



Tecnologie per IoT

Daniele Pagliar

Luca Barbierato

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: 29/03 (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 be able to 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: 08-12/04 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: 17/05 3h
- In-between the 2st and 3nd parts you'll be introduced by prof. Patti to the Internet Protocol Suite (TCP/UDP/IP) and communication protocols (HTTP/MQTT) so you will have the possibility to understand what you are designing.



Lab Organization

- We'll exploit Discord to manage the laboratory!
 - <https://discord.gg/4gE8KQxrMQ>
 - Connect to the server and enter your Group # chat room to collaborate with your group member
 - Absolutely FORBIDDEN to access chat room of other groups!
 - If you need help or have a question, enter the Waiting Room and wait for being accepted in Consulenze where we will have the possibility of discussing about your concerns
- No report, just code!
 - Send me the code in a *.zip file
 - 1 WEEK AFTER EACH LABORATORY!



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 AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to. Circuit designers can make their own version of the module, extending it and improving it.



Common Features

- Most Arduino boards are based on **Atmel AVR architecture** microcontrollers
- 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 through two rows of pin headers 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 .



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





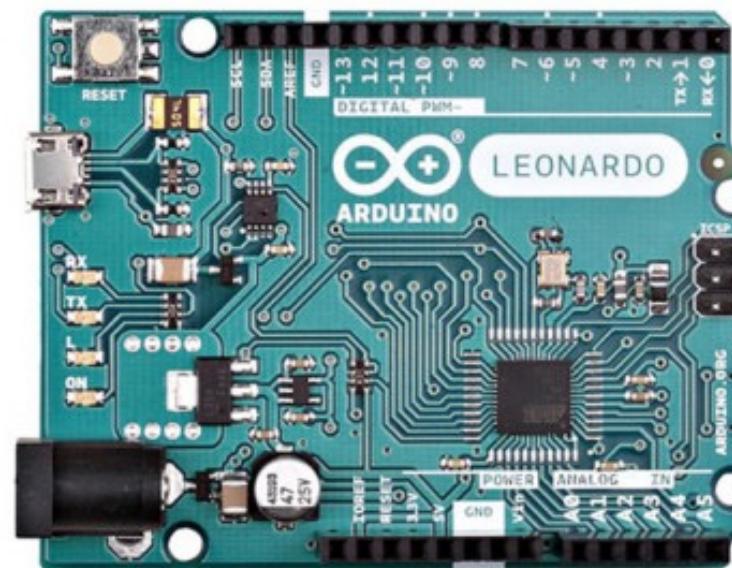
Arduino Uno (WiFi)

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

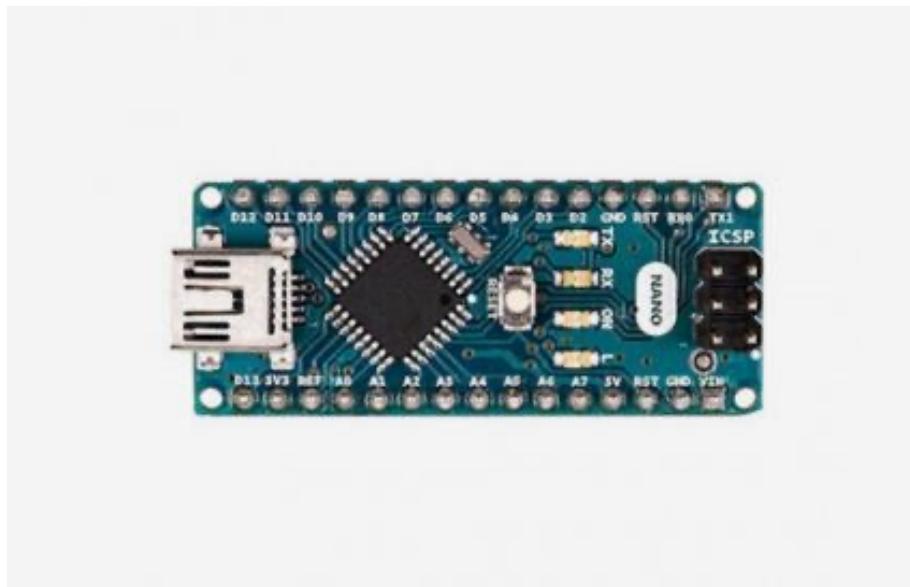
- Leonardo (ATmega32u4)
- Built-in USB capabilities (can act as a keyboard or mouse)





Other Arduinos

- Nano (ATmega328) and Micro (ATmega32U4)
- Similar to Uno and Leonardo respectively, but in a small form factor





Other Arduinos

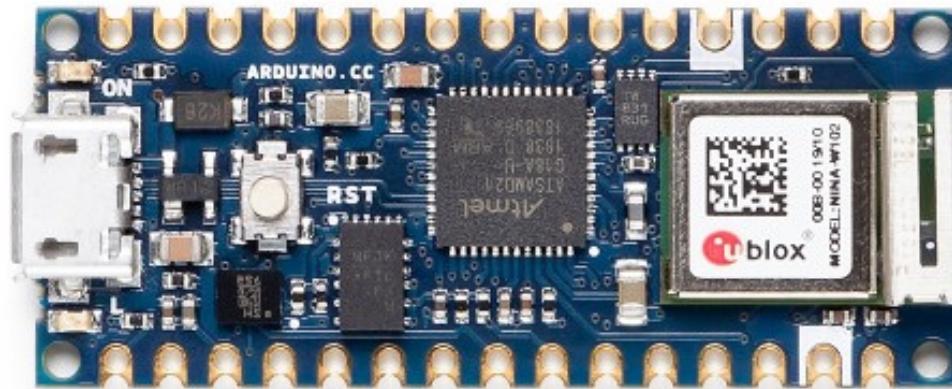
- Modern Arduinos replaced the 8-bit AVR architecture with **32-bit ARM Cortex**
- Arduino Due (Cortex M3)





Other Arduinos

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





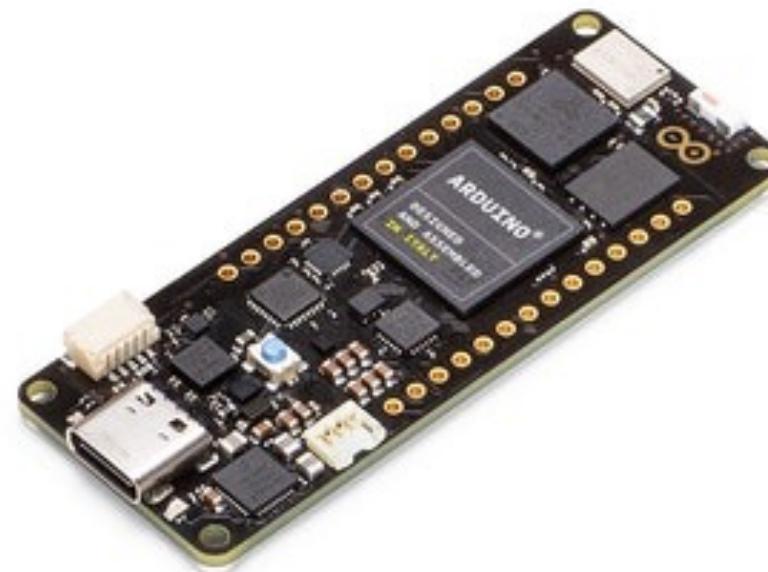
Other Arduinos

- 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.



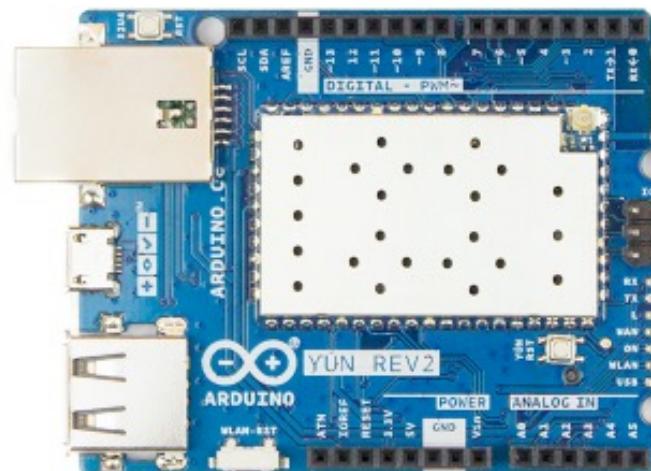


ARDUINO YÙN REV2



Arduino Yùn Rev 2

- Two processors:
 - Atmega 32U4 (AVR), for Arduino-style programming
 - Atheros AR9331 (MIPS), equipped with a minimal Linux (OpenWRT Linino), for advanced network connections and applications.





Yùn Specs

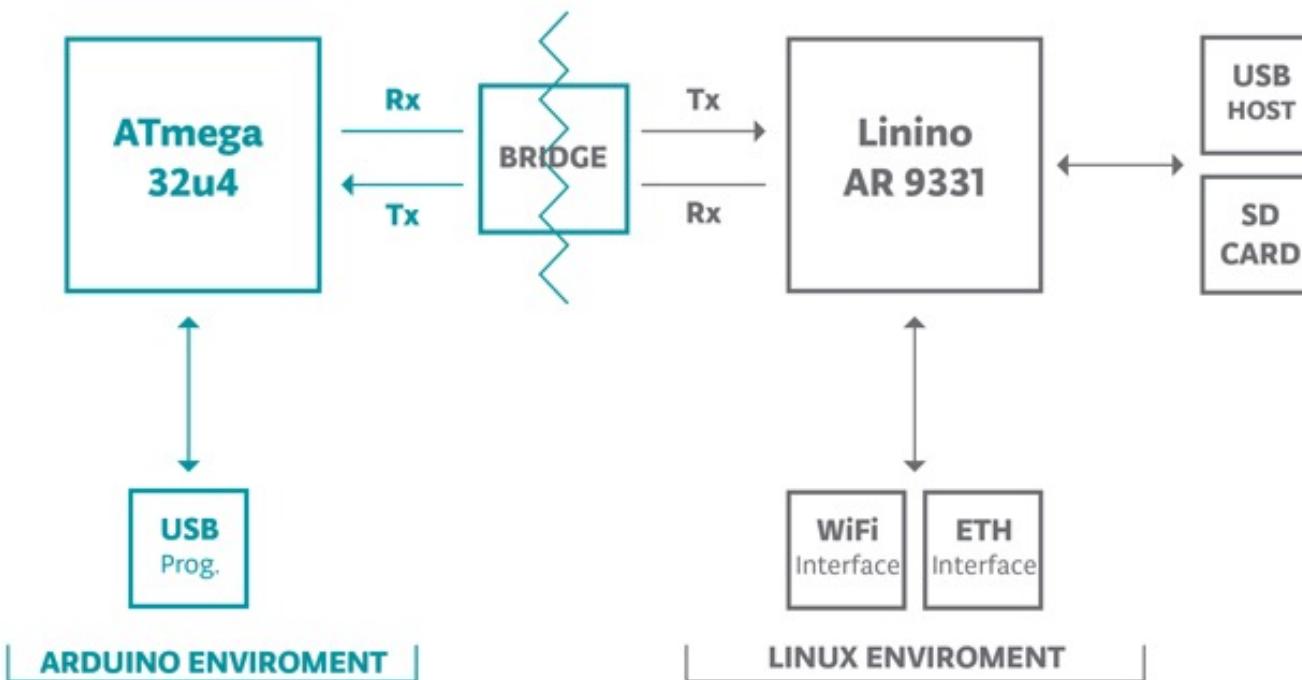
Microcontroller	ATmega32U4
Operating Voltage	5V
Input Voltage	5V
Digital I/O Pins	20
PWM Output	7
Analog I/O Pins	12
DC Current per I/O Pin	40 mA on I/O Pins; 50 mA on 3,3 Pin
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

Processor	Atheros AR9331
Architecture	MIPS
Operating Voltage	3.3V
Ethernet	802.3 10/100Mbit/s
WiFi	802.11b/g/n 2.4 GHz
USB Type	2.0 Host
Card Reader	Micro-SD
RAM	64 MB DDR2
Flash Memory	16 MB
Clock Speed	400 MHz



Bridge Port

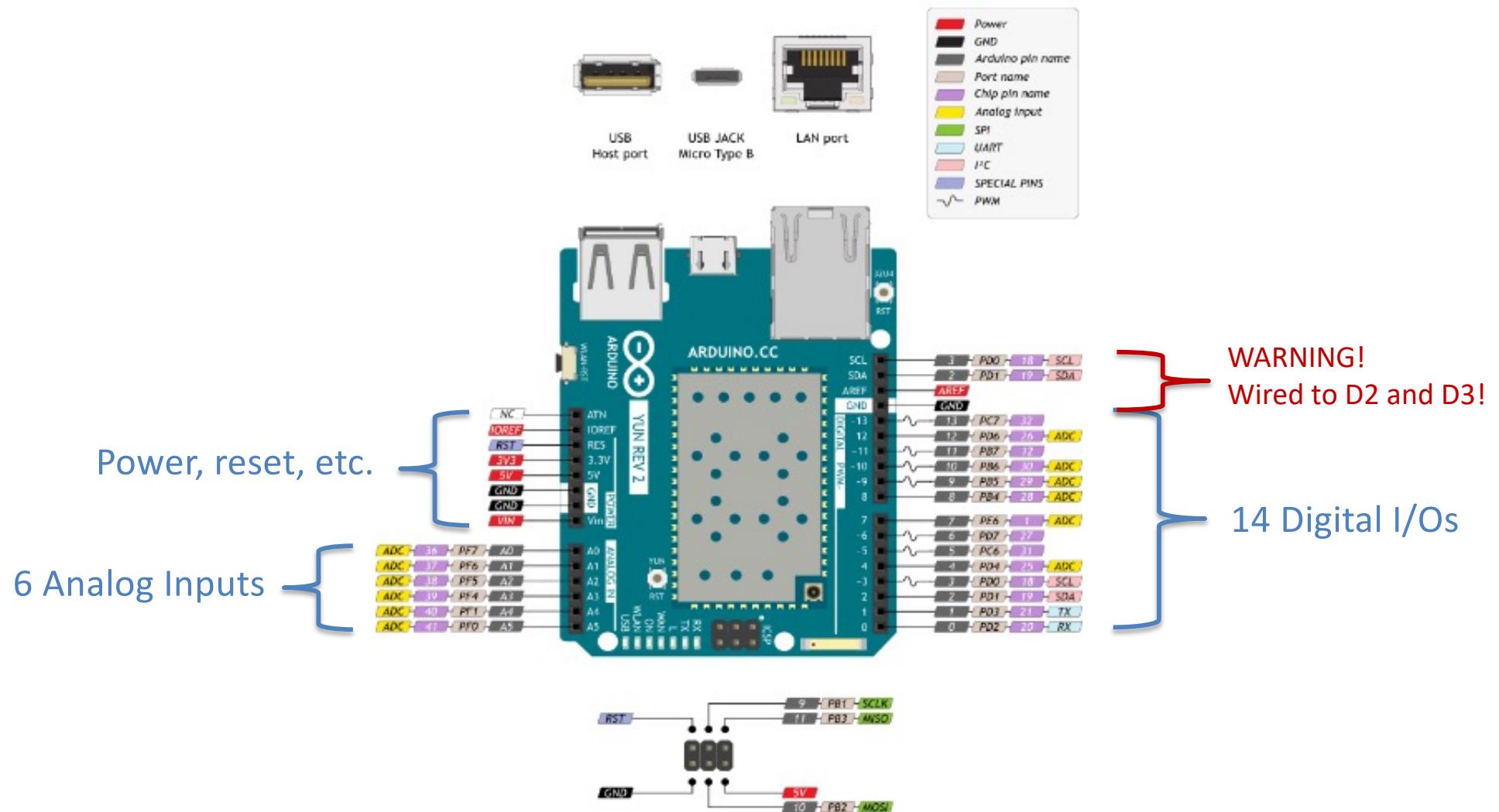
- The two processors communicate through a Bridge serial port





Yùn Pinout

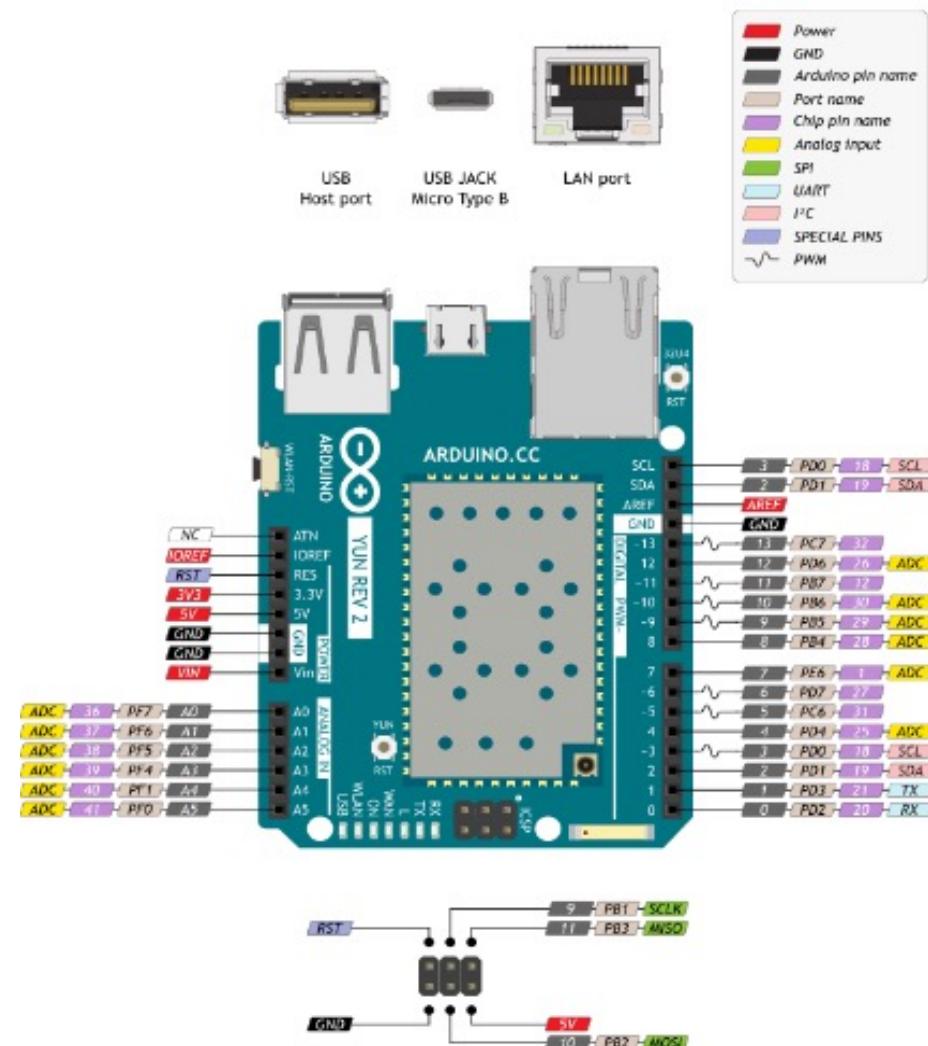
ARDUINO YUN REV 2 / PINOUT





Yùn Pinout

ARDUINO YUN REV 2 / PINOUT



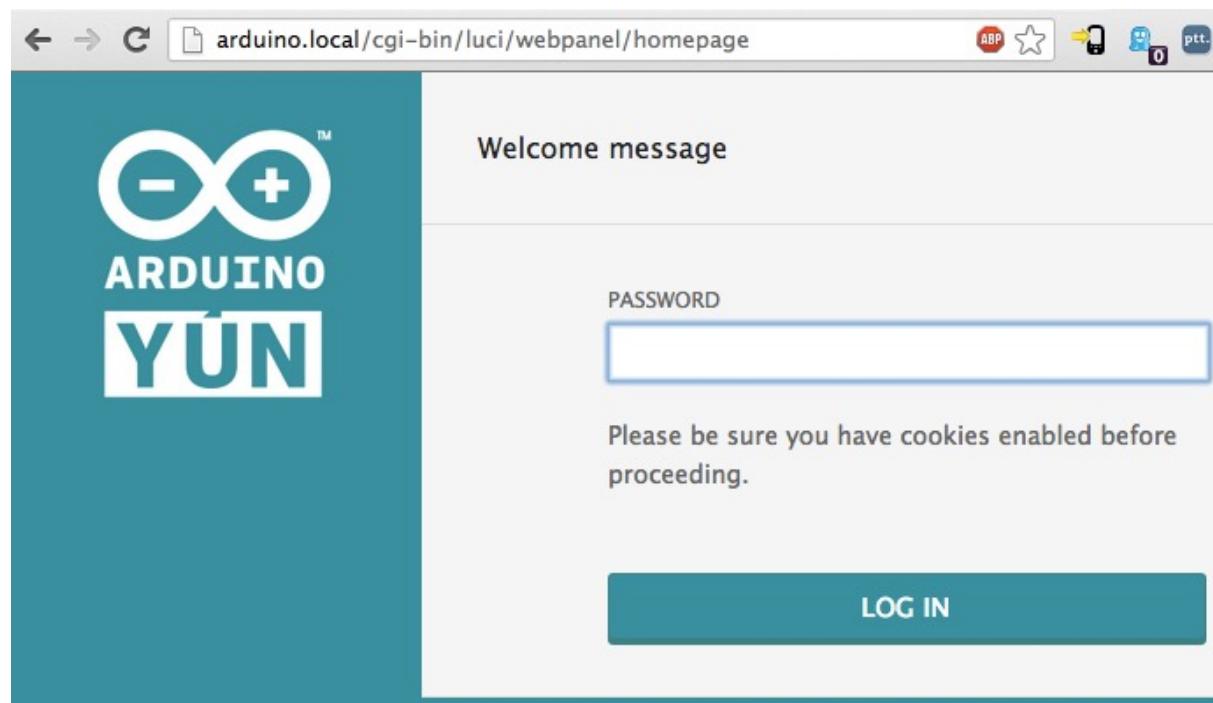


ARDUINO YÙN FIRST SETUP



First Setup

- At first boot, the Yùn creates a WiFi network called *ArduinoYun-XXXXXXXXXXXX*
 - Connect to it, open a browser and go to <http://arduino.local> or <http://192.168.240.1>
 - You should see this page (default password: **arduino**)





First Setup

- In the following configuration page, follow the instructions to configure the Yún and let it connect to your home WiFi network or smartphone hotspot.

WELCOME TO ARDUINO, YOUR ARDUINO YÚN

CONFIGURE

WIFI (WLAN0) CONNECTED

Address	192.168.240.1
Netmask	255.255.255.0
MAC Address	B4:21:8A:00:00:10
Received	105.72 KB
Trasmitted	160.48 KB

WIRED ETHERNET (ETH1) DISCONNECTED

MAC Address	B4:21:8A:08:00:10
Received	0.00 B
Trasmitted	0.00 B

YÚN BOARD CONFIGURATION

YÚN NAME *

PASSWORD

CONFIRM PASSWORD

TIMEZONE *

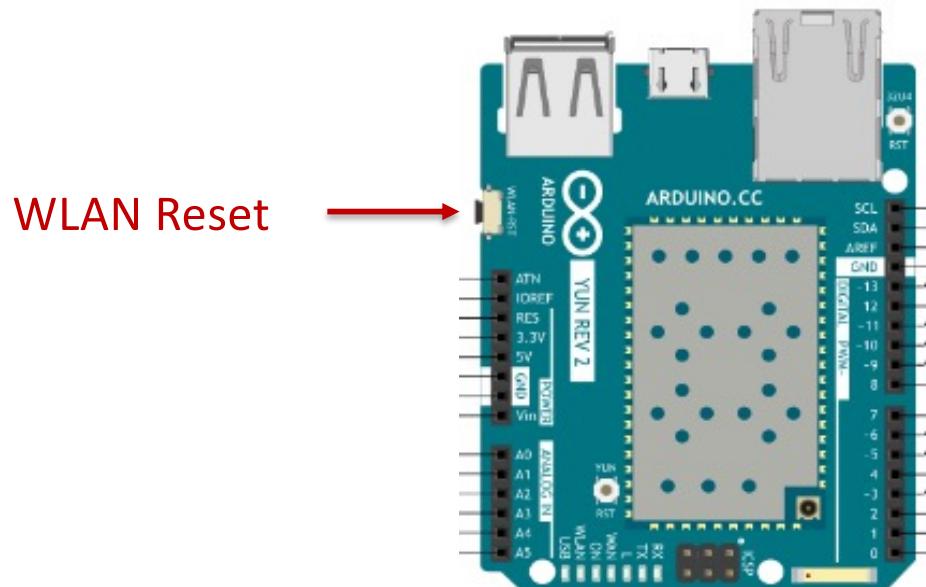
DISCARD

CONFIGURE & RESTART



First Setup

- If for any reason your Yùn does not create the WiFi network at boot, perform a factory restore holding the **WLAN reset** button for more than 30s
 - Other instructions for setup can be found [here](#)





THE ARDUINO IDE

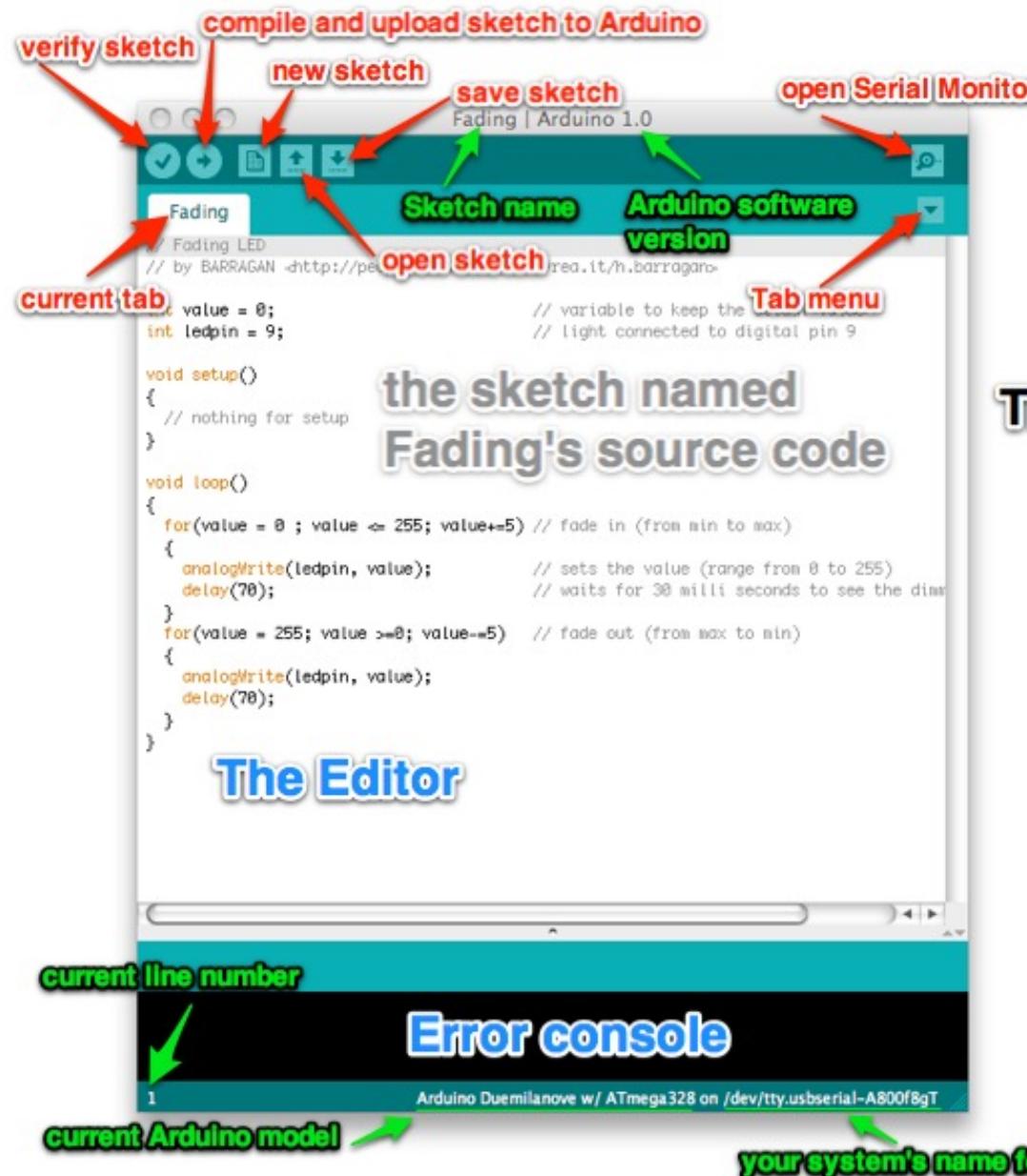


The Arduino IDE

- 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



The Arduino IDE

Source: STEMify



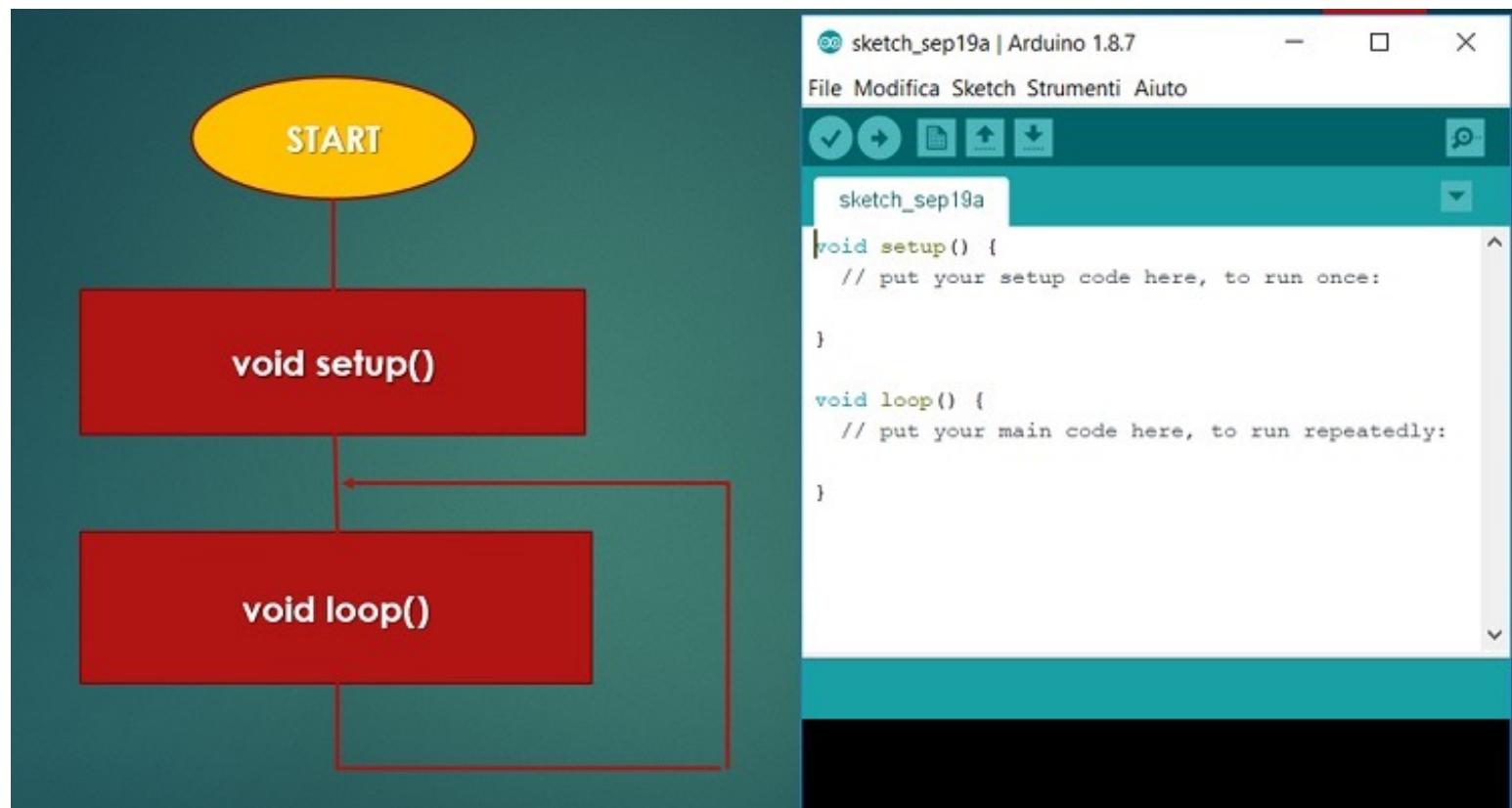
Arduino Sketches

- 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/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](#).

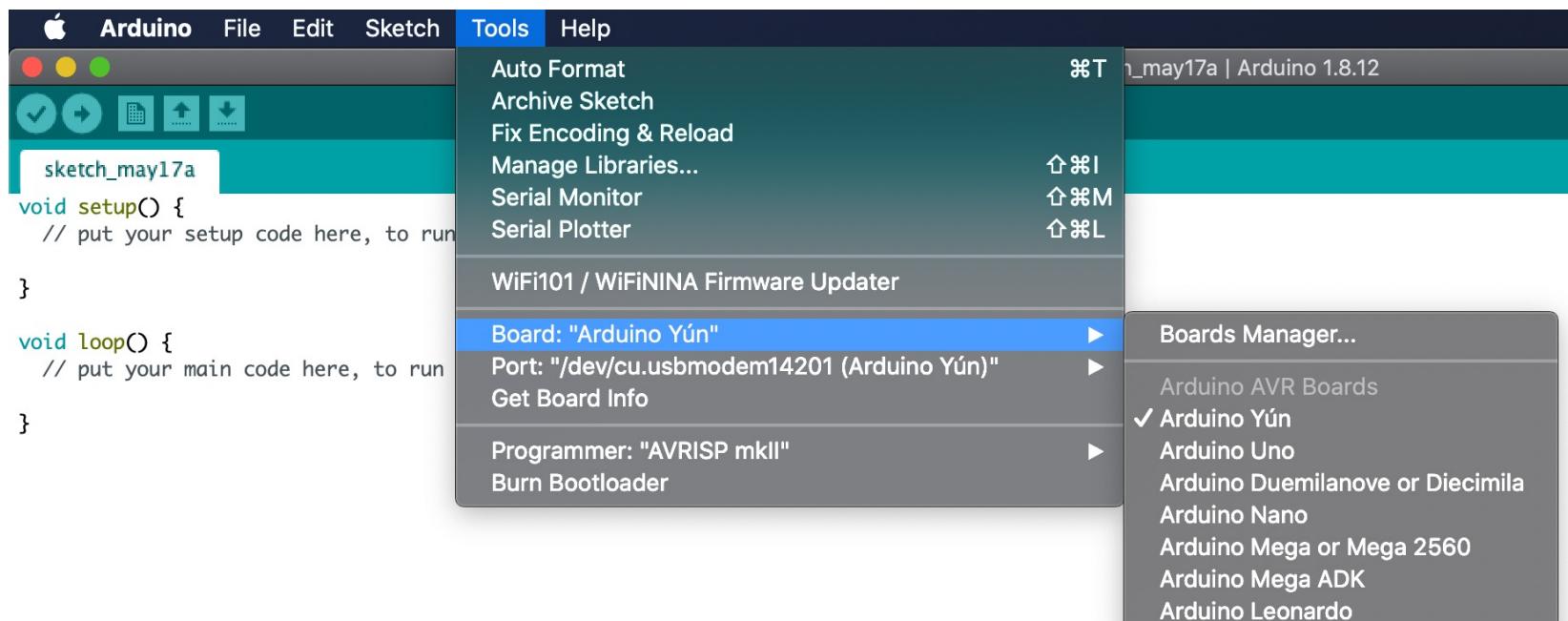


FIRST SKETCH



First Sketch

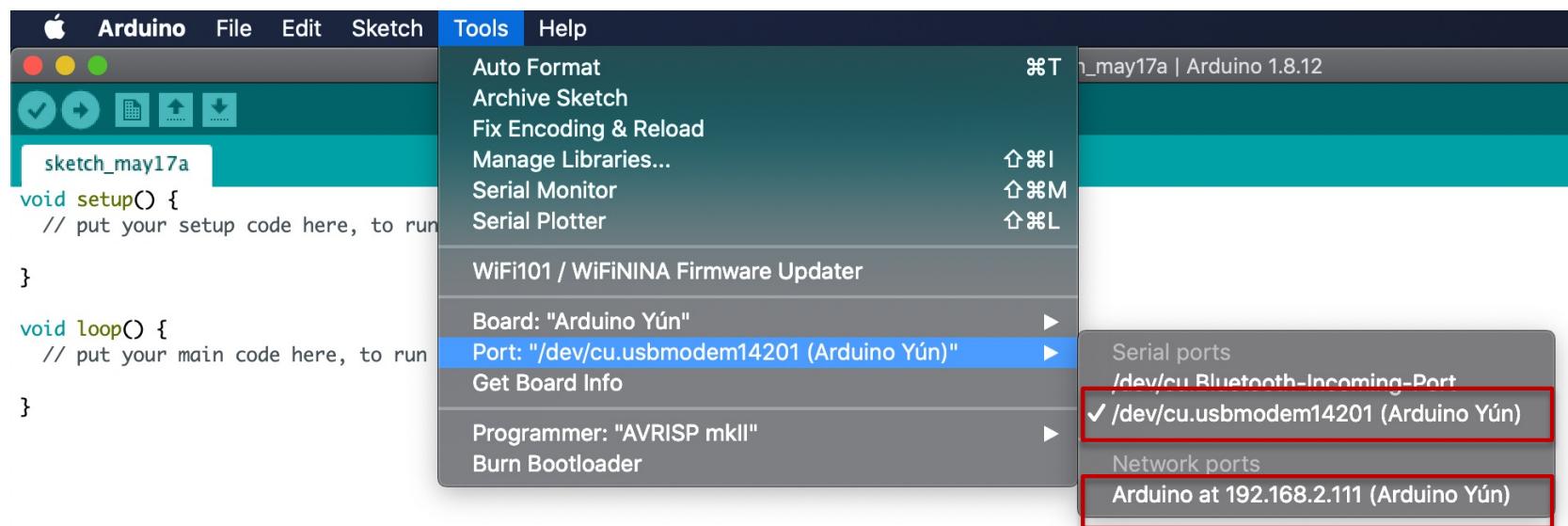
- Let's test if we can program the Yún
 1. Connect the Yún to your PC using a micro-USB cable
 2. In the Arduino IDE, under "Tools/Board", select the Yún.





First Sketch

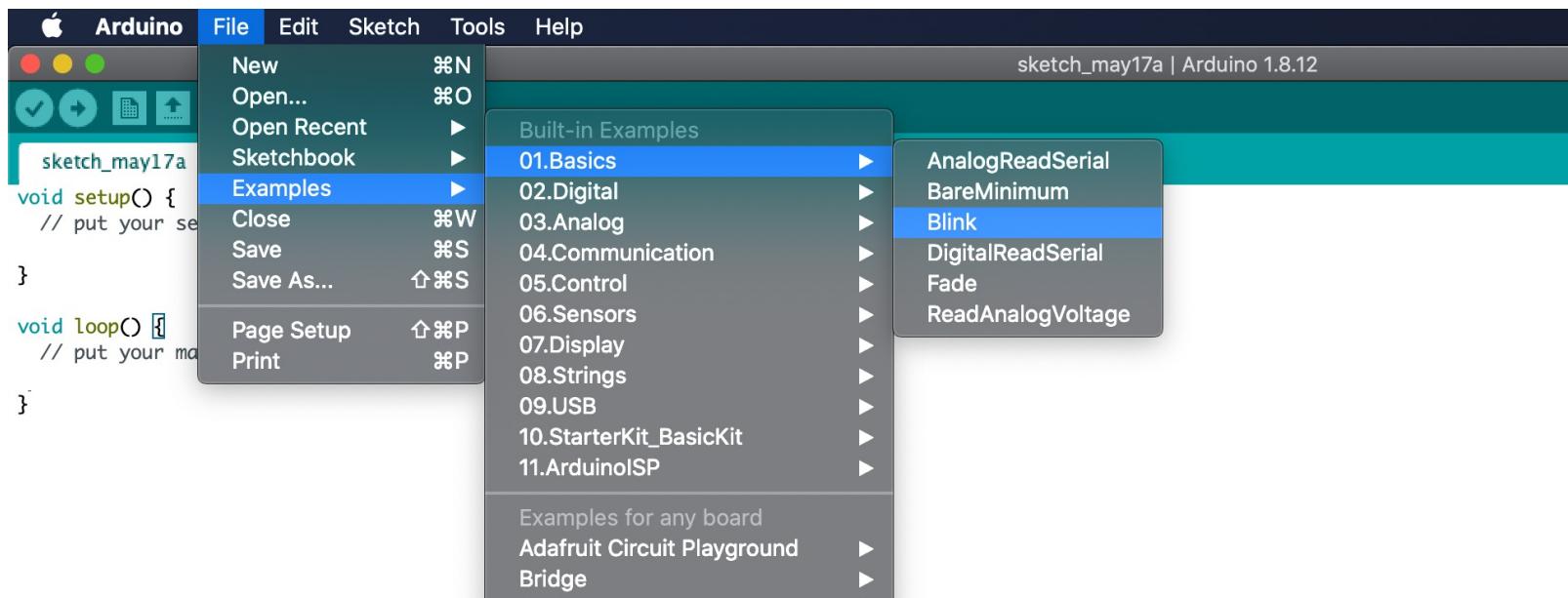
- Let's test if we can program the Yún
 - 3. Under "Tools/Port" select the appropriate port:
 - Normally, we will use the Serial port
 - Network programming is useful if the Yún is not physically connected to your PC.





First Sketch

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





First Sketch

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

Blink | Arduino 1.8.12

by Colby Newman

This example code is in the public domain.

```
http://www.arduino.cc/en/Tutorial/Blink
*/
// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
    delay(1000);                      // wait for a second
    digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
    delay(1000);                      // wait for a second
}
```

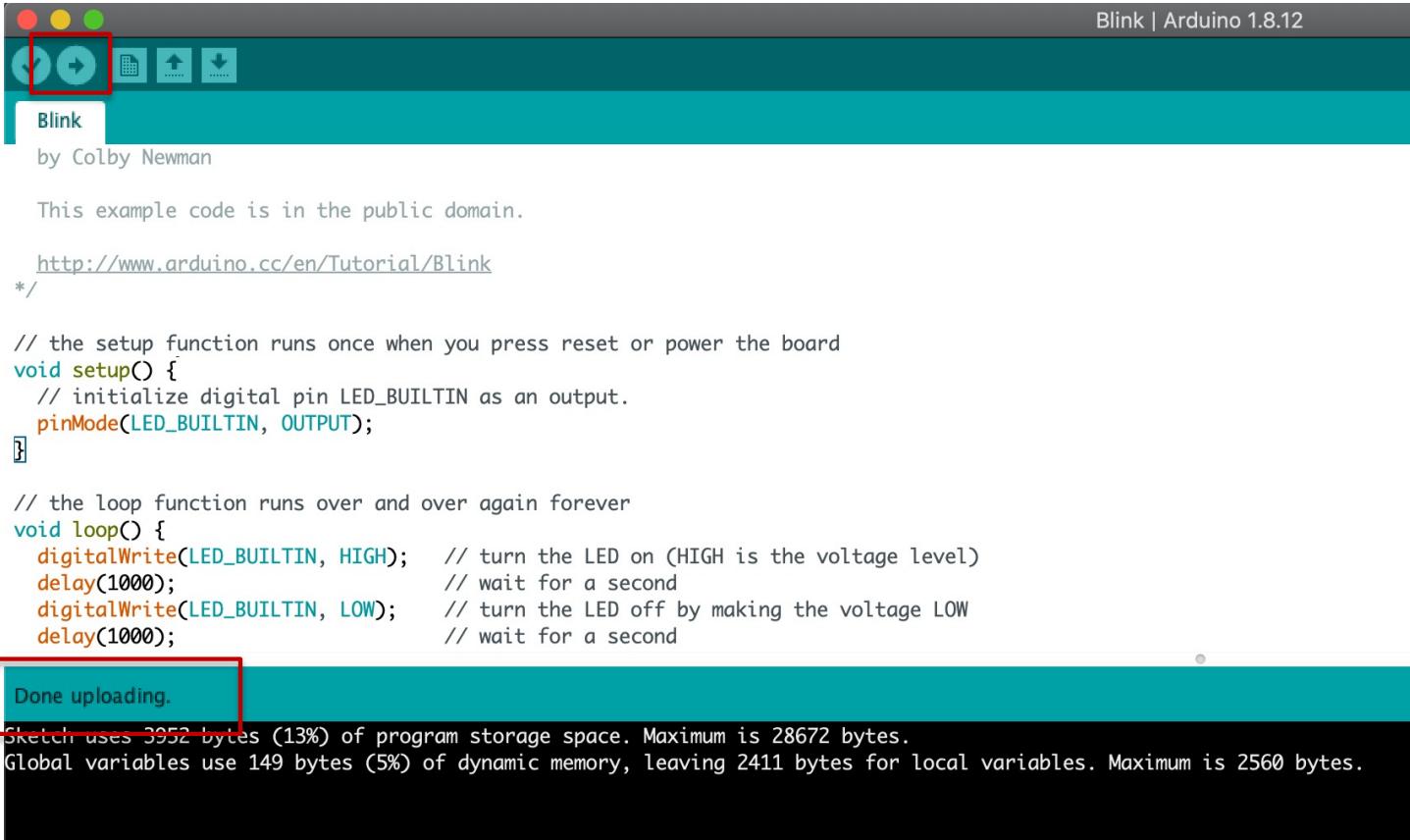
Done compiling.

Sketch uses 3952 bytes (13%) of program storage space. Maximum is 28672 bytes.
Global variables use 149 bytes (5%) of dynamic memory, leaving 2411 bytes for local variables. Maximum is 2560 bytes.



First Sketch

- Let's test if we can program the Yun
6. Upload the sketch to the Yùn



Blink | Arduino 1.8.12

Sketch: Blink by Colby Newman

This example code is in the public domain.

```
http://www.arduino.cc/en/Tutorial/Blink
*/
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

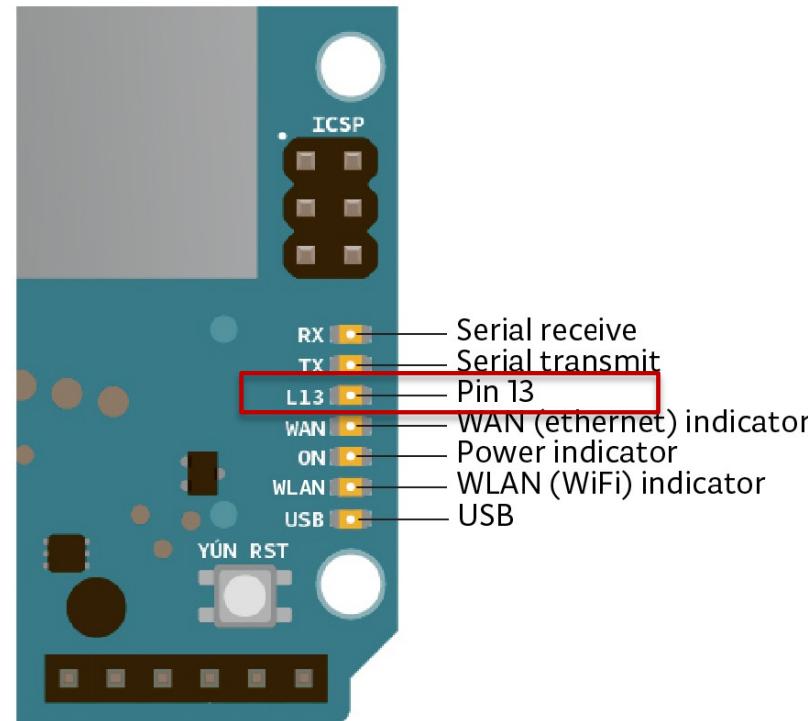
Done uploading.

Sketch uses 3952 bytes (13%) of program storage space. Maximum is 28672 bytes.
Global variables use 149 bytes (5%) of dynamic memory, leaving 2411 bytes for local variables. Maximum is 2560 bytes.



First Sketch

- Let's test if we can program the Yun
 - 6. Check if the sketch is working
 - Internal LED (pin 13) blinking with a 2s period.





PART 1: EXERCISE 1



Exercise 1

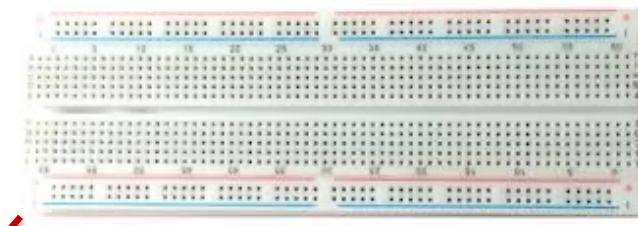
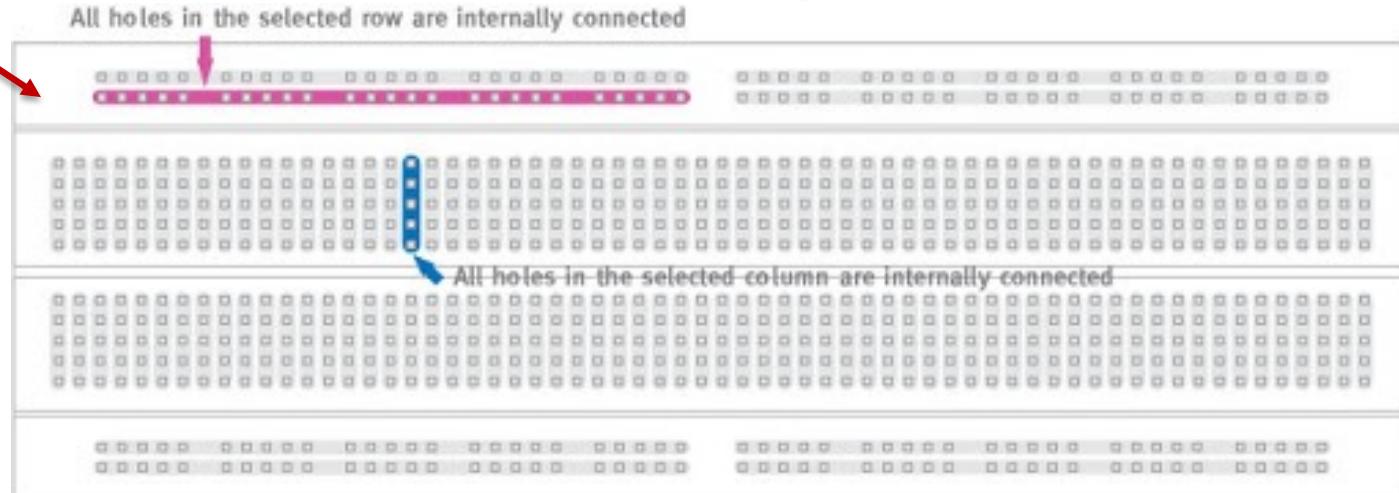
- HW components involved:
 - Breadboard
 - LEDs
 - Resistors
- SW components involved:
 - Digital GPIO outputs
 - Interrupts



Breadboard

- Build simple circuits without soldering

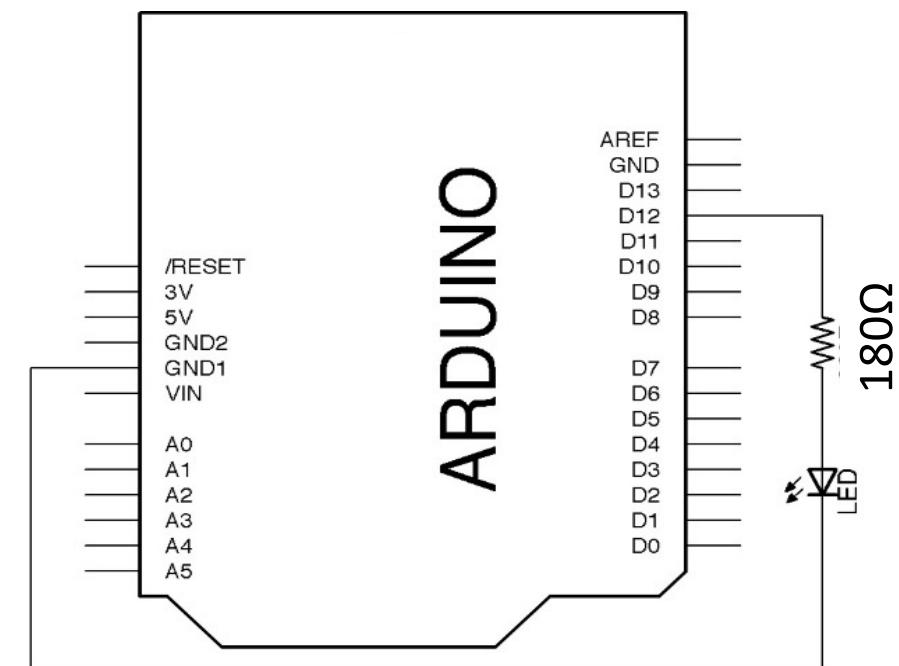
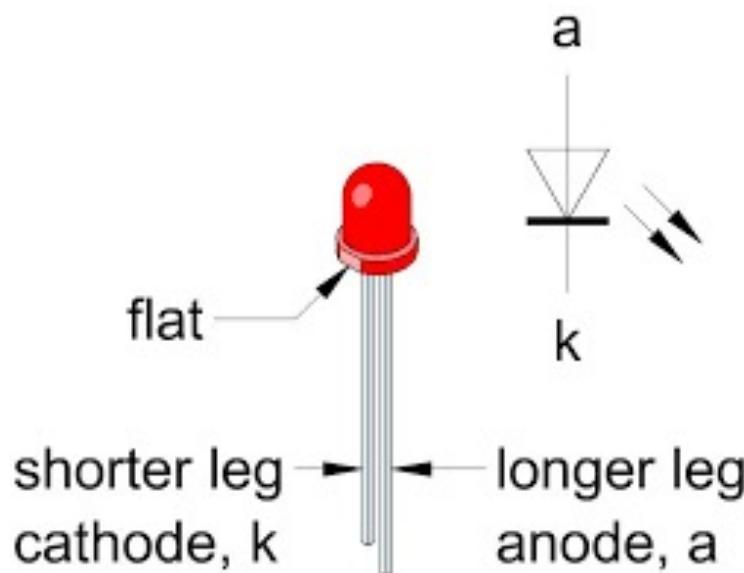
Normally for
VCC/GND





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).



Caution: longer leg is the anode

Circuit



Resistors Color-coding

- We use 180 Ohm resistor: $5V/180\Omega = 27mA < 40mA$ (max per GPIO pin)

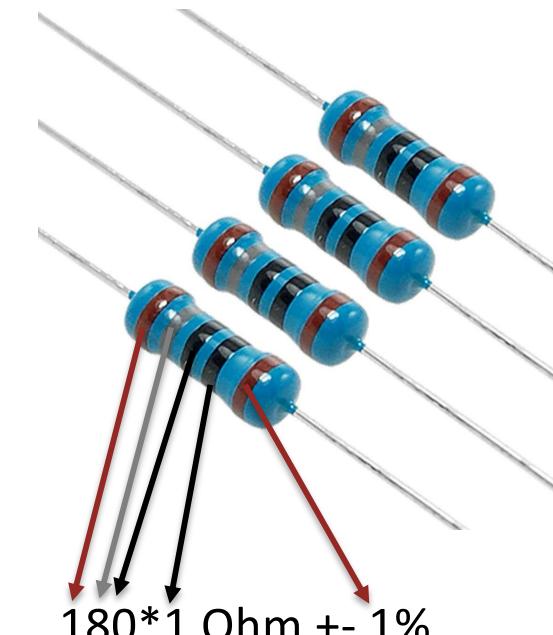
4-Band-Code 560k Ω $\pm 5\%$

2%, 5%, 10%

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

0.1%, 0.25%, 0.5%, 1% 237 Ω $\pm 1\%$

5-Band-Code

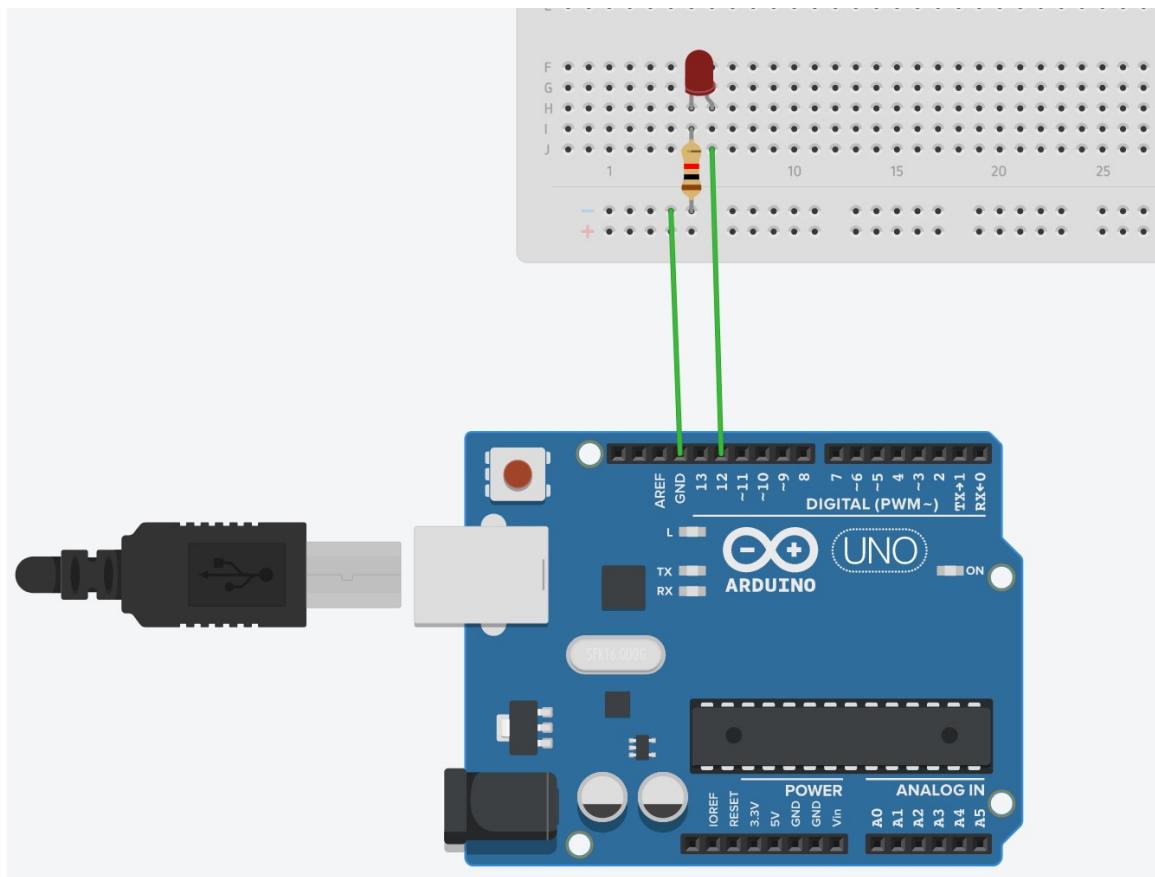


180 * 1 Ohm $\pm 1\%$



LED Connection

- Breadboard connection (with minimal wires)





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 Yún. Also, some of the most complex sensors/actuators that we'll use are not there
 - However, you can try these first examples using the Uno and the generic components available (LEDs, resistors, etc.)



Code

```
#include <TimerOne.h>
```

```
const int RLED_PIN = 12;  
const int GLED_PIN = 11;
```

```
const float R_HALF_PERIOD = 1.5;  
const float G_HALF_PERIOD = 3.5;
```

```
int greenLedState = LOW;  
int redLedState = LOW;
```

- TimerOne.h allows to attach interrupts to the MCU's Timer1

- Normally used to generate PWM signals
- If you want to use the library and also use PWM outputs you have to take care (see library documentation [here](#)).

- GPIO pins are identified by integer constants

- global variables to store the current state of the two LEDs

- LOW and HIGH MACROS for 0 and 1 respectively



Code

- In the setup() function:
 - set the two LED pins as outputs
 - Initialize the timer
 - Attach an ISR function to the timer expiration

```
void setup() {
    pinMode(RLED_PIN, OUTPUT);
    pinMode(GLED_PIN, OUTPUT);
    Timer1.initialize(G_HALF_PERIOD * 1e06);
    Timer1.attachInterrupt(blinkGreen);
}
```



Code

- In the ISR: toggle one LED state asynchronously w.r.t. loop()

```
void blinkGreen() {  
    greenLedState = !greenLedState;  
    digitalWrite(GLED_PIN, greenLedState);  
}
```



Code

- In the loop() function: toggle the other LED state using the delay() function

```
void loop() {  
    redLedState = !redLedState;  
    digitalWrite(RLED_PIN, redLedState);  
    delay(R_HALF_PERIOD * 1e03);  
}
```



Complete Code

```
#include <TimerOne.h>

const int RLED_PIN = 12;
const int GLED_PIN = 11;

const float R_HALF_PERIOD = 1.5;
const float G_HALF_PERIOD = 3.5;

int greenLedState = LOW;
int redLedState = LOW;

void blinkGreen() {
    greenLedState = !greenLedState;
    digitalWrite(GLED_PIN, greenLedState);
}

void setup() {
    pinMode(RLED_PIN, OUTPUT);
    pinMode(GLED_PIN, OUTPUT);
    Timer1.initialize(G_HALF_PERIOD * 1e06);
    Timer1.attachInterrupt(blinkGreen);
}

void loop() {
    redLedState = !redLedState;
    digitalWrite(RLED_PIN, redLedState);
    delay(R_HALF_PERIOD * 1e03);
}
```



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 established
 - The program will not start until you open the Serial monitor in the IDE
- Print a newline-terminated string.

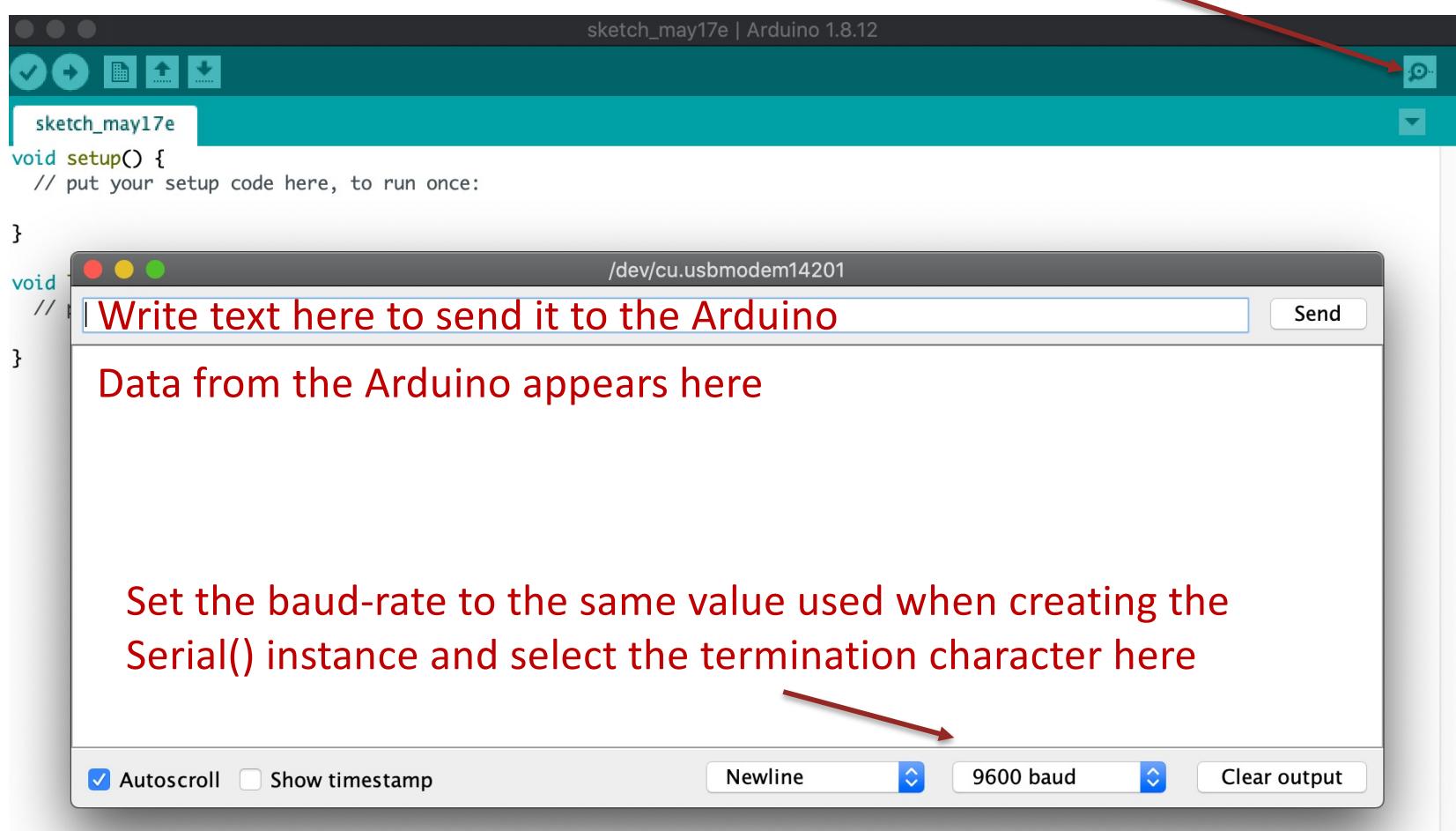
```
void serialPrintStatus() {  
    if (Serial.available() > 0) {  
        int inByte = Serial.read();  
  
        if(inByte == ...) {  
            //etc|
```

- 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 select Port





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 the 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

```
Lab 1.2 Starting
LED 2 Status: 1
LED 3 Status: 0
LED 3 Status: 1
Invalid command
LED 2 Status: 1
```

Autoscroll Show timestamp No line ending 9600 baud Clear output



PART 1: EXERCISE 3



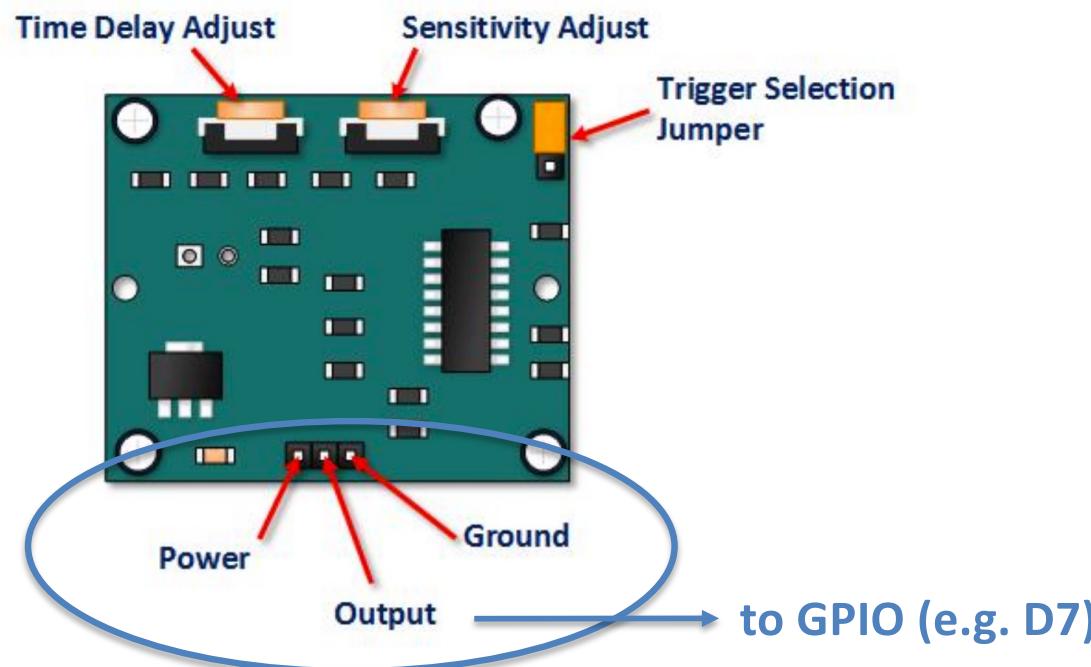
Exercise 3

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



PIR Motion Sensor: HC-SR501

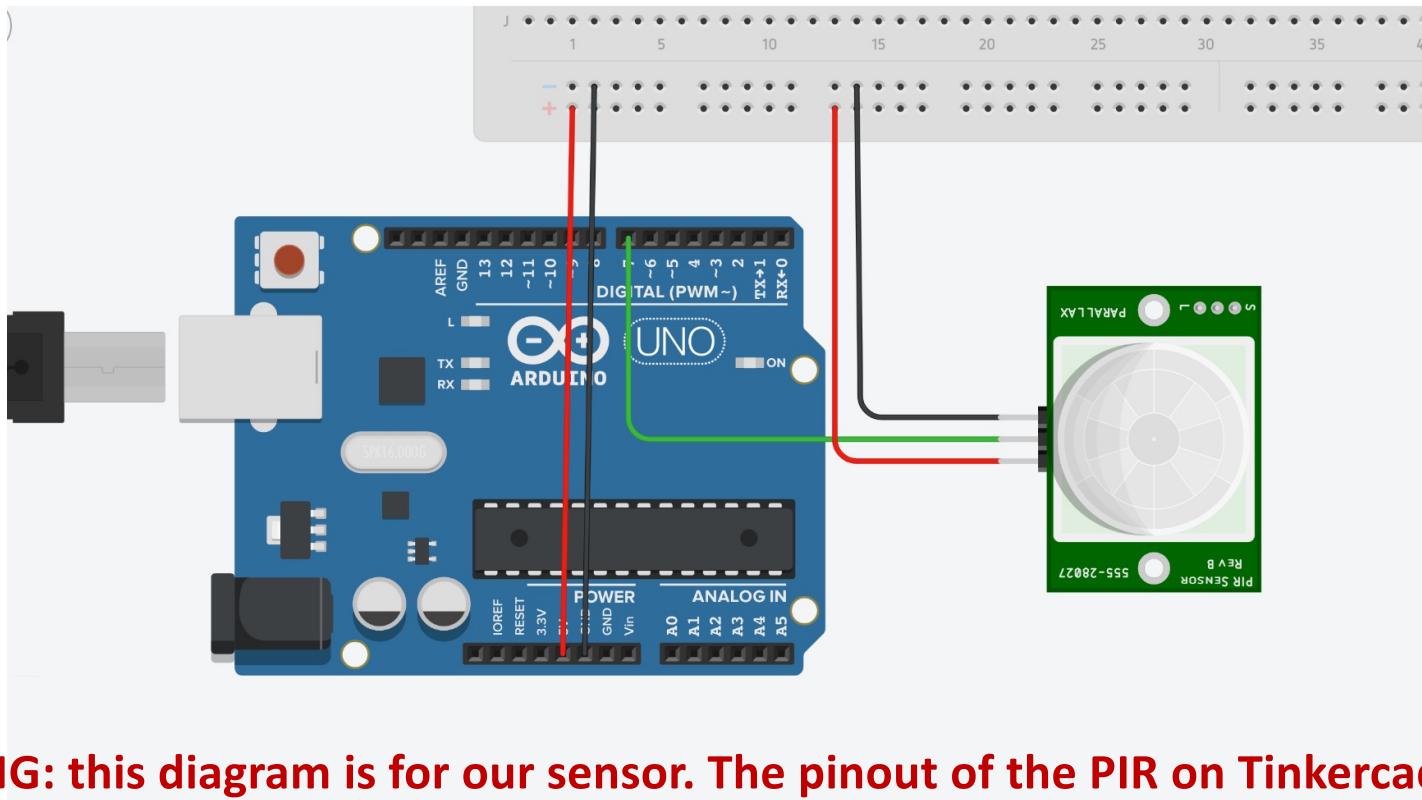
- Two potentiometers to set the length of the detection and the sensitivity
- One jumper to select between 2 operating modes (single and repeatable trigger)
- Use the datasheet to understand each configuration parameter: [link](#)





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

- Not all GPIO pins support interrupts (see the numbers for “Micro, Leonardo, other 32u4-based” at [this page](#))
- `tot_count` is touched by ISR and `loop()` → `volatile`

```
const int LED_PIN = 12;  
const int PIR_PIN = 7;  
  
volatile int tot_count = 0;
```

- Attach the ISR function to the appropriate GPIO interrupt.

```
void setup() {  
    pinMode(PIR_PIN, INPUT);  
    attachInterrupt(digitalPinToInterrupt(PIR_PIN), checkPresence, CHANGE);  
    // etc...
```

PIN number to Interrupt number ISR function pointer RISING, FALLING, CHANGE, etc.



Exercise 3: Output Example

```
Lab 1.3 starting
Total people count: 0
Total people count: 1
Total people count: 1
Total people count: 3
```

Send

Autoscroll Show timestamp No line ending 9600 baud Clear output



PART 1: EXERCISE 4



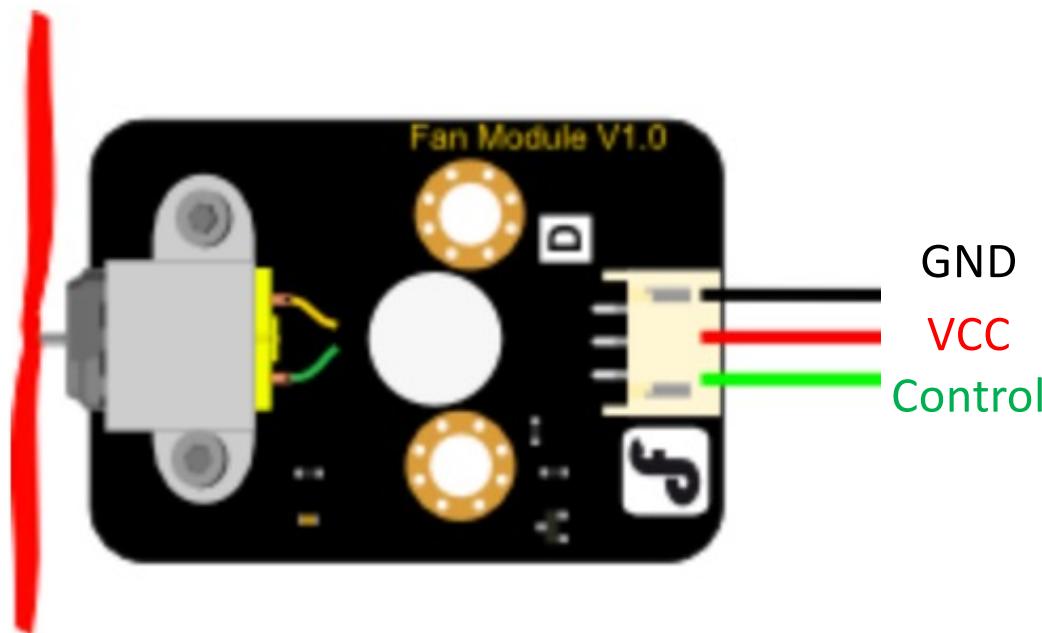
Exercise 4

- 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

- Not all GPIOs support PWM. Those that do, have a “tilde” symbol (~) next to the PIN number
- 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



The screenshot shows a terminal window titled '/dev/cu.usbmodem14201'. The window contains the following text output:

```
Increasing speed: 204.00
Increasing speed: 229.50
Increasing speed: 255.00
Decreasing speed: 229.50
Decreasing speed: 204.00
Decreasing speed: 178.50
Decreasing speed: 153.00
Decreasing speed: 127.50
Decreasing speed: 102.00
Decreasing speed: 76.50
Decreasing speed: 51.00
Decreasing speed: 25.50
Decreasing speed: 0.00
Already at min speed
Already at min speed
```

At the bottom of the window, there are several configuration options:

- Autoscroll Show timestamp
- No line ending
- 9600 baud
- Clear output



PART 1: EXERCISE 5



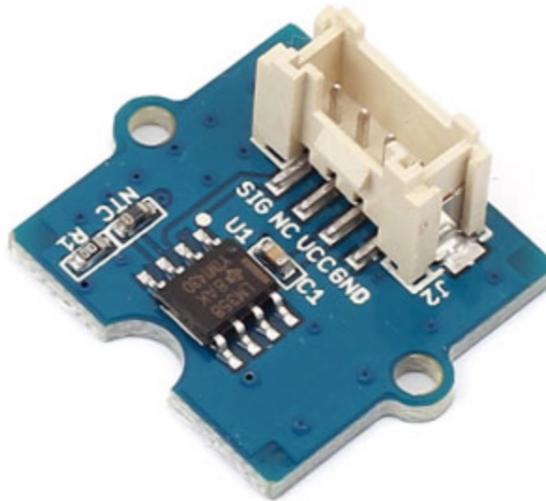
Exercise 5

- 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 if needed (to use male/male cables instead of female/male)



Grove Temperature Sensor

- The sensor produces an output **voltage** which depends (through a variable **resistance**, 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° C
 - **B = 4275K**; defines the shape of the (non-linear) relation between resistance and temperature. A definition can be found [here](#).

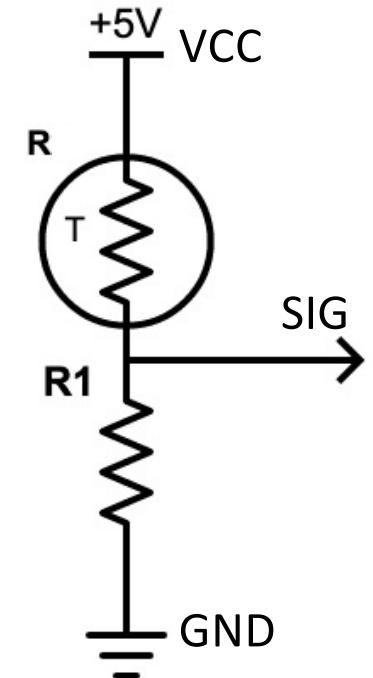


Grove Temperature Sensor

- Step 1:
 - $R_1 = R_0 = 100\text{KOhm}$
 - We can extract R from V_{sig} using a simple resistive divider equation:

$$V_{sig} = \frac{R_0}{R + R_0} * VCC \rightarrow R = \left(\frac{VCC}{V_{sig}} - 1 \right) * R_0$$

- 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 datasheet:

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

- Warning: in this equation both T and 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

A screenshot of a terminal window titled '/dev/cu.usbmodem14201'. The window contains the following text:

```
Lab 1.5 starting
temperature = 18.38
temperature = 18.38
temperature = 19.74
temperature = 24.39
temperature = 22.78
temperature = 22.06
temperature = 21.50
temperature = 21.18
temperature = 20.86
temperature = 20.62
temperature = 20.38
```

The terminal window has a 'Send' button in the top right corner. At the bottom, there are several configuration options: 'Autoscroll' (checked), 'Show timestamp' (unchecked), 'No line ending' (dropdown menu), '9600 baud' (dropdown menu), and 'Clear output' (button).

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



PART 1: EXERCISE 6



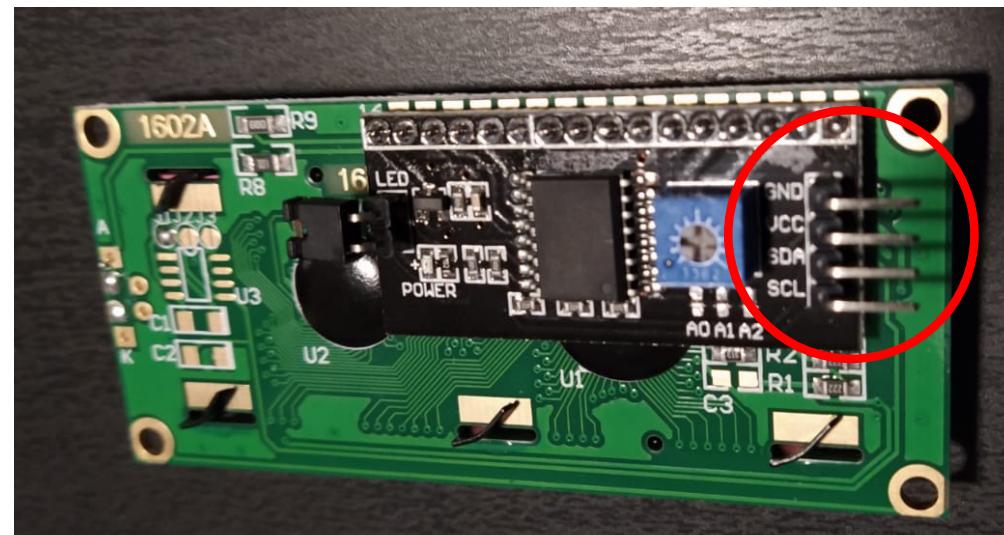
Exercise 6

- HW components involved:
 - Temperature Sensor
 - LCD Display (with I2C connection)
- SW component involved:
 - LCD library (based on Wire)



DFRobot I2C Display

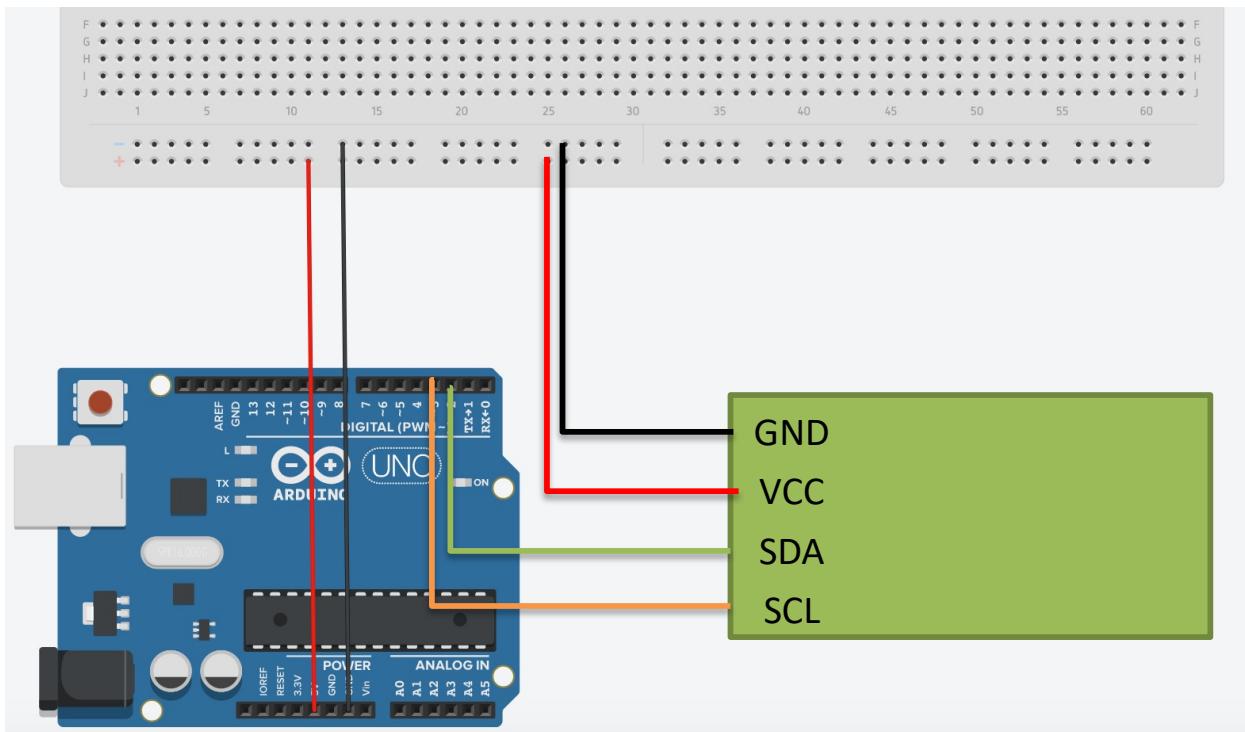
- A sort of datasheet can be found [here](#)
 - **Careful:** we won't use the library provided at that link, but a standard library provided by the Arduino IDE's Library Manager, compatible with this display
 - So don't copy the code! It won't work





Breadboard Connection

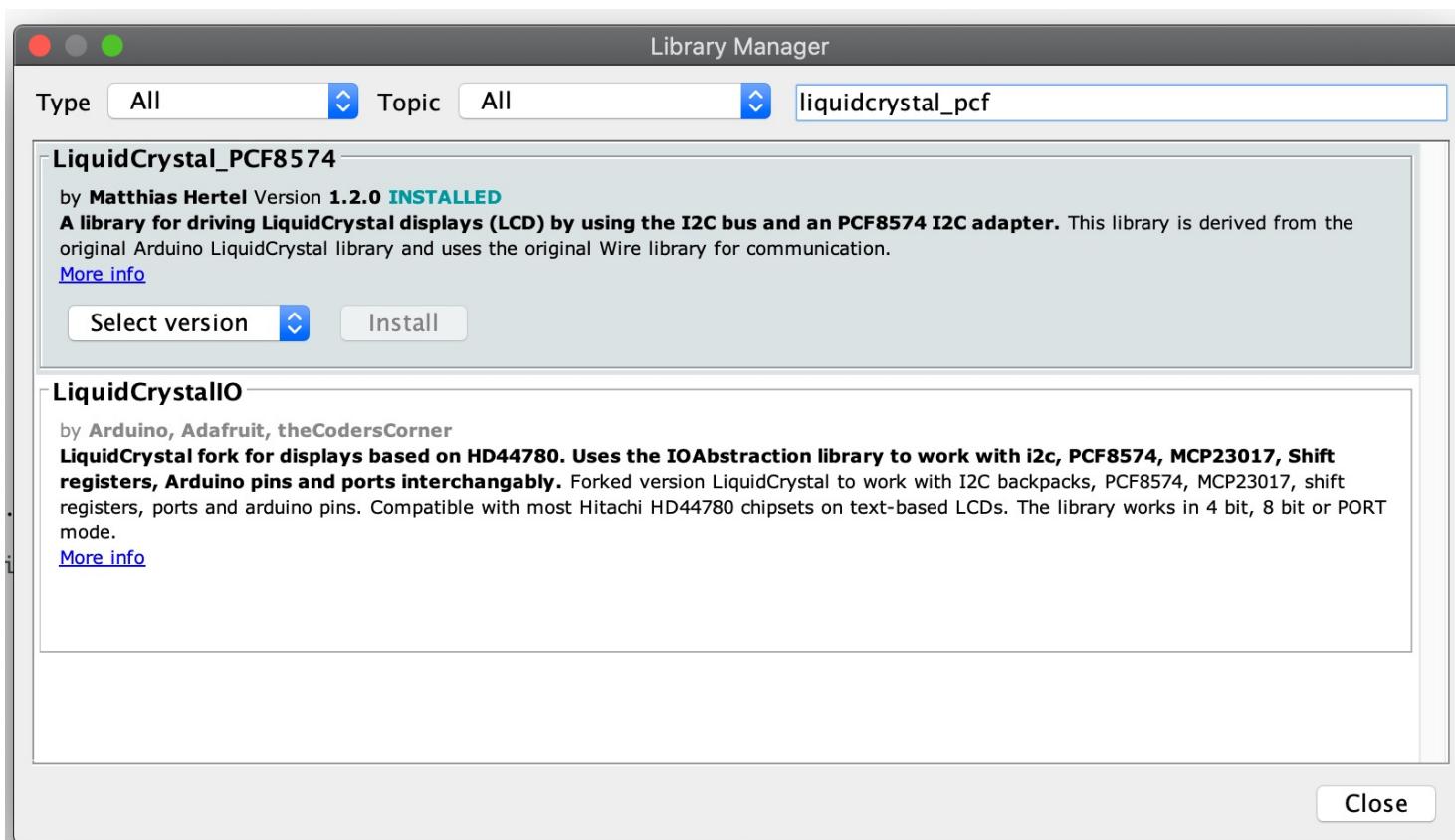
- The Yùn has two additional slots for SDA and SCL pins (for shields) but these are wired to D2 and D3 respectively!
 - Check pinout slide at the beginning





LCD Library

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





LCD Library

- The library contains a lot of methods to control the LCD display via I2C
 - `init()` 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 Yùn, this library uses pins D2/D3 by default



New Code Elements

- Include the library and create a display class instance
 - The “datasheet” says that the default I2C address is 0x20 but this is **wrong!**
 - Indeed, the three jumpers A2, A1 and A0 on the rear of the display are all disconnected → Address = 0x27

```
#include <LiquidCrystal_PCF8574.h>

LiquidCrystal_PCF8574 lcd(0x27);
```



New Code Elements

- In `setup()` initialize the communication 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

