



Digital
Challenge

2024-2-RS01-KA210-SCH-000271542



Co-funded by
the European Union



Digital
Challenge



Co-funded by
the European Union

PROJECT DOCUMENT

Digital Challenge

Cover page

PROJECT NUMBER

2024-2-RS01-KA210-SCH-000271542



Table of Contents

Introduction: First Steps in Microcontroller Programming with Arduino	4
Development Environments: Arduino IDE, Tinkercad, and Circuit Designer	7
Arduino IDE	7
Tinkercad & Circuit Designer	7
Sensors and Actuators Used in Later Projects	7
Exercise 1: First Program Upload — Blink	8
Exercise 2: Wiring an External LED on a Breadboard	9
Introduction: Processing — Real-Time Visualization for Arduino Projects	13
The Processing Development Environment (PDE)	17
Exercise 1: Your First Processing Sketch	23
Exercise 2: Receiving Data from Arduino	24
Exercise 3: Parsing Multiple Values (CSV Format)	26
Exercise 4: Interactive Processing – Sending Commands Back to Arduino	28
Project 1: Temperature & Humidity — DHT11	35
Exercise 1: Connecting the DHT11 Sensor	36
Exercise 2: Data Visualization in Processing	38
Project 2: Light Sensor — Photoresistor (LDR)	41
Exercise 1: Circuit Setup & Arduino Code	42
Exercise 2: Processing Visualization	43
Project 3: Sonar — Ultrasonic Sensor HC-SR04	46
Exercise 1: Measure Distance & Display in Processing	47
Exercise 2: Control SG90 Servo from Processing	49
Exercise 3: Ultrasonic Sensor Controlling Servo	52
Exercise 4: Build a Radar Scanner (Sonar)	54
Project 4: Arduino Ohm Meter with Processing Display	57
Project 5: Automatic Irrigation System	61
Exercise 1: Data Acquisition & Visualization	63
Exercise 2: Control Water Pump from Processing	65
Exercise 3: Full Automatic Irrigation System	67
Hackathon Challenges & Applied Learning Integration	73
Pedagogical and Methodological Framework	81

Learn by Example

Basic Exercises for Arduino + Processing

Digital Challenge Workshop

#	Project	Topics Covered
1	Temperature & Humidity — DHT11	Sensors, serial comm., Processing visualization
2	Light Sensor — LDR	Analog input, voltage divider, bar chart
3	Sonar — Ultrasonic Sensor HC-SR04	Distance, servo control, radar display
4	Ohm Meter	Voltage divider, Ohm's law, resistance display
5	Automatic Irrigation System	Soil moisture, relay, pump control

Introduction: First Steps in Microcontroller Programming with Arduino

Description

In this lesson, students are introduced to the Arduino Uno board, the basic elements of a microcontroller system, and the first steps in working with electronic circuits. The focus is on connecting software with hardware, as this is the foundation of every STEM project — from idea, through simulation, to real-world assembly.

The lesson is prepared for Erasmus+ STEM documentation in a way that encourages exploratory learning, practical application of knowledge, and understanding of how a simple program can control a real device.

This module serves as the introductory foundation for all subsequent sensor and Processing projects in the Digital Challenge workshop series.

Objectives

Students will:

- understand what Arduino is and what a microcontroller board is used for,
- distinguish the basic parts of the Arduino Uno board,
- learn how to use a breadboard and resistor in a simple circuit,
- connect programming code with real physical component wiring,
- upload their first program using the Arduino IDE environment,
- develop the habit of first checking the circuit diagram before performing real wiring.

Learning Outcomes

After the lesson, students will be able to:

- explain the basic purpose of the Arduino platform,
- identify the most important parts of the Arduino Uno board,
- assemble a simple circuit on a breadboard,
- run and modify a basic Arduino program,
- use concepts such as `pinMode`, `digitalWrite`, and `delay`,
- connect a simulation and the real construction of the same circuit.

Required Equipment

Component	Quantity	Purpose
Arduino Uno	1	Main development board
USB cable	1	Connection to computer and program upload
Breadboard	1	Quick connection without soldering

Component	Quantity	Purpose
LED diode	1–2	Visual indicator of program execution
220Ω resistor	1–2	Current limiting for LED
Jumper wires	~4	Connecting components
Arduino IDE	1	Writing and uploading programs
Tinkercad or Circuit Designer	1	Circuit simulation and verification

Theoretical Background

What is Arduino?

Arduino is a hardware and software platform based on a microcontroller. In practice, this means that a small board can read signals from sensors, process them, and then control LEDs, motors, relays, displays, or other devices.

Due to its simple wiring, large number of examples, and extensive library support, Arduino is very suitable for beginners, as well as for rapid prototyping in school and project activities.

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

Why Arduino Uno?

Arduino Uno is the most common choice for first exercises because it is stable, well documented, and easy to use. It has enough input/output pins for school experiments while not overwhelming the student with unnecessary complexity.

Once the basics are learned on the Uno board, moving to other models such as Nano, Mega, or ESP32 becomes much easier.

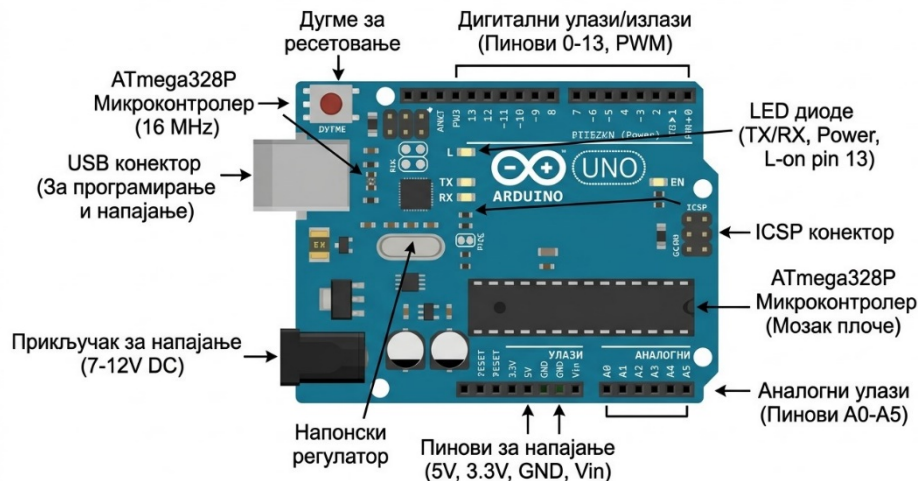
Main Parts of the Arduino Uno Board

Before the first wiring, it is important for students to know where to find:

- USB connector — for connecting to the computer and uploading programs,
- Digital pins (0–13) — for digital input and output signals,
- Analog pins (A0–A5) — for reading analog sensor values,
- GND — ground (negative reference),
- 5V and 3.3V — power supply pins,
- Built-in LED — connected to pin 13, useful for the first test,
- Reset button — restarts the program,
- Microcontroller chip (ATmega328P) — the processor of the board.

✓ Tip: Always identify GND and power pins before connecting any component.

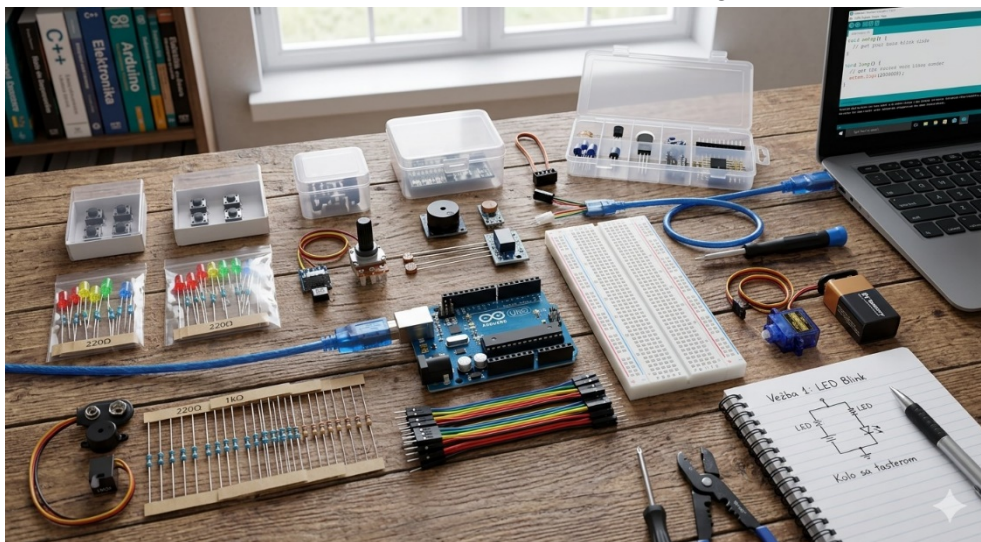
ОСНОВНИ ДЕЛОВИ ARDUINO UNO ПЛОЧЕ



Basic Electronics Starter Kit

For the first exercises, a basic starter kit is used, which typically includes an Arduino board, USB cable, breadboard, connecting wires, resistors, LED diodes, a push button, sensors, and other small electronic components. Such a kit allows a large number of interesting experiments to be performed right from the beginning.

Once students learn to recognize components and connect them correctly, they can complete a whole series of exercises without fear of incorrect wiring.



Breadboard, Resistors, and LED

A breadboard is a board for quickly connecting electronic components without soldering. Connections can be easily changed, ideas tested, and circuits re-verified. The resistor limits current — for an LED, it is necessary to prevent damage.

When working with an LED, especially in real wiring, it is important to know which leg is the anode (+) and which is the cathode (–), and how to correctly use GND.

- 💡 LED anode (longer leg) connects to the signal pin through the resistor. LED cathode (shorter leg) connects to GND.

Development Environments: Arduino IDE, Tinkercad, and Circuit Designer

Programs for Arduino are most commonly written in the Arduino IDE environment. This is where code is written, verified, and uploaded to the board. Tinkercad and Circuit Designer are useful for planning circuits and initial verification, since they allow simulation before real wiring.

Arduino IDE

Arduino IDE is the official programming environment for Arduino boards. In it, the program is written, the code is checked for errors, and the program is sent to the microcontroller. The environment is simple to use and suitable for beginners.

Arduino IDE enables:

- 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 software is free and can be downloaded from the official Arduino website. After installation, select the correct board type and COM port, then upload the program to the device.

Tinkercad & Circuit Designer

Tinkercad and Circuit Designer enable virtual component wiring and circuit testing before real-world work, which is especially useful in teaching and during first exercises with the Arduino platform.

✓ Best practice: Always simulate the circuit first in Tinkercad, then build it physically on the breadboard.

Sensors and Actuators Used in Later Projects

In the following exercises, students will work with a variety of sensors and actuators, for example:

Sensor / Actuator	Type	Used In
DHT11	Temperature & humidity sensor	Project 1
LDR (Photoresistor)	Light sensor	Project 2
HC-SR04	Ultrasonic distance sensor	Project 3
Relay module	Electrical switch (actuator)	Project 5
Soil moisture sensor	Capacitive sensor	Project 5
SG90 Servo motor	Position actuator	Project 3

Exercise 1: First Program Upload — Blink

This exercise verifies that the Arduino board is correctly connected to the computer, the right COM port is selected, and the program is successfully uploaded. Instead of an external circuit, the built-in LED on pin 13 is used — this is the best initial test.

Arduino Code

```
const int ledPin = 13;

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

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Co
de
Exp
lan
atio

n

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

How to Upload the Program

- Open Arduino IDE.
- Go to Tools → Board → Arduino Uno.
- Go to Tools → Port → select the correct COM port.
- Click the Upload button (→ arrow icon).
- The built-in LED on the board should start blinking once per second.

✓

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

Exercise 2: Wiring an External LED on a Breadboard

In this exercise, the student moves from the built-in LED to an external LED, assembling a real electronic circuit for the first time. Here, the connection between the schematic, physical wiring, and the program is clearly demonstrated.

Wiring

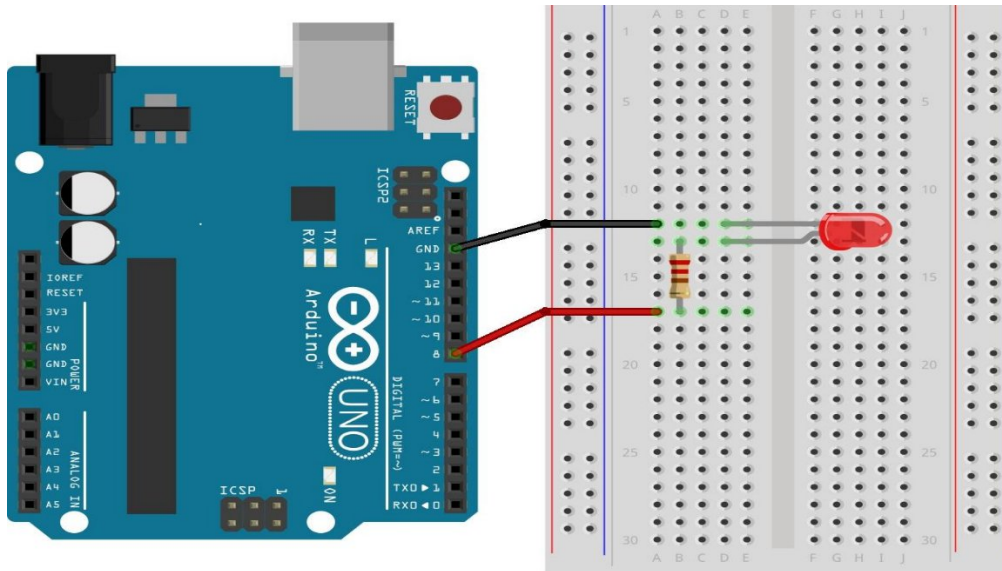


Figure 3: Physical Breadboard Wiring

Arduino Uno	Component	Connection
Pin 8	LED anode (+)	Through 220Ω resistor
GND	LED cathode (-)	Directly to ground

Arduino Code

```
const int ledPin = 8;

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

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

Wh
at
Doe

s This Program Do?

- turns the LED on pin 8 on,
- waits half a second (500 ms),
- turns the LED off,
- repeats the whole process indefinitely.

Student Tasks

Level	Task	Goal
● Basic	Change the delay value	Understand the effect of pause duration
● Intermediate	Move LED to a different pin	Practice pin assignment in code
● Advanced	Add a push button to control the LED	Learn INPUT_PULLUP and digitalWrite
● Challenge	Create a 3-LED sequential light effect	Multiple outputs, timing logic

Tips & Common Mistakes

Pay attention to the following:

- always identify where GND and power pins are before connecting anything,
- never connect an LED directly without a current-limiting resistor,
- before uploading code, verify that the correct COM port is selected,
- if the circuit does not work, first check physical connections, then the code,
- start with simple circuits before moving to more complex ones.

💡 A systematic approach: Schematic → Simulation (Tinkercad) → Real wiring → Upload code → Test.

Real-World Applications

This approach to learning is used in:

- educational workshops and STEM activities,
- smart home prototype development,
- first IoT projects and connected devices,
- meteorological and measurement systems,
- robotic and automated solutions,
- interactive art and light installations,
- wearable electronics and maker projects.

Cross-Curricular Connections

Subject	Connection	Example
Physics	Measurement and understanding of phenomena	Voltage, current, resistance, sensors
Computer Science	Programming and algorithms	setup, loop, conditions, loops
Technology & Engineering	Electronics and practical work	Component wiring, circuit assembly
Mathematics	Logic and analysis	Timing, value comparison, binary states
Art & Design	Creative application	LED effects, interactive displays

DigComp Competencies

The lesson develops the following DigComp 2.2 competencies:

DigComp Area	Competency Developed
1. Information & Data Literacy	Understanding digital signals and sensor data
2. Communication & Collaboration	Presenting results and working in pairs/groups
3. Digital Content Creation	Writing and modifying Arduino programs (sketches)
4. Safety	Safe use of electronic components, correct voltage levels
5. Problem Solving	Debugging circuits and code systematically

Specifically, the lesson develops:

- digital literacy and an introduction to computational thinking,
- problem solving through systematic debugging,
- basic programming skills with a real microcontroller,
- understanding of the hardware-software interface,
- teamwork and collaboration in a STEM environment.

Project Extension Ideas

Students who complete both exercises can extend the project with:

- adding a servo motor and controlling movement with code,
- connecting a light sensor (LDR) to automatically control the LED brightness,
- using the Serial Monitor to print messages and sensor values,
- simulation in Tinkercad before real-world assembly,
- moving to ESP32 in a later project phase for WiFi connectivity,
- adding a push button and programming toggle behavior,
- building a traffic light system with 3 LEDs and timed sequence.

Extension	Difficulty	New Skills
Serial Monitor messages	● Basic	Serial.begin(), Serial.println()
Push button toggle	● Intermediate	digitalRead, INPUT_PULLUP, state logic
LDR light control	● Intermediate	analogRead, map(), sensor calibration
Servo motor control	● Advanced	Servo library, PWM signals
ESP32 WiFi LED control	● Challenge	WiFi library, web server, IoT concepts

Conclusion

This lesson represents the first step in working with the Arduino platform and lays the foundation for all subsequent STEM activities in the Digital Challenge workshop series. Through the Blink exercise and real LED wiring, students see how a program, electronic components, and a physical circuit come together as a single system.

The skills learned here — reading a schematic, writing `setup()` and `loop()`, using `digitalWrite()` and `delay()`, and wiring on a breadboard — are directly applicable in every subsequent project, from temperature sensors to sonar and automatic irrigation systems.

Further Reading & Next Steps

As a continuation of this lesson, the following projects in the Digital Challenge series are recommended:

Next Project	Topic	New Concepts
Project 1	Temperature & Humidity — DHT11	Libraries, serial communication, Processing
Project 2	Light Sensor — LDR	Analog input, voltage divider, bar chart
Project 3	Sonar — HC-SR04	Distance measurement, servo, radar display
Project 4	Ohm Meter	Ohm's Law, voltage divider measurement
Project 5	Automatic Irrigation	Relay, pump control, threshold logic

✓ Recommended path: Complete the Blink exercises → move to Project 1 (DHT11) → continue with all 5 projects in sequence.

Introduction: Processing — Real-Time Visualization for Arduino Projects

Description

Processing is a free, open-source programming environment designed for visual output, animation, and interactive applications. In the context of Arduino projects, Processing acts as the visual front-end: it receives data sent by Arduino over a serial connection and displays it in real time as text, bars, graphs, colors, shapes, or animations.

This introduction prepares students for all five projects in the Digital Challenge workshop series, where Processing is used to visualize sensor data — from temperature and humidity to distance, light intensity, resistance, and soil moisture.

The lesson follows the same structure as the Arduino introduction and is designed for Erasmus+ STEM documentation, encouraging exploratory learning and practical understanding of how data moves from the physical world into a visual interface.

Objectives

Students will:

- understand what Processing is and how it differs from Arduino IDE,
- install and configure the Processing environment,
- understand the structure of a Processing sketch (setup() and draw()),
- learn to receive serial data from Arduino,
- display sensor values as text, shapes, and color-coded bars,
- use the serialEvent() function for efficient data handling,
- create a complete real-time visualization connected to an Arduino sensor.

Learning Outcomes

After the lesson, students will be able to:

- explain the role of Processing in an Arduino + PC system,
- write and run a basic Processing sketch,
- use the Serial library to receive data from Arduino,
- parse comma-separated values sent by Arduino,
- display sensor readings using text(), rect(), ellipse(), and fill(),
- respond to incoming serial data using serialEvent(),
- build a real-time display for at least one sensor project.

Required Software & Equipment

Component	Version / Details	Purpose
Processing IDE	4.x (latest) — processing.org	Writing and running visualizations

Component	Version / Details	Purpose
Arduino IDE	2.x (latest) — arduino.cc	Programming Arduino board
Arduino Uno	Hardware board	Sending sensor data over serial
USB cable	Standard A-B or A-micro	Connecting Arduino to computer
Any sensor	DHT11, LDR, HC-SR04, etc.	Generating data to visualize

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

Theoretical Background

What is Processing?

Processing is a programming language and environment built on top of Java, created specifically for visual design and interactive applications. It was developed at MIT Media Lab in 2001 and is widely used in digital art, data visualization, education, and creative coding.

In the Arduino ecosystem, Processing plays the role of the visual dashboard — it cannot directly control hardware, but it excels at receiving data and turning numbers into compelling visual output.

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, Python, Javascript
Sketch structure	setup() + loop()	setup() + draw()

How Arduino and Processing Work Together

The two environments communicate through serial communication over the USB cable:

```
Arduino —(Serial.println)→ USB cable → Processing
                                   |
                                   serialEvent()
                                   |
                                   display on screen
```

The data flow in all five Digital Challenge projects follows this pattern:

- Sensor reads a physical value (temperature, distance, light...),
- Arduino processes the value and sends it via `Serial.println()`,
- Processing receives the string, parses it, converts to a number,
- Processing draws text, bars, colors, or shapes on the screen.

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

The Processing Sketch Structure

Every Processing program (called a sketch) has two mandatory functions — exactly like Arduino:

Function	When it runs	Typical use
<pre>void setup() { // Runs ONCE at the start // Set window size, initialize serial port } void draw() { // Runs CONTINUOUSLY — like loop() in Arduino // Draw shapes, text, update the display }</pre>		
<code>mousePressed()</code>	When mouse button is clicked	UI buttons, interactions
<code>keyPressed()</code>	When keyboard key is pressed	Send commands back to Arduino

The Serial Library

Processing communicates with Arduino using its built-in Serial library. Key steps:

```
import processing.serial.*;    // 1. Import the library  
  
Serial myPort;                // 2. Declare serial port object  
  
void setup() {  
  size(400, 300);  
  // 3. List available ports (helpful for finding the right one)  
  printArray(Serial.list());  
  // 4. Open the port — index [0] is usually the first available  
  myPort = new Serial(this, Serial.list()[0], 9600);  
}
```

✓ If `Serial.list()[0]` is wrong, check the index. Run `printArray(Serial.list())` to see all available ports.

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

Key Processing Drawing Functions

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

Color in Processing

Processing uses RGB color values (0–255) for all coloring functions:

```
fill(255, 0, 0);      // Red
fill(0, 255, 0);     // Green
fill(0, 0, 255);     // Blue
fill(255, 255, 0);   // Yellow
fill(0);              // Black (shorthand)
fill(255);            // White (shorthand)
fill(128, 0, 128);   // Purple

// With transparency (alpha channel 0-255):
fill(255, 0, 0, 150); // Semi-transparent red
```

The map() Function — Essential for Sensor Visualization

The `map()` function rescales a value from one range to another. This is essential when displaying sensor data as bars, since sensor values (0–1023) need to fit on screen:

```
// Sensor value 0-1023 mapped to bar width 0-300 pixels:  
float barWidth = map(sensorValue, 0, 1023, 0, 300);  
rect(50, 100, barWidth, 30);  
  
// Temperature 0-50°C mapped to color red component:  
float r = map(temperature, 0, 50, 0, 255);  
fill(r, 0, 0);
```

💡 `map()` is used in every single project in the Digital Challenge series. Master it early!

The Processing Development Environment (PDE)

The Processing Development Environment (PDE) includes a text editor, a compiler, and a display window. It enables the creation of software within a carefully designed set of constraints. The PDE consists of a simple text editor for writing code, a message area, a text console, tabs for managing files, a toolbar with buttons for common actions, and a series of menus.

See link: <https://processing.org/environment>

PDE Component	Purpose
Text Editor	Write and edit Processing sketch code
Message Area	Feedback while saving, exporting, and displaying errors
Console	Text output from sketches — <code>print()</code> and <code>println()</code> results
Toolbar	Run and Stop buttons for quick sketch control
Tabs	Manage multiple files within one sketch
Menus	File, Edit, Sketch, Debug, Tools, Help

💡 In Processing, a computer program is called a sketch. Sketches are stored in the Sketchbook — a folder on your computer.

Toolbar Buttons

Button	Action
▶ Run	Compiles the code and opens a new display window (Java mode)
■ Stop	Terminates a running sketch

File Menu

Command	Description
New	Creates a new sketch named as the current date (format: sketch_YYMMDDa)
Open...	Open a sketch in a new window
Open Recent	Select from recently closed sketches
Sketchbook...	Show the list of all sketches in the sketchbook
Examples...	Browse built-in and contributed code examples
Close	Close the current sketch window
Save	Save the open sketch in its current state
Save As...	Save with a different name (does not replace the previous version)
Export Application...	Export as a standalone executable and open the output folder
Preferences...	Configure Processing settings (on macOS: Processing menu)
Quit	Exit Processing and close all windows (on macOS: Processing menu)

Edit Menu

Command	Description
Undo	Reverse the last command or entry
Redo	Reverse the last Undo action
Cut	Remove and copy selected text to clipboard
Copy	Copy selected text to clipboard
Copy as HTML	Copy code formatted as HTML (same appearance as in PDE)
Paste	Insert clipboard contents at cursor position
Select All	Select all text in the currently open file
Auto Format	Format code into a more readable layout (formerly called Beautify)
Comment / Uncomment	Comment or uncomment the selected text
Increase Indent	Indent selected text by two spaces
Decrease Indent	Remove two spaces from the indent of selected text

Command	Description
Find...	Find and optionally replace a text string
Find Next	Find the next occurrence of the search string
Find Previous	Find the previous occurrence of the search string

Sketch Menu

Command	Description
Run	Compile, open display window, and run the sketch
Present	Run the sketch fullscreen on a solid-color background (Esc to exit)
Tweak	Run with live editing of colors and variable values (sketch must be saved first)
Stop	Stop the currently running sketch
Import Library...	Add import statements; select Add Libraries... to install new ones
Show Sketch Folder	Open the folder containing the current sketch files
Add File...	Add an image, font, or media file to the sketch's data folder

Debug Menu

Command	Description
Enable Debugger	Activate the debugger (Run button changes to Debug)
Continue	Advance code execution to the next breakpoint
Step	Advance one line at a time
Step Into	Advance into a user-defined function call
Step Out	Advance out of a function to the calling area
Toggle Breakpoint	Add or remove a breakpoint (line number replaced with <>)

💡 The Debugger is activated through Debug → Enable Debugger, or by clicking the butterfly icon in the upper-right corner of the PDE.

Tools Menu

Command	Description
Archive Sketch	Save a .zip copy of the current sketch in the same folder
Color Selector...	Pick colors and copy their HSB, RGB, or Hex values
Create Font...	Convert system fonts to Processing VLB format for use in sketches
Theme Selector	Choose or create custom PDE color themes
Movie Maker	Create a QuickTime movie from a sequence of exported images
Manage Tools...	Open the Contribution Manager to browse and install tools

Preferences

The PDE is highly configurable. The most common preferences are modified in the Preferences window (File menu on Windows/Linux, Processing menu on macOS):

Preference	Description
Sketchbook location	Set the folder used as the Sketchbook
Language	Select menu language (restart required)
Editor font size	Set the font size in the code editor
Console font size	Set the font size in the console
Background color when Presenting	Color used when running a sketch with Present
Continuously check for errors	Enable/disable real-time error detection
Code completion with Ctrl-Space	Enable/disable code autocomplete
Suggest import statements	Auto-suggest libraries when their code is detected
Increase maximum available memory	Allocate more RAM for sketches using large media files
Run sketches on display	Select which monitor to use for the sketch window (multi-monitor)

Sketches and Sketchbook

All Processing projects are called sketches. Each sketch has its own folder — the main file has the same name as the folder, with a .pde extension (Processing Development Environment).

```
Sketch name:  MyProject
Folder:      MyProject/
Main file:   MyProject/MyProject.pde
Media files: MyProject/data/ ← images, fonts, sounds go here
```

Sketches are stored in the Sketchbook by default. To find its location:

- Go to File → Preferences (or Processing → Preferences on macOS)
- Check the Sketchbook location field at the top

✓ Add images, fonts, or sound files via Sketch → Add File... — they are automatically placed in the data/ folder.

Renderers

Processing has three built-in screen renderers selected through the `size()` function:

Renderer	Description	Best for
Default (no argument)	Standard 2D renderer	Most 2D sketches and sensor displays
P2D	Faster but less accurate 2D renderer (OpenGL)	High-speed 2D graphics
P3D	3D geometry with camera, lighting, materials (OpenGL)	3D visualizations, radar
FX2D	High-quality 2D renderer (separate JavaFX library)	Large or high-resolution displays

```
// Default 2D renderer:
void setup() { size(400, 300); }

// P3D renderer for 3D graphics:
void setup() { size(400, 300, P3D); }
```

Coordinate System

Processing uses a coordinate system with the origin (0, 0) in the upper-left corner. X increases to the right; Y increases downward.

```
// In a 320×240 window:
// (0, 0)      = upper-left corner
// (320, 240) = lower-right corner
// (319, 239) = last visible pixel (pixels draw right and below the
// coordinate)

// In P3D, Z=0 is at the screen surface; negative Z goes into the scene.
```

💡 Remember: Y increases downward. `rect(50, 100, 200, 30)` draws starting 100 pixels from the TOP — not the bottom.

Programming Styles

Processing supports two levels of sketch complexity:

Style	Description	Use when
Static sketch	No <code>setup()</code> or <code>draw()</code> — code runs once and stops	Learning basics: coordinates, shapes, colors
Active sketch	Uses <code>setup()</code> + <code>draw()</code> loop for animation and interaction	Sensor data, real-time display, interactive UI

Static Sketch Example

```
// Runs once — draws a yellow rectangle:
size(200, 200);
background(255);
noStroke();
fill(255, 204, 0);
rect(30, 20, 50, 50);
```

Active Sketch Example — follows mouse

```
void setup() {
  size(200, 200);
  noStroke();
  fill(0, 102, 153, 204);
}

void draw() {
  background(255);
  rect(width - mouseX, height - mouseY, 50, 50);
  rect(mouseX, mouseY, 50, 50);
}
```

Tabs and Multiple Files

For larger sketches, code can be split across multiple tabs. Click the arrow button to the right of the tabs to create, rename, or delete tabs.

When a program with multiple tabs is run, all tabs are compiled together. Classes in other tabs become inner classes — they cannot have static variables. For full Java-style separate files, add a tab with a .java extension.

☒ Tabs are intended for advanced users. For the Digital Challenge projects, a single-file sketch is sufficient.

Adding Libraries, Tools, and Modes

What to add	How to add
Library	Sketch → Import Library → Add Library... → select → Install
Tool	Tools → Manage Tools... → select → Install
Mode	Mode menu (upper-right) → Add Mode... → select → Install
Examples	File → Examples... → Add Examples button → select → Install

For the Digital Challenge projects, install the Serial library (built-in) and verify it is available:

```
// At the top of every project sketch:
import processing.serial.*;
```

Command Line Interface (CLI)

Processing includes a CLI for running sketches from the terminal — useful for automation and headless setups:

OS	Command
Windows	processing --help more
Linux	processing --help
macOS	cd /Applications/Processing.app/Contents/MacOS then ./Processing --help

Exercise 1: Your First Processing Sketch

Before connecting to Arduino, students learn the Processing environment by creating a simple animated sketch that draws shapes and text — no hardware required.

Goal

Draw a window with a colored rectangle, some text, and a circle that changes color over time.

Processing Code

```
int frameCount = 0;

void setup() {
  size(500, 300);
  textAlign(CENTER, CENTER);
}

void draw() {
  background(30); // Dark gray background
  frameCount++;

  // Animated color using sin wave
  float r = 127 + 127 * sin(frameCount * 0.05);
  float g = 127 + 127 * sin(frameCount * 0.03);

  // Colored rectangle
  fill(r, g, 100);
  rect(50, 80, 400, 60, 10); // Rounded corners

  // White text centered in rectangle
  fill(255);
  textSize(22);
  text("Processing + Arduino = Power!", 250, 110);

  // Animated circle
  fill(r, 50, 200);
  ellipse(250, 220, 60, 60);
}
```

What This Sketch Demonstrates

Concept	Where in the code
Window size setup	size(500, 300) in setup()
Continuous animation	draw() runs ~60 times per second
Dynamic color using math	sin() function on frameCount
Drawing a rectangle	rect(x, y, width, height, radius)
Drawing text	text(), textSize(), textAlign()
Drawing a circle	ellipse(cx, cy, width, height)

✓ Press Ctrl+R (Windows/Linux) or Cmd+R (Mac) to run. Press Ctrl+. to stop.

Exercise 2: Receiving Data from Arduino

This exercise connects Processing to an Arduino board. Arduino sends a number over serial every 500ms, and Processing displays it on screen.

Step 1 — Arduino Code (upload first)

```
int counter = 0;

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

void loop() {
  Serial.println(counter); // Send number to Processing
  counter++;
  if (counter > 100) counter = 0;
  delay(500);
}
```

⚠ Upload to Arduino and close the Serial Monitor before running Processing.

Step 2 — Processing Code

```
import processing.serial.*;

Serial myPort;
String incomingData = "";
int receivedValue = 0;

void setup() {
  size(400, 250);
  printArray(Serial.list()); // Check available ports
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(240);

  // Display received value as text
  fill(50);
  textSize(20);
  text("Received from Arduino: " + receivedValue, 50, 80);

  // Display as a bar
  fill(100, 150, 255);
  float barW = map(receivedValue, 0, 100, 0, 300);
  rect(50, 120, barW, 40);

  // Label on bar
  fill(0);
  textSize(16);
  text(receivedValue + " / 100", 50 + barW/2, 140);
}

void serialEvent(Serial p) {
  incomingData = p.readStringUntil('\n');
  if (incomingData != null) {
    incomingData = trim(incomingData);
    if (incomingData.matches("\\d+")) {
      receivedValue = int(incomingData);
    }
  }
}
```

Code Explanation

Part	Explanation
import processing.serial.*	Imports the Serial communication library
Serial myPort	Object that represents the serial port connection
Serial.list()[0]	Selects the first available COM port
serialEvent(Serial p)	Called automatically when new data arrives from Arduino
readStringUntil('\n')	Reads one line of text (up to newline character)
trim()	Removes whitespace and newline characters from string

Part	Explanation
<code>matches("\\d+")</code>	Checks if string contains only digits (safe parsing)
<code>map(value, 0,100, 0,300)</code>	Scales value 0–100 to pixel width 0–300

Exercise 3: Parsing Multiple Values (CSV Format)

In projects 1, 3, and 5, Arduino sends multiple values in a single line, separated by commas. This exercise teaches students how to parse that format in Processing.

Arduino Code — Sending Two Values

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

void loop() {
  float temperature = 23.5;    // In real project: sensor reading
  float humidity     = 61.0;
  Serial.print(temperature);
  Serial.print(",");          // Comma separator
  Serial.println(humidity);    // Newline at end
  delay(1000);
}
```

Processing Code — Parsing Two Values

```
import processing.serial.*;

Serial myPort;
float temperature = 0;
float humidity    = 0;

void setup() {
  size(500, 300);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(255);
  fill(0);
  textSize(22);
  text("Temperature: " + nf(temperature, 1, 1) + " °C", 50,
100);
  text("Humidity:      " + nf(humidity, 1, 1) + " %", 50, 150);

  // Temperature bar (red)
  fill(220, 50, 50);
  rect(50, 170, map(temperature, 0, 50, 0, 380), 25);

  // Humidity bar (blue)
  fill(50, 100, 220);
  rect(50, 210, map(humidity, 0, 100, 0, 380), 25);
}

void serialEvent(Serial p) {
  String data = p.readStringUntil('\n');
  if (data != null && data.contains(",")) {
    String[] values = split(trim(data), ",");
    if (values.length == 2) {
      temperature = float(values[0]);
      humidity    = float(values[1]);
    }
  }
}
```

CSV Parsing Pattern — Used in All Projects

Step	Code	Explanation
1. Read line	<code>readStringUntil('\n')</code>	Wait for complete line ending with newline
2. Check not null	<code>if (data != null)</code>	Avoid errors on empty reads
3. Check format	<code>data.contains(",")</code>	Confirm it's CSV format
4. Split string	<code>split(trim(data), ",")</code>	Divide into array at comma
5. Check length	<code>values.length == 2</code>	Ensure correct number of values
6. Convert	<code>float(values[0])</code>	Parse string to number

💡 This exact pattern appears in Projects 1, 3, and 5. Once you learn it here, you will recognize it everywhere.

Exercise 4: Interactive Processing – Sending Commands Back to Arduino

Processing can also send data back to Arduino — for example, to turn a pump on/off (Project 5) or control a servo (Project 3). This exercise shows how to use mouse clicks and keypresses to send commands.

Arduino Code — Receiving Commands

```
void setup() {  
  Serial.begin(9600);  
  pinMode(13, OUTPUT);          // Built-in LED  
}  
void loop() {  
  if (Serial.available() > 0) {  
    char cmd = Serial.read();  
    if (cmd == '1') digitalWrite(13, HIGH); // LED ON  
    else if (cmd == '0') digitalWrite(13, LOW); // LED OFF  
  }  
}
```

Processing Code — Buttons and Keyboard Control

```
import processing.serial.*;
Serial myPort;
boolean ledOn = false;

void setup() {
  size(400, 220);
  myPort = new Serial(this, Serial.list()[0], 9600);
  textAlign(CENTER, CENTER);
}

void draw() {
  background(240);
  fill(30);
  textSize(18);
  text("Arduino LED Control", 200, 30);
  text("Click buttons or press 1 / 0 on keyboard", 200, 60);

  // ON button
  fill(ledOn ? color(0, 200, 0) : color(180));
  rect(60, 100, 120, 50, 8);
  fill(255);
  text("ON", 120, 125);

  // OFF button
  fill(!ledOn ? color(200, 50, 50) : color(180));
  rect(220, 100, 280, 50, 8);
  fill(255);
  text("OFF", 280, 125);

  // Status
  fill(30);
  textSize(16);
  text("Status: LED is " + (ledOn ? "ON" : "OFF"), 200, 190);
}

void mousePressed() {
  if (mouseX>60&&mouseX<180&&mouseY>100&&mouseY<150)
  { myPort.write('1'); ledOn=true; }
  if (mouseX>220&&mouseX<340&&mouseY>100&&mouseY<150)
  { myPort.write('0'); ledOn=false; }
}

void keyPressed() {
  if (key=='1') { myPort.write('1'); ledOn=true; }
  if (key=='0') { myPort.write('0'); ledOn=false; }
}
```

Student Tasks

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

Common Errors & Troubleshooting

Error / Problem	Likely Cause	Solution
Port not found / no Serial.list()	Arduino not connected or driver missing	Check USB, install CH340 driver if needed
Port busy / in use	Arduino Serial Monitor is still open	Close Serial Monitor in Arduino IDE first
No data received	Wrong port index in Serial.list()[0]	Run printArray(Serial.list()) and check index
Baud rate mismatch	Arduino uses 9600, Processing uses different	Make both Serial.begin(9600) and new Serial(...,9600)
NullPointerException	data is null (incomplete line)	Always check: if (data != null)
Garbage values on screen	Parsing fails on non-numeric strings	Add: if (data.matches("\\d+"))
Processing sketch freezes	draw() blocked by heavy computation	Move serial reading to serialEvent()

✓ Golden rule: If nothing works, restart both Arduino and Processing and try again. Most serial issues resolve on reconnect.

Processing vs. Arduino IDE — Quick Reference

Concept	Arduino IDE	Processing
Program entry point	setup() + loop()	setup() + draw()
Print to console	Serial.println()	println()
Read input	Serial.read()	myPort.readStringUntil()
Delay / pause	delay(ms)	delay(ms) (avoid in draw())
Variable types	int, float, char, bool	int, float, String, boolean
Draw rectangle	— (no screen)	rect(x, y, w, h)
Draw text	— (no screen)	text(str, x, y)
Color	— (no screen)	fill(r, g, b) / stroke(r, g, b)
Event handling	—	mousePressed(), keyPressed()
Import library	#include	import library.*

Real-World Applications

Processing-based visualization is used in:

- industrial sensor dashboards and real-time monitoring systems,
- scientific data visualization in research laboratories,
- interactive museum exhibits and educational installations,
- creative coding and generative digital art,
- data journalism and information visualization,
- health monitoring systems and biofeedback devices,
- environmental monitoring (air quality, weather stations),
- game development prototyping and interactive media.

Cross-Curricular Connections

Subject	Connection	Example in Projects
Computer Science	Programming, data structures, event handling	serialEvent(), arrays for graphs
Mathematics	Coordinate systems, scaling, functions	map(), sin(), graphing data
Physics	Sensor data interpretation, measurement	Temperature bars, distance radar
Visual Arts	Color theory, composition, design	UI design, color-coded displays
Technology	Systems thinking, hardware-software integration	Arduino + Processing pipeline

DigComp Competencies

The lesson develops the following DigComp 2.2 competencies:

DigComp Area	Competency Developed
1. Information & Data Literacy	Receiving, parsing, and displaying real-time sensor data
2. Communication & Collaboration	Designing clear visual interfaces for data communication
3. Digital Content Creation	Writing Processing sketches and creating visual programs
4. Safety	Understanding data validation and error handling
5. Problem Solving	Debugging serial communication and visualization logic

Specifically, the lesson develops:

- computational thinking through event-driven programming,
- data literacy — transforming raw numbers into visual meaning,
- creative coding and interface design skills,
- systematic problem solving and debugging,
- integration thinking — connecting hardware to software.

Project Extension Ideas

Students who master the basics can extend their Processing skills with:

- drawing a real-time scrolling graph of the last 100 sensor readings,
- adding sound alerts using the Minim audio library,
- saving sensor data to a CSV file using PrintWriter,
- creating a full dashboard with multiple sensor displays,
- adding a clock and timestamp to every logged reading,
- building a web-based version using p5.js (the JavaScript version of Processing),
- connecting multiple Arduinos to one Processing sketch.

Extension	Difficulty	Key Concept
Real-time scrolling graph	● Intermediate	Arrays, shifting values, line()
Save data to CSV file	● Intermediate	PrintWriter, saveStrings()
Sound alarm on threshold	● Advanced	Minim library, audio playback
Multi-sensor dashboard	● Advanced	Multiple serial ports or multiplexed data
p5.js web version	● Challenge	JavaScript, HTML Canvas, WebSerial API

How This Introduction Connects to All 5 Projects

Project	Sensor	Processing Visualization	Key Technique
1 — DHT11	Temperature & humidity	Two-value text + bars	CSV parsing, 2 values
2 — LDR	Light intensity	Single bar + color change	map(), background color
3 — HC-SR04	Distance (sonar)	Radar sweep display	Trigonometry, ellipse, line
4 — Ohm Meter	Resistance	Numeric display + gauge	nf(), arc(), text
5 — Irrigation	Soil moisture	Status display + bar	Threshold logic, fill color

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

Conclusion

Processing is the visual bridge between the physical world and the screen. Once students understand how data flows from a sensor through Arduino's serial port into Processing's `draw()` function, they have the foundation to build any real-time visualization they can imagine.

This introduction covers everything needed for all five Digital Challenge projects: setting up the serial connection, reading and parsing data, drawing shapes and text, using `map()` for scaling, and sending commands back to Arduino. These skills, combined with the Arduino knowledge from the previous introduction, form a complete STEM toolkit for sensor-based interactive projects.

Next Steps

After completing this introduction, proceed to the projects in this order:

Step	Project	New Processing Skills
Introduction	Intro to Arduino	Serial communication basics, setup/loop
← You are here	Intro to Processing	setup/draw, serialEvent, map, shapes
Project 1	DHT11 Temperature & Humidity	CSV parsing, two-bar display
Project 2	Light Sensor — LDR	Single value, dynamic background color

Step	Project	New Processing Skills
Project 3	Sonar — HC-SR04	Radar display, trigonometry, servo control
Project 4	Ohm Meter	Formatted numbers, gauge visualization
Project 5	Automatic Irrigation	Status display, pump control UI

Project 1: Temperature & Humidity — DHT11

Description

In this exercise, students build a simple system for measuring air temperature and humidity using an Arduino Uno and a DHT11 sensor. The collected data is transferred to a computer via serial communication and visualized in real time using Processing.

The exercise connects the fundamentals of electronics, programming, physics, and data visualization, and serves as an introduction to IoT (Internet of Things) systems and working with sensors.

Objectives

Students will:

- learn how digital sensors work,
- connect the DHT11 sensor to an Arduino microcontroller,
- read data from the sensor,
- use serial communication,
- visualize data in real time,
- develop a simple alert system based on preset threshold values.

Learning Outcomes

After the workshop, students will be able to:

- properly connect a digital sensor to a microcontroller,
- use libraries in the Arduino IDE environment,
- read and process data from the sensor,
- understand the principle of serial communication,
- visualize data in the Processing environment,
- analyze changes in temperature and humidity,
- independently modify the system by adding new functionalities.

Required Equipment

Component	Quantity	Purpose
Arduino Uno	1	Microcontroller
DHT11 sensor	1	Temperature and humidity measurement
10kΩ resistor	1	Pull-up resistor
Breadboard	1	Component connection
Jumper wires	~5	Component connection

Component	Quantity	Purpose
USB cable	1	Connection to computer
Arduino IDE	1	Arduino programming
Processing	1	Data visualization

Theoretical Background

What is DHT11?

DHT11 is a digital sensor that measures temperature and relative air humidity, and sends data in digital form. The sensor is suitable for beginners because it does not require analog-to-digital conversion, is simple to connect, and ready-made libraries are available.

What is Relative Humidity?

Relative humidity represents the percentage of water vapor in the air relative to the maximum amount the air can hold at a given temperature and normal atmospheric pressure.

Serial Communication

Arduino sends data to the computer serially over the USB port. For example:

```
Serial.println("24.5,60");
```

This means: temperature = 24.5 °C, humidity = 60 %

Exercise 1: Connecting the DHT11 Sensor

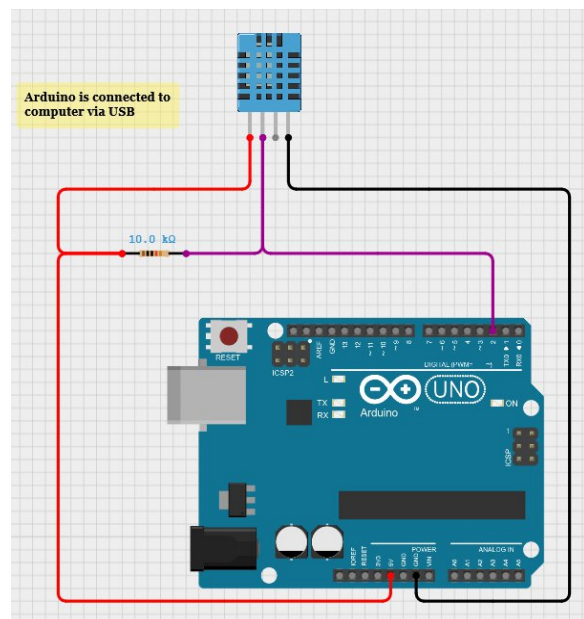
Wiring

DHT11 Pin	Arduino Pin
VCC	5V
GND	GND
DATA	Digital pin 2

If using the "raw" 3-pin sensor version:

- place a 10kΩ resistor between VCC and DATA.

✓ Tip: The 4-pin DHT11 module already has a built-in pull-up resistor.



Arduino Program

Step 1: Library Installation

In Arduino IDE:

- Sketch → Include Library → Manage Libraries
- Search for: "DHT sensor library" by Adafruit
- Install the library

Step 2: Arduino Code

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature(); // Celsius

  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print(temperature);
  Serial.print(",");
  Serial.println(humidity);

  delay(2000);
}
```

Code Explanation

Command	Explanation
dht.readTemperature()	Reads the temperature
dht.readHumidity()	Reads the humidity
Serial.print()	Sends data to the computer
delay(2000)	Waits 2 seconds

✓ Data is sent as comma-separated values because Processing handles them easily.

Exercise 2: Data Visualization in Processing

Processing Code

```
Serial myPort;
String data;
float temp, hum;

void setup() {
  size(400, 200);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(255);
  fill(0);
  textSize(20);
  text("Temperature: " + temp + " °C", 50, 80);
  text("Humidity: " + hum + " %", 50, 120);

  if (myPort.available() > 0) {
    data = myPort.readStringUntil('&apos;\n&apos;');
    if (data != null && data.contains(",")) {
      String[] values = split(trim(data), ",");
      if (values.length == 2) {
        temp = float(values[0]);
        hum = float(values[1]);
      }
    }
  }
}
```

What does the program display?

- It reads the data sent by the Arduino,
- It displays temperature and humidity,
- It visualizes the values using graphical bars.

Student Tasks

Level	Task
● 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

Alert System Example

```
if (temperature > 30) {  
  fill(255, 0, 0);  
  text("WARNING: HIGH TEMPERATURE!", 40, 50);  
}
```

Troubleshooting

Problem	Possible Cause
No data received	Incorrect COM port
"Sensor error" message	Poorly connected sensor
Unstable values	Poor contact on breadboard
Processing not receiving data	Serial communication speed mismatch

✓ Tip: Use `Serial.list()` to find the correct COM port.

Real-World Applications

Such systems are used in: smart homes, weather stations, greenhouses, server rooms, IoT systems, and industrial automation.

Cross-Curricular Connections

Subject	Connection
Physics	Temperature, humidity, measurement
Computer Science	Programming and data processing
Technology & Engineering	Electronics and sensors
Mathematics	Data analysis and graphical representation

DigComp Competencies

This workshop develops:

- digital literacy,
- problem-solving skills,
- data analysis,
- programming skills,
- collaboration in a STEM environment.

Ideas for Project Extension

- Adding an OLED display
- Sending data over a WiFi network
- Creating a web dashboard
- Connecting the system to a cloud platform
- Building a mini weather station

Conclusion

This exercise serves as an introduction to working with sensors, microcontrollers, and data visualization. By combining the Arduino Uno platform and the Processing environment, students gain practical knowledge in electronics, programming, and data analysis through a real-world STEM project.

Project 2: Light Sensor — Photoresistor (LDR)

Description

This project reads ambient light intensity using a photoresistor (LDR) connected to Arduino. The analog value (0–1023) is sent over serial to Processing, which displays it as text and as a visual bar in real time.

Objectives

Students will:

- understand how a photoresistor (LDR) detects changes in light intensity,
- learn how analog sensors work with an Arduino Uno microcontroller,
- build a voltage divider circuit,
- read and interpret analog sensor values,
- use serial communication between Arduino and a computer,
- visualize sensor data in real time using Processing,
- develop simple interactive visualizations based on environmental data.

Learning Outcomes

After completing the workshop, students will be able to:

- correctly connect an LDR sensor using a voltage divider circuit,
- read analog values from an Arduino analog input,
- explain the relationship between light intensity and electrical resistance,
- transfer sensor data using serial communication,
- create basic real-time visualizations in Processing,
- analyze how environmental light conditions affect sensor readings,
- modify the project by adding new visual or interactive functionalities.

Required Equipment

Component	Quantity	Purpose
Arduino Uno	1	Microcontroller
Photoresistor (LDR)	1	Light sensing
10kΩ resistor	1	Voltage divider
Breadboard	1	Component connection
Jumper wires	~5	Component connection
USB cable	1	Connection to computer
Processing IDE	1	Data visualization

Theoretical Background

How Does a Photoresistor Work?

A photoresistor (LDR) changes its resistance depending on the amount of light. In bright light, resistance decreases; in darkness, resistance increases.

Voltage Divider Circuit

The LDR is connected in series with a fixed 10kΩ resistor. The midpoint voltage at A0 varies with light intensity.

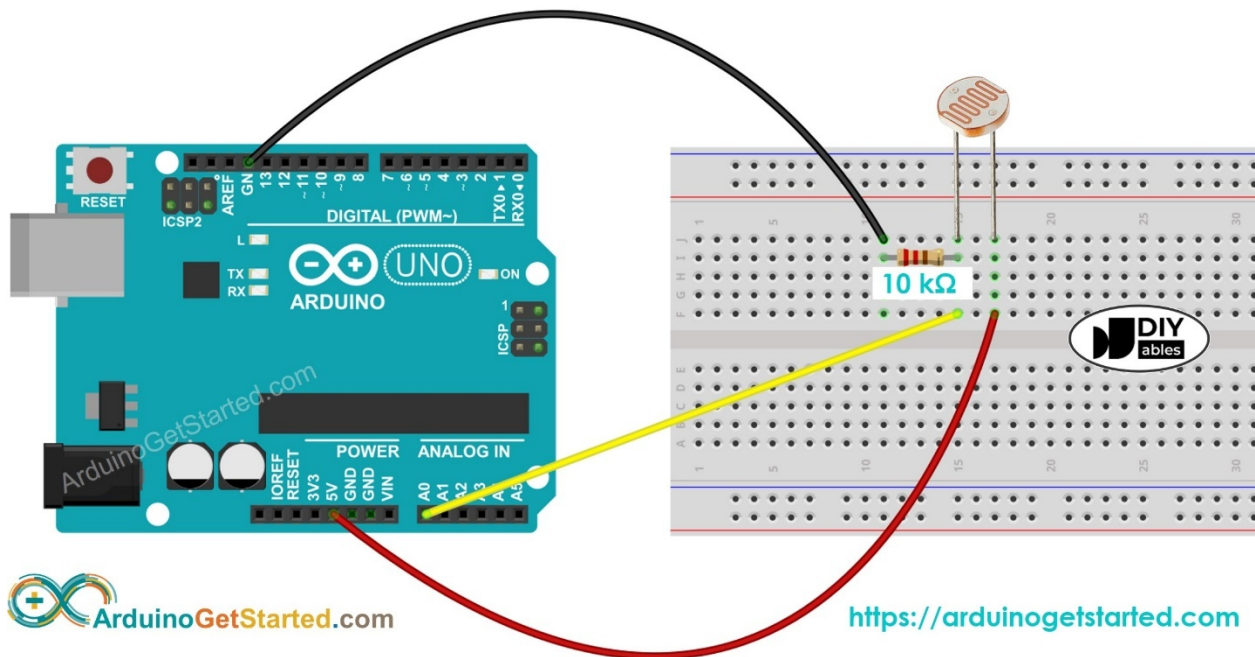
💡 $V_{out} = V_{in} \times R_2 / (R_1 + R_2)$. Here R_2 is the LDR.

Exercise 1: Circuit Setup & Arduino Code

Wiring

Component	Connects To
LDR (leg 1)	5V
LDR (leg 2)	A0 and 10kΩ resistor (leg 1)
10kΩ resistor (leg 2)	GND

✓ The junction between LDR and resistor connects to A0 on Arduino.



Arduino Code

```
const int sensorPin = A0;

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

void loop() {
  int lightValue = analogRead(sensorPin); // 0 (dark) to 1023 (bright)
  Serial.println(lightValue);
  delay(200);
}
```

Exercise 2: Processing Visualization

Processing Code — Text Display

```
import processing.serial.*;
Serial myPort;
String val = "";
int lightLevel = 0;

void setup() {
  size(400, 200);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(255);
  fill(0);
  textSize(20);
  text("Light Intensity: " + lightLevel, 50, 100);

  if (myPort.available() > 0) {
    val = myPort.readStringUntil('\n');
    if (val != null) { lightLevel = int(trim(val)); }
  }
}
```

Processing Code — Bar + Serial Event

```
import processing.serial.*;
Serial myPort;
String val = "";
int lightLevel = 0;

void setup() {
  size(400, 200);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(255);
  fill(0);
  textSize(20);
  text("Light Intensity: " + lightLevel, 50, 50);

  fill(0, 100, 255);
  rect(50, 100, map(lightLevel, 0, 1023, 0, 300), 30);
}

void serialEvent(Serial p) {
  val = p.readStringUntil('\n');
  if (val != null) {
    val = trim(val);
    if (val.matches("\\d+")) { lightLevel = int(val); }
  }
}
```

Student Tasks

Level	Task
● Basic	Display the numeric light value
● Intermediate	Draw a colored bar proportional to light level
● Advanced	Change background color based on light intensity
● Challenge	Log values with timestamps to a text file

Troubleshooting

Problem	Possible Cause
Value stays at 0	LDR not connected or open circuit
Value stays at 1023	Resistor missing or GND not connected
Unstable readings	Poor breadboard contact

Ideas for Project Extension

- Adding an LED that automatically reacts to light intensity
- Creating an automatic night-light system
- Using multiple LDR sensors for directional light detection
- Connecting the system to a WiFi module for remote monitoring
- Building a simple smart home lighting prototype
- Displaying data on an OLED or LCD screen
- Logging light intensity data for long-term analysis

Conclusion

This project introduces students to analog sensors, voltage divider circuits, and real-time data visualization. By combining the Arduino Uno platform with the Processing environment, students gain practical experience in electronics, programming, and interactive sensor systems through a hands-on STEM activity.

Project 3: Sonar — Ultrasonic Sensor HC-SR04

Description

This project uses the HC-SR04 ultrasonic sensor to measure distance. Data is visualized in Processing in three progressively advanced exercises: text display, servo control, and a full radar/sonar display.

Objectives

Students will:

- understand the working principle of ultrasonic distance measurement,
- learn how the HC-SR04 Ultrasonic Sensor measures distance using sound waves,
- connect and program ultrasonic sensors and servo motors using an Arduino Uno,
- use serial communication for transferring measurement data,
- visualize sensor data in real time using Processing,
- develop interactive radar and sonar-style visualizations,
- explore the connection between physics, electronics, and programming through a practical STEM project.

Learning Outcomes

After completing the workshop, students will be able to:

- explain how ultrasonic sensing works,
- calculate distance based on echo return time,
- correctly connect and program the HC-SR04 sensor,
- control a servo motor using Arduino,
- transfer sensor data to Processing through serial communication,
- create real-time graphical visualizations of distance measurements,
- develop simple radar-like interfaces,
- analyze how sensor position and object distance affect measurements,
- independently extend the system with additional features and interactions.

Theoretical Background

Physics of Ultrasonic Sensing

Ultrasonic sensors emit high-frequency sound waves (~40 kHz). When these waves hit an object, they bounce back. The sensor measures the time for the echo to return:

$$\text{Distance} = (\text{Speed of Sound} \times \text{Time}) / 2$$

Speed of sound ≈ 343 m/s in air. Division by 2 accounts for the round trip.

Required Equipment

Component	Quantity	Purpose
Arduino Uno	1	Microcontroller
HC-SR04 ultrasonic sensor	1	Distance measurement
SG90 servo motor	1	Radar sweep (Ex. 2 & 3)
Plastic holder for sensor	1	Mount sensor on servo
Breadboard + jumper wires	1 set	Component connection
USB cable	1	Connection to computer
Processing IDE	1	Visualization

Exercise 1: Measure Distance & Display in Processing

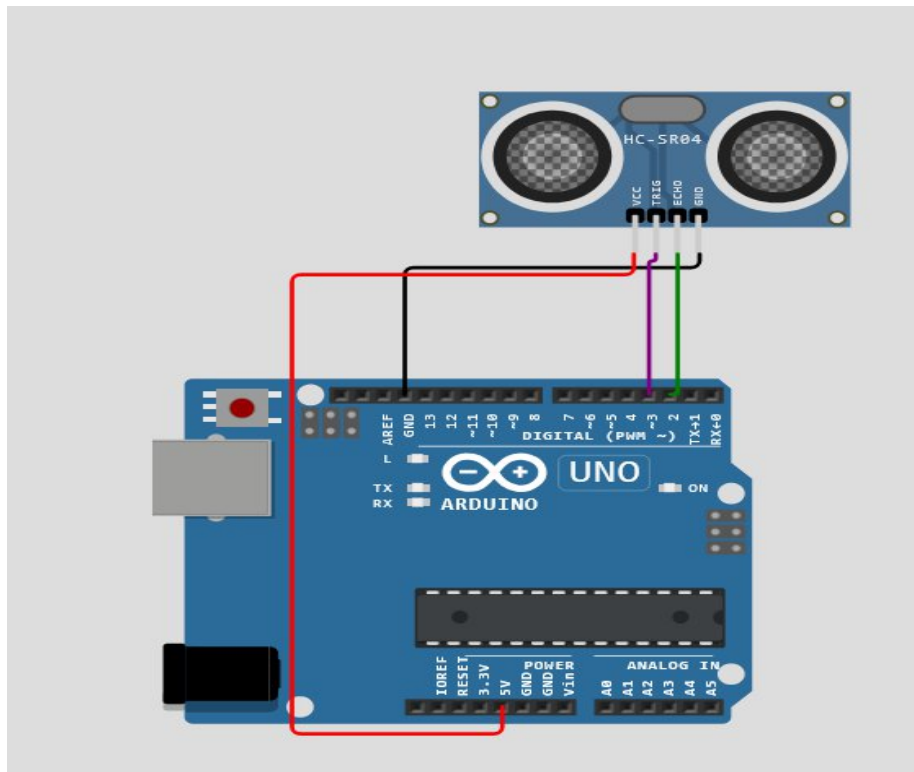
Components

- Arduino Uno
- HC-SR04 ultrasonic sensor
- USB cable
- Processing IDE

Wiring

HC-SR04 Pin Arduino Pin

VCC	5V
GND	GND
TRIG	Pin 3
ECHO	Pin 2



Arduino Code

```
#define trigPin 3
#define echoPin 2
void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  float distance = duration * 0.0343 / 2;
  Serial.println(distance);
  delay(200);
}
```

Processing Code – Visualize Distance

```
import processing.serial.*;
Serial myPort;
String val;
void setup() {
  size(400, 200);
  myPort = new Serial(this, Serial.list()[0], 9600);
}
void draw() {
  if (myPort.available() > 0)
  {
    val = myPort.readStringUntil('\n');
    if (val != null) {
      background(255);
      fill(0);
      textSize(20);
      text("Distance: " + val.trim() + " cm", 50, 100);
    }
  }
}
```

Processing Code – Visualize Distance with horizontal bar

```
import processing.serial.*;
Serial myPort;
float distance;

void setup() {
  size(600, 200);
  printArray(Serial.list());
  myPort = new Serial(this, Serial.list()[0], 9600);
  myPort.bufferUntil('\n');
}

void draw() {
  background(255);
  fill(0);
  textSize(24);
  text("Distance: " + nf(distance, 0, 2) + " cm", 50, 50);
  // Horizontal bar
  fill(100, 200, 255);
  rect(50, 100, map(distance, 0, 200, 0, width - 100), 30);
}

void serialEvent(Serial myPort) {
  String data = myPort.readStringUntil('\n');
  if (data != null) {
    data = trim(data);
    distance = float(data);
  }
}
```

Exercise 2: Control SG90 Servo from Processing Components

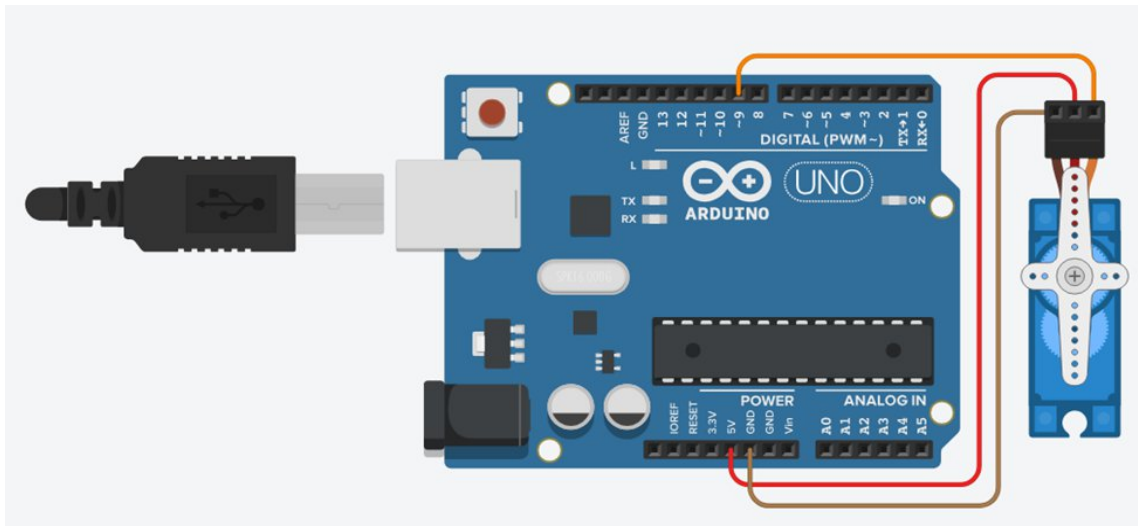
- Arduino Uno
- SG90 Servo motor
- USB cable

Wiring

Servo Pin	Arduino Pin
Signal (orange)	Pin 9
VCC (red)	5V
GND (brown)	GND

[The Beginners Guide to Micro Servos](https://docs.arduino.cc/learn/electronics/servo-motors/)

<https://docs.arduino.cc/learn/electronics/servo-motors/>



Arduino Code

```
#include <Servo.h>

import processing.serial.*;

Serial myPort;
boolean isSweeping = false;

void setup() {
    size(400, 200);
    myPort = new Serial(this, Serial.list()[0], 9600);
    textAlign(CENTER, CENTER);
}

void draw() {
    background(240);
    textSize(16);
    fill(0);
    text("Servo Control", width/2, 30);

    drawButton("START", 100, 100, isSweeping);
    drawButton("STOP", 250, 100, !isSweeping);
}

void drawButton(String label, int x, int y, boolean active) {
    fill(active ? color(0, 200, 0) : color(180));
    rect(x, y, 100, 40, 7);
    fill(0);
    text(label, x + 50, y + 20);
}

void mousePressed() {
    if (mouseX > 100 && mouseX < 200 && mouseY > 100 && mouseY < 140) {
        myPort.write("1\n");
        isSweeping = true;
    } else if (mouseX > 250 && mouseX < 350 && mouseY > 100 && mouseY < 140) {
        myPort.write("0\n");
        isSweeping = false;
    }
}
```

Processing Code — START / STOP Buttons

How It Works

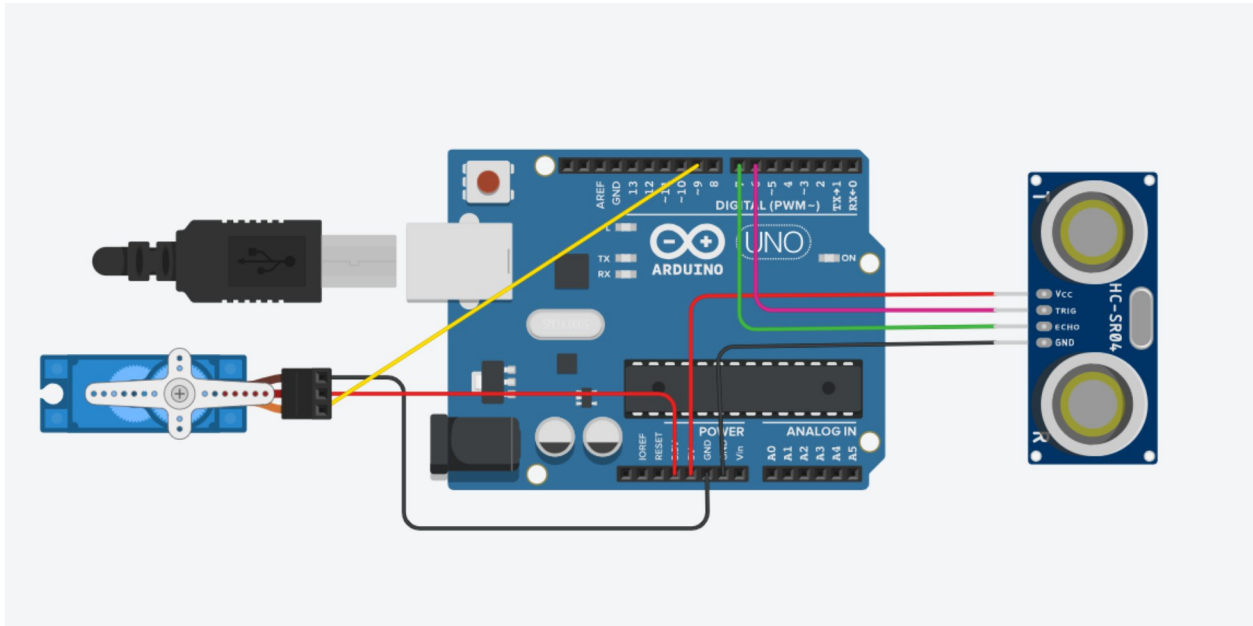
- **Start button** sends "START" → Arduino begins sweeping the servo.
- **Stop button** sends "STOP" → Arduino halts movement.
- You can expand this with angle sliders, speed control, or even real-time feedback.

Exercise 3: Ultrasonic Sensor Controlling Servo

The servo rotates to 90° when an object is detected closer than 50 cm, and returns to 0° otherwise.

Arduino Code

```
#include <Servo.h>
// constants won't change
const int TRIG_PIN = 6; // Pin connected to Ultrasonic Sensor's TRIG pin
const int ECHO_PIN = 7; // Pin connected to Ultrasonic Sensor's ECHO pin
const int SERVO_PIN = 9; // Arduino pin connected to Servo Motor's pin
const int DISTANCE_THRESHOLD = 50; // centimeters
Servo servo; // create servo object to control a servo
// variables will change:
float duration_us, distance_cm;
void setup() {
  Serial.begin(9600); // initialize serial port
  pinMode(TRIG_PIN, OUTPUT); // set arduino pin to output mode
  pinMode(ECHO_PIN, INPUT); //set arduino pin to input mode
  servo.attach(SERVO_PIN); // attaches the servo on pin 9 to the servo
  object
  servo.write(0);
}
void loop() {
  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(ECHO_PIN, HIGH);
  // calculate the distance
  distance_cm = 0.017 * duration_us;
  if(distance_cm < DISTANCE_THRESHOLD)
    servo.write(90); // rotate servo motor to 90 degree
  else servo.write(0); // rotate servo motor to 0 degree
  // print the value to Serial Monitor
  Serial.print("distance: ");
  Serial.print(distance_cm);
  Serial.println(" cm");
  delay(500);
}
```



Components

- Arduino Uno
- HC-SR04 ultrasonic sensor
- SG90 servo motor
- Plastic holder (to rotate sensor)
- Processing IDE

Tutorial Reference

[Arduino Ultrasonic Sensor + Servo Motor Tutorial](#)

Student Task

Write a Processing program that displays the measured distance in two visual formats:

- a) As a numerical value using text()
- b) As a horizontal bar proportional to the distance

Exercise 4: Build a Radar Scanner (Sonar)

🔧 This exercise is suitable as a project topic.

Arduino Code — Radar Sweep

```
#include <Servo.h>
Servo myServo;
#define trigPin 9
#define echoPin 10
void setup() {
  Serial.begin(9600);
  myServo.attach(6);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  for (int angle = 0; angle <= 180; angle += 2) {
    myServo.write(angle);
    delay(50);

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH);
    float distance = duration * 0.0343 / 2;

    Serial.print(angle);
    Serial.print(",");
    Serial.println(distance);
  }
}
```

Processing Code — Radar Display

```
import processing.serial.*;

Serial myPort;
String data;
int angle;
float distance;

void setup() {
  size(600, 600);
  myPort = new Serial(this, Serial.list()[0], 9600);
  background(0);
}

void draw() {
  if (myPort.available() > 0) {
    data = myPort.readStringUntil(' ');
    if (data != null && data.contains(",")) {
      String[] parts = split(trim(data), ",");
      if (parts.length == 2) {
        angle = int(parts[0]);
        distance = float(parts[1]);

        float r = map(distance, 0, 100, 0, 250);
        float x = width/2 + r * cos(radians(angle));
        float y = height - r * sin(radians(angle));

        stroke(0, 255, 0);
        fill(0, 255, 0);
        ellipse(x, y, 5, 5);
      }
    }
  }
}
```

Components

- Arduino Uno
- HC-SR04 ultrasonic sensor
- SG90 servo motor
- Plastic holder (to rotate sensor)
- Processing IDE

Student Tasks

Level	Task
● Basic	Display distance as text on radar screen
● Intermediate	Draw radar grid lines and circles
● Advanced	Color-code dots by distance (green/yellow/red)
● Challenge	Add sweep line animation and sound alert

Ideas for Project Extension

- Adding sound effects or alarms when objects are detected
- Creating a fully animated radar interface
- Adding multiple ultrasonic sensors for wider coverage
- Connecting the system to a WiFi module for remote monitoring
- Building a smart parking assistant prototype
- Integrating LEDs to indicate object distance
- Storing radar data for later analysis
- Developing a simple obstacle-avoidance robot
- Combining the system with temperature or light sensors for multifunctional monitoring

Conclusion

This project introduces students to ultrasonic sensing, real-time measurement systems, servo motor control, and interactive data visualization. By combining the HC-SR04 Ultrasonic Sensor, Arduino Uno, and Processing environments, students gain practical experience in physics, electronics, programming, and interactive system design through a hands-on STEM project inspired by real-world radar and sonar technologies.

Project 4: Arduino Ohm Meter with Processing Display

Description

This project uses a voltage divider to measure an unknown resistor. Arduino calculates the resistance using Ohm's Law and sends the result to Processing, which displays it in real time.

Objectives

Students will:

- understand the principles of voltage dividers and Ohm's Law,
- learn how to measure an unknown resistance using an Arduino Uno,
- use analog inputs to read voltage values,
- calculate resistance values from measured voltages,
- transfer calculated data using serial communication,
- visualize measurements in real time using Processing,
- develop practical understanding of electrical measurement systems.

Learning Outcomes

After completing the workshop, students will be able to:

- explain how a voltage divider circuit works,
- apply Ohm's Law to calculate an unknown resistance,
- correctly connect resistors in a voltage divider configuration,
- use Arduino analog inputs for voltage measurement,
- process and transmit measurement data through serial communication,
- create basic real-time visualizations in Processing,
- interpret how resistor values influence circuit voltage,
- independently improve the project by adding additional visualization or measurement features.

Required Equipment

Component	Quantity	Purpose
Arduino Uno	1	Microcontroller
Known resistor (e.g. 1kΩ)	1	Reference for voltage divider
Unknown resistor	1	Component to measure
Breadboard + jumper wires	1 set	Component connection
USB cable	1	Connection to computer

Component	Quantity	Purpose
Processing IDE	1	Display

Theoretical Background

Voltage Divider & Ohm's Law

A voltage divider uses two resistors in series. The voltage at the midpoint depends on their ratio:

$$V_{out} = V_{in} \times R_2 / (R_1 + R_2) \rightarrow R_2 = R_1 \times (V_{in}/V_{out} - 1)$$

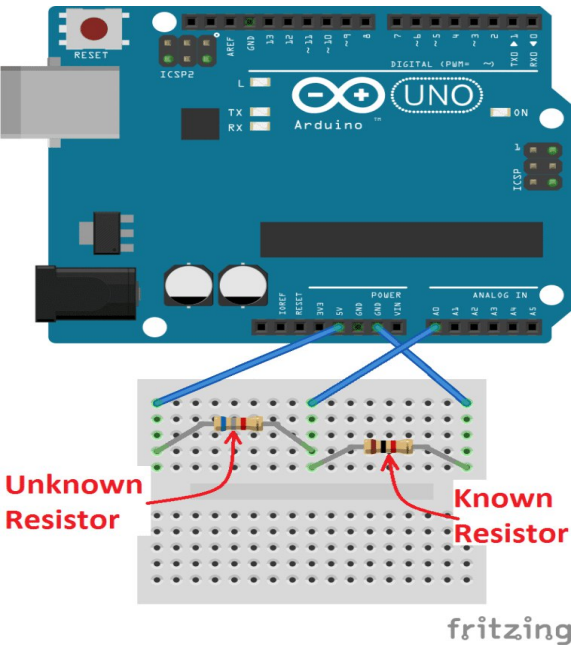
Arduino reads V_{out} via analog pin A0 and calculates R_2 (the unknown resistor).

Step 1: Circuit Setup

Wiring

Connection	Description
R1 (known, e.g. 1kΩ)	Between 5V and A0
R2 (unknown)	Between A0 and GND
A0	Junction between R1 and R2

💡 The higher R_2 relative to R_1 , the more voltage appears at A0.



Step 2: Arduino Code

```
vconst int analogPin = A0;
const float Vin = 5.0;
const float R1 = 1000.0; // Known resistor in ohms

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

void loop() {
  int raw = analogRead(analogPin);
  float Vout = (raw * Vin) / 1023.0;
  float R2 = R1 * (Vin / Vout - 1);

  Serial.println(R2); // Send resistance to Processing
  delay(500);
}
```

Step 3: Processing Code

How It Works

```
import processing.serial.*;

Serial myPort;
String val = "";
float resistance = 0;

void setup() {
  size(400, 200);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(255);
  fill(0);
  textSize(20);
  text("Measured Resistance:", 50, 80);
  text(nf(resistance, 1, 2) + " Ω", 50, 120);

  if (myPort.available() > 0) {
    val = myPort.readStringUntil('&apos;\n&apos;');
    if (val != null) {
      resistance = float(trim(val));
    }
  }
}
```

- Arduino reads the voltage at the midpoint of the voltage divider.
- It calculates the unknown resistance using Ohm's Law.
- The resistance is sent over serial to Processing.
- Processing displays the value in real time.

Student Tasks

Level	Task
● Basic	Display the resistance value as text
● Intermediate	Color the text based on resistance range
● Advanced	Draw a dial / gauge visualization
● Challenge	Log multiple measurements and plot a graph

Troubleshooting

Problem	Possible Cause
Reading shows 0 or infinity	R2 not connected or short circuit
Reading is inaccurate	R1 value entered incorrectly in code
Fluctuating values	Poor breadboard contact

Ideas for Project Extension

- Adding automatic resistor classification by resistance range
- Creating a graphical gauge or analog-style meter
- Displaying resistor color codes alongside measured values
- Measuring variable resistors such as potentiometers
- Logging resistance measurements to a CSV file
- Building a simple digital multimeter prototype
- Adding OLED or LCD display support
- Creating a calibration mode for more accurate measurements
- Comparing measured values with nominal resistor tolerances

Conclusion

This project introduces students to electrical measurement techniques, voltage divider circuits, and real-time data visualization. By combining the Arduino Uno platform with the Processing environment, students gain practical experience in electronics, programming, and applied physics through a hands-on STEM activity focused on measuring and analyzing electrical resistance.

Project 5: Automatic Irrigation System

Description

This project uses a capacitive soil moisture sensor to measure the water content in soil. Students build an automatic irrigation system that monitors soil conditions in real time and controls a water pump accordingly. The sensor data is transferred to a computer via serial communication and visualized using Processing.

The project integrates electronics, programming, physics, and environmental science, and serves as a practical introduction to automation, IoT (Internet of Things) systems, and sensor-based control.

Objectives

Students will:

- learn how capacitive soil moisture sensors work,
- connect the sensor and relay module to an Arduino microcontroller,
- read and interpret analog sensor data,
- use serial communication to transfer data to a computer,
- visualize soil moisture data in real time using Processing,
- control a water pump via a relay module from both Arduino and Processing,
- build a fully automatic irrigation system with threshold-based logic.

Learning Outcomes

After the workshop, students will be able to:

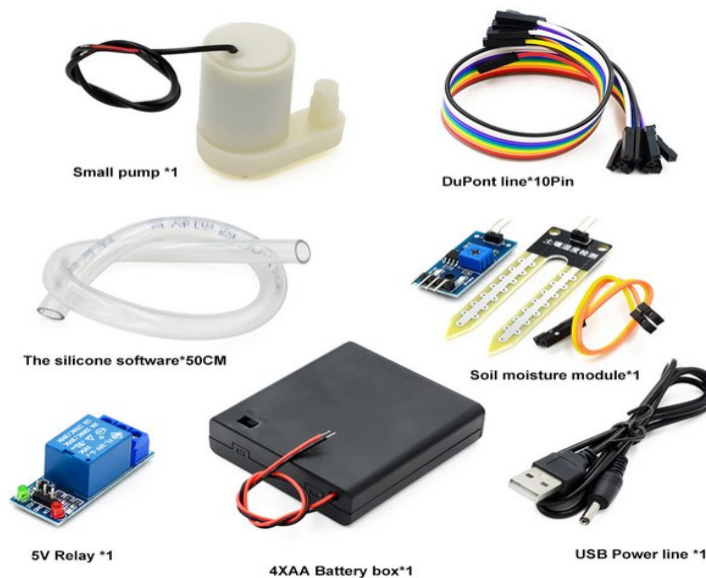
- properly connect a capacitive soil moisture sensor and relay to Arduino,
- read and scale analog sensor values,
- understand and apply threshold-based control logic,
- use serial communication to send data between Arduino and Processing,
- visualize sensor data as text, bars, and status indicators in Processing,
- send control commands from Processing to Arduino,

design a complete automated system combining sensor input and actuator output

Required Equipment

Component	Quantity	Purpose
Arduino Uno	1	Microcontroller
Capacitive Soil Moisture Sensor v1.2	1	Soil moisture measurement
Relay module	1	Pump switching
Water pump (small)	1	Watering
	1	Power for pump

Component	Quantity	Purpose
Silicone hose (50 cm)	1	Water delivery
Breadboard + jumper wires	1 set	Component connection
USB cable	1	Connection to computer
Processing IDE	1	Visualization



Theoretical Background

How Does the Soil Moisture Sensor Work?

The capacitive soil moisture sensor outputs an analog voltage that decreases as soil moisture increases. Dry soil → high value (~1023). Wet soil → low value (~300). Unlike resistive sensors, it does not corrode over time.

💡 Physics: Wet soil has higher permittivity, changing the capacitance and output voltage of the sensor circuit.

Relay Module

A relay is an electrically operated switch. A small 5V signal from Arduino controls a higher-voltage pump circuit. When relay input is HIGH the switch is typically open (pump OFF); when LOW, the switch closes (pump ON). Always check the active logic of your specific relay module.

Serial Communication

Arduino sends sensor data to the computer serially over the USB port. For example:

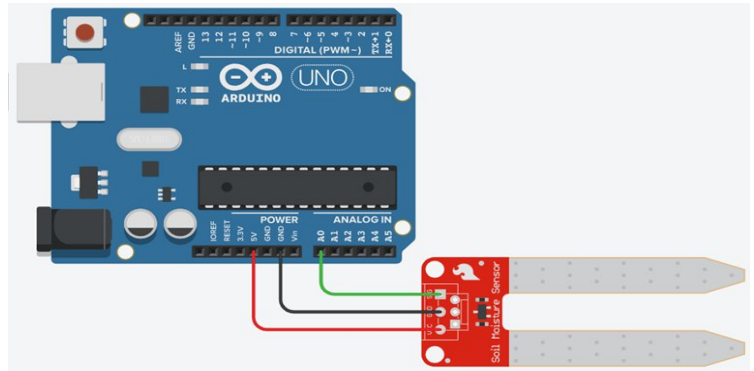
```
Serial.println(moisture);
```

Processing reads this value and uses it to update the visual display and determine the pump status.

Exercise 1: Data Acquisition & Visualization

Wiring — Sensor

Sensor Pin	Connects To
VCC	3.3V or 5V
GND	GND
AOUT	Arduino A0



Arduino Code

```
const int sensorPin = A0;

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

void loop() {
  int moisture = analogRead(sensorPin);
  // 0 (wet) to 1023 (dry)
  Serial.println(moisture);
  delay(300);
}
```

The sensor outputs an analog voltage that decreases as soil moisture increases

Processing Code — Text + Bar Display

```
import processing.serial.*;

Serial myPort;
String val = "";
int moisture = 0;

void setup() {
    size(400, 200);
    myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
    background(255);
    fill(0);
    textSize(20);
    text("Soil Moisture: " + moisture, 50, 60);

    // Visual bar
    fill(0, 100, 255);
    rect(50, 100, map(moisture, 1023, 300, 0, 300), 30);
    // Adjust range as needed
}

void serialEvent(Serial p) {
    val = p.readStringUntil('&apos;\n&apos;');
    if (val != null) {
        val = trim(val);
        if (val.matches("\\d+")) {
            moisture = int(val);
        }
    }
}
```

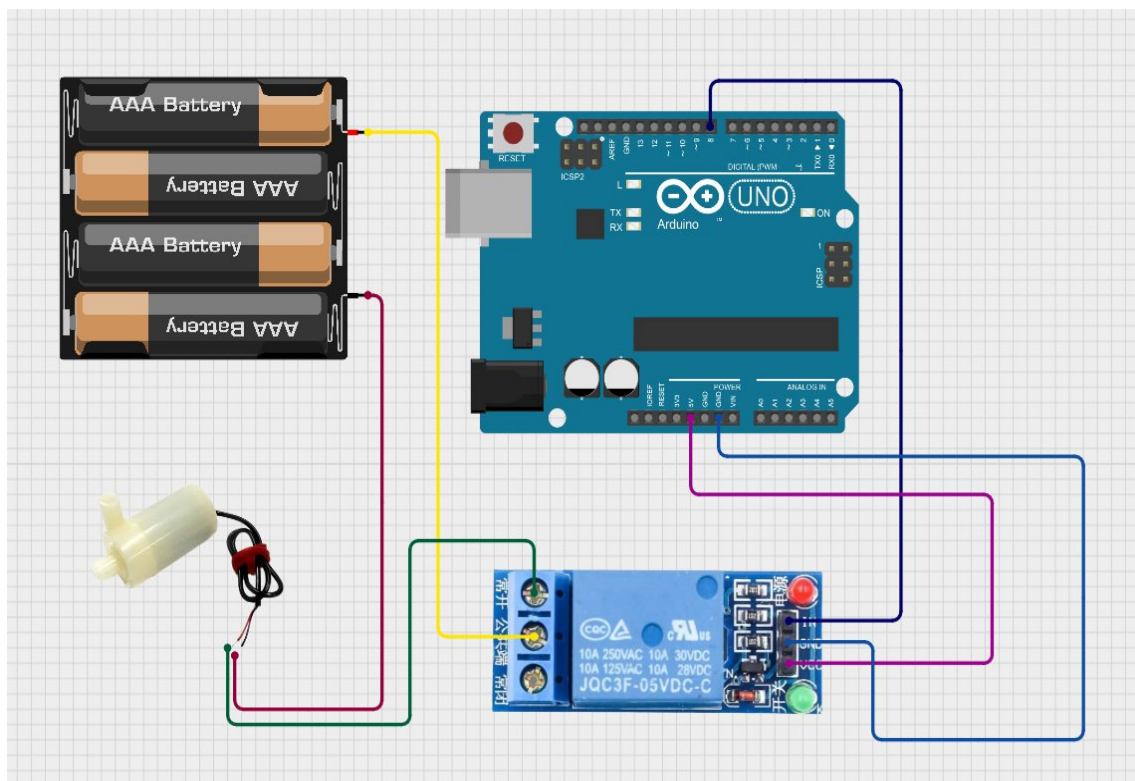
Exercise 2: Control Water Pump from Processing

Additional Components:

- Relay Module
- Water Pump
- External 12V power supply (for pump)

Wiring — Relay & Pump

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



Arduino Code

```
void setup() {
  Serial.begin(9600);
  pinMode(9, OUTPUT);
  digitalWrite(9, HIGH); // Ensure Pump starts OFF
}

void loop() {
  if (Serial.available() > 0) {
    while (Serial.available() > 1) { // Flush extra
bytes
      Serial.read();
    }

    char cmd = Serial.read();

    if (cmd == '1') {
      digitalWrite(9, LOW); // Pump ON
    } else if (cmd == '0') {
      digitalWrite(9, HIGH); // Pump OFF
    }
  }
}
```

```
import processing.serial.*;

Serial myPort;

void setup() {
  size(300, 100);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(200);
  fill(0);
  text("Press '1' to turn ON pump, '0' to turn
OFF", 20, 50);
}

void keyPressed() {
  if (key == '1' || key == '0') {
    myPort.write(key);
  }
}
```

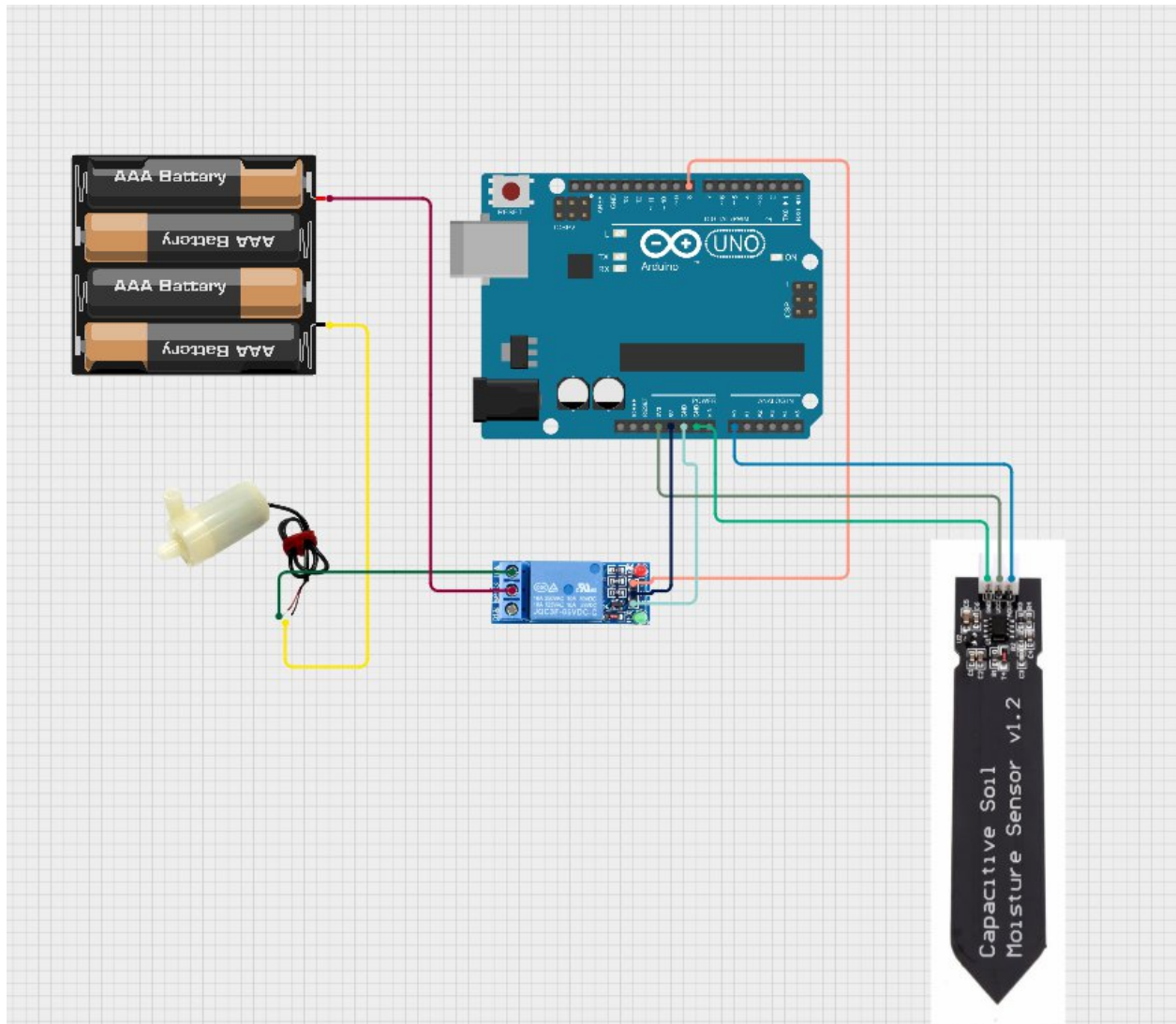
Proces
sing
Cod
e

Exe
rcis
e 3:

Full Automatic Irrigation System

🚀 This exercise is suitable as a complete project.

Goal: Combine sensor reading and pump control so watering happens automatically when soil is too dry, with real-time Processing display.



Arduino Code

```
#define RELAY_PIN 8
#define SENSOR_PIN A0
#define THRESHOLD 600

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

void loop() {
  int moisture = analogRead(SENSOR_PIN);
  Serial.println(moisture);

  if (moisture > THRESHOLD) {
    digitalWrite(RELAY_PIN, HIGH); // Dry → Water ON
  } else {
    digitalWrite(RELAY_PIN, LOW);  // Wet → Water OFF
  }

  delay(1000);
}
```

Processing Code — Status Display

```
import processing.serial.*;

Serial myPort;
String val;

void setup() {
  size(400, 200);
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  if (myPort.available() > 0) {
    val = myPort.readStringUntil('&apos;\n&apos;');
    if (val != null) {
      int moisture = int(trim(val));
      background(255);
      fill(0);
      textSize(20);
      text("Soil Moisture: " + moisture, 50, 100);
      if (moisture > 600) {
        fill(255, 0, 0);
        text("Status: DRY - Watering", 50, 140);
      } else {
        fill(0, 150, 0);
        text("Status: WET - Idle", 50, 140);
      }
    }
  }
}
```

Student Tasks

Level	Task
● Basic	Display moisture value and status text
● Intermediate	Color-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	Possible Cause
Pump does not turn on	Relay wiring incorrect or pin mismatch
Sensor always reads max	Sensor not in soil or damaged
Pump runs continuously	Threshold too high — lower THRESHOLD value
Processing shows no data	Wrong COM port or baud rate mismatch

✓ Tip: Use `Serial.list()` to find the correct COM port. Adjust `Serial.list()[0]` if needed.

Real-World Applications

Automatic irrigation systems and soil moisture monitoring are used in:

- smart gardens and home automation,
- agricultural farming and precision irrigation,
- greenhouse climate control systems,
- urban farming and vertical gardens,
- IoT platforms for environmental monitoring,
- research stations for soil and plant science,
- water-saving systems in arid regions.

Cross-Curricular Connections

Subject	Connection
Physics	Capacitance, analog voltage, electric circuits, relay switching
Biology & Ecology	Plant biology, soil composition, water cycle
Computer Science	Programming, serial communication, data processing
Technology & Engineering	Electronics, sensor integration, actuator control
Mathematics	Data scaling, threshold values, graph interpretation
Environmental Science	Water conservation, sustainable agriculture, climate

DigComp Competencies

The workshop develops the following DigComp 2.2 competencies:

DigComp Area	Competency Developed
1. Information & Data Literacy	Collecting and interpreting real-time sensor data
2. Communication & Collaboration	Presenting results and working in teams
3. Digital Content Creation	Programming in Arduino and Processing environments
4. Safety	Understanding safe use of electrical components and relays
5. Problem Solving	Diagnosing hardware faults and debugging code

Specifically, the workshop develops:

- digital literacy and computational thinking,
- problem solving and systematic debugging,
- real-time data acquisition and analysis,
- programming microcontrollers and visual interfaces,
- collaboration in a hands-on STEM environment.

Project Extension Ideas

Students who complete all exercises can extend the project with:

- Adding an OLED display to show moisture level directly on the device
- Sending irrigation data over a WiFi network using ESP8266 or ESP32
- Creating a web dashboard to monitor soil moisture remotely
- Connecting to a cloud platform (e.g. ThingSpeak, Blynk) for data logging
- Building a multi-zone irrigation system for several plant containers
- Adding a real-time moisture graph in Processing
- Integrating a weather API to skip watering on rainy days
- Adding a battery backup and solar panel for an off-grid system

Extension	Difficulty	New Skills
OLED display	🌀 Intermediate	I2C communication, display libraries
WiFi data sending	🌀 Advanced	ESP32/ESP8266, network programming
Cloud dashboard	🌀 Advanced	API integration, data visualization
Multi-zone system	🌀 Challenge	Multiple sensors, logic control
Solar-powered	🌀 Challenge	Power management, hardware design

Conclusion

This project serves as a practical introduction to automation, sensor-based control, and real-time data visualization. By combining the Arduino and Processing environments with a soil moisture sensor and relay module, students gain hands-on experience in electronics, programming, and environmental monitoring.

The automatic irrigation system demonstrates how simple digital tools can solve real-world problems — from conserving water in home gardens to managing large-scale agricultural operations. Through three progressive exercises, students develop skills that directly connect to IoT development, automation engineering, and sustainable technology.

Hackathon Challenges & Applied Learning Integration

In addition to guided workshop activities, the Digital Challenge program includes a series of competitive hackathon-style challenges designed to encourage creativity, independent problem solving, rapid prototyping, and the practical application of acquired STEM skills in realistic scenarios.

These challenges represent the transition from structured learning exercises toward open-ended engineering and problem-solving tasks. Students are expected to combine knowledge from electronics, physics, programming, sensor systems, serial communication, visualization, and automation in order to develop functional solutions under limited time conditions.

The hackathon format promotes:

- applied learning,
- collaborative problem solving,
- engineering design thinking,
- creativity and innovation,
- troubleshooting under pressure,
- and real-time decision making.

Unlike traditional classroom exercises, the challenges intentionally simulate authentic engineering situations in which students must:

- analyze the problem independently,
- select appropriate hardware and software tools,
- design circuits,
- develop algorithms,
- test and debug systems,
- and present working prototypes.

The challenge tasks are directly connected to concepts developed throughout the workshop collection.

Workshop Area	Related Hackathon Challenges
Voltage dividers & analog measurements	Unknown resistor measurement
LEDs & electrical measurements	LED turn-on voltage challenge
Light sensors & signal processing	Morse code optical communication
Temperature monitoring systems	Alien detection temperature challenge
Ultrasonic sensing & visualization	Parking sensor and radar systems
Real-time visualization in Processing	All Processing-based tasks
Automation & threshold systems	Irrigation and parking control logic

The progressive structure of the challenges enables students to move from:

- simple measurements,
- toward interactive systems,
- and finally to fully autonomous and communication-based projects.

Special emphasis is placed on:

- system stability,
- code organization,
- reliability of measurements,
- clarity of visualization,
- and successful demonstration of functional prototypes.

The fictional science-fiction scenarios used in several tasks (alien invasion, invisible robots, secure light communication) are intentionally designed to increase student engagement and motivation while still requiring the application of real scientific and engineering principles.

Through these activities, students develop practical competencies related to:

- sensor integration,
- embedded programming,
- serial communication,
- signal processing,

Pedagogical and Methodological Framework

- visualization,
- automation,
- and human-machine interaction.

The hackathon component also strengthens:

- teamwork,
- time management,
- presentation skills,
- resilience during debugging,
- and collaborative engineering workflows.

Assessment during the hackathon activities is based not only on obtaining correct technical results, but also on:

- design quality,
- functionality,
- creativity,
- efficiency,
- code structure,
- stability,
- and the ability to explain and demonstrate the developed solution.

This approach supports authentic STEM learning by connecting theoretical concepts with hands-on experimentation, rapid prototyping, and real-world engineering practices in a highly motivating project environment.

Hackathon Challenges – 1. day

Challenge 1.

Your task is to determine the unknown resistance values of the blue, red, and green resistors using the installed software, development systems, and components available at your workstation.

10 points total

- 2 points — correctly connected circuit
- 2 points — appropriate software
- 3 points — correctly determined blue resistor $R_1 = \underline{\hspace{2cm}}$
- 1.5 points — correctly determined red resistor $R_2 = \underline{\hspace{2cm}}$
- 1.5 points — correctly determined green resistor $R_3 = \underline{\hspace{2cm}}$

Challenge 2.

Using the available components, development systems, and installed software, determine the turn-on voltage of red/yellow/green LED diodes (the voltage across the diode at which it begins to emit light). The current through the LED must not exceed 50 mA.

12 points total

- 4 points — circuit without exceeding 50 mA
- 3 points — correct measurement method
- 2 points — result for red LED $U_1 = \underline{\hspace{2cm}}$
- 1 point — result for yellow LED $U_2 = \underline{\hspace{2cm}}$
- 1 point — result for green LED $U_3 = \underline{\hspace{2cm}}$
- 1 point — explanation / presentation of the measurement

Challenge 3.

In the near future, an alien invasion has occurred. The only communication channel they cannot control is light-based communication. Your task is to send a general call, **CQ**, in Morse code to anyone who can see your signal. The optimal current for operating the LED used to transmit the signal is 15 mA. A dot lasts 200 ms. Using the installed software, development systems, and components available at your workstation, solve this problem.

15 points total

- 3 points — correct resistor calculation
- 2 points — correctly connected circuit
- 4 points — correct Morse code for C
- 4 points — correct Morse code for Q
- 2 points — correct spacing / signal repetition

Challenge 4.

Aliens have invisible robots that can only be detected by measuring the temperature in a room. If the temperature, without any visible external cause, exceeds 30°C, there is a possibility that an invisible robot is nearby. Measure the temperature in the classroom using the available components, development systems, and installed software.

a) The system must operate independently of a computer and provide a light signal when the temperature exceeds 30°C.

b) Display the measured values on a computer using Processing.

a) 10 points • 2 points — correctly connected sensor • 3 points — accurate temperature reading • 3 points — correct response at 30°C • 2 points — stable operation without a computer	b) 10 points • 3 points — serial data transmission • 3 points — correct display of values • 2 points — real-time updates • 2 points — clarity of the display
--	---

Challenge 5.

Due to increasing traffic congestion, it is necessary to park vehicles as close to each other as possible. Therefore, parking sensors are needed to indicate the distance between a vehicle and an obstacle. Your task is to design a simple parking sensor using the available resources, which displays the proximity of the vehicle using a green and a red LED.

The green LED shines at full brightness as long as the distance between the vehicle and the obstacle is greater than 15 cm. Its brightness decreases as the vehicle approaches the obstacle. For distances less than 10 cm, the red LED turns on, and its brightness increases to maximum as the distance decreases to 5 cm. For distances less than 5 cm, the red LED starts blinking rapidly.

a) The system must operate independently of a computer.

b) Design your own way of displaying the distance and present it on a computer using Processing.

a) 15 points • 3 points — correct distance measurement • 3 points — green LED works for >15 cm • 3 points — green LED decreases as the obstacle approaches • 3 points — red LED increases in the 10–5 cm range • 3 points — red LED blinks rapidly for <5 cm	b) 10 points • 4 points — distance display in Processing • 3 points — logical and clear visualization • 3 points — originality of the display
---	--

Special Challenge.

The only secure communication channel during an alien invasion is light. Morse code and the first five letters of the alphabet are used to send encoded messages via light signals. A dot in Morse code lasts 200 ms. Using the installed software, development systems, and components available at your workstation, create a prototype device that will display the received light message. For decoding, use the table:

Code	Symbol	Code	Symbol	Code	Symbol
AAA	A	ADA	M	BAC	Y
AAB	B	ADB	N	BAD	Z
AAC	C	ADC	O	EAA	0
AAE	D	ADD	P	EAB	1
ABA	E	ADE	Q	EAC	2
ABC	F	AEA	R	EAD	3
ABD	G	AEB	S	EAE	4
ABE	H	AEC	T	EBA	5
ACA	I	AED	U	EBB	6
ACB	J	AEE	V	EBC	7
ACD	K	BAA	W	EBD	8
ACE	L	BAB	X	EBE	9

50 points total

- 4 points — correct detection of light pulses
- 4 points — proper distinction between short/long signals
- 10 points — correct recognition of letters
- 10 points — correct recognition of three-letter codes
- 5 points — system stability
- 7 points — program organization
- 10 points — successful demonstration on a test example

Hackathon Challenges – 2. Day

Challenge 1: Measuring Distance with ESP32 and HC-SR04

Objective: Connect an ultrasonic sensor to the ESP32, measure distance, and visualize data.

Total points: 35

Task 1 — Draw the Circuit [5 pts]

Using Circuit Designer (<https://app.circuitdesigner.com/>), draw the schematic connecting the ESP32 to the HC-SR04 ultrasonic sensor.

- Clearly label all pins: VCC, GND, TRIG, ECHO.
- Export or screenshot and include in your submission folder.

💡 *ESP32 operates at 3.3 V logic; the HC-SR04 ECHO pin outputs 5 V — use a voltage divider or level shifter if required.*

Task 2 — Hardware Setup [5 pts]

Physically connect the ESP32 and HC-SR04 on a breadboard following your schematic.

- Double-check VCC (5 V or 3.3 V) and GND before powering on.
- The evaluator will inspect your breadboard connections.

Task 3 — Arduino IDE Program [10 pts]

Write a C++ program (Arduino sketch) that:

- Sends a 10 μ s trigger pulse on the TRIG pin.
- Measures the echo duration on the ECHO pin using `pulseIn()`.
- Calculates distance in centimetres (speed of sound = 0.034 cm/ μ s).
- Prints the distance to Serial Monitor every 500 ms.

Test at a minimum of two different distances and compare with a ruler.

Task 4 — Verification [5 pts]

Open the Serial Monitor (9600 baud). Capture a screenshot showing:

- At least 5 consecutive readings.
- Two clearly different distance values.

Task 5 — Processing Visualization [10 pts]

Write a Processing (Java mode) sketch that:

- Reads distance values from the serial port.
- Displays the numeric distance value on screen.
- Draws a filled rectangle whose width is proportional to the measured distance.

💡 *Use Serial library in Processing. Match baud rate with Arduino sketch (9600).*

Challenge 2: Controlling a Servo Motor with Distance

Objective: Use the ultrasonic sensor reading to control a servo motor position.

Total points: 30

Task 1 — Draw the Circuit [5 pts]

In Cirkuit Designer, draw the connection between ESP32, SG90 servo, and HC-SR04.

- Label all connections clearly.
- Show the servo signal pin connection.

Task 2 — Hardware Setup [10 pts]

Connect ESP32, SG90 servo, and HC-SR04 on a breadboard.

- Power the servo from the external 5 V rail (or USB), NOT directly from ESP32 GPIO.
- Confirm stable operation before programming.

💡 *SG90 servo draws up to 250 mA — use an external power source to avoid resetting the ESP32.*

Task 3 — Arduino IDE Program [10 pts]

Write a C++ program that:

- Reads distance from HC-SR04 continuously.
- Rotates the servo to 0° (left) when distance < 10 cm.
- Rotates the servo to 180° (right) when distance > 20 cm.
- Holds the servo at 90° (centre) for distances between 10–20 cm.
- Prints distance and servo angle to Serial Monitor.

Task 4 — Verification [5 pts]

Demonstrate the servo reacting correctly to three distance ranges. The evaluator will hold an object at different distances during assessment.

Challenge 3: Sonar Scanner with Processing Visualization

Objective: Build a rotating sonar system and render a radar-style display in Processing.

Total points: 50

Task 1 — Draw the Circuit [5 pts]

In Cirkuit Designer, draw the full connection (ESP32 + SG90 + HC-SR04).

- The sensor and servo should be shown mounted together on a holder.

Task 2 — Hardware Setup [10 pts]

Physically assemble the sonar unit:

- Mount HC-SR04 on top of the SG90 servo arm so it sweeps left/right.
- Connect power and signal wires without restricting rotation.

Task 3 — Arduino IDE Program [15 pts]

Write a C++ program that:

- Sweeps the servo from 0° to 180° and back in steps of 1°.
- Measures distance at each angle.
- Sends data over Serial in the format: ANGLE,DISTANCE\n
- Includes a short delay (≤ 30 ms per step) to allow sensor readings to stabilize.

Task 4 — Verification [5 pts]

Open Serial Monitor. Confirm angle values sweep 0–180 and distance values are plausible.

Task 5 — Processing Visualization [15 pts]

Write a Processing sketch that:

- Reads ANGLE,DISTANCE lines from the serial port.
- Renders a radar-style display: green sweep line on a black background.

Pedagogical and Methodological Framework

- Plots a dot or arc at the measured distance for each angle.
- Adds angle and distance labels around the display.

💡 Use polar-to-Cartesian conversion: $x = \text{dist} * \cos(\text{angle})$, $y = \text{dist} * \sin(\text{angle})$.

Challenge 4 (Advanced): Wi-Fi Sonar with Android Processing

Objective: Transmit sonar data over Wi-Fi (UDP) and visualize it on an Android Processing app.

Total points: 30 | Bonus challenge — attempt only after completing Challenges 1–3.

Task 1 — Arduino IDE Program [15 pts]

Write a C++ program that:

- Connects the ESP32 to a known Wi-Fi network (SSID/password in code).
- Sweeps servo 0°–180° and measures distance at each step.
- Sends ANGLE,DISTANCE as a UDP packet to a specified Android IP and port.
- Repeats the sweep continuously.

💡 Use the *WiFiUDP* library on ESP32. Confirm ESP32 and Android are on the same network.

Task 2 — Android Processing Visualization [15 pts]

Write a Processing for Android sketch that:

- Listens for UDP packets on the specified port.
- Parses ANGLE,DISTANCE from each packet.
- Renders a full radar-style display with sweep line and distance dots.
- Runs smoothly at ≥ 20 fps on an Android device.

💡 Use the *ketai* or *processing.net* UDP library for Android mode. Test on a real device if possible.

Pedagogical and Methodological Framework

Introduction

The Digital Challenge workshop collection is designed as a hands-on STEM educational program that combines electronics, programming, physics, engineering, data visualization, and real-world problem solving through project-based learning activities.

The workshops are based on practical experimentation using microcontrollers, sensors, actuators, and visualization tools, enabling students to develop both technical and transversal competencies through active participation and collaborative work.

The educational approach promotes:

- learning by doing,
- inquiry-based learning,
- experimentation,
- computational thinking,
- interdisciplinary problem solving,
- creativity and innovation.

The materials are suitable for:

- formal education,
- extracurricular STEM activities,
- makerspaces,
- hackathons,
- robotics clubs,
- non-formal learning environments,
- and teacher training activities.

Educational Philosophy

The workshop collection is based on the idea that students learn most effectively when they actively design, build, test, analyze, and improve real systems.

Instead of focusing only on theoretical explanations, students:

- interact with physical devices,
- collect and interpret real-world data,
- develop functional prototypes,
- visualize information,
- and solve authentic engineering problems.

The methodology encourages:

- experimentation,
- iterative design,
- troubleshooting,
- teamwork,
- and reflective learning.

Special emphasis is placed on connecting scientific concepts with practical applications in everyday life and modern technology.

Methodological Approach

The workshops combine several complementary educational approaches:

Collaborative & Peer Learning

The workshops are designed to support:

- teamwork,
- collaborative problem solving,
- peer mentoring,
- knowledge sharing,
- and presentation of project results.

Students are encouraged to discuss solutions, compare approaches, and collaboratively debug hardware and software systems.

Project-Based Learning (PBL)

Students learn through the development of complete functional projects such as:

- environmental monitoring systems,
- radar simulations,
- smart irrigation systems,
- automated sensor systems,
- and interactive visual interfaces.

Projects progress from basic functionality toward more advanced extensions and open-ended challenges.

Inquiry-Based Learning

Students are encouraged to:

- explore sensor behavior,
- test hypotheses,
- compare measurements,
- calibrate systems,
- identify errors,
- and improve performance through experimentation.

The workshops promote scientific thinking and evidence-based reasoning.

Challenge-Based Learning

Each workshop includes:

- progressive exercises,
- practical tasks,
- troubleshooting situations,
- and advanced challenges.

This structure allows students with different experience levels to participate successfully while still providing opportunities for deeper exploration.

STEM Integration

The workshop collection integrates multiple STEM disciplines within a unified practical framework.

Discipline	Examples
Physics	Sound propagation, voltage, resistance, capacitance, sensors
Electronics	Circuits, voltage dividers, relays, actuators
Computer Science	Programming, serial communication, data processing
Engineering	System integration, automation, prototyping
Mathematics	Data analysis, scaling, graphical representation
Environmental Science	Irrigation systems, sustainability, environmental monitoring

The interdisciplinary structure helps students understand how scientific and technological concepts interact in real-world systems.

DigComp & Digital Competencies

The workshops support the development of competencies aligned with the DigComp 2.2 framework, including:

DigComp Area	Competencies Developed
Information & Data Literacy	Collecting, interpreting, and visualizing sensor data
Communication & Collaboration	Teamwork and presentation of project results
Digital Content Creation	Programming microcontrollers and visual interfaces
Safety	Safe handling of electronic components and systems
Problem Solving	Debugging hardware and software systems

Additionally, the workshops develop:

- computational thinking,
- algorithmic reasoning,
- digital literacy,
- creativity,
- critical thinking,
- and engineering design skills.

21st Century Skills

The workshop activities promote:

- critical thinking,
- creativity,
- collaboration,
- communication,
- adaptability,
- initiative,
- problem solving,
- and technological literacy.

Students develop both technical competencies and transversal skills relevant to future STEM education and careers.

Assessment Methodology

Assessment may include:

- successful hardware implementation,
- code functionality,
- problem-solving process,
- debugging ability,
- data interpretation,
- creativity of solutions,
- teamwork and collaboration,
- project presentation,
- and documentation quality.

The workshops support both:

- formative assessment during activities,
- and summative evaluation through final projects and demonstrations.

Inclusion & Accessibility

The workshop structure supports inclusive STEM education through:

- progressive difficulty levels,
- beginner-to-advanced tasks,
- collaborative work,
- peer support,
- visual learning approaches,
- and practical hands-on activities.

The materials are designed to encourage participation from students with different levels of prior experience and to support broader engagement in STEM fields.

Special attention can be given to:

- encouraging girls in STEM,
- supporting underrepresented groups,
- and adapting activities for different educational contexts.

Safety Guidelines

Students should:

- disconnect power before modifying circuits,
- avoid short circuits,
- keep liquids away from electronic components,
- handle sensors and actuators carefully,
- and verify wiring before powering systems.

Projects involving relays, pumps, or external power supplies require additional supervision and attention to electrical safety.

Reflection & Evaluation

Reflection activities are included to encourage deeper understanding of:

- system behavior,
- engineering decisions,
- troubleshooting processes,
- and real-world applications.

Example reflection questions:

- What was the biggest technical challenge?
- How could the system be improved?
- What real-world problem does this project solve?
- How reliable are the measurements?
- What additional sensors or features could be added?

Real-World Relevance

The workshop collection demonstrates how simple digital systems can be applied to:

- environmental monitoring,
- smart agriculture,
- automation,
- robotics,
- smart homes,
- IoT systems,
- industrial monitoring,
- and interactive visualization.

Students gain experience with technologies and concepts directly connected to modern engineering and digital systems.

Replication & Adaptation

The workshops are designed to be:

- low-cost,
- modular,
- scalable,
- and replicable.

They can be adapted for:

- short workshops,
- semester projects,
- hackathons,
- STEM camps,
- classroom teaching,
- and online or hybrid learning environments.

Most activities use widely available hardware components and open-source software tools.