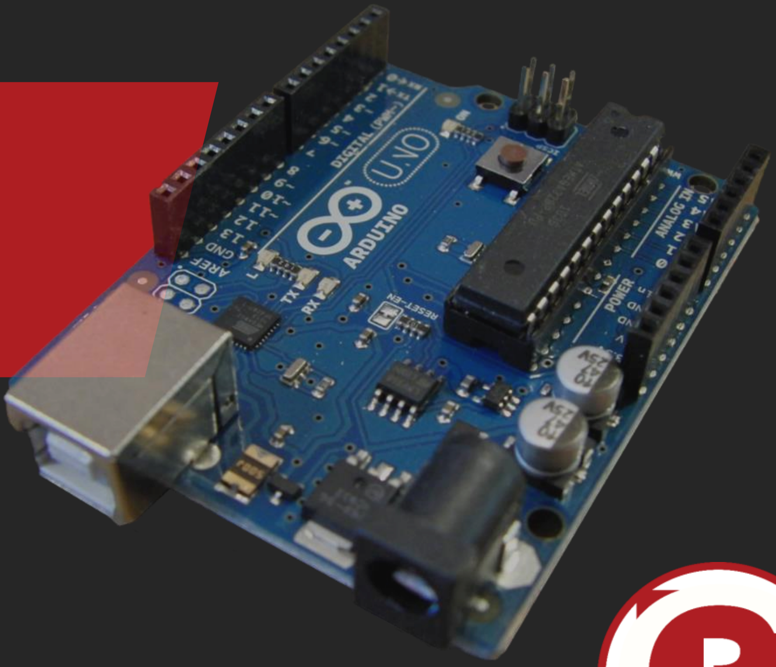
CRAFT #4

Mobile Robotics Programming

Introduction to Arduino

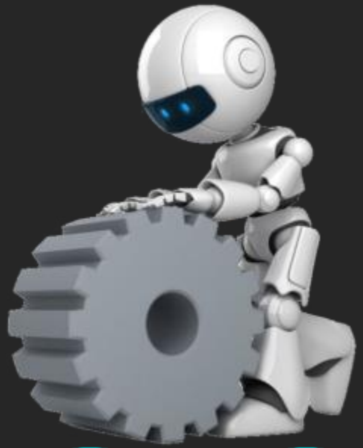


Micael S. Couceiro micael@ingeniarius.pt
Beril Yalcinkaya beril@ingeniarius.pt

15/07/2024

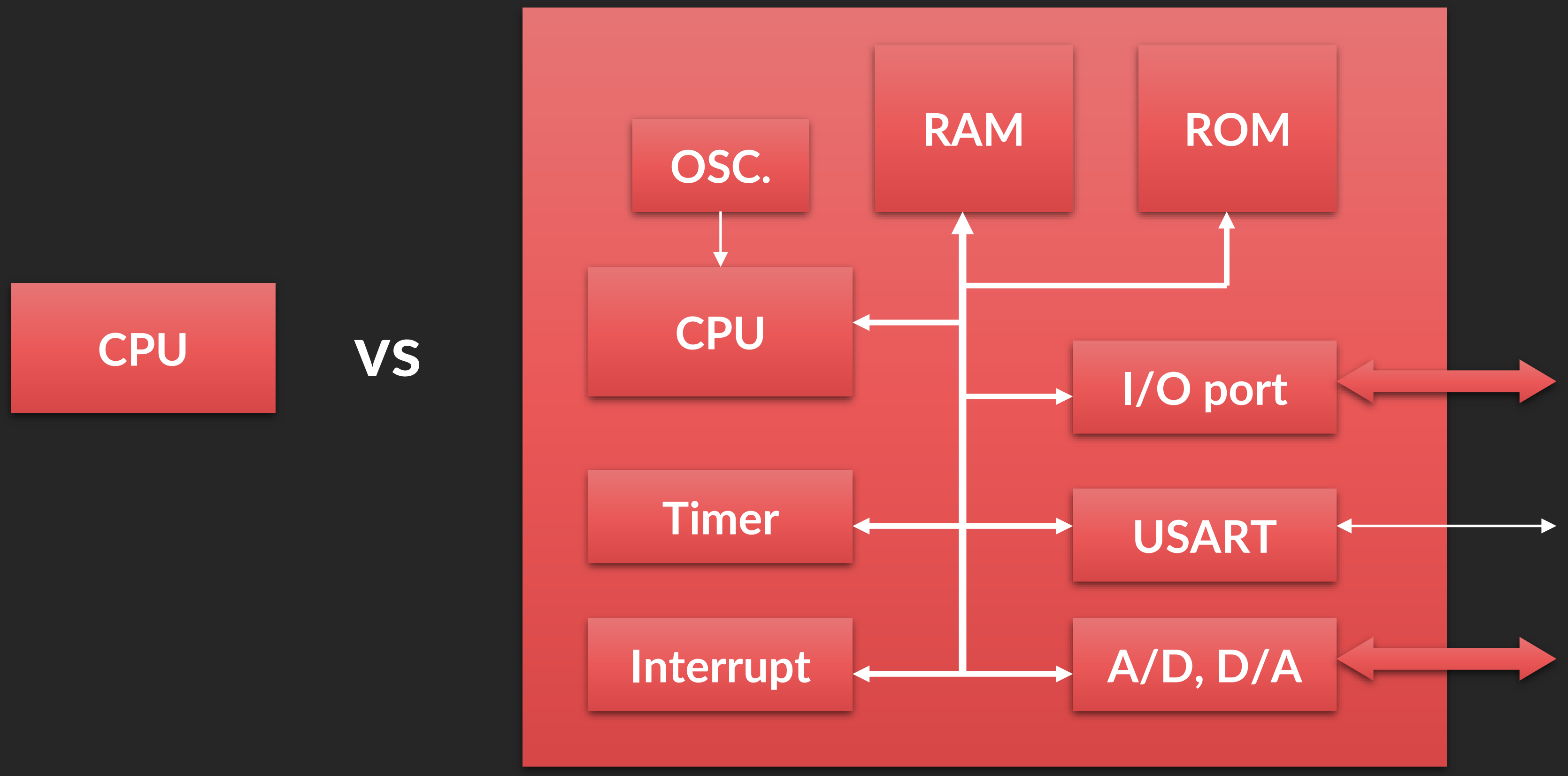
CONTENT

- µP vs µC
- Low-Level and High-Level Languages
- Why Arduino?
- Arduino Board - Hardware
- Arduino IDE
- Arduino Software
- Exercises

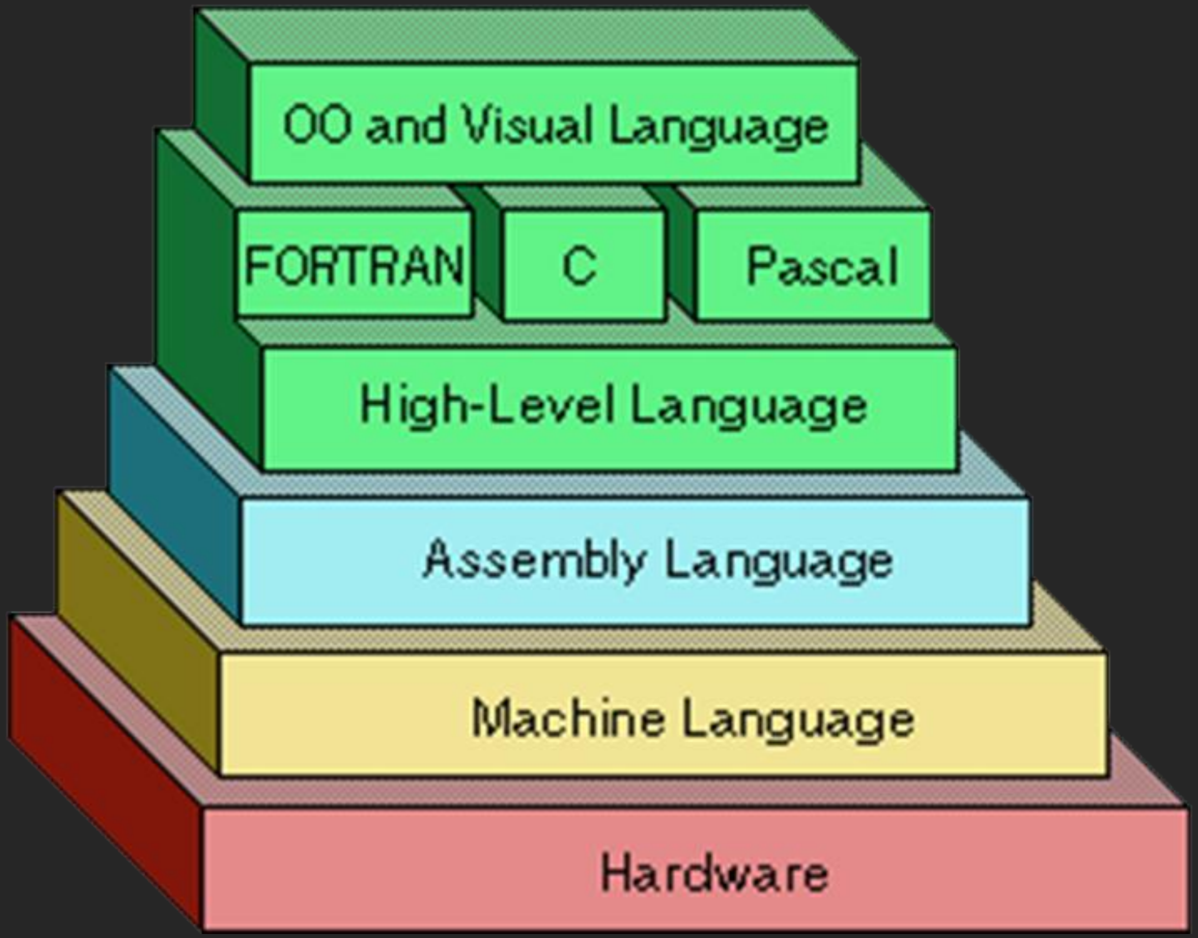


ARDUINO

µP vs µC



LOW-LEVEL AND HIGH-LEVEL LANGUAGES



LOW-LEVEL AND HIGH-LEVEL LANGUAGES

Low-Level

→Allows to generate **more compact** and **faster** code

→All the particular characteristics of µC can be manipulated

→The size of the source file is much bigger

→The program is less readable

→The development of the program is more complex, taking longer to develop and being more prone to errors

High-Level

→The size of the source file is **smaller**

→The program **is more readable** because each instruction usually has a discernible meaning to a human reader

→The development of the program is **simpler and faster**, being this development **less prone to errors**

→Generates a less compact and slower code

→The access may not be available to certain specific features of the µC

WHY ARDUINO?

- Quick and efficient way to develop new projects instead of…just talking about them (or asking an expert to do so)!

- New versions of Arduino always aim to achieve ever-improving performances when compared to the previous ones.

- Allows simple coding in C/C++ language, overcoming the inherent questions of the low-level programming complexity (e.g., Assembly).

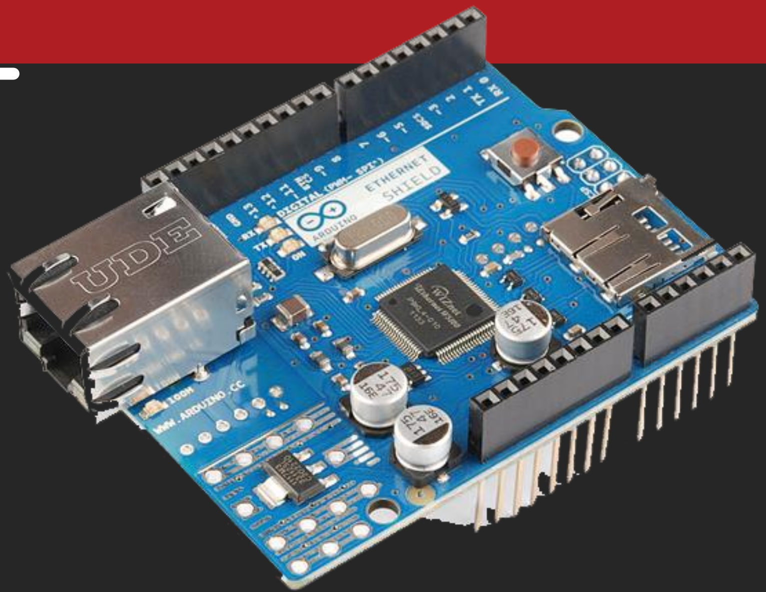
- Open community and free development *software.*

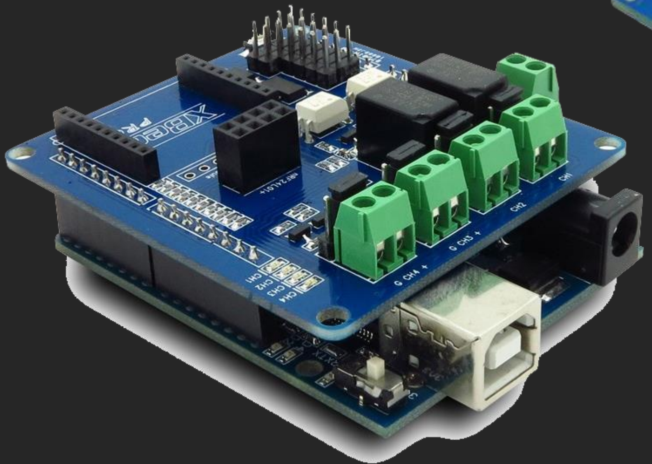
WHY ARDUINO?

- Faster and less favorable prototyping errors - Why developing projects from scratch ? Not worth to reinvent the wheel !
- Arduino’s philosophy is based on reusing other equipment, tools and functions (tinkering) and modular development (patching).
- Arduino boards allows integrating shields that extend the properties and characteristics of the boards (e.g. , power drive, ZigBee communication, etc ...).

SHIELDS

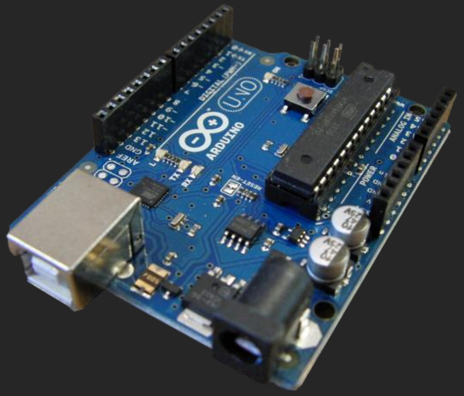
- What are Shields?





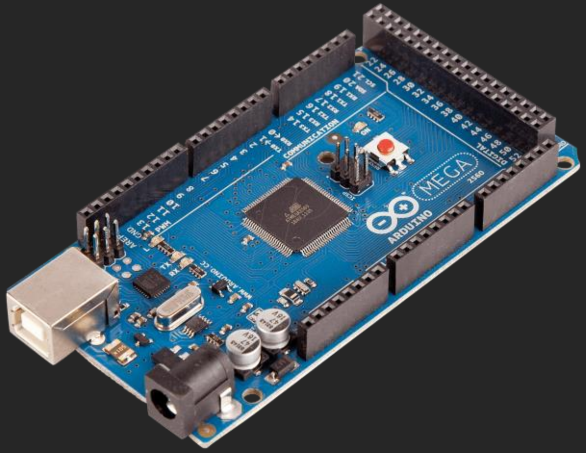
ARDUINO BOARDS

- What are the differences? Some boards:



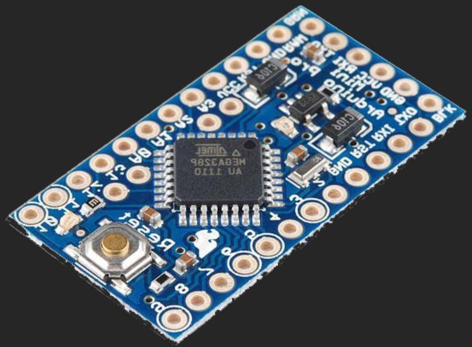
Arduino Uno

- µC: ATmega328P
  16 Mhz 8 bits
- RAM: 2 kbytes
- Memory: 32 kbytes
- Digital I/O: 14
- PWM Pins: 6
- Analog Pins: 6



Arduino Mega

- µC: ATmega2560
  16 Mhz 8 bits
- RAM: 8 kbytes
- Memory: 256 kbytes
- Digital I/O: 54
- PWM Pins: 15
- Analog Pins: 16

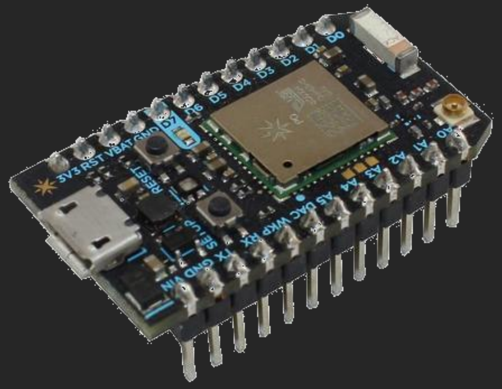


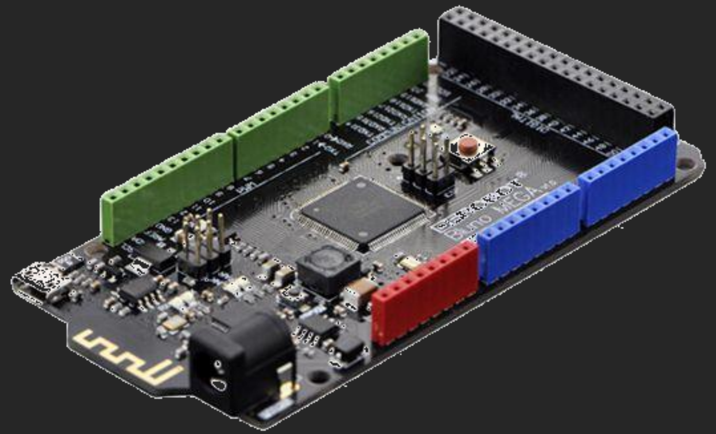
Arduino Mini Pro

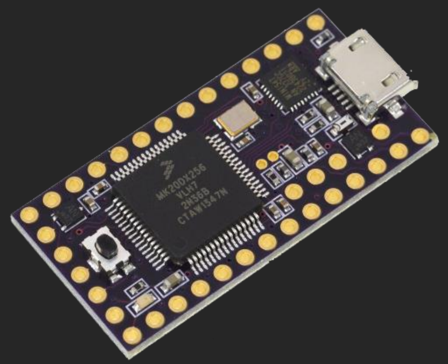
- µC: ATmega328
  8 Mhz 8 bits
- RAM: 2 kbytes
- Memory: 32 kbytes
- Digital I/O: 14
- PWM Pins: 6
- Analog Pins: 6

ARDUINO BOARDS

- Arduino-based non-Arduino Boards







Particle Photon

- STM32F205 120Mhz ARM Cortex M3
- 1MB Flash
- 128KB RAM
- 18 mixed-signal GPIO and advanced peripherals
- Cypress BCM43362 Wi-Fi chip

Bluno Mega

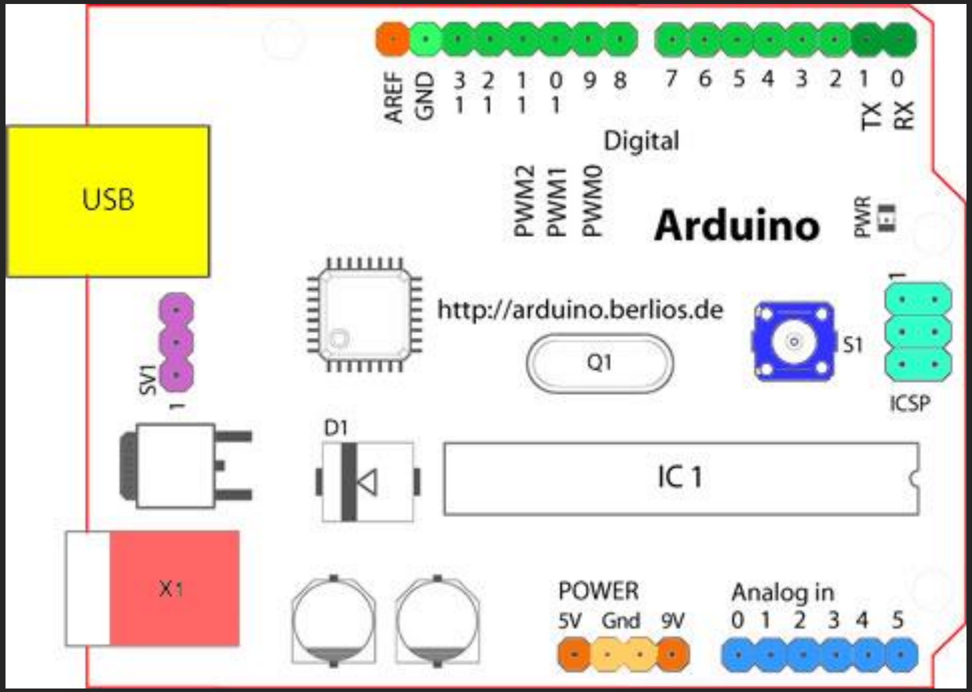
- As Arduino Mega but with Bluetooth 4.0

Teensy 3.2

- Supported on the Arduino IDE
- μC: Cortex-M4 72 Mhz 32 bits
- RAM: 64 Kbytes
- Memory: 256 Kbytes

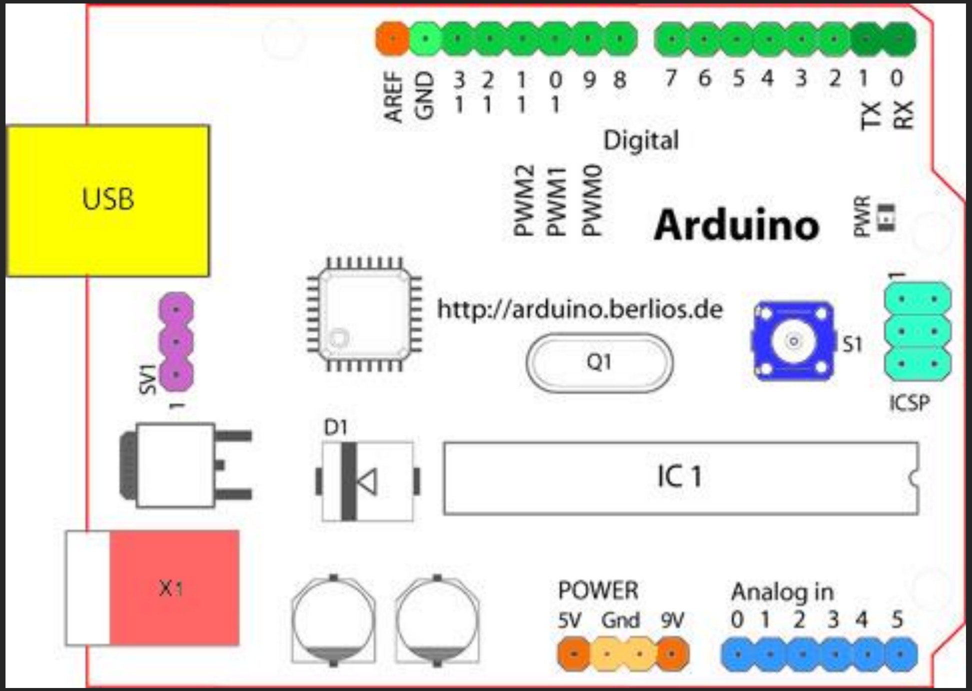
ARDUINO UNO BOARD

Starting clockwise from the top center:



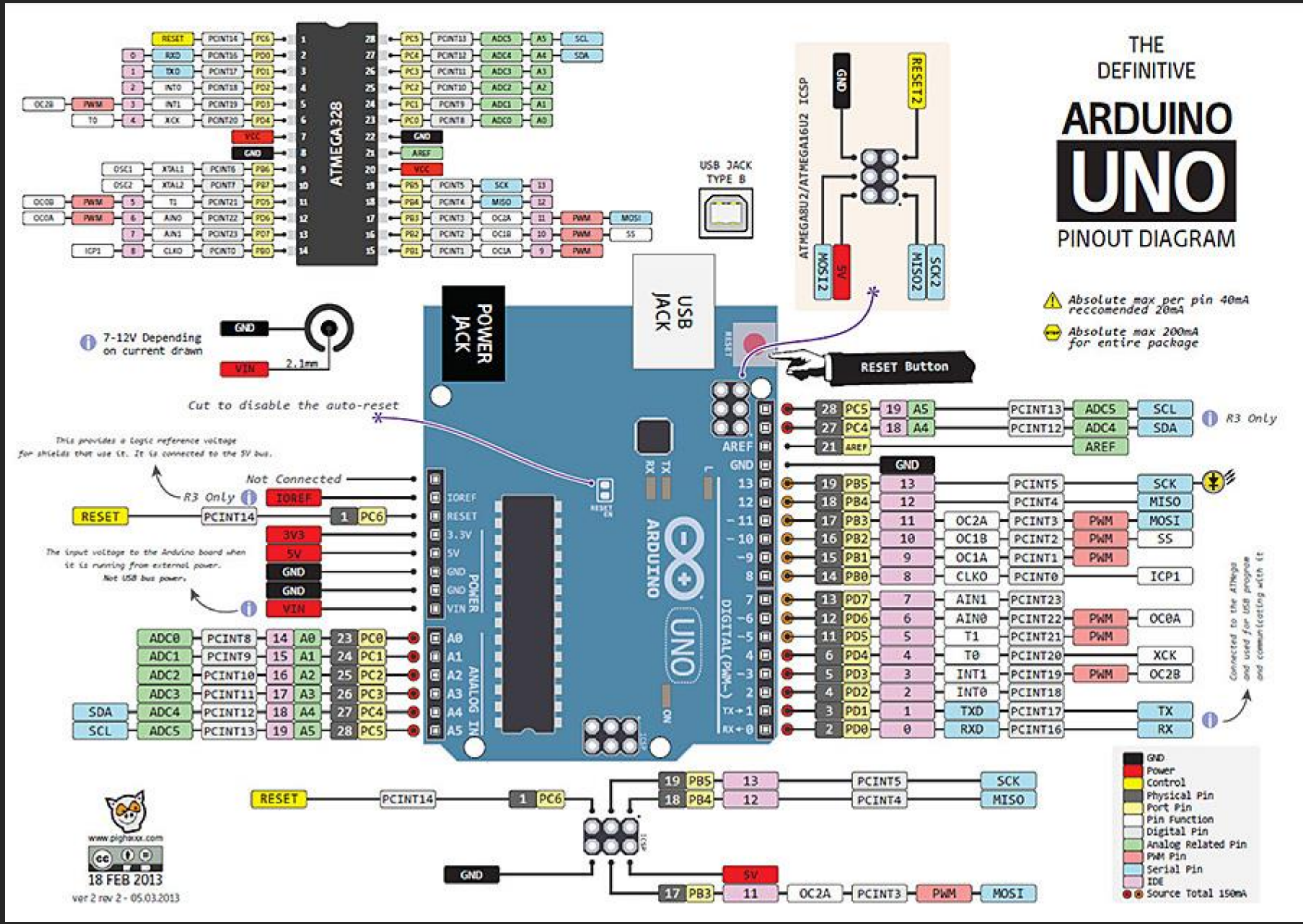
- Analog Reference pin
- Digital Ground
- Digital Pins 2-13
- Digital Pins 0-1/Serial In/Out - TX/RX - These pins cannot be used for digital i/o (digitalRead and digitalWrite) if you are also using serial communication (e.g. Serial.begin).
- Reset Button - S1

ARDUINO UNO BOARD



- In-circuit Serial Programmer
- Analog In Pins 0-5
- Power
- Ground Pins
- External Power Supply In (9-12VDC) - X1 *(pink)*
- Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) *(yellow)*

ARDUINO UNO BOARD



ARDUINO UNO BOARD

Digital Pins

-The digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands.

Serial: 0 (RX) and 1 (TX).

-Used to receive (RX) and transmit (TX) TTL serial data.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

ARDUINO UNO BOARD

Digital Pins

•**PWM:** 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.

•**SPI:** 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.

• **I2C:** Communication Protocol. It can communicate with several devices at once.

•**LED:** 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

ARDUINO UNO BOARD

Analog Pins

- The analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function.
- Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19.
- Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins.

ARDUINO UNO BOARD

Power Pins

.**VIN (sometimes labelled "9V"):** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

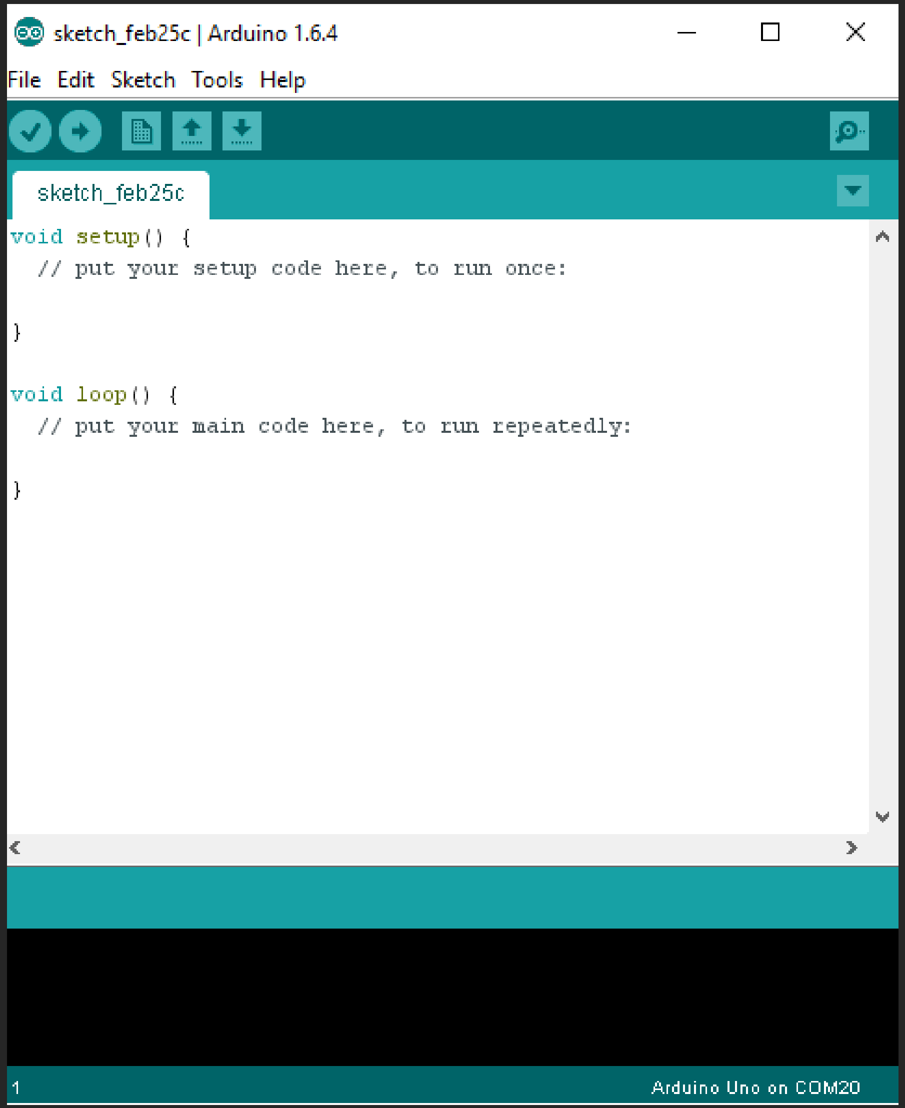
5V: The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.

3V3: A 3.3 volt supply generated by the on-board FTDI chip.

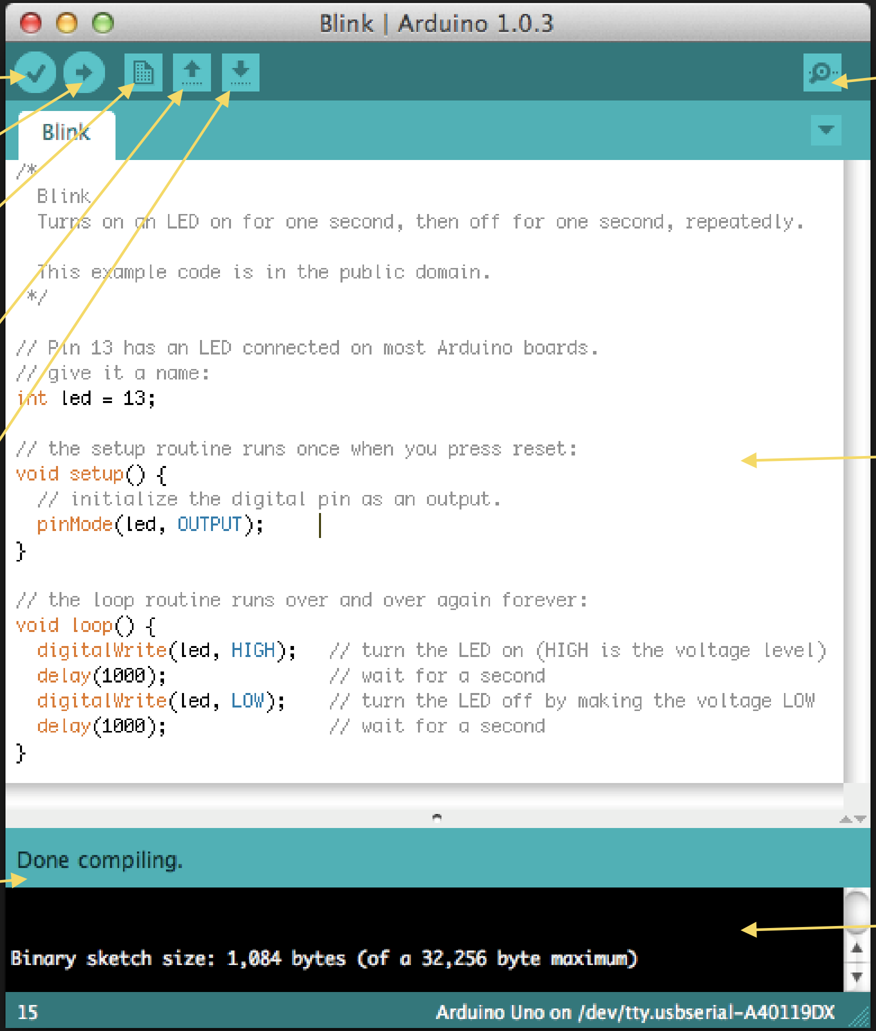
GND: Ground pins.

ARDUINO IDE

- Integrated Development Environment
- Alows you to create, compile, upload and debug your code to Arduino!
- C Language

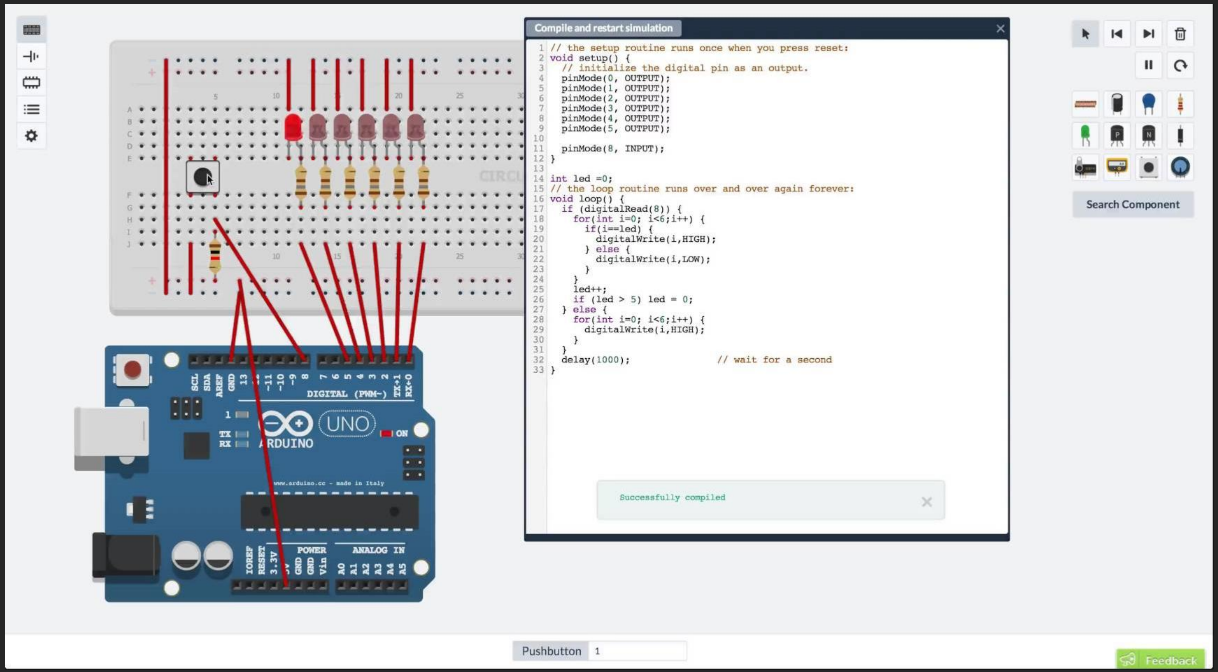


ARDUINO IDE


Blink | Arduino 1.0.3
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
This example code is in the public domain.
*/
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
// the setup routine runs once when you press reset:
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop() {
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
Done compiling.
Binary sketch size: 1,084 bytes (of a 32,256 byte maximum)
15
Arduino Uno on /dev/tty.usbserial-A40119DX
Compile
Upload
New
Open
Save
State
Terminal
Your code!
Console

TINKERCAD

- You can simulate Arduino in your web browser!



So if you don't have a Arduino and components at home, that's no longer na excuse!

https://www.tinkercad.com/#/?type=circuits&collection=designs

SKETCH

- It is the file where you write your code
- There are two fundamental functions: **void setup()** and **void loop()**

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Starts here

And stays in infinite cycle

SKETCH

- **void setup()**

- The setup function runs once when you press reset or power the board

- Pin’s configuration
  <u>Example:</u>
  pinMode(13, OUTPUT)

SKETCH

- **void loop()**

-The loop function runs over and over again forever

-It is the place where you write your code

```
void loop() {
  digitalWrite(13, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(13, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

SKETCH

- But first you need to declare all your constants and variables!

```
// these constants won't change:
const int ledPin = 13;      // led connected to digital pin 13
const int knockSensor = A0; // the piezo is connected to analog pin 0
const int threshold = 100;  // threshold value to decide when the detected sound is a knock or not


// these variables will change:
int sensorReading = 0;      // variable to store the value read from the sensor pin
int ledState = LOW;         // variable used to store the last LED status, to toggle the light

void setup() {
```

VARIABLES

```
int n;
int m = 5;
int t = 10;

void setup() {
  int a = 2;

  n = a;

  n = (n * m) - t;
}

void loop() {
  // put your main code here, to run repeatedly:


}
```

- Several types: “char”, “int”, “float”, “double”, etc
  - char - characters (8 bits)
  - int – integer numbers
  - float – numbers with decimal places
  - double –equal to float but with more decimal places
  - Store values;
- In C++ they have to be declared;
- Have their own “scope”
  - Local and global variables

FUNCTIONS

- The code occurs always inside functions.
  - “{“ and “}” are used to indicate the beginning and the end of the function.

  - Functions have names:
    - “SUM” is different of “Sum”

  - They can be called to execute sub-routines

  - They can return values and accept parameters

HOW TO USE FUNCTIONS?

```
void setup() {
  // put your setup code here, to run once:
  int valor = Soma_dois(5, 10);
}

void loop() {
  // put your main code here, to run repeatedly:

}


int Soma_dois(int _a, int _b){
  int soma = _a + _b;
  return soma;
}
```

Ex: “Soma_dois” was created- Returns an integer. Receives 2 integers.

This functions adds the 2 values that are received and returns the output.

It is called inside the setup where it stores the output in a variable

“IF”, “CASE”, “FOR”, “WHILE”, “DO WHILE”

Controle the code flow;

- Based on conditions (Ex: value < 5);
    - Use comparations
        - == equal (atention, == is for comparation and 1 = is atribuition)
        - != not equal
        - < less than
        - > greater than
        - <= less or equal
        - >= greater or equal

IF

```
/* Se valor for igual a 5, executa o que está dentro de {  } */
if (valor == 5) {
  // Do stuff
}
/* Se a condição anterior não se verificou, passa para esta verificação */
else if ( valor < 5) {
   // Do other stuff
}
/* Se nenhuma condição se verificar, executa isto */
else {
  // Do other other stuff
}
```

WHILE

```
/*
    Este ciclo é executado até "n" ser diferente de 1. Neste caso dura 5 ciclos,
até "valor" dar 5, ai o if é executado e muda "n" para 1
*/
int n = 0;
int valor = 0;
while ( n != 1) {

  if ( valor == 5) {
    n = 1
  }

  valor ++;
}
```

DO WHILE

```
/*
    Este ciclo é executado até "n" ser diferente de 1. Neste caso dura 5 ciclos,
até "valor" dar 5, ai o if é executado e muda "n" para 1
*/
int n = 0;
int valor = 0;
do{
  if ( valor == 5) {
    n = 1
  }
  valor ++;
}while ( n != 1);
```

FOR

```
/* Este é um ciclo que executa 10 vezes. De 0 até 9*/
for (int i = 0; i < 10; i++) {

}

/* Este é um ciclo que executa 5 vezes. Incrementa de 10 em 10 */
for (int i = 5; i < 50; i+=10) {

}

/* Este é um ciclo que executa 10 vezes. De 9 até 0 */
for (int n = 9; i >= 0; i--) {

}
```

THE BASIC FUNCTIONS

- pinMode(pin, mode);
- digitalWrite(pin, state);
- digitalRead(pin);
- analogRead(pin);
- analogWrite(pin, value);

And in next slides we will use:

- delay(ms);
- Serial.print(string);
- attachInterrupt(pin, event, mode);

There are more functions...

pinMode()

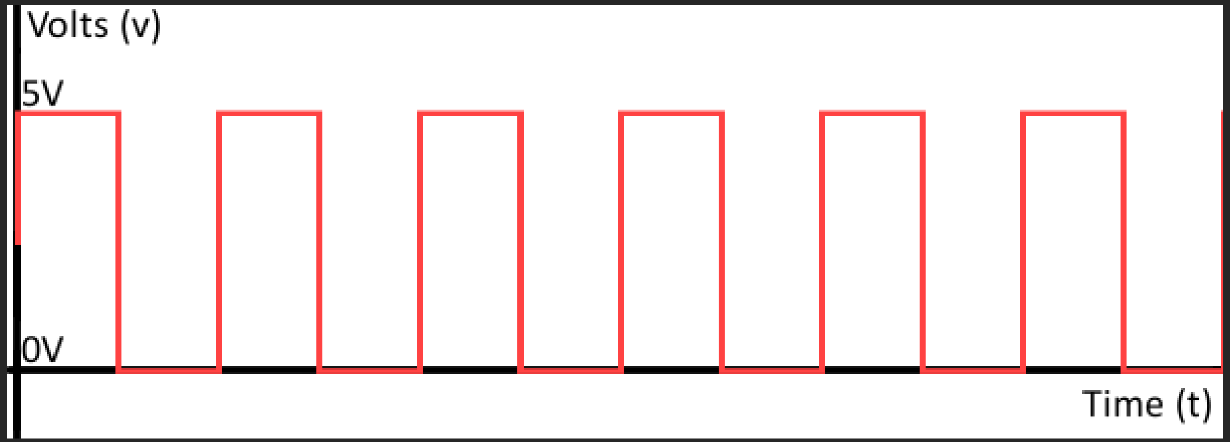
	Analog (A0..A5)	Digital (0..13)
INPUT	Digital Input, Pull-Up **Off**	Digital Input, Pull-Up **Off**
INPUT_PULLUP	Digital Input, Pull-Up **On**	Digital Input, Pull-Up **On**
OUTPUT	Digital Output	Digital Output

Analog Pins can be used as Digital Pins
pinMode(INPUT, Ax) isn't necessary for analogRead()

STATE

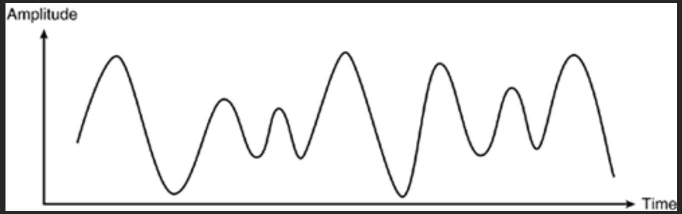
- When we use digtalWrite(pin, state) and state = digitalRead(pin)

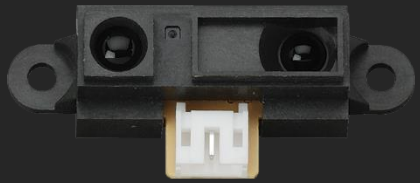
- state can be:
  - LOW ⇔ 0 (0V);
  - HIGH ⇔ 1 (5V);

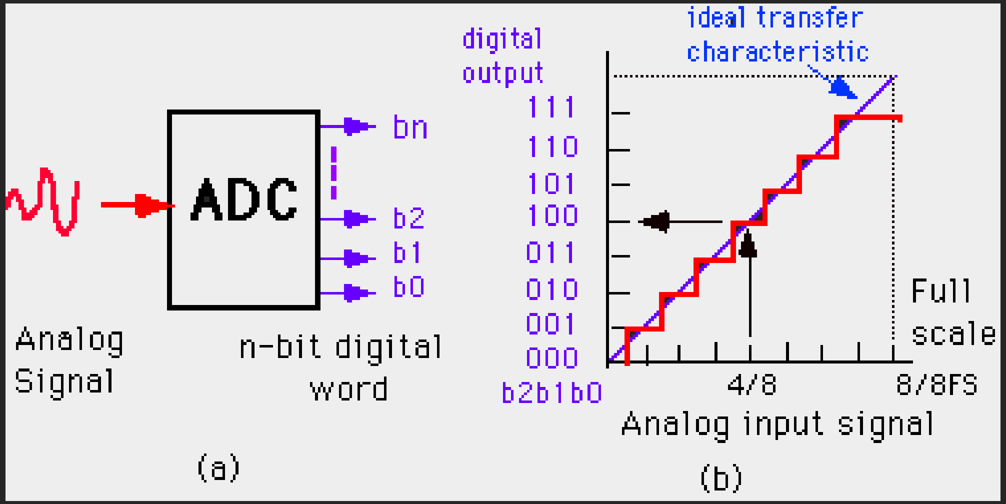


ANALOG INPUT

- One analog signal varies over a range of values
- The Arduino UNO has one 10 bits ADC*
- Can read signals from 0v to 5v
- With a resolution of 10 bits, 2^10, 1024 different values
- So allows a resolution of 4.8mV (5/1024)
- analogRead(pin) returns na int 0 to 1023



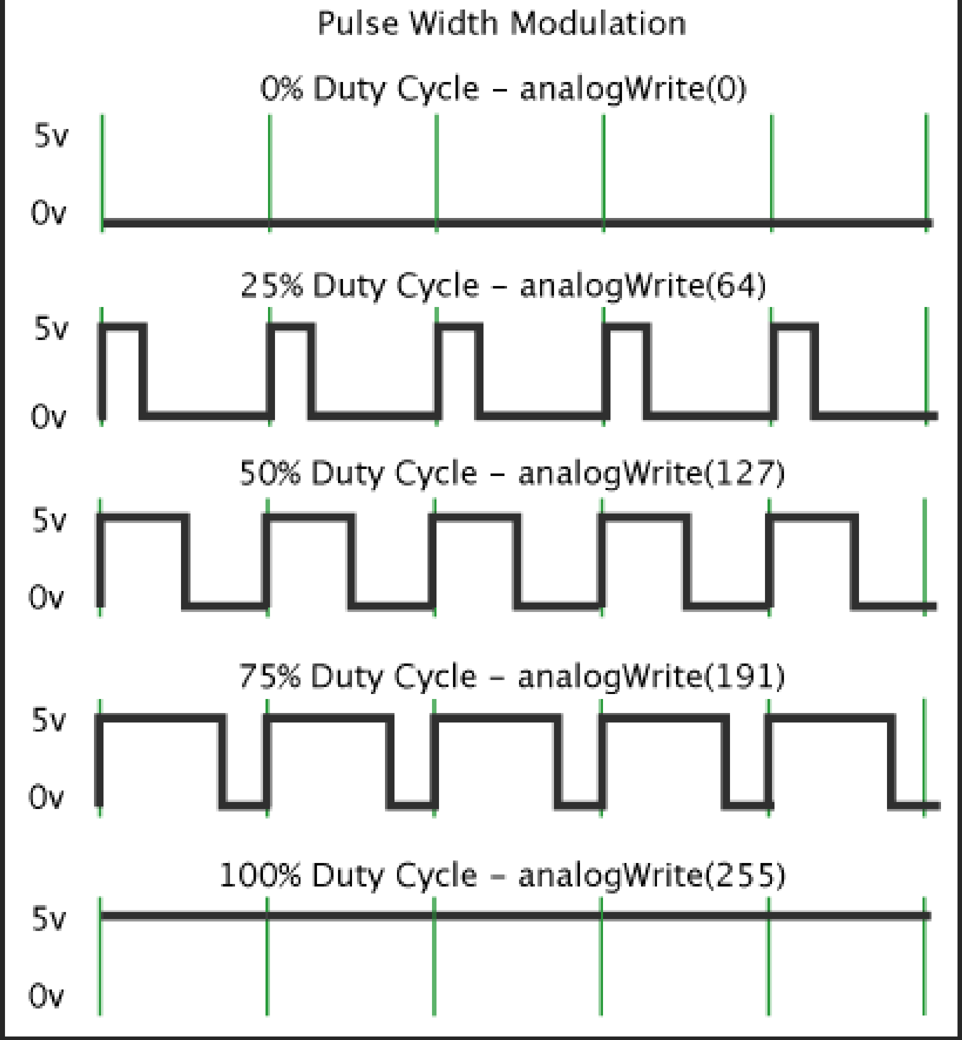




*ADC – Analog to Digital Converter

ANALOG OUTPUT

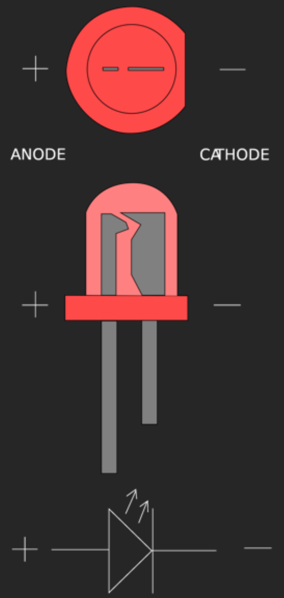
- analogWrite(pin, value);
- PWM - Pulse Width Modulation
- value varies from 0 to 255
- Like Duty Cycle 0% to 100%
- Duty Clycle = Ton*100/Period

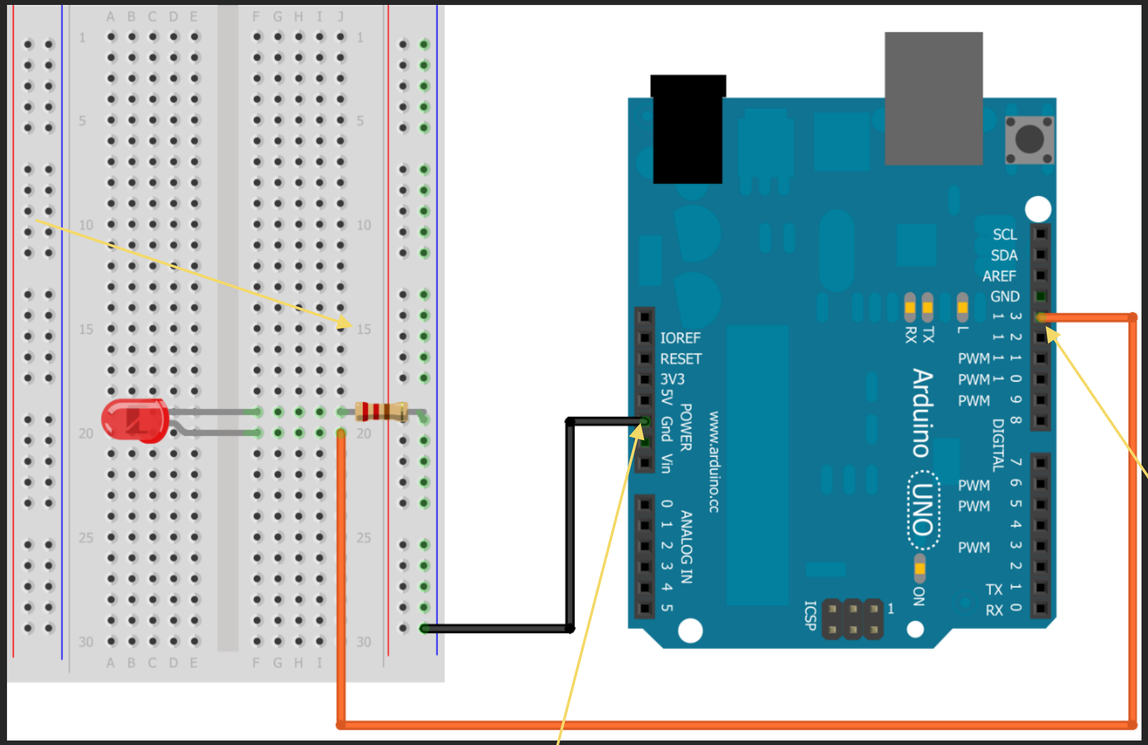


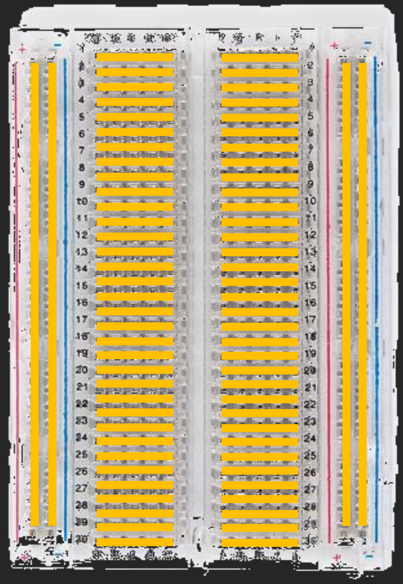
DIGITAL OUTPUT – BLINKING LED

- Let's make the hardware connection

R 220 Ohm







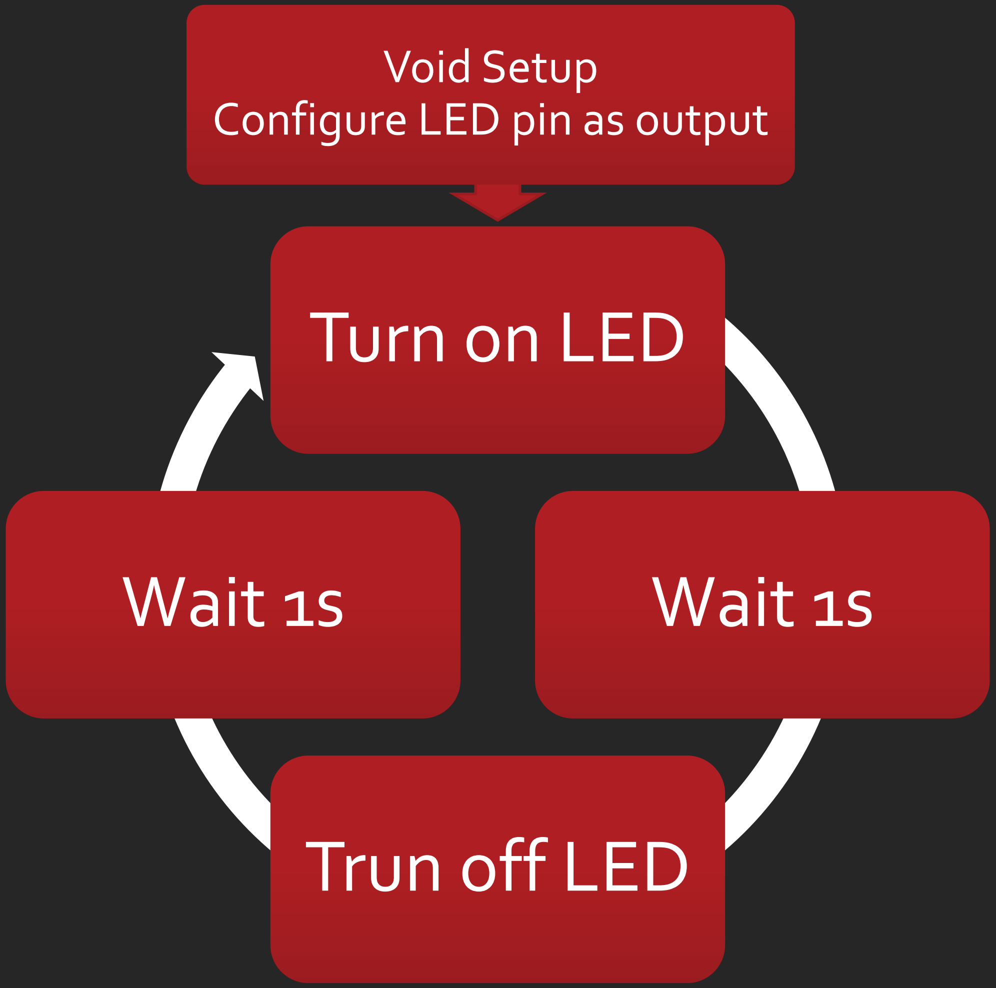
Pin Digital 13

Ground

DIGITAL OUTPUT – BLINKING LED

- Now we create the software

How does it work?



DIGITAL OUTPUT – BLINKING LED

- And here it is:

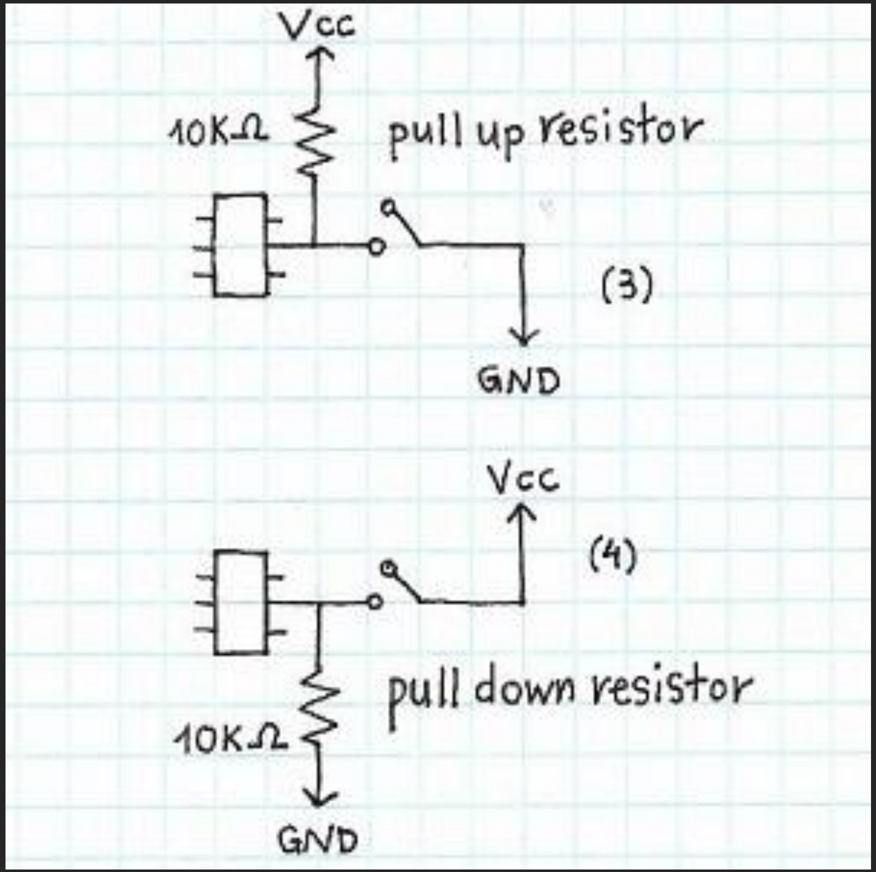
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

DIGITAL OUTPUT – BLINKING LED

- Functions to retain:
  - digitalWrite(pin, state);
  - delay(milliseconds);

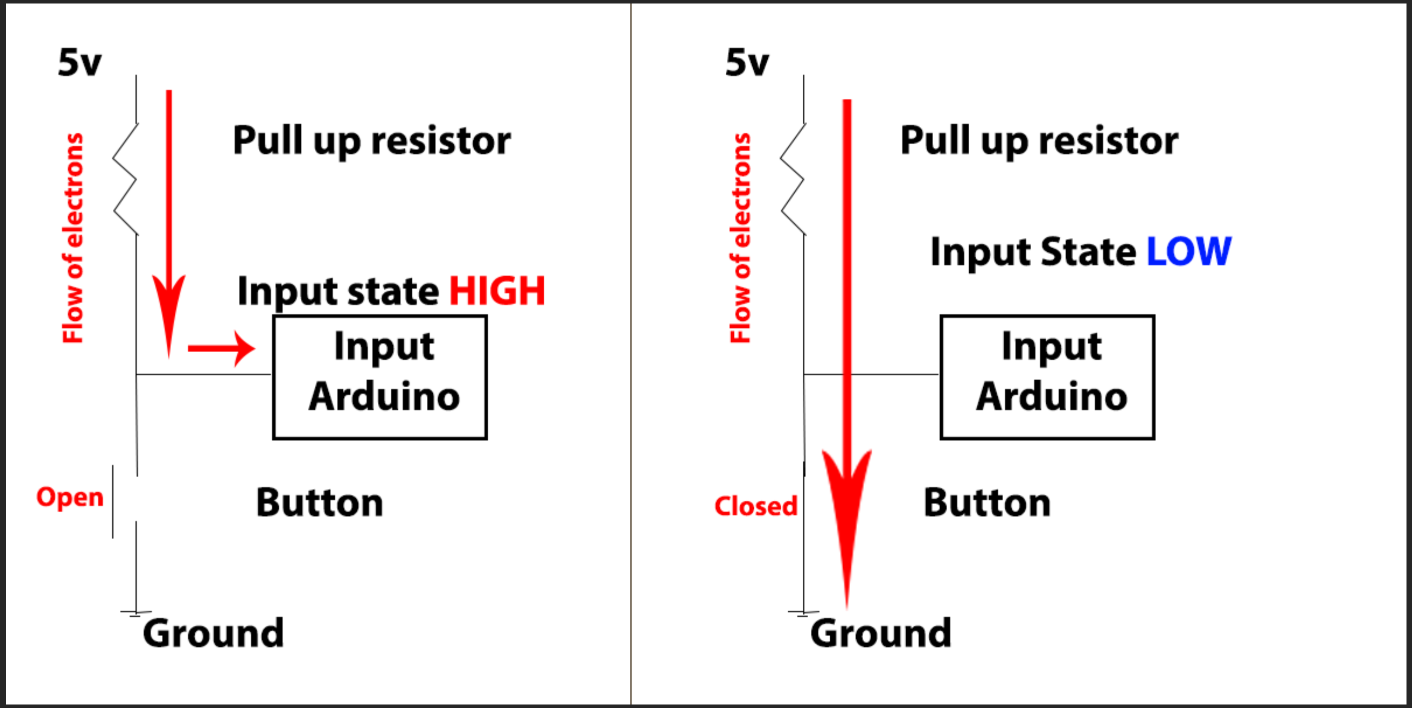
DIGITAL INPUTS – PUSH BUTTON

- First: Pull-up and pull-down



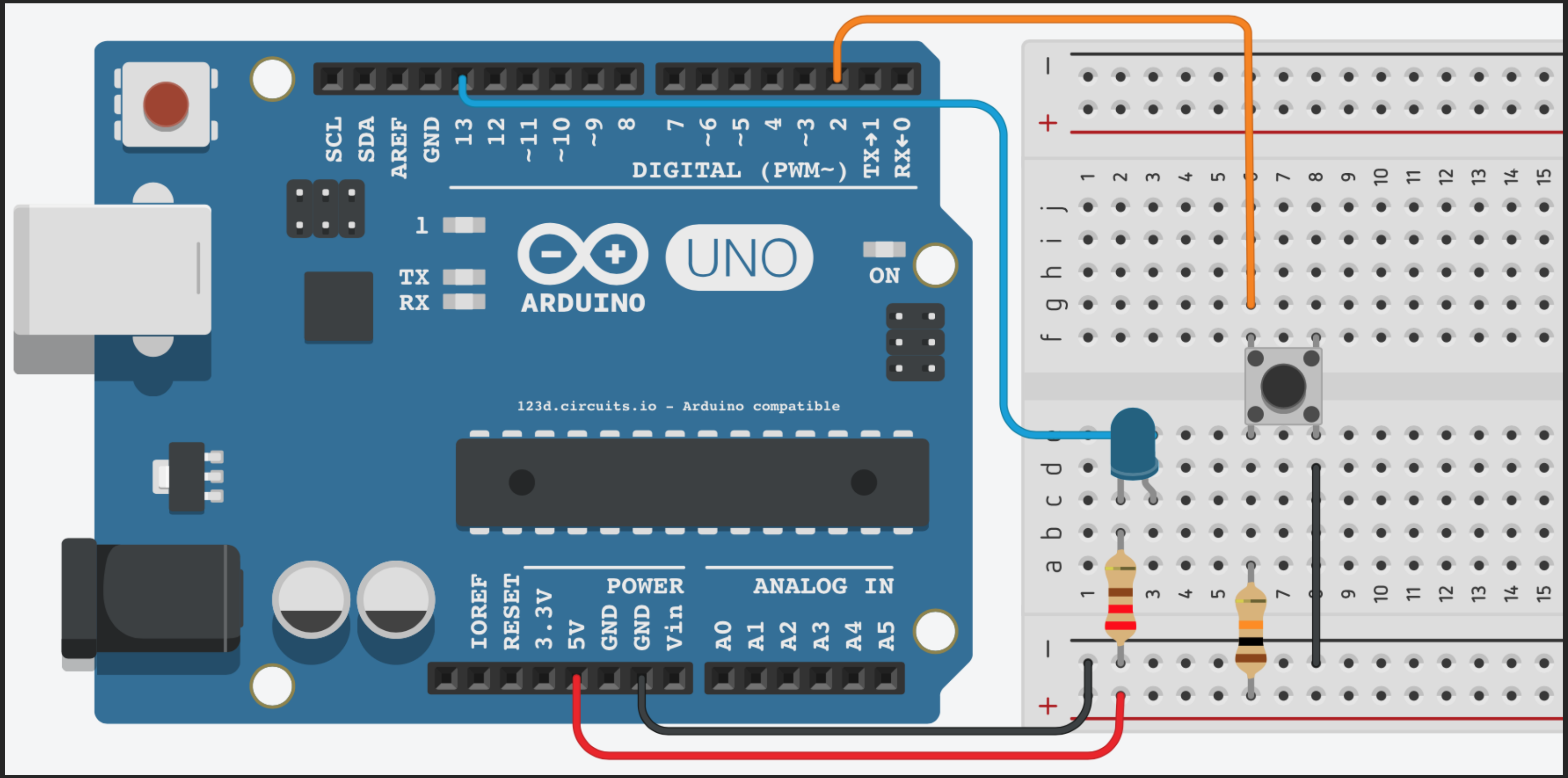
DIGITAL INPUTS – PUSH BUTTON

- Pull-up



DIGITAL INPUTS – PUSH BUTTON

- Let's make the hardware connection



DIGITAL INPUTS – PUSH BUTTON

- And here it is:

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

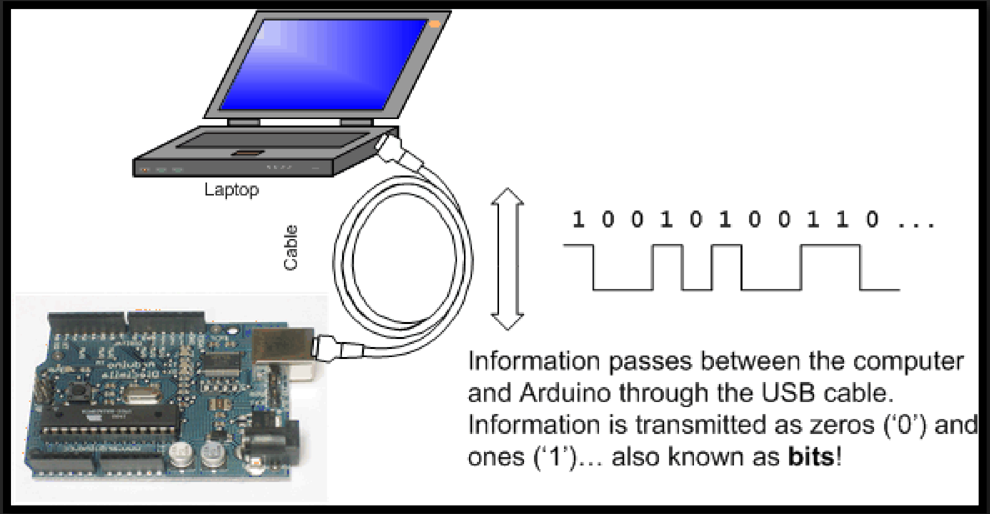
DIGITAL INPUTS – PUSH BUTTON

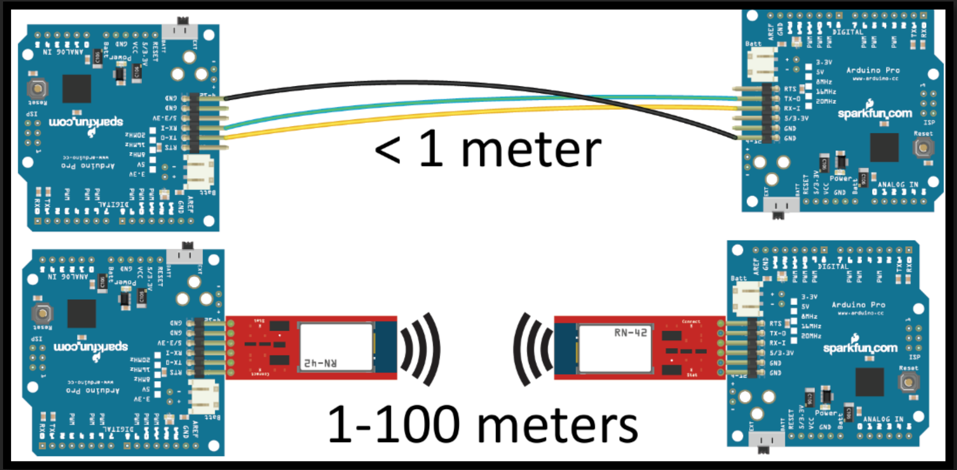
- Functions to retain:
  - digitalRead(pin);

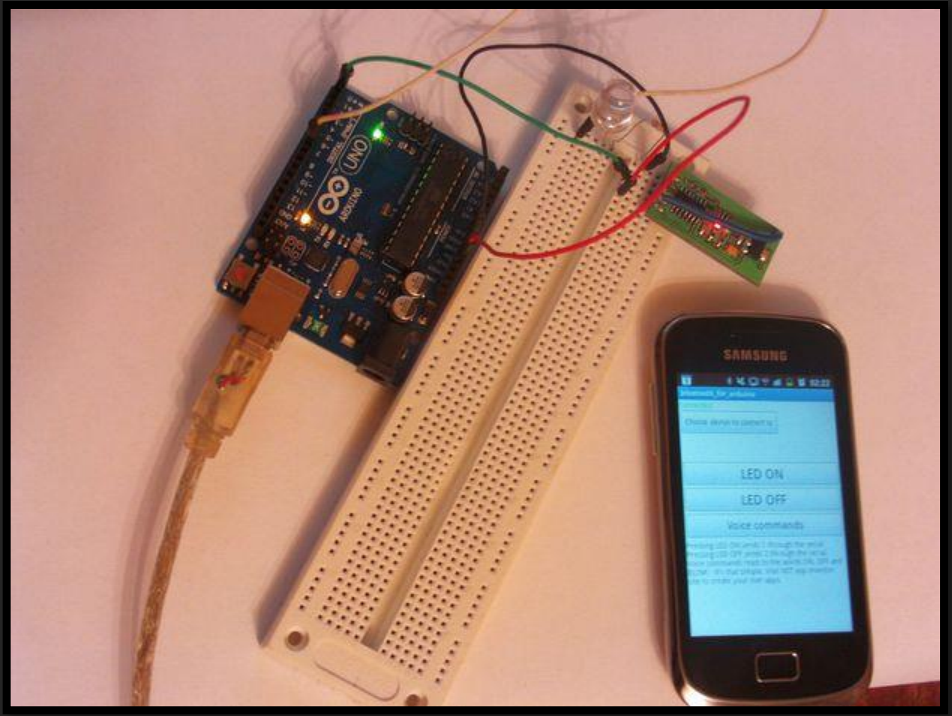
SERIAL COMMUNICATION

- **Serial.begin(baudrate) –** configures the serial to that specific baudrate and inicializes it.
- **Serial.print(string/variable) –** prints the string or the value of the variable
- **Serial.println(string/variable) -** prints the string or the value of the variable but makes a paragraph.

SERIAL COMMUNICATION







SERIAL COMMUNICATION

- An example:

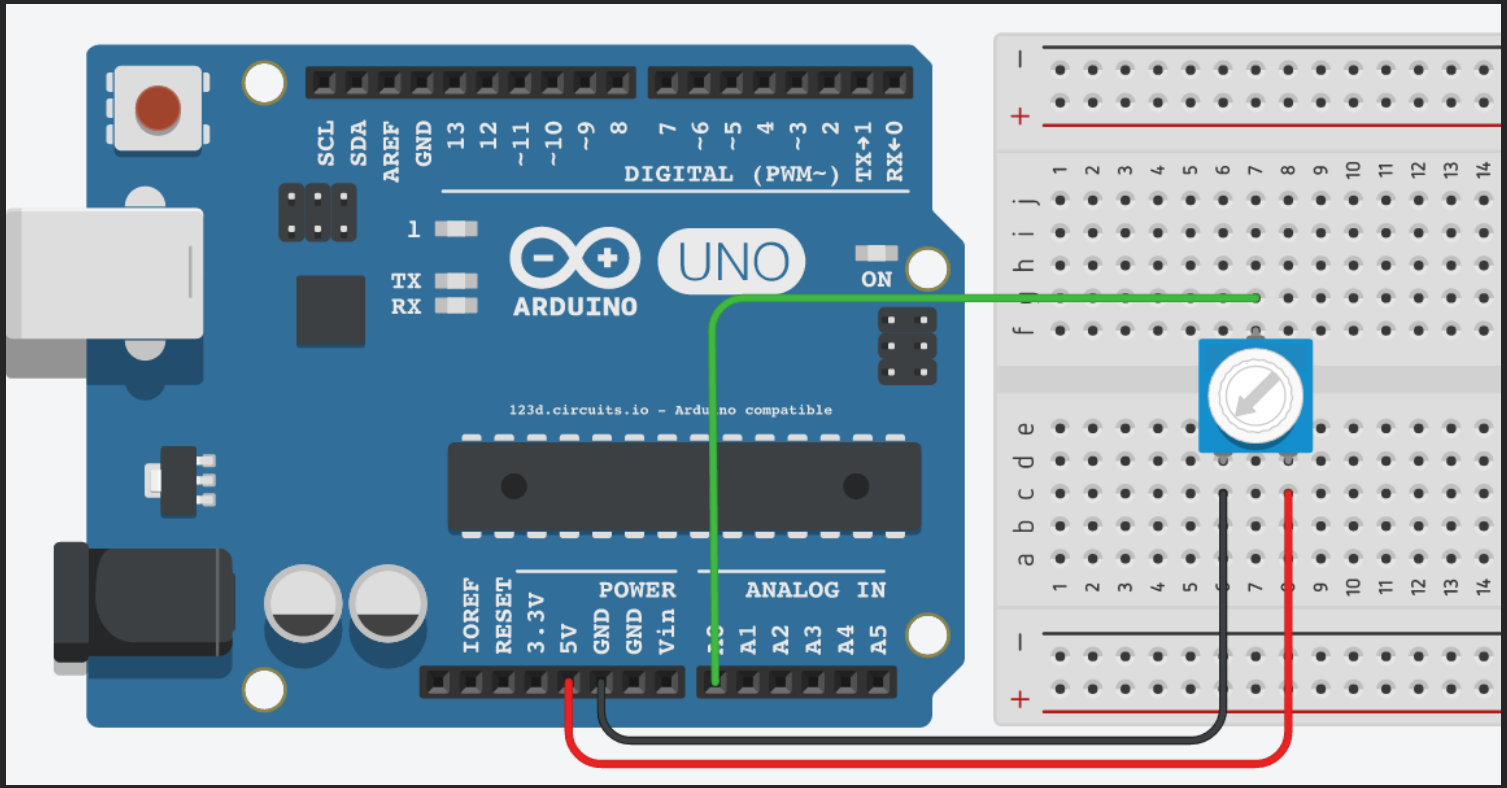
```

char example = 'H';

void setup() {
  //start serial connection
  Serial.begin(9600);
}

void loop() {

  Serial.println(example); // print the value
  delay(10); // delay in between reads for stability

}
```

SERIAL COMMUNICATION

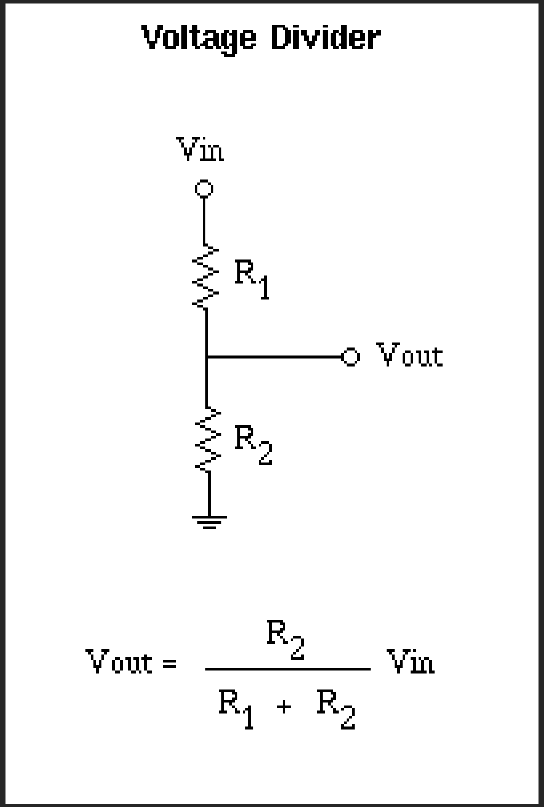
- Functions to retain:
  - Serial.begin(9600);
  - Serial.println(string);

ANALOG INPUT - POTENTIOMETER

- Let's make the hardware connection



- Voltage Divider



ANALOG INPUT - POTENTIOMETER

- And here it is:

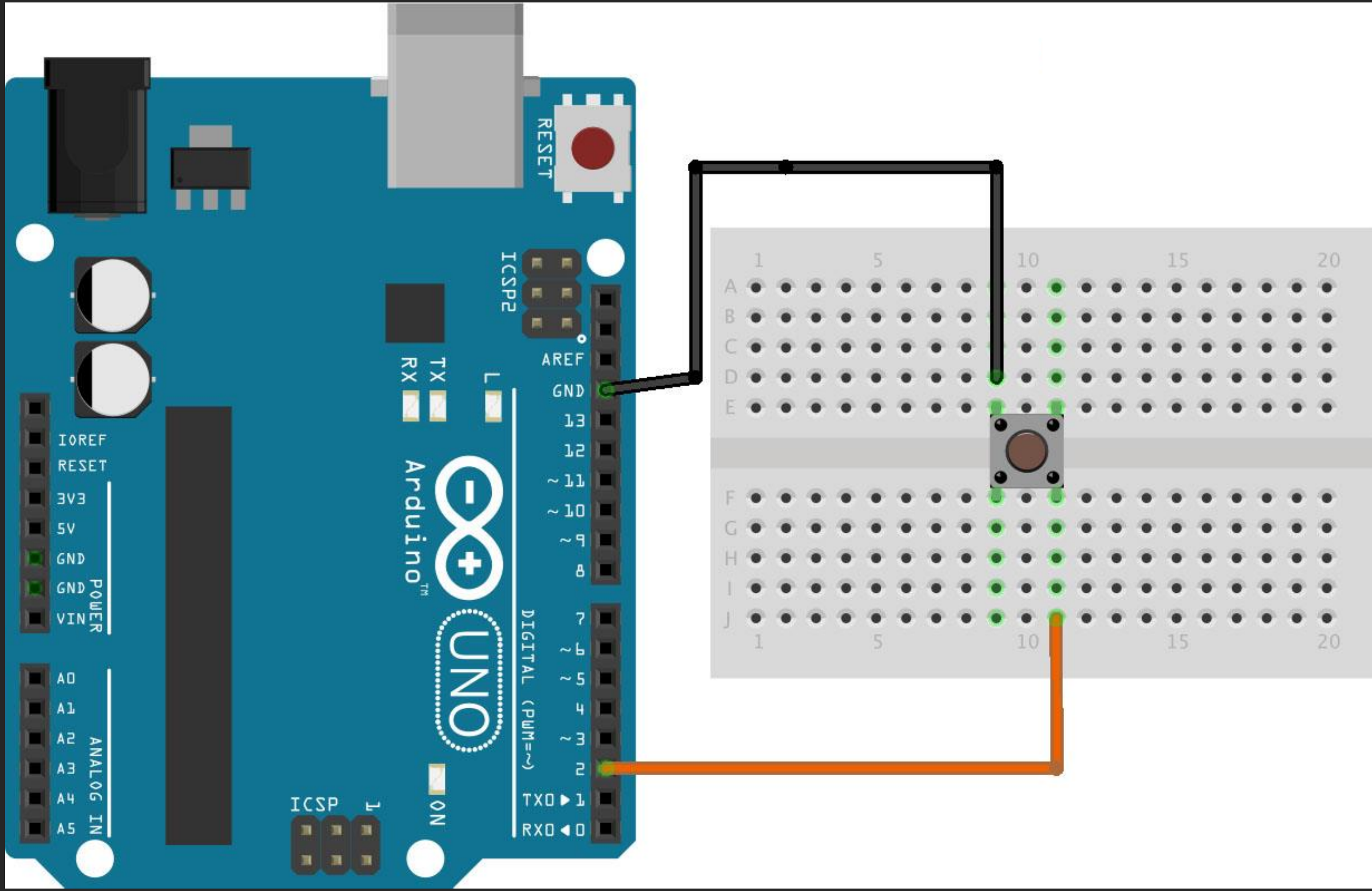
```
void setup() {

  //start serial connection
  Serial.begin(9600);
}

void loop() {

  int value = analogRead(A0);

  Serial.println(value); // print the value
  delay(10); // delay in between reads for stability

}
```

ANALOG INPUT - POTENTIOMETER

- Functions to retain:
  - analogRead(pin);

NON-BLOCKING DELAYS

- Let's make the hardware connection



NON-BLOCKING DELAYS

- And here it is:

```
//Global Variables
const byte BUTTON=2; // our button pin
const byte LED=13; // LED (built-in on Uno)

unsigned long buttonPushedMillis; // when button was released
unsigned long ledTurnedOnAt; // when led was turned on
unsigned long turnOnDelay = 2500; // wait to turn on LED
unsigned long turnOffDelay = 5000; // turn off LED after this time
bool ledReady = false; // flag for when button is let go
bool ledState = false; // for LED is on or not.

void setup() {
 pinMode(BUTTON, INPUT_PULLUP);
 pinMode(LED, OUTPUT);
 digitalWrite(LED, LOW);
}

void loop() {
 // get the time at the start of this loop()
 unsigned long currentMillis = millis();

 // check the button
 if (digitalRead(BUTTON) == LOW) {
  // update the time when button was pushed
  buttonPushedMillis = currentMillis;
  ledReady = true;
 }
```

NON-BLOCKING DELAYS

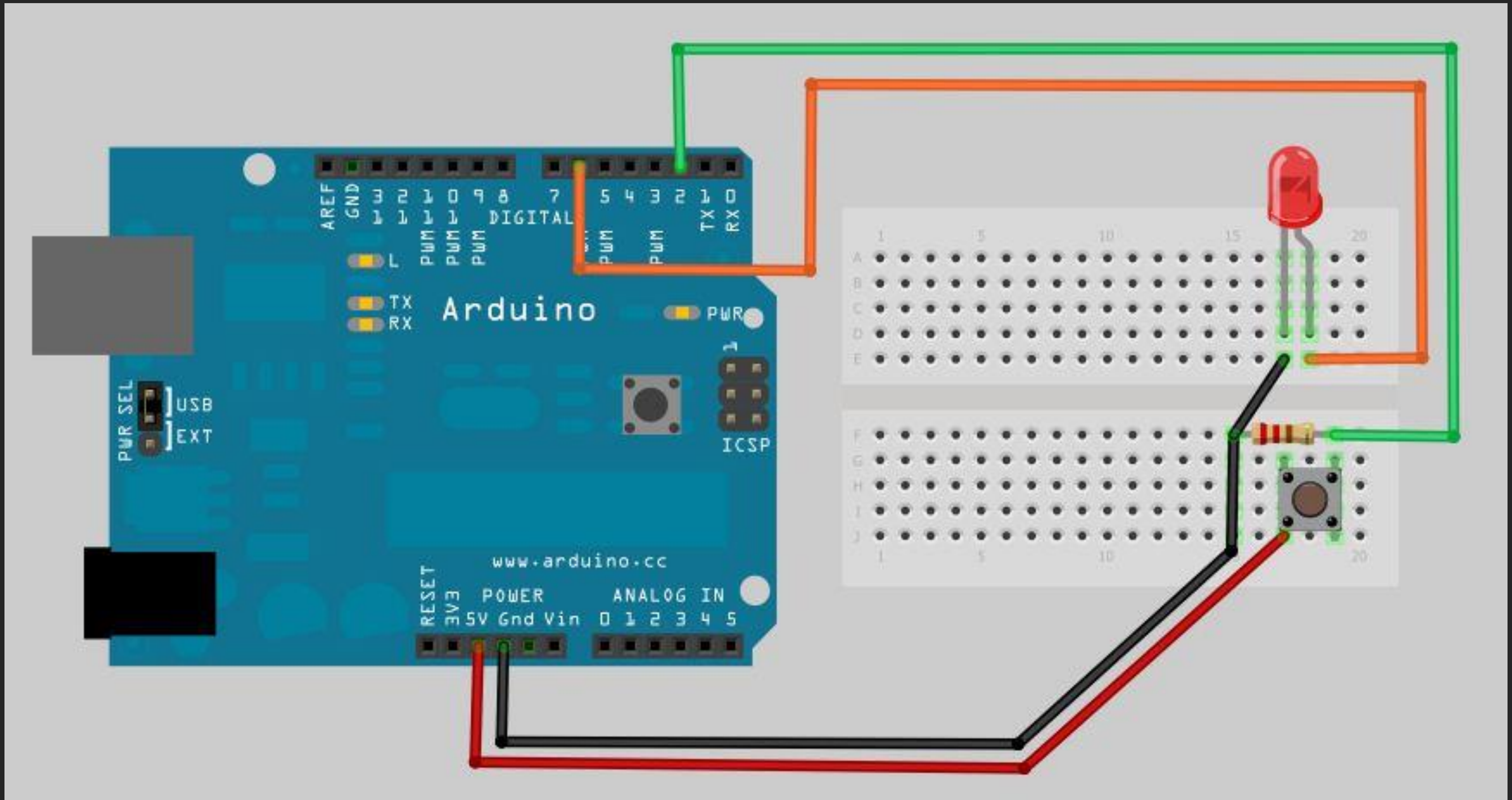
```
  // make sure this code isn't checked until after button has been let go
  if (ledReady) {
    //this is typical millis code here:
    if ((unsigned long)(currentMillis - buttonPushedMillis) >= turnOnDelay) {
      // okay, enough time has passed since the button was let go.
      digitalWrite(LED, HIGH);
      // setup our next "state"
      ledState = true;
      // save when the LED turned on
      ledTurnedOnAt = currentMillis;
      // wait for next button press
      ledReady = false;
    }
  }

  // see if we are watching for the time to turn off LED
  if (ledState) {
    // okay, led on, check for now long
    if ((unsigned long)(currentMillis - ledTurnedOnAt) >= turnOffDelay) {
      ledState = false;
      digitalWrite(LED, LOW);
    }
  }
}
```

NON-BLOCKING DELAYS

- Functions to retain:
  - millis();

EXTERNAL INTERRUPTIONS

- Let's make the hardware connection



EXTERNAL INTERRUPTIONS

- And here it is:

```
int ledPin = 13;
int buttonPin = 2;

int ledToggle;
int previousState = HIGH;
unsigned int previousPress;
volatile int buttonFlag;
int buttonDebounce = 20;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2), button_ISR, CHANGE);
}
```

EXTERNAL INTERRUPTIONS

```
void loop()
{
  if((millis() - previousPress) > buttonDebounce && buttonFlag)
  {
    previousPress = millis();
    if(digitalRead(buttonPin) == LOW && previousState == HIGH)
    {
      ledToggle =! ledToggle;
      digitalWrite(ledPin, ledToggle);
      previousState = LOW;
    }

    else if(digitalRead(buttonPin) == HIGH && previousState == LOW)
    {
      previousState = HIGH;
    }
    buttonFlag = 0;
  }
}
```

```
void button_ISR()
{
  buttonFlag = 1;
}
```

EXTERNAL INTERRUPTIONS

- Functions to retain:
  - attachinterrupt(#attach, function, state);

TEST YOUR SKILLS

- **Exercise #1**

  Pick up on the blink example…

- Change LED’s duty cycle to 75%

- Change the pin to which the LED is connected from pin 13 to pin 2 (Note that both the circuit AND the program must be changed)

- Hook up 8 LEDs to pins 2 through 9 (with resistors, of course.) Modify the code to turn on each one in order and then extinguish them in order

- Now that you have 8 LEDs working, make them turn on and off in a pattern different from the one in exercise 3

TEST YOUR SKILLS

- Exercise #2

  Pick up on the blink example…

1. Add two push buttons as “gas” and “brake” buttons; the “gas” button should speed up the blinking rate of the LED, and the “brake” button should slow it down

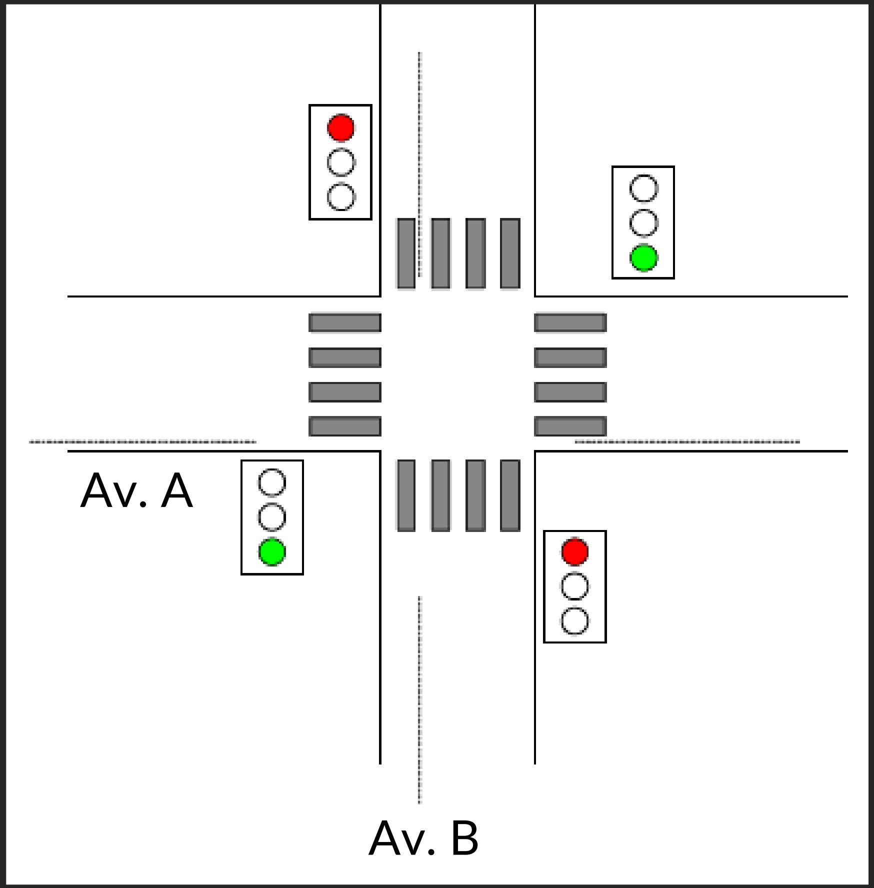
2. Do the same, but using a potentiometer and a button - ONLY when the button is pressed, the LED should blink according to the value of the potentiometer (in other words, you can adjust the potentiometer, but it has no effect until you press the “ON” button)

3. Find out how long it is between the last statement in the loop() function and the first one.

TEST YOUR SKILLS

- Exercise #3

Implement a way to control road traffic lights in a given road with Arduino. At the crossroads of A and B Avenues it is intended to send correct signals in order to control the traffic lights and ensure safety of pedestrians.



TEST YOUR SKILLS

- Exercise #3

Daytime operation:

	Time(s)	60s	1s	30s	1s
Avenue A	Red				
	Yellow				
	Green				
Avenue B	Red				
	Yellow				
	Green				
Pedestrian Avenue A	Red				
	Green				
Pedestrian Avenue B	Red				
	Green				

Night-time operation:

The yellow traffic lights should be flashing, with a period of 2 seconds. And the duty cycle should be 50%. There is a clock circuit external to the Arduino that generates a signal of 5V in the period from 7:00 a.m. to 10:00 p.m. and 0V in the remainder of the day.

TEST YOUR SKILLS

- ## Exercise #4

Let's build a calculator!

1. Communication protocol: Send a number (one digit integer number), followed by the operator (+, -, *, /), and another number (one digit integer number). The Arduino will reply with the result of the operation on the two numbers. Note that the division may look strange - it is an integer division and, therefore, there will not be anything after the decimal point

2. Notes:
   a) To receive numbers, create a function called waitForNum()
   b) Use Serial.available() to know when there is some data in the buffer and Serial.read() to read one character at a time
   c) Remember that what you receive from Serial.read() is the ASCII code of the character introduced and, therefore, you should convert the return into a number. A simple way is to subtract that function return to '0' (zero). Why? Because, fortunately, all the numbers are in sequence in the ASCII table, so you can simply subtract the decimal value of '0' from whatever you read in, and you'll be left with the number itself, e.g., '5' - '0' = 5 .

#3 TASK

Test your Arduino skills directly in your robot:

1. Create a function, let us call it sensL(), that reads the left range sensor and converts its measurements in millimetres

2. Do the same as in point 1, but for the right sensR() and front sensF() range sensors

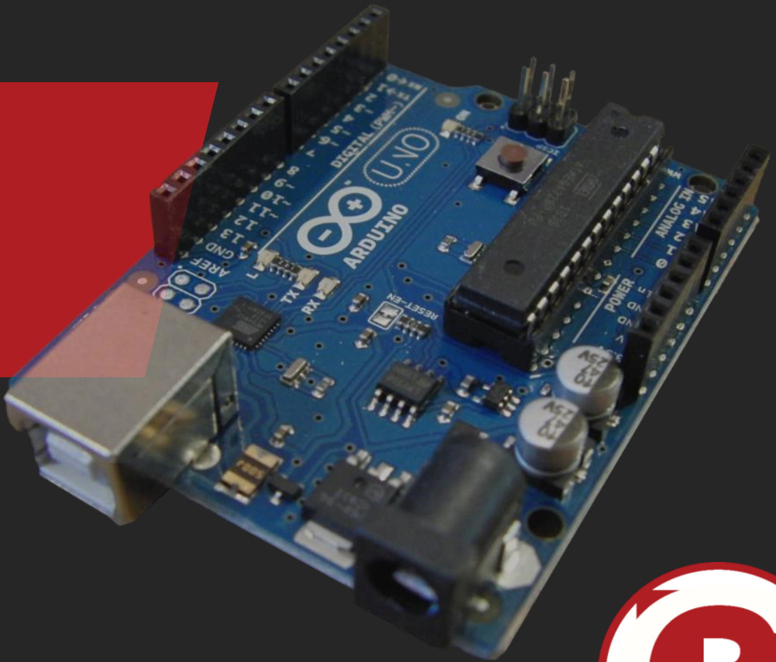
#3 TASK

3. Create a function that reads the number of pulses counted by the encoders on each wheel since last request
    a. To do so, include the library Encoder.h in your Arduino project
    b. Associate the digital pins of external interrupts of each motor's encoder to the Encoder type of variable encL and encR
    c. Create a function that reads the absolute values of encoders
    d. Create a function that is called periodically (you may use the millis() function or, if you feel wild, a timer interrupt) and returns the difference since last request – a frequency of 10Hz should be ensured

- **This task should be delivered by the 21st July, 23h59**

CRAFT #4

Thank you



Micael S. Couceiro micael@ingeniarius.pt
Beril Yalcinkaya beril@ingeniarius.pt

15/07/2024