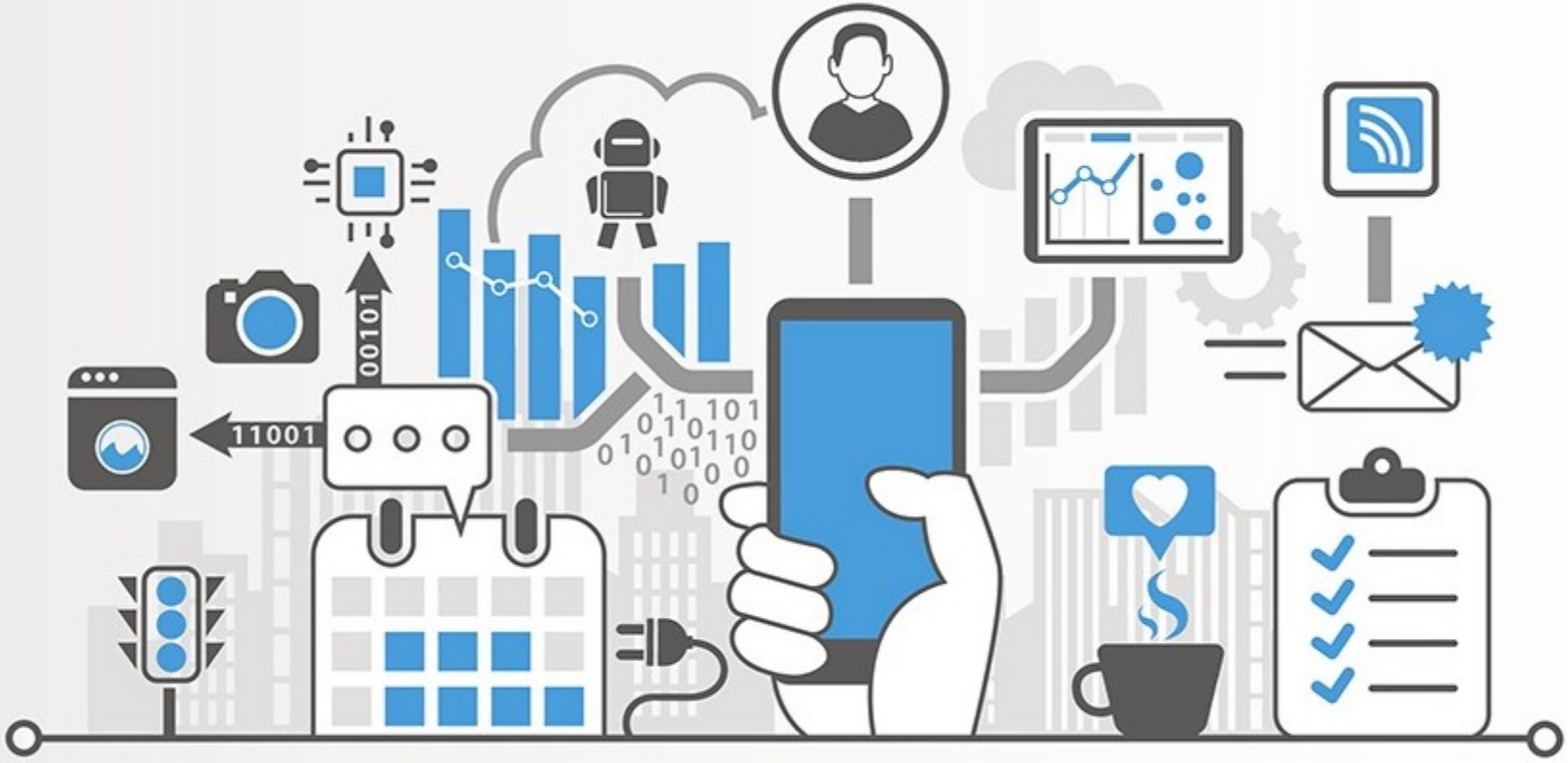
# Tecnologie per IoT

Daniele Jahier Pagliari

## Lab1: Hardware




# LAB ORGANIZATION

# Lab Organization

- **1st part:**
  - 6 basic (guided) exercises to familiarize with the Arduino environment and with the sensors/actuators available in the kit
  - Dedicated class: (today) 3h

- **2nd part:**
  - 1 complex exercise that uses the entire kit to implement a (simplified) "**smart-home controller**".
  - You'll receive only the specifications and you'll implement the code as you want.
  - In this lab, the controller will be fully local (no Internet). Then, in the SW lab, you'll enhance its functionality by connecting it to the "cloud".
  - Dedicated classes: 1.5h + 3h = 4.5h

# Lab Organization

- **3rd part:**
  - 3 basic (guided) exercises to learn how to connect the Arduino to the Internet, using either a Client/Server or a Publish/Subscribe communication paradigm.
  - Useful as a basis for the SW Lab
  - Dedicated class: 3h (tentative)

- In-between the 2st and 3nd parts you'll be introduced by prof. Patti to the Internet Protocol Stack (TCP/UDP/IP) and communication protocols (HTTP/MQTT), which are needed for Part 3.

# Lab Organization

- No report, just code!
  - Send me the code in a *.zip file
- Deadline:
  - **June 26th (together with SW labs)**


# ARDUINO INTRODUCTION

# What is Arduino?

*"Arduino is an **open-source electronics platform** based on **easy-to-use** hardware and software. **Arduino boards** are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the **Arduino programming language** (based on Wiring), and the **Arduino Software (IDE)**, based on Processing."*

(source: arduino.cc)

# What is Arduino?

- **Inexpensive** - Arduino boards are relatively inexpensive (less than $50)
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well
- **Open source and extensible software and hardware** - The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the C++ programming language on which it's based. Similarly, you can add C++ code directly into your Arduino programs if you want to. Circuit designers can make their own version of a module, extending it and improving it.

# Common Features

- Most Arduino boards are based on **Atmel AVR architecture** microcontrollers
  - **Not the one we'll use!!**

- Arduino boards include **all auxiliary components** needed for the microcontroller to work, such as a crystal oscillator and a set of external peripherals and connectors.

# Common Features

- Most Arduino boards use a USB port to:
  - Supply **power** to the entire board, through a voltage regulator
  - **Program** the microcontroller, tunneling JTAG data.
  - Let the microcontroller **communicate** with a PC, tunneling Serial (RS-232) data

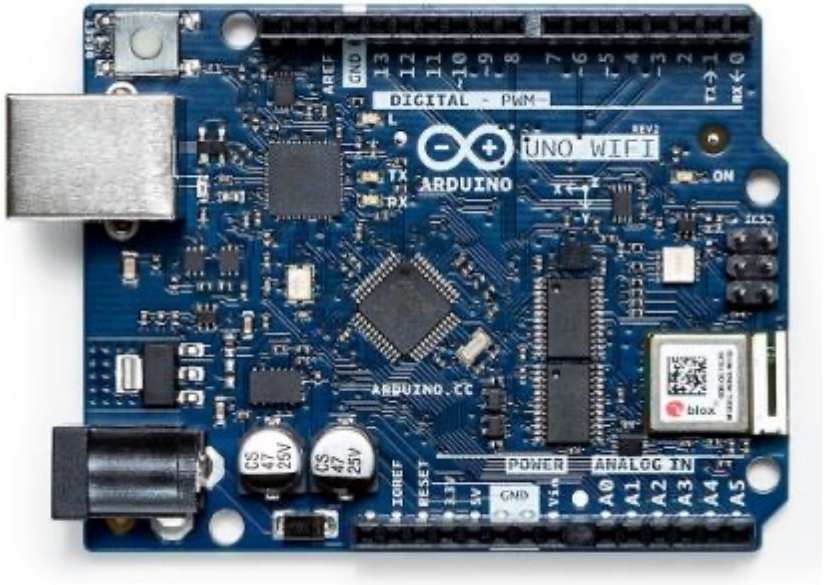
# Common Features

- **GPIO pins** are made available on the board to allow easy connection of external peripherals, for example through a breadboard.

- Most boards use a standard pinout and form factor, which allows the development of **shield boards.**
  - Shields are boards with the same form factor as the main Arduino board, which simplify the interface with complex peripherals .
  - The board we use in this lab is again an exception from this point of view.

# Arduino Uno (WiFi)

- Based on the original Arduino board
- ATmega4809 8-bit microcontroller
- Onboard IMU (Inertial Measurement Unit)
- Wi-Fi Module with integrated TCP/IP stack



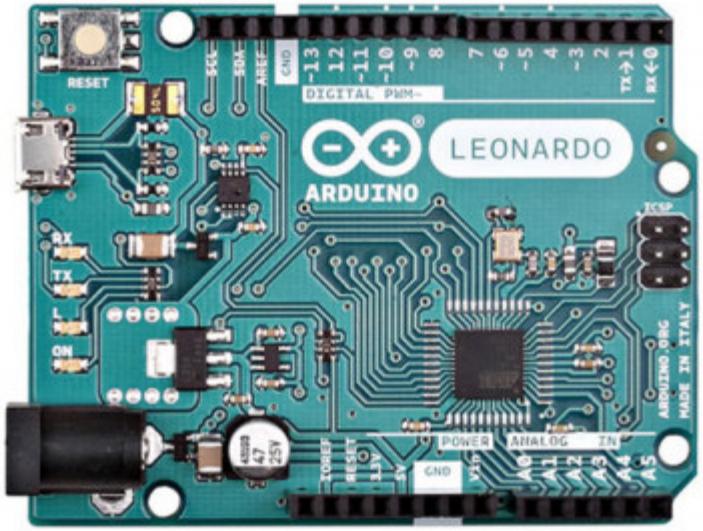
# Arduino Uno (WiFi)

| | |
|---|---|
| Microcontroller | ATmega4809 (datasheet) |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7 - 12V |
| Digital I/O Pins | 14 — 5 Provide PWM Output |
| PWM Digital I/O Pins | 5 |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 20 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 48 KB (ATmega4809) |
| SRAM | 6,144 Bytes (ATmega4809) |
| EEPROM | 256 Bytes (ATmega4809) |
| Clock Speed | 16 MHz |

# Other Arduinos

- Leonardo (ATmega32u4)

- Built-in USB peripheral capabilities (can act as a keyboard or mouse)



# Other Arduinos

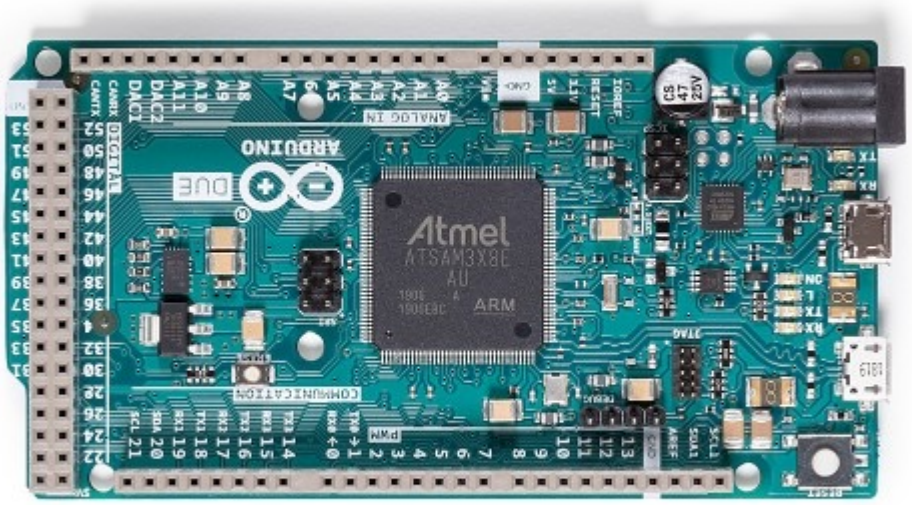
- Nano (ATmega328) and Micro (ATmega32U4)

- Similar to Uno and Leonardo respectively, but in a small form factor


# Other Arduinos

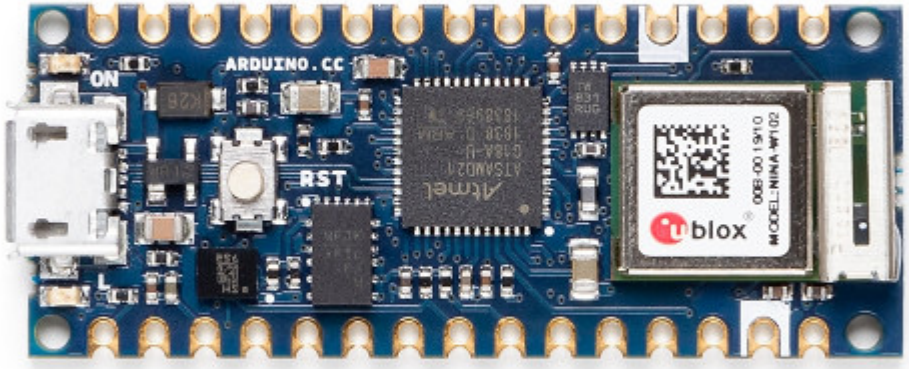
- Modern Arduinos replaced the 8-bit AVR architecture with **32-bit ARM Cortex**

- Arduino Due (Cortex M3)



# Other Arduinos

- Nano 33 IOT (Cortex-M0)
- Supports BLE and WiFi

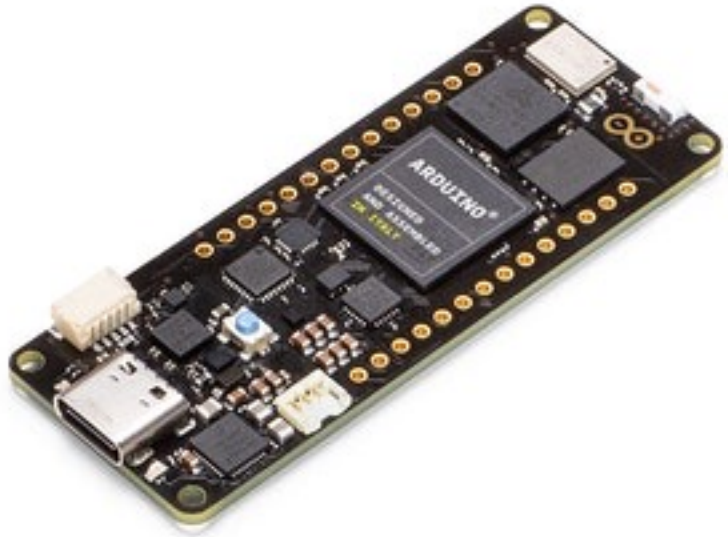


# Other Arduinos

- Etc...
- Many more official Arduino versions
- And even more unofficial «clones»:
  - Sometimes very similar (but cheaper)
  - Sometimes completely different, only sharing the programming environment
    - Example: ESP8266-based boards (Sparkfun Thing etc.)
    - Unofficial list.

# Not just for hobbyists

- **Arduino Pro**: powerful boards for industrial applications
  - Machines sensing and control
  - Edge intelligence
  - Etc.

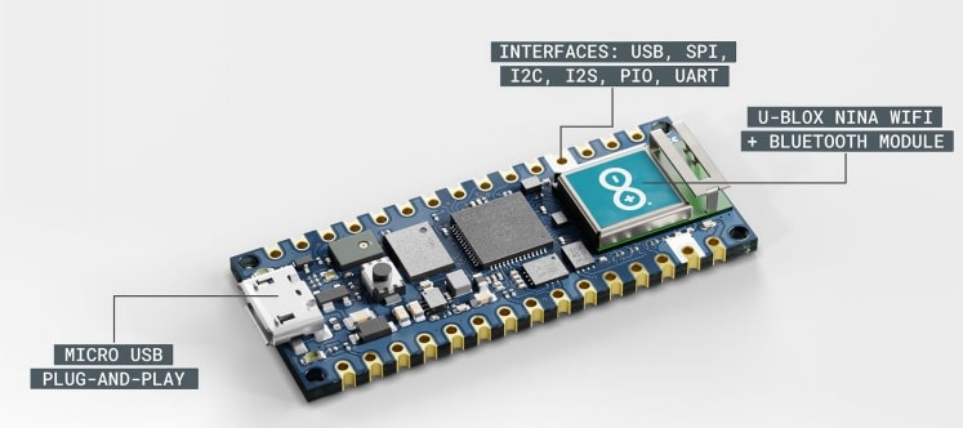



# ARDUINO NANO RP2040 CONNECT

# Nano RP2040 Connect

- A “marriage” between Arduino and another great Open HW project (Raspberry Pi)
  - Brings the Raspberry Pi RP2040 to the Arduino Nano form factor

- 32-bit Arm Cortex-M0 MCU
- Bluetooth and WiFi Module (U-blox Nina W102)
- Built-in sensors and actuators:
  - Accelerometer, gyroscope, RGB LED, temperature sensor, and microphone.



# RP2040 Specs

| | | |
|---|---|---|
| **Pins** | **Built-in LED Pin** | 13 |
| | **Digital I/O Pins** | 20 |
| | **Analog input pins** | 8 |
| | **PWM pins** | 20 (except A6, A7) |
| | **External interrupts** | 20 (except A6, A7) |
| **Connectivity** | **Wi-Fi** | Nina W102 uBlox module |
| | **Bluetooth®** | Nina W102 uBlox module |
| | **Secure element** | ATECC608A-MAHDA-T Crypto IC |
| **Sensors** | **IMU** | LSM6DSOXTR (6-axis) |
| | **Microphone** | MP34DT05 |
| **Communication** | **UART** | RX/TX |
| | **I2C** | A4 (SDA), A5 (SCL) |
| | **SPI** | D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS). |

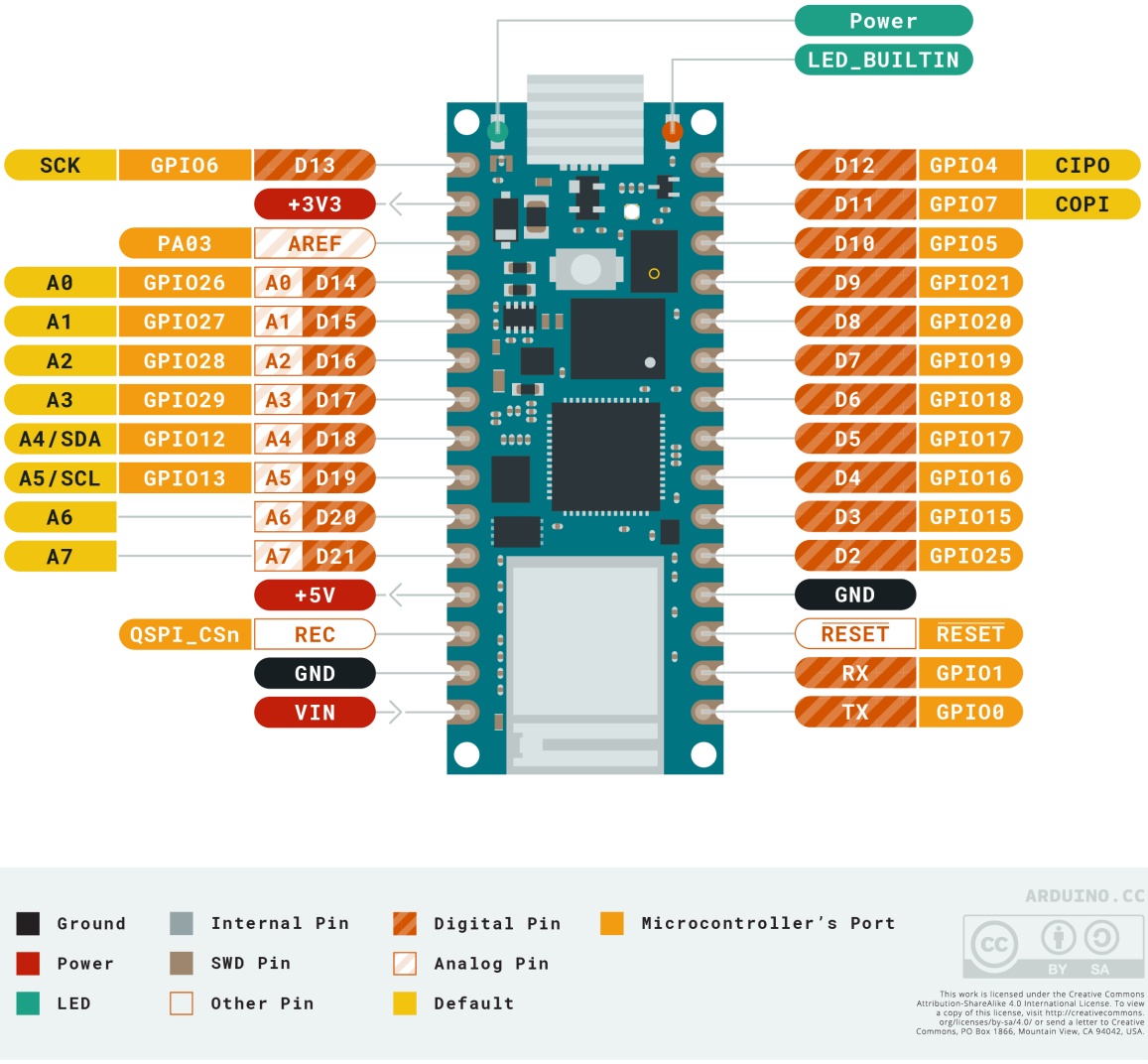
| | | |
|---|---|---|
| **Power** | **I/O Voltage** | 3.3V |
| | **Input voltage (nominal)** | 5-18V |
| | **DC Current per I/O Pin** | 12 mA |
| **Clock speed** | **Processor** | 133 MHz |
| **Memory** | **SRAM** | 264 KB |
| | **AT25SF128A-MHB-T** | 16MB Flash IC |
| | **Nina W102 uBlox module** | 448 KB ROM, 520KB SRAM, 2MB Flash |

Very high-end MCU!
(= easier life for you)

# Pinout

- **Key slide** for the labs

**Warning: A6/A7 can't be used for PWM or interrupts**




# THE ARDUINO IDE

# The Arduino IDE

- The next step is to download and run the **IDE installer** from here
  - We suggest using the desktop IDE for these labs

- A detailed «getting started» guide on the IDE can be found here

# The Arduino IDE


Source: arduino.cc

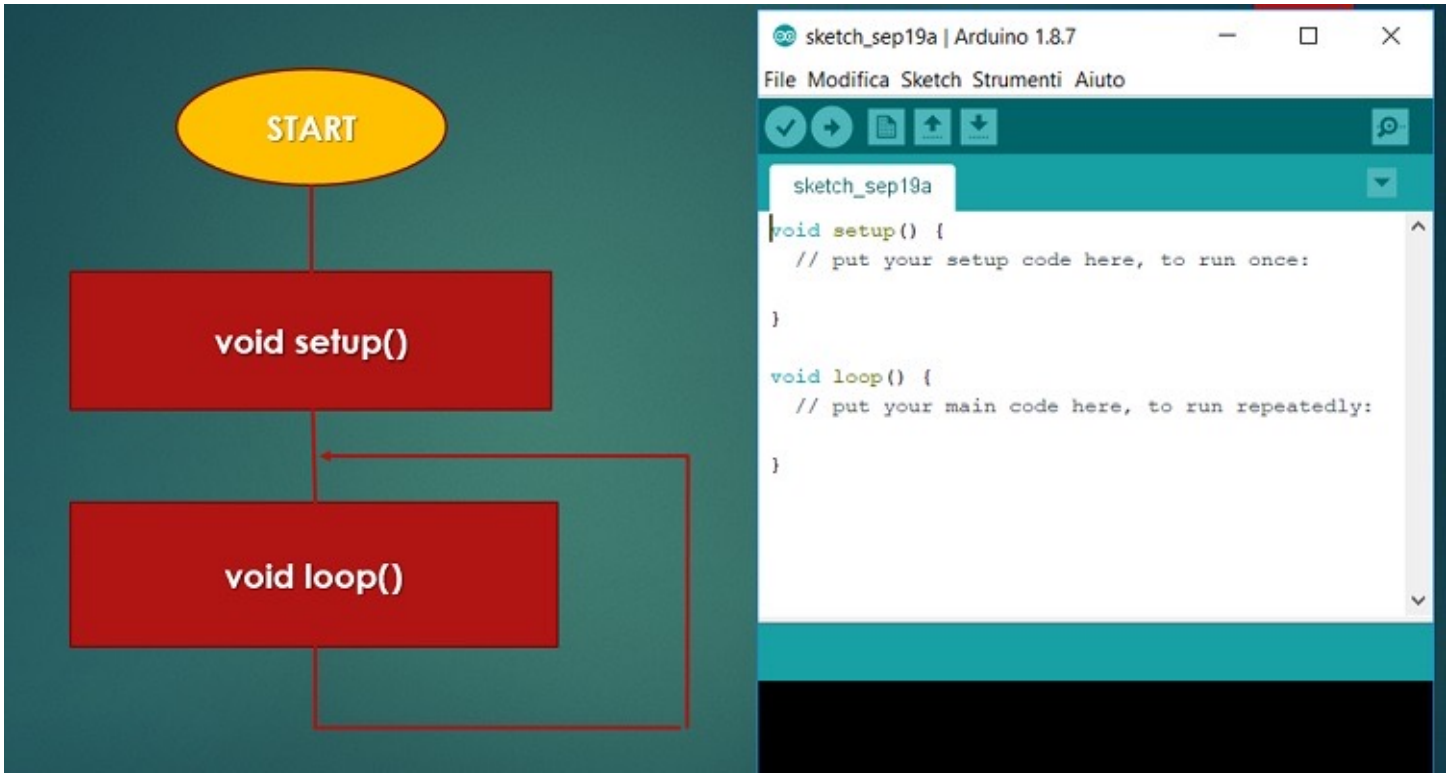
# Arduino Sketches

- Arduino programs are called **sketches** and have `.ino` extension.

- They are based on a programming language called itself **Arduino**, based on Wiring
  - Under the hood, **it's actually C++**
  - With a simplified program structure to speed-up development

# Arduino Sketches

- Two required functions: `setup()` and `loop()`
  - Plus any number of programmer-defined functions



# Arduino Sketches

- `setup()`: initial settings
  - Set GPIO pin as input or output
  - Initialize serial connection
  - Write initial commands to configure I2C peripheral
  - etc.

- `loop()`: repeated main loop of operations
  - Read value from a sensor
  - Drive an actuator
  - Send periodic info to a Serial connection
  - Etc.
  - Each loop() invocation corresponds to one iteration, so **global variables** are needed for data persistence across iterations.

# Arduino Sketches

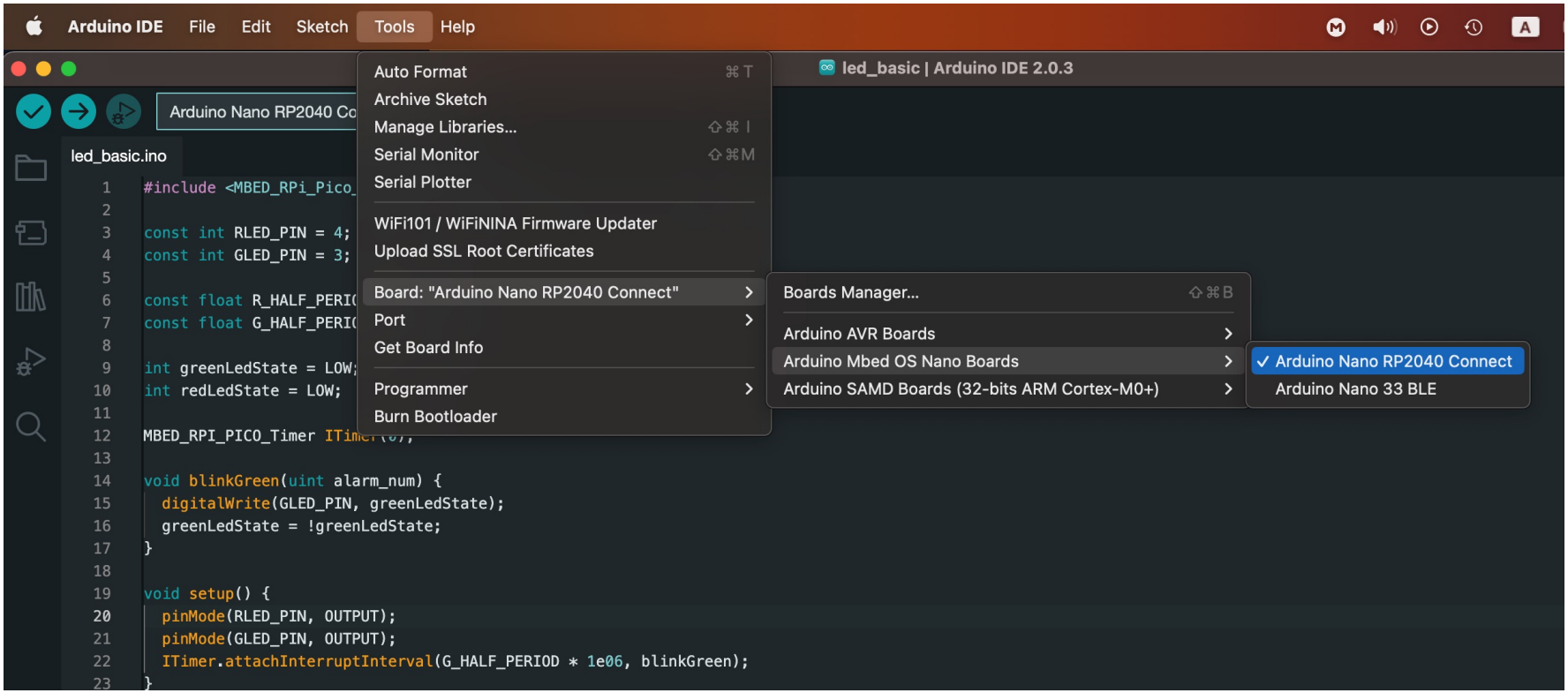
- The basic Arduino library includes functions to perform common operation, such as:
  - Get the time elapsed since the start of execution: `micros()` / `millis()`
  - Wait for some time: `delay()` / `delayMicroseconds()`
  - Set GPIO pins mode: `pinMode()`
  - Read/Write digital or analog values from GPIOs: `digitalRead()` / `digitalWrite()` and `analogRead()` / `analogWrite()`
  - Attach ISRs to digital GPIO inputs: `attachInterrupt()`
  - Perform math operations: `abs()`, `max()`, `min()`, `cos()` …

- We cannot explain every function in detail and you are (soon going to be) engineers, so you'll have to check the documentation yourselves here.

# FIRST SKETCH

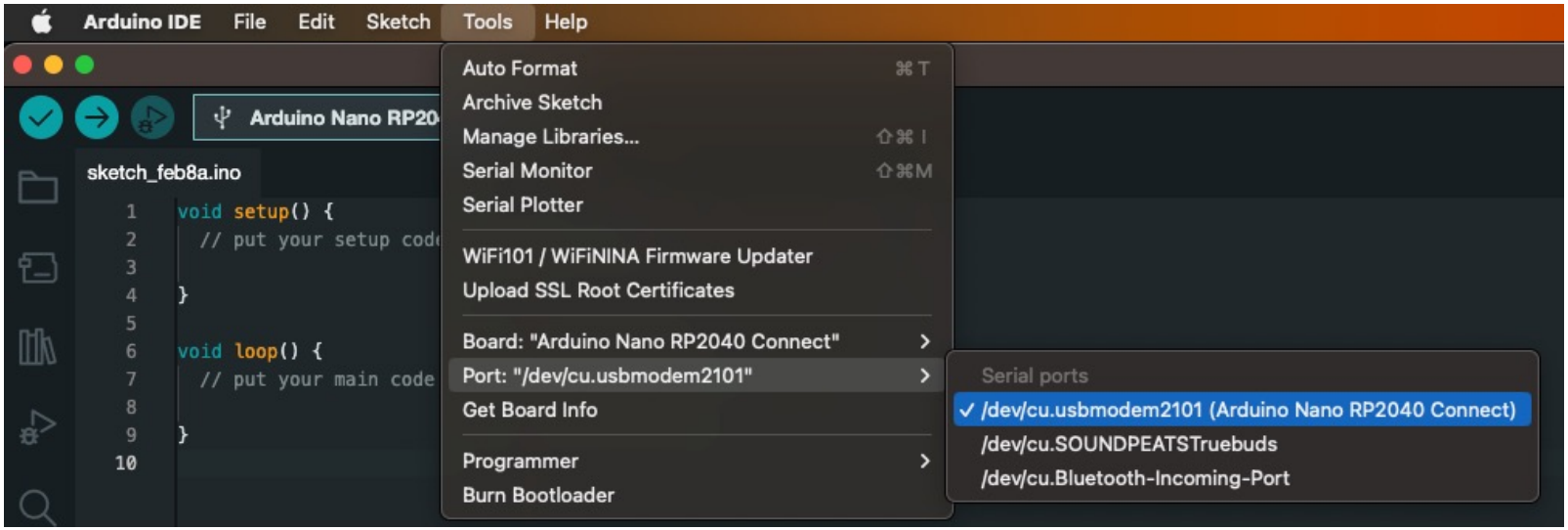
# First Sketch

- Let's test if we can program the RP2040
  1. Connect the board to your PC using the micro-USB cable
  2. In the Arduino IDE,  under "Tools/Board", select the RP2040



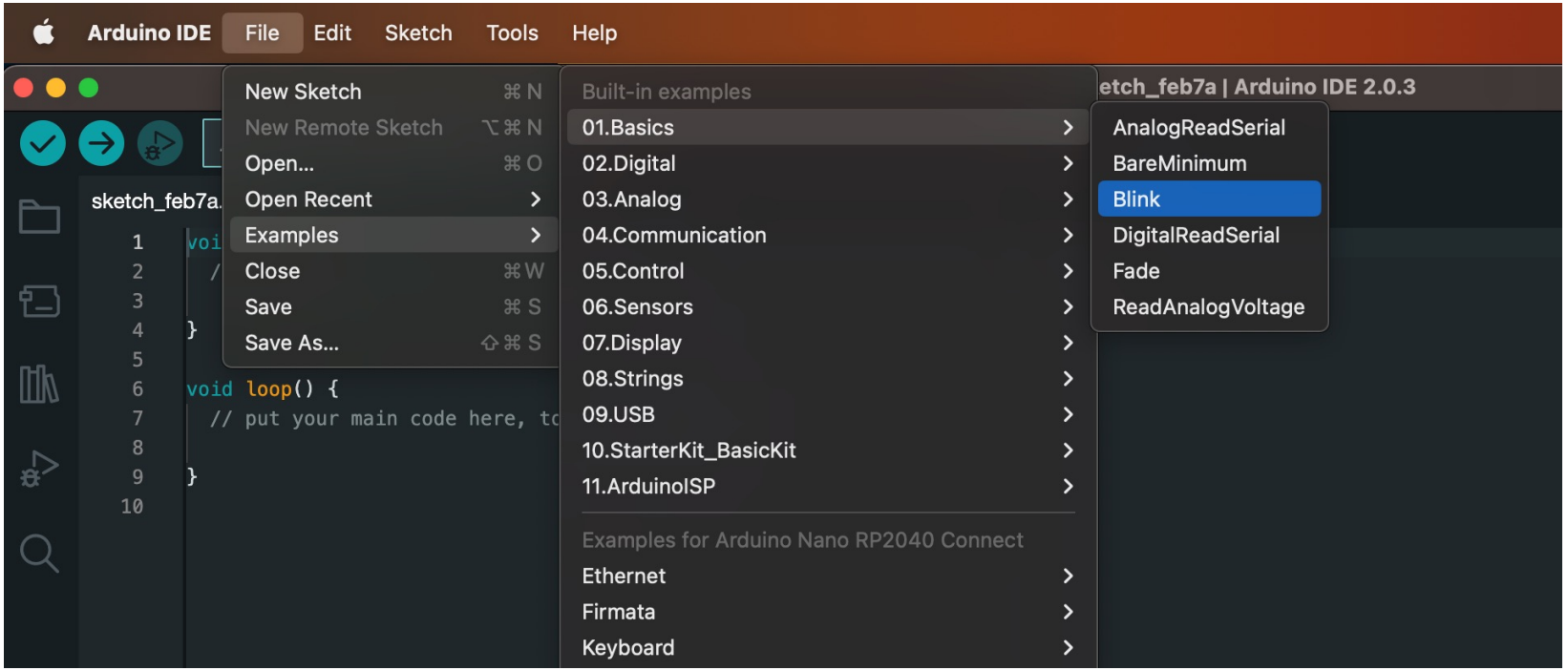
# First Sketch

- Let's test if we can program the RP2040
  3. Under "Tools/Port" select the appropriate Serial port:



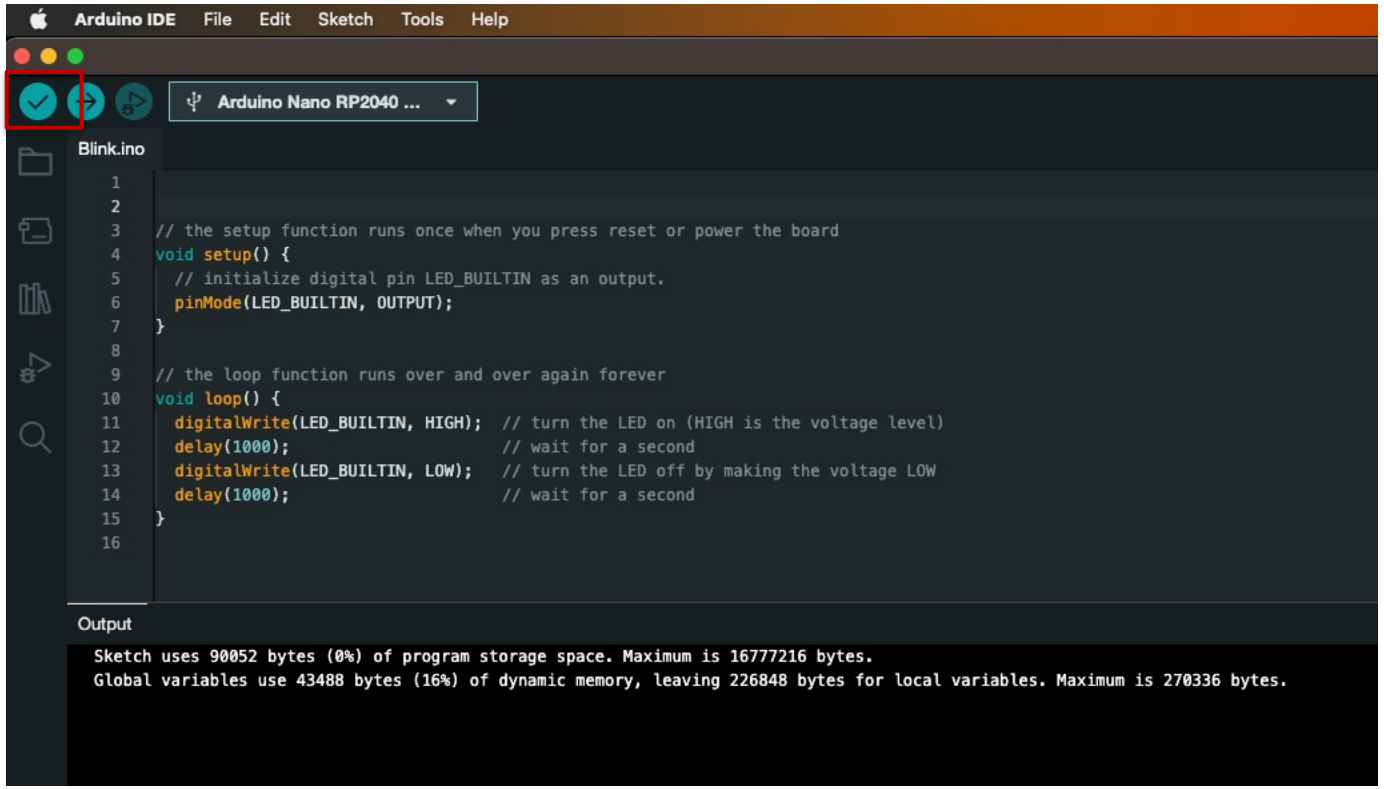
# First Sketch

- Let's test if we can program the RP2040
  4. Under "File/Examples/01. Basics/" Select the Blink example



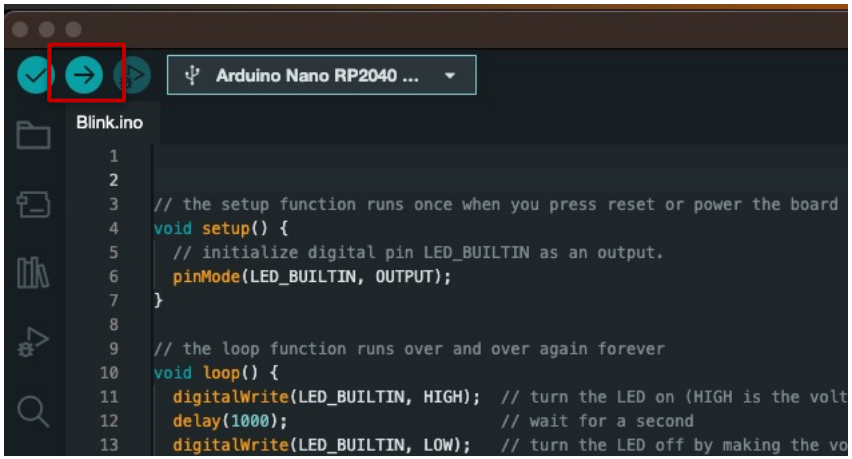
# First Sketch

- Let's test if we can program the RP2040
  5. Verify (i.e. compile) the sketch



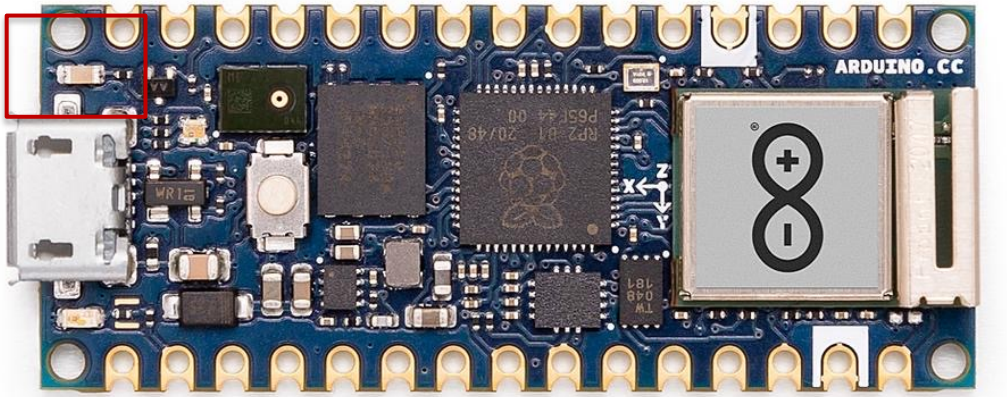

# First Sketch

- Let's test if we can program the RP2040
  6. Upload the sketch to the board




# First Sketch

- Let's test if we can program the RP2040
  6. Check if the sketch is working
     - Internal red LED (left of the microUSB connector) should blink with 2s period.



# Upload Problems

- If you have problems uploading the sketch to the RP2040, do the following:
  - Press the **reset button twice**
  - This brings the board in bootloader mode
  - The red LED should turn on, then flash for a moment, then turn off
  - You can now upload your sketch

- If you entered bootloader mode by mistake, you can exit resetting the board again (single press)

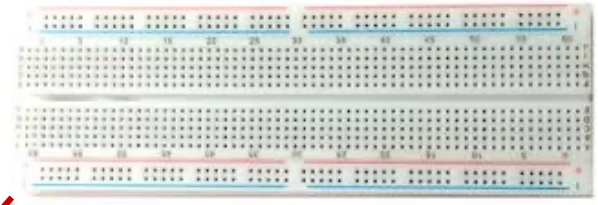

# PART 1: EXERCISE 1

# Exercise 1

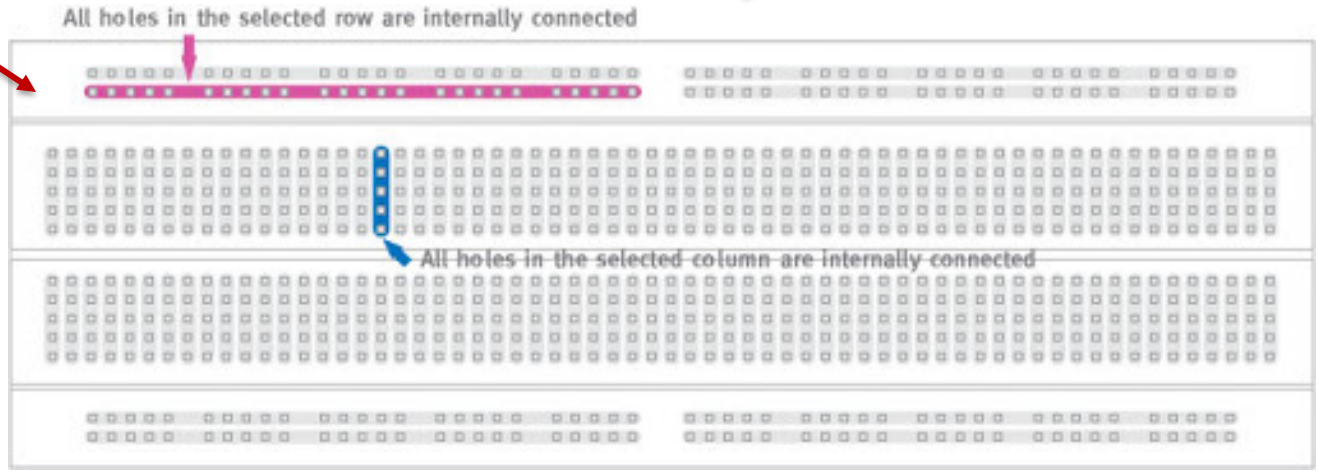
- HW components involved:
  - Breadboard
  - LEDs
  - Resistors
- SW components involved:
  - Digital GPIO outputs
  - Interrupts

# Breadboard

- Build simple circuits without soldering

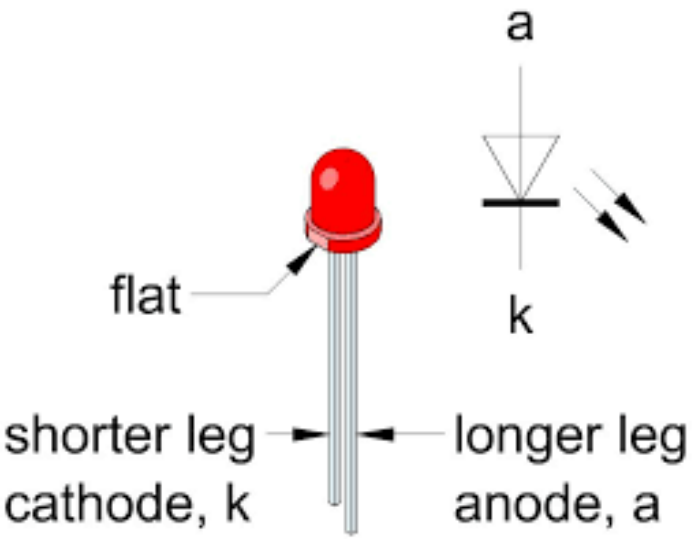


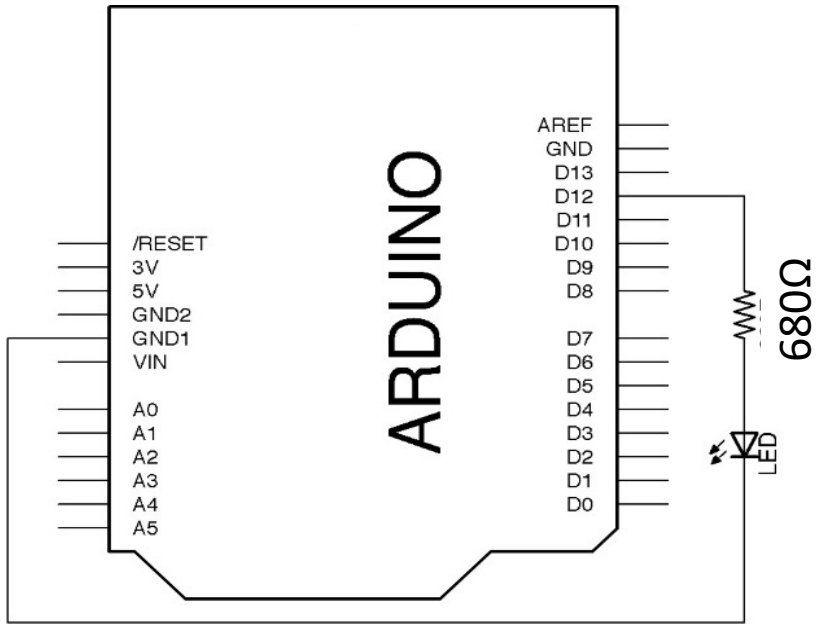
Normally for
VCC/GND



# LED Connection

- A LED is a diode, so we need a series resistor to limit the amount of current flowing through it (i.e. avoid a short circuit between VCC and GND).



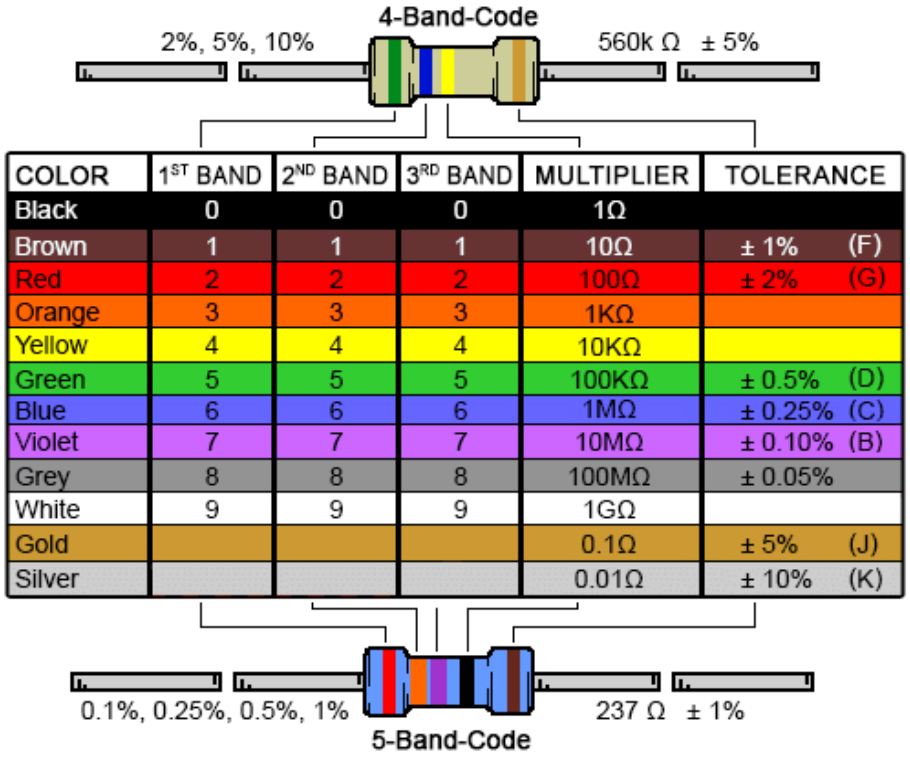




**Circuit**

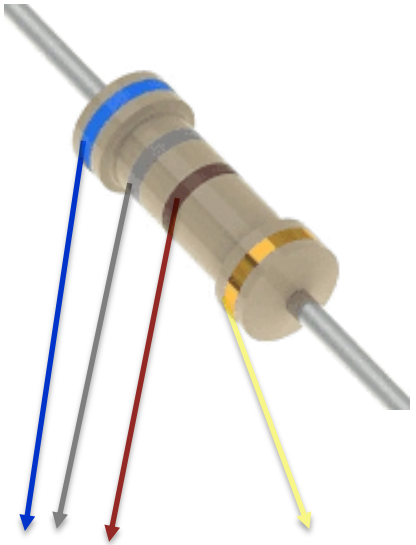
**Caution: longer leg is the anode**

# Resistors Color-coding

- Our Arduino Board has a GPIO HIGH voltage of 3.3V.
- We use a 680 Ohm resistor: (3.3V – 0.6V)/680 Ohm = 3.9 mA < 8 mA (max current per GPIO pin)




| COLOR | 1ST BAND | 2ND BAND | 3RD BAND | MULTIPLIER | TOLERANCE |
|---|---|---|---|---|---|
| Black | 0 | 0 | 0 | 1Ω | |
| Brown | 1 | 1 | 1 | 10Ω | ± 1% (F) |
| Red | 2 | 2 | 2 | 100Ω | ± 2% (G) |
| Orange | 3 | 3 | 3 | 1KΩ | |
| Yellow | 4 | 4 | 4 | 10KΩ | |
| Green | 5 | 5 | 5 | 100KΩ | ± 0.5% (D) |
| Blue | 6 | 6 | 6 | 1MΩ | ± 0.25% (C) |
| Violet | 7 | 7 | 7 | 10MΩ | ± 0.10% (B) |
| Grey | 8 | 8 | 8 | 100MΩ | ± 0.05% |
| White | 9 | 9 | 9 | 1GΩ | |
| Gold | | | | 0.1Ω | ± 5% (J) |
| Silver | | | | 0.01Ω | ± 10% (K) |

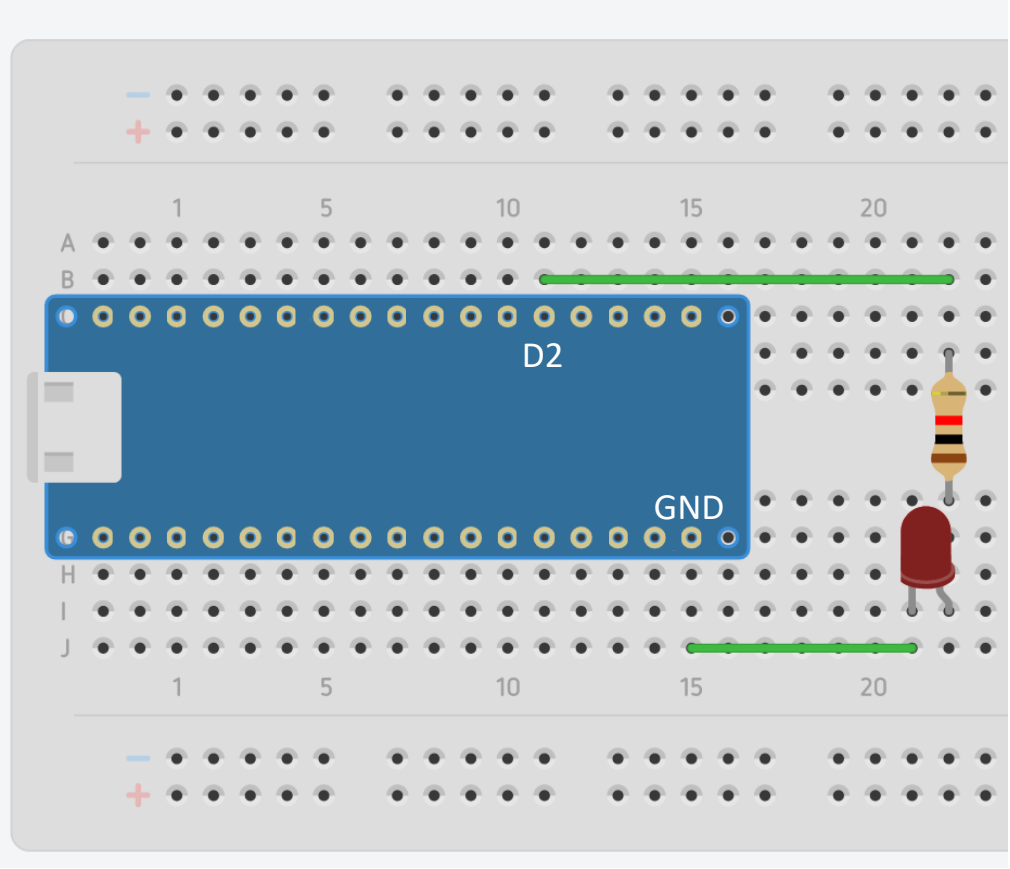


68*10 Ohm +- 5%

Caution: some of you might have a 5-band resistor, but the concept is similar

# LED Connection

- Breadboard connection (example)



# LED Connection

- **For those who don't have the kit**:
  - You can play with simple Arduino circuits using an online emulator on Tinkercad Circuits.
  - You'll be able to use a breadboard to connect components as well as writing and emulating the Arduino code
  - Unfortunately, it only supports other Arduino models, not the RP2040. Also, some of the most complex sensors/actuators that we'll use are not there
  - However, you can try these first examples using the Arduino Uno and the generic components available (LEDs, resistors, etc.)

# Code

```
1   #include <MBED_RPi_Pico_TimerInterrupt.h>
2
3   const int RLED_PIN = 2;
4   const int GLED_PIN = 3;
5
6   const long R_HALF_PERIOD = 1500L;
7   const long G_HALF_PERIOD = 3500L;
8
9   int redLedState = LOW;
10  int greenLedState = LOW;
11
12  MBED_RPI_PICO_Timer ITimer1(1);
```

- Library that allows to attach interrupt service routines to the board Timers.
  - See library documentation here.
- GPIO pins are identified by integer constants
- Global constants for the switching semi-period (in ms)
  - L suffix for ”long int” constants
  - Probably superfluous on 32bit MCU
- Global variables to store the current state of the two LEDS
  - LOW and HIGH MACROS for 0 and 1 respectively
- Create an instance of the timer class using Timer1
  - Timers 0-3 are available
  - Careful, some timers are used for other functions, check library docs.

# Code

- In the setup() function:
  - set the two LED pins as outputs
  - Attach an ISR function to the timer expiration
    - Careful, the timer library requires an interval in **microseconds**!

```
21  void setup() {
22    pinMode(RLED_PIN, OUTPUT);
23    pinMode(GLED_PIN, OUTPUT);
24    ITimer1.setInterval(G_HALF_PERIOD * 1000, blinkGreen);
25  }
```

# Code

- In the ISR: toggle one LED state asynchronously with respect to loop()
  - The ISR must be a void function receiving a uint input corresponding to the timer number.
  - The library docs clarify that you must call `TIMER_ISR_START/TIMER_ISR_END` upon entry/exit.

```
14  void blinkGreen(uint alarm_num) {
15    TIMER_ISR_START(alarm_num);
16    digitalWrite(GLED_PIN, greenLedState);
17    greenLedState = !greenLedState;
18    TIMER_ISR_END(alarm_num);
19  }
```

# Code

- In the loop() function: toggle the other LED state using the delay() function
  - Careful: `delay()` wants a milliseconds value.

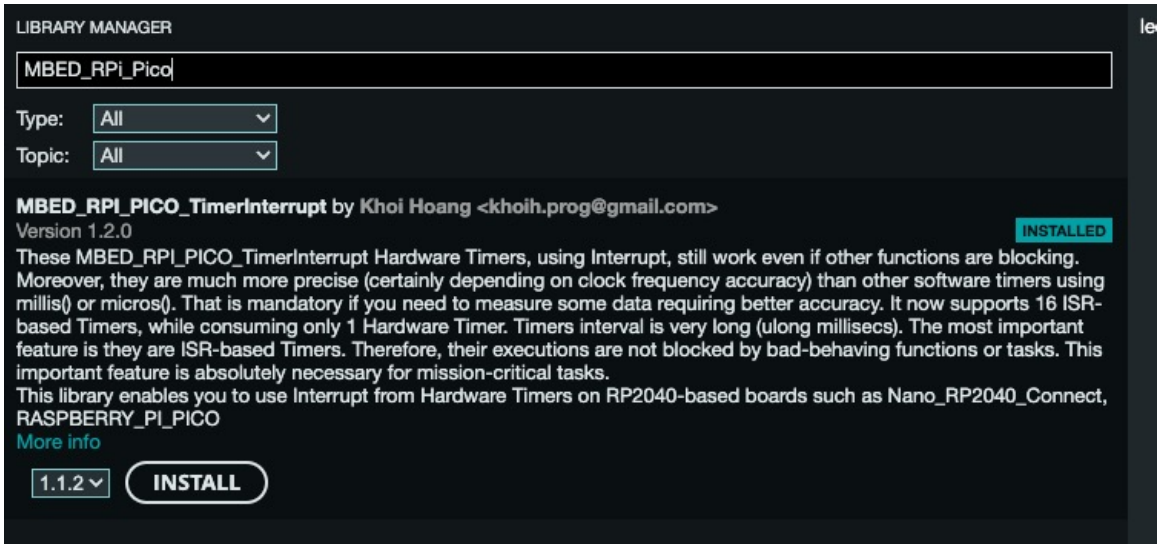
```
27  void loop() {
28    digitalWrite(RLED_PIN, redLedState);
29    redLedState = !redLedState;
30    delay(R_HALF_PERIOD);
31  }
```

# Complete Code

```
#include <MBED_RPi_Pico_TimerInterrupt.h>

const int RLED_PIN = 2;
const int GLED_PIN = 3;

const long R_HALF_PERIOD = 1500L;
const long G_HALF_PERIOD = 3500L;

int redLedState = LOW;
int greenLedState = LOW;

MBED_RPI_PICO_Timer ITimer1(1);

void blinkGreen(uint alarm_num) {
  TIMER_ISR_START(alarm_num);
  digitalWrite(GLED_PIN, greenLedState);
  greenLedState = !greenLedState;
  TIMER_ISR_END(alarm_num);
}

void setup() {
  pinMode(RLED_PIN, OUTPUT);
  pinMode(GLED_PIN, OUTPUT);
  ITimer1.setInterval(G_HALF_PERIOD * 1000, blinkGreen);
}

void loop() {
  digitalWrite(RLED_PIN, redLedState);
  redLedState = !redLedState;
  delay(R_HALF_PERIOD);
}
```

# Installing Libraries

- The Timer library used in the previous code is probably not installed by default.
- Install procedure:
  - Go to Tools/Manage Libraries…
  - Filter by name
  - Click Install



# Alternative Implementations

- As for all Software, there are many ways to obtain the same result.

- For this first example, we will see some alternatives, so that you can learn about additional functionalities & libraries available for the board
  - Might be useful for Part 2

# Code: Version 2

- Instead of the Timer Library, you can use the Scheduler library
  - A simple multi-task scheduler for the Arduino
  - The scheduler is **non-preemptive!** Each task must release control to allow the others to run:
    - Implicitly with a wait statement (e.g. `delay()`)
    - Explicitly with the `yield()` function.
  - Library documentation here.
  - Should come pre-installed!

# Code: Version 2

- Include the Scheduler library

```
1   #include <Scheduler.h>
2
```

- Define constants and global variables as before

- In setup(), use the scheduler to start a secondary loop (loop2)

```
12  void setup() {
13    pinMode(RLED_PIN, OUTPUT);
14    pinMode(GLED_PIN, OUTPUT);
15    Scheduler.startLoop(loop2);
16  }
```

# Code: Version 2

- Define the two loops, each performing a **short operation** and then releasing control through a delay() call.

```
18  void loop() {
19    digitalWrite(RLED_PIN, redLedState);
20    redLedState = !redLedState;
21    delay(R_HALF_PERIOD);
22  }
23
24  void loop2() {
25    digitalWrite(GLED_PIN, greenLedState);
26    greenLedState = !greenLedState;
27    delay(G_HALF_PERIOD);
28  }
```

Default loop, always executed in all sketches

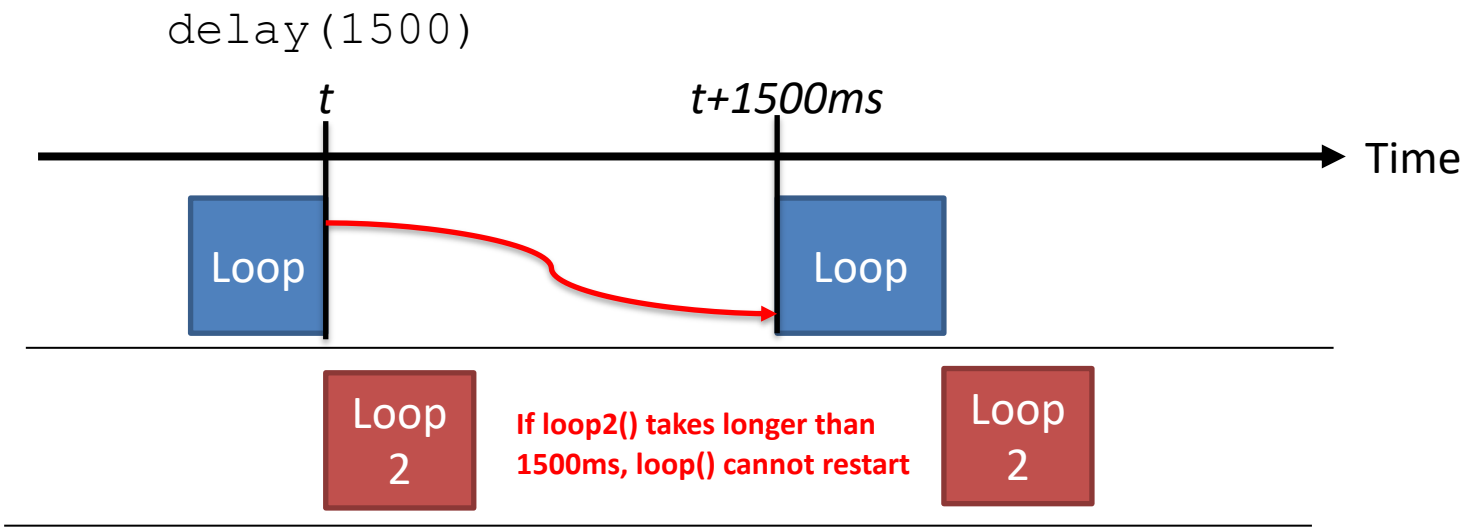
Additional loop, executed because we called Scheduler.startLoop()

# Complete Code: Version 2

```
#include <Scheduler.h>

const int GLED_PIN = 2;
const int RLED_PIN = 3;

const long R_HALF_PERIOD = 1500L;
const long G_HALF_PERIOD = 3500L;

int redLedState = LOW;
int greenLedState = LOW;

void setup() {
  pinMode(RLED_PIN, OUTPUT);
  pinMode(GLED_PIN, OUTPUT);
  Scheduler.startLoop(loop2);
}

void loop() {
  digitalWrite(RLED_PIN, redLedState);
  redLedState = !redLedState;
  delay(R_HALF_PERIOD);
}

void loop2() {
  digitalWrite(GLED_PIN, greenLedState);
  greenLedState = !greenLedState;
  delay(G_HALF_PERIOD);
}
```

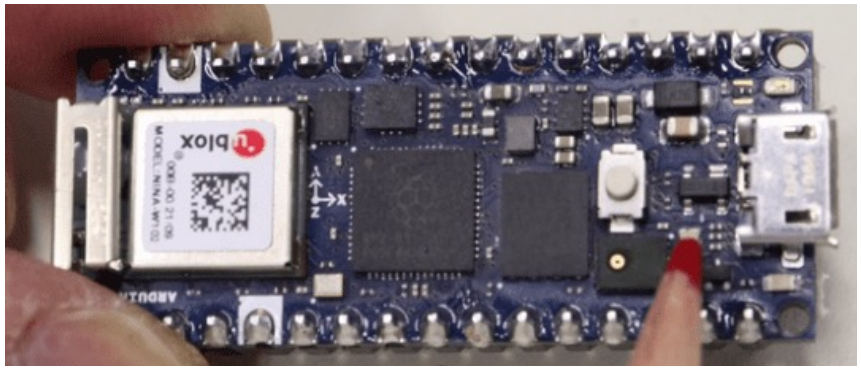
# Scheduler

- **IMPORTANT**: the Arduino MCU is **single-core**
- The loops are **not** running in parallel, but using **time-sharing!**
- This means that loops should release control often to give the "illusion" of parallelism.
- Differently from the Timer solution (Version 1), no "interruption"



# EXTRA - Code: Version 3

- The RP2040 also has an internal RGB LED.
  - This exercise required you to use esternal LEDs so that you could also learn how to physically connect them to the board
  - In future labs, you may also use the internal LED, e.g. as a status indicator, or for debugging purposes.



# EXTRA - Code: Version 3

- There are some important differences between a normal GPIO (external LED) and the internal RGB Led in terms of programming

- First, the RGB led is connected to pins 16 and 18 on the WiFi Ublox NINA Module. So, in order to use the RGB LED, we need to go through the WiFi module.

- In practice, this means installing (if needed) and including the WiFiNINA library

```
3    #include <WiFiNINA.h>
```

# EXTRA - Code: Version 3

- The three LED colors have a common anode pin, and are controlled from the cathode (”active-low” logic)
- The three cathode pins are assigned to constants `LEDR, LEDG, LEDB` in your sketch.

- If you want to rename the pins, you should use a define statement (**not a const variable**, different w.r.t. GPIOs)

```
4    #define RLED_PIN   LEDR
```

# EXTRA - Code: Version 3

- For internal reasons, the digitalWrite() function for NINA pins is also slightly different w.r.t. the one for GPIOs.
- Rather than an **int**, the written value should be a variable of type PinStatus, which is an enum type defined as follows:

```
typedef enum {
  LOW     = 0,
  HIGH    = 1,
  CHANGE  = 2,
  FALLING = 3,
  RISING  = 4,
} PinStatus;
```

Luckily, LOW and HIGH have the same values as for GPIOs

# EXTRA - Code: Version 3

- In practice, you can still write 0/1 to the pin with appropriate type-casting:

```
digitalWrite(RLED_PIN, (PinStatus) redLedState);
```

- Lastly, and very importantly: **the internal RGB LED cannot be accessed from within Interrupt Service Routines**
  - This will make your sketch crash.

- The rest of the code is identical to the case of GPIOs

- **Extra Exercise:** Try to modify Ex. 1 to use the Internal LED.


# PART 1: EXERCISE 2

# Exercise 2

- HW components involved:
  - Same as 1, plus Serial port
- SW component involved:
  - Arduino Serial library

# New Code Elements

```
void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Lab 1.2 Starting");
  //etc...
```

- Create a Serial connection with a specified baud rate
- Wait until the connection is estabilished
  - The program will not start until you open the Serial monitor in the IDE
- Print a newline-terminated string.

```
void serialPrintStatus() {
  if (Serial.available() > 0) {
    int inByte = Serial.read();

    if(inByte == ...) {
      //etc
```

- Check if there are bytes available on the Serial buffer.
- Read a single byte from the buffer

# Serial Monitor

Click here (or Tools/Serial Monitor) to open the monitor for the selected Port


Set the baud-rate to the same value used when creating the Serial() instance and select the termination character here

# New Code Elements

- The full reference for the `Serial` class can be found here:
  - Many more utility functions to read/write various kinds of data
  - Example: attach ISR when new data is available: `serialEvent()`
  - Parse integers or floats from the buffer: `parseInt(), parseFloat()`
  - Flush the buffer: `flush()`
  - Etc.

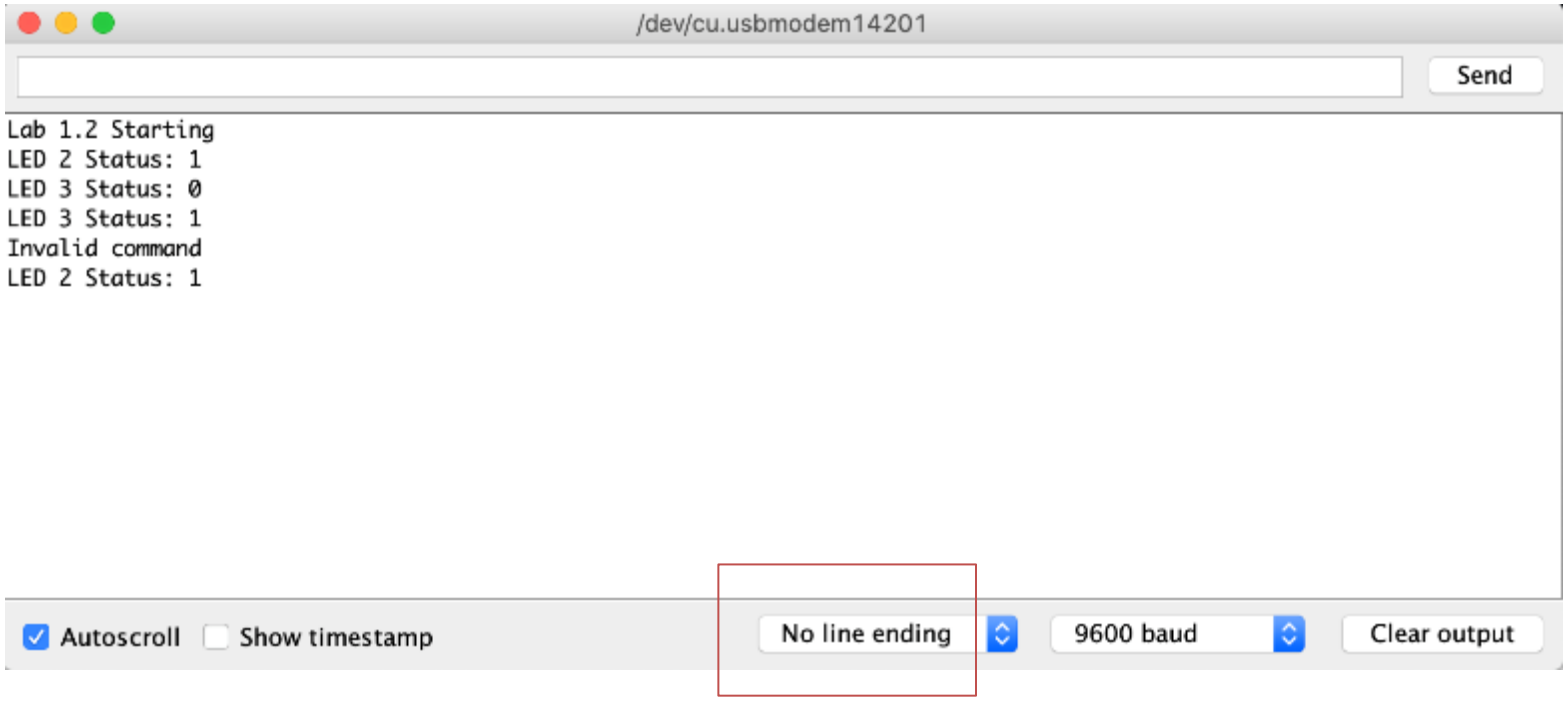
- Arduino can use char-based C-style strings but also supports a more flexible `String` class:
  - Easier concatenation and processing… but **more memory!!**

# New Code Elements

- Important: variables used both by the `loop()` function and by an ISR should be declared as `volatile`
  - This qualifier tells the compiler to avoid register-based optimizations for the variable
  - The variable is always written-back to memory
  - It prevents Read-After-Write dependency violations.

- Additionally, if there are pieces of code that cannot be interrupted, you can use the `noInterrupts()` and `interrupts()` functions to disable and re-enable interrupts as needed

# Exercise 2: Output Example




# PART 1: EXERCISE 3

# Exercise 3

- HW components involved:
  - PIR Sensor
  - Serial port
- SW component involved:
  - GPIO interrupts

# PIR Motion Sensor

- You might have received two different types of PIR sensor in your kit.

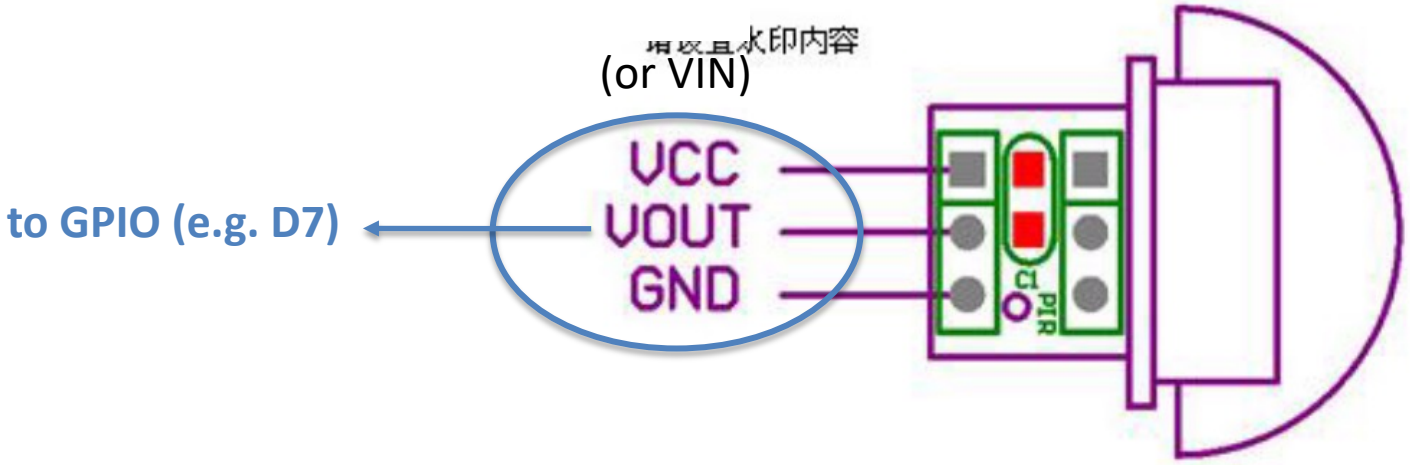
HC-SR312 (Chinese Datasheet)

HC-SR501 (Datasheet)

- Both have similar pinouts and functionality

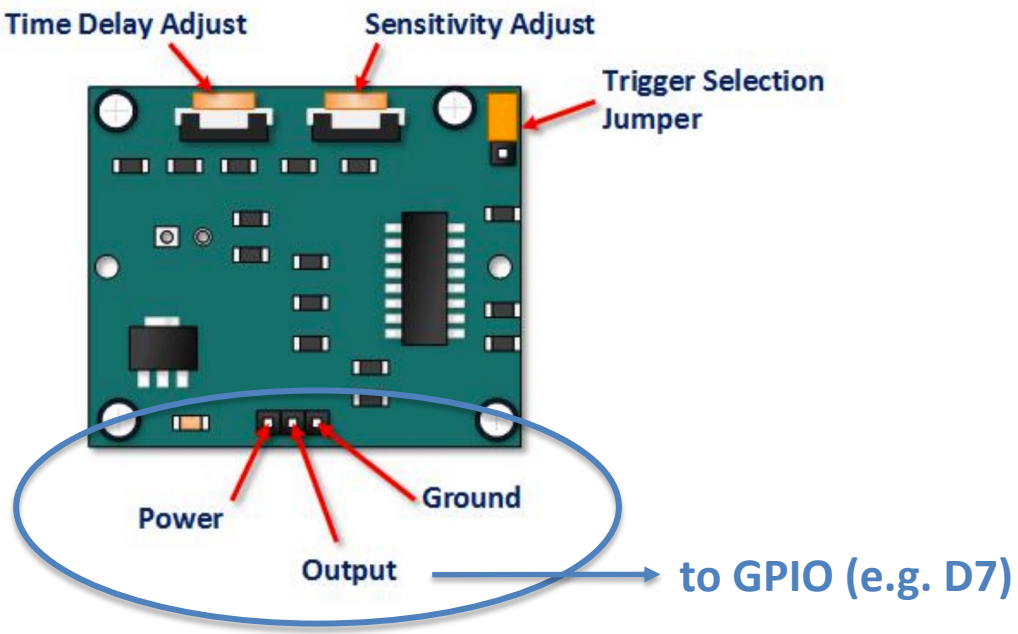
# PIR Motion Sensor: HC-SR312

- Pinout:



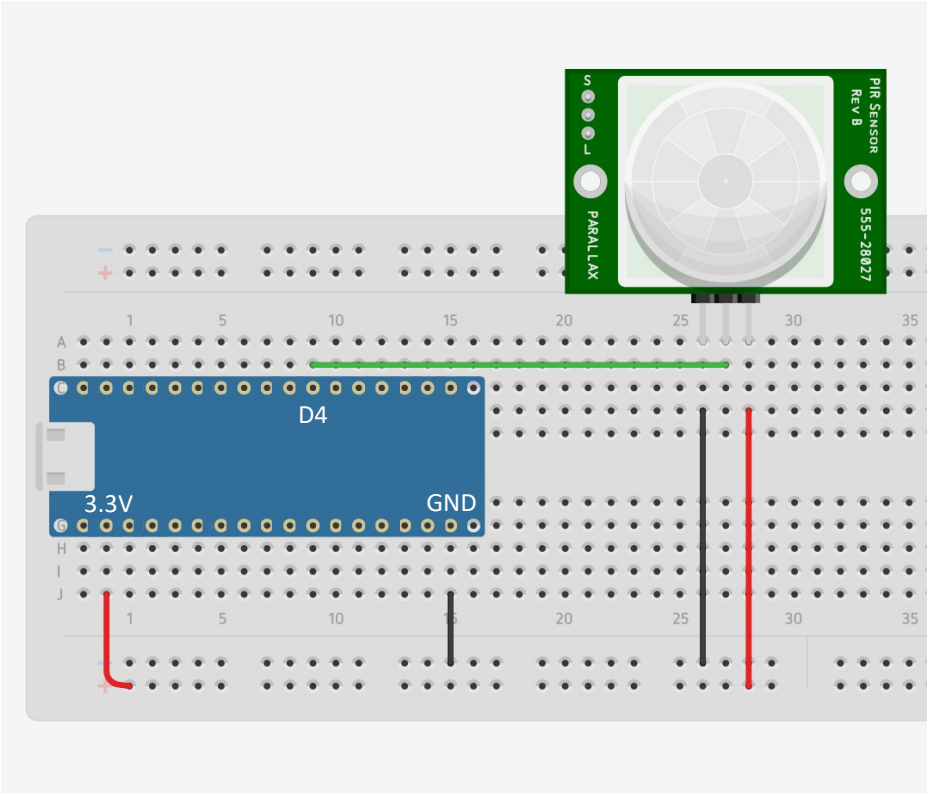
# PIR Motion Sensor: HC-SR501

- Two potentiometers to set the length of the detection and the sensitivity
- One jumper to select between 2 operating modes (single and repeatable trigger)
- Use the datasheet to understand each configuration parameter: link



# PIR Sensor connection

- Suggestion: add new components without disconnecting the previous ones (LEDs). In part 2 you'll use them all together




**WARNING: this diagram is for our sensor. The pinout of the PIR on Tinkercad is different!**

# New Code Elements

- All GPIO pins (except A6 and A7) support interrupts
- `tot_count` is touched by ISR and `loop()` → `volatile`

```
1   const int LED_PIN = 2;
2   const int PIR_PIN = 4;
3
4   volatile int tot_count = 0;
```

- Attach the ISR function to the appropriate GPIO interrupt.

```
14  pinMode(PIR_PIN, INPUT);
15  attachInterrupt(digitalPinToInterrupt(PIR_PIN), checkPresence, CHANGE);
16  //etc...
```

PIN number to Interrupt number

ISR function pointer

RISING, FALLING, CHANGE, etc.

# Exercise 3: Output Example

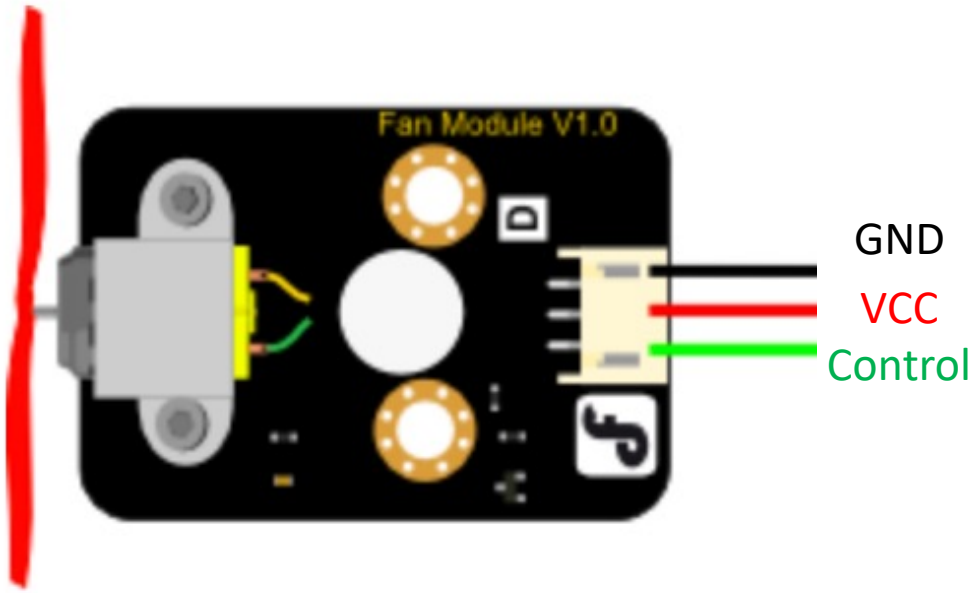

# PART 1: EXERCISE 4

# Exercise 4

- HW components involved:
  - DC Motor (fan)
  - Serial port
- SW component involved:
  - PWM outputs

# PWM Fan: DFR0332

- Datasheet: link




- Breadboard connection very similar to PIR
  - Unfortunately, 3-input DC motor is not available on Tinkercad

# New Code Elements

- All GPIO pins except A6 and A7 support PWM.
- To specify a duty cycle, simply do an `analogWrite()` to the PIN
  - DC values are integers between 0 (0%) and 255 (100%)

```
float current_speed = 0;

void setup() {
  pinMode(FAN_PIN, OUTPUT);
  analogWrite(FAN_PIN, (int) current_speed);
  //etc.
```

# Exercise 4: Output Example

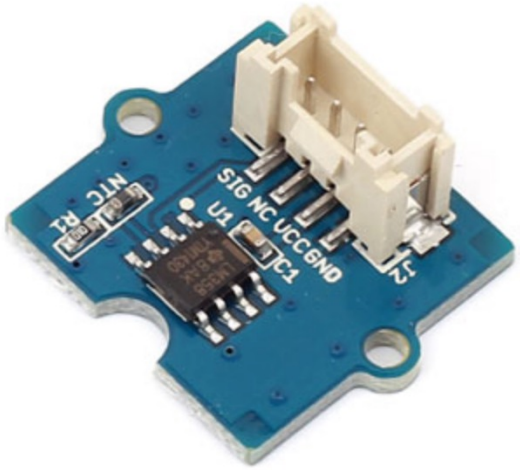

# PART 1: EXERCISE 5

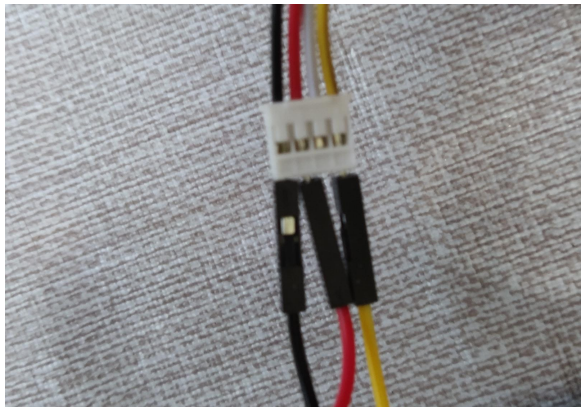
# Exercise 5

- HW components involved:
  - Temperature Sensor
  - Serial port
- SW component involved:
  - Analog inputs

# Grove Temperature Sensor

- Entire sensor “datasheet”: link
- Thermistor datasheet: link



- Breadboard connection again very similar to previous sensors
  - You can plug your jumper cables in the output of the provided cable

# Grove Temperature Sensor

- The sensor produces an output **voltage** which depends (through a variable **resistence**, i.e. the thermistor) on the **temperature**
  - We read the voltage, we need to recover the temperature

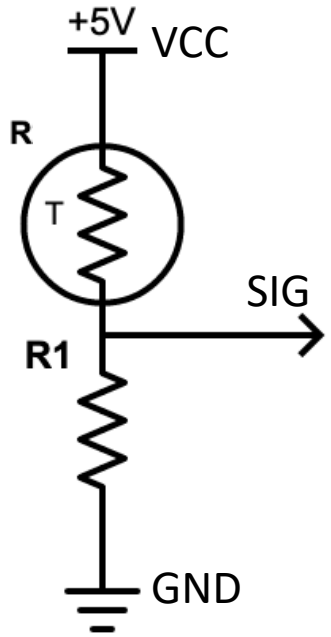
- Two-step process:
  1. From voltage to resistance
  2. From resistance to temperature

- The sensor has two main parameters:
  - **R0 = 100KOhm;** thermistor resistance at nominal T0 = 25$^{o}$ C
  - **B = 4275K;** defines the shape of the (non-linear) relation between resistance and temperature. A definition can be found here.

# Grove Temperature Sensor

- Step 1:
  - R1 = R0 = 100KOhm
  - We can extract R from $V_{sig}$ using a simple resistive divider equation:

$$V_{sig} = \frac{R_1}{R + R_1} * VCC \Rightarrow R = (\frac{VCC}{V_{sig}} - 1) * R_1$$




  - The Arduino ADCs have 10-bit resolution
  - GND = 0, VCC = 1023

# Grove Temperature Sensor

- Step 2:
  - The conversion from R to T can be obtained inverting the definition of B from the datasheeet:

$$B = \frac{\ln\left(\frac{R}{R_0}\right)}{\frac{1}{T} - \frac{1}{T_0}} \quad \Rightarrow \quad T = \frac{1}{\frac{\ln\left(\frac{R}{R_0}\right)}{B} + \left(\frac{1}{T_0}\right)}$$

  - Warning: in this equation both T an T0 are in Kelvin
  - We want to print all temperature values in Celsius

# New Code Elements

- Analog pins are identified with the MACRO Ax where x is the PIN number on the board.

```
const int TEMP_PIN = A1;
const int B = 4275;
const long int R0 = 100000;
```

- Simply call `analogRead()` to read the voltage value from the sensor.

```
void loop() {
  int a = analogRead(TEMP_PIN);
```

# Exercise 5: Output Example


- Touch the thermistor on blow gently on it to see the temperature change


# PART 1: EXERCISE 6

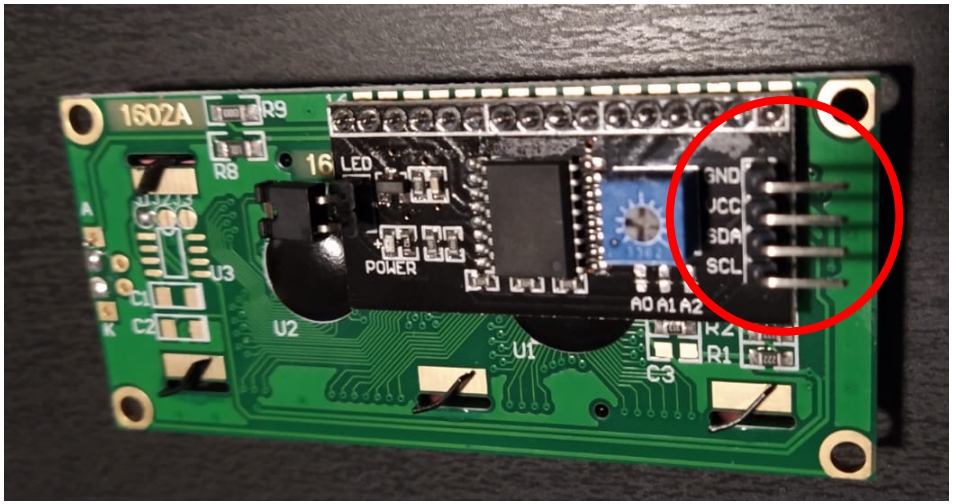
# Exercise 6

- HW components involved:
  - Temperature Sensor
  - LCD Display (with I2C connection)

- SW component involved:
  - LCD library (based on Wire)

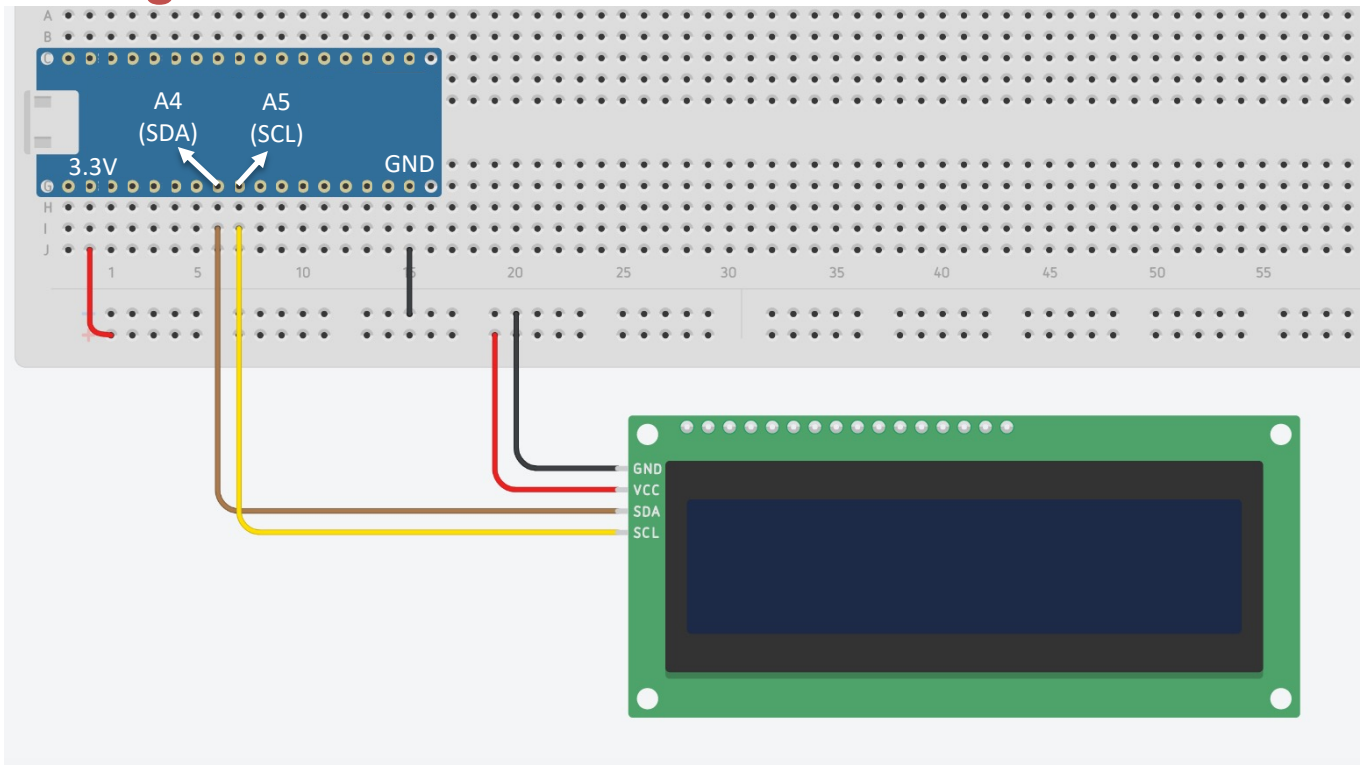
# DFRobot I2C Display

- A “sort” of datasheet can be found here
  - **Careful:** we won’t use the library provided at that link, but a standard library provided by the Arduino IDE’s Library Manager, compatible with this display
  - So don’t copy the code! It won’t work



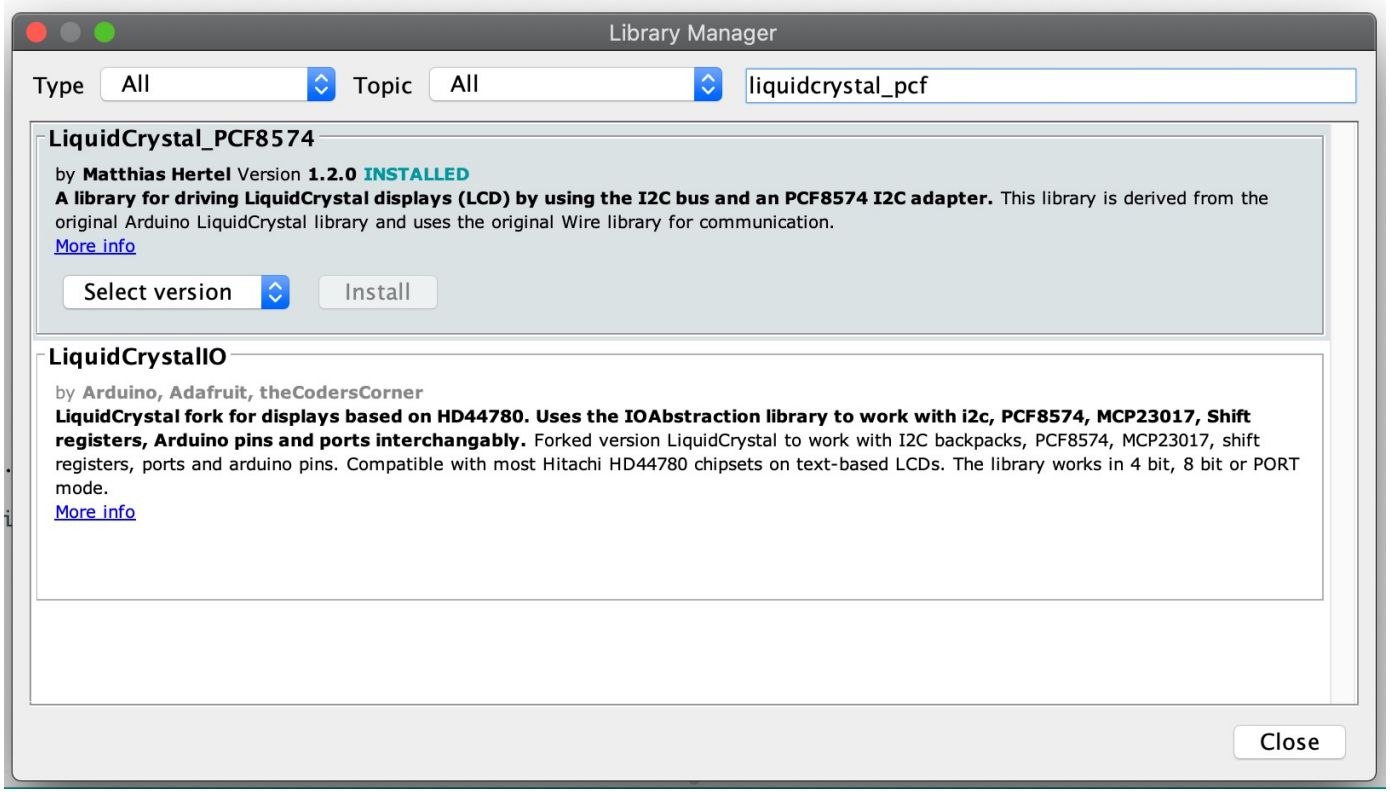
# Breadboard Connection

- The Arduino uses Pins A4 and A5 as SDA and SCL
  - Check pinout slide at the beginning
- Careful: you have two ”types” of LCDs, one has a different pin ordering (VCC, GND, SCL, SDA). **Check PIN names before connecting!!**



# LCD Library

- In the Arduino IDE, select “Tools/Manage Libraries…”
- Look for the `LiquidCrystal_PCF8574` library and install the latest version



# LCD Library

- Install latest version of the `LiquidCrystal_PCF8574` library.
- The library contains a lot of methods to control the LCD display via I2C
  - `begin()` to initialize the display
  - `clear()` clear the display content
  - `home()` set the cursor to row 0, column 0
  - `setCursor(row, col)` sets the cursor to row, col
  - `setBacklight(brightness)` sets the brightness level (0 to 255)
  - etc.
  - You can inspect the source code directly to view the available methods

# LCD Library

- The display class defined in the library inherits from the `Print` class
  - Which means that you can call the `print()` method to print strings directly
  - You don't need to `write()` individual chars one by one.

- The library is based on the `Wire` library for I2C communication.
  - On the Arduino, this library uses pins A4/A5 by default

# New Code Elements

- Include the library and create a display class instance

```
#include <LiquidCrystal_PCF8574.h>

LiquidCrystal_PCF8574 lcd(0x27);
```

I2C Address of the Display

# New Code Elements

- The “datasheet” says that the the address is depends on the three jumper caps on the rear.
- Some of you might have soldered pins instead
- You should all have either address **0x27 or 0x20** by default

| A2 | A1 | A0 | IIC Address |
|---|---|---|---|
| 0 | 0 | 0 | 0x20 |
| 0 | 0 | 1 | 0x21 |
| 0 | 1 | 0 | 0x22 |
| 0 | 1 | 1 | 0x23 |
| 1 | 0 | 0 | 0x24 |
| 1 | 0 | 1 | 0x25 |
| 1 | 1 | 0 | 0x26 |
| 1 | 1 | 1 | 0x27 |

- **0: The Jumper Cap is connected**
- **1: The Jumper Cap is disconnected**

# New Code Elements

- In `setup()` initialize the comunication with the display, set backlight and clear the screen. Then, print a message

```
void setup() {
  lcd.begin(16, 2);
  lcd.setBacklight(255);
  lcd.home();
  lcd.clear();
  lcd.print("Temperature:");
}
```

# Exercise 6: Output Example