduino can read new commands (like STOP) between steps
- ☐ C Because for-loops are not allowed
- ☐ D To save memory

Reflection

- Bats "see" with sound. What are the advantages and disadvantages of sound compared with light for detecting objects?
- Which surfaces were hard for your sensor to detect? Why?
- Where could a radar like yours be useful at school or at home?

CONCLUSION

You explored ultrasonic sensing, real-time measurement, servo control and interactive visualization. Combining the HC-SR04, a servo and Processing, you built a working radar inspired by real sonar technology — and used physics, trigonometry and programming together in one system.

Arduino Ohm Meter with Processing Display

Turn your Arduino into a measuring instrument.

Build a digital ohm meter: a voltage divider with one known resistor lets Arduino calculate an unknown resistance using Ohm's law, and Processing shows the result like a real instrument.

BY THE END OF THIS MODULE YOU CAN...

1. Apply Ohm's law and the voltage-divider rule to a real circuit.
2. Rearrange the divider formula to solve for an unknown resistor.
3. Convert ADC readings to voltage and estimate measurement accuracy.
4. Handle edge cases (open circuit, short circuit) in code.
5. Present a measurement professionally with units and formatting.

KEY VOCABULARY

Ohm's law: $U = R \cdot I$ — voltage equals resistance times current.

Series circuit: Components connected one after another; the same current flows through all of them.

Reference resistor: The known resistor (R_{known}) used to compare against the unknown one.

Resolution: The smallest change an instrument can detect. The Uno ADC: $5 \text{ V} / 1023 \approx 4.9 \text{ mV}$.

Tolerance: How far a real resistor may differ from its printed value (gold band = $\pm 5 \%$).

Required equipment

COMPONENT	QTY	PURPOSE
Arduino Uno	1	Microcontroller
Known resistor (e.g. 1 k Ω)	1	Reference for the voltage divider
Unknown resistors	several	Components to measure
Breadboard + jumper wires	1 set	Component connection
USB cable	1	Connection to computer
Processing IDE	1	Display

THEORY Voltage Divider & Ohm's Law

Two resistors in series share the 5 V supply. The voltage at the midpoint depends on their **ratio** — so if we know one resistor and measure the voltage, we can calculate the other.

FORMULA

$$V_{A0} = V_{in} \times R_{known} / (R_x + R_{known})$$

R_x (unknown) sits between 5V and A0, R_{known} between A0 and GND — exactly as in the diagram.

FORMULA

$$R_x = R_{known} \times (V_{in} / V_{A0} - 1)$$

Rearranged to solve for the unknown resistor.



CORRECTED FROM THE ORIGINAL HANDBOOK

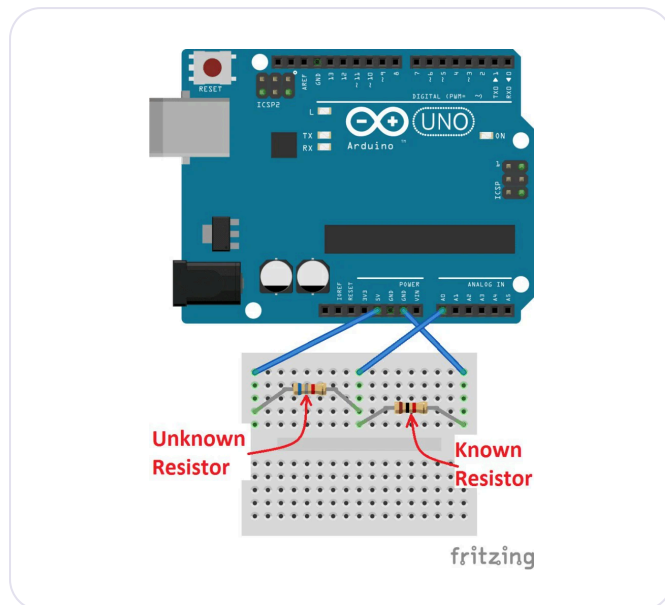
The original wiring table placed the known resistor on the 5V side, which does not match the diagram or the formula. This edition follows the diagram: **unknown on top (5V → A0), known on the bottom (A0 → GND).**



NOTE

The **larger** the unknown resistor, the **lower** the voltage at A0. Accuracy is best when R_x is close to R_{known} .

HANDS-ON Step 1 — Circuit Setup



— Unknown resistor between 5V and A0; known resistor between A0 and GND.

CONNECTION	DESCRIPTION
R _x (unknown)	Between 5V and A0
R _{known} (e.g. 1 kΩ)	Between A0 and GND
A0	Junction between R _x and R _{known}



TIP

Measure your known resistor with a multimeter first and type the **exact** value into the code — that alone improves accuracy.

HANDS-ON Step 2 — Arduino Code

OhmMeter.ino

● ARDUINO · C++

```
1 const int analogPin = A0;
2 const float Vin = 5.0;
3 const float R_known = 1000.0;    // known resistor in ohms
4
5 void setup() {
6   Serial.begin(9600);
7 }
8
9 void loop() {
10  int raw = analogRead(analogPin);
11  if (raw == 0) {
12    Serial.println(-1);           // nothing connected (open circuit)
13  } else {
14    float Vout = raw * Vin / 1023.0;
15    float Rx = R_known * (Vin / Vout - 1);
16    Serial.println(Rx);           // send resistance to Processing
17  }
18  delay(500);
19 }
```

UPDATED Fixed the `vconst` typo and added a guard for an open circuit (division by zero).

OhmDisplay.pde

PROCESSING · JAVA

```
1 import processing.serial.*;
2
3 Serial myPort;
4 float resistance = 0;
5
6 void setup() {
7   size(400, 200);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13   background(255);
14   fill(0);
15   textSize(20);
16   text("Measured Resistance:", 50, 80);
17   if (resistance < 0) text("no resistor", 50, 120);
18   else text(nf(resistance, 1, 2) + " \u03A9", 50, 120);
19 }
20
21 void serialEvent(Serial p) {
22   String val = p.readStringUntil('\n');
23   if (val != null) resistance = float(trim(val));
24 }
```

- Arduino reads the voltage at the midpoint of the divider.
- It calculates the unknown resistance with the rearranged formula.
- The value is sent over serial to Processing.
- Processing displays it in real time with units.

Student tasks

LEVEL	TASK	GOAL
Basic	Display the resistance value as text	
Intermediate	Colour the text based on the resistance range	
Advanced	Draw a dial / gauge visualization	
Challenge	Log multiple measurements and plot a graph	

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
Shows “no resistor” or a huge value	R _x not connected / open circuit	Check both legs of R _x are in the right rows.
Shows $\approx 0\ \Omega$	R _x shorted (both legs in the same row)	Place each leg in a different row.
Reading is inaccurate	R _{known} value in code is wrong	Measure R _{known} and update the constant.
Fluctuating values	Poor breadboard contact	Average 10 readings; press parts in firmly.

P4 • Exercises

P4.1 BASIC CALCULATE • 10 MIN

Derive the formula

Starting from $V_{A0} = V_{in} \cdot R_{known} / (R_x + R_{known})$, show step by step that $R_x = R_{known} \cdot (V_{in} / V_{A0} - 1)$.

Hint: Multiply both sides by $(R_x + R_{known})$, then divide by V_{A0} , then subtract R_{known} .

P4.2 BASIC CALCULATE • 10 MIN

From reading to ohms

With $R_{known} = 1\text{ k}\Omega$, Arduino reads **raw = 310**. Calculate V_{A0} and R_x . Which standard resistor is it probably? (1 k Ω , 2.2 k Ω , 3.3 k Ω , 4.7 k Ω)

P4.3 INTERMEDIATE DEBUG • 15 MIN

Debug the textbook

The first edition of this handbook said: “ $R1$ (known) between 5V and A0, $R2$ (unknown) between A0 and GND” but used the formula $R2 = R1 \cdot (V_{in}/V_{out} - 1)$. Using $R1 = 1\text{ k}\Omega$ and a real unknown of 2.2 k Ω , calculate what the meter would display. What is the correct formula for that wiring?

Hint: First find the real V_{out} for that wiring: $V_{out} = 5 \times 2200 / (1000 + 2200)$.

P4.4 INTERMEDIATE CODE • 15 MIN

Colour by range

Colour the displayed value: **green** below 1 k Ω , **orange** 1 k Ω –10 k Ω , **purple** above 10 k Ω . Also display large values in k Ω (e.g. 4.70 k Ω).

P4.5 ADVANCED CODE • 25 MIN

Analog gauge

Draw a semicircular gauge (like an old multimeter) from 0 to 10 k Ω with tick marks every 1 k Ω and a needle pointing at the current value.

Hint: Use `arc(cx, cy, d, d, PI, TWO_PI)`. Map resistance to an angle between PI and TWO_PI .

Accuracy investigation

Measure **at least 8 resistors** (100 Ω ... 100 k Ω) with your Arduino meter **and** a real multimeter. Calculate the percentage error for each. Plot error vs. resistance. When is your meter most accurate, and why? Propose an improvement (e.g. auto-ranging).

Self-check quiz

1. Ohm's law states...

- ☐ A $U = I / R$
- ☐ B $U = R \cdot I$
- ☐ C $R = U \cdot I$
- ☐ D $I = U \cdot R$

2. In this circuit, if R_x gets bigger, V_{A0} ...

- ☐ A gets bigger
- ☐ B gets smaller
- ☐ C stays at 2.5 V
- ☐ D becomes negative

3. If $R_x = R_{\text{known}}$, the reading on A0 is about...

- ☐ A 0
- ☐ B 256
- ☐ C 512
- ☐ D 1023

4. Why does the code check `raw == 0`?

- ☐ A To save power
- ☐ B To avoid dividing by zero when nothing is connected
- ☐ C Because the ADC starts at 1
- ☐ D To blink the LED

5. What does `nf(1234.5678, 1, 2)` return?

- ☐ A "1234"
- ☐ B "1234.57"
- ☐ C "1.23"
- ☐ D "1234.5678"

Reflection

- How does a commercial multimeter measure resistance? Is it similar to your circuit?
- What sources of error did you notice? Which could you reduce?
- Why is it important that a formula matches the actual circuit? Where else in science does this matter?

CONCLUSION

This project introduced electrical measurement, voltage dividers and real-time display. By combining the Arduino Uno with Processing you built a working ohm meter, practised rearranging formulas, and learned to question whether a formula really matches the circuit in front of you — a core skill for every engineer.

Automatic Irrigation System

A plant that waters itself — sensing, deciding and acting.

Measure soil moisture with a capacitive sensor, switch a water pump through a relay, and combine both into a closed-loop automatic irrigation system with a live Processing dashboard.

BY THE END OF THIS MODULE YOU CAN...

1. Explain how a capacitive soil moisture sensor works and why it does not corrode.
2. Explain what a relay is and why it is needed to switch a pump.
3. Use threshold logic to make automatic decisions in code.
4. Control an actuator manually from Processing and automatically from Arduino.
5. Improve a control system with hysteresis to prevent rapid on/off switching.

KEY VOCABULARY

Capacitive sensor: Measures how well the soil stores electric charge (permittivity) — water changes this a lot.

Permittivity: How strongly a material responds to an electric field. Water ≈ 80 , dry soil ≈ 4 .

Relay: An electrically operated switch: a small signal from Arduino switches a separate, more powerful circuit.

Active-LOW: A module that switches ON when its input is LOW (common for relay boards).

Threshold: A limit value at which the system changes its decision.

Hysteresis: Using two thresholds (switch on at one, off at another) so the system does not flicker.

Closed loop: A system that measures the result of its own action and corrects itself.

Required equipment

COMPONENT	QTY	PURPOSE
Arduino Uno	1	Microcontroller
Capacitive soil moisture sensor v1.2	1	Soil moisture measurement
Relay module (5 V)	1	Pump switching
Small submersible water pump (3–6 V)	1	Watering
Battery box / external supply matching the pump	1	Power for the pump
Silicone hose (50 cm)	1	Water delivery
Breadboard + jumper wires	1 set	Component connection
USB cable + Processing IDE	1	Connection & visualization

THEORY Theoretical Background



— Components: pump, hose, relay, capacitive sensor, battery box, wires.

How does the soil moisture sensor work?

The capacitive sensor outputs an analog voltage that **decreases as moisture increases**. Dry soil → high value (~1023 in air, ~600+ in dry soil). Wet soil → low value (~300). Its electrodes are covered, so unlike resistive sensors it does **not corrode** over time.



PHYSICS

Wet soil has a much higher **permittivity** than dry soil. This changes the capacitance of the sensor and therefore its output voltage.

The relay module

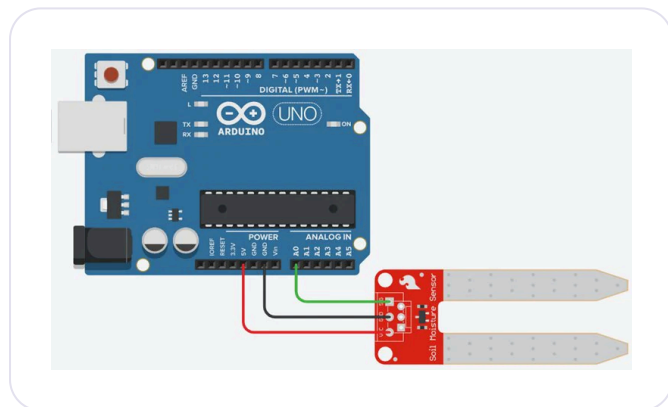
A relay is an electrically operated switch. A tiny 5 V signal from Arduino controls a separate pump circuit that would be too powerful for an Arduino pin. Many relay boards are **active-LOW**: input LOW → switch closes → pump ON; input HIGH → pump OFF. Always check your module!



WATER + ELECTRICITY SAFETY

Use only **low-voltage** batteries or adapters (≤ 12 V). Never connect anything to mains (230 V). Keep the Arduino, breadboard and laptop away from the water container, and dry your hands before touching the circuit. Wipe up spills immediately.

HANDS-ON Exercise 1 — Data Acquisition & Visualization



— Soil sensor wiring.

SENSOR PIN	CONNECTS TO
VCC	3.3V or 5V
GND	GND
AOUT	Arduino A0

SoilMoisture.ino

● ARDUINO · C++

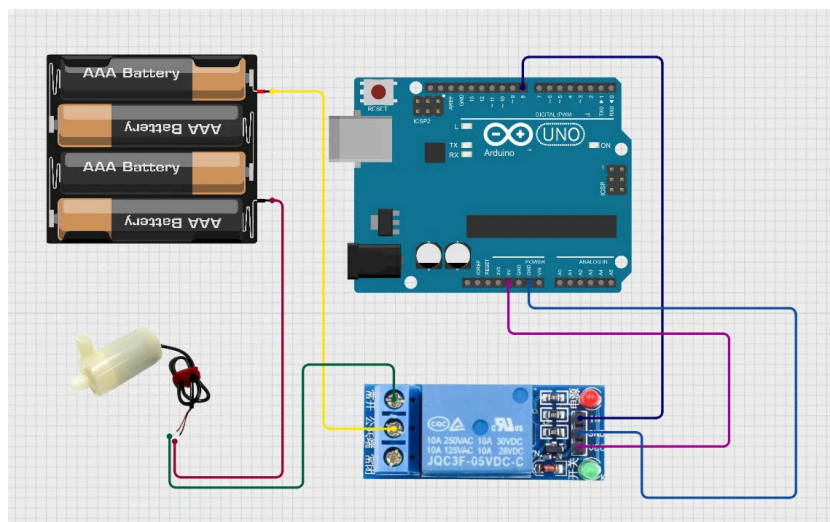
```
1 const int sensorPin = A0;
2
3 void setup() {
4   Serial.begin(9600);
5 }
6
7 void loop() {
8   int moisture = analogRead(sensorPin); // ~300 (wet) ... ~1023 (dry)
9   Serial.println(moisture);
10  delay(300);
11 }
```

```

1  import processing.serial.*;
2
3  Serial myPort;
4  int moisture = 0;
5
6  void setup() {
7      size(400, 200);
8      myPort = new Serial(this, Serial.list()[0], 9600);
9      myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13     background(255);
14     fill(0);
15     textSize(20);
16     text("Soil Moisture: " + moisture, 50, 60);
17
18     fill(0, 100, 255);          // full bar = wet
19     float w = map(moisture, 1023, 300, 0, 300); // reversed range: low value = wet
20     rect(50, 100, constrain(w, 0, 300), 30); // adjust 300/1023 after calibration
21 }
22
23 void serialEvent(Serial p) {
24     String val = p.readStringUntil('\n');
25     if (val != null) {
26         val = trim(val);
27         if (val.matches("\\d+")) moisture = int(val);
28     }
29 }

```

HANDS-ON Exercise 2 — Control the Water Pump from Processing



— Relay and pump wiring with an external battery pack.

CONNECTION	DESCRIPTION
Relay IN	Arduino D8
Relay VCC	Arduino 5V
Relay GND	Arduino GND
Relay COM	+ terminal of external power supply
Relay NO	Pump (+) terminal
Pump (–) terminal	Power supply GND

PumpControl.ino

● ARDUINO · C++

```

1  const int RELAY_PIN = 8;
2  const int PUMP_ON  = LOW;    // active-LOW relay module
3  const int PUMP_OFF = HIGH;   // swap these if your module is active-HIGH
4
5  void setup() {
6    Serial.begin(9600);
7    pinMode(RELAY_PIN, OUTPUT);
8    digitalWrite(RELAY_PIN, PUMP_OFF); // make sure the pump starts OFF
9  }
10
11 void loop() {
12   if (Serial.available() > 0) {
13     char cmd = Serial.read();
14     if (cmd == '1')    digitalWrite(RELAY_PIN, PUMP_ON);
15     else if (cmd == '0') digitalWrite(RELAY_PIN, PUMP_OFF);
16     // any other character (e.g. newline) is ignored
17   }
18 }

```

UPDATED The original used pin 9 while the wiring table uses D8. Now consistently D8, with named ON/OFF constants.

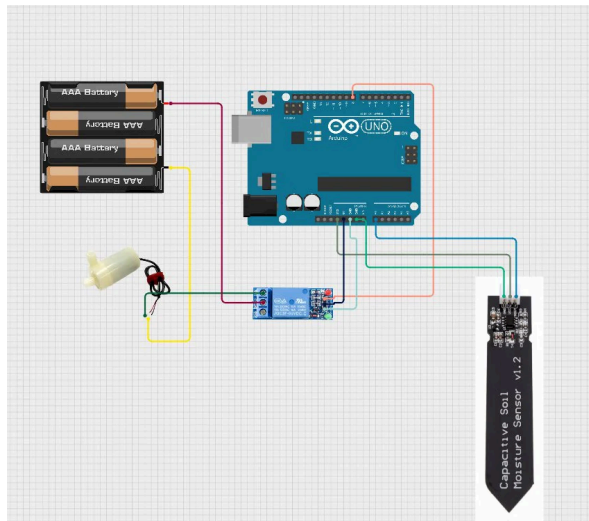
```
1 import processing.serial.*;
2
3 Serial myPort;
4
5 void setup() {
6   size(360, 100);
7   myPort = new Serial(this, Serial.list()[0], 9600);
8 }
9
10 void draw() {
11   background(200);
12   fill(0);
13   text("Press '1' to turn ON pump, '0' to turn OFF", 20, 50);
14 }
15
16 void keyPressed() {
17   if (key == '1' || key == '0') myPort.write(key);
18 }
```

PROJECT Exercise 3 — Full Automatic Irrigation System



LEVEL UP

This exercise is suitable as a complete project. **Goal:** combine sensing and pump control so watering happens automatically when the soil is too dry, with a real-time Processing display.



— Complete system: soil sensor on A0, relay on D8, pump on the external supply.

```
1 #define RELAY_PIN 8
2 #define SENSOR_PIN A0
3 #define THRESHOLD 600 // above = too dry
4
5 const int PUMP_ON = LOW; // active-LOW relay
6 const int PUMP_OFF = HIGH;
7
8 void setup() {
9   Serial.begin(9600);
10  pinMode(RELAY_PIN, OUTPUT);
11  digitalWrite(RELAY_PIN, PUMP_OFF);
12 }
13
14 void loop() {
15   int moisture = analogRead(SENSOR_PIN);
16   Serial.println(moisture);
17
18   if (moisture > THRESHOLD) digitalWrite(RELAY_PIN, PUMP_ON); // dry → water
19   else digitalWrite(RELAY_PIN, PUMP_OFF); // wet → stop
20   delay(1000);
21 }
```

UPDATED Uses the same active-LOW constants as Exercise 2 — the original switched the relay the opposite way.

```
1 import processing.serial.*;
2
3 Serial myPort;
4 int moisture = 0;
5
6 void setup() {
7   size(400, 200);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13   background(255);
14   fill(0);
15   textSize(20);
16   text("Soil Moisture: " + moisture, 50, 100);
17
18   if (moisture > 600) {
19     fill(255, 0, 0);
20     text("Status: DRY \u2013 Watering", 50, 140);
21   } else {
22     fill(0, 150, 0);
23     text("Status: WET \u2013 Idle", 50, 140);
24   }
25 }
26
27 void serialEvent(Serial p) {
28   String val = p.readStringUntil('\n');
29   if (val != null && trim(val).matches("\\d+")) moisture = int(trim(val));
30 }
```

Student tasks

LEVEL	TASK	GOAL
Basic	Display the moisture value and status text	
Intermediate	Colour-coded moisture bar (blue = wet, red = dry)	
Advanced	Add a manual pump override button in Processing	
Challenge	Log moisture data with timestamps to a CSV file	

Troubleshooting

PROBLEM	LIKELY CAUSE	FIX
Pump does not turn on	Relay wiring incorrect or pin mismatch	Relay IN must match RELAY_PIN (8). Listen for the relay "click".
Pump is ON when it should be OFF	Relay is active-HIGH, not active-LOW	Swap the PUMP_ON / PUMP_OFF constants.
Sensor always reads max	Sensor not in soil or damaged	Insert up to the line; never submerge the electronics.
Pump runs continuously	Threshold too low for your soil	Calibrate: measure dry and wet values and choose a threshold between them.
Relay clicks on/off rapidly	Value hovers around the threshold	Add hysteresis (exercise P5.4).
Processing shows no data	Wrong COM port or baud rate	<code>printArray(Serial.list())</code> and use 9600 on both sides.

P5 • Exercises

P5.1 BASIC CALCULATE • 10 MIN

Calibrate to percent

Your sensor reads **820 in dry soil** and **340 in freshly watered soil**. Convert a reading of **580** into a moisture percentage (0 % = dry, 100 % = wet). Write the Processing `map()` call.

P5.2 BASIC EXPLAIN • 5 MIN

Why a relay?

An Arduino pin can supply about **20 mA** at 5 V. A small pump needs about **200 mA**. Explain why we can't connect the pump directly to a pin, and what the relay does instead.

P5.3 INTERMEDIATE CODE • 20 MIN

Manual override button

Add an on-screen **WATER NOW** button to the Processing dashboard that sends `1` while held and `0` when released. Show the pump state with an icon or colour.

Hint: Use `mousePressed()` and `mouseReleased()`; check the mouse is inside the button rectangle.

P5.4 ADVANCED CODE • 20 MIN

Add hysteresis

With a single threshold the relay can click on and off every second when the value hovers around 600. Change the Arduino code so the pump turns **ON above 650** and only turns **OFF below 450**. Explain why this is better for the pump and the plant.

P5.5 ADVANCED DEBUG • 10 MIN

The thirsty pump

A team's pump runs **all the time**, even in wet soil, and stops only when the soil is dry. Their code is the original Exercise 3 with `HIGH` = water ON. What is wrong, and how do you prove it with a single test?

Water-saving study

Run two identical plants for one week: one watered by hand daily, one by your automatic system. Measure the **water used** (fill the reservoir to a mark, measure refills) and log moisture. Present which method uses less water and keeps moisture more stable.

1. Define fair-test conditions (same plant, pot, soil, location).
2. Log moisture every 10 minutes to CSV.
3. Record water volume used per day.
4. Present a chart and a conclusion with numbers.

Self-check quiz

1. As soil gets wetter, the capacitive sensor's value...

- ☐ A increases
- ☐ B decreases
- ☐ C stays the same
- ☐ D becomes negative

2. What is the main job of the relay?

- ☐ A Measure moisture
- ☐ B Let a small Arduino signal switch the pump's separate power circuit
- ☐ C Store water
- ☐ D Increase the voltage of the Arduino pin

3. An active-LOW relay module turns ON when its input is...

- ☐ A HIGH
- ☐ B LOW
- ☐ C floating
- ☐ D PWM

4. What problem does hysteresis solve?

- ☐ A The sensor corroding
- ☐ B Rapid on/off switching near the threshold
- ☐ C Wrong baud rate
- ☐ D Low battery

5. Which is the safest power choice for the pump in a classroom?

- ☐ A Mains 230 V
- ☐ B A low-voltage battery pack matching the pump
- ☐ C The Arduino 5V pin directly
- ☐ D A car battery

Reflection

- Your system is a "closed loop". Find two other closed-loop systems in everyday life.
- What could go wrong if the sensor fails while the pump is ON? How could you design a safety timeout?

- How could automatic irrigation help farmers in regions affected by drought?

CONCLUSION

You built a practical automatic system that senses, decides and acts. Combining a soil moisture sensor, a relay and Processing, you gained hands-on experience in electronics, programming and environmental monitoring — the same principles behind smart greenhouses and precision agriculture that help conserve water worldwide.

Appendix A — Answer key

Try every exercise before you look here. For open-ended design tasks, the key describes what a strong answer contains.

F1 • First Steps with Arduino

F1.1 Predict the rhythm

The LED flashes briefly (on for 0.2 s, off for 0.8 s). One cycle lasts $200 + 800 = 1000 \text{ ms} = 1 \text{ s}$, so it flashes **60 times per minute**. The duty cycle (time on ÷ total time) is 20 %.

F1.2 Choose the resistor

$R = (5.0 - 3.0) / 0.010 = 200 \Omega$. The closest standard value that is **not smaller** is **220 Ω** — the current will then be $(2.0 / 220) \approx 9.1 \text{ mA}$, which is safe.

F1.3 SOS beacon

SOS.ino

● ARDUINO · C++

```
1 const int ledPin = 8;
2
3 void flash(int ms) {
4   digitalWrite(ledPin, HIGH);
5   delay(ms);
6   digitalWrite(ledPin, LOW);
7   delay(200);
8 }
9
10 void setup() {
11   pinMode(ledPin, OUTPUT);
12 }
13
14 void loop() {
15   for (int i = 0; i < 3; i++) flash(200); // S
16   for (int i = 0; i < 3; i++) flash(600); // O
17   for (int i = 0; i < 3; i++) flash(200); // S
18   delay(2000);
19 }
```

F1.4 Find three bugs

1) Missing semicolon after `const int ledPin = 8`. 2) `pinMode` must be `OUTPUT`, not `INPUT`. 3) `ledpin` ≠ `ledPin` — names are case-sensitive.

F1.5 Push-button lamp

ButtonLamp.ino

● ARDUINO · C++

```
1 const int ledPin = 8;
2 const int buttonPin = 2;
3
4 void setup() {
5   pinMode(ledPin, OUTPUT);
6   pinMode(buttonPin, INPUT_PULLUP);
7 }
8
9 void loop() {
10  bool pressed = digitalRead(buttonPin) == LOW;
11  digitalWrite(ledPin, pressed ? HIGH : LOW);
12 }
```

F1.6 Traffic light controller

TrafficLight.ino

● ARDUINO · C++

```
1 const int RED = 10, YELLOW = 9, GREEN = 8;
2
3 void lights(bool r, bool y, bool g, int ms) {
4   digitalWrite(RED, r);
5   digitalWrite(YELLOW, y);
6   digitalWrite(GREEN, g);
7   delay(ms);
8 }
9
10 void setup() {
11   pinMode(RED, OUTPUT);
12   pinMode(YELLOW, OUTPUT);
13   pinMode(GREEN, OUTPUT);
14 }
15
16 void loop() {
17   lights(0, 0, 1, 4000); // green
18   lights(0, 1, 0, 1000); // yellow
19   lights(1, 0, 0, 4000); // red
20   lights(1, 1, 0, 1000); // red + yellow
21 }
```

Quiz: 1-B 2-B 3-B 4-B 5-C

F2 • Processing — Real-Time Visualization

F2.1 map() by hand

a) $512 \times 300 / 1023 \approx 150.1$ px · b) $25 \times 255 / 50 = 127.5$ · c) $(750 - 1023) \times 100 / (300 - 1023) = (-273) (100) / (-723) \approx 37.8$ %

F2.2 Where will it appear?

The first circle is exactly in the **centre**. The rectangle is a 20 px strip along the **bottom** edge. The last circle is centred on the **top-right corner**, so only a quarter of it is visible.

F2.3 Bouncing ball

Bounce.pde

• PROCESSING • JAVA

```
1 float x = 100, y = 100, dx = 3, dy = 2;
2 color c = color(61, 245, 255);
3
4 void setup() {
5   size(500, 300);
6   noStroke();
7 }
8
9 void draw() {
10  background(20);
11  x += dx;
12  y += dy;
13  if (x < 15 || x > width - 15) { dx = -dx; c = color(random(255), random(255),
random(255)); }
14  if (y < 15 || y > height - 15) { dy = -dy; c = color(random(255), random(255),
random(255)); }
15  fill(c);
16  ellipse(x, y, 30, 30);
17 }
```

F2.4 Why is the screen blank?

1) Baud rate is **115200** but Arduino uses **9600**. 2) No `background()` in `draw()`, so numbers pile on top of each other (and default fill is drawn on grey — text may be unreadable). 3) `readStringUntil()` can return **null** — check before `trim()`. Adding `myPort.bufferUntil('\n')` in `setup()` also helps.

F2.5 Real-time line graph

LiveGraph.pde

● PROCESSING · JAVA

```
1 import processing.serial.*;
2
3 Serial myPort;
4 int[] vals = new int[100];
5
6 void setup() {
7   size(600, 300);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13   background(15);
14   stroke(61, 245, 255);
15   strokeWeight(2);
16   noFill();
17   beginShape();
18   for (int i = 0; i < vals.length; i++) {
19     float x = map(i, 0, vals.length - 1, 20, width - 20);
20     float y = map(vals[i], 0, 100, height - 20, 20); // reversed: bigger = higher
21     vertex(x, y);
22   }
23   endShape();
24 }
25
26 void serialEvent(Serial p) {
27   String s = p.readStringUntil('\n');
28   if (s == null) return;
29   s = trim(s);
30   if (!s.matches("\\d+")) return;
31   for (int i = 0; i < vals.length - 1; i++) vals[i] = vals[i + 1];
32   vals[vals.length - 1] = int(s);
33 }
```

F2.6 Three-value protocol

ThreeValues.pde (receiver)

● PROCESSING · JAVA

```
1 import processing.serial.*;
2 Serial myPort;
3 float px = 0, py = 0;
4 boolean pressed = false;
5
6 void setup() {
7   size(500, 500);
8   myPort = new Serial(this, Serial.list()[0], 9600);
9   myPort.bufferUntil('\n');
10 }
11
12 void draw() {
13   background(20);
14   fill(pressed ? color(198, 255, 61) : color(109, 74, 255));
15   float x = map(px, 0, 1023, 25, width - 25);
16   float y = map(py, 0, 1023, 25, height - 25);
17   ellipse(x, y, 50, 50);
18 }
19
20 void serialEvent(Serial p) {
21   String line = p.readStringUntil('\n');
22   if (line == null) return;
23   String[] v = split(trim(line), ",");
24   if (v.length != 3) return;
25   for (String s : v) if (!s.matches("\\d+")) return; // reject malformed lines
26   px = float(v[0]);
27   py = float(v[1]);
28   pressed = v[2].equals("1");
29 }
```

Quiz: 1-B 2-C 3-B 4-C 5-B 6-B

P1 • Temperature & Humidity — DHT11

P1.1 Read the datasheet

Real temperature is between **20 °C and 24 °C**, humidity between **43 % and 53 %**. A freezer (≈ -18 °C) is **outside** the 0–50 °C range, so the DHT11 is not suitable — a DHT22 (–40...80 °C) would be.

P1.2 Parse it in your head

25.0,40.0 → temp 25.0, hum 40.0. · Error line → **unchanged** (no comma). · 26.1, → split gives ["26.1", ""], length 2, so temp = 26.1 and hum = **NaN** — a hidden bug! · 27.3,55.0,1 → length 3 → **unchanged**. Improvement: also check that both parts are not empty.

P1.3 Comfort zone indicator

Add to draw()

● PROCESSING · JAVA

```
1 boolean tempOk = temp >= 20 && temp <= 24;
2 boolean humOk = hum >= 40 && hum <= 60;
3 int bad = (tempOk ? 0 : 1) + (humOk ? 0 : 1);
4
5 if (bad == 0) fill(0, 200, 90);
6 else if (bad == 1) fill(255, 200, 0);
7 else fill(230, 40, 40);
8 ellipse(440, 40, 40, 40);
9
10 fill(0);
11 textSize(14);
12 text(bad == 0 ? "Comfortable" : bad == 1 ? "Check climate" : "Uncomfortable", 330, 80);
```

P1.4 Dew point estimate

(a) $25 - 40/5 = 17\text{ °C}$ · (b) $18 - 10/5 = 16\text{ °C}$ — very close to the air temperature, so fog or condensation on windows is likely. In Processing: `float dew = temp - (100 - hum) / 5.0;`

P1.5 Log to a CSV file

Logging additions

● PROCESSING · JAVA

```
1 PrintWriter output;
2
3 // in setup():
4 output = createWriter("weather.csv");
5 output.println("time,temperature,humidity");
6
7 // in serialEvent(), after parsing successfully:
8 String stamp = nf(hour(), 2) + ":" + nf(minute(), 2) + ":" + nf(second(), 2);
9 output.println(stamp + "," + temp + "," + hum);
10
11 void keyPressed() {
12     if (key == 's') { // press S to save and stop
13         output.flush();
14         output.close();
15         exit();
16     }
17 }
```

P1.6 Classroom climate study

Open-ended. A good report includes: a clear hypothesis, a description of the method (sensor position, interval), a chart with labelled axes and units, averages compared in a table, a discussion of sensor accuracy ($\pm 2\text{ °C}$ / $\pm 5\%$) and a practical recommendation.

Quiz: 1-B 2-B 3-B 4-C 5-B

P2 • Light Sensor — Photoresistor (LDR)

P2.1 Voltage divider practice

(a) $5 \times 10/11 = 4.55 \text{ V} \rightarrow \approx 930$ · (b) $5 \times 10/20 = 2.50 \text{ V} \rightarrow \approx 512$ · (c) $5 \times 10/110 = 0.45 \text{ V} \rightarrow \approx 93$

P2.2 Swap the parts

The behaviour is **inverted**: bright light now gives a **low** reading and darkness a **high** one, because $V_{A0} = 5 \times R_{LDR} / (R_{LDR} + R_{\text{fixed}})$.

P2.3 Automatic night-light

NightLight.ino

● ARDUINO · C++

```
1 const int sensorPin = A0;
2 const int ledPin = 9;
3 const int THRESHOLD = 300;
4
5 void setup() {
6   Serial.begin(9600);
7   pinMode(ledPin, OUTPUT);
8 }
9
10 void loop() {
11   int light = analogRead(sensorPin);
12   bool dark = light < THRESHOLD;
13   digitalWrite(ledPin, dark ? HIGH : LOW);
14   Serial.println(light);
15   delay(200);
16 }
```

P2.4 Smooth dimming with PWM

solution

● ARDUINO · C++

```
1 void loop() {
2   int light = analogRead(A0);
3   int brightness = map(light, 0, 1023, 255, 0);
4   analogWrite(9, constrain(brightness, 0, 255));
5   Serial.println(light);
6   delay(50);
7 }
```

P2.5 Calibrate with min/max

Calibration additions

PROCESSING · JAVA

```
1 int lo = 1023, hi = 0;
2
3 // in serialEvent(), after lightLevel is updated:
4 lo = min(lo, lightLevel);
5 hi = max(hi, lightLevel);
6
7 // in draw():
8 float w = (hi > lo) ? map(lightLevel, lo, hi, 0, 300) : 0;
9 rect(50, 100, w, 30);
10 text("calibrated: " + lo + " - " + hi, 50, 170);
11
12 void keyPressed() {
13   if (key == 'r' || key == 'R') { lo = 1023; hi = 0; }
14 }
```

P2.6 Sunrise logger

Open-ended. Use `delay(60000)` or `millis()` on Arduino (or a timer in Processing), log with `PrintWriter`, and plot in Processing or a spreadsheet. Discuss clouds, blinds, artificial lighting, and the direction the window faces.

Quiz: 1-B 2-B 3-B 4-C 5-B

P3 · Sonar — Ultrasonic Sensor HC-SR04

P3.1 Echo arithmetic

(a) $1166 \times 0.0343 / 2 \approx 20.0 \text{ cm}$ · (b) $t = 2 \times 200 / 0.0343 \approx 11\,662 \mu\text{s} \approx 11.7 \text{ ms}$

P3.2 Why divide by two?

Every reading will be **exactly twice** the real distance, because the measured time covers the path **to** the object **and back**.

P3.3 Temperature matters

0 °C: $v = 331.3 \text{ m/s} \rightarrow \text{reported} = 100 \times 343 / 331.3 \approx 103.5 \text{ cm}$ (+3.5 %). · 35 °C: $v \approx 352.5 \text{ m/s} \rightarrow \text{reported} \approx 97.3 \text{ cm}$ (-2.7 %). Combine with a DHT11 (Project 1) to correct it!

P3.4 Text in the numbers

`float("distance: 23.4 cm")` returns **NaN** because the text isn't a number. **Fix A (Arduino):** send only the number: `Serial.println(distance_cm);` · **Fix B (Processing):** extract it: `String[] m = match(data, "[0-9.]+"); if (m != null) distance = float(m[1]);`

P3.5 Polar to screen

30°, 60 cm: $r = 150 \rightarrow x = 300 + 150 \cdot 0.866 \approx 430$, $y = 600 - 150 \cdot 0.5 = 525$. · 135°, 100 cm: $r = 250 \rightarrow x = 300 + 250 \cdot (-0.707) \approx 123$, $y = 600 - 250 \cdot 0.707 \approx 423$.

P3.6 Full radar display

```
1 import processing.serial.*;
2
3 Serial myPort;
4 int angle = 0;
5 float distance = 0;
6 final float MAX_CM = 100;
7 float R; // radius in px
8
9 void setup() {
10   size(800, 450);
11   R = width / 2 - 20;
12   myPort = new Serial(this, Serial.list()[0], 9600);
13   myPort.bufferUntil('\n');
14   background(0);
15 }
16
17 void draw() {
18   noStroke();
19   fill(0, 18); // fading trail
20   rect(0, 0, width, height);
21
22   translate(width / 2, height - 10); // radar centre at the bottom
23   drawGrid();
24
25   stroke(61, 255, 154); // sweep line
26   strokeWeight(3);
27   line(0, 0, R * cos(radians(angle)), -R * sin(radians(angle)));
28
29   if (distance > 0 && distance < MAX_CM) {
30     float r = map(distance, 0, MAX_CM, 0, R);
31     if (distance < 30) fill(255, 60, 60);
32     else if (distance < 60) fill(255, 210, 60);
33     else fill(61, 255, 154);
34     noStroke();
35     ellipse(r * cos(radians(angle)), -r * sin(radians(angle)), 10, 10);
36   }
37
38   resetMatrix();
39   fill(61, 255, 154);
40   textSize(16);
41   text("Angle: " + angle + "\u00B0 Distance: " + nf(distance, 0, 1) + " cm", 20, 30);
42 }
43
44 void drawGrid() {
45   noFill();
46   stroke(61, 255, 154, 90);
47   strokeWeight(1);
48   for (int cm = 25; cm <= MAX_CM; cm += 25) {
49     float d = map(cm, 0, MAX_CM, 0, R) * 2;
50     arc(0, 0, d, d, PI, TWO_PI);
51   }
52   for (int a = 0; a <= 180; a += 30) {
53     line(0, 0, R * cos(radians(a)), -R * sin(radians(a)));
54   }
```

```

55 }
56
57 void serialEvent(Serial p) {
58   String data = p.readStringUntil('\n');
59   if (data == null) return;
60   String[] parts = split(trim(data), ",");
61   if (parts.length != 2) return;
62   angle = int(parts[0]);
63   distance = float(parts[1]);
64 }

```

Quiz: 1-C 2-B 3-B 4-B 5-B

P4 • Arduino Ohm Meter with Processing Display

P4.1 Derive the formula

$V_{A0} (R_x + R_{\text{known}}) = V_{\text{in}} \cdot R_{\text{known}} \rightarrow R_x + R_{\text{known}} = V_{\text{in}} \cdot R_{\text{known}} / V_{A0} \rightarrow R_x = R_{\text{known}} \cdot V_{\text{in}} / V_{A0} - R_{\text{known}} = R_{\text{known}} (V_{\text{in}} / V_{A0} - 1) \checkmark$

P4.2 From reading to ohms

$V_{A0} = 310 \times 5 / 1023 \approx 1.515 \text{ V} \rightarrow R_x = 1000 \times (5 / 1.515 - 1) \approx 2300 \Omega \rightarrow$ most likely a **2.2 k Ω** resistor (within $\pm 5 \%$).

P4.3 Debug the textbook

$V_{\text{out}} = 5 \times 2200 / 3200 = 3.44 \text{ V}$. Wrong formula: $1000 \times (5 / 3.44 - 1) \approx 455 \Omega$ — completely wrong! For that wiring the correct formula is **$R_2 = R_1 \cdot V_{\text{out}} / (V_{\text{in}} - V_{\text{out}})$** = $1000 \times 3.44 / 1.56 \approx 2200 \Omega \checkmark$. Lesson: always check that the formula matches the circuit.

P4.4 Colour by range

solution

• PROCESSING • JAVA

```

1 String label;
2 if (resistance < 1000) {
3   fill(0, 170, 80);
4   label = nf(resistance, 1, 1) + " \u03A9";
5 } else {
6   fill(resistance < 10000 ? color(255, 140, 0) : color(130, 60, 200));
7   label = nf(resistance / 1000.0, 1, 2) + " k\u03A9";
8 }
9 textSize(36);
10 text(label, 50, 140);

```

P4.5 Analog gauge

Gauge.pde (draw part)

● PROCESSING · JAVA

```
1 void drawGauge(float value, float maxValue) {
2   float cx = width / 2, cy = height - 30, r = 150;
3   noFill();
4   stroke(40);
5   strokeWeight(3);
6   arc(cx, cy, r * 2, r * 2, PI, TWO_PI);
7
8   strokeWeight(1);
9   for (int i = 0; i <= 10; i++) {           // ticks
10    float a = map(i, 0, 10, PI, TWO_PI);
11    line(cx + cos(a) * (r - 12), cy + sin(a) * (r - 12),
12         cx + cos(a) * r,      cy + sin(a) * r);
13  }
14
15  float a = map(constrain(value, 0, maxValue), 0, maxValue, PI, TWO_PI);
16  stroke(220, 40, 40);                      // needle
17  strokeWeight(4);
18  line(cx, cy, cx + cos(a) * (r - 20), cy + sin(a) * (r - 20));
19 }
```

P4.6 Accuracy investigation

Expected result: smallest error when $R_x \approx R_{\text{known}}$; errors grow for very small or very large R_x , because V_{A0} approaches 5 V or 0 V where one ADC step (≈ 4.9 mV) represents a large change in resistance. Improvement: switch between several known resistors (auto-ranging) and average multiple readings.

Quiz: 1-B 2-B 3-C 4-B 5-B

P5 • Automatic Irrigation System

P5.1 Calibrate to percent

$(820 - 580) / (820 - 340) \times 100 = 240 / 480 \times 100 = 50 \%$. In code: `float pct = constrain(map(moisture, 820, 340, 0, 100), 0, 100);`

P5.2 Why a relay?

The pump needs $\sim 10\times$ more current than the pin can safely give — it could damage the Arduino. The relay lets the weak Arduino signal **switch** a separate, stronger power supply, while the two circuits stay electrically separated.

P5.3 Manual override button

solution

PROCESSING • JAVA

```
1 boolean pumping = false;
2
3 // in draw():
4 fill(pumping ? color(61, 155, 255) : color(180));
5 rect(50, 160, 140, 30, 6);
6 fill(255);
7 text("WATER NOW", 65, 182);
8
9 void mousePressed() {
10     if (mouseX > 50 && mouseX < 190 && mouseY > 160 && mouseY < 190) {
11         myPort.write('1');
12         pumping = true;
13     }
14 }
15
16 void mouseReleased() {
17     if (pumping) {
18         myPort.write('0');
19         pumping = false;
20     }
21 }
```

P5.4 Add hysteresis

Between 450 and 650 the system keeps its previous state, so it doesn't flicker. This reduces wear on the relay and pump, and waters the plant thoroughly instead of in tiny bursts. A home thermostat works the same way.

Hysteresis.ino (loop)

ARDUINO • C++

```
1 const int DRY_ON = 650;
2 const int WET_OFF = 450;
3 bool pumpOn = false;
4
5 void loop() {
6     int moisture = analogRead(SENSOR_PIN);
7     Serial.println(moisture);
8
9     if (!pumpOn && moisture > DRY_ON) pumpOn = true;
10    if (pumpOn && moisture < WET_OFF) pumpOn = false;
11
12    digitalWrite(RELAY_PIN, pumpOn ? PUMP_ON : PUMP_OFF);
13    delay(1000);
14 }
```

P5.5 The thirsty pump

Their relay is **active-LOW**, so **HIGH** actually switches the pump OFF and **LOW** switches it ON — the logic is inverted. Test: upload a sketch that only does `digitalWrite(8, HIGH)` and listen/look: if the pump is off, the module is active-LOW. Fix by swapping the constants.

P5.6 Water-saving study

Open-ended. Strong reports discuss variables controlled, show a moisture-vs-time chart for both plants, compare total water (ml), and reflect on threshold choice, evaporation and sensor placement.

Quiz: 1-B 2-B 3-B 4-B 5-B

Appendix B — Teacher notes & errata

This edition keeps every lab from the original Digital Challenge handbook. All sketches were reviewed; the table below lists every change to the original code or text so that existing printed copies can be corrected.

Suggested scheme of work

MODULE	LESSONS	ASSESSMENT IDEA
F1 Arduino Basics	2 × 45 min	Traffic light controller
F2 Processing Basics	3 × 45 min	Three-value protocol
P1 Weather Station	2 × 45 min	Classroom climate study
P2 Light Meter	2 × 45 min	Sunrise logger
P3 Radar Scanner	4 × 45 min	Full radar display
P4 Ohm Meter	2 × 45 min	Accuracy investigation
P5 Smart Garden	4 × 45 min	Water-saving study

Errata

WHERE	ISSUE IN THE ORIGINAL	WHAT CHANGED
Processing · Lab 1	<code>int frameCount = 0;</code> redeclares Processing's built-in <code>frameCount</code> variable.	Uses a separate counter <code>t</code> .
Project 1 · Processing	Sketch was missing <code>import processing.serial.*;</code> and contained HTML entities (<code>&apos;</code>).	Import added; reading moved to <code>serialEvent()</code> ; bars and alarm included.
Project 2 · Theory	Callout said "R2 is the LDR", but in the wiring the LDR is on the 5V side.	Formula rewritten as $V_{A0} = 5\text{ V} \times R_{\text{fixed}} / (R_{\text{LDR}} + R_{\text{fixed}})$.
Project 3 · Ex. 2	Servo sweep used blocking for-loops, so STOP was ignored until a sweep finished.	Non-blocking "one step per loop()" sweep.
Project 3 · Ex. 1 & 4	<code>pulseIn()</code> without timeout can freeze for 1 s when there is no echo.	Timeout of 30 000 µs added.
Project 4 · Wiring	Wiring table (known resistor on the 5V side) contradicted the diagram and formula.	Table follows the diagram: unknown 5V→A0, known A0→GND.
Project 4 · Arduino	Typo <code>vconst</code> ; division by zero when no resistor is connected.	Typo fixed; open-circuit guard sends -1.
Project 5 · Ex. 2	Wiring used D8, code used pin 9.	Consistently D8 with named constants.
Project 5 · Ex. 3	Relay logic (HIGH = ON) contradicted Ex. 2 and the active-LOW explanation.	<code>PUMP_ON</code> / <code>PUMP_OFF</code> constants used everywhere.
Project 5 · Equipment	12 V supply listed, while the kit pump/battery box is low-voltage.	Supply must match the pump rating; safety box added.



Co-funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them. Project 2024-2-RS01-KA210-SCH-000271542.