جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

أكاديمية كاوست
KAUST ACADEMY

Internet of Things (IoT)

DAY 1 What IoT is, and your first circuit running your own code

August 2 to 6, 2026

KGSP Summer Enrichment Program 2026
KAUST Academy, KAUST

Salah Abdeljabar
salah.abdeljabar@kaust.edu.sa

🗺️ On Today's Agenda

1. **What is IoT?**: the definition, the origin, and the scale
2. **Sensors & actuators**: the two halves of every device
3. **Microcontrollers**: tiny computers built for one job, and why the ESP32
4. **Your board**: the pinout, the power pins, and which GPIOs are safe
5. 🔧 **Set up the toolchain**: Arduino IDE, board package, first upload
6. **Just enough electricity**: circuits, voltage, current, Ohm's Law, polarity
7. 🔧 **Project 1**: a pushbutton turns an LED on and off

By the end you will have your own code running on your own board, with a circuit you wired yourself.

👋 The Teaching Team

Salah Abdeljabar
Course Instructor

Fawaz Al-Ghamdi
Teaching Assistant

Meshari Alsughayyir
Teaching Assistant

Dark

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

أكاديمية كاوست
KAUST ACADEMY

Internet of Things (IoT)

A five-day introduction, from a single sensor to a connected city.

August 2 to 6, 2026 · KGSP Summer Enrichment Program 2026

DAY 1

What IoT is, and the devices that make it work

Open slides →

PDF

Workshop Website 🌐

Everything for this workshop lives in one place:

- 📚 **Slides & materials** for every day
- 🛠️ **Hands-on labs** and code examples
- 🔗 **Resources** and further reading

Visit it live at:

2026-iot.abdeljabar.net

What is IoT? 🌐

The **Internet of** Things

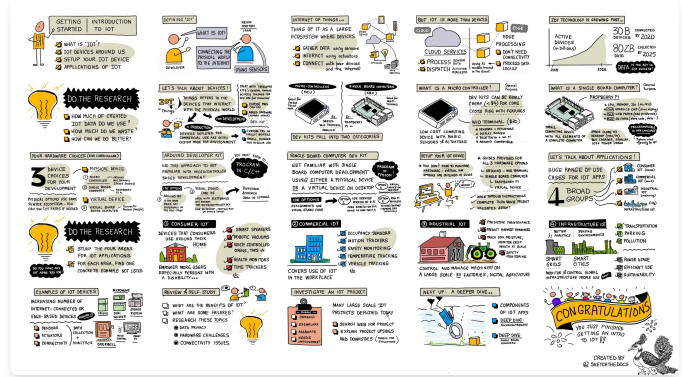
The Origin of IoT

The term **Internet of Things** was coined by Kevin Ashton in **1999**, to describe connecting the Internet to the **physical world** through sensors.

Today it means **any device that interacts with the world around it**, by:

- 📥 **Gathering data** with sensors: temperature, speed, location
- 📤 **Acting on the world** with actuators: switches, lights, sound

The **T** in IoT stands for **Things**: the devices bridging the **digital** and **physical** worlds.



Sketchnote by Nitya Narasimhan

In a nutshell

Everyday **things** + **sensors / actuators** + the **Internet** (and the cloud behind it).

The Scale of IoT 📈

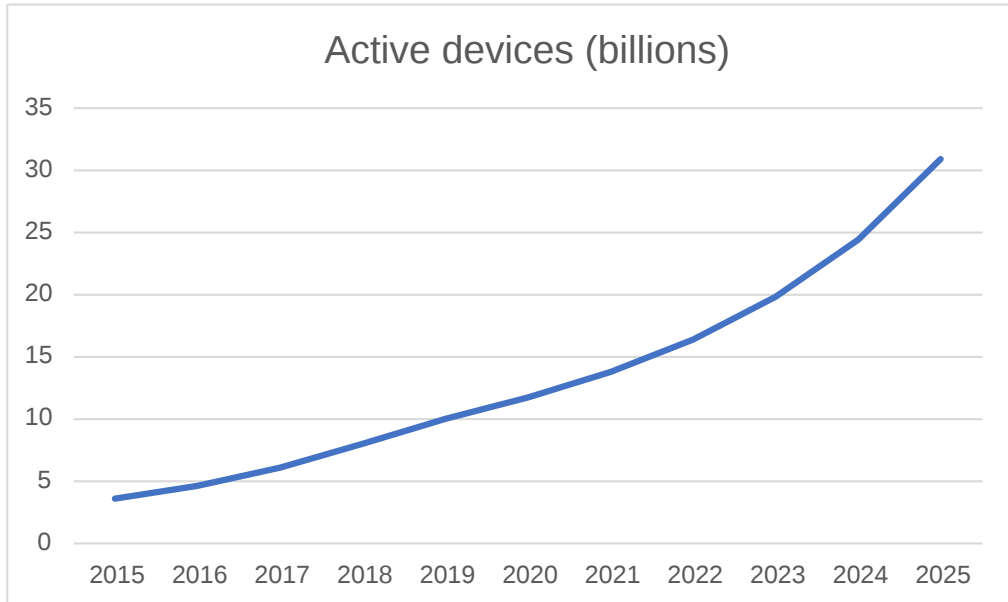
From a handful of devices to tens of billions

Explosive Device Growth

By the end of **2020**, an estimated 30 billion IoT devices were connected to the Internet.

- Under **5 billion** in 2015
- Over **30 billion** by 2025
- Growth shows no sign of slowing

> **Tip**
>
> Roughly **4 connected devices for every person** on Earth.



An Ocean of Data 🌊

Every one of those devices generates data, and the totals are staggering.

By 2025

IoT devices will gather almost **80 zettabytes** of data: that's **80 trillion gigabytes**.

Why it matters

This data is the **key to IoT's success**: the value comes from understanding it and acting on it.

Successful IoT means knowing what data to gather, how to gather it, and how to make decisions from it.

Sensors & Actuators ⚙️

How IoT devices feel and act on the physical world

Two Halves of Every IoT Device

📥 Sensors: *input*

Gather information **from** the physical world by converting what they measure into an electrical signal.

- Temperature, speed, location
- Light, sound, movement
- Pressure, humidity

📤 Actuators: *output*

Convert an electrical signal **into** a real-world interaction.

- Turn on a switch or light
- Make a sound
- Move a motor

The Core Idea

Note

Sensors **read** the world and turn it into an electrical signal. Actuators take an electrical signal and use it to change the world. Many devices use both.

Everyday example

A **smart thermostat**: a temperature **sensor** reads the room, an **actuator** switches the heater on or off.

Microcontrollers (MCUs) 

Tiny computers-on-a-chip built for one job

What is a Microcontroller?

A microcontroller (**MCU**) is a small, low-power computer on a single chip:

🧠 CPU	💾 Memory	🔌 I/O
Runs your program, one or more cores	Program storage + RAM, on the chip	Pins to talk to sensors & actuators

- Low cost: often **US$0.03–0.50**, dev kits from ~US$4
- Low power: can run **months or years** on a battery
- Programmed in **C/C++** to do a few very specific tasks

> **Tip**
> Searching for "**MCU**"? Add "microcontroller", or you'll get the Marvel Cinematic Universe!

The ESP32: Our Workshop MCU

This workshop uses the `ESP32`, a popular and capable microcontroller.

- **Dual-core** processor
- **Wi-Fi + Bluetooth** built in
- Dozens of **GPIO** pins
- Cheap and widely supported

How we program it

The **Arduino framework** in C/C++: `setup()` runs once, `loop()` runs forever.

```
void setup() {
  Serial.begin(115200);
  Serial.println("Hello, IoT!");
}

void loop() {
  // runs forever
}
```

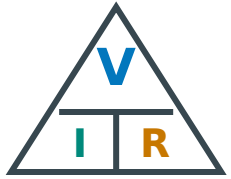
🌙 Dark

ESP32 Basic Starter Kit

A hands-on course that takes you from your first blinking LED to Wi-Fi web servers, cloud dashboards, and an IoT device you build yourself.

LAFVIN Basic Starter Kit · ESP32 DEVKIT V1

Basics · Sensors · Wi-Fi / Web · Display · Cloud / MQTT · Capstone · Reference



ESP32 Basic Starter Kit

🧰

A hands-on companion to this workshop: **3 foundation guides**, **13 build projects**, and **2 reference pages**, each with wiring, annotated code, and troubleshooting.

- 🪛 First blinking LED to Wi-Fi web servers, a cloud dashboard, and a chat bot
- 📖 Every project is a full write-up plus an illustrated guide page
- 🎓 A capstone project to design and demo your own device

Open the guide ↗

FOUNDATION 2 · BOARD TOUR

Introduction, Board Tour & Setup

Kit: LAFVIN Basic Starter Kit for ESP32 · **Board:** ESP32 DEVKIT V1 (ESP-WROOM-32, 30-pin)

The foundation for every other project: what the ESP32 is, how to read its pins, and how to upload your first sketch.

Brand new to electronics? Read Foundation 1: Electronics & Communication Basics first.

🎯 Objective

Foundation 2 · Board Tour 🔍

What the ESP32 is, and a tour of the DEVKIT V1 board before any wiring starts.

- Find the USB port, regulator, EN/BOOT buttons, and header rows
- Read the pinout: which GPIOs are safe, input-only, or off limits
- Tell digital pins from analog-capable ones
- Use the BOOT-button trick when an upload won't connect

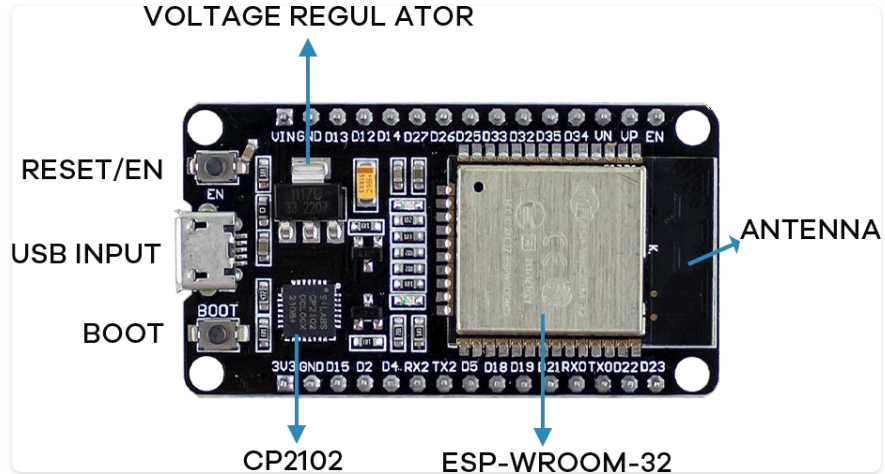
Open the guide ↗

What the ESP32 Is

A low-cost **System-on-a-Chip** from Espressif: one part holds the processor, the radios, and the peripherals.

- **Dual-core** 32-bit CPU, up to **240 MHz**
- **Wi-Fi** (2.4 GHz) and **Bluetooth** on the chip
- **520 KB** SRAM, **4 MB** flash (typical)
- ADC, DAC, PWM, I²C, SPI, UART, touch

> **Note**
>
> The board is not the chip. The **DEVKIT V1** wraps it with a USB port, a 3.3 V regulator, buttons, and headers so it is easy to program.



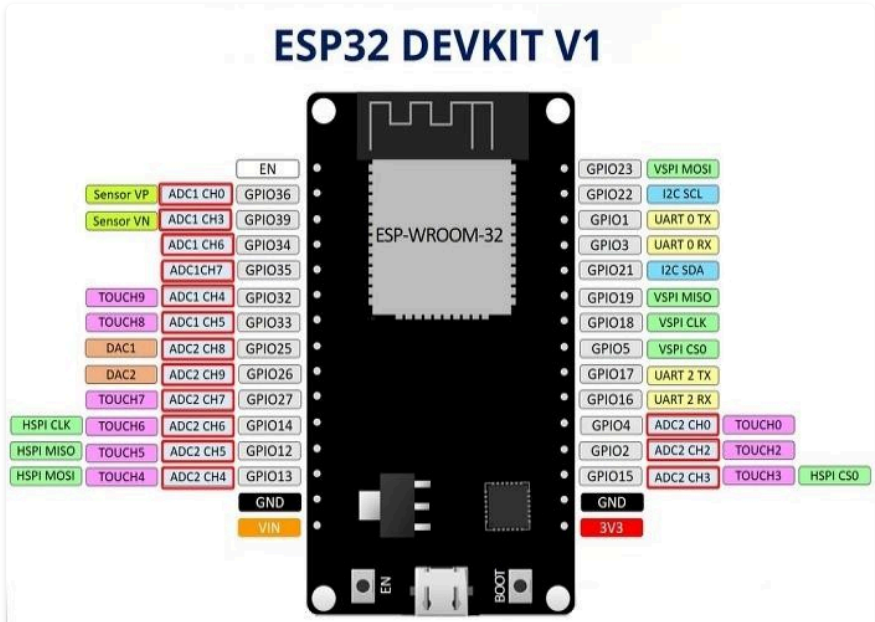
USB, regulator, EN/BOOT buttons, and the ESP-WROOM-32 module.

The Pinout and Power

Most pins are **multiplexed**: you pick their job (UART, I²C, SPI, PWM) in code. A few are fixed in hardware.

- **I²C:** SDA 21, SCL 22
- **SPI:** MOSI 23, MISO 19, CLK 18, CS 5
- **UART0 (USB):** TX0 1, RX0 3
- **DAC:** 25 and 26, true analog out

> ⚠ **Warning**
>
> Power logic from `3V3`. `VIN` is the raw **5 V** from USB, for 5 V modules like the PIR and relay. GPIOs are **3.3 V**: never put 5 V on one.



Not Every Pin Is Equal

Pick from the **safe** list first. Some pins decide how the board boots, some are input-only, and a few are wired to internal flash and must never be touched.

Category	Pins	Notes
✅ Safe, general purpose	4, 13, 16, 17, 18, 19, 21, 22, 23, 25, 26, 27, 32, 33	First choice for input and output
⚠️ Strapping	0, 2, 5, 12, 15	Set boot mode; a wrong level can block uploading
🟨 Input-only	34, 35, 36, 39	No output, no internal pull-up or pull-down
⛔ Do not use	6, 7, 8, 9, 10, 11	Wired to the internal SPI flash

- **Current:** keep each pin under about **12 mA** (40 mA absolute), which is why every LED gets a **220 Ω** resistor.

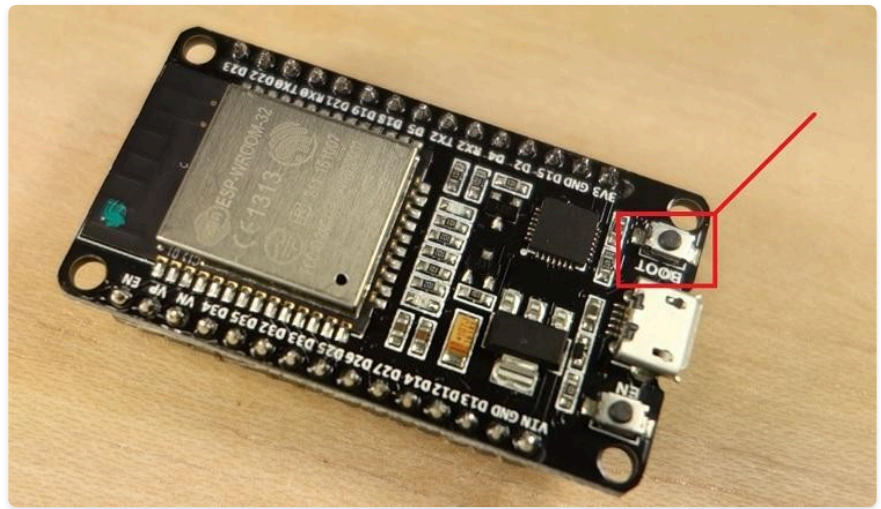
Uploading and the BOOT Trick

Click **Upload**, wait for **"Done uploading"**, then open the **Serial Monitor** at **115200** baud (every sketch uses this rate).

If it stalls at `Connecting...`:

1. Hold the **BOOT** button
2. Click **Upload**
3. Release **BOOT** when you see `Connecting...`
4. Press **EN/RESET** to run the sketch

> **Tip**
>
> Garbage in the Serial Monitor is almost always the wrong baud rate. Set it to **115200**.



FOUNDATION 3 · TOOLCHAIN

Setting up the Arduino IDE for ESP32

A picture-by-picture install guide. Do it once, then every project just needs **Upload**.
Brand new? See Foundation 2: Introduction & Setup first.

> **The short version:** install Arduino IDE → add the ESP32 boards URL → install the **esp32** package → pick the **DOIT ESP32 DEVKIT V1** board and its port → upload a test sketch.

Foundation 3 · IDE Setup

Install the Arduino IDE, add ESP32 board support, and upload a first sketch, step by step with screenshots.

- Add ESP32 support through the Boards Manager URL
- Select the right **board** and **port**, the two settings that trip up most first days
- Upload a sketch you didn't write, to confirm the whole toolchain works
- Recover from an upload that won't connect, using the BOOT button

Open the guide ↗

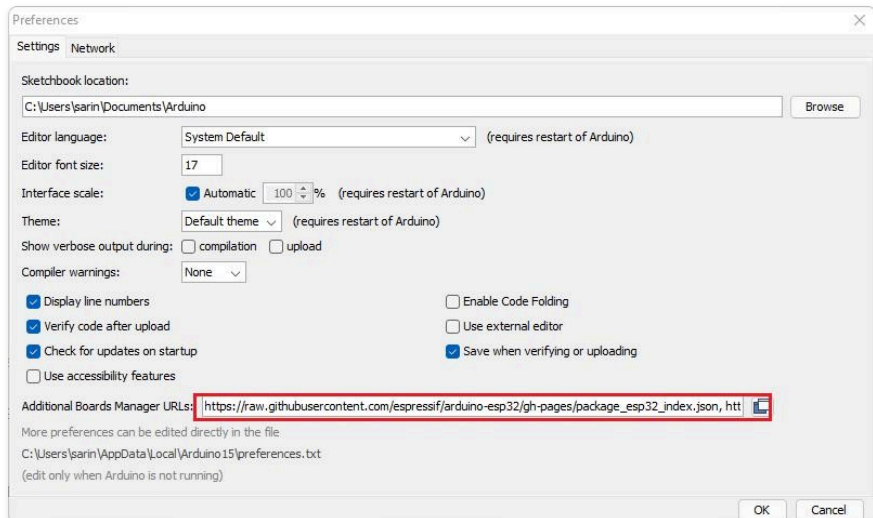
Add ESP32 Board Support

The ESP32 is not built into Arduino. You point the IDE at Espressif's **package index**, and it learns to compile for the chip.

Open **File → Preferences** and paste this into **Additional Boards Manager URLs**:

```
https://raw.githubusercontent.com/espressif/
arduino-esp32/gh-pages/package_esp32_index.json
```

Any other board family is added the same way. Separate multiple URLs with a comma.

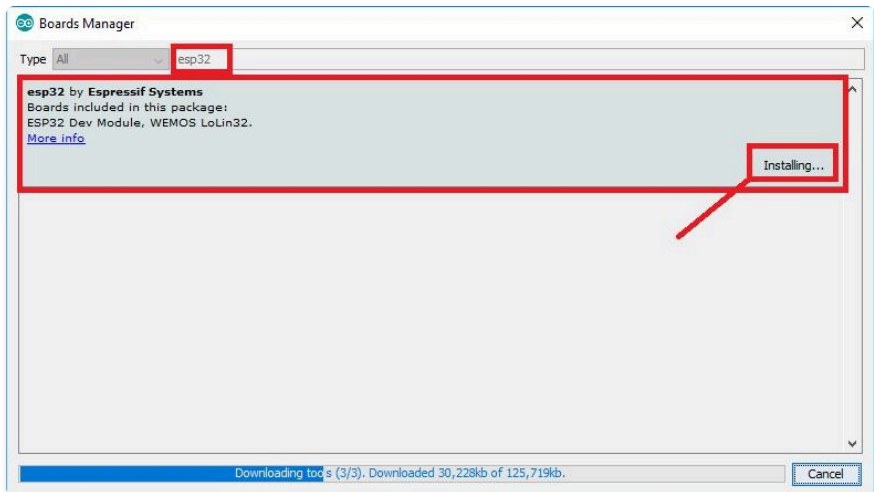


Install the esp32 Package

Open **Tools → Board → Boards Manager**, search **esp32**, and install **"esp32 by Espressif Systems"**.

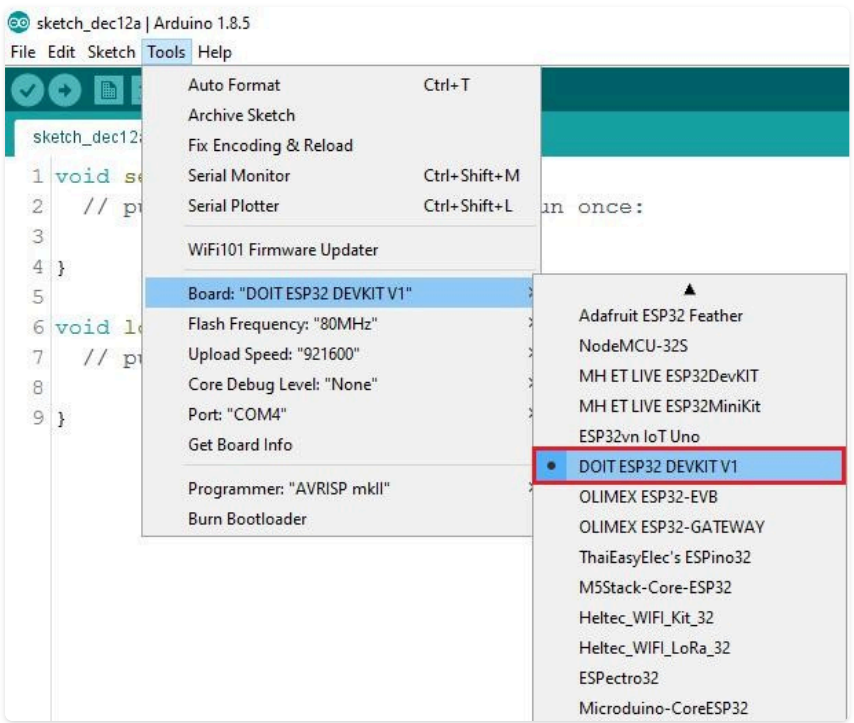
- Wait for the download to finish and show **INSTALLED**
- Then pick **Tools → Board → DOIT ESP32 DEVKIT V1**

> ⚠ **Warning**
>
> Take **version 3.0.0 or newer**. Core 3.x changed the PWM functions, and Projects 3 and 6 use the newer `ledcAttach()` form.

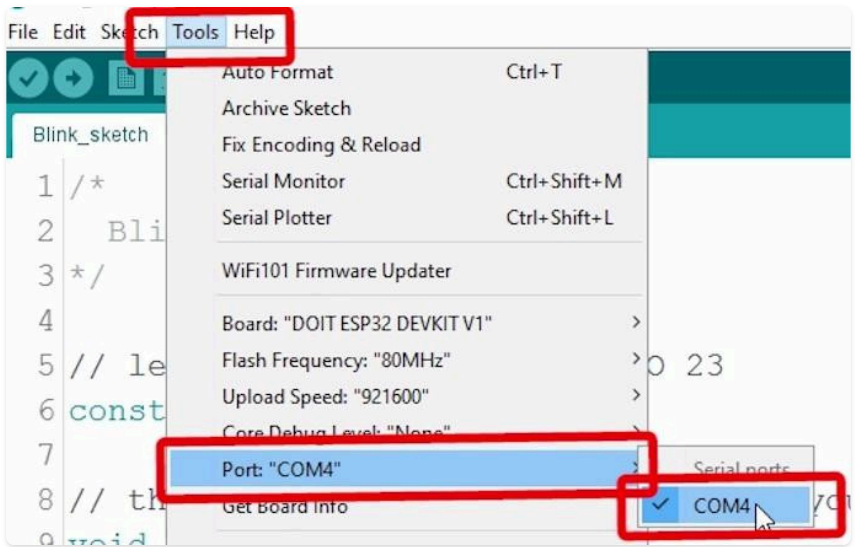


Board and Port: the Two First-Day Traps

The **board** decides how code is compiled; the **port** decides where it is sent. Getting either wrong gives an error that does not name the cause.



Tools → **Board** → DOIT ESP32 DEVKIT V1



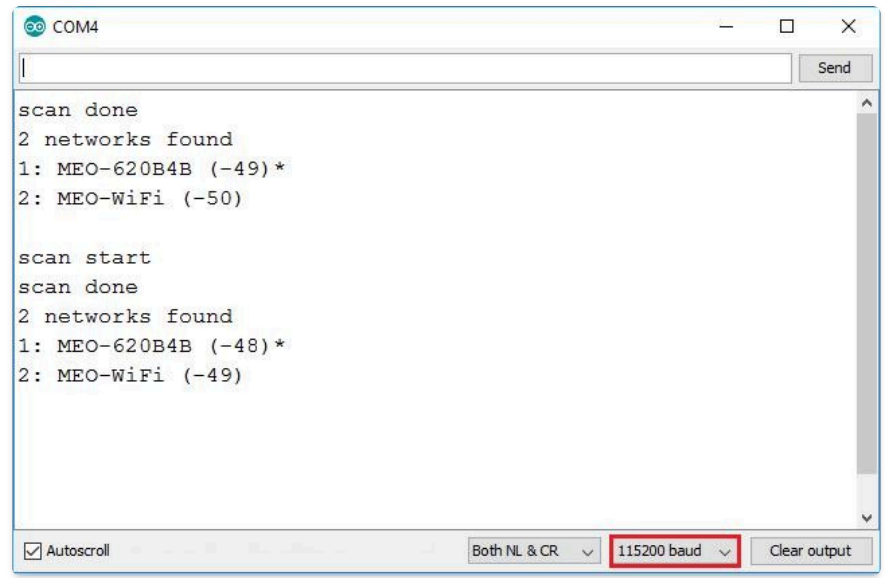
Tools → **Port** → your board's port (`COM4` , `/dev/ttyUSB0`)

Prove It Works: WiFiScan

Before wiring anything, upload a sketch you did not write, so a later problem is your circuit, not the toolchain.

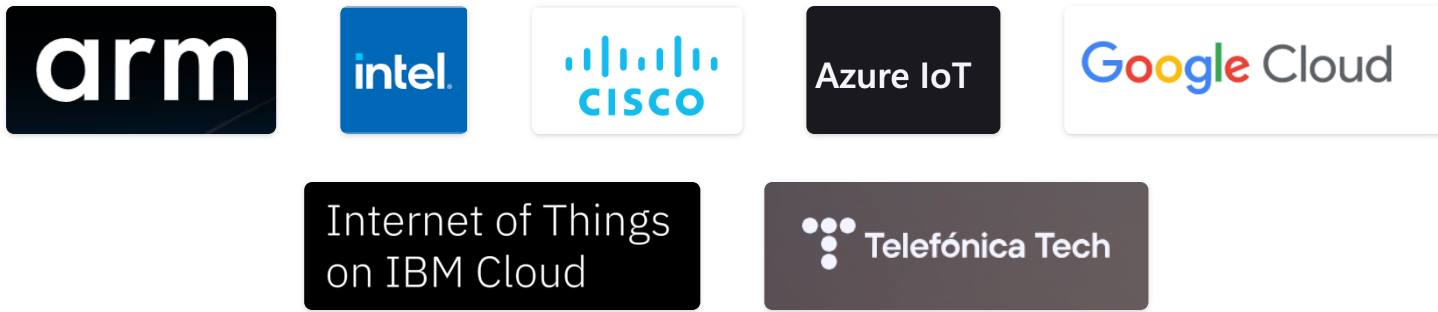
- **File → Examples → WiFi (ESP32) → WiFiScan**
- Click **Upload**, wait for **"Done uploading"**
- Open **Serial Monitor** at **115200**, press **EN/RESET**
- Nearby Wi-Fi networks appear: the toolchain works

> **Note**
>
> If upload stalls at `Connecting...`, hold **BOOT**, click Upload, release **BOOT** when it connects.



Everyone Is Betting on IoT 🏢

Every major technology company is active in this space, from the silicon up to the cloud.



Note

Chip makers, network vendors, cloud providers, and telecom operators are all investing here: a sign that IoT is infrastructure, not a niche.

No Full OS: Just Enough

MCUs are too small for a desktop OS. Two common approaches let you program them:

⚙️ RTOS

A lightweight **real-time OS** for handling peripherals on time.

- FreeRTOS, Zephyr, Azure RTOS
- Multi-threading, networking, GUI

🔧 Arduino framework

The most popular choice for learning and making.

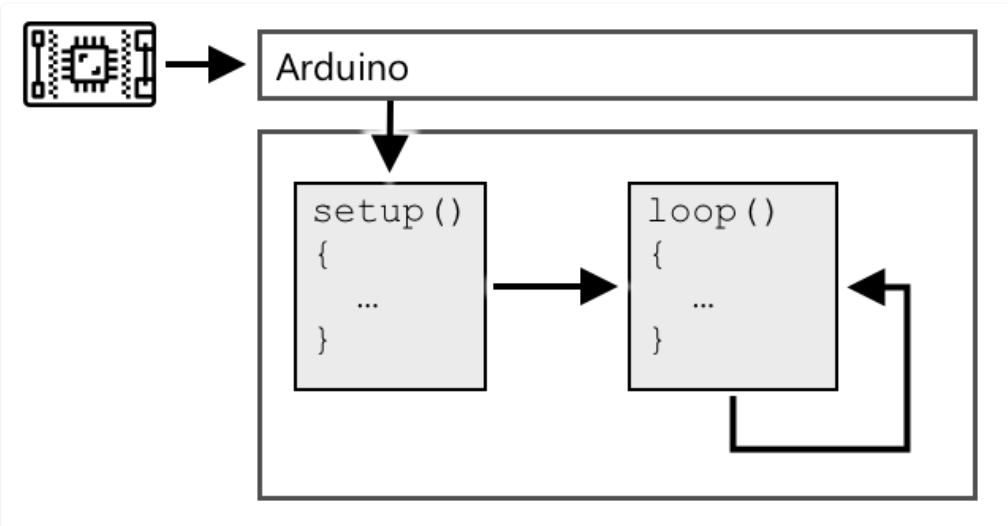
- Open-source, coded in **C/C++**
- Huge library ecosystem
- Portable across boards

The Arduino Sketch

An Arduino program is built from just two functions:

- `setup()` : runs **once** at start (Wi-Fi, pins…)
- `loop()` : runs **forever**, again and again

> **Tip**
>
> This **event-loop** pattern quietly powers most desktop apps too.



What Is a Circuit? ⚡

The closed loop everything electrical is built on

Voltage: The Push

Every electrical source has **two terminals**: one held at a **higher** voltage (+) and one at a **lower** voltage (–).

- Electricity wants to flow from **high** voltage to **low** voltage
- Think of an inflated balloon: the air is under pressure, but it only moves once you open a path
- A wire attached to **one** terminal is not a path, so nothing happens

Only one terminal connected: no path, no flow.

5 V down to 0 V through the LED: current flows.

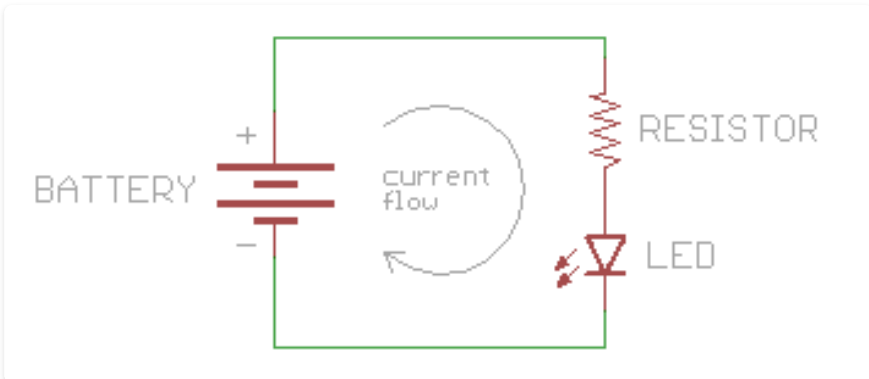
Source: SparkFun, What is a Circuit? (CC BY-SA 4.0)

A Circuit Is a Loop

> A circuit is a **path that starts and stops at the same place**.

Leave the **+** terminal, pass through something useful, arrive back at the **–** terminal.

- Break the loop anywhere and everything stops
- The **pump** analogy: a supply has an outlet side and an inlet side, and the water has to come back
- Direction is a convention: we draw current flowing **+ to –**



Battery, resistor, LED, back to the battery.

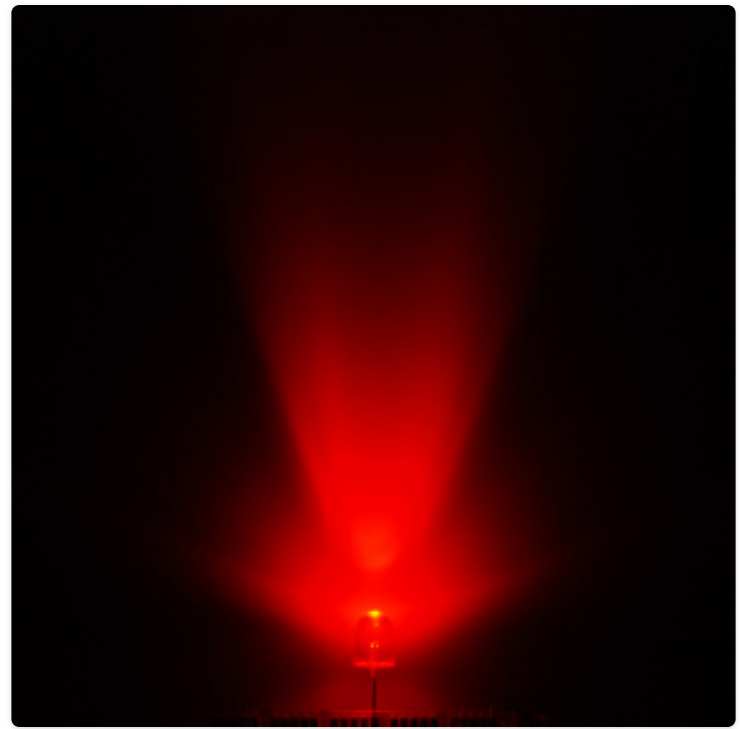
Source: SparkFun, What is a Circuit? (CC BY-SA 4.0)

Loads: What the Current Is For

Anything in the loop that **consumes current and does a job** is a **load**. The name is literal: a load holds the current back, the way a weight holds back someone carrying it.

💡 LED	⚙️ Motor	🔥 Resistor
out as **light**	out as **motion**	out as **heat**

> ⓘ **No load, no limit**
>
> A loop with nothing in it to restrict the current is the one case you never want. That is a **short circuit**, and it is what the next module is about.



A load at work: current in, light out.

Source: SparkFun, Electric Power (CC BY-SA 4.0)

Voltage, Current, Resistance 

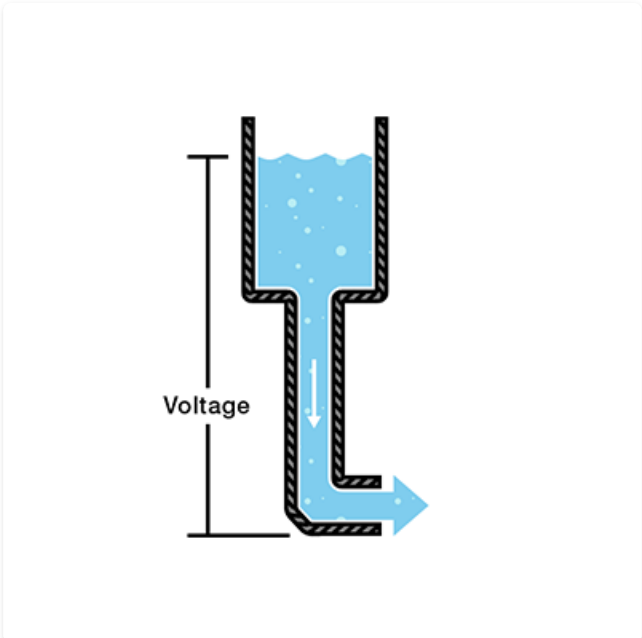
Three quantities, one water tank to remember them by

Voltage Is Pressure

Picture a tank of water with a hose at the bottom. The **height of the water** sets the pressure at the nozzle.

- Water = **charge**
- Pressure = **voltage**
- Flow = **current**

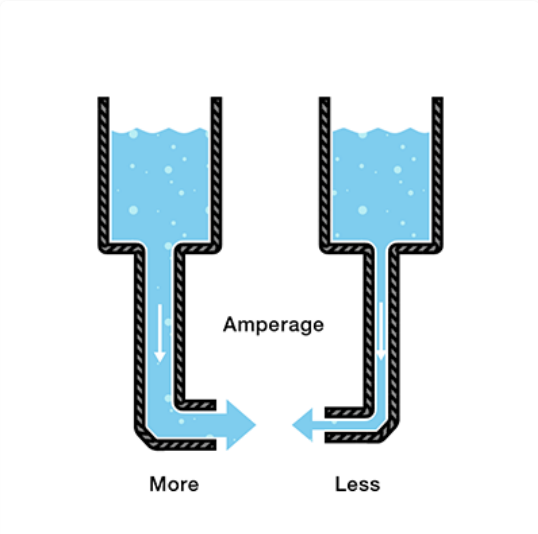
A full tank pushes hard. As it empties, the pressure drops and the stream weakens: exactly why a torch dims as its batteries drain.



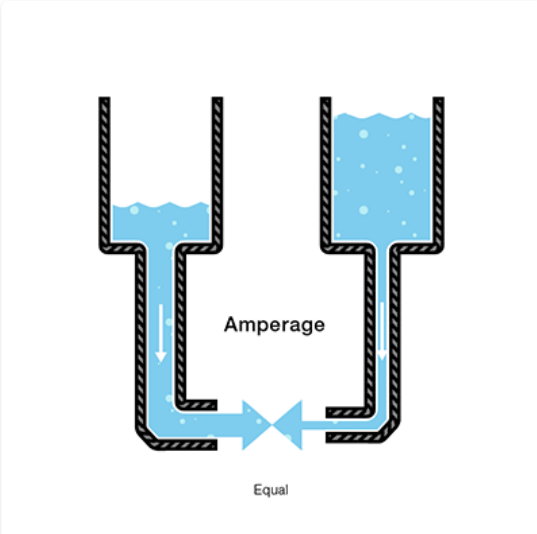
Water height = pressure = voltage.

Source: SparkFun, Ohm's Law (CC BY-SA 4.0)

Current Is Flow, and the Hose Decides



Same pressure, different hose. Equal water levels, but the narrow hose passes less flow. Same voltage, less current.



Same flow, different pressure. To get equal current through the narrow hose, fill its tank higher: more voltage.

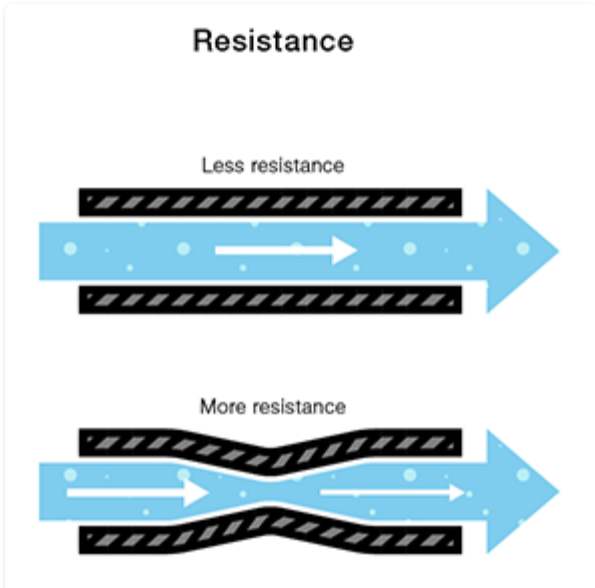
Source: SparkFun, Ohm's Law (CC BY-SA 4.0)

Resistance Is the Pipe

The last piece of the analogy: **hose width = resistance**. A pinched pipe fights the flow.

- **1 ohm** is the resistance where **1 volt** pushes **1 ampere** through
- Written **Ω**, the Greek capital omega
- Copper wire has almost none, a resistor has a chosen amount, air has effectively infinite

Every real component resists a little, which is why long wires and thin traces lose voltage along the way.



Pinch the pipe and you've added resistance.

Source: SparkFun, Ohm's Law (CC BY-SA 4.0)

Ohm’s Law ⚖️

One equation, and the only one you have to remember

V = I × R

In 1827 Georg Ohm measured what everyone had guessed: in a resistive circuit, the three quantities are locked together.

$$V = I \times R$$

- **V** = voltage, in volts
- **I** = current, in amperes (from *intensité de courant*)
- **R** = resistance, in ohms

Raise the voltage and current goes up. Raise the resistance and current goes down.

Georg Simon Ohm, 1789 to 1854

Source: SparkFun, Ohm's Law (CC BY-SA 4.0)

Rearranged: Solve for Whichever One Is Missing

You almost always know two of the three. **Cover the one you want** and the triangle spells out the form that gives it to you.

$$V = I \times R$$

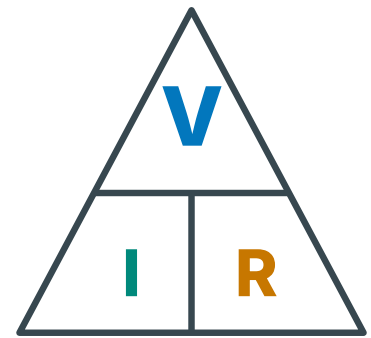
want the **voltage**

$$I = \frac{V}{R}$$

want the **current**

$$R = \frac{V}{I}$$

want the **resistor**



Hover a letter to cover it.

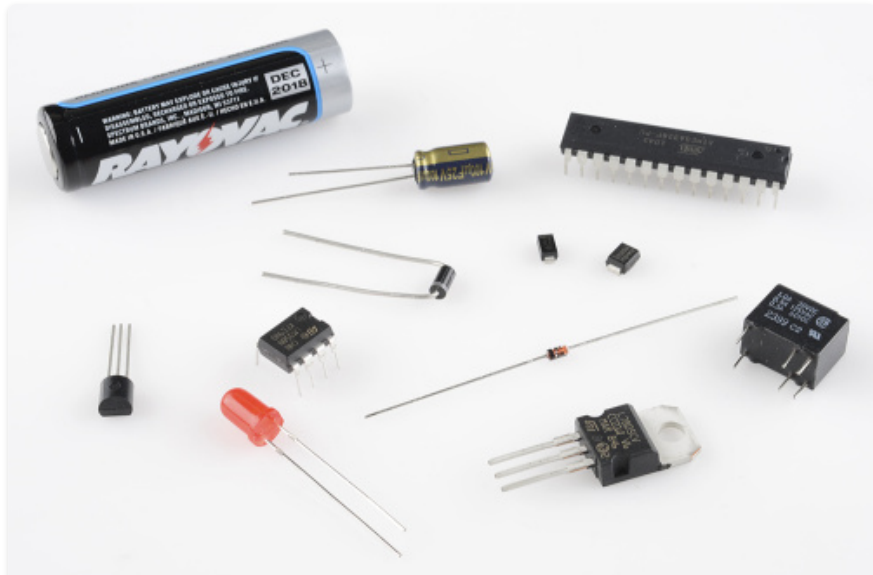
The third form is the useful one

Choosing a part is nearly always `R = V / I`: you have a supply, you know the current a component can survive, and you are solving for the resistor to put in the way.

Polarized vs Non-Polarized

- A **non-polarized** part works no matter which way round it’s connected: a resistor, a plain wire
- A **polarized** part has a **+** pin and a **-** pin with different jobs. Swap them and it won’t work as intended, and it may be damaged
- Matters most on a breadboard or PCB, where it’s easy to place a part backwards

Before wiring anything unfamiliar in, check: does this part care which way round it goes?



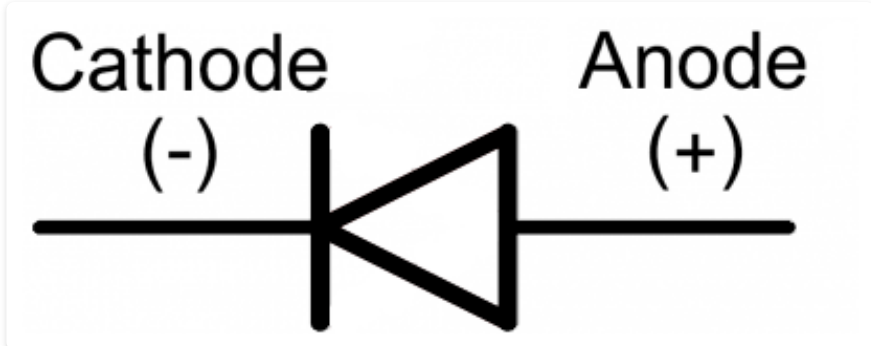
Batteries, ICs, capacitors, diodes: all polarized.

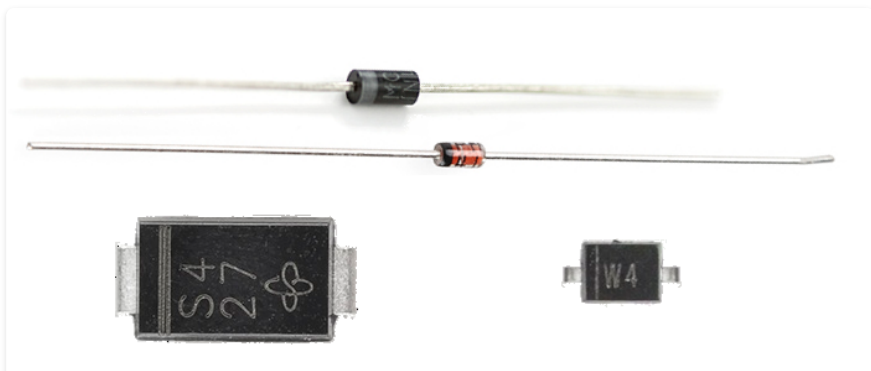
Source: SparkFun, Polarity (CC BY-SA 4.0)

Diodes: Anode In, Cathode Out

- A diode lets current flow **one way only**
- The **anode** (+) is where current enters; the **cathode** (-) is where it leaves
- The symbol's bar marks the cathode, current flows in the direction the arrow points
- On the real part, a painted **line or band** near one lead marks the same cathode

LEDs are polarized diodes too, with their own markers: see the next module.



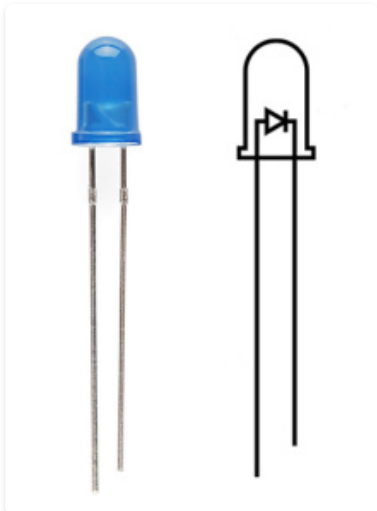


Source: SparkFun, Polarity (CC BY-SA 4.0)

What an LED Is

- **LED** = Light Emitting Diode: a diode that gives off light when current crosses it the right way
- More efficient and cooler-running than a traditional bulb
- Like any diode, it's polarized: the longer leg is the **anode** (+), the shorter leg or flat edge on the casing is the **cathode** (-)

Wired backwards, an LED simply stays dark. It won't break, and no current flows.



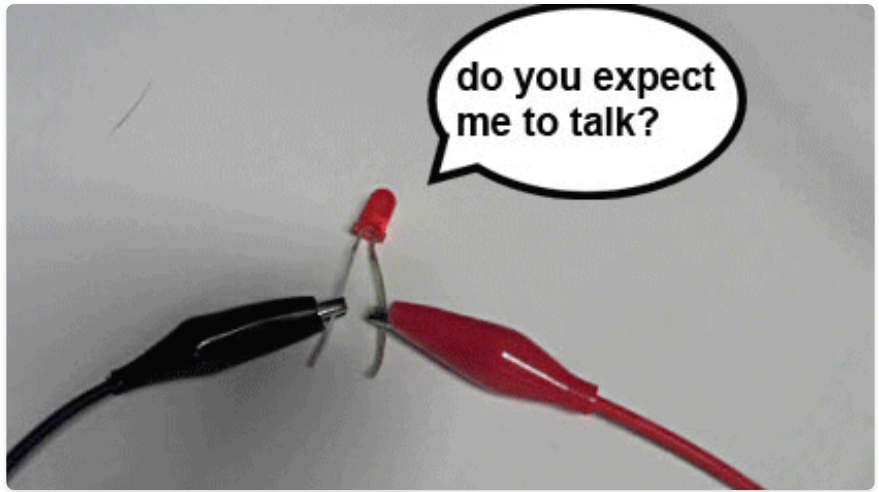
Long leg in, short leg out.

Source: SparkFun, Light Emitting Diodes (LEDs) (CC BY-SA 4.0)

Two Rules Before Wiring One Up

- **More current means more light**: brightness tracks how much current the LED draws
- **There is such a thing as too much power**: wire an LED straight across a battery with nothing else in the loop, and it draws far more current than it can survive

> ⓘ **The fix**
>
> A current-limiting resistor in series holds the current to a safe level. See the next module for how to size one.



No resistor, no mercy.

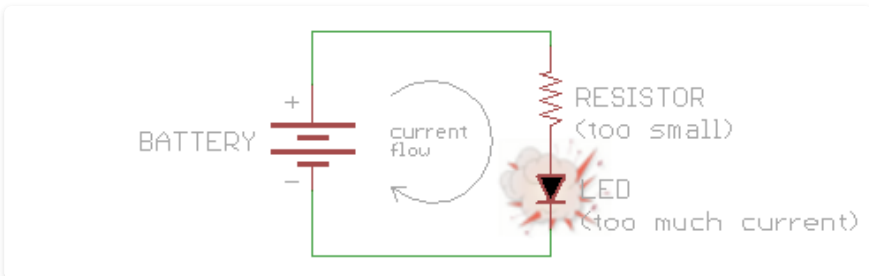
Source: SparkFun, Light Emitting Diodes (LEDs) (CC BY-SA 4.0)

The Problem: 9 V Straight into an LED

An LED is fragile. Wire one across a 9 V battery with nothing else in the loop and ask Ohm's Law what happens:

$$I = \frac{V}{R} = \frac{9\,\text{V}}{0\,\Omega} = \infty$$

- Division by zero: the maths says **unlimited current**
- Reality says the LED absorbs whatever the battery can push, once
- The fix is to put a known resistance in the loop **on purpose**



Nothing to hold the current back.

Source: SparkFun, Ohm's Law (CC BY-SA 4.0)

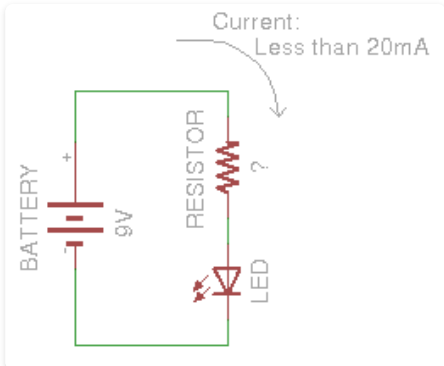
Do the Arithmetic

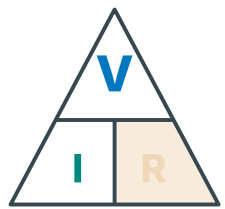
A small red LED is rated for about **18 mA**. Start from what you know and solve for the missing term.

- **V** = 9 V, from the battery
- **I** = 0.018 A, the most the LED should see
- **R** = the unknown

$$R = \frac{V}{I} = \frac{9\,\text{V}}{0.018\,\text{A}} = 500\,\Omega$$

Resistors are sold in standard values, so round **up** to the next one: **560 Ω**. A little more resistance means a little less current, and the LED stays comfortably inside its rating.





R = V / I

Solve for the box marked "?": cover the R.

Source: SparkFun, Ohm's Law (CC BY-SA 4.0)

PROJECT 1 · DIGITAL I/O

Inputs & Outputs: Button → LED

Press a pushbutton → an LED turns on; release it → the LED turns off. The "hello world" of physical computing.

New here? Start with Foundation 2: Introduction & Setup.

Builds on: Foundation 1's current-limiting math and Foundation 2's GPIO safety list, put to work for the first time.

⏱️ ~20 min 📊 Beginner 🔖 Prerequisite: Foundation 1 & 2

⬇ Download the sketch

Project 1 · Inputs & Outputs 🔌

A pushbutton turns an LED on and off: digital I/O end to end.

- Configure pins with `pinMode()` in `setup()`
- Read a button with `digitalRead()`, drive an LED with `digitalWrite()`
- Fix a floating input with a pull-down resistor
- Handle contact bounce with a short debounce delay

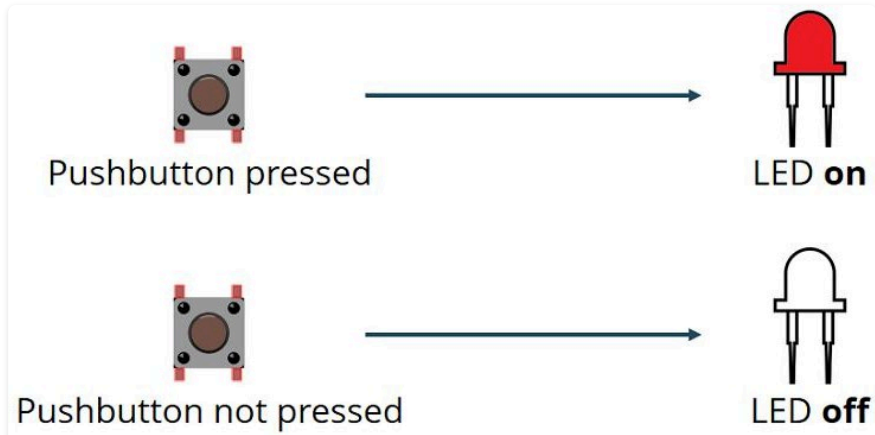
Key pins: button `4` in, LED `5` out

The Idea: Read, Decide, Act

The smallest complete physical-computing program: read the world, decide, act on the world.

- **GPIO 4** = button, read with `digitalRead()`, a digital **input**
- **GPIO 5** = LED, driven with `digitalWrite()`, a digital **output**
- The LED lights while the button reads **HIGH**

Digital I/O means a pin is only ever **HIGH** (3.3 V) or **LOW** (0 V), the opposite of the analog reading in Project 2.

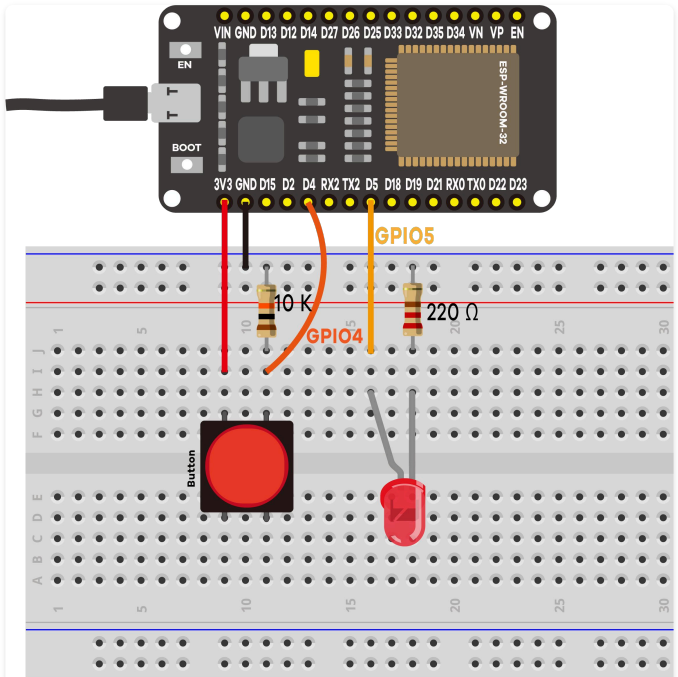


Wiring the Circuit

- **LED (output):** GPIO 5 → **220 Ω** → LED anode (long leg); cathode → **GND**
- **Button (input):** 3.3 V → button → GPIO 4, and from that node → **10 kΩ** → **GND**

> ⚠ **Warning**
>
> A plain `INPUT` pin connects no resistor, so an open pin **floats** and reads noise. The **10 kΩ pull-down** holds GPIO 4 at LOW until the button ties it to 3.3 V.

LEDs are polarized: long leg (anode) toward the resistor, and always use a series resistor.



Code Walkthrough: Read → Compare → Act

```
void loop() {
  buttonState = digitalRead(buttonPin);      // 1. READ the button
  if (buttonState != lastButtonState) {      // 2. did it CHANGE?
    if (buttonState == HIGH)
      digitalWrite(ledPin, HIGH);            // 3. pressed  -> LED on
    else
      digitalWrite(ledPin, LOW);             //    released -> LED off
    lastButtonState = buttonState;           // 4. remember for next time
  }
  delay(50);                                 // 5. debounce
}
```

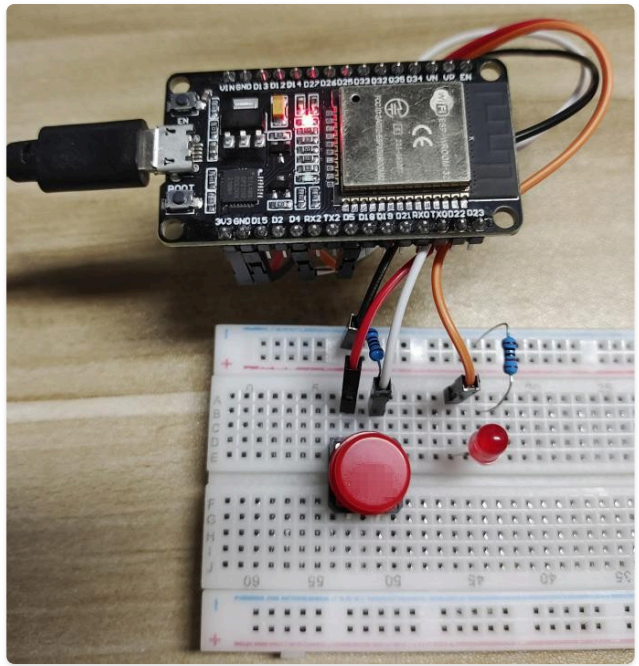
- `pinMode(pin, INPUT / OUTPUT)` in `setup()` sets each pin's **direction** before use.
- Comparing against `lastButtonState` is **edge detection**: it turns a fast polling loop into clean press and release **events**, so the Serial Monitor stays readable.
- A mechanical button's contacts **bounce** for a few ms; the short `delay(50)` rides past it.

It Works

Press the button and the LED lights, with one `PRESSED` event on the Serial Monitor. Release, and it goes dark.

What you learned:

- Every GPIO is an input or an output; `pinMode()` decides which
- `digitalRead()` and `digitalWrite()` are **digital** I/O
- A floating input needs a **pull-down** (or `INPUT_PULLUP` in software)
- Buttons bounce; a short delay is the simplest cure



REFERENCE · GPIO PINOUT

ESP32 DEVKIT V1: GPIO Reference

A lookup table for the 30-pin ESP-WROOM-32 board: pick a pin without breaking booting or Wi-Fi.

> **Rule of thumb:** for a plain input or output, reach for a ✅ **safe** pin first: **GPIO 4, 13, 16, 17, 18, 19, 21, 22, 23, 25, 26, 27, 32, 33**.

Reference · GPIO Pinout 📌

The full ESP32 DEVKIT V1 pin table: safe pins, strapping pins, input-only pins, plus ADC, touch, DAC, and the default bus pins.

- Which GPIOs are safe to use freely
- Which pins must be left alone at boot (strapping pins)
- Which pins are input-only, and which carry ADC/DAC/touch
- Default I²C and SPI bus pins

Open the guide ↗

Thank you!

For spending the day building the Internet of Things with us.

Questions? Ask now, or write to me any time.

Slides, labs, and resources

KGSP Summer Enrichment Program 2026
KAUST Academy, KAUST

Salah Abdeljabar
salah.abdeljabar@kaust.edu.sa