



Co-funded by the  
Erasmus+ Programme  
of the European Union



## Erasmus+ KA210-VET

### Small-scale partnerships in vocational education and training

**Project Title: “Using Arduinos in Vocational Training”**

**Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

**\*\*\*\*\* UsingARDinVET GUIDEBOOK \*\*\*\*\***





Co-funded by the  
Erasmus+ Programme  
of the European Union



**“This project is Funded by the Erasmus+ Program of the European Union.  
However, European Commission cannot be held responsible for any use which may be  
made of the information contained therein”**



Co-funded by the  
Erasmus+ Programme  
of the European Union



***COORDINATOR:***

**LICEUL TEHNOLOGIC ELENA CARAGIANI ( Tecuci, Romania)**

***PARTNERS:***

**Ahi Evran Mesleki Eğitim Merkezi (Ankara, Türkiye)**

**2 EK Peiraia (Piraeus / Greece)**

**Instituto De Educación Secundaria "María Molner" (SEGOVIA, Spain)**

**IPIAS G. GIORGI (Potenza, Italy)**



Co-funded by the  
Erasmus+ Programme  
of the European Union



## Project Summary

### “Using Arduinos in Vocational Training”

#### Objectives:

The best way to learn something is by doing and experiencing it. The most important education-training materials are the experimental sets in VET.

When the curriculum of VET schools are analyzed, it is seen that it is difficult to give Arduinos education-training with these sets. We have prepared this project to overcome these problems, to provide a more efficient environment and experimental sets for our students in Arduinos lesson, and to ensure that learning is more permanent.

#### Implementation:

\*5 Transnational meetings; 5 TPMs will be held during the two years project. The participants of these meetings are project teams of the partners.

\*Workshop of "Our Project is meeting with VET schools, electronic, ICT industries, labour markets": The results, products, will be presented to the workshop participants.



Co-funded by the  
Erasmus+ Programme  
of the European Union



- Creating project team and tools.
- Project Website.
- Training Kits and Set.
- Using ARDinVET Guidebook.
- Training Videos.
- Project DVD.
- 5 Newsletters.
- Planting Erasmus trees.

**Results:**

- To change students' perception of "Teaching, Learning, Using Arduinos in Vocational Training".
- To decrease absenteeism level in Arduinos' lessons.
- To make teachers learn innovative methodologies about Arduinos.
- To provide better educational services to their students.
- To create positive school climate and improve the learning environment.
- To improve Arduinos' workshops and lessons in a better way at schools.
- Increasing employment of graduates.
- Development of intercultural dialogues.



## CONTENTS:

NO	MODULE NAME	PAGE
1	Introduction to Arduinos	7
2	Arduino Input/Output Module and Training Kit.	23
3	LCD Module and Training KIT	73
4	Keypad Module and Training KIT	96
5	Dot Matrix Display Module and Training KIT	117
6	Motor Module and Training KIT	144
7	Sensor Module And Training KIT	170
8	Annexex	197



Co-funded by the  
Erasmus+ Programme  
of the European Union



# **PROJECT MODULES and KITS of Using ARDin VET**



Co-funded by the  
Erasmus+ Programme  
of the European Union



## **Erasmus+ KA210-VET**

### **Small-scale partnerships in vocational education and training**

**Project Title: “Using Arduinos in Vocational Training”**

**Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

### **INTRODUCTION to ARDUINOS (BASICS of ARDUINOS)**





## INTRODUCING the ARDUINO

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

In other words, the Arduino is a small computer that you can program to read information from the world around you and send commands to the outside world. All of this is possible because you can connect several devices and components to the Arduino to do what you want. You can do amazing projects with it, there is no limit for what you can do, and using your imagination everything is possible!

In simple terms, the Arduino is a tiny computer system that can be programmed with your instructions to interact with various forms of input and output. The current Arduino board model, the Uno, is quite small in size compared to the average human hand.

### What is an Arduino?

The Arduino is the board shown in the figure below.

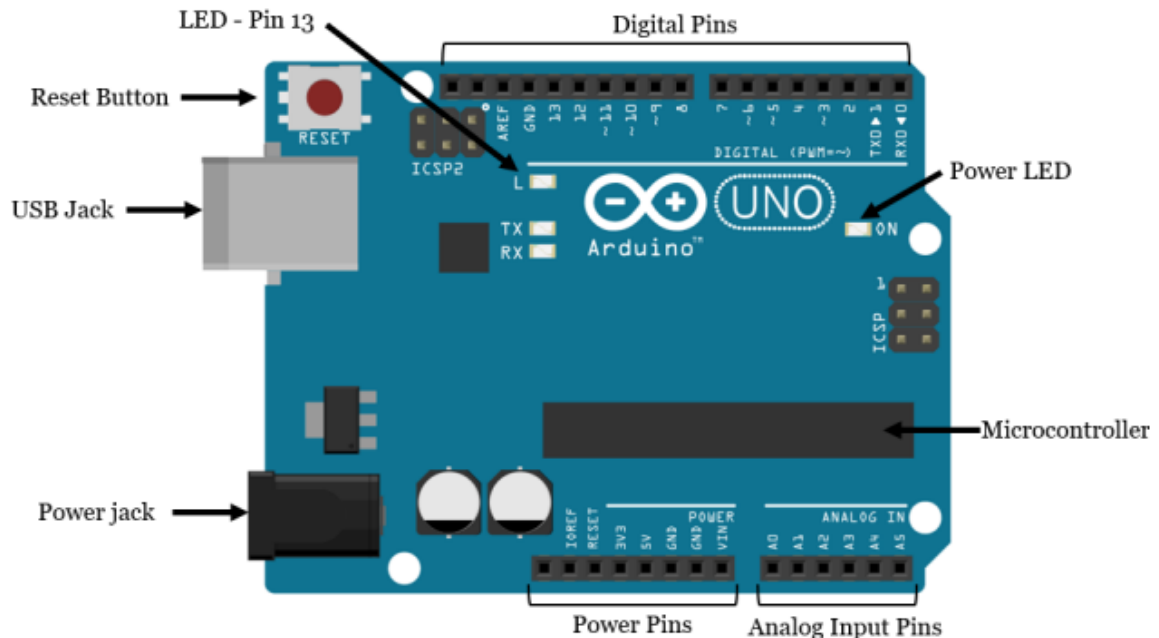


Basically, it is a small development board with a brain (also known as a microcontroller) that you can connect to electrical circuits. This makes it easy to read inputs – read data from the outside – and control outputs - send a command to the outside. The brain of this board (Arduino Uno) is an ATmega328p chip where you can store your programs that will tell your Arduino what to do.



## Exploring the Arduino Uno Board

In the figure below, you can see an Arduino board labeled. Let's see what each part does.



- **Microcontroller:** the ATmega328p is the Arduino brain. Everything on the Arduino board is meant to support this microcontroller. This is where you store your programs to tell the Arduino what to do.
- **Digital pins:** Arduino has 14 digital pins, labeled from 0 to 13 that can act as inputs or outputs.
  - o When set as inputs, these pins can read voltage. They can only read two states: HIGH or LOW.
  - o When set as outputs, these pins can apply voltage. They can only apply 5V (HIGH) or 0V (LOW).
- **PWM pins:** These are digital pins marked with a ~ (pins 11, 10, 9, 6, 5 and 3). PWM stands for “pulse width modulation” and allows the digital pins output “fake” varying amounts of voltage. You’ll learn more about PWM later.
- **TX and RX pins:** digital pins 0 and 1. The T stands for “transmit” and the R for “receive”.

The Arduino uses these pins to communicate with other electronics via Serial. Arduino also uses these pins to communicate with your computer when uploading new code. Avoid using these pins for other tasks other than serial communication, unless you’re running out of pins.
- **LED attached to digital pin 13:** This is useful for an easy debugging of the Arduino sketches.



- **TX and RX LEDs:** these LEDs blink when there is information being sent between the computer and the Arduino.
- **Analog pins:** the analog pins are labeled from A0 to A5 and are often used to read analog sensors. They can read different amounts of voltage between 0 and 5V. Additionally, they can also be used as digital output/input pins like the digital pins.
- **Power pins:** the Arduino provides 3.3V or 5V through these pins. This is really useful since most components require 3.3V or 5V to operate. The pins labelled as “GND” are the ground pins.
- **Reset button:** when you press that button, the program that is currently being run in your Arduino restarts. You also have a Reset pin next to the power pins that acts as reset button. When you apply a small voltage to that pin, it will reset the Arduino.
- **Power ON LED:** will be on since power is applied to the Arduino.
- **USB jack:** you need a male USB A to male USB B cable (shown in figure below) to upload programs from your computer to your Arduino board. This cable also powers your Arduino.



- **Power jack:** you can power the Arduino through the power jack. The recommended input voltage is 7V to 12V. There are several ways to power up your Arduino: for example; rechargeable batteries, disposable batteries, wall-warts and solar panel.

### Arduino Features

Arduino Uno; has Atmel Atmega 328P microcontroller and also has USB connection input, power jack input, reset button.. Arduino has everything that a microcontroller should have.



Microcontroller	Atmega328P
Working voltage	5V
Input voltage (recommended)	7-12V
Input voltage (limit)	6-20V
Digital input / output pins	14
PWM input / output pins	6
Analog input pin	6
Dc current per input / output pin	20mA
DC current for 3.3V	50mA
Flash memory	32 KB
Sram	2KB
EEPROM	1 KB
Clock Speed	16 MHz



Length	68.6 mm
Width	53.4 mm
Weight	25 g
Figure: Arduino Uno Features	

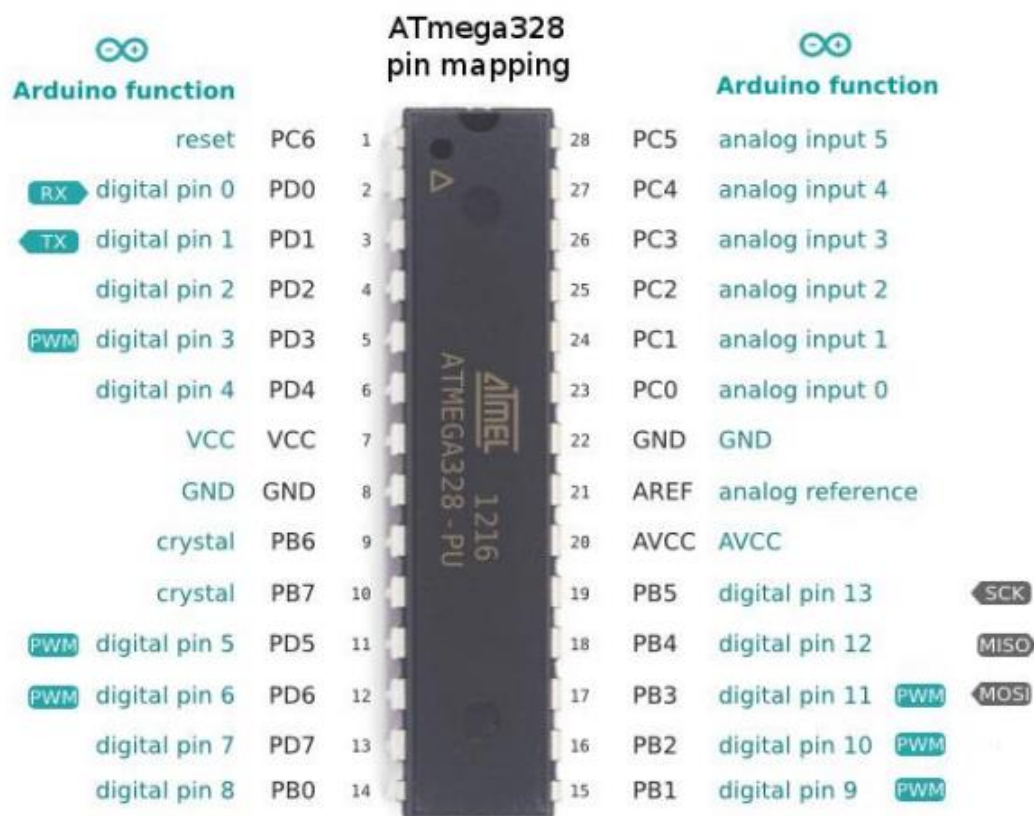


Figure:

Atmega 328P Pins



## OTHER TYPES of ARDUINOS

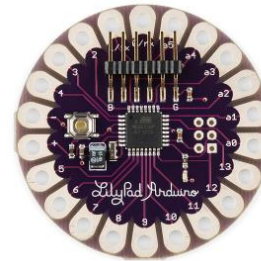
### ARDUINO MEGA

It has the Atmega 2560 microcontroller on it. It has 54 digital input-output pins, 16 analog inputs, 4 hardware serial ports, and a 16 mhz crystal oscillator. It is powered by both USB and DC adapter. Generally, the card, which has the same features as the Arduno UNO, is preferred in larger projects because it has more pins.



### ARDUINO LILYPAD

Lilypad is designed to be sewn on dresses and fabrics. In this way, it can be used in interesting projects that can be designed to be wearable. It has an Atmega 168V microcontroller on it.



### ARDUINO ETHERNET

It has an Ethernet chip and an Ethernet port for making internet-connected projects. There is also an SD-Card slot on the card, which has the Atmega 328 model as a microcontroller.



### ARDUINO BLUETOOTH

There is a Bluetooth module on Arduino BT, ideal for making applications communicating with the Bluetooth protocol. This module can also be used to program Arduino via Bluetooth.





## ARDUINO MINI

It is an Arduino model designed to be operated on a breadboard or integrated into another design. There is Atmega 168 or Atmega 328 model microcontroller on it. It is ideal for applications where small size is particularly important.



## ARDUINO NANO

It is a very small and designed model suitable for applications on the circuit board, and has an Atmega 328 or Atmega 168 microcontroller, voltage regulator, serial to USB converter chip, DC voltage input port and mini USB port.



## ARDUINO LEONARDO

It is one of the Arduino boards, which contains an Atmega 32u4 microcontroller on the Arduino Leonardo and does not require an additional chip for USB connection. With 20 digital inputs / outputs and 12 analog inputs, the microcontroller on the board has a surface mount cover. Thanks to its USB connection capabilities, Leonardo can be connected to the computer as a mouse or keyboard.



## ARDUINO ESPLORA

--	--



Co-funded by the  
Erasmus+ Programme  
of the European Union



Esplora is an Arduino board that contains various sensors, unlike the others. Thanks to the sensors on the card, it is possible to perform many applications without the need for other additions and excessive electronic knowledge. Esplora is equipped with a slide potentiometer, light and sound sensor, temperature sensor, sound generator, 2-axis mini analog joystick, 3-color LED and an accelerometer. Esplora is also equipped with Atmega 32U4 AVR microcontroller like Leonardo. Applications that can act as a mouse or keyboard can be developed when connected to a computer with its micro USB connection.



### Downloading the Arduino IDE

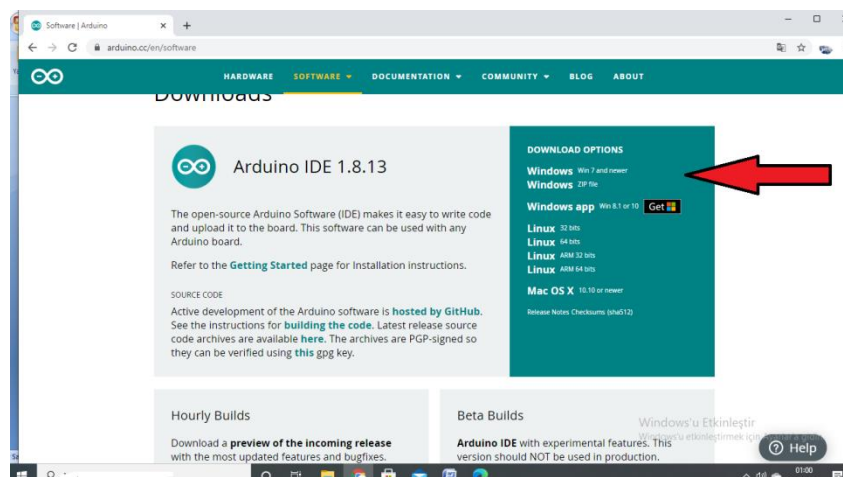
The Arduino IDE (Integrated Development Environment) is where you develop your programs that will tell the Arduino what to do.

To install the Arduino IDE for Windows, we have to follow instructions.

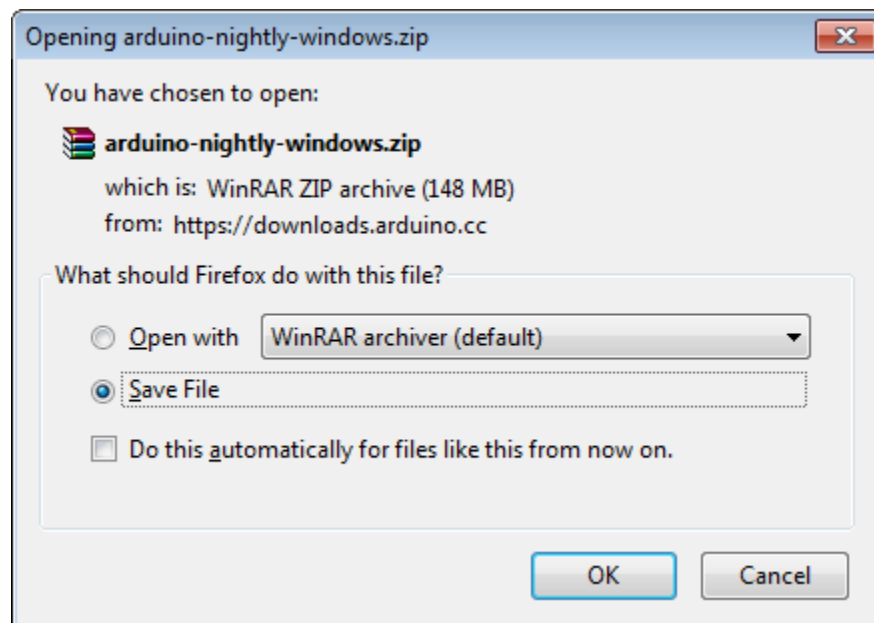
You can load new programs onto the main chip, the ATmega328p, via USB using the Arduino IDE.

To download the Arduino IDE, please click on the following link:

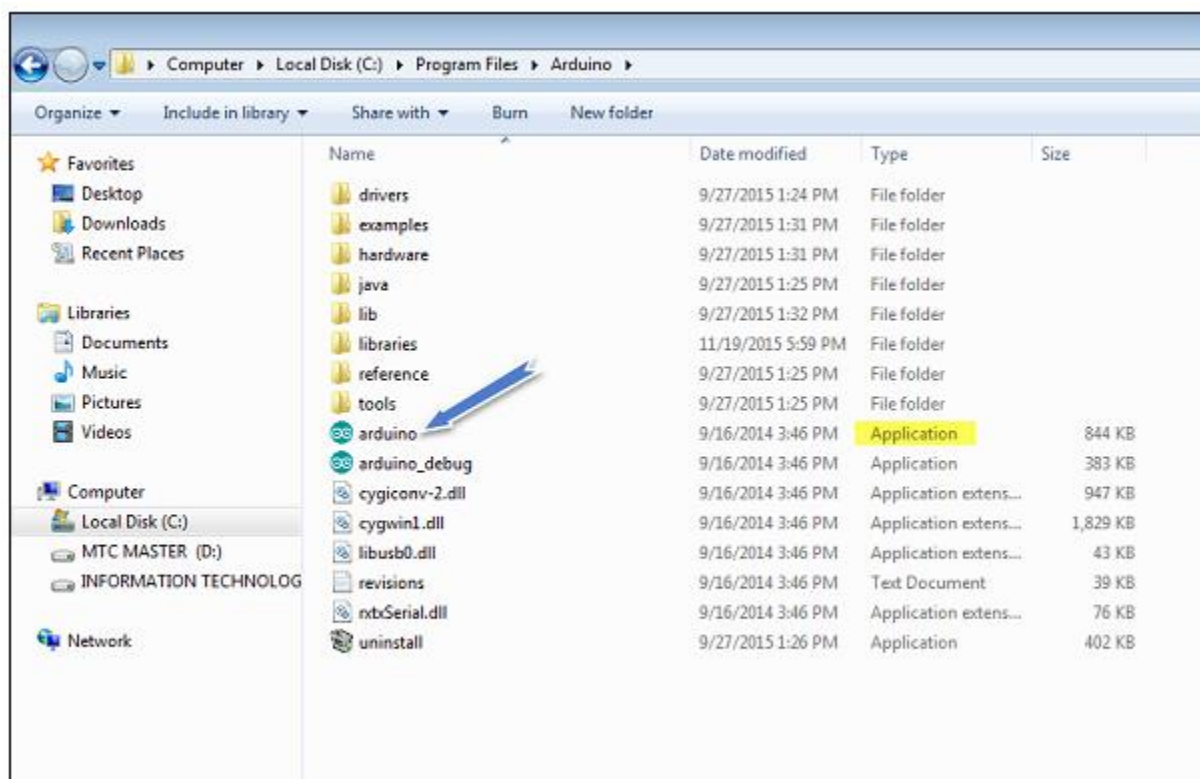
<https://www.arduino.cc/en/Main/Software>.



Select which Operating System you're using and download it. After our Arduino IDE software is downloaded, we need to unzip the folder.



Inside the folder, we can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE. Then, simply follow the installation wizard to install the Arduino IDE.

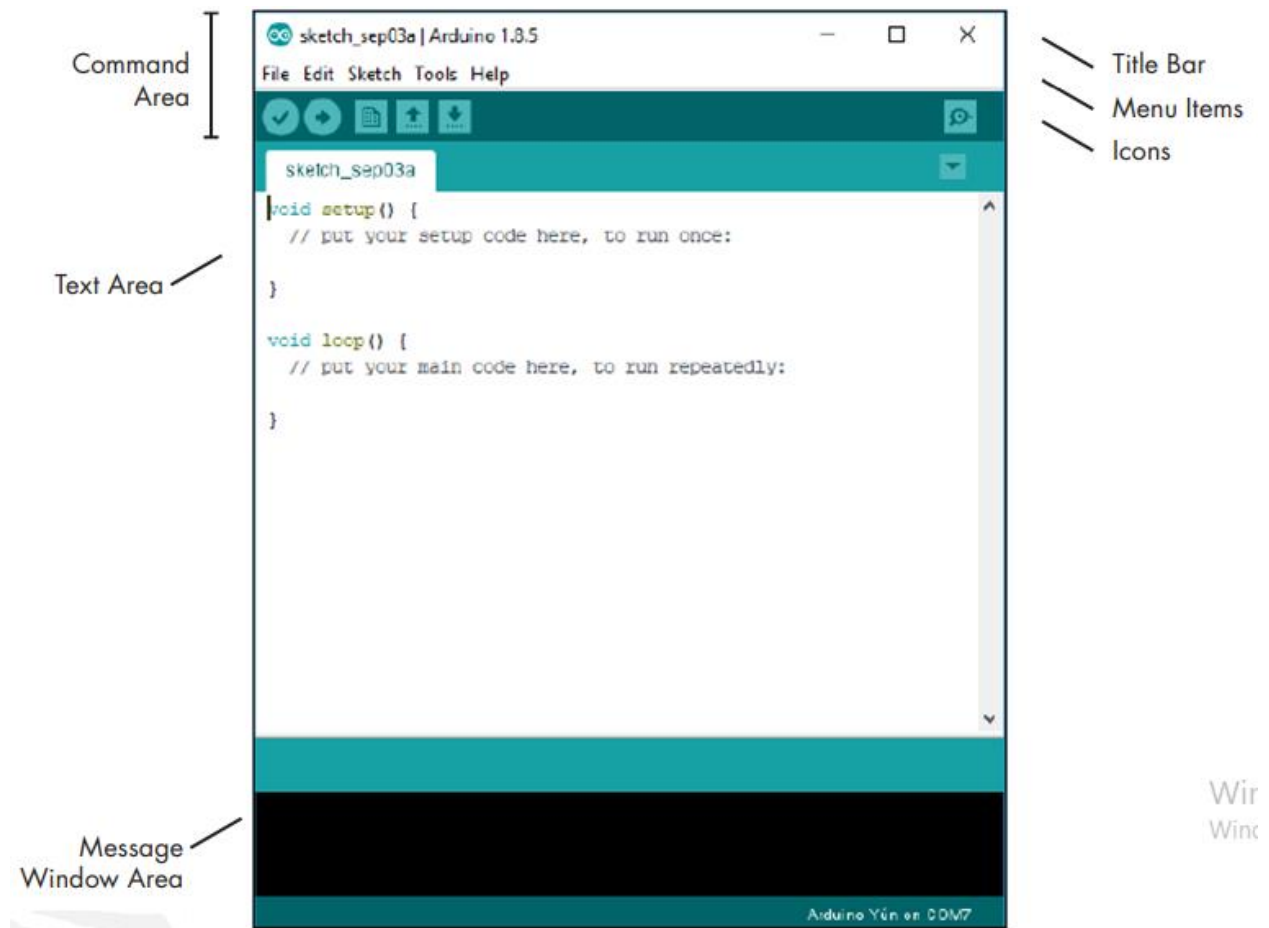




## Arduino IDE Window to Write Programs

When you first open the Arduino IDE, you should see something similar to the figure below.

As shown in Figure below, the Arduino IDE resembles a simple word processor. The IDE is divided into three main areas: the command area, the text area, and the message window area.



### Menu Items

As with any word processor or text editor, you can click one of the menu items to display its various options.

**File:** Contains options to save, load, and print sketches; a thorough set of example sketches to open; as well as the Preferences submenu.

**Edit:** Contains the usual copy, paste, and search functions common to any word processor

**Sketch:** Contains the function to verify your sketch before uploading to a board, and some sketch folder and import options.

**Tools:** Contains a variety of functions as well as the commands to select the Arduino board type and USB port.

**Help:** Contains links to various topics of interest and the version of the IDE.









### What the Sketch is

An Arduino sketch is a set of instructions that you create to accomplish a particular task; in other words, a sketch is a program.

The sketch is nothing more than a set of instructions for the Arduino to carry out. Sketches created using the Arduino IDE are saved as .pde files. To create a sketch, you need to make the three main parts: Variable declaration, the Setup function, and the main Loop function.

### Arduino IDE Toolbar Buttons

Below the menu toolbar are six icons. Mouse over each icon to display its name. The icons, from left to right, are as follows:

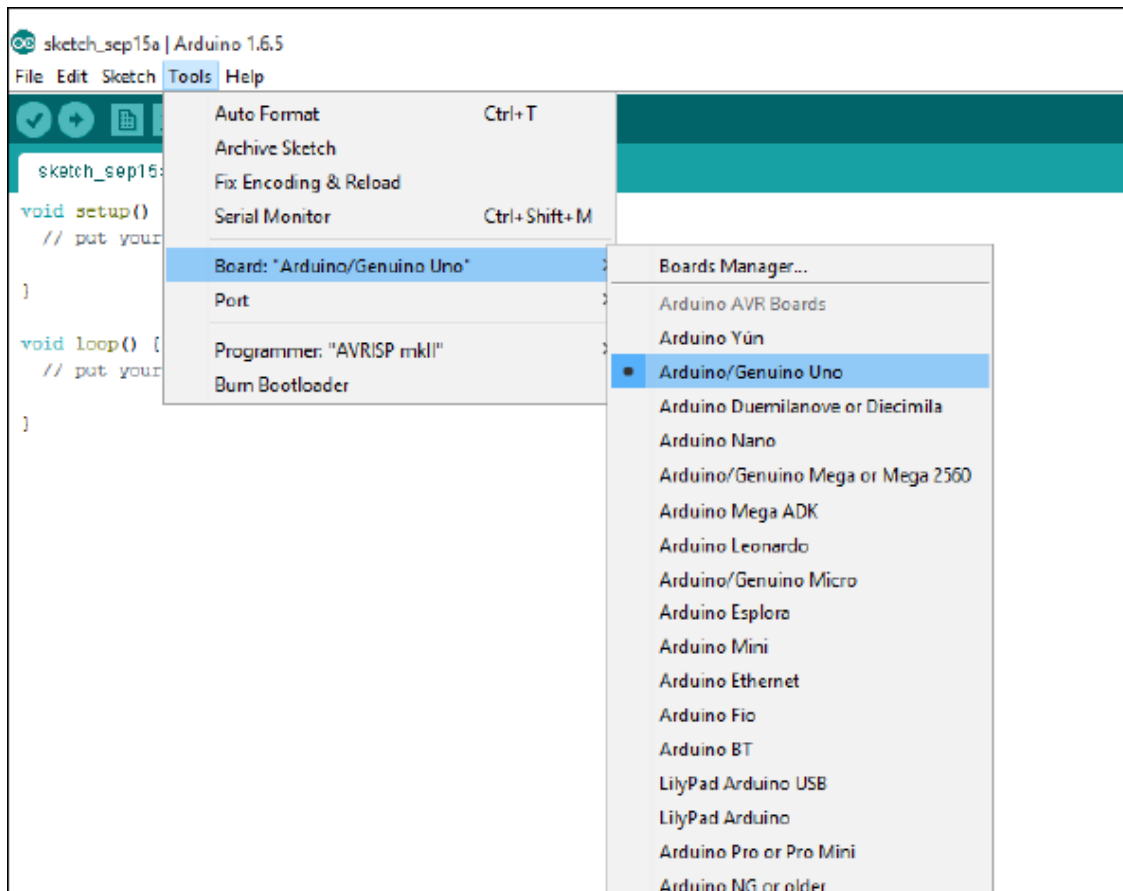
	<b>Verify (Compile):</b> Click this to check that the Arduino sketch is valid and doesn't contain any programming mistakes.
	<b>New:</b> Click this to open a new blank sketch in a new window.
	<b>Open:</b> Open Click this to open a saved sketch.
	<b>Save:</b> Click this to save the open sketch. If the sketch doesn't have a name, you will be prompted to create one.
	<b>Upload:</b> Click this to verify and then upload your sketch to the Arduino board.
	<b>Serial Monitor:</b> Click this to open a new window for use in sending and receiving data between your Arduino and the IDE.

### Connecting your Arduino

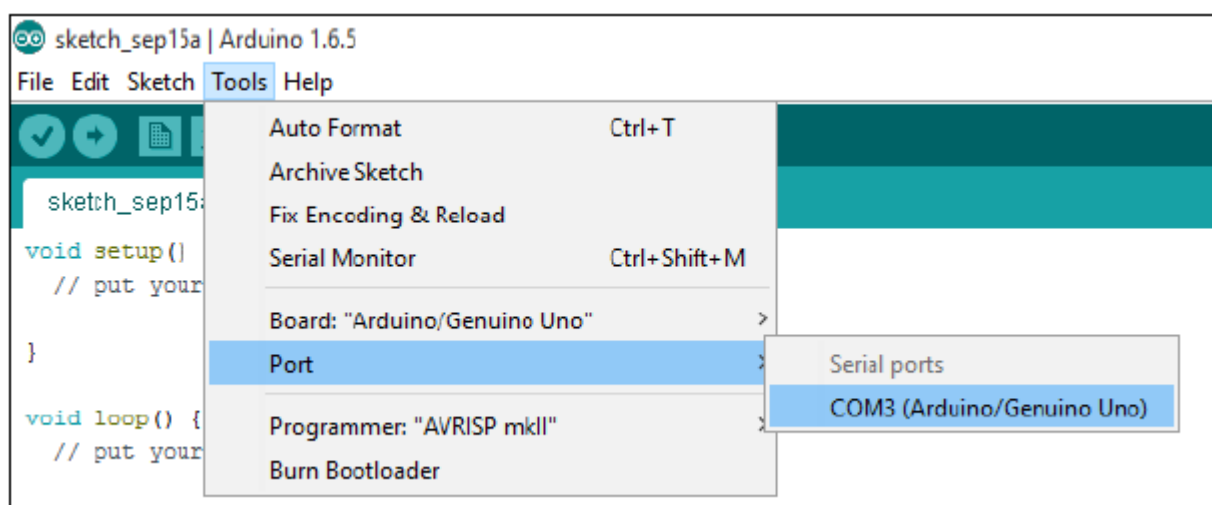
Connect your Arduino UNO to your computer via USB.

After connecting your Arduino with a USB cable, you need to make sure that the Arduino IDE has selected the right board.

In our case, we're using Arduino Uno, so we should go to **Tools** □ **Board:** □ **Arduino/Genuino Uno.**



Then, you should select the serial port where your Arduino is connected to. Go to **Tools**□**Port** and select the right port.

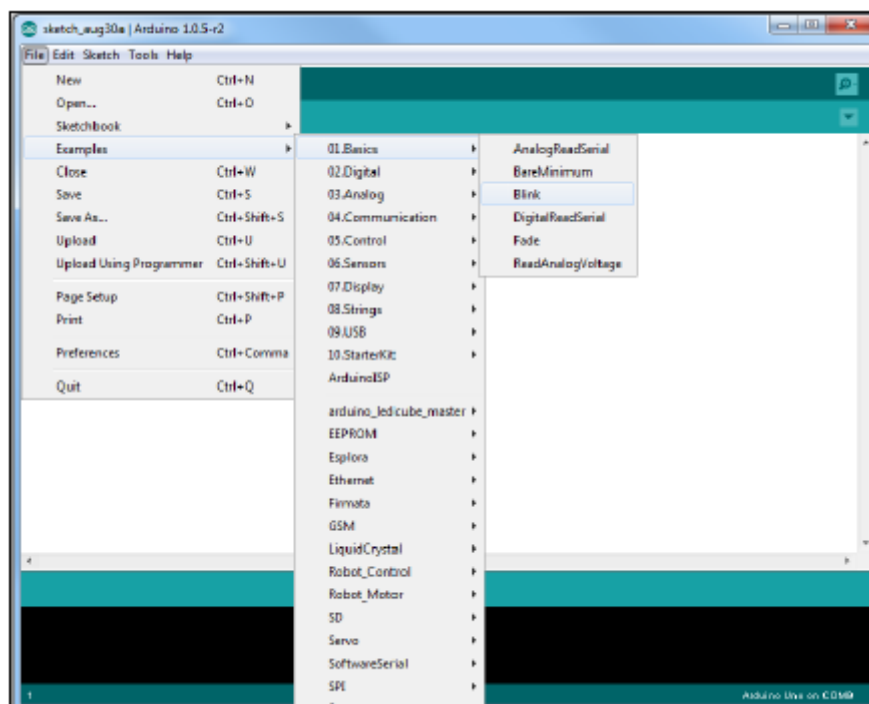




## Uploading an Arduino Sketch

To show you how to upload code to your Arduino board, we'll show you a simple example. This is one of the most basic examples – it consists in blinking the on-board LED or digital pin 13 every second.

1. Open your Arduino IDE.
2. Go to **File** ▢ **Examples** ▢ **01.Basics** ▢ **Blink**



By default, the Arduino IDE comes pre-configured for the Arduino UNO. Click the **Upload** button and wait a few seconds.



After a few seconds, you should see a **Done uploading** message.



```
Done uploading.  
Binary sketch size: 1.084 bytes (of a 32.256 byte maximum)  
2  
Arduino Uno on COM9
```

This code simply blinks the on-board LED on your Arduino UNO (highlighted with red color). You should see the little LED turn on for one second, and turn off for another second repeatedly.



### Control an Output and Read an Input

An Arduino board contains digital pins, analog pins and PWM pins.

### Difference between digital, analog and PWM

In **digital pins**, you have just two possible states, which are on or off. These can also be referred as High or Low, 1 or 0 and 5V or 0V.

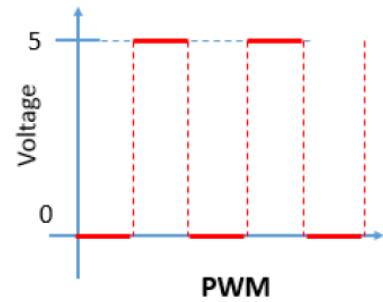
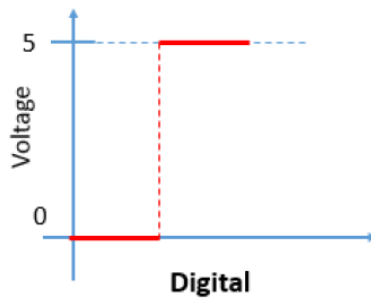
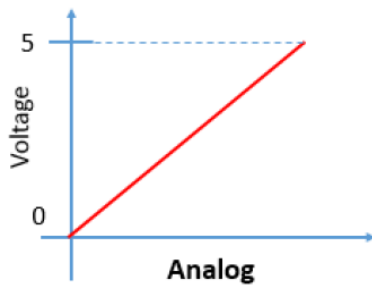
For example, if an LED is on, then, its state is High or 1 or 5V. If it is off, you'll have Low, or 0 or 0V.

In **analog pins**, you have unlimited possible states between 0 and 1023. This allows you to read sensor values. For example, with a light sensor, if it is very dark, you'll read 1023, if it is very bright you'll read 0. If there is a brightness between dark and very bright you'll read a value between 0 and 1023.

PWM pins are digital pins, so they output either 0 or 5V. However these pins can output "fake" intermediate voltage values between 0 and 5V, because they can perform "Pulse Width Modulation" (PWM). PWM allows to "simulate" varying levels of power by oscillating the output voltage of the Arduino.



Co-funded by the  
Erasmus+ Programme  
of the European Union



### Controlling an output

To control a digital output you use the `digitalWrite()` function and between brackets you write, the pin you want to control, and then HIGH or LOW.

To control a PWM pin you use the `analogWrite()` function and between brackets you write the pin you want to control and a number between 0 and 255.

### Reading an input

To read an analog input you use the function `analogRead()` and for a digital input you use `digitalRead()`.

**Note:** The best way for you to learn Arduino is practising. So, make many projects and start building something.



Co-funded by the  
Erasmus+ Programme  
of the European Union



## Erasmus+ KA210-VET

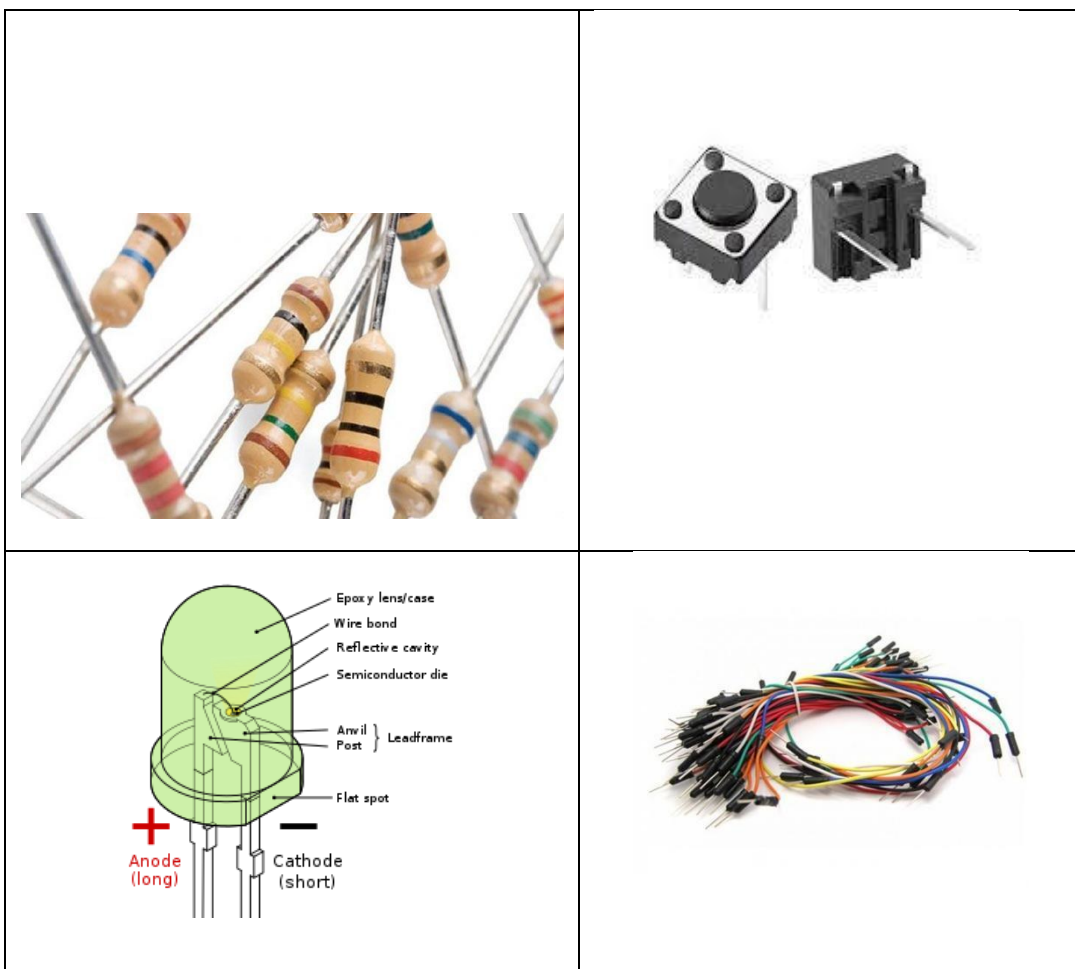
### Small-scale partnerships in vocational education and training

**Project Title: “Using Arduinos in Vocational Training”**

**Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

### Arduino Input/Output Module and Training Kit





## Planning Our Projects

When starting our first projects, you might be tempted to write your sketch immediately after you've come up with a new idea. But before you start writing, a few basic preparatory steps are in order. After all, your Arduino board isn't a mind-reader; it needs precise instructions, and even if these instructions can be executed by the Arduino, the results may not be what you expected if you overlooked even a minor detail.

Whether you are creating a project that simply blinks a light or an automated model railway signal, a detailed plan is the foundation of success. When designing your Arduino projects, follow these basic steps:

1. Define your objective. Determine what you want to achieve.
2. Write your algorithm. An algorithm is a set of instructions that describes how to accomplish your project. Your algorithm will list the steps necessary for you to achieve your project's objective.
3. Select your hardware. Determine how it will connect to the Arduino.
4. Write your sketch. Create your initial program that tells the Arduino what to do.
5. Wire it up. Connect your hardware, circuitry, and other items to the Arduino board.
6. Test and debug. Does it work? During this stage, you identify errors and find their causes, whether in the sketch, hardware, or algorithm.

The more time you spend planning your project, the easier time you'll have during the testing and debugging stage.

## Basic structure of a sketch

The Arduino program is called as "sketch". A sketch can be divided in three parts.

```
sketch_jan29a | Arduino 1.8.9
File Edit Sketch Tools Help

sketch_jan29a $

1. Name variable

void setup() {
  2. Setup (absolutely necessary for the program)
  // put your setup code here, to run once:

}

void loop() {
  3. Loop (absolutely necessary for the program)
  // put your main code here, to run repeatedly:

}
```



### **1. Name variable:**

In the first part elements of the program are named. This part is not absolutely necessary.

### **2. Setup (absolutely necessary for the program):**

The setup will be performed only once. Here you are telling the program for example what Pin (slot for cables) should be an input and what should be an output on the boards.

Defined as Output: The pin should put out a voltage. For example: With this pin a LED is meant to light up.

Defined as an Input: The board should read out a voltage. For example: A switch is actuated. The board recognized this, because it gets a voltage on the Input pin.

### **3. Loop (absolutely necessary for the program):**

This loop part will be continuously repeated by the board. It assimilates the sketch from beginning to end and starts again from the beginning and so on.

## **Further Syntax Rules**

It is necessary to pay attention to these rules while writing Arduino programs. Otherwise, our program will fail.

### **;(Semicolon):**

;(Semicolon) is used to end a statement. Forgetting to end a line in a semicolon will result in a compiler error.

Example: `int a=13;`

### **{ } (Curly Braces):**

Curly braces are a major part of the Arduino programming language. They are used in several different constructs, and this can sometimes be confusing for beginners. An opening curly brace "{" must always be followed by a closing curly brace "}".

The main uses of curly braces: Functions, Loops, Conditional statements

Example:

```
void myfunction(datatype argument)
{      statements(s)      }
```

### **// (Single line comment) and /\* \*/ (Multi-line comment):**



These are lines in the program that are used to inform yourself or others about the way the program works. They are ignored by the compiler, and not exported to the processor, so they don't take up any space on the Atmega chip.

Comments only purpose are to help you understand (or remember) how your program works or to inform others how your program works. There are two different ways of marking a line as a comment:

Example:

```
x = 5;    // This is a single line comment. Anything after the slashes is a comment
// to the end of the line
```

```
x = 5;    /*This is a multi-line comment. ....
This is the end of a multi-line comment. */
```

### **#define:**

`#define` allows the programmer to give a name to a constant value before the program is compiled. Defined constants in arduino don't take up any program memory space on the chip. In general, the `const` keyword is preferred for defining constants and should be used instead of `#define`.

Example:

```
#define ledPin 3    // The compiler will replace any mention of ledPin with the value 3 at
compile time.
```

### **#include:**

`#include` is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

Example:

```
#include <servo.h>
```

## **Arduino - Data Types**

Data types are used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.

Expressions that are used to store any information in memory and can change the value during program flow are called variables. Variables can be numbers, characters, or logical expressions.



The appropriate data type should be selected according to the variable type. A certain area is allocated according to the type of data, used in defining the variable in memory.

The following table provides all the data types that you will use during Arduino programming.

Type	Contains
boolean	can contain either true or false
char	-128 to 127
byte	0 to 255
unsigned char	0 to 255
int	-32,768 to 32,767
unsigned int	0 to 65,535
word	(same as unsigned int)
long (or long int)	-2,147,483,648 to 2,147,483,647
unsigned long	0 to 4,294,967,295
float	-3.4028235E+38 to 3.4028235E+38
double	(same as float)

### Arduino - Variables & Constants

While defining the variable, the name of the variable, the value of the variable and the appropriate data type for the variable must be determined.

The definition is made as seen in the example below:

**int LED=12;**

Here: **int**=data type                      **LED**=variable name                      **12**=variable value

Variables, which Arduino uses, have a property called scope. A scope is a region of the program and there are three places where variables can be declared. They are:

- Inside a function or a block, which is called local variables.
- In the definition of function parameters, which is called formal parameters.



- Outside of all functions, which is called global variables.

A constant is a variable *qualifier* that modifies the behavior of the variable, making a variable "read-only". This means that the variable can be used just as any other variable of its type, but its value cannot be changed. You will get a compiler error if you try to assign a value to a const variable.

Constants defined with the const keyword obey the rules of variable scoping that govern other variables. This, and the pitfalls of using #define, makes the const keyword a superior method for defining constants and is preferred over using #define.

Example:

```
const float pi = 3.14;
```

#### Notes:

#define **or** const: You can use either const or #define for creating numeric or string constants. For arrays, you will need to use const. In general const is preferred over #define for defining constants.

## Arduino – Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. In Arduinos, the following types of operators may be used.

### Arithmetic Operators:

Assume variable A holds 10 and variable B holds 20 then –

Operator name	Operator simple	Example
assignment operator	=	A = B
addition	+	A + B will give 30
subtraction	-	A - B will give -10



multiplication	*	A * B will give 200
division	/	B / A will give 2
modulo	%	B % A will give 0

### Comparison Operators:

Assume variable A holds 10 and variable B holds 20 then –

Operator name	Operator symbol	Example
equal to	==	(A == B) is not true
not equal to	!=	(A != B) is true
less than	<	(A < B) is true
greater than	>	(A > B) is not true
less than or equal to	<=	(A <= B) is true
greater than or equal to	>=	(A >= B) is not true

### Boolean Operators

Assume variable A holds 10 and variable B holds 20 then –

Operator name	Operator simple	Example
and	&&	(A && B) is true
or		(A    B) is true
not	!	!(A && B) is false



## Bitwise Operators

Assume variable A holds 60 and variable B holds 13 then –

Operator name	Operator simple	Example
and	&	(A & B) will give 12 which is 0000 1100
or		(A   B) will give 61 which is 0011 1101
xor	^	(A ^ B) will give 49 which is 0011 0001
not	~	(~A ) will give -60 which is 1100 0011
shift left	<<	A << 2 will give 240 which is 1111 0000
shift right	>>	A >> 2 will give 15 which is 0000 1111

## Arduino - I/O Functions (Commands)

Functions allow structuring the programs to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program. They will become clearer when we show actual program examples in circuits and Arduino programmes below. To help explain the various command functions, we've broken them down into separate Commands

The pins on the Arduino board can be configured as either inputs or outputs. We will explain the functioning of the pins in those modes. It is important to note that a majority of Arduino analog pins, may be configured, and used, in exactly the same manner as digital pins.

### pinMode() Function:

The pinMode() function is used to configure a specific pin to behave either as an input or an output. It is possible to enable the internal pull-up resistors with the mode INPUT\_PULLUP.

Syntax:   pinMode(pin, mode)  
          Void setup () {



```
pinMode (pin , mode);  
    }
```

#### Parameters:

pin: the Arduino pin: number to set the mode of.

mode: INPUT, OUTPUT, or INPUT\_PULLUP.

#### Examples:

```
pinMode(13, OUTPUT);    // sets the digital pin 13 as output  
pinMode(5, INPUT);      // sets the digital pin 5 as input
```

### **digitalWrite() Function:**

The digitalWrite() function is used to write a HIGH or a LOW value to a digital pin. If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW. If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT\_PULLUP to enable the internal pull-up resistor.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.

#### Syntax:

```
digitalWrite (pin ,value);
```

pin: the number of the pin whose mode you wish to set

value:HIGH (1), or LOW(0).

#### Example:

```
digitalWrite(LED, HIGH);    // turn on led  
digitalWrite(LED, LOW);     // turn off led
```



### **digitalRead() Function:**

The digitalRead() function reads the value from a specified digital pin, either HIGH or LOW.

Syntax: digitalRead(pin)

pin: the Arduino pin number you want to read.

Examples:

```
val = digitalRead(inPin); // read the input pin
```

Note: The analog input pins can be used as digital pins, referred to as A0, A1, etc.

### **delay() Function:**

The delay() function pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax: delay(ms):

ms: the number of milliseconds to pause. Allowed data types: unsigned long.

Examples:

```
delay(1000);           // waits for 1 second  
delay(2000);           // waits for 2 seconds
```

### **analogWrite() Function:**

The analogWrite() function writes an analog value (PWM wave) to a pin. It can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

Examples:

Sets the output to the LED proportional to the value read from the potentiometer.

```
val = analogRead(analogPin); // read the input pin  
analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023, analogWrite values from 0 to 255
```



## **analogRead( ) Function**

Arduino is able to detect whether there is a voltage applied to one of its pins and report it through the `digitalRead()` function. There is a difference between an on/off sensor (which detects the presence of an object) and an analog sensor, whose value continuously changes. In order to read this type of sensor, we need a different type of pin.

In the lower-right part of the Arduino board, you will see six pins marked “Analog In”. These special pins not only tell whether there is a voltage applied to them, but also its value. By using the `analogRead()` function, we can read the voltage applied to one of the pins.

This function returns a number between 0 and 1023, which represents voltages between 0 and 5 volts. For example, if there is a voltage of 2.5 V applied to pin number 0, `analogRead(0)` returns 512.

Syntax: `analogRead(pin);`

pin: the number of the analog input pin to read from 0 to 5.

Example:

```
val = analogRead(analogPin);    // read the input pin
Serial.println(val);             // debug value
```

## **if Function:**

The if statement checks for a condition and executes the following statement or set of statements if the condition is 'true'.

Syntax:

```
if (condition) {
//statement(s)
}
```

**condition:** a boolean expression (i.e., can be true or false).

**Examples:** ( The brackets may be omitted after an if statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120) {digitalWrite(LEDpin, HIGH);}

if (x > 120) {
digitalWrite(LEDpin1, HIGH);
digitalWrite(LEDpin2, HIGH);
}
```



```
}  
    // all are correct
```

### if-else Command:

The if...else allows greater control over the flow of code than the basic if statement, by allowing multiple tests to be grouped. An else clause (if at all exists) will be executed if the condition in the if statement results in false. The else can proceed another if test, so that multiple, mutually exclusive tests can be run at the same time.

Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default else block is executed, if one is present, and sets the default behavior. An unlimited number of such else if branches are allowed.

#### Syntax:

```
if (condition is TRUE) {  
    // do Thing A  
}  
else  
    //if NOT, do Thing B  
}
```

```
if (condition1) {  
    // do Thing A  
}  
else if (condition2) {  
    // do Thing B  
}  
else {  
    // do Thing C  
}
```

Examples: ( Below is an extract from a code for temperature sensor system.)

```
if (temperature >= 70) {  
    // Danger! Shut down the system.  
}  
else if (temperature >= 60) { // 60 <= temperature < 70  
    // Warning! User attention required.  
}  
else { // temperature < 60  
    // Safe! Continue usual tasks.  
}
```



### **for Command:**

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

### **Syntax:**

```
for (initialization; condition; increment) {  
    // statement(s);  
}
```

### **Parameters:**

initialization: happens first and exactly once.

condition: each time through the loop, condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

increment: executed each time through the loop when condition is true.

### **Examples:**

```
for (int i = 0; i <= 255; i++) {  
    analogWrite(PWMpin, i);  
}  
  
for (int x = 2; x < 100; x = x * 1.5) {  
    println(x);  
}  
  
for (int i = 0; i > -1; i = i + x) {  
    analogWrite(PWMpin, i);  
}
```

### **switch...case Command**

Like **if** statements, switch/ case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The **break** keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.



### Syntax:

```
switch (var) {  
  case label1:  
    // statements  
    break;  
  case label2:  
    // statements  
    break;  
  default:  
    // statements  
    break;  
}
```

### Example Code:

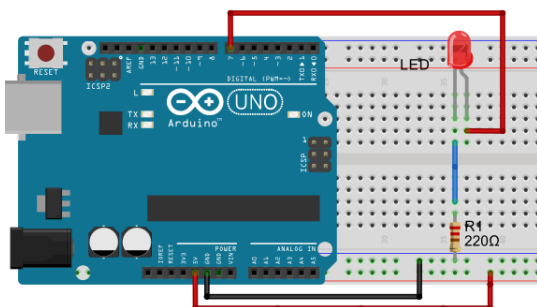
```
switch (var) {  
  case 1:  
    //do something when var equals 1  
    break;  
  case 2:  
    //do something when var equals 2  
    break;  
  default:  
    // if nothing else matches, do the default  
    // default is optional  
    break;  
}
```

**var:** a variable whose value to compare with various cases. Allowed data types: int, char.  
**label1, label2:** constants. Allowed data types: int, char.

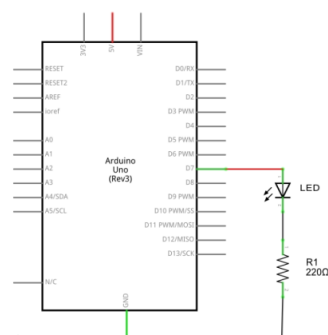
### Note:

Arduino circuits are shown in two ways;

- 1-) Breadboard view
- 2-) Schematic view



1-) Breadboard view



2-) Schematic view

1-

We will use Breadboard view which is used more.

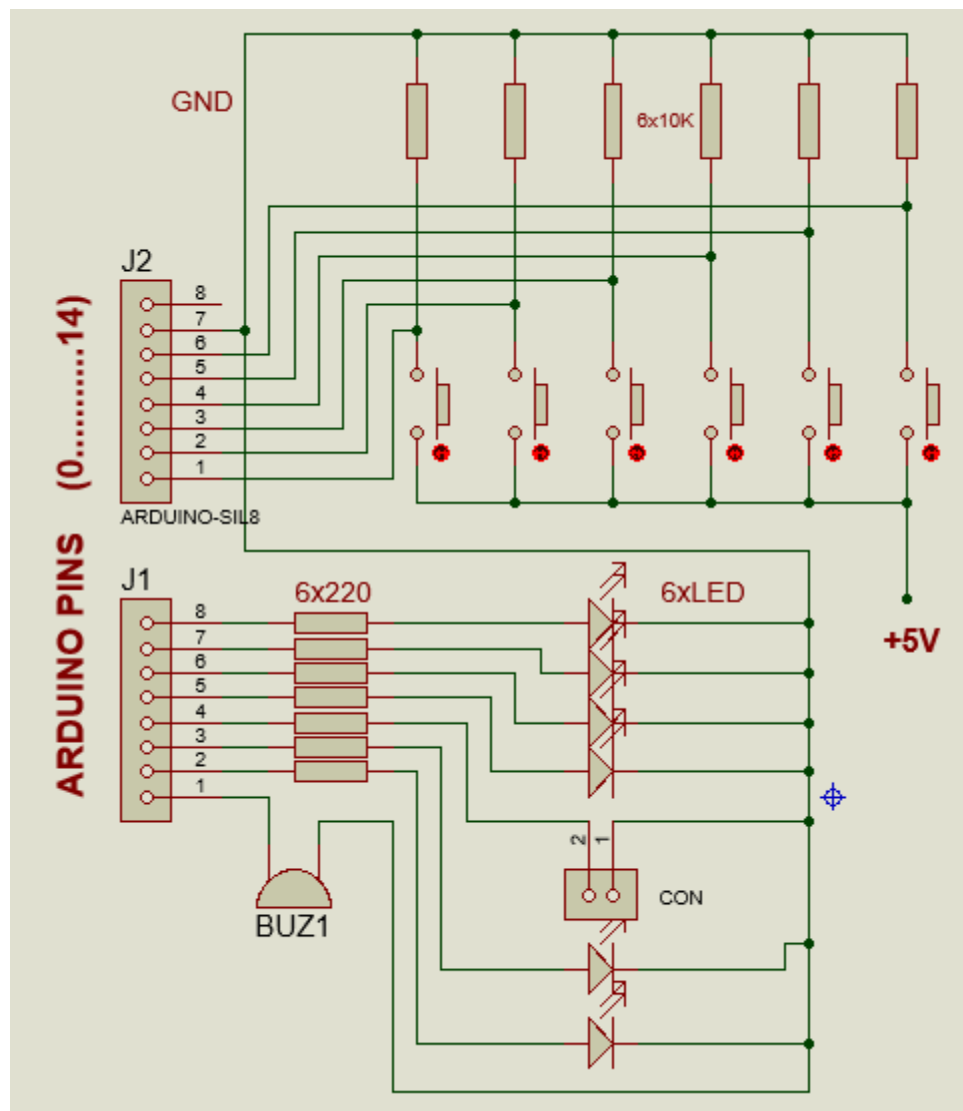
Enough! Let's make something!



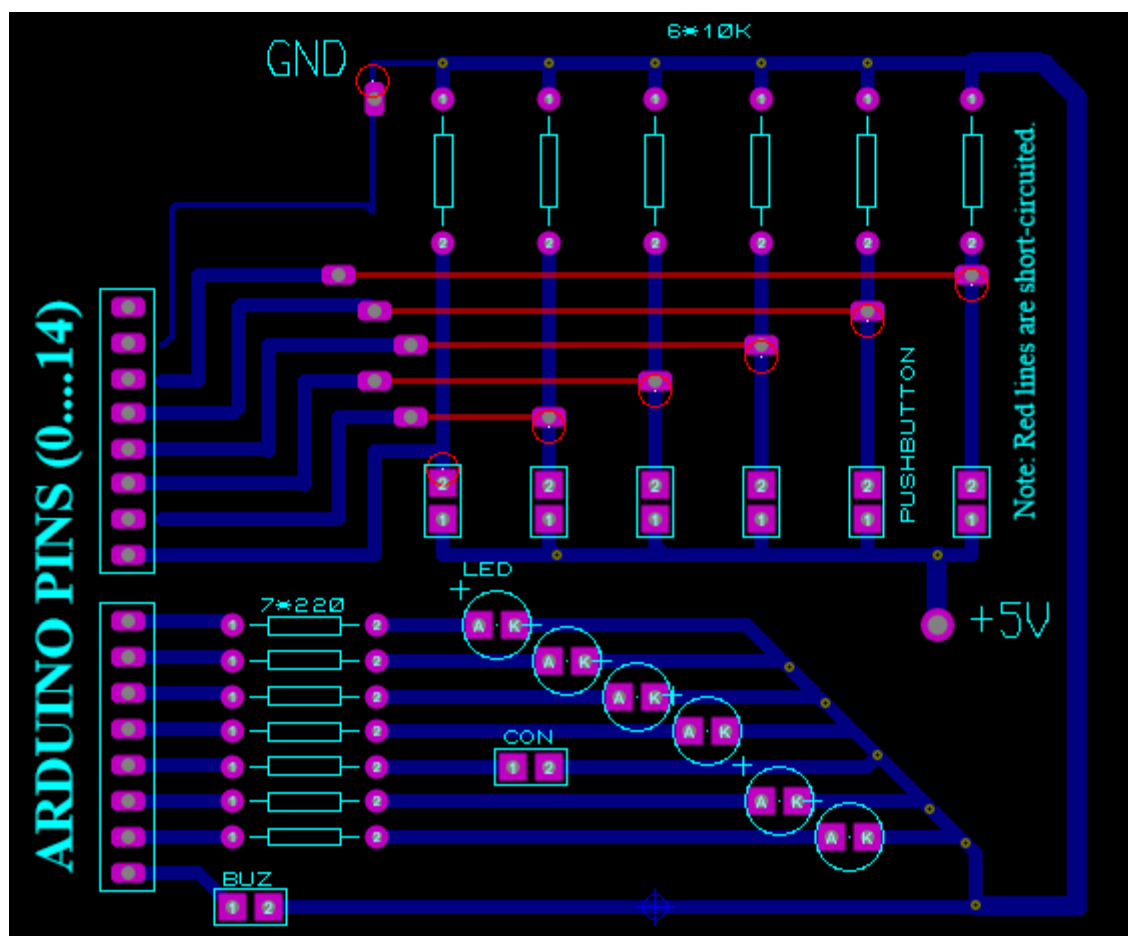
### Circuits of Arduino Button and LED Module Kit

Most pins on the Arduino can be configured as input or output. The Button and LED Module Kit is first training KIT of the UsingARDinVET to make students learn I/O systems of Arduinos. So, it can be named as I/O Arduino training KIT. In this Training Kit, as shown as below, 6 buttons are connected to Arduino as inputs. And a buzzer, a 2-pin connector and 6 LEDs are connected as outputs.

As students can use this training kit to learn I/O systems of Arduinos, this training kit can make them test Arduino circuit more easy. Also, they can use a breadboard to test Arduino circuits and experiments.



**Figure:** Open Circuit of Button and LED Module Kit



**Figure:** PCB Schema of Button and LED Module Kit

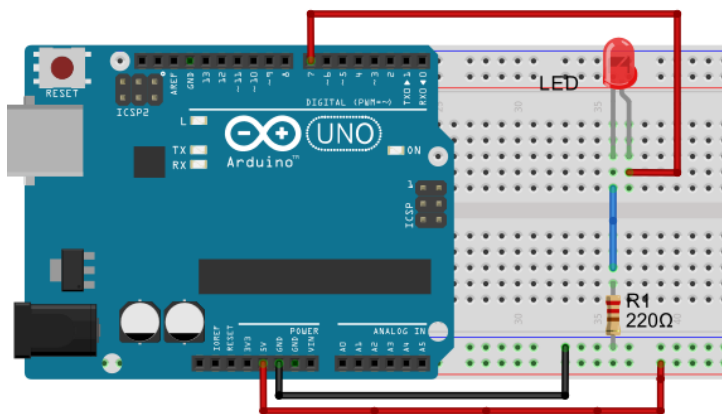


## Sample Arduino Circuits and Programs

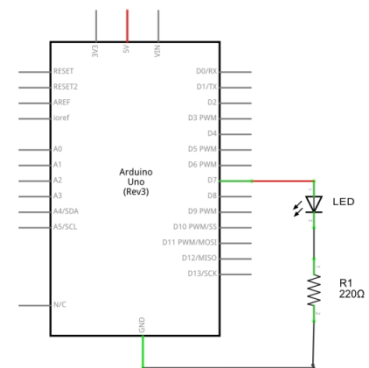
### Circuit 1:

#### Circuit title: LED flashing Program

Circuit description: An LED is connected to 13th pin of the Arduino. The LED is flashed continuously with 1second interval.



1-) Breadboard view



2-) Schematic view

#### /\* LED flashing Program, Switching a LED on and off \*/

```
int led = 7;                // integer variable led is declared

void setup() {              // the setup() method is executed only once
  pinMode(led, OUTPUT);    // the led PIN is declared as digital output
}

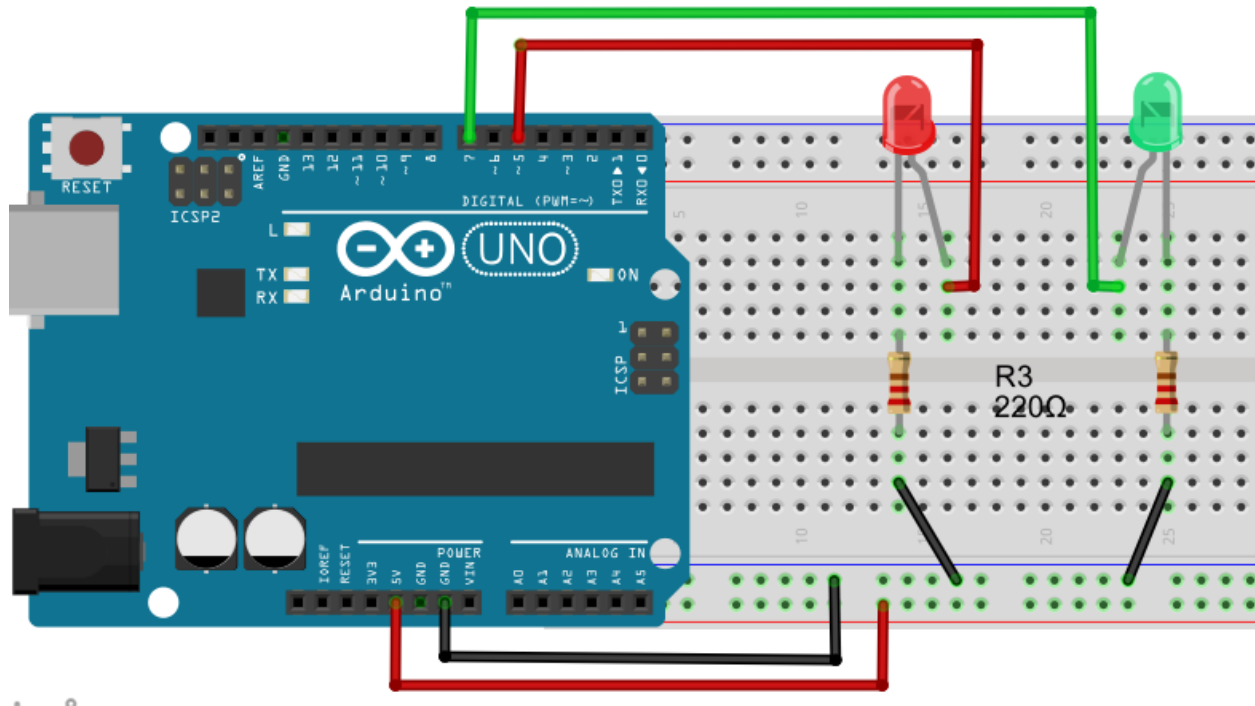
void loop() {               // the loop() method is repeated
  digitalWrite(led, HIGH);  // switching on the led
  delay(1000);              // stopping the program for 1000 milliseconds
  digitalWrite(led, LOW);   // switching off the led
  delay(1000);              // stopping the program for 1000 milliseconds
}
```



## Circuit 2:

**Circuit title:** Flip-Flop , 2 LEDs flashing Program

**Circuit description:** 2 LEDs blink sequentially with 2 second intervals.



```
/* Flip Flop */
```

```
int greenLED=5;  
int redLED=7;
```

```
// Pin where the green LED is attached  
// Pin where the red LED is attached
```

```
void setup() {  
  pinMode(greenLED, OUTPUT);  
  pinMode(redLED, OUTPUT);  
}
```

```
// green LED pin is initialised as OUTPUT  
// red LED pin is initialised as OUTPUT
```

```
void loop(){  
  digitalWrite(greenLED, HIGH);  
  digitalWrite(redLED, LOW);  
  pause(2000);
```

```
// switch on green LED  
// switch off red LED
```

```
  digitalWrite(greenLED, LOW);  
  digitalWrite(redLED, HIGH);  
  pause(2000);
```

```
// switch off green LED  
// switch on red LED
```

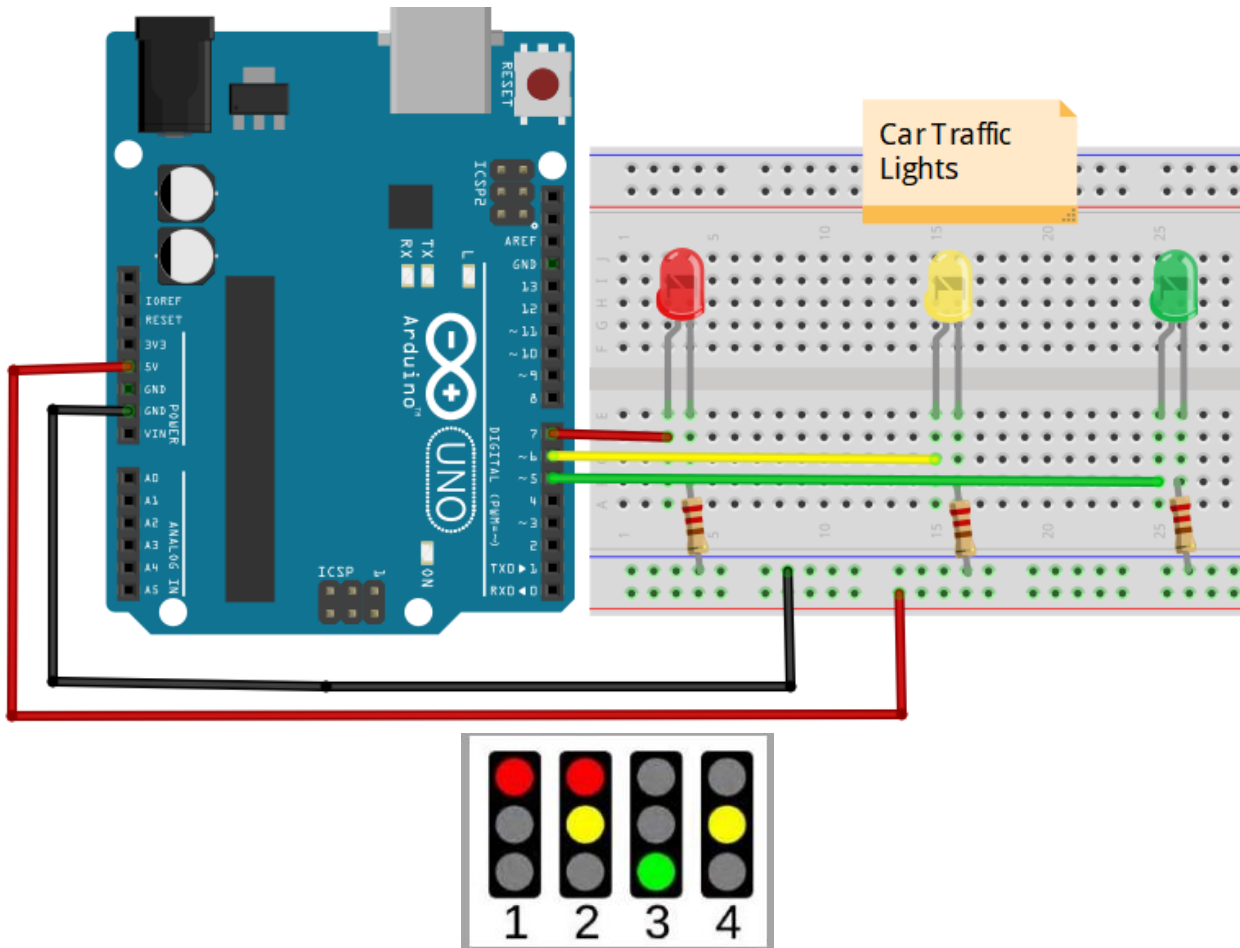
```
}
```



### Circuit 3:

**Circuit title:** Traffic Lights

**Circuit description:** In this Project, We are going to build a traffic lights system. There are 3 LEDs with different colors (green, yellow and red).



**/\* Traffic Lights \*/**

```
int redLED = 7;  
int yellowLED = 6;  
int greenLED = 5;
```

```
void setup() {  
  pinMode(redLED, OUTPUT);  
  pinMode(yellowLED, OUTPUT);  
  pinMode(greenLED, OUTPUT);  
}
```

// here, we are initializing our pins as outputs

```
void loop() {
```

```
  digitalWrite(redLED, HIGH);
```

// redLED is ON for 9 seconds



```
digitalWrite(yellowLED, LOW);  
digitalWrite(greenLED, LOW);  
delay(9000);
```

```
digitalWrite(redLED, HIGH);  
digitalWrite(yellowLED, HIGH);  
digitalWrite(greenLED, LOW);  
delay(2000);
```

// redLED and yellowLED are ON for 2seconds

```
digitalWrite(redLED, LOW);  
digitalWrite(yellowLED, LOW);  
digitalWrite(greenLED, HIGH);  
delay(9000);
```

// greenLED is ON for 9 seconds

```
digitalWrite(redLED, LOW);  
digitalWrite(yellowLED, HIGH);  
digitalWrite(greenLED, LOW);  
delay(2000);
```

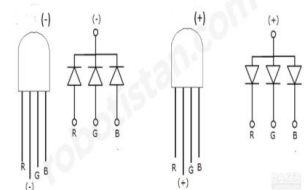
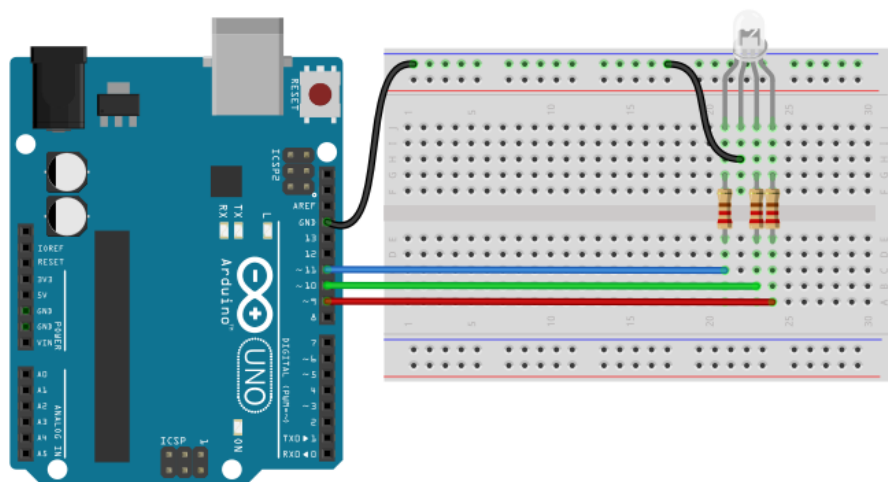
// Again, yellowLED is ON for 2seconds

```
/* The loop starts again */  
}
```

#### Circuit 4:

#### Circuit title: RGB LED APPLICATION

**Circuit** Explanation: The RGB LED is the 3 LEDs, connected in common, placed in a single case. It is possible to control the light intensity of three colors digitally. In addition, desired colors can be obtained by using the PWM technique.





/\* We will flash each color of RGB LED in 1 second intervals. If we want to display white light, we need to turn on all the LEDs..\*/

```
const int BlueLed=11;    // we connect the blue led to pin-11
const int GreenLed=10;   // we connect the green led to pin-10
const int RedLed=9;      // we connect the red led to pin-11
```

// We assign the pins to which the LEDs are connected as outputs.

```
void setup() {
  pinMode(BlueLed,OUTPUT);
  pinMode(GreenLed,OUTPUT);
  pinMode(RedLed,OUTPUT); }
```

// The loop starts here.

```
void loop() {
  digitalWrite(BlueLed, LOW);    // RedLed is ON.
  digitalWrite(GreenLed, LOW);
  digitalWrite(RedLed, HIGH);
  delay(1000);
```

```
  digitalWrite(BlueLed, LOW );  // GreenLed is ON.
  digitalWrite(GreenLed, HIGH);
  digitalWrite(RedLed, LOW );
  delay(1000);
```

```
  digitalWrite(BlueLed, HIGH);  // BlueLed is ON.
  digitalWrite(GreenLed, LOW);
  digitalWrite(RedLed, LOW);
  delay(1000);
```

// We display the white color by activating all the leds.

```
  digitalWrite(BlueLed, HIGH);
  digitalWrite(GreenLed, HIGH);
  digitalWrite(RedLed, HIGH);
  delay(1000);
}
```

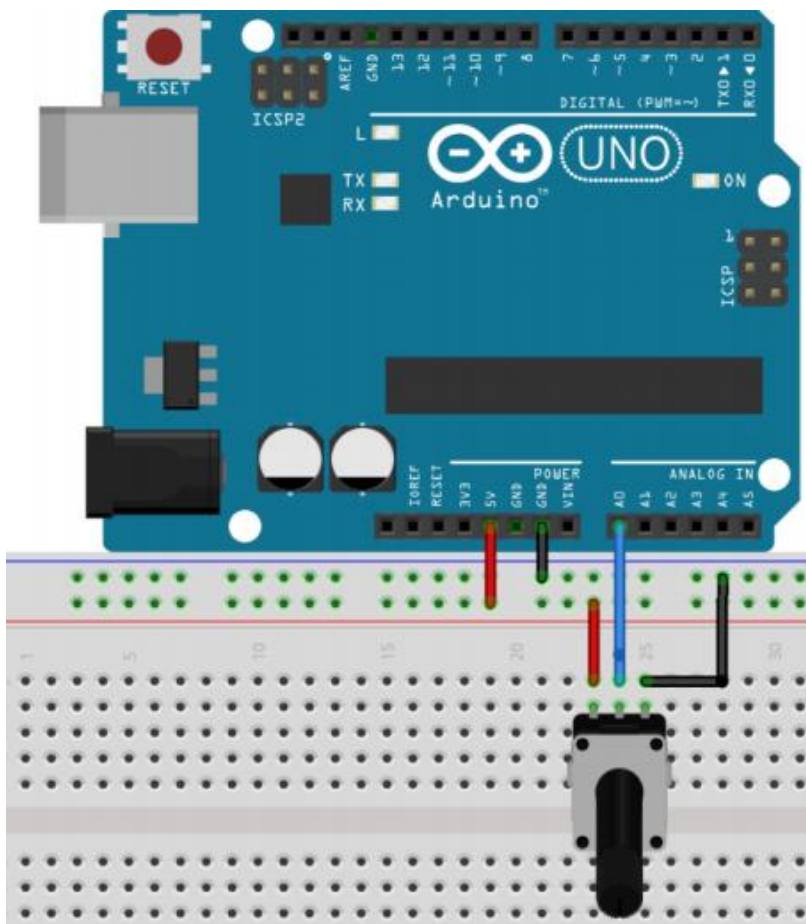


## Circuit 5:

### Circuit title: Reading Analog Voltage, Reading the Value From Potentiometer

**Circuit Explanation:** Here, we learn how to read an analog input on analog pin-0. The input is converted from `analogRead()` into voltage, and printed out to the serial monitor of the Arduino IDE.

**Potentiometer:** A potentiometer (or pot) is a simple electro-mechanical transducer. It converts rotary or linear motion from the input operator into a change of resistance. This change is (or can be) used to control anything from the volume of a hi-fi system to the direction of a huge container ship.



`/* ReadAnalogVoltage :` Reads an analog input on pin 0, converts it to voltage, and prints the result to the serial monitor.

Graphical representation is available using serial plotter (Tools > Serial Plotter menu).

Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground. `*/`

`// the setup routine runs once when you press reset:`



Co-funded by the  
Erasmus+ Programme  
of the European Union



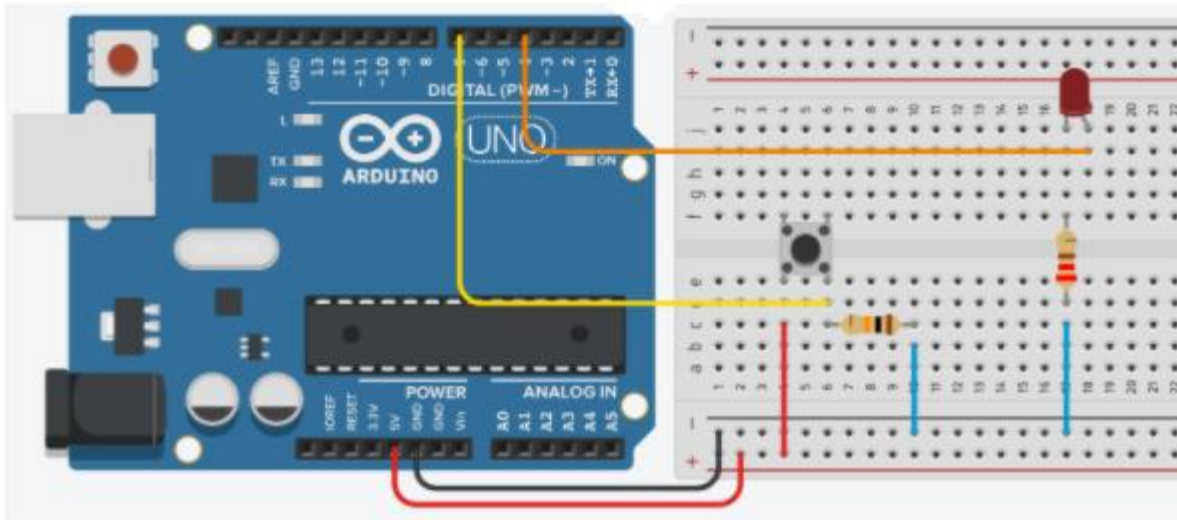
```
void setup()    {  
    // initialize serial communication at 9600 bits per second:  
  
    Serial.begin(9600); }  
  
    // the loop routine runs over and over again forever:  
  
void loop() {  
    // read the input on analog pin 0:  
  
    int sensorValue = analogRead(A0);  
  
    // print out the value you read:  
  
    Serial.println(sensorValue);  
  
    delay(1000);    // delay between reads for stability  
}
```



### Circuit 6:

#### Circuit title: Using buttons on Arduinos

**Circuit Explanation:** When pressing a pushbutton attached to pin 7, the button turns on and off a (LED), connected to digital pin 4,



/\* Button Turns on and off a light emitting diode(LED) connected to digital pin 4, when pressing a pushbutton attached to pin 7. \*/

```
void setup()
```

```
{
```

```
  pinMode(4, OUTPUT); // pin-4 is output
```

```
  pinMode(7, INPUT); // pin7 is input. 7
```

```
}
```

```
void loop()
```

```
{
```

```
  if (digitalRead(7) == HIGH)           // If pin7 is high,
```

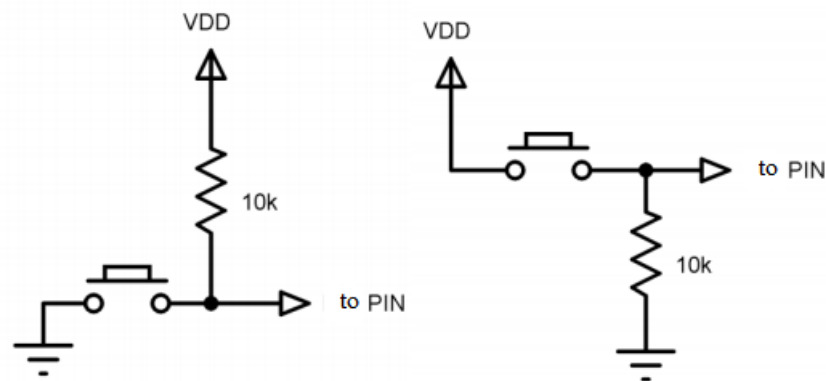
```
    digitalWrite(4, HIGH);              // Led is ON,
```

```
  if (digitalRead(7) == LOW)            // If pin7 is low,
```

```
    digitalWrite(4, LOW);               // Led is OFF.
```

```
}
```

**NOTE:** IF-THEN command also can be used for connecting buttons to Inputs pins of Arduinos. These connection can be done in two different ways. Pull\_Up connection and Pull-down connection.

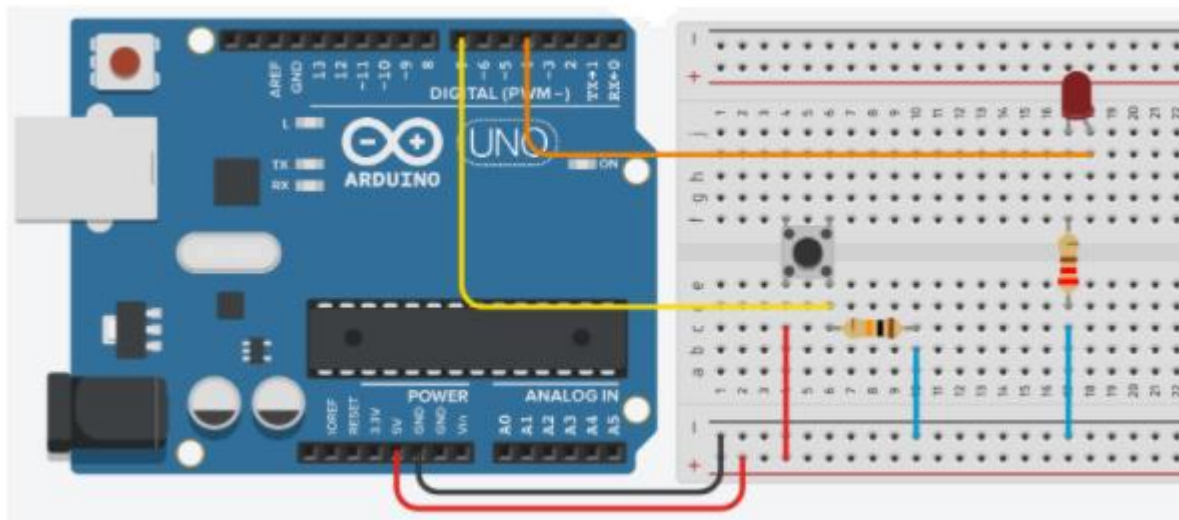


**Figure:** Switches or Buttons that can be used for the IF...THEN command

### Circuit 7:

#### Circuit title: Using ELSE command on Arduinos

**Circuit Explanation:** When pressing a pushbutton attached to pin 7, the button turns on and off a (LED), connected to digital pin 4. Also, We will learn to determine the pins with the "int" variable.



```
int led = 4;
int buton =7;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buton, INPUT);
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



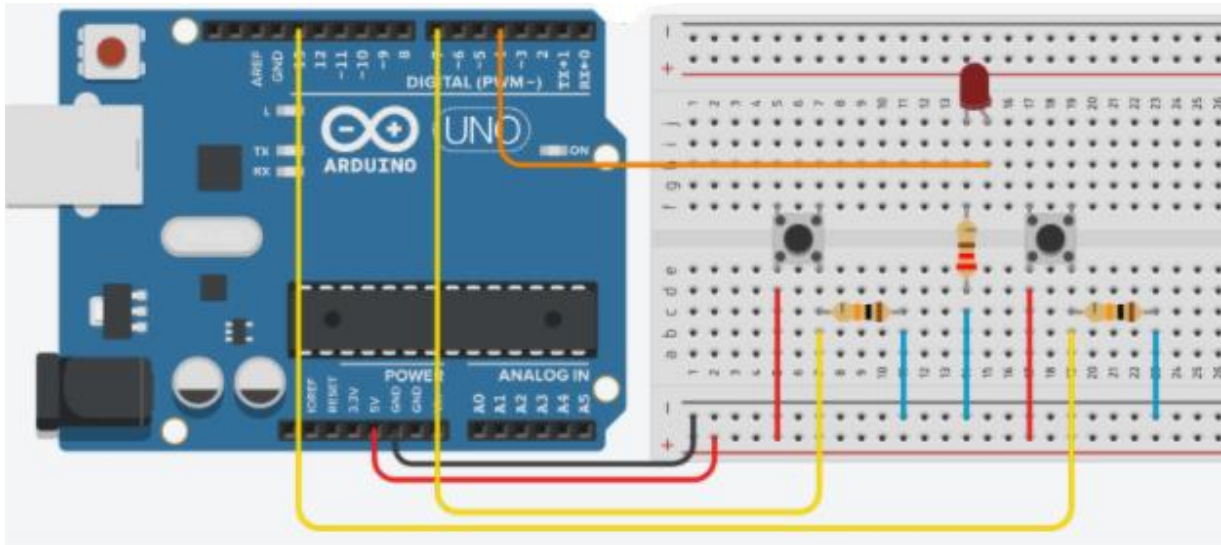
```
}  
void loop()  
{  
    if (digitalRead(buton) == HIGH)    // The button is ON,  
        digitalWrite(led, HIGH);        // Led is ON  
    else                                // If not,  
        digitalWrite(led, LOW);          // Led is OFF.  
}
```



### Circuit 8:

#### Circuit title: 2 Buttons versus 1 LED

**Circuit Explanation:** One button turns on the Led, another button turn off the Led.



#### **/\* 2 buttons versus 1 LED**

Buttons are connected with 10K resistors in series. **\*/**

```
int led = 4;           // The pin-4 is assigned as "led"
int button1 = 7;       // The pin-7 is assigned as "button1"
int button2 = 13;      // The pin-13 is assigned as "button2"


void setup()
{
  pinMode(led, OUTPUT);           // led=Output
  pinMode(button1, INPUT);        // button1=Input
  pinMode(button2, INPUT);        // button2=Input
}

void loop()
{
  if (digitalRead(button1) == HIGH) // IF "button1" is ON (active),
    digitalWrite(led, HIGH);        // Led is ON.
  if (digitalRead(button2) == HIGH) // IF "button2" is ON (active),
    digitalWrite(led, LOW);         // Led is OFF.  }
```



## HOW-TO Use the ARDUINO SERIAL MONITOR

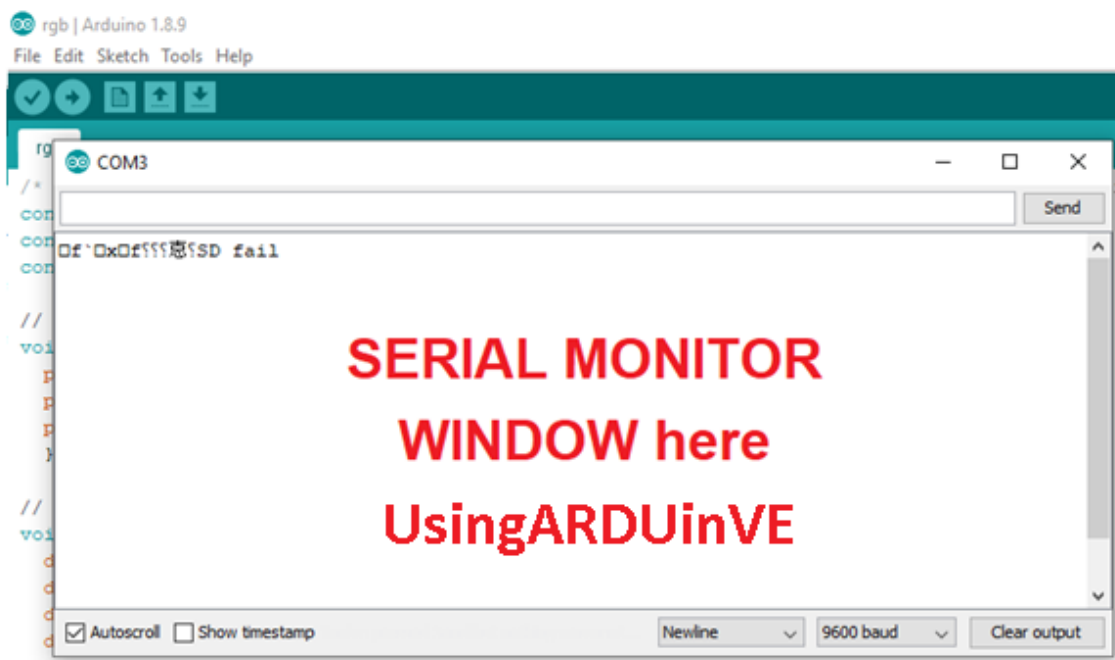
The Arduino IDE has a feature that can be a great help in debugging sketches or controlling Arduino from your computer's keyboard.

**The Serial Monitor** is a separate pop-up window that acts as a separate terminal that communicates by receiving and sending Serial Data. See the icon on the far right of the image here .

Serial Data is sent over a single wire (but usually travels over USB in our case) and consists of a series of 1's and 0's sent over the wire. Data can be sent in both directions (In our case on two wires).

You will use the Serial Monitor to debug Arduino Software Sketches or to view data sent by a working Sketch. You must have an Arduino connected by USB to your computer to be able to activate the Serial Monitor.

Open the Serial Monitor by clicking on the Serial Monitor box in the IDE. It should look like the screenshot below. Make SURE the baud (speed) is set to 9600. It is located in the bottom right corner. (The important thing is that it is set the same in our program and here. Since the default here is 9600, we set our





## Main Commands to Use Serial Monitor

**Serial.begin();** Sets the data rate in bits per second (baud) for serial data transmission. For communicating with Serial Monitor, make sure to use one of the baud rates listed in the menu at the bottom right corner of its screen. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

Syntax: `Serial.begin(speed);`

Example: `Serial.begin(9600);`

**Serial.print();** Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example:

`Serial.print(78; )` gives "78"

`Serial.print('N');` gives "N"

`Serial.print("Hello Arduino");` gives "Hello Arduino"

**Serial.println();** Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as `Serial.print()`.

`Serial.println(val);`

`Serial.println(val, format);`

`Serial.println(13, BIN);` gives "1101", binary value of 13 on Serial Monitor.

**NOTE:** The only difference between `Serial.print` and `Serial.println` is that `Serial.println` means that the next thing sent out the serial port after this one will start on the next line. There is a third new thing you may have noticed. There is something in quotes ( " "). This is called a string.



Co-funded by the  
Erasmus+ Programme  
of the European Union

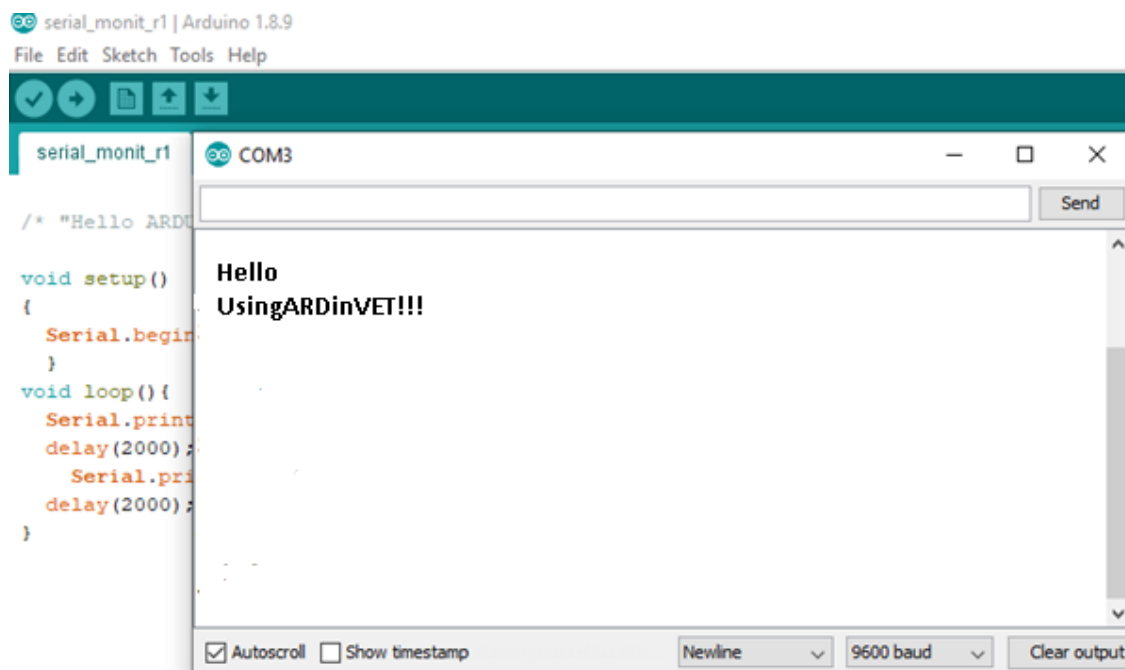


### Circuit 9:

**Circuit title:** "Hello UsingARDinVET" Application in Serial Monitor"

**Circuit Explanation:** Hello UsingARDinVET will be seen on the serial Monitor.

```
/*    "Hello UsingARDinVET" Application in Serial Monitor */  
void setup()  
{  
    Serial.begin(9600);    // Serial communication speed  
}  
void loop()  
{  
    Serial.println(" HELLO ");  
    delay(2000);  
    Serial.println(" UsingARDinVET!!!");  
    delay(2000);  
}
```

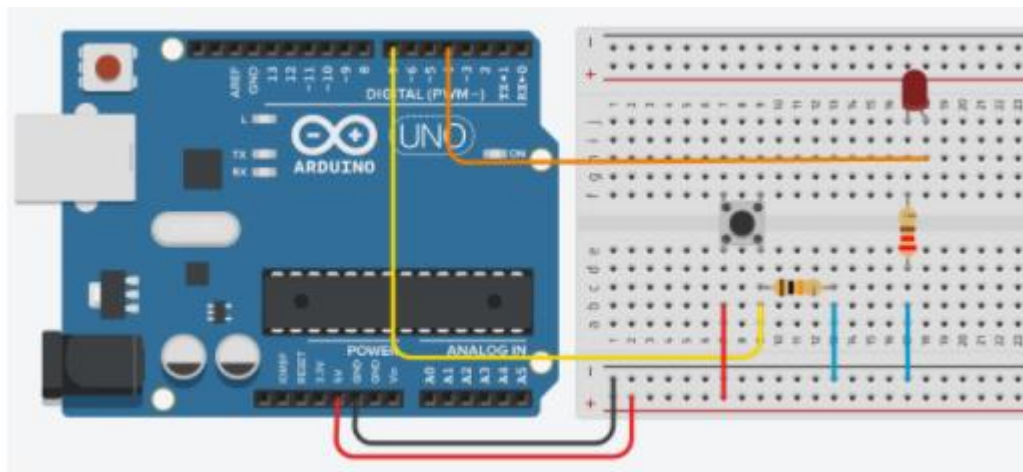




## Circuit 10:

**Circuit title:** " Button Status Analysis on Serial Monitor"

**Circuit Explanation:** If the button is pressed, "Led is ligthing, LED=ON" message is on the serial monitor and the Led is lighting. If the buton is not pressed, ("Button is not pressed") message is on the serial monitor and the the Led is lighting.



**/\* Button Status Analysis on Serial Monitor \*/**

```
void setup()
{
  pinMode(4, OUTPUT);
  pinMode(7, INPUT);
  Serial.begin(9600);
}

void loop()
{
  if (digitalRead(7) == HIGH)    // Read the digital signal on Pin-7
  {
    digitalWrite(4, HIGH);
    Serial.println("Led is ligthing, LED=ON");
    delay(250);
  }
  else                          // If the previous condition is not met.
  {
    digitalWrite(4, LOW);
    Serial.println("Button is not pressed");
  }
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
    delay(1000);  
  }  
}  
//The view of the serial monitor is seen below.
```

---

```
Button is not pressed  
Button is not pressed  
Led is ligthing, LED=ON  
Led is ligthing, LED=ON  
Led is ligthing, LED=ON  
Led is ligthing, LED=ON  
Led is ligthing, LED=ON  
Button is not pressed
```

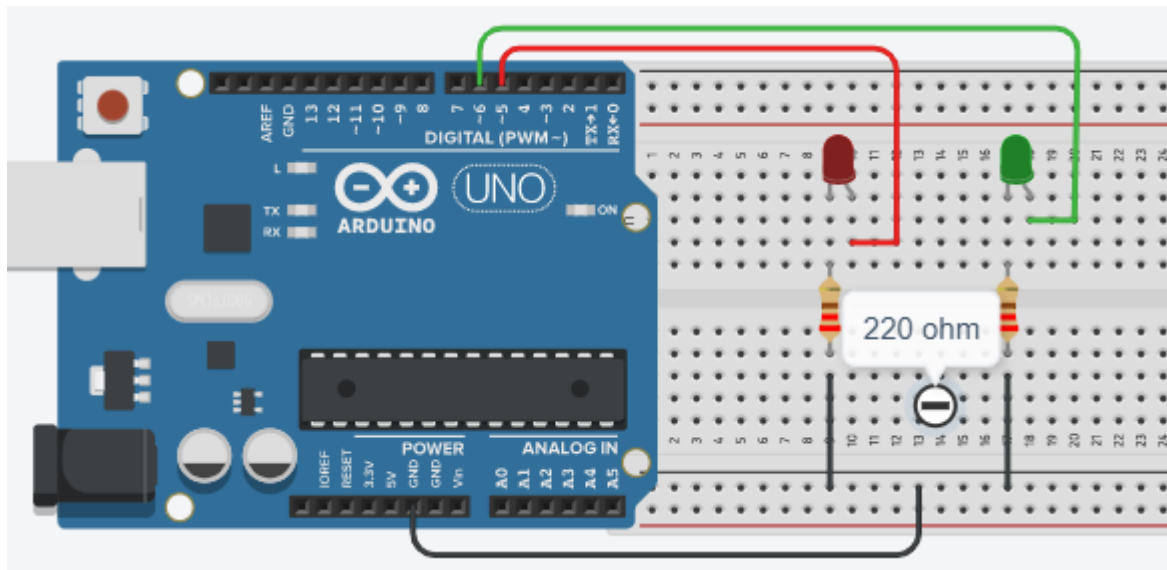
---



## Circuit 11:

**Circuit title:** " Sending data from the serial monitor."

**Circuit Explanation:** If the "A" character on the keyboard is pressed, led1 is ON.  
If the "3" character on the keyboard is pressed, led2 is ON.  
If the "-" character on the keyboard is pressed, led1 and led2 is OFF.



```
/* Sending data from the serial monitor */
```

```
char character;  
#define led1 5  
#define led2 6
```

```
void setup() {  
  pinMode(led1, OUTPUT);  
  pinMode(led2, OUTPUT);  
  Serial.begin(9600);  
  Serial.println("--- enter a character ---");  
  Serial.println("-----");  
}
```

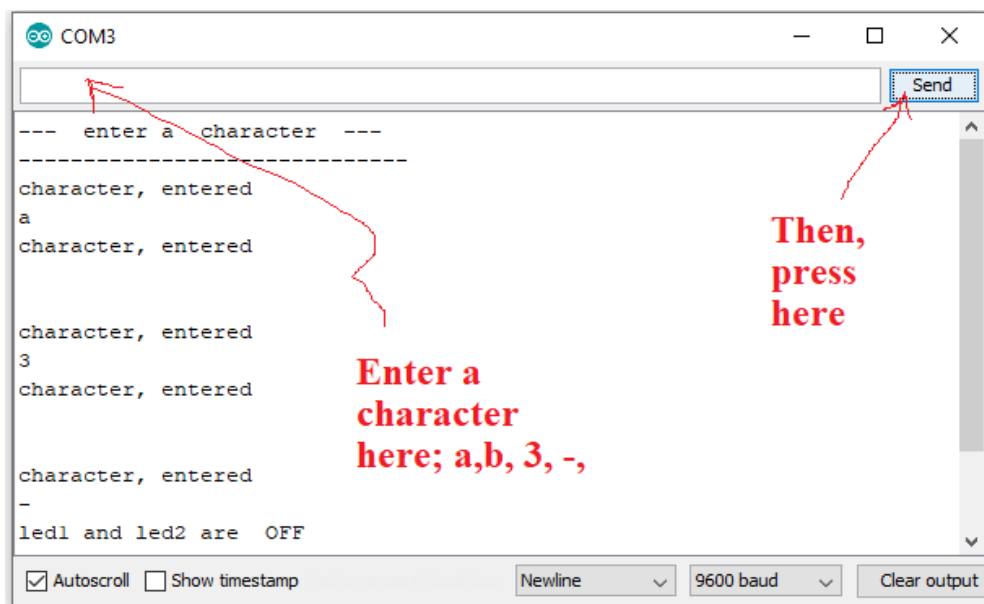
```
void loop()  
{  
  if (Serial.available()>0)  
  {  
    character=Serial.read();  
    Serial.println ("character, entered ");  
    Serial.println (character);
```

```
    if (character=='A') digitalWrite (led1, 1);
```

“



```
if (character=='a') digitalWrite (led1, 1);  
  
if (character=='3') digitalWrite (led2, 1);  
  
if (character=='-')  
{  
  digitalWrite(led1,0);  
  digitalWrite(led2,0);  
  Serial.println ("led1 and led2 are OFF");  
}  
}  
}
```

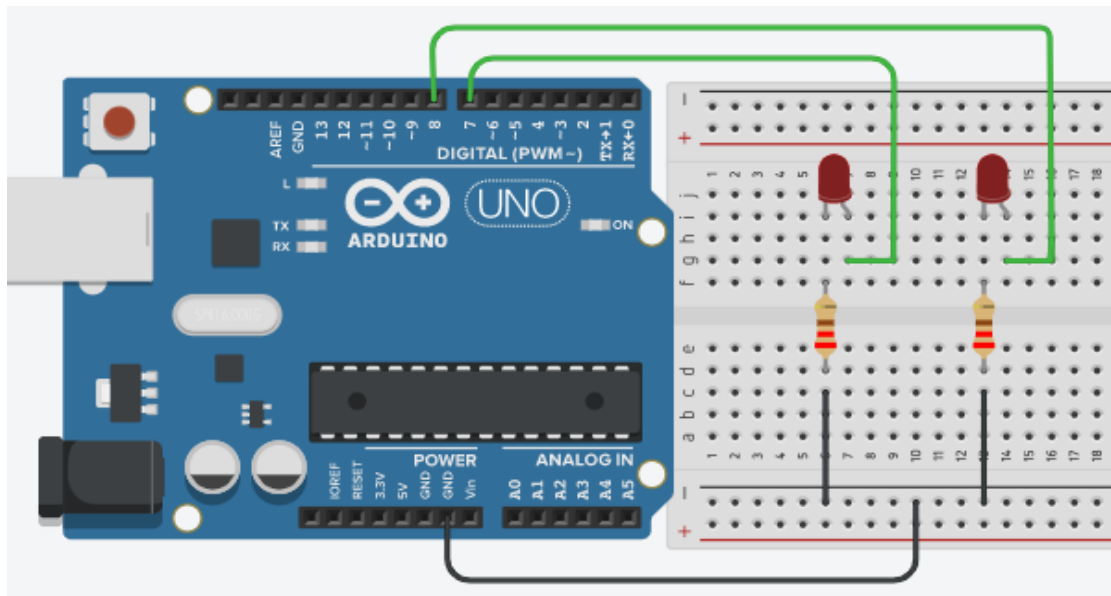




## Circuit 12:

**Circuit title:** "Learning the FOR Command"

### Circuit Explanation:



```
/*  "Learning the FOR Command"  */
```

```
int led1 = 7;  
int led2 = 8;
```

```
void setup() {  
  Serial.begin(9600);  
  pinMode(7,OUTPUT);  
  pinMode(8,OUTPUT);  
}
```

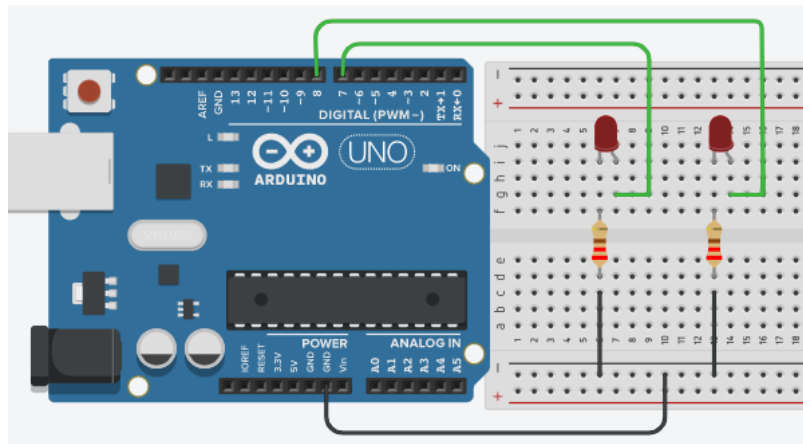
```
void loop()  
{  
  for(int i=1; i<6; i++)  
  {  
    Serial.println(i);  
    digitalWrite(led1, 1);  
    digitalWrite(led2, 1);  
    delay(1000);  
  
    digitalWrite(led1, 0);  
    digitalWrite(led2, 0);  
    delay(1000);  
  }  
}
```



**Circuit 13: Circuit title: " FOR Command versus Serial Monitor"**

**Circuit Explanation:** In this application, the LEDs will blink 6 times. Numbers 1, 2, 3, 4, 5,6 will appear on the serial monitor. Then it will be entered in the while loop by exiting the for loop. And the program will stop here. To restart, the reset button must be pressed.

**Here, we are using the same circuit as the previous one.**



**/\* "FOR Command versus Serial Monitor" \*/**

```
int led1 = 7;  
int led2 = 8;
```

```
void setup()      {  
  Serial.begin(9600);  
  pinMode(7,OUTPUT);  
  pinMode(8,OUTPUT);  
}
```

```
void loop() {
```

```
  for(int i=1; i<=6; i++)          // the value of i = 1, 2,3, 4,5, 6
```

```
  {
```

```
    Serial.println(i);              // Write the values of i, on the serial monitor  
    digitalWrite(led1, 1);  
    digitalWrite(led2, 1);  
    delay(1000);  
    digitalWrite(led1, 0);  
    digitalWrite(led2, 0);  
    delay(1000);  
  }
```

```
  while(1);                        // the for loop doesn't get into an infinite loop.
```

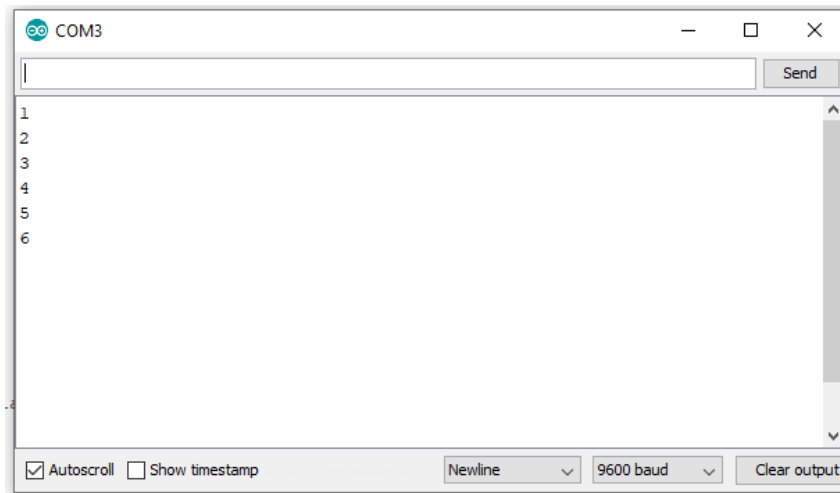
```
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



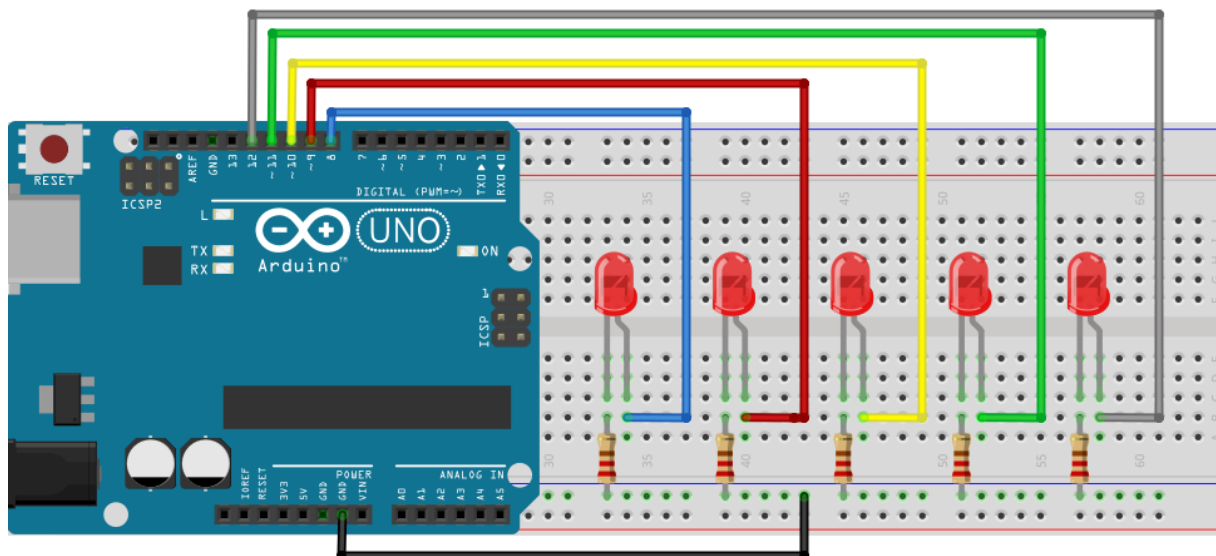
**NOTE:** To restart, the reset button must be pressed.



#### Circuit 14:

**Circuit title:** " Knight Rider with 5 Leds by using the FOR Command "

**Circuit Explanation:**



**/\* Knight Rider with 5 Leds by using the FOR Command \*/**

```
void setup() {  
  pinMode(8, OUTPUT);  
  pinMode(9, OUTPUT);  
  pinMode(10, OUTPUT);
```



```
pinMode(11, OUTPUT);
pinMode(12, OUTPUT);  }

void loop() {

  for (int b=8; b<=12; b++)          // Upcounter  starts here

  {

    digitalWrite(b, HIGH);
    delay(150);
    digitalWrite (b, LOW);
    delay(150);

  }

  for (int b=12; b>=8; b--)          // Downcounter  starts here

  {

    digitalWrite (b, HIGH);
    delay(150);
    digitalWrite (b, LOW);
    delay(150);

  }
}
```



### Circuit 15:

**Circuit title:** " Producing random colors with RGB led, Using the switch/case command"

**Circuit Explanation:** Here, Random command and Switch/Case command are used in the application. In this circuit, red, green, blue colors will be obtained randomly.

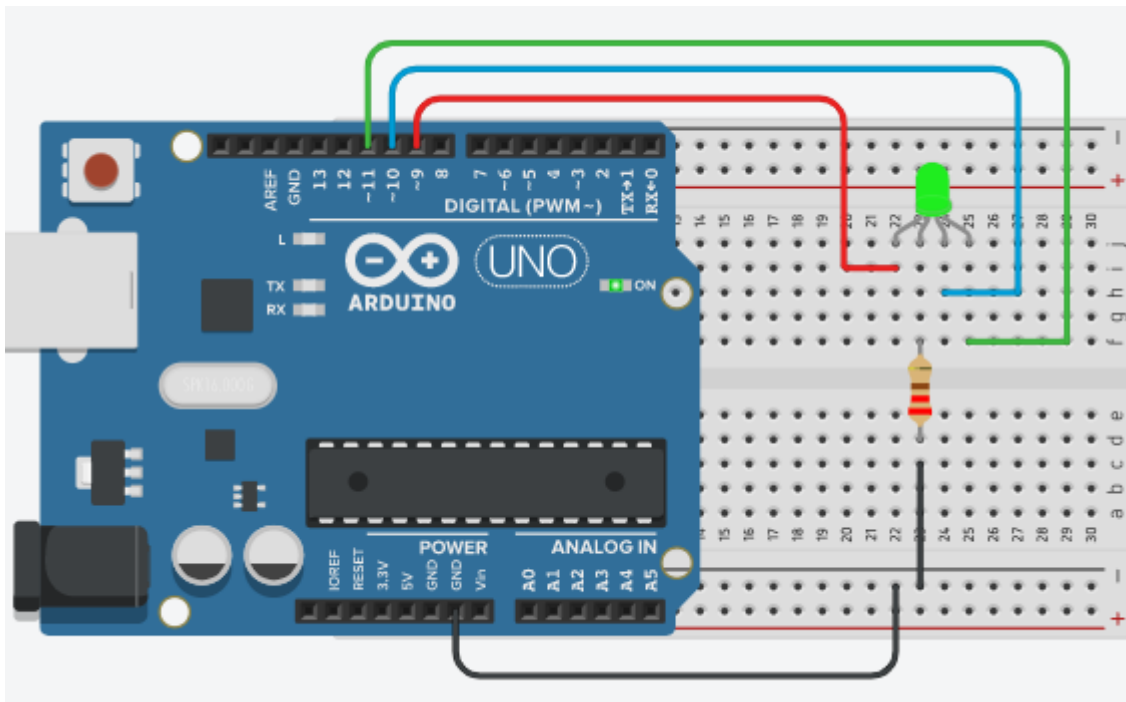
#### NOTE:

**random() :** The random function generates random numbers.

Syntax: random(max), random(min, max)

min: lower bound of the random value, (optional).

max: upper bound of the random value.



**/\* Producing random colors with RGB led, Using the switch/case command \*/**

```
#define R 9
#define G 10
#define B 11
int colour;
int dly=3000;
```

```
void setup() {
  pinMode(R, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(B, OUTPUT);
}
```



```
Serial.begin(9600);
}

void loop()
{
  colour=random(4);
  Serial.println(colour);

  switch(colour) {

    case 0:
      digitalWrite (R, 1);          //RedLED=ON
      digitalWrite (G, 0);
      digitalWrite (B, 0);
      delay(dly);
      break;

    case 1:
      digitalWrite (R, 0);          //GreenLED=ON
      digitalWrite (G, 1);
      digitalWrite (B, 0);
      delay(dly);
      break;

    case 2:
      digitalWrite (R, 0);          //BlueLED=ON
      digitalWrite (G, 0);
      digitalWrite (B, 1);
      delay(dly);
      break;

    case 3:
      digitalWrite (R, 0);          //No colour
      digitalWrite (G, 0);
      digitalWrite (B, 0);
      delay(dly);
      break;
  }
}
```

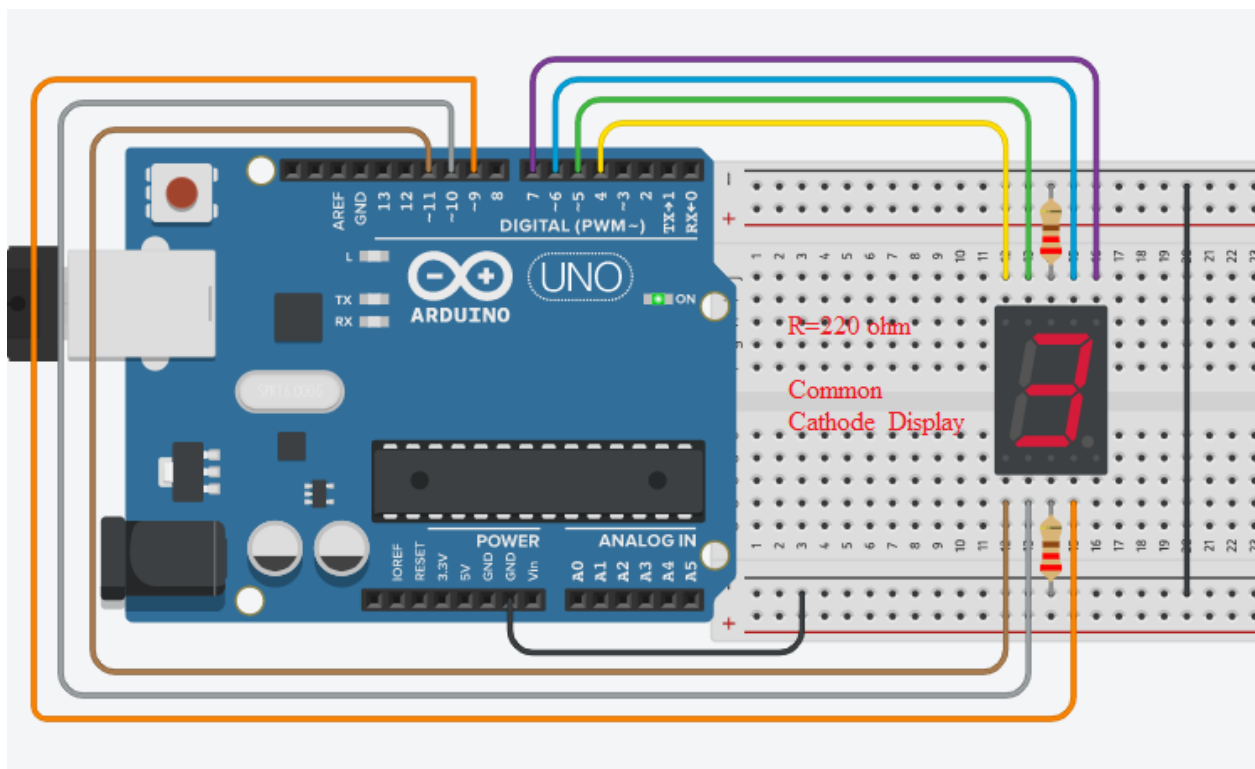


### Circuit 16:

**Circuit title:** " 0-9 UpCounter with 7segment Common-Anode Display ”

**Circuit Explanation:** In order to see a number on the Display, the corresponding LED is lit from the 7 LEDs, represented by the letters a, b, c, d, e, f, g.

**NOTE:** A seven-segment display is a form of electronic display device for displaying decimal numerals.



/\* " 0-9 UpCounter with 7segment Common-Cathode Display " \*/



```
int    a=6, b=7, c=9, d=10, e=11, f=5, g=4;
int number;
```

```
void setup() {
pinMode(a, OUTPUT);
pinMode(b, OUTPUT);
pinMode(c, OUTPUT);
pinMode(d, OUTPUT);
pinMode(e, OUTPUT);
pinMode(f, OUTPUT);
pinMode(g, OUTPUT);
}
```

```
void loop() {
for(number=0; number<=9; number++) {
delay(1000);
switch(number) {
```

```
    case 0:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, HIGH);
digitalWrite (f, HIGH);
digitalWrite (g, LOW);
break;
```

```
    case 1:
digitalWrite (a, LOW);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, LOW);
digitalWrite (e, LOW);
digitalWrite (f, LOW);
digitalWrite (g, LOW);
break;
```

```
    case 2:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, LOW);
digitalWrite (d, HIGH);
digitalWrite (e, HIGH);
digitalWrite (f, LOW);
digitalWrite (g, HIGH);
break;
```

```
    case 3:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, LOW);
digitalWrite (f, LOW);
digitalWrite (g, HIGH);
break;
```

```
    case 4:
digitalWrite (a,LOW );
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, LOW);
digitalWrite (e, LOW);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;
```

```
    case 5:
digitalWrite (a, HIGH);
digitalWrite (b, LOW);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, LOW);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;
```



case 6:

```
digitalWrite (a, HIGH);  
digitalWrite (b, LOW);  
digitalWrite (c, HIGH);  
digitalWrite (d, HIGH);  
digitalWrite (e, HIGH);  
digitalWrite (f, HIGH);  
digitalWrite (g, HIGH);  
break;
```

case 7:

```
digitalWrite (a, HIGH);  
digitalWrite (b, HIGH);  
digitalWrite (c, HIGH);  
digitalWrite (d, LOW);  
digitalWrite (e, LOW);  
digitalWrite (f, LOW);  
digitalWrite (g, LOW);  
break;
```

case 8:

```
digitalWrite (a, HIGH);
```

```
digitalWrite (b, HIGH);  
digitalWrite (c, HIGH);  
digitalWrite (d, HIGH);  
digitalWrite (e, HIGH);  
digitalWrite (f, HIGH);  
digitalWrite (g, HIGH);  
break;
```

case 9:

```
digitalWrite (a, HIGH);  
digitalWrite (b, HIGH);  
digitalWrite (c, HIGH);  
digitalWrite (d, HIGH);  
digitalWrite (e, LOW);  
digitalWrite (f, HIGH);  
digitalWrite (g, HIGH);  
break;  
}  
}  
}
```

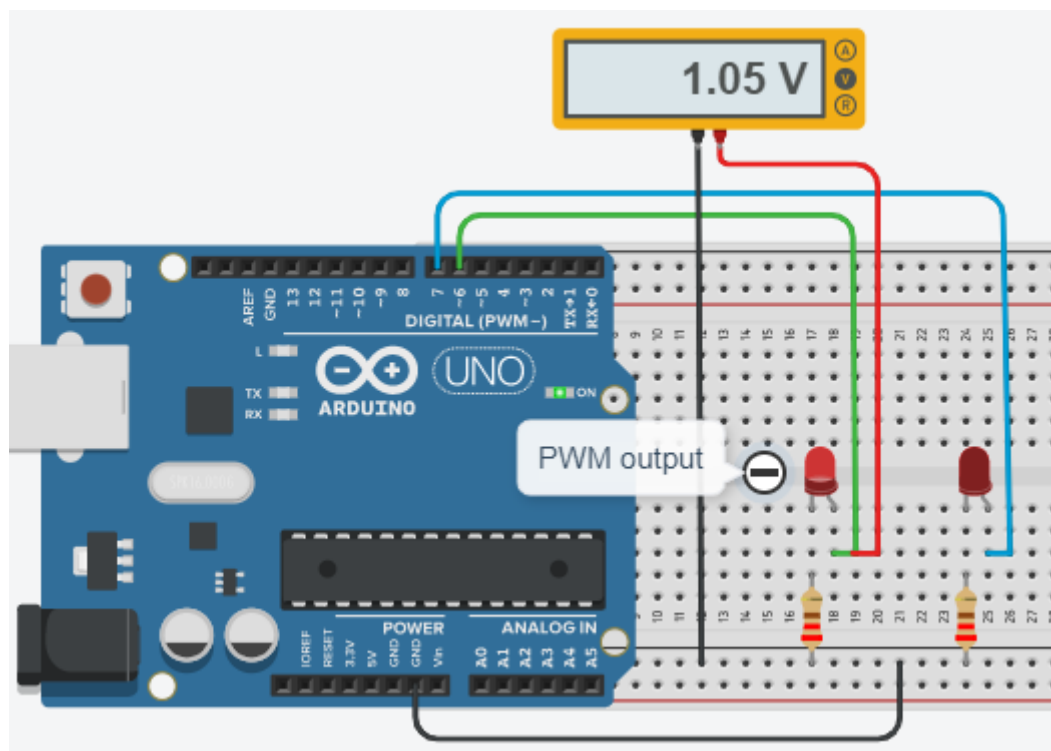
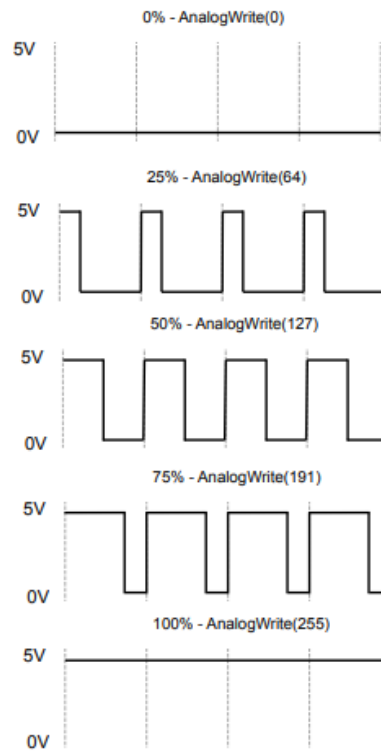
## Circuit 17:

**Circuit title:** " Using the PWN Technique"

**Circuit Explanation:** In practice, pin-6 as a PWM output and pin-7 as a digital output are used. Thus, it is aimed to observe the difference between them. Applications such as led brightness adjustment, motor speed control can be performed with the PWM method.

**NOTE:** The Arduino supports PWM (on certain pins marked with a tilde(~) on your Arduino board - pins 3, 4, 5, 9, 10 and 11) at 500Hz. (500 times a second.) You can give it a value between 0 and 255. 0 means that it is never 5V. 255 means it is always 5V. To do this you make a call to analogWrite() with the value. The ratio of "ON" time to total time is called the "duty cycle". A PWM output that is ON half the time is said to have a duty cycle of 50%.

You can think of PWM as being on for  $x/255$  where  $x$  is the value you send with analogWrite(). Below is an example showing what the pulses look like:





**`/* Learning the PWN Technique by using analogWrite() */`**

**`#define led1 6`**

**`#define led2 7`**

**`void setup() {  
pinMode(led1, OUTPUT);  
pinMode(led2, OUTPUT);  
}`**

**`void loop() {`**

**`analogWrite(led1, 60);`**      // The value of 60 is sent to Led1 pin, i.e 1.05 V.

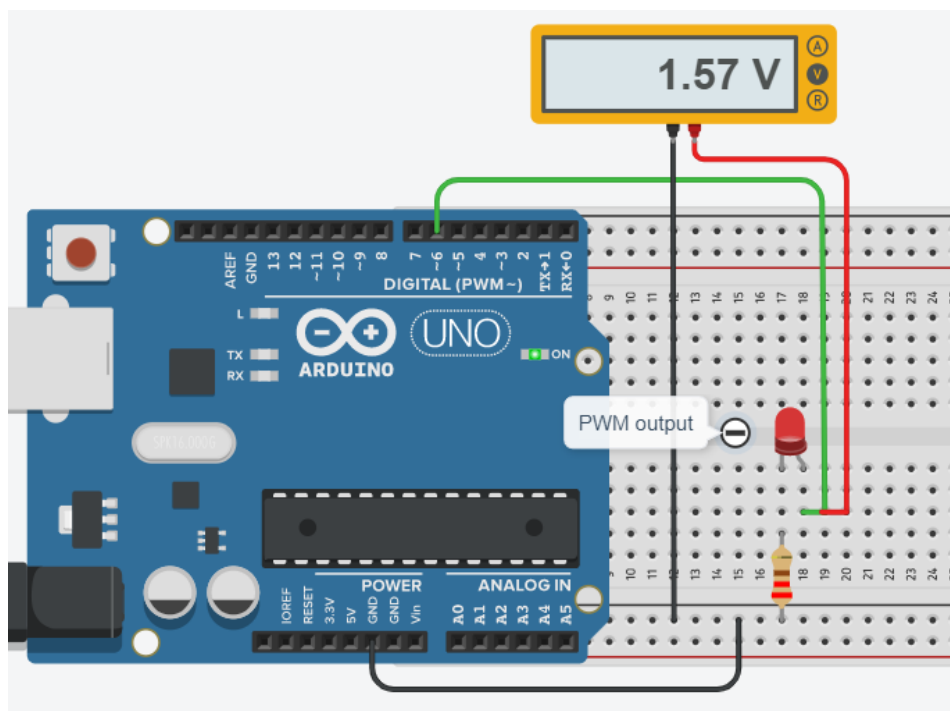
**`analogWrite(led2,60);`**      // The value of 60 is sent to Led2 pin, i.e 1.05 V.

**`delay (100);`**                      // only, the led1 lights.  
**`}`**

### Circuit 18:

**Circuit title:** " To change the brightness of an LED from minimum to maximum."

**Circuit Explanation:** By using the PWM technique, the brightness of an LED from minimum to maximum is changed. Here, analogWrite () and the for commands will be used together.





**/\* To change the brightness of an LED from minimum to maximum \*/**

```
#define led1 6
int a;

void setup() {
  Serial.begin(9600);
  pinMode(led1, OUTPUT); }

void loop() {

  for (a=0; a<=255; a++) {

    Serial.println(a);

    analogWrite(led1, a);

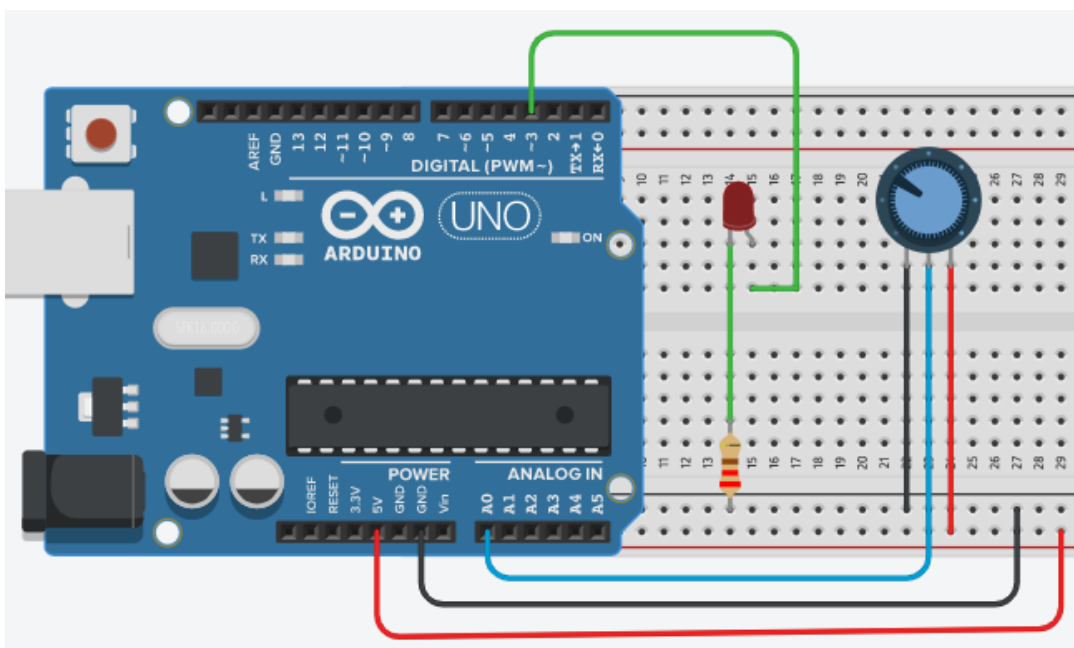
    delay (100);
  } }
```

**NOT:** Values ranging from 0 to 5 Volts are displayed on the voltmeter. Counting numbers from 0 to 255 are displayed on the serial monitor.

### Circuit 19:

**Circuit title:** " Potentiometer fades led."

**Circuit Explanation:** By using the potentiometer (10K), the brightness of an LED from minimum to maximum is changed. Here, the function of map() is used.





### Note:

#### map() Function:

Re-maps a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The constrain() function may be used either before or after this function, if limits to the ranges are desired.

The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged.

Syntax: map(value, fromLow, fromHigh, toLow, toHigh);

Example: val = map(val, 0, 1023, 0, 255);

```
/* Fotentiometer fades led */
```

```
int LED_PIN = 3;
```

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

```

```
void loop() {
```

```
  // reads the input on analog pin A0 (value between 0 and 1023)
```

```
  int analogValue = analogRead(A0);
```

```
  // scales it to brightness (value between 0 and 255)
```

```
  int brightness = map(analogValue, 0, 1023, 0, 255);
```

```
  // sets the brightness LED that connects to pin 3
```

```
  analogWrite(LED_PIN, brightness);
```

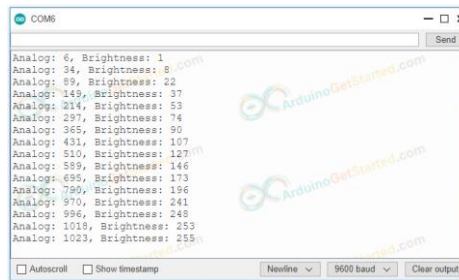
```
  // print out the value
```

```
  Serial.print("Analog: ");
```

```
  Serial.print(analogValue);
```



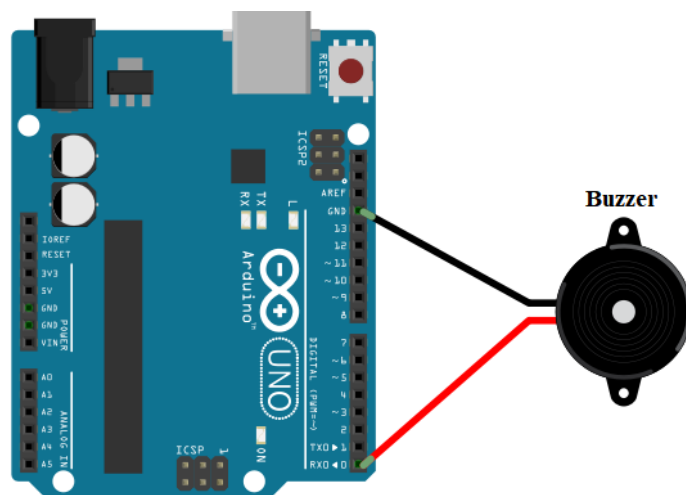
```
Serial.print(" Brightness: ");  
  
Serial.println(brightness);  
  
delay(100);  
// Serial Monitor screen i below  
}
```



## Circuit 20:

**Circuit title:** " To use a buzzer"

**Circuit Explanation:** A buzzer is an audio signal device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers in the industry is as an alarm devices, which makes a buzzing or beeping noise while need buzzing.



**/\* To use a buzzer in Arduino circuits\*/**

```
#define buzzer_pin 0
```

```
void setup() {
```



```
pinMode(buzzer_pin, OUTPUT);    }

void loop()    {

    digitalWrite(buzzer_pin, HIGH);

    delay(500);

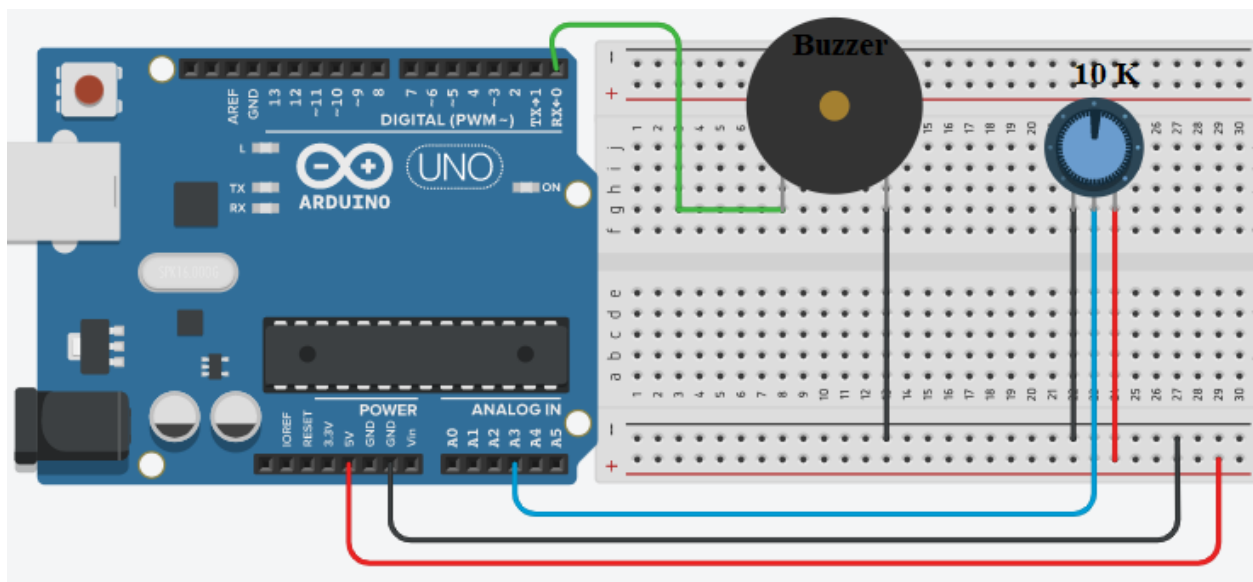
    digitalWrite(buzzer_pin, LOW);

    delay(500);    }
```

### Circuit 21:

**Circuit title:** " To control a buzzer with Potentiometer ”

**Circuit Explanation:** “ When the the value of potentiometer is higher than 500, the buzzer sounds.”



**/\* To control a buzzer with Potentiometer \*/**

```
const int POT_PIN = A3;           // Arduino pin connected to Pot pin
const int BUZZER_PIN = 0;         // Arduino pin connected to Buzzer's pin
const int ANALOG_THRESHOLD = 500;
int analogValue;
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
void setup() {  
  
    pinMode(BUZZER_PIN, OUTPUT);    // set arduino pin to output mode  
  
}  
  
void loop() {  
  
    analogValue = analogRead(POT_PIN);    // read the input on analog pin  
  
    if(analogValue > ANALOG_THRESHOLD)  
        digitalWrite(BUZZER_PIN, HIGH);    // turn on Buzzer  
  
    else  
        digitalWrite(BUZZER_PIN, LOW);    // turn off Buzzer  
  
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



## **Erasmus+ KA210-VET**

### **Small-scale partnerships in vocational education and training**

**Project Title: “Using Arduinos in Vocational Training”**

**Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

### **LCD Module and Training KIT**





## LCD Module and Training KIT

In this module we want to explain how to display status messages or sensor readings of Arduino on LCD displays . They are extremely common and a fast way to add a readable interface to your project.

An LCD is short for Liquid Crystal Display. It is a display unit which uses liquid crystals to produce a visible image. When current is applied to this special kind of crystal, it turns opaque blocking the backlight that lives behind the screen. As a result that particular area will become dark compared to other. And that's how characters are displayed on the screen.

### Interfacing 16×2 Character LCD Module with Arduino

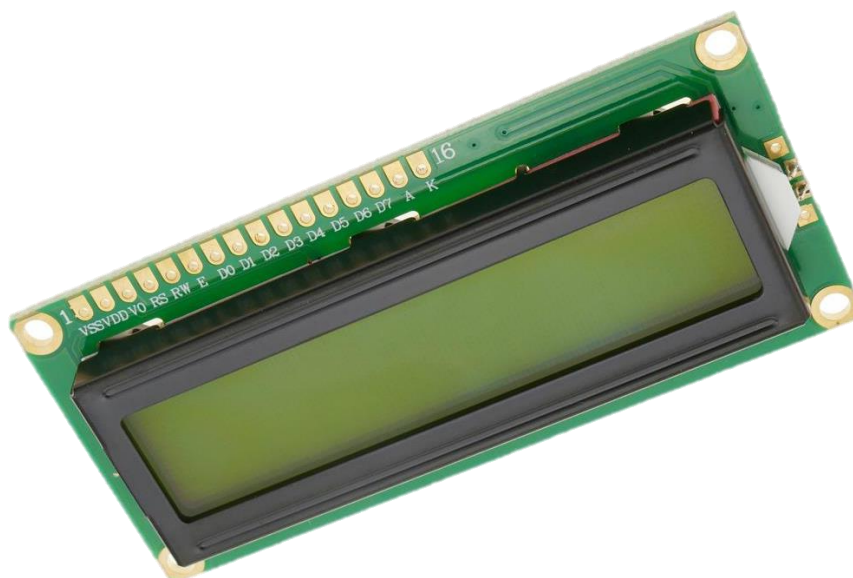
There are different kind of LCD displays that you can connect to your Arduino. The most common one is based on parallel interface LCD controller chip from Hitachi called the HD44780.

These LCDs are ideal for displaying text/characters only, hence the name 'Character LCD'. The display has an LED backlight and can display 32 ASCII characters in two rows with 16 characters on each row.

There is a little rectangles for each character on the display, each of these rectangles is a grid of 5×8 pixels.

Although they display only text, they do come in many sizes and colors: for example, 16×1, 16×4, 20×4, with white text on blue background, with black text on green and many more.

The Arduino community has already developed a library to handle HD44780 LCDs (**LiquidCrystal Library**); so we'll have them interfaced in a few time.





Below is the LCD pinout:

- **VSS:** is a ground pin and should be connected to the ground of Arduino;
- 
- **VDD:** connected to 5 V;
- **VD:** for contrast adjustment (connected to the central pin of the potentiometer) ;
- **RS:** controls in which lcd memory zone the sent data are stored;
- **R/W:** pin to select read/write mode;
- **E:** if enabled, allows the LCD module to perform special instructions;
- **D0 to D7:** data transmission;
- **A and K:** anode and cathode to provide backlight to the LCD module.

### Arduino - LCD Functions (Commands)

LiquidCrystal lcd() - Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters. Syntax:

LiquidCrystal(rs, enable, d4, d5, d6, d7)

LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)

LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)

LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)

Parameters:

rs: the number of the Arduino pin that is connected to the RS pin on the LCD

rw: the number of the Arduino pin that is connected to the RW pin on the LCD (optional)

enable: the number of the Arduino pin that is connected to the enable pin on the LCD

d0, d1, d2, d3, d4, d5, d6, d7: the numbers of the Arduino pins that are connected to the corresponding data pins on the LCD. d0, d1, d2, and d3 are optional; if omitted, the LCD will be controlled using only the four data lines (d4, d5, d6, d7).

lcd.begin() - Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display. begin() needs to be called before any other LCD library commands. Syntax:  
lcd.begin(cols, rows)



Parameters:

lcd: a variable of type LiquidCrystal

cols: the number of columns that the display has

rows: the number of rows that the display has

lcd.print() - Prints text to the LCD. Syntax:

```
lcd.print(data)
```

```
lcd.print(data, BASE)
```

Parameters:

lcd: a variable of type LiquidCrystal

data: the data to print (char, byte, int, long, or string)

BASE (optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

byte print() will return the number of bytes written, though reading that number is optional

lcd.setCursor() - Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed. Syntax:

```
lcd.setCursor(col, row)
```

Parameters:

lcd: a variable of type LiquidCrystal

col: the column at which to position the cursor (with 0 being the first column)

row: the row at which to position the cursor (with 0 being the first row)

lcd.clear(); - Clears the LCD screen and positions the cursor in the upper-left corner.

Syntax: lcd.clear()

Parameters: lcd: a variable of type LiquidCrystal

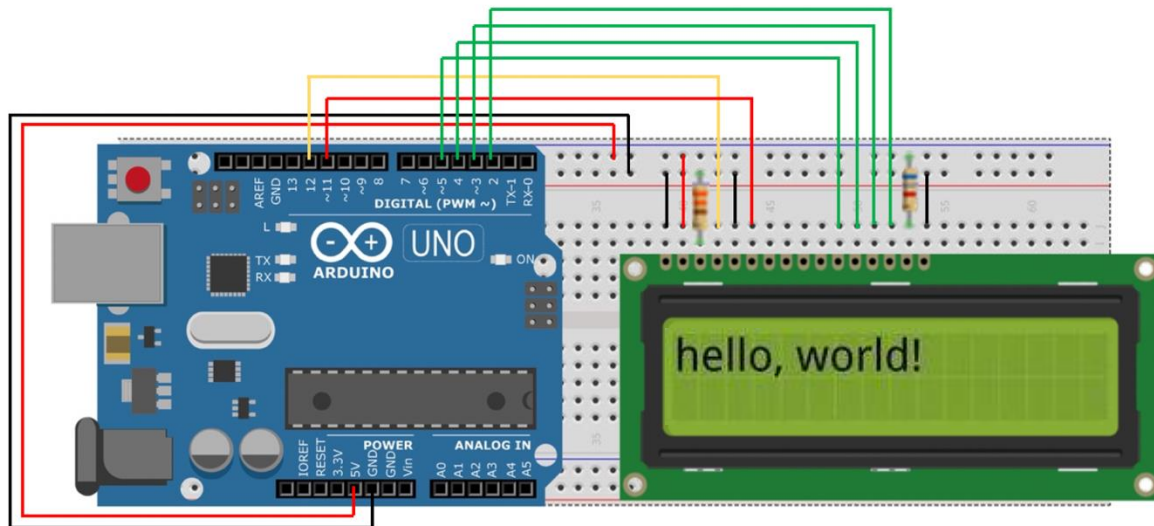


### Circuit 1:

**Circuit title:** "Print a message to the LCD"

**Circuit Explanation:** It is possible to connect an LCD display to the microcontroller and print the desired messages on the screen.

**Note:** We have to include the library LiquidCrystal.h to use the LCD functions described below.



```
/* Print a message */

#include <LiquidCrystal.h> // include the library code

//constants for the number of rows and columns in the LCD

const int numRows = 2;
const int numCols = 16;

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  lcd.begin(numCols, numRows);

  lcd.print("hello, world!"); // Print a message to the LCD.
}

void loop() {
}
```



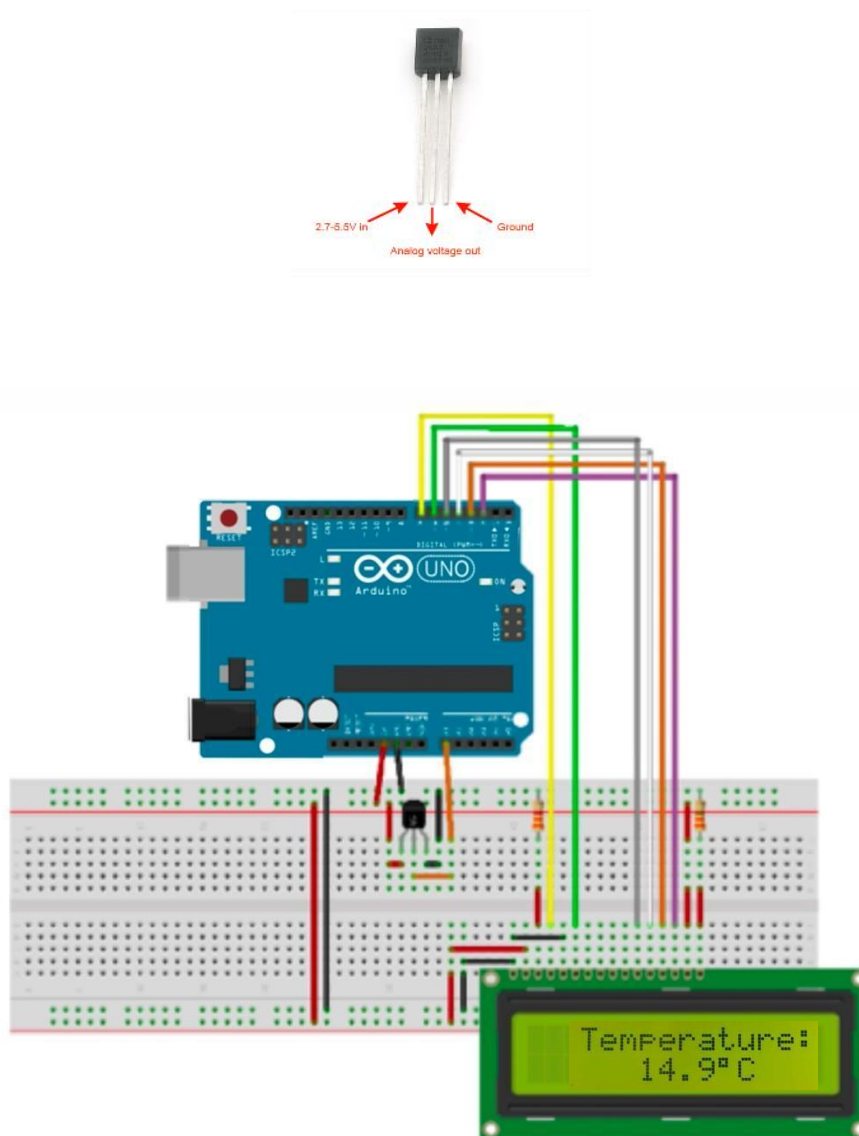
## Circuit 2:

**Circuit title:** "Digital Thermometer"

**Circuit Explanation:** To create a digital thermometer with Arduino. The temperature will be taken with the LM35 temperature sensor and must be displayed on an LCD display and updated every second.

The Vo pin of the LCD display adjusts the contrast of the characters displayed in the display. As mentioned above, it must be connected to a 3.3 k $\Omega$  resistance connected to GND. However, in some displays you may need to connect the Vo pin directly to GND

**Note:** the LM35 sensor has 3 terminals: one for the power supply, one for mass and one for the output of the voltage proportional to the detected temperature, which is equal to 10 mV for each degree centigrade, and is calibrated in degrees Celsius.



/\* Digital Thermometer \*/



```
#include <LiquidCrystal.h> //Library to drive LCD display

#define pin_temp A0 //Temperature sensor Vout foot connection pin

float temp = 0; //Variable in which the detected temperature will be stored

LiquidCrystal lcd(7, 6, 5, 4, 3, 2); //Initializing the library with LCD display pins

void setup()

{
  lcd.begin(16, 2); //Setting the number of columns and rows in the display
  LCD lcd.setCursor(0, 0); //Move the cursor over the first row (row 0) and the first
  column lcd.print ("Temperature:"); Print the message 'Temperature:' on the first line

  /*Imposed ADC Vref at 1.1V
  (for greater accuracy in temperature calculation)

  IMPORTANT: If you use Arduino Mega replace INTERNAL with INTERNAL1V1 */
  analogReference(INTERNAL);
}

void loop()
{
  /*calculate the temperature =====*/
  temp = 0;

  for (int i = 0; i < 5; i++) { //It executes the next statement 5 times
    temp += (analogRead(pin_temp) / 9.31); // It calculates temperature and sum at variable
    'temp'
  }
  temp /= 5; //It calculates the mathematical average of temperature values
  /*=====*/

  /*I see the temperature on the LCD display
  =====*/
  lcd.setCursor(0, 1); //lcd.print(temp); Move the cursor over the first column
  and the second row lcd.print(temp);
  // Mold on LCD display temperature

  lcd.print(" C"); // Mold a space and the 'C' font on the display
  /*=====*/
  delay(1000); //Delay by one second (it can be changed)
}
```

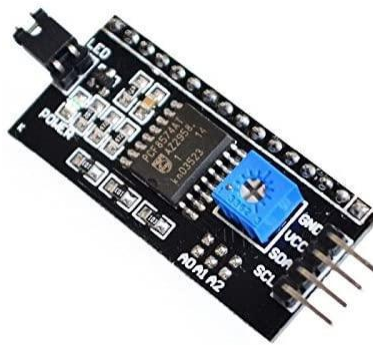


## Interfacing I2C LCD with Arduino

If you want to connect an LCD display with Arduino, you have to consume a lot of pins on the Arduino. Even in 4-bit mode, the Arduino still requires a total of seven connections, which is half of the available digital I/O pins.

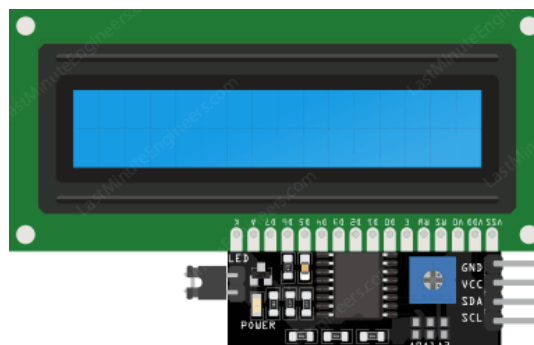
The solution is to use an LCD display that interfaces with I2C protocol. It only consumes two I/O pins which can be not even part of a digital I/O pin set and can be shared with other I2C devices as well.

The most diffuse I2C LCD display consists of a HD44780 based character LCD display and an I2C LCD adapter.



The most important part of the adapter is the 8-Bit I/O Expander chip – PCF8574. This chip converts the I2C data from an Arduino into the parallel data required by the LCD display. The board also comes with a small potentiometer to make fine adjustments to the contrast of the display. If you are using more than a device on the same I2C bus, you may need to set a different I2C address for the board, so that it does not conflict with another I2C device. For this reason, the board has three solder jumpers (A0, A1 and A2).

An I2C LCD has only 4 pins that interface it to the Arduino.



The pinout is as follows:

- **GND:** is a ground pin and should be connected to the ground of Arduino;



- **VCC:** supplies power to the module and the LCD. Connect it to the 5V output of the Arduino or a separate power supply;
- **SDA:** is a Serial Data pin. This line is used for both transmit and receive. Connect to the SDA pin on the Arduino;
- **SCL:** is a Serial Clock pin. This is a timing signal supplied by the Bus Master device. Connect to the SCL pin on the Arduino.

On the Arduino boards with the R3 layout, the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. They are also known as A5 (SCL) and A4 (SDA). Refer the below table to identify the correct pins depending on the Arduino model.

	SCL	SDA
Arduino Uno	A5	A4
Arduino Nano	A5	A4
Arduino Mega	21	20
Leonardo/Micro	3	2

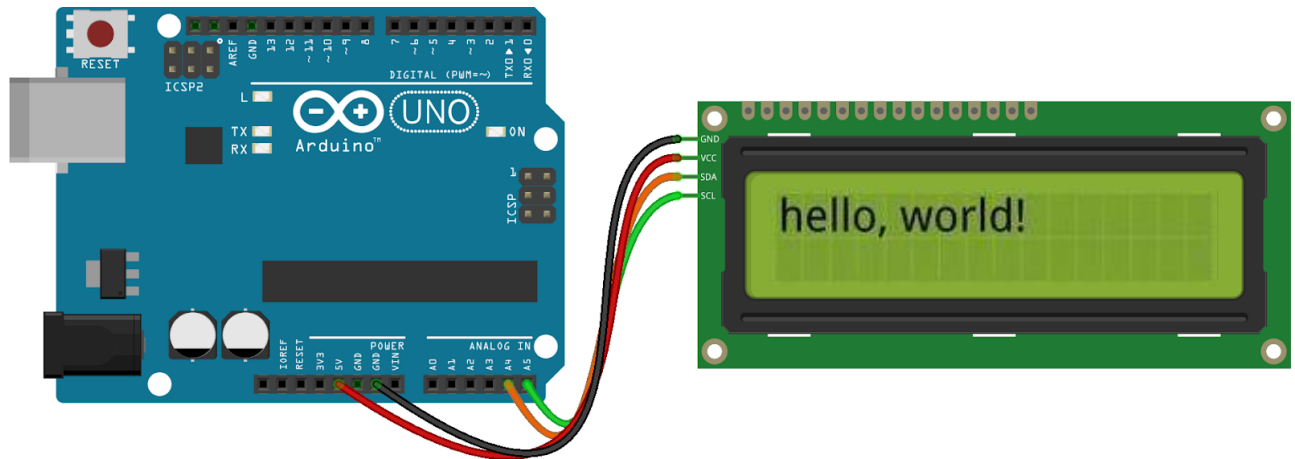


### Circuit 3:

**Circuit title:** "Print a message to the I2C LCD"

**Circuit Explanation:** It is possible to connect an LCD display to the microcontroller with only 4 pins and print the desired messages on the screen.

**Note:** We have to install a library called LiquidCrystal\_I2C. This library is an improved version of the LiquidCrystal library that comes packaged with your Arduino IDE.



```
/* Print a message on a I2C Display */
```

```
#include <LiquidCrystal_I2C.h>
```

```
LiquidCrystal_I2C lcd(0x3F,16,2); // set the LCD address to 0x3F for a 16 chars and 2 line display
```

```
void setup() {
```

```
  lcd.init();
```

```
  lcd.clear();
```

```
  lcd.backlight();    // Make sure backlight is on
```

```
  // Print a message on both lines of the LCD.
```

```
  lcd.setCursor(2,0); //Set cursor to character 2 on line 0
```

```
  lcd.print("Hello world!");
```

```
  lcd.setCursor(2,1); //Move cursor to character 2 on line 1
```



```
lcd.print("with I2C protocol!");  
  
}  
  
void loop() {  
  
}
```

### Interfacing OLED Graphic Display Module with Arduino

Another possibility to use the I2C protocol is choosing an OLED (Organic Light-Emitting Diode) display. They're super-light and thin, and produce a brighter and crisper picture.



An OLED display works without a backlight. This is why the display has such high contrast, extremely wide viewing angle and can display deep black levels. Absence of backlight significantly reduces the power required to run the OLED.

As we can see from the figure the pinout is the classic four-pin I2C interface already seen above. The Arduino community has already developed a few libraries to handle these OLED displays, such as Adafruit's SSD1306 library. To install the library navigate to the Sketch > Include Library > Manage Libraries... Wait for Library Manager to download libraries index and update list of installed libraries.

### Arduino - OLED Graphic Display Module Functions (Commands)

```
pinMode();  
display.begin()  
display.clearDisplay();
```

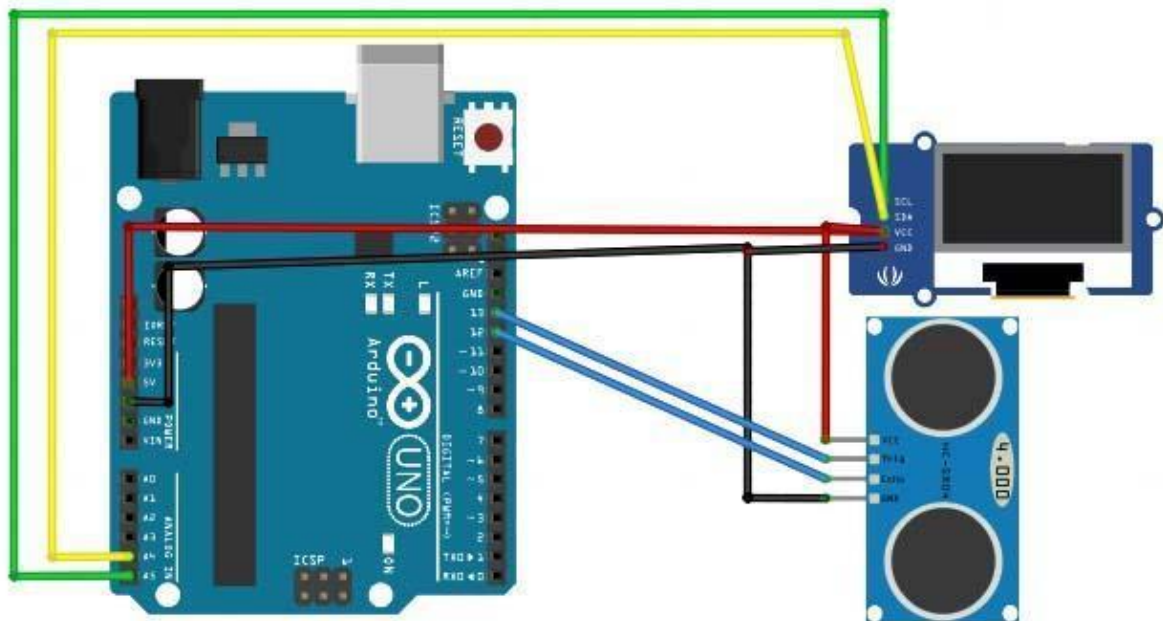


#### Circuit 4:

**Circuit title:** "Distance sensor"

**Circuit Explanation:** The distance will be taken with the HC-SR04 ultrasonic sensor and be displayed on a OLED display.

**Note:** There are only four pins that you need to worry about on the HC-SR04: VCC (Power), Trig (Trigger), Echo (Receive), and GND (Ground).



```
/* Distance sensor */
```

```
#include <SPI.h>      // this library allows you to communicate with SPI devices, with the  
                      // Arduino as the master device.
```

```
#include <Wire.h>     // this library allows you to communicate with I2C / TWI devices
```

```
#include <Adafruit_GFX.h>    // the OLED display's libraries
```

```
#include <Adafruit_SSD1306.h>
```

```
#define CommonSenseMetricSystem
```

```
#define trigPin 13    // define the pins of the sensor
```

```
#define echoPin 12
```



```
void setup() {
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //initialize with the I2C addr 0x3C
  (128x64)
  display.clearDisplay();
}

void loop() {
  long duration, distance;

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

  duration = pulseIn(echoPin, HIGH);
  distance = (duration/2) / 29.1;
  display.setCursor(22,20); //oled display setting cursor
  display.setTextSize(3); //size of the text
  display.setTextColor(WHITE); //if you write black it erases things
  display.println(distance); //print our variable
  display.setCursor(85,20);
  display.setTextSize(3);

  display.println("cm");

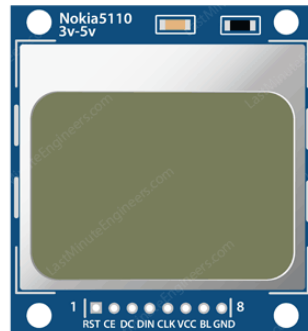
  display.display();

  delay(500);
  display.clearDisplay();
  Serial.println(distance);//debug
}
```



## Interfacing Nokia 5110 Graphic LCD Display with Arduino

You can interface Arduino with little LCDs similar to that Nokia used in their 3310 and 5110 cell phones. these displays are small (only about 1.5"), inexpensive, easy to use, fairly low power (as low as 6 to 7mA only) and can display text as well as bitmaps.



These are graphic display of 84×48 pixels. They interfaces to microcontrollers through a serial bus interface similar to SPI. The LCD also comes with a backlight in different colors such as red, green, blue and white. The backlight is nothing but four LEDs spaced around the edges of the display.

It has 8 pins that interface it to the Arduino, the pinout is as follows:

- **RST:** resets the display. It's an active low pin meaning; you can reset the display by pulling it low. You can also connect this pin to the Arduino reset so that it will reset the screen automatically;
- **CE(Chip Enable):** is used to select one of many connected devices sharing same SPI bus;
- **D/C(Data/Command):** pin tells the display whether the data it's receiving is a command or displayable data;
- **DIN:** is a serial data pin for SPI interface;
- **CLK:** is a serial clock pin for SPI interface;
- **VCC:** supplies power for the LCD which we connect to the 3.3V volts pin on the Arduino;
- **BL(Backlight):** controls the backlight of the display. To control its brightness, you can add a potentiometer or connect this pin to any PWM-capable Arduino pin;
- **GND:** is a ground pin and should be connected to the ground of Arduino.

You can connect data transmission pins to any digital I/O pin. The LCD has 3v communication levels, so we cannot directly connect these pins to the Arduino. One way is to add resistors inline with each data transmission pin. Just add 10kΩ resistors between the CLK, DIN, D/C, and RST



pins and a 1k $\Omega$  resistor between CE. The backlight(BL) pin is connected to 3.3V via 330 $\Omega$  current limiting resistor. You can add a potentiometer or connect this pin to any PWM-capable Arduino pin, if you wish to control its brightness.

The Arduino community has already developed a few libraries to handle these NOKIA displays, such as Adafruit's PCD8544 Nokia 5110 LCD library. To install the library navigate to the Sketch > Include Library > Manage Libraries... Wait for Library Manager to download libraries index and update list of installed libraries.

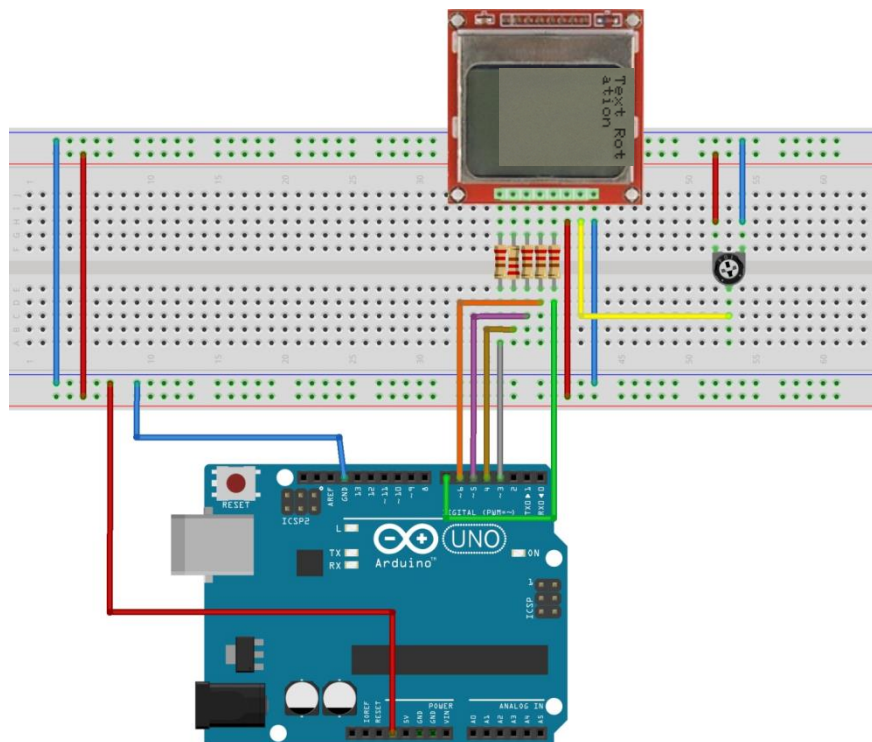
### Circuit 5:

#### Circuit title: "Text Rotation"

**Circuit Explanation:** You can rotate the contents of the display by calling setRotation() function. It allows you to view your display in portrait mode, or flip it upside down.

**Note:** The function accepts only one parameter that corresponds to 4 cardinal rotations. This value can be any non-negative integer starting from 0. Each time you increase the value, the contents of the display are rotated 90 degrees counter clockwise. For example:

- 0 – Keeps the screen to the standard landscape orientation.
- 1 – Rotates the screen 90° to the right.
- 2 – Flips the screen upside down.
- 3 – Rotates the screen 90° to the left.





```
/* Text Rotation */

#include <SPI.h>      // this library allows you to communicate with SPI devices, with the
                      // Arduino as the master device.

#include <Adafruit_GFX.h>      // the OLED display's libraries
#include <Adafruit_PCD8544.h>

// Declare LCD object for software SPIAdafruit_PCD8544(CLK,DIN,D/C,CE,RST);
Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);

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

    //Initialize Display
    display.begin();

    display.setContrast(57);      // you can change the contrast around to adapt the display

    display.clearDisplay();      // clear the buffer.

    // Text Rotation
    while(1)
    {
        display.clearDisplay();
        display.setRotation(rotatetext);
        display.setTextSize(1);
        display.setTextColor(BLACK);
        display.setCursor(0,0);
        display.println("Text Rotation");
        display.display();
        delay(1000);
        display.clearDisplay();
        rotatetext++;
    }
}

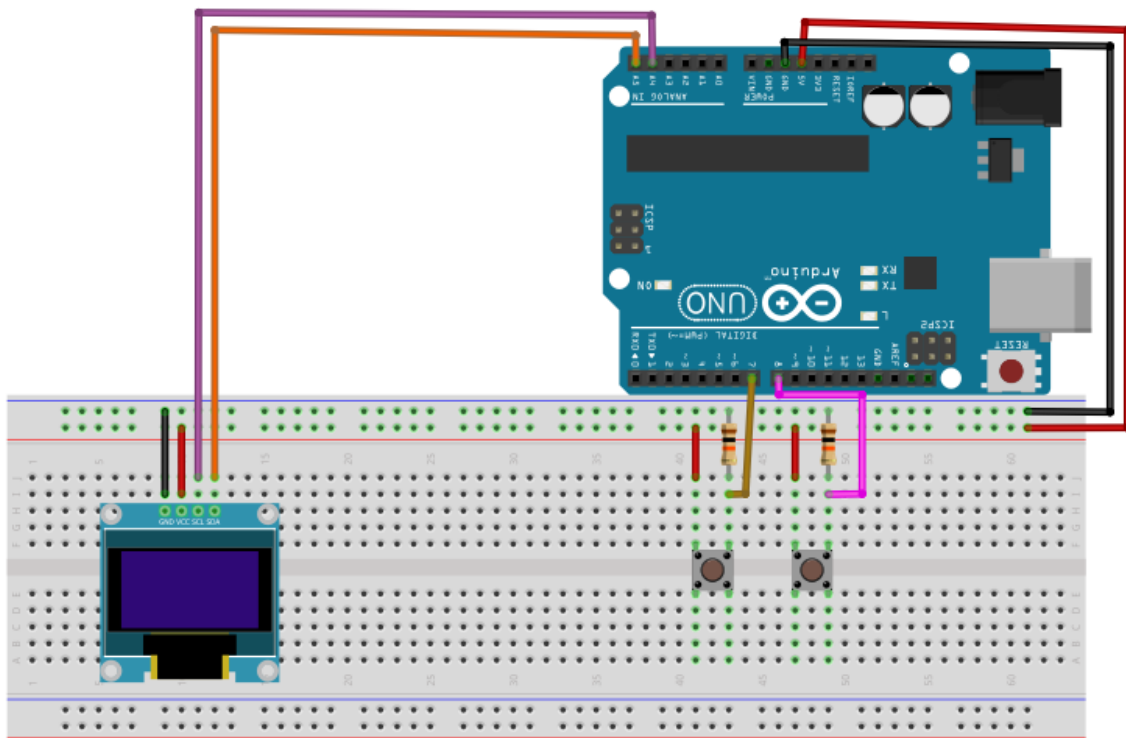
void loop() {}
```



### Circuit 6:

**Circuit title:** "Button counter"

**Circuit Explanation:** an OLED display showing a number, which can be incremented and decremented at the click of two different buttons.



```
#include <U8glib.h> //lcd libraries
#include "U8glib.h"
U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0); //display
model

int button_plus = 8, button_minus = 7; //declaration pin buttons
int state_plus, state_minus, number=0;

void setup() {

  pinMode(button_plus,INPUT);
  pinMode(button_minus,INPUT);

  Serial.begin(9600);

  u8g.setFont(u8g_font_fub25n); //font to be used for the write of the number
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
void write_number(void) {
    u8g.setPrintPos(10,50);
    u8g.print(number);
}

void loop() {

    //buttons
    state_plus = digitalRead(button_plus);
    state_minus = digitalRead(button_minus);

    if(state_plus==1)
    {
        number++;
        delay(50);
    }

    if(state_minus==1)
    {
        number--;
        delay(50);
    }

    //write de number sul display

    u8g.firstPage();
    do {
        write_number();
    } while ( u8g.nextPage() );

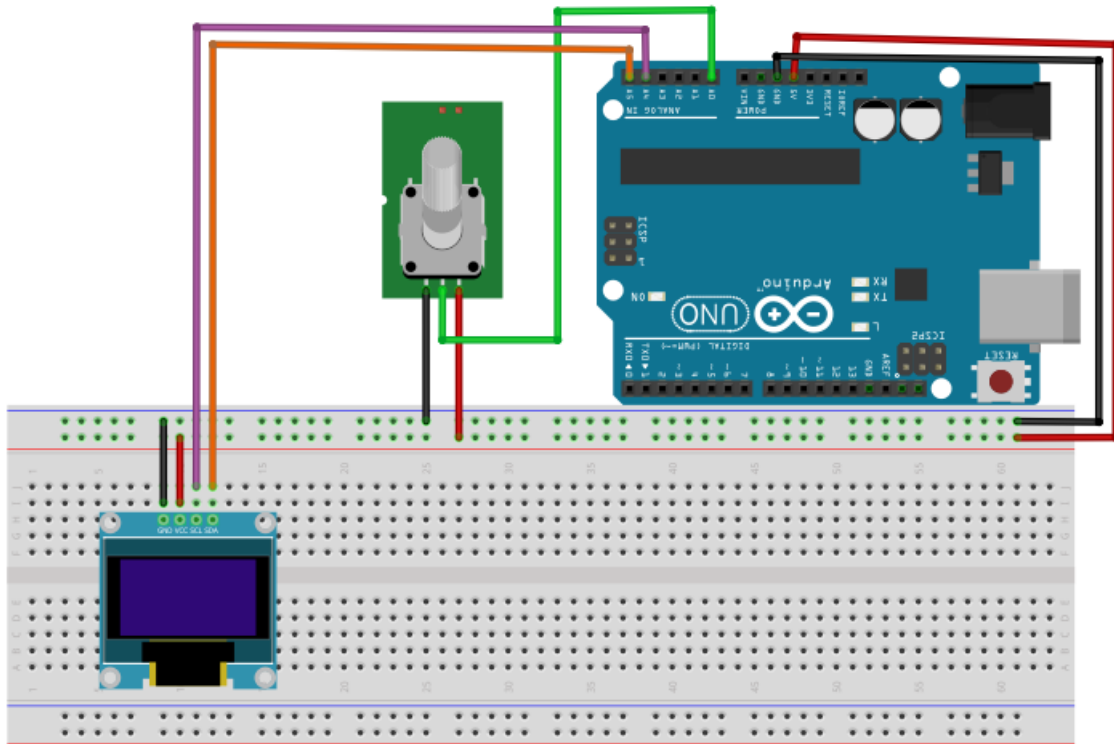
    u8g.firstPage();
}
```



### Circuit 7:

**Circuit title:** "Counter with potentiometer"

**Circuit Explanation:** an OLED display that shows a number, incrementing and decrementing as a potentiometer is turned.



```
#include <U8glib.h> //lcd libraries
#include "U8glib.h"
U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0); //display
model

int potentiometer = 0; //declaration and potentiometer
int state_pot, stop_pot, difference, number=0;

void setup() {
  Serial.begin(9600);
  u8g.setFont(u8g_font_fub25n); //font to be used for the write of the number
}

void write_number(void) {
  u8g.setPrintPos(10,50);
  u8g.print(number);
}
```



```
void loop() {
  state_pot = analogRead(potentiometer);

  //potentiometer
  stop_pot = state_pot; //save the value when it is stop to see if the potentiometer turns clockwise
or counterclockwise
  delay(100);
  state_pot = analogRead(potentiometer);
  difference = stop_pot - state_pot;
  if((difference > 2) || (difference < -2)) //is used to avoid making trades if the potentiometer is stop
  {
    number = number - difference / 5; //if difference is greater than 0 then it turns clockwise and adds
otherwise it subtracts
    delay(100);
  }

  //write the number on display
  u8g.firstPage();
  do {
    write_number();
  } while
  ( u8g.nextPage() );
  u8g.firstPage();
}
```

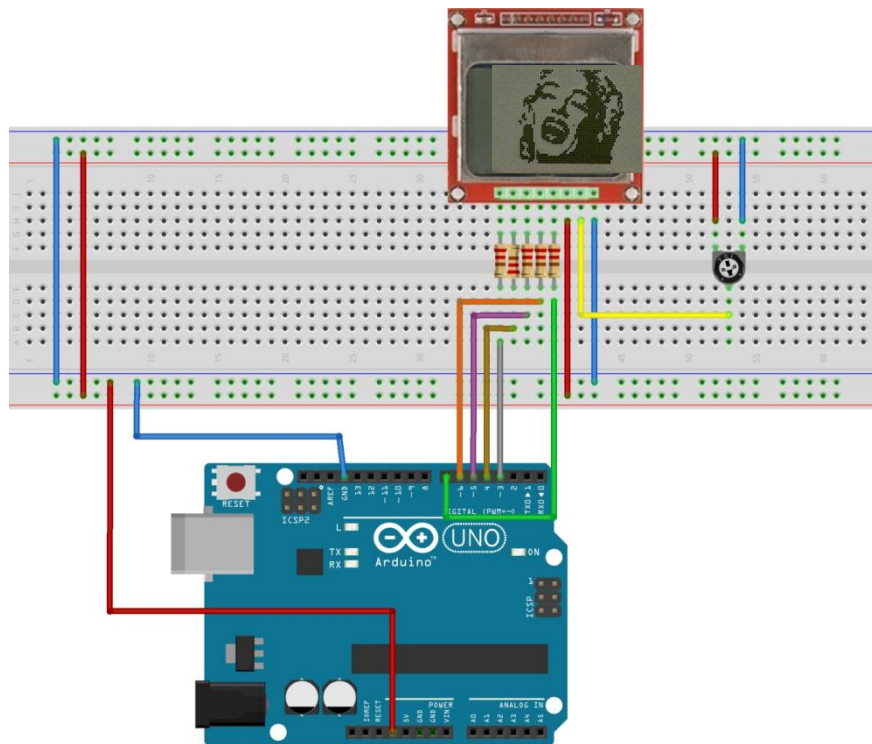


### Circuit 8:

**Circuit title:** "Marilyn Bmp Image"

**Circuit Explanation:** How to draw bitmap images to the Nokia 5110 LCD Display. In this example there is a portrait of Marilyn Monroe.

**Note:** The screen resolution of Nokia 5110 LCD display is 84×48 pixels, so images larger than that will not display correctly. To show bitmap image on the Nokia 5110 LCD display we need to call drawBitmap() function. It takes six parameters: top left corner X coordinate, top left corner Y coordinate, byte array of monochrome bitmap, width of bitmap in pixels, height of bitmap in pixels and Color. In our example, the bitmap image is 84×48 in size. So, X & Y coordinates are set to 0 while width & height is set to 84 and 48.



```
/* Marilyn Bmp Image */
```

```
#include <SPI.h>
```

```
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_PCD8544.h>
```

```
Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);
```

```
// 'Marilyn Monroe 84x48', 84x48px
```

```
const unsigned char MarilynMonroe [] PROGMEM = {
```



0x00, 0x00, 0x00, 0x7f, 0x00, 0x02, 0xfe, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xbe,  
0x00,  
0x00, 0x1f, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x3f, 0x80,  
0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0xf0, 0x00, 0x00, 0x1f, 0xe1, 0x80, 0x00, 0x00, 0x00, 0x00,  
0x00, 0xc0,  
0x00, 0x00, 0x0f, 0xf1, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0e, 0xd8,  
0xe0,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x80, 0x00, 0x07, 0xe0, 0x70, 0x00, 0x00, 0x00,  
0x00, 0x03,  
0x3f, 0xe0, 0x00, 0x07, 0xf0, 0x78, 0x00, 0x00, 0x00, 0x00, 0x01, 0xe0, 0x70, 0x00, 0x0f,  
0xee,  
0x7c, 0x00, 0x00, 0x00, 0x00, 0x03, 0xc0, 0x00, 0x00, 0x0f, 0xf7, 0x1c, 0x00, 0x00, 0x00,  
0x00,  
0x07, 0x80, 0x00, 0x0f, 0xc7, 0xf3, 0x1e, 0x00, 0x00, 0x00, 0x00, 0x07, 0xc0, 0x00, 0x0f,  
0xf3,  
0xdf, 0x7f, 0x80, 0x00, 0x00, 0x00, 0x07, 0xfe, 0x00, 0x08, 0x7d, 0xef, 0xff, 0xc0, 0x00,  
0x00,  
0x00, 0x7f, 0xff, 0x80, 0x30, 0x0f, 0xfc, 0xe0, 0xc0, 0x00, 0x00, 0x01, 0x9e, 0x73, 0xc0,  
0xe0,  
0x07, 0xf8, 0xc1, 0xc0, 0x00, 0x00, 0x03, 0xfc, 0x00, 0x01, 0xc0, 0x0f, 0xfd, 0xe1, 0x80,  
0x00,  
0x00, 0x03, 0xf8, 0x00, 0x01, 0x9c, 0x0f, 0xff, 0xc1, 0xc0, 0x00, 0x00, 0x02, 0xc0, 0x00,  
0x01,  
0x9f, 0xbf, 0xfe, 0x01, 0x40, 0x00, 0x00, 0x02, 0x60, 0x00, 0x03, 0x07, 0xef, 0xff, 0x01,  
0x40,  
0x00, 0x00, 0x00, 0x60, 0x00, 0x07, 0x01, 0xf7, 0xff, 0x80, 0xc0, 0x00, 0x00, 0x00, 0x50,  
0x01,  
0xdf, 0x00, 0x7f, 0xff, 0x1c, 0x80, 0x00, 0x00, 0x00, 0x40, 0x01, 0xff, 0x00, 0x1f, 0xff,  
0x1e,  
0xe0, 0x00, 0x00, 0x02, 0x08, 0x00, 0x3f, 0x80, 0x07, 0xef, 0x03, 0xe0, 0x00, 0x00, 0x06,  
0x08,  
0x00, 0x03, 0xc0, 0x07, 0xdf, 0x07, 0xc0, 0x00, 0x00, 0x06, 0x08, 0x0f, 0x81, 0x80, 0x1f,  
0xdf,  
0x1f, 0x80, 0x00, 0x00, 0x03, 0x08, 0x1f, 0x98, 0x00, 0x3f, 0xfe, 0x19, 0x80, 0x00, 0x00,  
0x18,  
0x08, 0x3f, 0xfe, 0x00, 0x7f, 0xfe, 0x3f, 0x00, 0x00, 0x00, 0x08, 0x08, 0x30, 0x3f, 0x00,  
0xff,  
0xff, 0x3f, 0x00, 0x00, 0x00, 0x01, 0xe0, 0x76, 0x0f, 0x89, 0xff, 0xff, 0x9f, 0x00, 0x00,  
0x00,  
0x03, 0xe0, 0x7f, 0xc3, 0x81, 0xff, 0xfe, 0x9f, 0x80, 0x00, 0x00, 0x03, 0xf0, 0x7f, 0xf3,  
0xc3,  
0xff, 0xfe, 0x1f, 0x00, 0x00, 0x00, 0x03, 0xf0, 0x7f, 0xfd, 0xc3, 0xff, 0xfe, 0x5e, 0x00,  
0x00,  
0x00, 0x03, 0xf0, 0x7f, 0xff, 0xc3, 0xff, 0xf3, 0x1e, 0x00, 0x00, 0x00, 0x03, 0xf0, 0x71,  
0xff,  
0x87, 0xff, 0xe3, 0xff, 0x00, 0x00, 0x00, 0x07, 0xf0, 0x7c, 0x3f, 0x87, 0xff, 0xe3, 0xfe,  
0x00,



```
0x00, 0x00, 0x0f, 0xf0, 0x3c, 0xff, 0x05, 0xff, 0xf3, 0xfc, 0x00, 0x00, 0x00, 0x0f, 0xf0,
0x0f,
0xfe, 0x09, 0xff, 0xf7, 0xfc, 0x00, 0x00, 0x00, 0x08, 0xf8, 0x01, 0xfc, 0x19, 0xff, 0xff,
0xf8,
0x00, 0x00, 0x00, 0x0c, 0x78, 0x00, 0x00, 0x13, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x0e,
0x78,
0x00, 0x00, 0x23, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x0e, 0xf8, 0x00, 0x00, 0x47, 0xff,
0xff,
0xf0, 0x00, 0x00, 0x00, 0x0c, 0xfa, 0x00, 0x01, 0x8f, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00,
0x08,
0x7b, 0x00, 0x03, 0x3f, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x0c, 0x3b, 0xf8, 0x0f, 0xff,
0xff,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x0f, 0xbb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00,
0x00,
0x07, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x71, 0xff, 0xff,
0xff,
0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x41, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0x00,
0x00
};

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

    display.begin();

    display.setContrast(57);

    display.clearDisplay();

    // Display bitmap
    display.drawBitmap(0, 0, MarilynMonroe, 84, 48, BLACK);
    display.display();

    // Invert Display
    //display.invertDisplay(1);
}

void loop() {}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



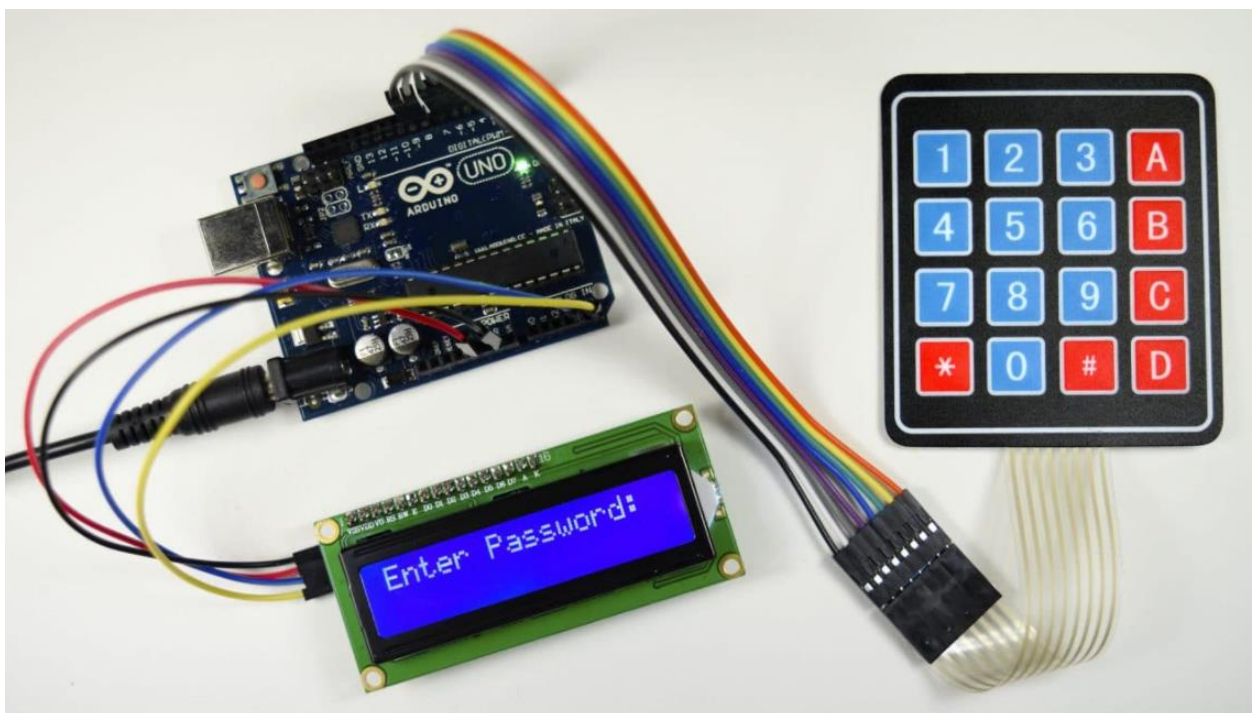
# Erasmus+ KA210-VET

Small-scale partnerships in vocational  
education and training

**Project Title: “Using Arduinos in Vocational  
Training” Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

## Keypad Module and Training Kit





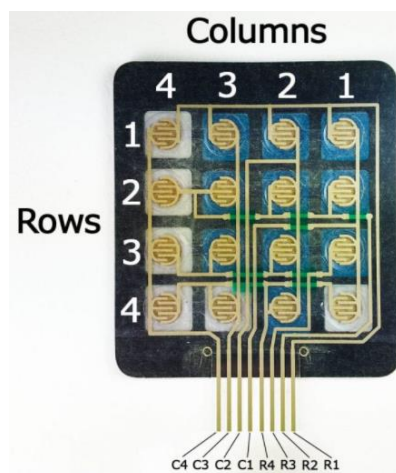
## What a keypad is?

A Keypad is a system of buttons arranged in a matrix that works as a switching device to provide a connection between a line and a column.

We will use a membrane keyboard with 4X4 matrix, because it is thin and has adhesive support, so that it can be glued on most flat surfaces.

Beneath each key is a membrane switch. Each switch in a row is connected to the other switches in the row by a conductive trace underneath the pad. Each switch in a column is connected the same way – one side of the switch is connected to all of the other switches in that column by a conductive trace. Each row and column is brought out to a single pin, for a total of 8 pins on a 4X4 keypad.

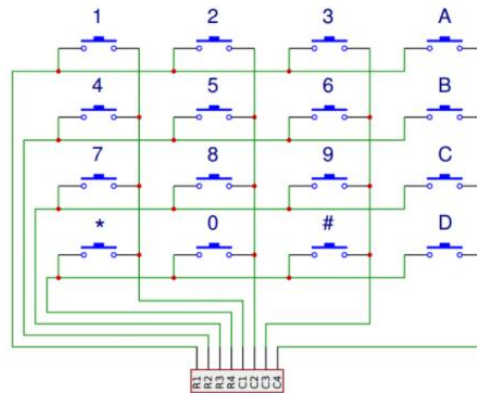
4X4 Keypad



Pressing a button closes the switch between a column and a row trace, allowing current to flow between a column pin and a row pin.



The schematic for a 4X4 keypad shows how the rows and columns are connected:

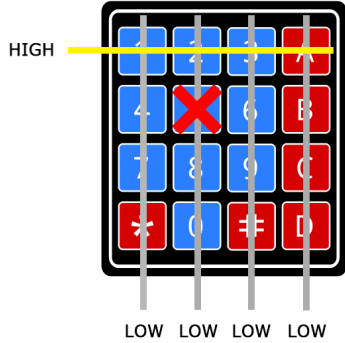
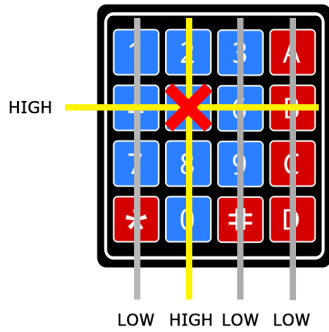


The Arduino detects which button is pressed by detecting the row and column pin that's connected to the button.

This happens in four steps:

<p>1. First, when no buttons are pressed, all of the column pins are held HIGH, and all of the row pins are held LOW</p>	
<p>2. When a button is pressed, the column pin is pulled LOW since the current from the HIGH column flows to the LOW row pin:</p>	

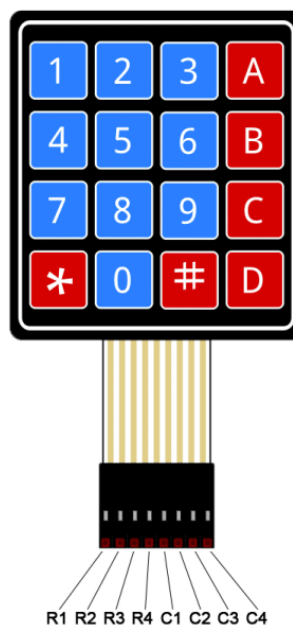


<p>3. The Arduino now knows which column the button is in, so now it just needs to find the row the button is in. It does this by switching each one of the row pins HIGH, and at the same time reading all of the column pins to detect which column pin returns to HIGH</p>	
<p>4. When the column pin goes HIGH again, the Arduino has found the row pin that is connected to the button:</p>	

From the diagram above, you can see that the combination of row 2 and column 2 could only mean that the number 5 button was pressed.

### CONNECT THE KEYPAD TO THE ARDUINO

The pin layout for most membrane keypads will look like this:

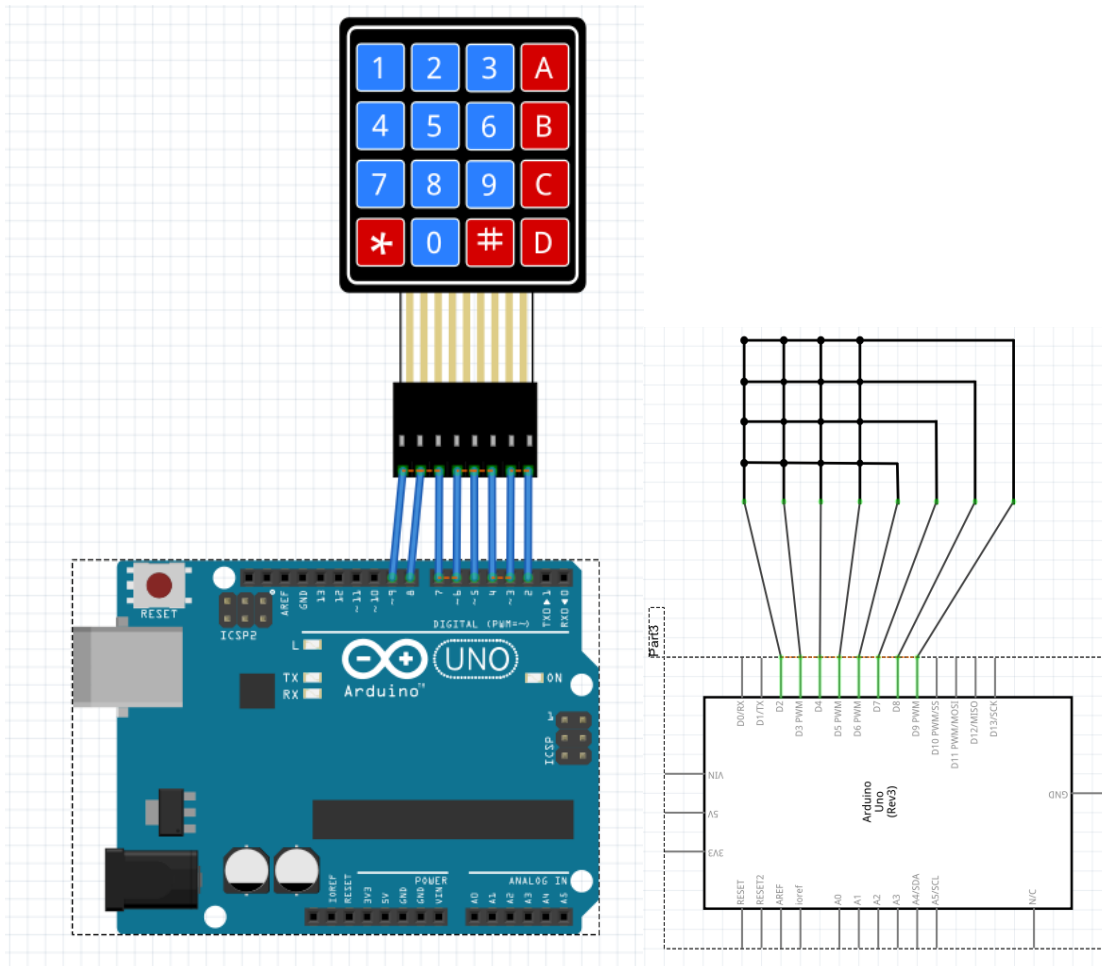




## Circuit 1

**Circuit title:** Serial monitor display the pressed key

**Circuit description:** Program will show us how to print each key press to the serial monitor.



1-) Breadboard view

2-) Schematic view

**/\* “Serial monitor display the pressed key” \*/**

```
#include <Keypad.h>
```

```
const byte ROWS = 4;
```

```
const byte COLS = 4;
```

```
char hexaKeys[ROWS][COLS] = {
```

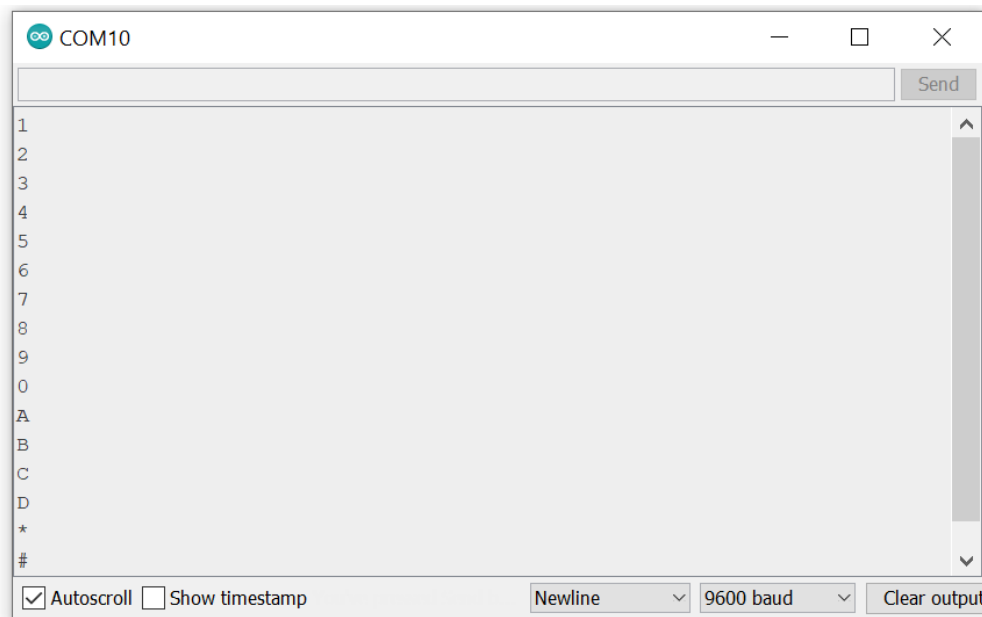
```
  {'1', '2', '3', 'A'},
```

```
  {'4', '5', '6', 'B'},
```

```
  {'7', '8', '9', 'C'},
```



```
{ '*', '0', '#', 'D' }  
};  
byte rowPins[ROWS] = { 9, 8, 7, 6 };  
byte colPins[COLS] = { 5, 4, 3, 2 };  
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);  
void setup(){  
  Serial.begin(9600);  
}  
void loop(){  
  char customKey = customKeypad.getKey();  
  if (customKey){  
    Serial.println(customKey);  
  }  
}
```

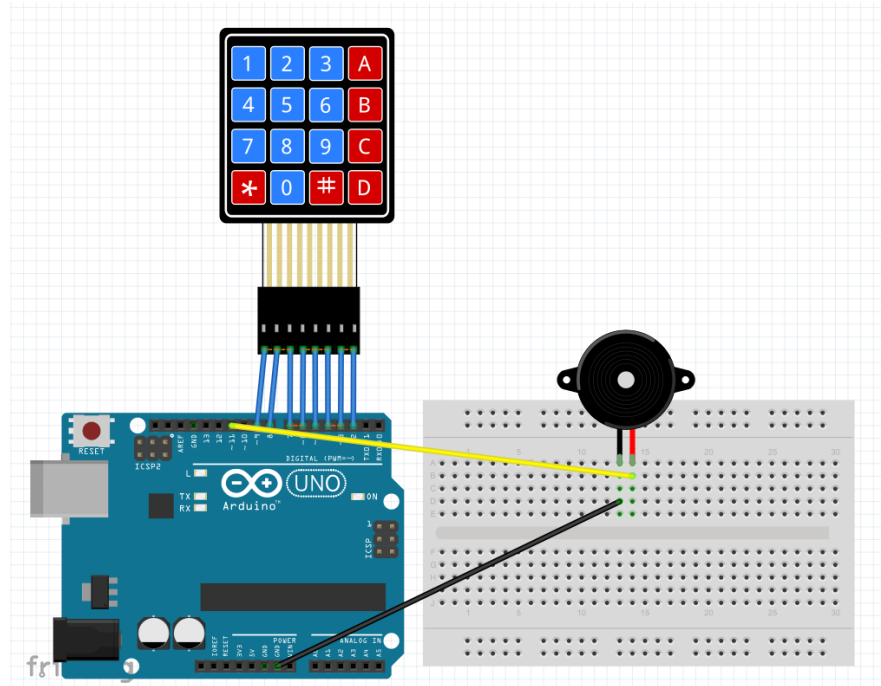




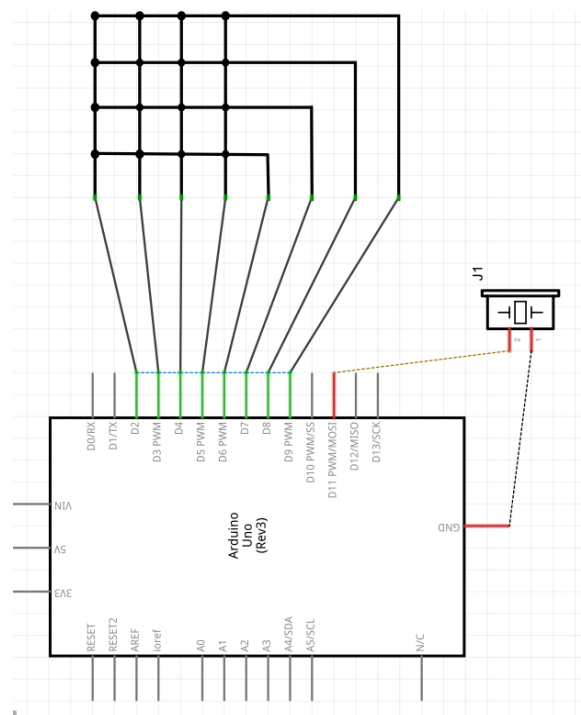
## Circuit 2

**Circuit title:** Arduino Keypad Beep

**Circuit description:** When a key on the keypad is pressed, the piezo buzzer beeps



1-) Breadboard view



2-) Schematic view



```
/*  “Arduino Keypad Beep”  */
```

```
#include <Keypad.h>
```

```
#include <ezBuzzer.h>
```

```
const int BUZZER_PIN = 11;
```

```
const int ROW_NUM  = 4; // four rows
```

```
const int COLUMN_NUM = 4; // four columns
```

```
char keys[ROW_NUM][COLUMN_NUM] = {
```

```
    {'1', '2', '3', 'A'},
```

```
    {'4', '5', '6', 'B'},
```

```
    {'7', '8', '9', 'C'},
```

```
    {'*', '0', '#', 'D'}
```

```
};
```

```
byte pin_rows[ROW_NUM] = {9, 8, 7, 6};    // connect to the row pinouts of the keypad
```

```
byte pin_column[COLUMN_NUM] = {5, 4, 3, 2}; // connect to the column pinouts of the keypad
```

```
Keypad keypad = Keypad(makeKeymap(keys), pin_rows, pin_column, ROW_NUM,  
COLUMN_NUM );
```

```
ezBuzzer buzzer(BUZZER_PIN); // create ezBuzzer object that attach to a pin;
```

```
void setup() {
```

```
    Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
    buzzer.loop(); // MUST call the buzzer.loop() function in loop()
```

```
    char key = keypad.getKey();
```

```
    if (key) {
```

```
        Serial.print(key); // prints key to serial monitor
```

```
        buzzer.beep(100); // generates a 100ms beep
```

```
    }
```

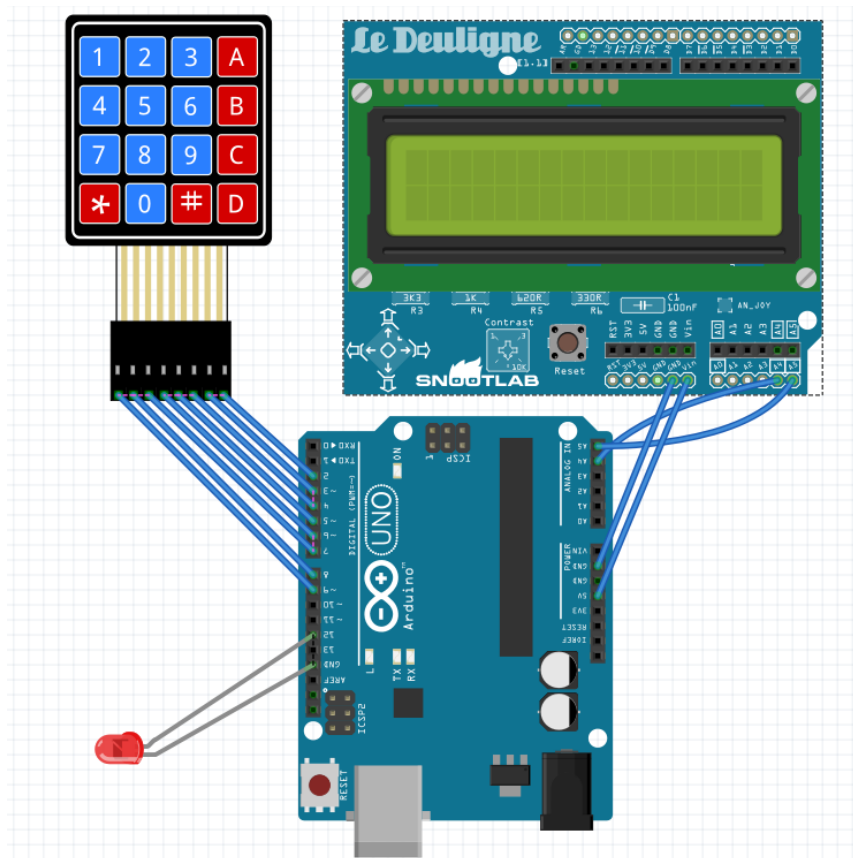
```
}
```



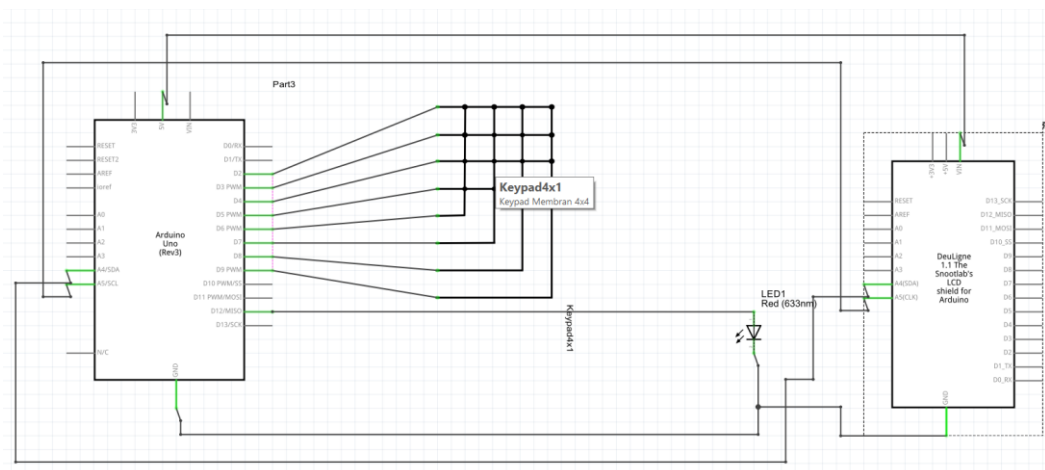
### Circuit 3

**Circuit title:** Arduino Unlocking code

**Circuit description:** Program that sets a keypad on the Arduino. An LED will light up when you type the correct code



1-) Breadboard view



2-) Schematic view



**/\* Arduino Unlocking code \*/**

```
#include <Keypad.h> //Libraries you can download them via Arduino IDE
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#define Solenoid 12          //Actually the Gate of the transistor that controls the solenoid
                             //in my case I use a simple LED
#include <liquidcrystal_i2c.h></liquidcrystal_i2c.h></lcd.h></wire.h></keypad.h>
#define I2C_ADDR 0x27 // LCD i2c Adress and pins
#define BACKLIGHT_PIN 3
#define En_pin 2
#define Rw_pin 1
#define Rs_pin 0
#define D4_pin 4
#define D5_pin 5
#define D6_pin 6
#define D7_pin 7
LiquidCrystal_I2C lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
const byte numRows= 4; //number of rows on the keypad
const byte numCols= 4; //number of columns on the keypad
int code = 1234; //here is the code
int tot,i1,i2,i3,i4;
char c1,c2,c3,c4;
//keymap defines the key pressed according to the row and columns just as appears on the
keypad char keymap[numRows][numCols]=
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```



```
//Code that shows the keypad connections to the arduino terminals
byte rowPins[numRows] = {9,8,7,6}; //Rows 0 to 3
byte colPins[numCols]= {5,4,3,2}; //Columns 0 to 3
//initializes an instance of the Keypad class
Keypad myKeypad= Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);
void setup()
{
  lcd.begin (16,2);
  lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
  lcd.setBacklight(HIGH);
  lcd.home ();
  lcd.print("ROMANIA DoorLock");
  lcd.setCursor(9, 1);
  lcd.print("Standby");
  pinMode(Solenoid,OUTPUT);
  delay(2000);
}
void loop()
{
  char keypressed = myKeypad.getKey(); //The getKey function keeps the program running, as
  long you didn't press "*" the whole thing below wouldn't be triggered
  if (keypressed == '*')          // and you can use the rest of your code simply
  {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Enter Code");      //when the "*" key is pressed you can enter the
    passcode
    keypressed = myKeypad.waitForKey(); // here all programs are stopped until you
    enter the four digits then it gets compared to the code above
    if (keypressed != NO_KEY)
    {
      c1 = keypressed;
```



```
    lcd.setCursor(0, 1);
    lcd.print("*");
}
keypressed = myKeypad.waitForKey();
if (keypressed != NO_KEY)
{
    c2 = keypressed;
    lcd.setCursor(1, 1);
    lcd.print("*");
}
keypressed = myKeypad.waitForKey();
if (keypressed != NO_KEY)
{
    c3 = keypressed;
    lcd.setCursor(2, 1);
    lcd.print("*");
}
keypressed = myKeypad.waitForKey();
if (keypressed != NO_KEY)
{
    c4 = keypressed;
    lcd.setCursor(3, 1);
    lcd.print("*");
}
i1=(c1-48)*1000;    //the keys pressed are stored into chars I convert them to int
then i did some multiplication to get the code as an int of xxxx
i2=(c2-48)*100;
i3=(c3-48)*10;
i4=c4-48;
tot=i1+i2+i3+i4;
if (tot == code) //if the code is correct you trigger whatever you want here it just print
a message on the screen
```



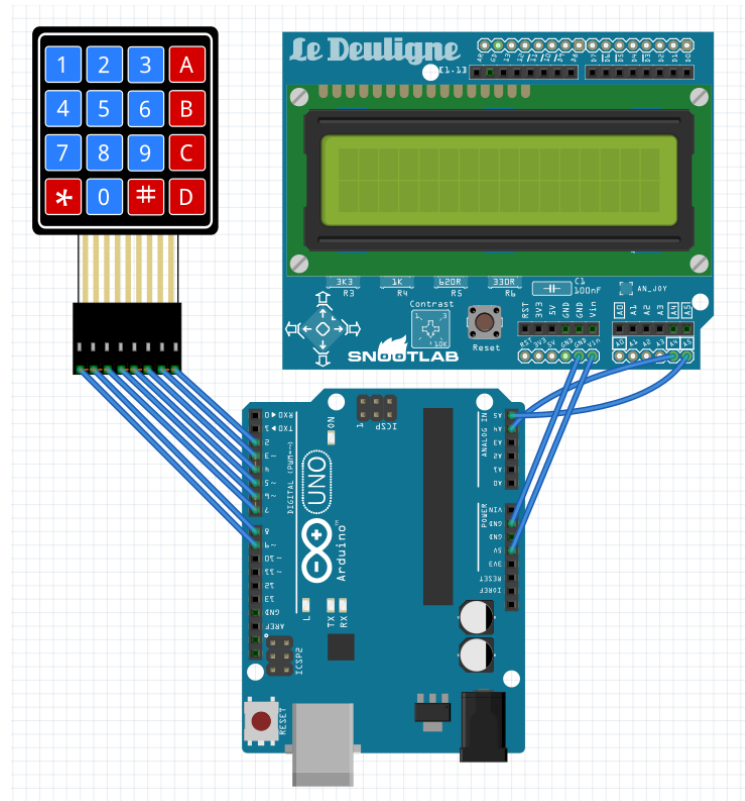
```
{  
  lcd.clear();  
  lcd.setCursor(0, 0);  
  lcd.print("Welcome");  
  digitalWrite(Solenoid,HIGH);  
delay(3000);  
digitalWrite(Solenoid,LOW);  
  lcd.setCursor(7, 1);  
  lcd.print("ROMANIA DoorLock");  
  
  delay(3000);  
  lcd.clear();  
  lcd.print("ROMANIA DoorLock");  
  lcd.setCursor(9, 1);  
  lcd.print("Standby");  
  
}  
else //if the code is wrong you get another thing  
{  
  lcd.clear();  
  lcd.setCursor(0, 0);  
  lcd.print("WRONG CODE");  
  delay(3000);  
  lcd.clear();  
  lcd.print("ROMANIA DoorLock");  
  lcd.setCursor(9, 1);  
  lcd.print("Standby");  
}  
}
```



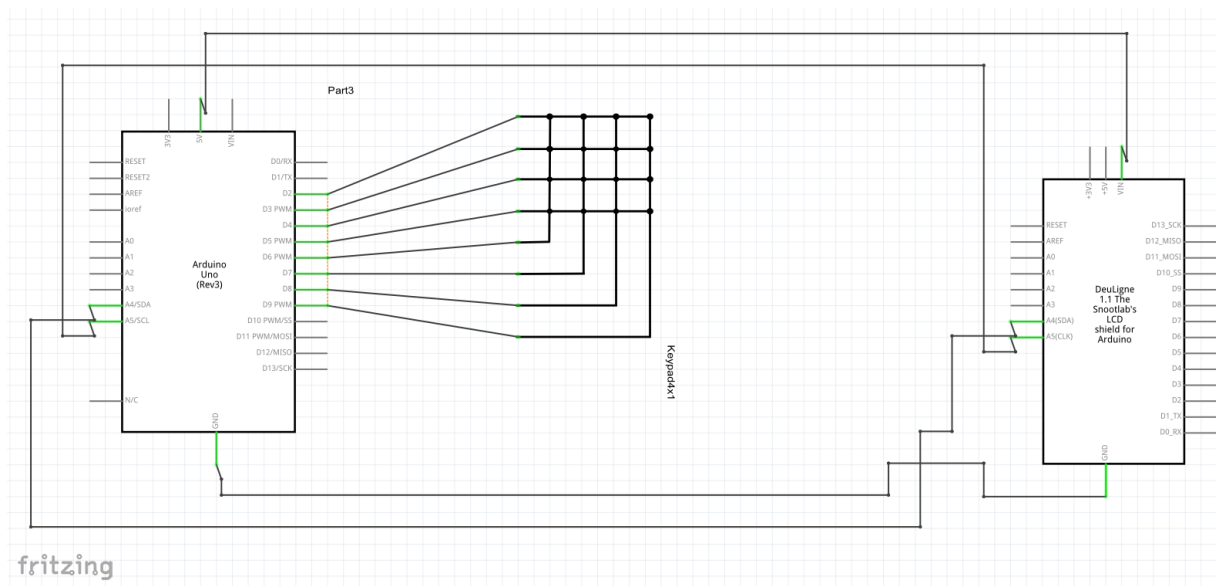
## Circuit 4

**Circuit title:** Arduino calculator

**Circuit description:** Program does basic mathematical calculations



1-) Breadboard view



2-) Schematic view



**/\* Arduino calculator \*/**

```
#include <Keypad.h>
#include <EEPROM.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#define I2C_ADDR 0x27      //I2C adress
#define BACKLIGHT_PIN 3    // Declaring LCD Pins
#define En_pin 2
#define Rw_pin 1
#define Rs_pin 0
#define D4_pin 4
#define D5_pin 5
#define D6_pin 6
#define D7_pin 7
const byte ROWS = 4; // Four rows
const byte COLS = 4; // Four columns
// Define the Keymap
char keys[ROWS][COLS] = {
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};
byte rowPins[ROWS] = {9,8,7,6}; //Rows 0 to 3
byte colPins[COLS]= {5,4,3,2}; //Columns 0 to 3
LiquidCrystal_I2C lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS ); // Create the
Keypad
long Num1,Num2,Number;
char key,action;
boolean result = false;
void setup()
```



```
{  
  lcd.begin (16,2);  
  lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);  
  lcd.setBacklight(HIGH); //Lighting backlight  
  lcd.print("Calculator Ready"); //Display a intro message  
  lcd.setCursor(0, 1); // set the cursor to column 0, line 1  
  lcd.print("A+= B=- C=* D=/"); //Display a intro message  
  delay(6000); //Wait for display to show info  
  lcd.clear(); //Then clean it  
      //      for(i=0 ; i<sizeof(code);i++){      //When you upload the code the first  
time keep it commented  
//      EEPROM.get(i, code[i]);      //Upload the code and change it to store it in the  
EEPROM  
//      }      //Then uncomment this for loop and reupload the code (It's done only once)  
      }  
void loop() {  
  key = kpd.getKey(); //storing pressed key value in a char  
  if (key!=NO_KEY)  
  DetectButtons();  
  if (result==true)  
  CalculateResult();  
  DisplayResult();  
}  
void DetectButtons()  
{  
  lcd.clear(); //Then clean it  
  if (key=='*') //If cancel Button is pressed  
  {Serial.println ("Button Cancel"); Number=Num1=Num2=0; result=false;}  
  if (key == '1') //If Button 1 is pressed  
  {Serial.println ("Button 1");  
  if (Number==0)  
  Number=1;
```



```
else
Number = (Number*10) + 1; //Pressed twice
}
    if (key == '4') //If Button 4 is pressed
{Serial.println ("Button 4");
if (Number==0)
Number=4;
else
Number = (Number*10) + 4; //Pressed twice
}
    if (key == '7') //If Button 7 is pressed
{Serial.println ("Button 7");
if (Number==0)
Number=7;
else
Number = (Number*10) + 7; //Pressed twice
}
    if (key == '0')
{Serial.println ("Button 0"); //Button 0 is Pressed
if (Number==0)
Number=0;
else
Number = (Number*10) + 0; //Pressed twice
}
    if (key == '2') //Button 2 is Pressed
{Serial.println ("Button 2");
if (Number==0)
Number=2;
else
Number = (Number*10) + 2; //Pressed twice
}
    if (key == '5')
```



```
{Serial.println ("Button 5");
  if (Number==0)
    Number=5;
  else
    Number = (Number*10) + 5; //Pressed twice
}
  if (key == '8')
{Serial.println ("Button 8");
  if (Number==0)
    Number=8;
  else
    Number = (Number*10) + 8; //Pressed twice
}
  if (key == '#')
{Serial.println ("Button Equal");
  Num2=Number;
  result = true;
}
  if (key == '3')
{Serial.println ("Button 3");
  if (Number==0)
    Number=3;
  else
    Number = (Number*10) + 3; //Pressed twice
}
  if (key == '6')
{Serial.println ("Button 6");
  if (Number==0)
    Number=6;
  else
    Number = (Number*10) + 6; //Pressed twice
}
```



```
        if (key == '9')
        {Serial.println ("Button 9");
        if (Number==0)
        Number=9;
        else
        Number = (Number*10) + 9; //Pressed twice
        }
        if (key == 'A' || key == 'B' || key == 'C' || key == 'D') //Detecting Buttons on Column 4
        {
        Num1 = Number;
        Number =0;
        if (key == 'A')
        {Serial.println ("Addition"); action = '+';}
        if (key == 'B')
        {Serial.println ("Subtraction"); action = '-'; }
        if (key == 'C')
        {Serial.println ("Multiplication"); action = '*';}
        if (key == 'D')
        {Serial.println ("Division"); action = '/';}
        delay(100);
        }
    }
    void CalculateResult()
    {
        if (action=='+')
        Number = Num1+Num2;
        if (action=='-')
        Number = Num1-Num2;
        if (action=='*')
        Number = Num1*Num2;
        if (action=='/')
        Number = Num1/Num2;
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
}  
void DisplayResult()  
{  
    lcd.setCursor(0, 0); // set the cursor to column 0, line 1  
    lcd.print(Num1); lcd.print(action); lcd.print(Num2);  
    if (result==true)  
    {lcd.print(" ="); lcd.print(Number);} //Display the result  
    lcd.setCursor(0, 1); // set the cursor to column 0, line 1  
    lcd.print(Number); //Display the result  
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



# **Erasmus+ KA210-VET**

**Small-scale partnerships in vocational  
education and training**

**Project Title: “Using Arduinos in Vocational  
Training” Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

## **Dot Matrix Module**





## Using Arduinos in Vocational Training

### UsingARDinVET

IPSIA G.Giorgi - Potenza (Italy)

## Dot Matrix Display Module

In the context of Arduino, a Dot Matrix is an LED display consisting of a two-dimensional grid of LEDs arranged in rows and columns. Each LED can be independently turned on or off, allowing for the creation of complex patterns and animations through the simultaneous control of multiple LEDs.

An LED matrix is useful for a wide range of applications. An 8x8 matrix, like the one shown in Figure 1, can be used to display letters or numbers. If you have several of these modules placed side by side, you can create a display with scrolling text.



Figure 1: An 8x8 dot matrix LED.

Note that the module is designed so that there is very little space between the LEDs and the edges of the module. When these types of matrix modules are mounted side by side, the distance between the last column or row on one module and the adjacent column or row on the next module is equal to the distance between the LEDs located at the center of the module. This maintains a consistent spacing when using multiple modules to create large displays.



## Circuit 1. Creating a Bar Graph

Consider, for instance, an 10x1 Dot Matrix display, which consists of 10 individual LEDs arranged in a single row. You can connect the LEDs as shown in Figure 2.

The following sketch turns on a series of LEDs, with the number being proportional to the value of a potentiometer connected to an analog input port.

```
const int analog Pin = A0 ;      // the pin at the potentiometer is attached to
```

```
const int led Count = 10 ;      // the number of LEDs in the bar graph
```

```
// an array of pin numbers to which LEDs are attached
```

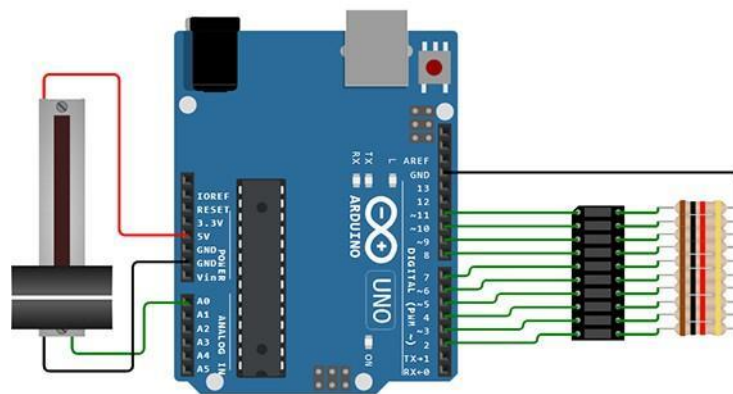


Figure 2: Bar Graph.

```
int led Pins [] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 };
```

```
void setup () {
```

```
  // loop over the pin array and set them all to output :
```

```
  for ( int this Led = 0 ; this Led < led Count ; this Led++ ) {
```

```
    pinMode ( led Pins [ this Led ] , OUTPUT ) ;
```

```
  }
```

```
}
```

```
void loop () {
```



```
// read the potentiometer :

int sensor Reading = analogRead ( analog Pin ) ;

// map the r e s u l t to a range from 0 to the number o f LEDs :

int led L e v e l = map( sensor Reading , 0 , 1023 , 0 , led Count ) ;

// l o o p over the LED array :

for ( int this Led = 0 ; this Led < led Count ; this Led++) {

    // i f the array element ' s index i s l e s s than l e d L e v e l ,

    // turn the pin for t h i s element on :

    if ( this Led < l e d L e v e l ) {

        d i g i t a l W r i t e ( l e d P i n s [ this Led ] , HIGH ) ;

    }

    // turn o f f a l l pins h i g h e r than the l e d L e v e l :

    e l s e {

        d i g i t a l W r i t e ( l e d P i n s [ this Led ] , LOW ) ;

    }

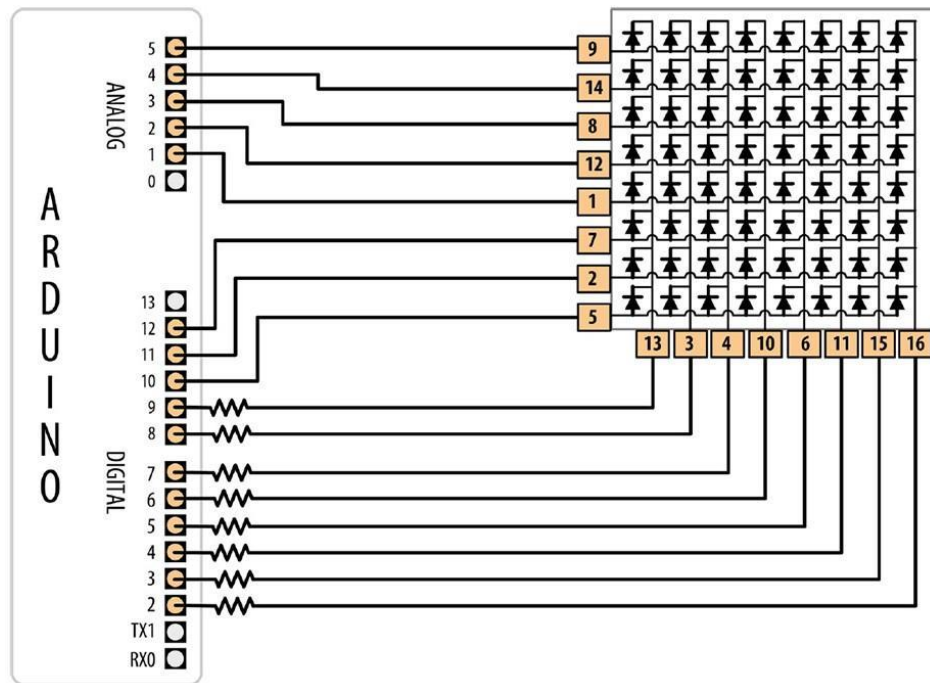
}
```

The pins connected to LEDs are held in the array ledPins. To change the number of LEDs, you can add (or remove) elements from this array, but make sure the variable ledCount is the same as the number of elements (which should be the same as the number of pins). The Arduino map function is used to calculate the number of LEDs that should be lit as a proportion of the sensor value. The code loops through each LED, turning it on if the proportional value of the sensor is greater than the LED number.

In an ideal world, a potentiometer at its lowest setting will return zero, but it's likely to drift in the real world. When the sensor is at maximum value, all the LEDs are lit. If you find that the last LED flickers when the potentiometer is at its maximum value, try lowering the second argument to map from 1023 to 1000 or so.



## Circuit 2. Controlling an LED Matrix



This sketch uses an LED matrix of 64 LEDs, with anodes connected in rows and cathodes in columns (as in the Jameco 2132349). Figure 3 shows the connections (Dual-color LED displays may be easier to obtain, and you can drive just one of the colors if that is all you need).

Figure 3: An LED matrix connected to 16 digital pins.

```
const int columnPins [] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };

const int rowPins [] = { 10 , 11 , 12 , A1 , A2 , A3 , A4 , A5 } ;

int pixel = 0 ;           // 0 to 63 LEDs in the matrix

int columnLevel = 0 ;     // pixel value converted into LED column

int rowLevel = 0 ;        // pixel value converted into LED row

void setup () {

  for (int i = 0 ; i < 8 ; i++) { pinMode (
    columnPins [ i ] , OUTPUT) ; pinMode
    ( rowPins [ i ] , OUTPUT) ;

  }
```



```
}
```

```
void loop() {  
  
    pixel=pixel+1;  
  
    if (pixel>63) pixel=0;  
  
    columnLevel=pixel/8;    // map to the number of columns row  
    Level=pixel%8;          // get the fractional value  
  
    for(int column=0; column<8; column++) { di  
        gitalWrite(columnPins[column], LOW);  
  
        for (int row=0; row<8; row++)  
  
            { if( columnLevel > column ) {  
  
                digitalWrite(rowPins[row], HIGH);  
            } else if ( columnLevel == column && rowLevel >= row ) {  
                digitalWrite(rowPins[row], HIGH);  
  
            } else {  
  
                // turn off all LEDs in this row  
  
                digitalWrite(columnPins[column],  
                    LOW);  
  
            }  
  
            delayMicroseconds(300);  
  
            digitalWrite(rowPins[row], LOW);    // turn off LED  
        }  
  
        // disconnect this column from Ground  
  
        digitalWrite(columnPins[column],  
            HIGH);  
  
    }  
}
```

The resistor's value must be chosen to ensure that the maximum current through a pin does not exceed 40 mA on the Arduino Uno. Because the current for up to eight LEDs can flow through



each column pin, the maximum current for each LED must be one-eighth of 40 mA, or 5 mA. Each LED in a typical small red matrix has a forward voltage of around 1.8 volts. Calculating the resistor that results in 5 mA with a forward voltage of 1.8 volts gives a value of  $680\Omega$ . Check your datasheet to find the forward voltage of the matrix you want to use. Each column of the matrix is connected through the series resistor to a digital pin. When the column pin goes low and a row pin goes high, the corresponding LED will light. For all LEDs where the column pin is high or its row pin is low, no current will flow through the LED and it will not light. The for loop scans through each row and column and turns on sequential LEDs until all LEDs are lit. The loop starts with the first column and row and increments the row counter until all LEDs in that row are lit; it then moves to the next column, and so on, lighting another LED with each pass through the loop until all the LEDs are lit.

You don't have to light an entire row at once. The following sketch will light one LED at a time as it goes through the sequence:

```
const int columnPins [] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };

const int rowPins [] = { 10 , 11 , 12 , A1 , A2 , A3 , A4 , A5 };

int pixel = 0; // 0 to 63 LEDs in the matrix

void setup () {

    for (int i = 0; i < 8; i++) {

        pinMode ( columnPins [ i ] , OUTPUT) ;// make all the LED pins outputs
        pinMode ( rowPins [ i ] , OUTPUT) ;

        digitalWrite ( columnPins [ i ] , HIGH );

    }

}

void loop () {

    pixel = pixel + 1;

    if ( pixel > 63 ) pixel = 0;

    int column = pixel / 8; // map to the number of columns

    int row = pixel % 8; // get the fractional value
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
digitalWrite(columnPins[column], LOW); // Connect this column to  
GND  
  
digitalWrite(rowPins[row], HIGH); // Take this row HIGH  
  
delay(125); // pause briefly  
  
digitalWrite(rowPins[row], LOW); // Take the row low  
  
digitalWrite(columnPins[column], HIGH); // Disconnect the column from GND  
}
```



### Circuit 3. Displaying Images on an LED Matrix

You want to display one or more images on an LED matrix, perhaps creating an animation effect by quickly alternating multiple images. This solution can use the same wiring as in figure 3. The sketch creates the effect of a heart beating by briefly lighting LEDs arranged in the shape of a heart. A small heart followed by a larger heart is flashed for each heartbeat (the images look like figure 4):

```
byte bigHeart [ ] = {
    B01100110 ,
    B11111111 ,
    B11111111 ,
    B11111111 ,
    B01111110 ,
```

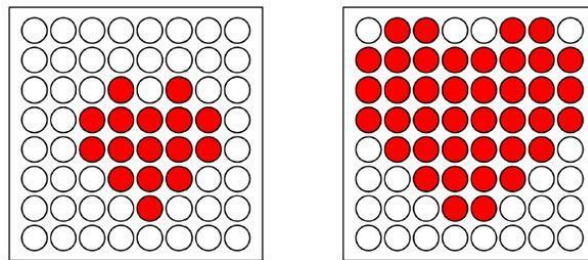


Figure 4: The two heart images displayed on each beats.

```
B00111100 ,
B00011000 ,
B00000000 } ;
```

```
byte smallHeart [ ] = {
    B00000000 ,
    B00000000 ,
    B00010100 ,
    B00111110 ,
    B00111110 ,
    B00011100 ,
    B00001000 ,
    B00000000 } ;
```

```
const int columnPins [ ] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 } ;
```

```
const int rowPins [ ] = { 10 , 11 , 12 , A1 , A2 , A3 , A4 , A5 } ;
```



```
void setup () {  
  
    for (int i = 0; i < 8; i++) {  
  
        pinMode ( rowPins [ i ] , OUTPUT); // make all the LED pins outputs  
        pinMode ( columnPins [ i ] , OUTPUT);  
  
        digitalWrite ( columnPins [ i ] , HIGH );    // disconnect column pins from Ground  
    }  
}  
  
void loop () {  
  
    int pulseDelay = 800;    // milliseconds to wait between beats  
  
    show ( smallHeart, 80 ); // show the small heart image for 80 ms  
  
    show ( bigHeart, 160 ); // followed by the big heart for 160 ms  
  
    delay ( pulseDelay );    // show nothing between beats  
}  
  
// Show a frame of an image stored in the array pointed to by the image  
// parameter. The frame is repeated for the given duration in milliseconds.  
void show ( byte * image ),  
unsigned long duration ) {  
  
    unsigned long start = millis();    // begin timing the animation  
    while ( start + duration > millis() )    // loop until the duration has passed  
    {  
  
        for (int row = 0; row < 8; row++) {  
  
            digitalWrite ( rowPins [ row ] , HIGH );  
  
            // connect row to +5 volts  
  
            for (int column = 0; column < 8; column++)
```



```

{bool pixel = bitRead ( image [ row ] , column ) ;

  if ( pixel == 1 ) {

    digitalWrite ( columnPins [ column ] , LOW ) ;      // connect column to
    Gnd

  }

  delay Microseconds ( 300 ) ;                          // a small delay for each LED dig
  italWrite ( columnPins [ column ] , HIGH ) ; // disconnect column from Gnd

}

digitalWrite ( rowPins [ row ] , LOW ) ;    // disconnect LEDs

}

}

}

```

The value written to the LED is based on images stored in the bigHeart and smallHeart arrays. Each element in the array represents a pixel (a single LED) and each array row represents a row in the matrix. A row consists of eight bits represented using binary format (as designated by the capital B at the start of each row). A bit with a value of 1 indicates that the corresponding LED should be on; a 0 means off. The animation effect is created by rapidly switching between the arrays. The loop function waits a short time (800 ms) between beats and then calls the show function, first with the smallHeart array and then followed by the bigHeart array. The show function steps through each element in all the rows and columns, lighting the LED if the corresponding bit is 1. The bitRead function is used to determine the value of each bit. A short delay of 300 microseconds between each pixel allows the eye enough time to perceive the LED. The timing is chosen to allow each image to repeat quickly enough (50 times per second) so that blinking is not perceptible.

Here is a variation that changes the rate at which the heart beats, based on the value from a sensor. You can test this using a variable resistor connected to analog input pin 0. Use the wiring and code shown earlier, except replace the loop function with this code:

```

void loop () {

  int sensor Value = analogRead ( A0 ) ;    // read the analog in value

  int pulse Rate =

    map( sensor Value , 0 , 1023 , 40 , 240 ) ;    // convert to beats / minute

```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
int pulse Delay = ( 60000 / pulse Rate );      // m i l l i s e c o n d s to wait between beats

show ( small Heart , 80 ); // show the small heart image f o r 100 ms

show ( big Heart , 160 ); // f o l l o w e d by the b i g heart for 200 ms

delay ( pulse Delay );      // show nothing between beats

}
```

This version calculates the delay between pulses using the map function to convert the sensor value into beats per minute.



#### Circuit 4. Controlling an Array of LEDs by using MAX72xx

You have an 8x8 array of LEDs to control, and you want to minimize the number of required Arduino pins. You can use a shift register to reduce the number of pins needed to control an LED matrix. This solution uses the MAX7219 or MAX7221 LED driver chip to provide this capability. Connect your Arduino, matrix, and MAX72xx as shown in gure 5).

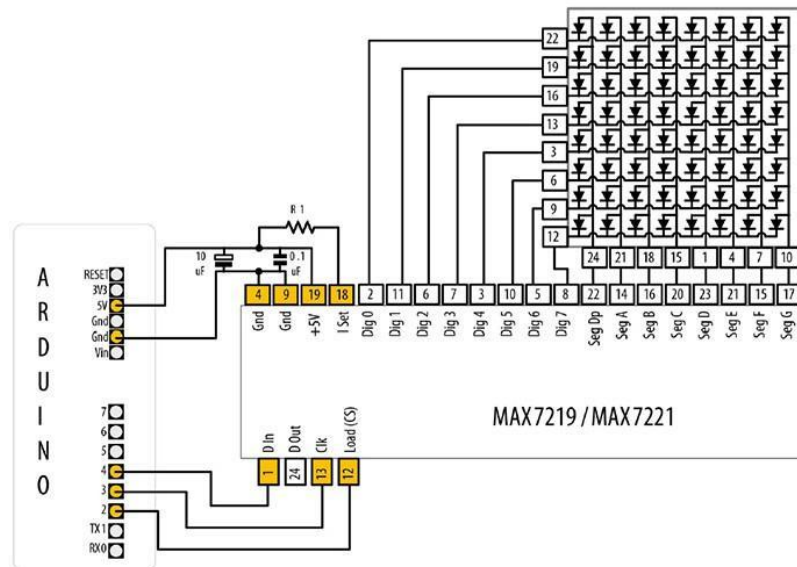


Figure 5: MAX72xx driving an 8x8 LED array.

This sketch is based on the MD\_MAX72XX library, which can display text, draw objects on the display, and perform various transformations on the display. You can find the library in the Arduino Library Manager.

```
#include <MD_MAX72xx.h>

// Pins to control 7219

#define LOAD_PIN 2

#define CLK_PIN 3

#define DATA_PIN 4

// Configure the hardware

#define MAX_DEVICES 1

#define HARDWARE_TYPE MD_MAX72XX:
:PAROLA_HW MD_MAX72XX mx =
```



```
MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, LOAD_PIN,  
MAX_DEVICES);
```

```
void setup () { mx. begin (); }
```

```
void loop () {  
  
  mx. clear (); // Clear the display  
  
  // Draw rows and columns  
  
  for (int r = 0; r < 8; r++)  
  
  { for (int c = 0; c < 8; c++) {  
  
    mx. setPoint ( r , c , true );  
  
    // Light each LED delay ( 50 );  
  
  }  
  
  // Cycle through available brightness levels  
  
  for (int k = 0; k <= MAX_INTENSITY; k++) {  
  
    mx. control (MD_MAX72XX::  
      INTENSITY, k ); delay ( 100 );  
  
  }  
  
}
```

A matrix is created by passing the hardware type, pin numbers for the data, load, and clock pins, and also the maximum number of devices (in case you are chaining modules). loop clears the display, then uses the setPoint method to turn pixels on. After the sketch draws a row, it cycles through the available brightness intensities and moves on to the next row.

The pin numbers shown here are for the green LEDs in the dual-color 8x8 matrix, available from Adafruit (part number 458). This sketch will work with a single-color matrix as well, since it only uses one of the two colors. If you find that your matrix is displaying text backward or not in the orientation you expect, you can try changing the hardware type in the line #define HARDWARE\_TYPE MD\_MAX72XX::PAROLA\_HW from PAROLA\_HW to one of GENERIC\_HW, ICSTATION\_HW, or FC16\_HW.

The resistor (marked R1 in figure 5) is used to control the maximum current that will be used to



Co-funded by the  
Erasmus+ Programme  
of the European Union



drive an LED. The MAX72xx datasheet has a table that shows a range of values. The green LED in the LED matrix shown in figure 5 has a forward voltage of 2 volts and a forward current of 20 mA. Table of resistor values (from MAX72xx datasheet) indicates  $28k\Omega$ , but to add a little safety margin, a resistor of  $30k\Omega$  or  $33k\Omega$  would be a suitable choice. The capacitors ( $0.1\mu F$  and  $10\mu F$ ) are required to prevent noise spikes from being generated when the LEDs are switched on and off.



## RGB LEDs

A Light-Emitting Diode (LED) is a small component that illuminates when current flows through it. RGB LEDs (Figure 6) operate on the same principle, but they internally contain three LEDs (Red, Green, and Blue) capable of combining to produce nearly any color output.



Figure 6: RGB LEDs.

The RGB color model is a way to represent colors by mixing red, green, and blue light (Figure 7). Each color channel's intensity determines the overall color displayed. Combining these primary colors at different levels generates millions of colors visible to the human eye. For example, to create purely blue light, you have to adjust the blue LED to the highest intensity while setting the green and red LEDs to the lowest. But, for white light, all three LEDs have to be set to their highest intensity.

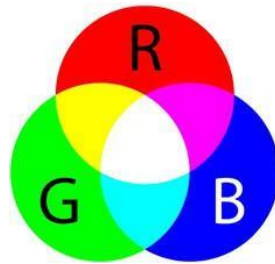


Figure 7: RGB color model.

RGB LEDs contain three LEDs inside, and usually, these three LEDs share a common anode or cathode. This categorizes RGB LEDs as either a common anode or a common cathode type (Figure 8).

### Circuit 5. Using a RGB LED

To achieve different colors with an RGB LED you need to control the brightness of each internal LED. This can be accomplished by using PWM signals with an Arduino.

In the circuit shown in the figure 9 the cathode is connected to GND, and the three anodes are connected to three digital pins on the Arduino Board through 220 Ohms resistors. It is important



Co-funded by the  
Erasmus+ Programme  
of the European Union



that the pins you use in your Arduino can output PWM signals.

The following code will make the RGB LED change a few colors:

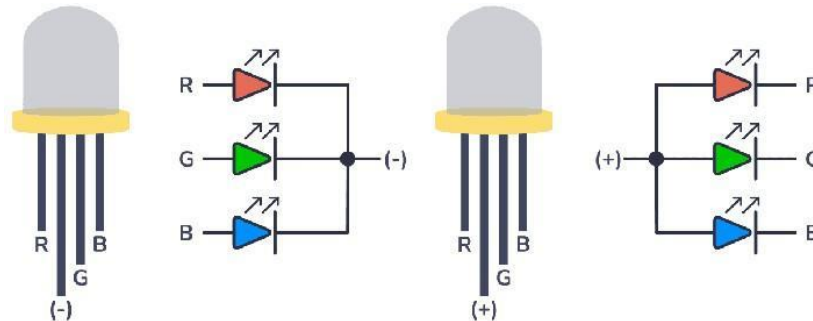


Figure 8: Types of RGB LEDs.

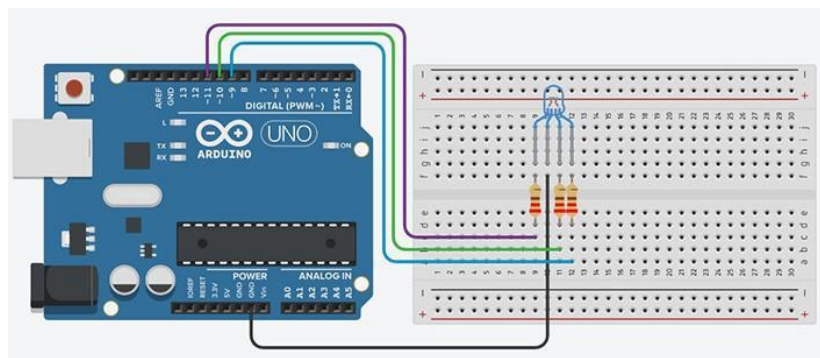


Figure 9: Connect an RGB LED to an Arduino.

```
// Declare the PWM LED pins
```

```
int redLED = 9 ;
```

```
int greenLED = 10 ;
```

```
int blueLED = 11 ;
```

```
void setup ( ) {
```

```
    // Declare the pins for the LED as Output pinMode (
    redLED , OUTPUT) ;
```

```
    pinMode ( greenLED , OUTPUT) ;
```

```
    pinMode ( blueLED , OUTPUT) ;
```

```
}
```

```
// A simple function to set the level for each color from 0 to 255
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
void setColor (

int redValue ,

int greenValue ,

int blueValue ) {

    analogWrite ( redLED , red Value ) ;
    analogWrite ( greenLED , green Value ) ;
    analogWrite ( blueLED , blue Value ) ;

}

void loop () {

    // Change a few colors

    setColor(255,0,0); // Red Color
    delay(1000);
    setColor(0,255,0); // Green Color
    delay(1000);
    setColor(0,0,255); // Blue Color
    delay(1000);
    setColor(255,255,0); // Yellow
    delay(1000);
    setColor(0,255,255); // Cyan
    delay(1000);
    setColor(255,0,255); // Magenta
    delay(1000);
    setColor(255,255,255); // White
    delay(1000);

}
```

In the setup function, pins 9, 10, and 11 are configured as outputs. The loop function repeatedly



calls the `setColor` function to display different colors at one-second intervals. The `setColor` function takes three parameters (red, green, and blue values) which can range from 0 to 255. These values are used in the `analogWrite` function, which outputs PWM signals to control the intensity of each RGB LED channel color.

### Circuit 6. Control RGB LED with potentiometer

In this example (Figure 10), we are going to modify the color of the RGB LED when we turn the potentiometer knob.

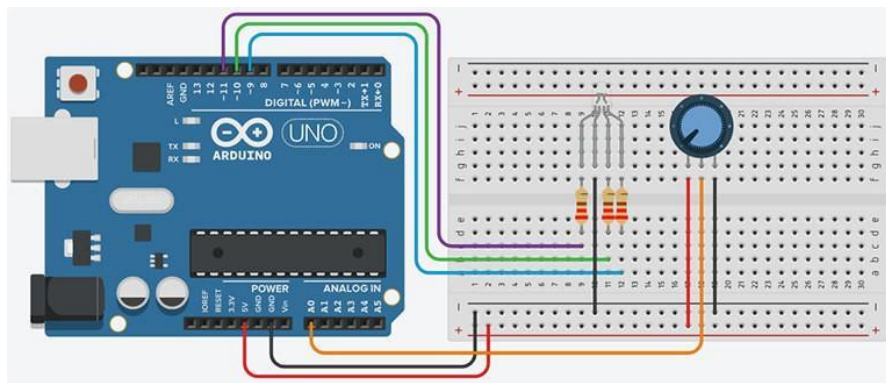


Figure 10: Arduino circuit with RGB LED and potentiometer.

Let's write the code for that.

```
#define RGB_RED_PIN 11
#define RGB_BLUE_PIN 10
#define RGB_GREEN_PIN 9
#define POTENTIOMETER_PIN A0

void setup ()
{
  pinMode (RGB_RED_PIN,
    OUTPUT) ; pinMode
    (RGB_BLUE_PIN, OUTPUT) ;
  pinMode (RGB_GREEN_PIN,
    OUTPUT) ;
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
void loop ()
{
    int potentiometer Value = analogRead (POTENTIOMETER_PIN);

    int rgb Value = map( potentiometer Value, 0 , 1023 , 0 , 1535 );

    int red ;

    int blue;
    int green;

    if ( rgb Value < 256 ) {
        red = 255 ;

        blue = rgb Value ;
        green = 0 ;
    }

    elseif ( rgb Value < 512 ) {
        red = 511 = rgb Value ; blue
        = 255 ;

        green = 0 ;
    }

    elseif ( rgb Value < 768 ) {
        red = 0 ;

        blue = 255 ;
        green = rgb Value = 512 ;
    }

    elseif ( rgb Value < 1024 ) {
        red = 0 ;

        blue = 1023 = rgb Value ;
        green = 255 ;
    }
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
elseif ( rgb Value < 1280 ) {  
    red = rgb Value = 1024 ; blue  
    = 0 ;  
  
    green = 255 ;  
}  
  
else {  
    red = 255 ;  
  
    blue = 0 ;  
  
    green = 1535 = rgb Value ;  
}  
  
analog Write (RGB_RED_PIN, red ) ;  
analog Write (RGB_BLUE_PIN, blue ) ;  
analog Write (RGB_GREEN_PIN, green  
);  
}
```

At first, we create a definition for each pin we are going to use. One for the potentiometer, and one for each color of the LED (we write the code as if we were controlling 3 different LEDs).

In the void setup(), we initialize all LEDs (in fact, the 3 legs of the RGB LED) to OUTPUT mode. Nothing to do for the potentiometer, as an analog pin is already in input mode by default.

In the void loop(), we first read the potentiometer's value with analogRead(). This gives us a value between 0 and 1023. Because we want to choose between 1536 different options, we use the map() function to transform this value from the range 0-1023 to the range 0-1535.

So, we have 6 different steps for changing the color. Also, you can note that the first color and the last color are the same (red).

For the 1st step: we set red to 255, and we increase the blue color from 0-255, according to the rgbValue we computed (in the range 0-1535).

If the rgbValue is more than 255, we go to step number 2. Now we have values from 256 to 511. We set blue to 255, and then decrease the red value. To do so, we need to subtract the rgbValue to the max value for this block, which is 511. As an example, if we enter the if structure with rgbValue = 400, then we have red = 511 - 400 = 111.

For step number 3, we keep blue to 255, and this time we increase green. The rgbValue is now



Co-funded by the  
Erasmus+ Programme  
of the European Union



between 512 and 767. So, to start from 0 and get to 255, we subtract 512 to each value we get.

Steps number 4, 5, and 6 are following the same logic as the previous steps.

Now, we have 3 values between 0-255, stored into 3 different variables. After the computation, we use `analogWrite()` on each leg of the RGB like if it were 3 different LEDs, with the corresponding values for red, blue, and green.



## NeoPixel LED

NeoPixels are intelligent RGB LED strips whose elements can be controlled individually. They use WS2812, WS2811 or WS6812 drivers and use a single wire protocol to control the colour of the embedded LED. The LEDs are integral with the controller body and they are sold assembled in various formats: flexible, matrix, ring and as individual elements.

Each cell has five pins (figure 11):

$V_{CC}$ : 5V power supply for the control circuit;

$V_{DD}$ : 5V power supply for the LED;

$V_{SS}$ : ground;

$D_{IN}$  : data input;

$D_{OUT}$  : data output to be connected to the next LED in the chain.

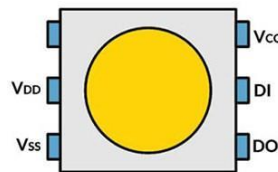


Figure 11: Pinout of a single WS2812 module.

In NeoPixel products, connections are simplified and only three pins are needed:

5V : for power supply;

GND: for ground, to be shared with the Arduino;

$D_{IN}$  : for data transmission.

Powering many LEDs requires a lot of power, so a power supply unit or battery with 5V and capable of supplying all the current required by the LEDs must be used. Between 5V and GND, it is advisable to put an electrolytic capacitor of a thousand microfarads to provide the inrush required to switch on the various LEDs. The data line must be connected to the Arduino via a  $470\ \Omega$  (figure 12).

There is no limit to the number of LEDs a NeoPixel element can contain. The only limitations are: the power consumed by the strip increases for each LED added (each LED requires a maximum of 60 mA);

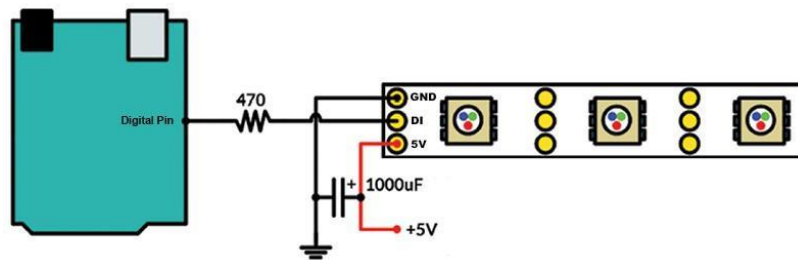


Figure 12: NeoPixel and Arduino interfacing.

the response time increases as the number of LEDs increases;

the memory required by the microcontroller increases as the number of LEDs increases.

Libraries exist to manage communication with the individual LEDs. The management library we will see is called Adafruit NeoPixel by Adafruit and can be installed via the Arduino Library Manager. To use the library, its definition must be included at the start of the sketch:

```
#include <Adafruit_NeoPixel.h>
```

The initialisation of the Adafruit\_NeoPixel object involves a number of parameters such as the number of LEDs to be controlled, the pin used for communication and the driver model:

```
Adafruit_NeoPixel pixels = Adafruit_NeoPixel (NUMPIXELS, PIN ,  
NEO_RGB + NEO_KHZ800 ) ;
```

The driver type is specified by combining various flags with the following significance:

NEO\_KHZ800: uses an 800 kHz transmission rates;

NEO\_RGB: pixels connected in RGB mode.

The pixels are initialised with:

```
pixels.begin();
```

It is possible to control the colour of each individual pixel using its index to set RGB values with:

```
pixels.setPixelColor(num_pixel, 0, 150, 0);
```

The colours are then transmitted with:

```
pixels.show();
```

The library functions also include a function to set the brightness of all LEDs:

```
strip.setBrightness(100);
```



## Circuit 7. Using the NeoPixel Strip

In this example, we will learn how to use Arduino to control the NeoPixel RGB LED strip and how to use the Adafruit NeoPixel library to set up the NeoPixels. Figure 13 shows a very simple example of connecting the NeoPixel LED strip to the Arduino board. To connect a strip of NeoPixel LEDs to an Arduino board, hook up three wires:

Power supply (+5V) goes to the plus of a power source.

Ground (GND) goes to power source ground and should be additionally connected to the board ground if powered from a separate power source.

Data input (DIN) goes to any digital pin of the board.

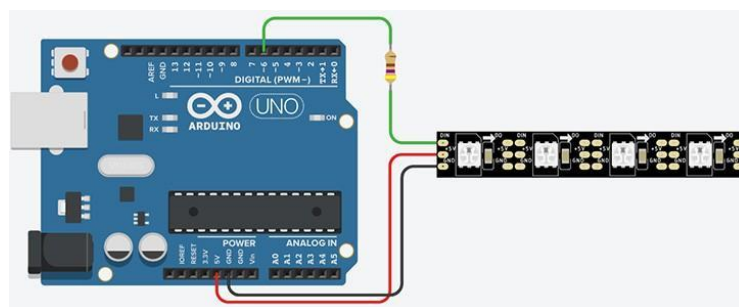


Figure 13: Connecting a NeoPixel strip to the Arduino.

The Adafruit\_NeoPixel library allows you to easily turn on a specific LED with a certain intensity and color, or to turn it on . Each LED can be controlled individually. Here we have four LEDs, with the first one numbered 0. For example, to make it light up red, you would use the command: `strip.setPixelColor(0, 255, 0, 0);` However, the LED will only respond to this command if it is followed by `strip.show()`.

```
#include <Adafruit_NeoPixel.h>
```

```
#define LED_PIN 6
```

```
#define LED_COUNT 4
```

```
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
```

```
void setup() {
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
    strip.begin (
    ); strip.clear();
    strip.show ();

}

void loop () {

    strip.setBrightness ( 50 );

    strip.setPixelColor(0,255,
    0,0); strip.show ();

    delay ( 1000 );

    strip.setPixelColor( 0, 0,0);
    0,
    strip.setPixelColor( 0, 255,0);
    1,
    strip.show();
    delay(1000);
    strip.setPixelColor( 0, 0, 0);
    1,
    strip.setPixelColor( 0, 0, 255)
    2,
    strip.show();
    delay(1000);
    strip.setPixelColor 0,0,0);
    (2,strip.setPixelCo 255,255,2
    lor(3,
    55);

    strip.show (
    ); delay ( 100
    0);

    strip.setPixelColor(3, 0,0, 0);

}
```



## Circuit 8. Lighting a NeoPixel Ring

The NeoPixel ring is a circular arrangement of individually addressable RGB LEDs, allowing for a wide range of color and brightness combinations.

In the circuit shown in Figure 14, the power supply is connected to the 5V pin, the GND pin is connected to the ground of the circuit, and the DIN pin is connected to a digital pin on the Arduino board.

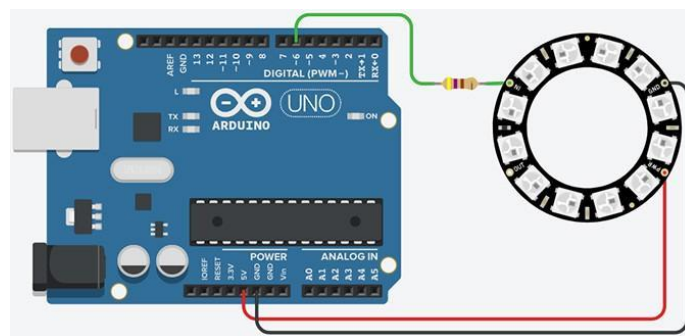


Figure 14: Neopixel ring interfacing with Arduino.

As shown in the code below, denote the pin to which the NeoPixel data input is connected and how many LEDs there are in our ring. The next step is to actually declare our NeoPixel object, which we'll call ring. In our setup() function, we call the begin() function on that ring. Then, we call show() to clear all the LEDs, and finally, we set the brightness with setBrightness(), which we call once at the beginning to tell our NeoPixel what the maximum brightness will be.

Next, let's start with a for loop that runs from 0 to the number of LEDs in our ring. Then, we set the color of the individual LED in our ring. The function setPixelColor() takes the arguments, in order: numLED, red, green, and blue. We'll enter i as the numLED so that the for loop runs through each one. For the color, we use random values for red, green, and blue. Call ring.show() afterwards to actually update the color in the ring. Finally, we add a delay of 50 milliseconds after applying each color to create a loading animation effect.

Next, we take the same loop and reverse it. To do this, we start from the last LED and count backwards to 0. Then, we set the pixel color to 0, 0, 0, which means no color, and add those same two lines..

```
#include <Adafruit_NeoPixel.h>
```

```
#define LED_PIN 6
```

```
#define LED_COUNT 16
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
Adafruit_NeoPixel ring(LED_COUNT, LED_PIN, NEO_RGB + NEO_KHZ800);
```

```
void setup()
```

```
{
```

```
  ring.begin()
```

```
  ;
```

```
  ring.show();
```

```
  ring.setBrightness(50);
```

```
}
```

```
void loop() {
```

```
  for(int i = 0; i < ring.numPixels(); i++){
```

```
    ring.setPixelColor(i, random(255), random(255),  
    random(255));
```

```
    ring.show();
```

```
    delay(50);
```

```
  }
```

```
  for(int i = ring.numPixels() - 1; i >= 0; i--)
```

```
  { ring.setPixelColor(i, 0, 0, 0);
```

```
    ring.show()
```

```
    ; delay(50);
```

```
  }
```

```
}
```

## Circuit 9. Controlling a NeoPixel Ring

This sketch uses the Adafruit Neopixels library (installed using the Arduino Library Manager) to change LED colors based on readings from an analog pin. Figure 15 shows the connection for a NeoPixel ring and a potentiometer to control the color:

```
#include <Adafruit_NeoPixel.h>
```

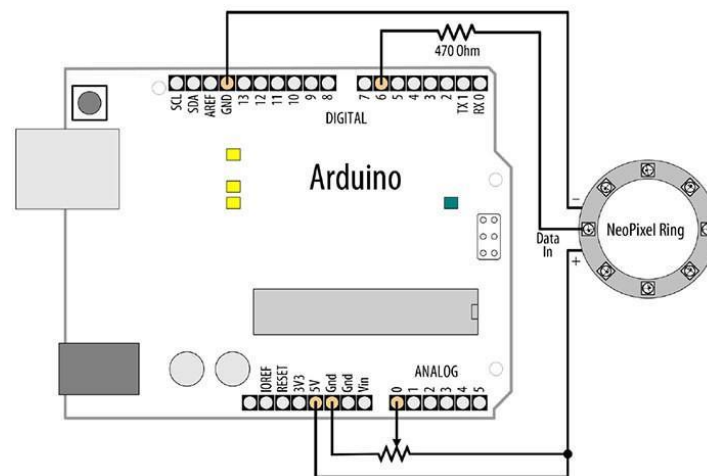


Figure 15: Connecting a NeoPixel ring.

```
const int sensor Pin = A0 ;    // analog pin for sensor

const int led Pin = 6 ;        // the pin the LED strip is connected to

const int count = 8 ;          // how many LEDs in the strip

// declare LED strip
Adafruit_NeoPixel leds = Adafruit_NeoPixel ( count , led Pin , NEO_GRB + NEO_KHZ800 ) ;

void setup () {

    leds.begin () ; // initialize LED strip
    for (int i = 0 ; i < count ; i++) {

        leds.setPixelColor ( i , leds.Color ( 0 , 0 , 0 ) ) ;    // turn each LED off
    }

    leds.show () ; // refresh the strip with the new pixels values ( all off )
}

void loop () {

    static unsigned int last_reading = 1 ;

    int reading = analogRead ( sensor Pin ) ;

    if ( reading != last_reading ) {    // If the value has changed
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
// Map the analog reading to the color range of the Neo Pixel

unsigned int mappedSensorReading = map( reading , 0 , 1023 , 0 , 65535 );

// Update the pixels with a slight delay to create a sweeping effect

for( int i = 0 ; i < count ; i ++ ) {

    leds.setPixelColor(i, leds.gamma32( leds.ColorHSV(
        mappedSensorReading , 255 , 128 ) ) );

    leds.show();

    delay( 25 );

}

last_reading = reading ;

}
```

You specify the number of LEDs in the strip count, the Arduino pin the data line is connected to ledPin, and the type of LED strip you are using (in this case: NEO\_GRB+NEO\_KHZ800). To set the color of an individual LED you use the led.setPixelColor method. You need to specify the number of the LED (starting at 0 for the first one) and the desired color. To transfer data to the LEDs you need to call led.show. You can alter multiple LED's values before calling led.show to make them change together. Values not altered will remain at their previous settings. When you create the Adafruit\_NeoPixel object, all the values are initialized to 0.

The NeoPixel library includes its own function for converting a hue to an RGB value: ColorHSV. The first argument is the hue, the second is the color saturation, and the third is brightness. The gamma32 function performs a conversion on the output of ColorHSV to compensate between the way that computers represent colors and the way that humans perceive them.



Co-funded by the  
Erasmus+ Programme  
of the European Union



## **Erasmus+ KA210-VET**

### **Small-scale partnerships in vocational education and training**

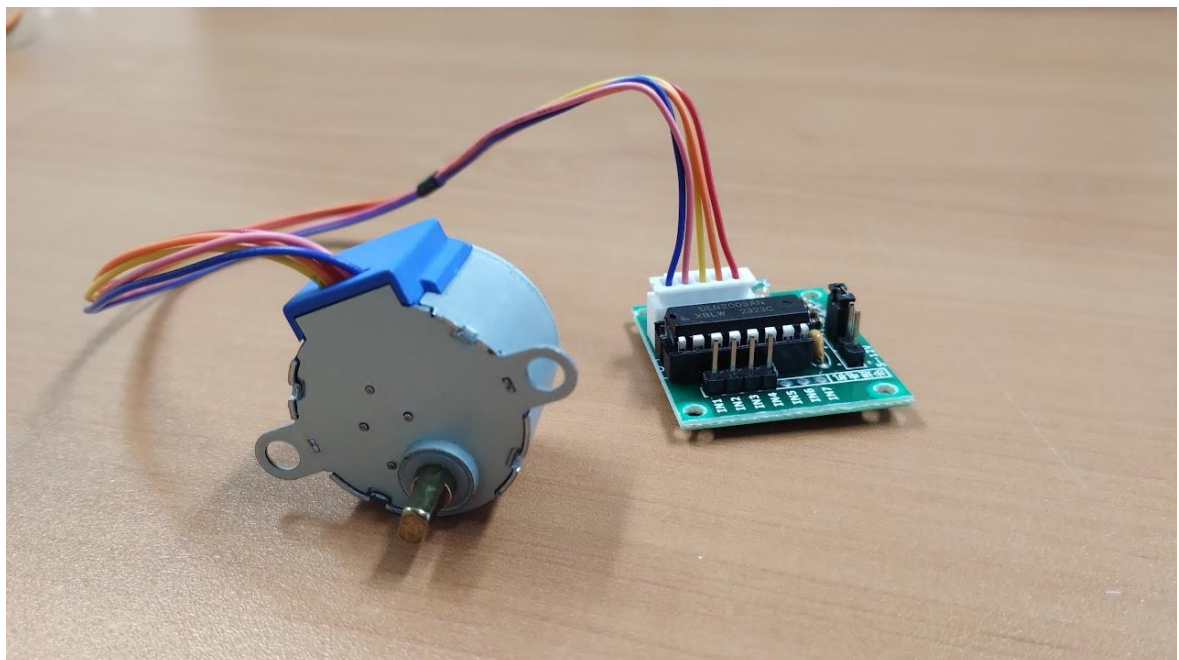
**Project Title: “Using Arduinos in Vocational Training”**

**Project Acronym: “UsingARDinVET”**

**Project No: “2023-1-RO01-KA210-VET-000156616”**

#### **INTRODUCTION to ARDUINOS**

**(Motors module and training kit)**





## DC MOTORS

### Overview

A DC (Direct Current) motor is a type of electric motor that runs on direct current electricity. It is one of the most widely used types of motors due to its simplicity, reliability, and ability to provide continuous rotation. DC motors are commonly found in appliances, toys, tools, and various other devices that require mechanical movement.

In this document, we will explain the basic working principle of DC motors, their components, and how they operate.

### Basic Principle of DC Motor Operation

A DC motor works on the principle of electromagnetic induction, which states that when a current-carrying conductor is placed in a magnetic field, it experiences a force that causes it to move. This force is known as the Lorentz force.

#### Key Principle:

A current flowing through a coil of wire generates a magnetic field around the coil.

The coil is placed in an external magnetic field (usually created by permanent magnets or electromagnets).

The interaction between the magnetic field of the coil and the external magnetic field generates a force, causing the coil to rotate.

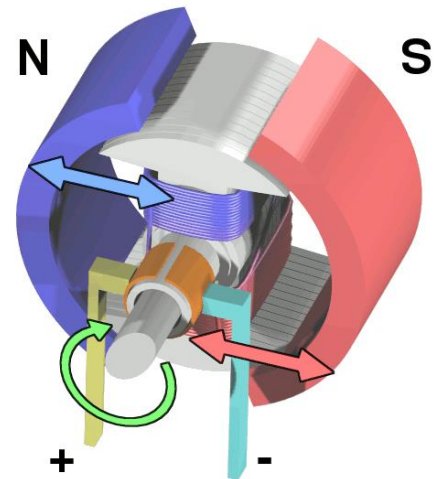


Figure SEQ Σχήμα \\* ARABIC 1 Elements of a dc

This rotational motion is transferred to the motor's shaft, providing mechanical movement.

### Factors Affecting DC Motor Operation

Several factors can influence the performance of a DC motor:

- **Voltage:** The speed and torque of a DC motor are directly related to the voltage applied to it. Higher voltage generally results in higher speed and torque.
- **Speed:** Increasing the applied voltage increases the motor's speed because it increases the current through the rotor windings, producing a stronger magnetic field and greater force.
- **Torque:** The torque, or rotational force, is proportional to the current. Higher current means higher torque.
- **Current:** The current determines how much torque the motor can produce. Higher current increases the torque but may also lead to overheating if the motor is not properly cooled.



Co-funded by the  
Erasmus+ Programme  
of the European Union



- **Resistance:** The resistance of the rotor windings affects the current flow. Higher resistance limits the amount of current that can flow, which reduces the motor's torque. On the other hand, lower resistance allows more current to flow, increasing torque and speed.
- **Magnetic Field Strength:** The strength of the stator's magnetic field also affects the motor's performance. Stronger magnetic fields result in greater interaction between the stator and rotor, producing more force and thus higher performance.

## Applications

Electric DC motors are used when a smooth continuous rotation is needed. Plain DC motors are voltage controlled so we can decide how much speed and torque the motor will produce. Plain DC motors can't be used when precision control over speed or position of the motor is needed. In these cases, we usually choose step motors or servomotors instead.

## Power requirements

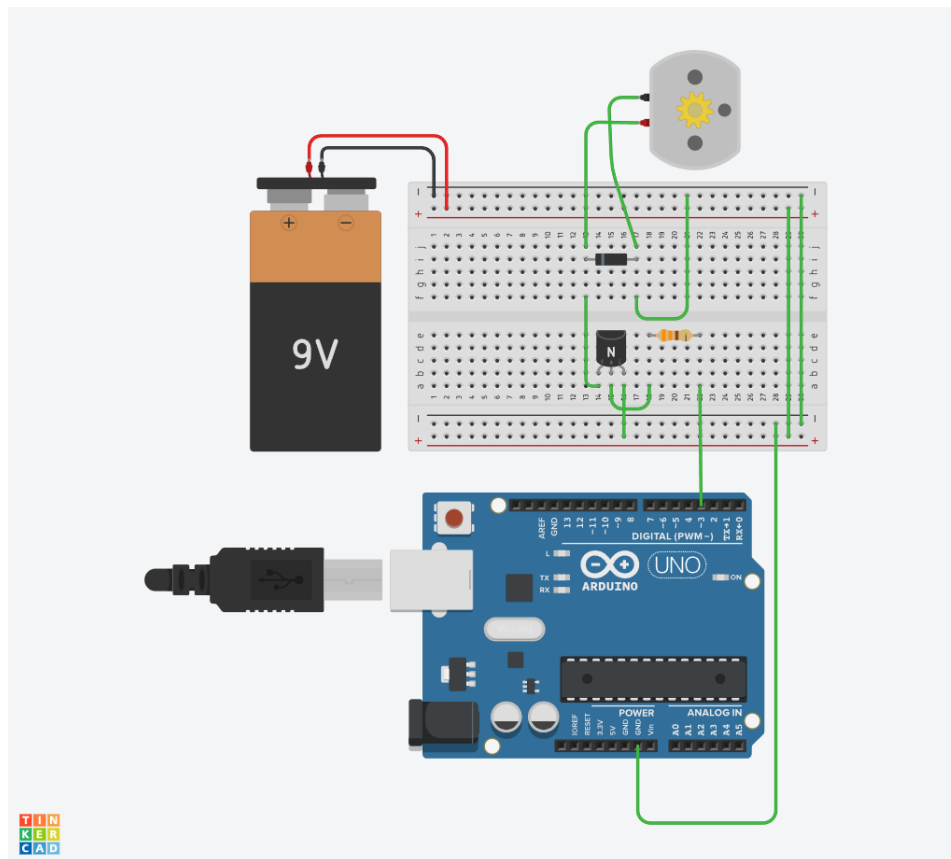
Most of the available DC motors consumptions are higher than the maximum current than Arduino can deliver and can cause damage to Arduino digital outputs, so we need external power supply to the motor and control this power supply using Arduino's output pins. The most usual way to control DC motors in Arduino is using a motor controller or building our own controller using a transistor.



## Circuit 1

**Circuit Title:** Driving a DC motor with a transistor

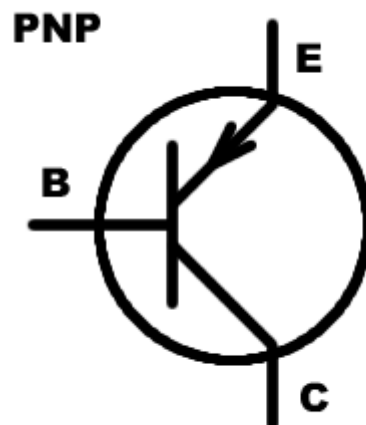
**Circuit Description:** This circuit uses a PNP 2N2222 transistor to control the 9V power supply using Arduino.



## Circuit 1

To build this you will need:

- 1 DC motor
- 1 PNP 2N2222 transistor
- 1 330 $\Omega$  resistor
- 1 Zener diode
- 1 9V battery



PNP Transistors can be used as a electronically controlled switch. The current will flow from emmitter pin (E) to colector pin (C) only when enought current flows from base pin (B) to colector (C). In other words we can control the current between E and C pins using B pin.

The circuit is very simple. The transistor will allow to pass 9V current from the batery to the motor only when pin 3 is active. As pin 3 is a PWM pin we can send different pulse wifth to control the speed of the motor using analogWrite method.

The diode is used to avoid the motor to send current to the circuit due to inertial movement of the motor. The resistance is used to adapt the output of the arduino pin to the correct input values of the transistor.

The code to control the motor using pin 3 is:

```
int motorPin = 3;                                //The pin attached to the motor

void setup()
{
  pinMode(motorPin, OUTPUT);                      //Defines the pin 3 as output
}

void loop()
{
  analogWrite(motorPin, 255);                      //Full speed
  delay(2000);
  analogWrite(motorPin, 100);                      //Medium Speed
  delay(2000);
  analogWrite(motorPin, 10);                      //Low speed
  delay(2000);
}
```

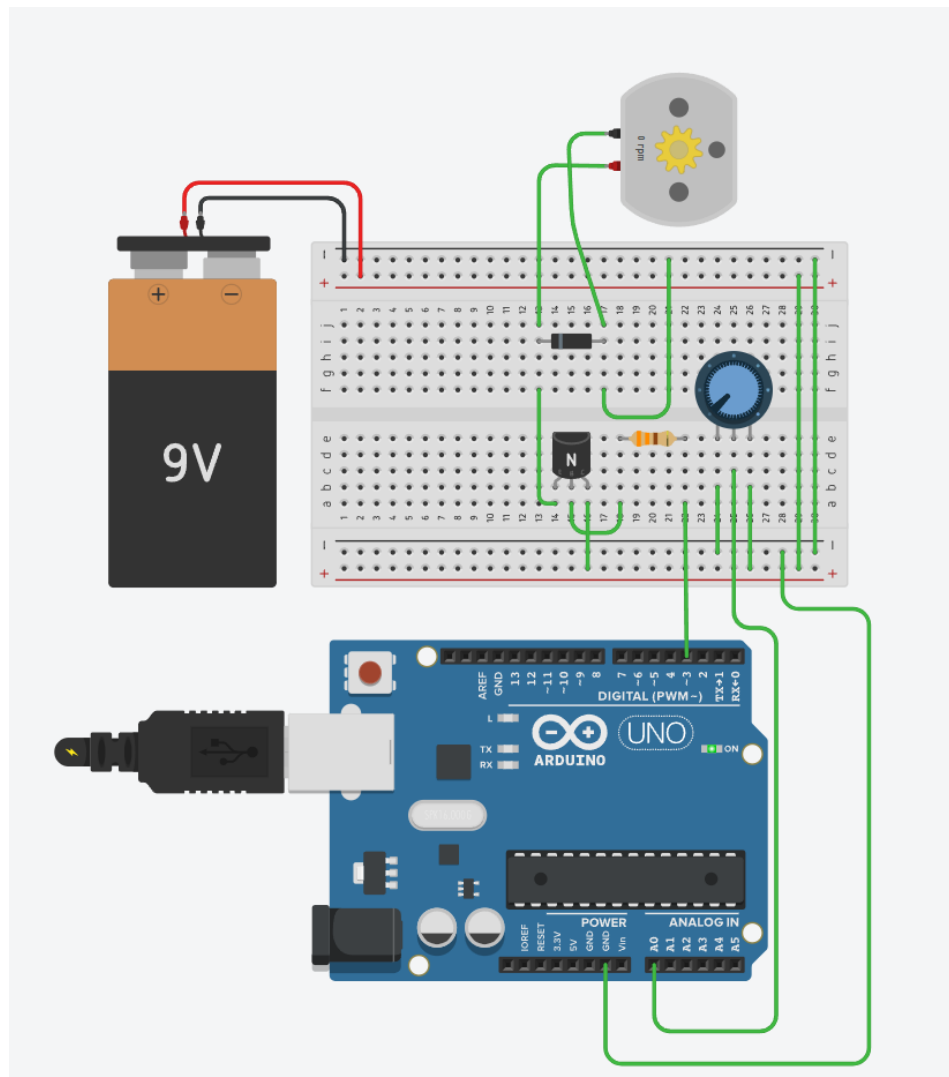


## Circuit 2

**Circuit Title:** Controlling a DC motor speed using a potentiometer

**Circuit Description:** This circuit uses a PNP 2N2222 transistor to control the 9V power supply using Arduino. Also a potentiometer is read using arduino analog pin A0 and used to control the speed of the motor.

Lets modify out circuit to add a potentiometer to control motor's speed.



**Circuit 2**

Example program 2: Controlling the motor with a potentiometer

```
int motorPin = 3;           //The pin attached to the motor
int controlPin = 0;         //The pin attached to the pot
int speed = 0;              //The speed assigned to the motor

void setup()
{
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
pinMode(motorPin, OUTPUT);    //Set up the motor pin as output pin
}

void loop()
{
    speed = analogRead(controlPin);    //Read the potentiometer value and store it on //speed
                                        variable
    analogWrite(motorPin, speed);    //Send speed to the motor
    delay(500);                    //Wait 500 milliseconds
}
```



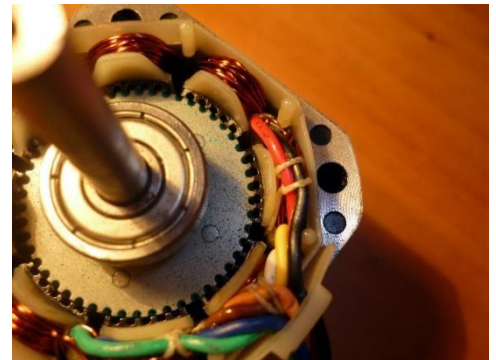
## STEPPER MOTORS

### Overview

A stepper motor is a type of electric motor that moves in discrete steps, making it ideal for precise control of position, speed, and direction. Unlike traditional DC motors, which continuously rotate, stepper motors divide a full rotation into multiple steps, each typically ranging from  $0.9^\circ$  to  $1.8^\circ$  per step, depending on the motor design. This unique feature allows them to achieve very accurate and repeatable movements without the need for encoders or other position-sensing devices.

### How Stepper Motors Work

The operation of a stepper motor is based on electromagnetism. Inside a stepper motor, there are multiple coils or windings arranged around a central rotor. When a current is passed through a specific coil, it creates a magnetic field that interacts with the rotor, causing it to rotate by a certain angle. By sequentially energizing different coils in a controlled pattern, the rotor is moved in precise increments (steps).



The key to controlling the movement of a stepper motor lies in the driver circuit, which regulates the sequence and timing of current pulses sent to the coils. The driver can operate in different modes, such as:

- **Full step:** Energizes one coil at a time, giving a larger step size (e.g.,  $1.8^\circ$  per step).
- **Half step:** Energizes two coils alternately, allowing smaller steps and smoother motion.
- **Microstepping:** Divides each full step into smaller steps, improving smoothness and resolution, and reducing vibration.

### Advantages of Stepper Motors

- **Precision and accuracy:** Stepper motors are highly accurate, making them ideal for applications that require exact positioning, such as 3D printers, CNC machines, and robotic arms.
- **No feedback required:** Stepper motors can operate without external feedback systems (like encoders) because the position of the rotor is determined by the number of steps moved.
- **Reliability:** These motors are simple in construction, leading to fewer parts that could fail.



Co-funded by the  
Erasmus+ Programme  
of the European Union



- **Low cost:** Stepper motors are relatively inexpensive, making them a cost-effective solution for many applications.

### Disadvantages of Stepper Motors

- **Low efficiency:** Stepper motors can be less energy-efficient compared to other types of motors, especially at higher speeds.
- **Torque drop at high speeds:** At high speeds, stepper motors tend to lose torque, which can affect performance in certain applications.
- **Vibration and noise:** Stepper motors can produce vibrations and noise, especially when operating at lower speeds or with low current.

### Applications for Stepper Motors

Stepper motors are widely used in applications that require precise control of position and rotation, including:

- **3D printers:** To accurately move the print head and build up the material layer by layer.
- **Robotics:** For precise control of robotic arms and other moving parts.
- **CNC machines:** To move the tool or workpiece in controlled increments.
- **Camera platforms:** For precise pan-and-tilt control in photography and videography.
- **Medical equipment:** For applications such as pumps, prosthetics, and diagnostic machinery.

Stepper motors are an essential component in many applications where precision and control are critical. Their ability to divide full rotation into small, discrete steps allows for accurate positioning, and their simplicity makes them a popular choice for both hobbyists and industrial engineers. Although they may not be the best option for high-speed, high-efficiency applications, their advantages in control and reliability make them indispensable in fields such as robotics, manufacturing, and automation. Usually, stepper motors designs use sensors to detect when the moving objects reach the beginning or end of the path, these sensors are called limit switch sensors and are essential to determine the initial position of a stepper motor.



Co-funded by the  
Erasmus+ Programme  
of the European Union

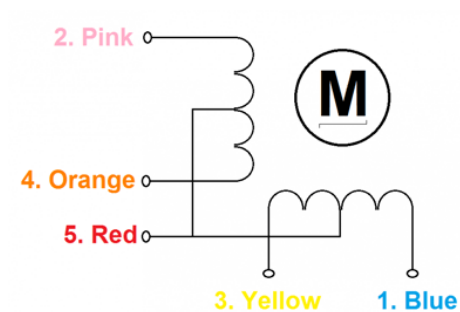


### Circuit 3:

**Circuit Title:** Moving a stepper motor one revolution forward and one revolution backwards

**Circuit Description:** This circuit uses a 28BYJ-48 stepper and a ULD2003 based controller to provide the power supply to the stepper motor and avoid damaging the Arduino board.

In this example we are going to use a small stepper motor called 28BYJ-48



**Figure 2 Connections of 28BYJ-48**

This motor has four coils with a unipolar scheme cabling and is usually controlled by a ULN2003 based circuit that provides current to each coil to avoid damaging the I/O pins of our Arduino.

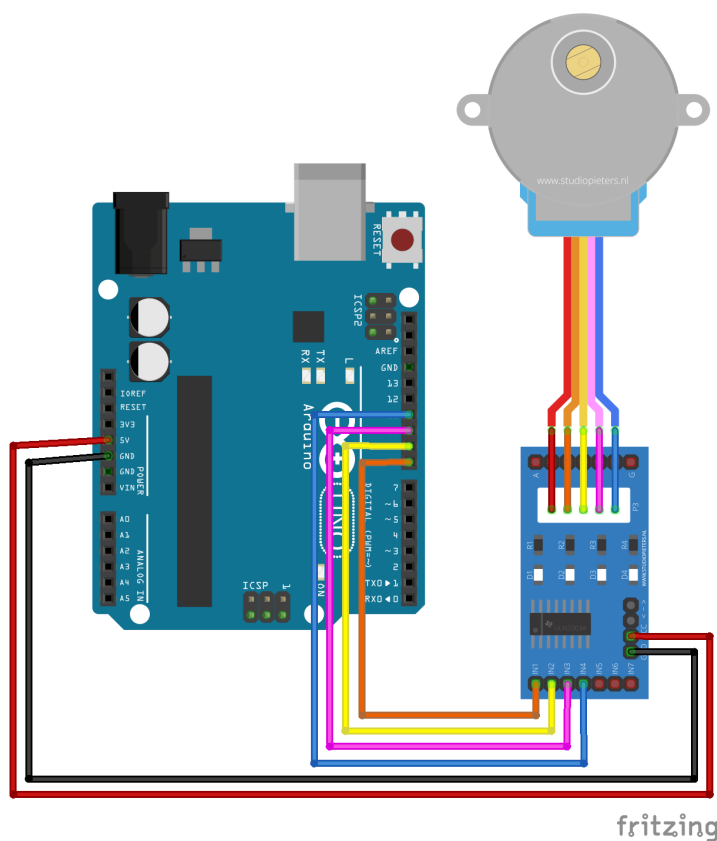
The 28BYJ-48 motor has 64 steps per revolution when use in half-step mode and has a internal 1:64 gear ratio so  $64 * 64 = 4096$  total steps per revolution.

In this project we are going to use:

- 1 28BYJ-48 Stepper motor
- 1 ULN2003 based controller
- Cables
- Arduino Uno board

Connect the stepper motor to the controller using the appropriate plug:

Now connect the controller to the Arduino board following this schema:



### Circuit 3:

To control the stepper motor we are going to use the Stepper.h library. The most useful methods of Stepper library are:



stepper(steps, pin1, pin2, pin3, pin4)	Creater a new stepper with the following parameters:  Steps: Number of steps per revolution  Pin1 and Pin 2: Pins attached to the motor  Pin 3 and Pin 4: (Optional): Pins attached to the motor if the motor have for cables
step(steps)	Move the stepper a specific number of steps
setSpeed(rpm)	Set the motor speed to a specific revolution per minute. Please consider the maximum speed of your motor.

The code to move the motor one revolution forward and one revolution backwards is:

```
#include <Stepper.h>                // Include the steper control library

Stepper stepper(4096, 8, 10, 9, 11); // Create a stepper with 4096 steps per revolution

void setup() {
}

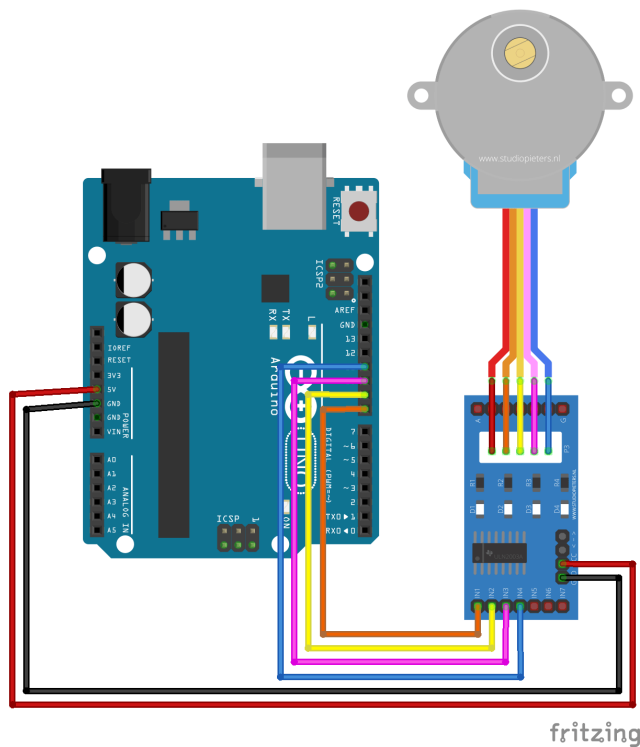
void loop {
  stepper.step(4096);                //One complete revolution
  delay(5000);                       //wait for 5 seconds
  stepper.step(-4096);               //One complete revolution backwards (negative)
  delay(5000);                       //wait for 5 seconds
}
```



### Circuit 4:

### Circuit Title: Moving a stepper motor like the seconds hand of a clock

Circuit Description: Using the same stepper and the same controller we are going to change the code to move the motor like the seconds hand of a clock: 1/60th of revolution every second.



### Circuit 4

The code to move the stepper motor  $1/60^{\text{th}}$  turn every second is:

```
#include <Stepper.h> // Include the stepper control library

Stepper stepper(4096, 8, 10, 9, 11); //Create a stepper with 4096 steps per revolution

void setup() {
  stepper.setSpeed(240) //240 rpm speed for quick seconds hand movement
}

void loop {
  stepper.step(68); //60 divided into 4096 equals aprox 68 steps
                  //per second
  delay(1000); //each second we move the motor
}
```



## SERVO MOTORS

### Overview

Servo motors are widely used in robotics, automation, and other fields where precise control of angular position is required. They differ from regular motors in that they can rotate to a specific position, rather than continuously rotating. This makes them ideal for tasks like moving robotic arms, controlling the position of a camera, or adjusting the angle of solar panels.

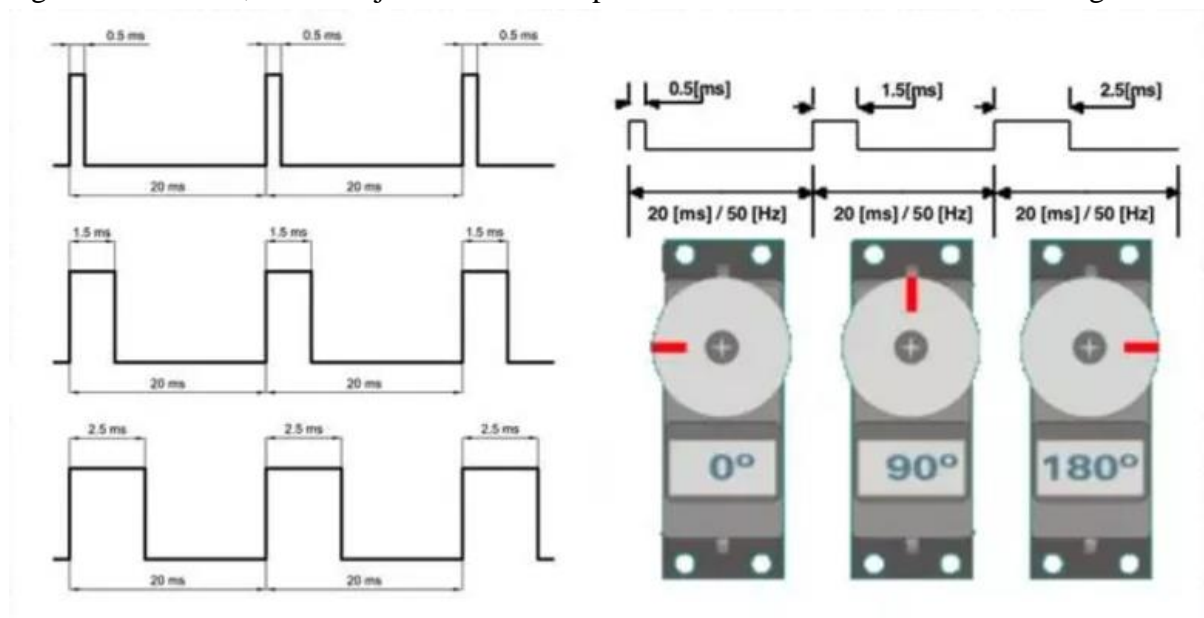
### How do servo motors work?

A servo motor is a small motor equipped with a feedback system that allows it to achieve precise angular movement. Unlike standard DC motors, which rotate continuously in either direction, a servo motor can only rotate through a limited range (typically  $0^\circ$  to  $180^\circ$ ), and its position can be controlled very precisely.

The servo motor consists of three key components:

- Motor: Drives the rotation.
- Feedback potentiometer: Monitors the motor's position.
- Control circuit: Receives the control signal and adjusts the motor's position accordingly.

Servo motors operate using Pulse Width Modulation (PWM) signals. The Arduino sends a PWM signal to the servo, which adjusts the motor's position based on the duration of the signal.



**Figure 4 How PWD controls servo motor position**

A short pulse (1 ms) might set the servo to  $0^\circ$ .



Co-funded by the  
Erasmus+ Programme  
of the European Union



A long pulse (2 ms) might set the servo to 180°.

A pulse of around 1.5 ms generally places the servo at the 90° position.

By varying the pulse width, the servo motor can be positioned anywhere within its operational range.

### Applications of Servo Motors

- **Robotics:** Servo motors are widely used in robots to control joints, wheels, or actuators.
- **Camera Gimbals:** To stabilize a camera, servo motors can control the orientation of the camera to keep it steady.
- **Antenna Positioning:** Servo motors are used in radio and satellite dishes to adjust antenna direction.
- **RC Vehicles:** Remote-controlled cars, planes, and boats often use servos to steer and control other mechanical components.

Servo motors are essential components for projects that require precise control of angles. With Arduino, controlling a servo is a simple task, thanks to the Servo library, which abstracts away the complexity of generating PWM signals. One of the biggest advantages of servos over stepper motors is that as the servo can detect its own position they usually don't need limit sensors like stepper or DC motors.

### Advantages of servo motors:

- Great precision without needing limit sensors.
- Can maintain their position even with external disturbances.

### Disadvantages of servo motors:

- Limited range of movement not suitable for continuous movement
- Limited maximum speed

### Example Project: Move a Servo motor:

The SG90 is a small and convenient servo motor that can be powered using Arduino pins. Bigger servos should have an external power supply.

For this project we are going to use:

- 1 Servo motor SG90
- Cables
- Arduino Uno board

The servo control pin (yellow) needs to be connected to a PWM digital pin, we are going to use ~3 pin.

We are going to use Servo.h library to help us to manage the servo. The most important Servo.h methods are:



Co-funded by the  
Erasmus+ Programme  
of the European Union



attach(pin, min, max)	<p>Attach the servo variable to a pin. The pin must be a PWM digital pin.</p> <p>The optional parameters min and max set the pulse width, in microseconds, that the servo expect to set the angle to minimum and maximum angle. If no min or max values are provided the default values 544 and 2400 milliseconds are used.</p>
write(angle)	<p>Set the servo angle at desired value. The servo will move to this position.</p>



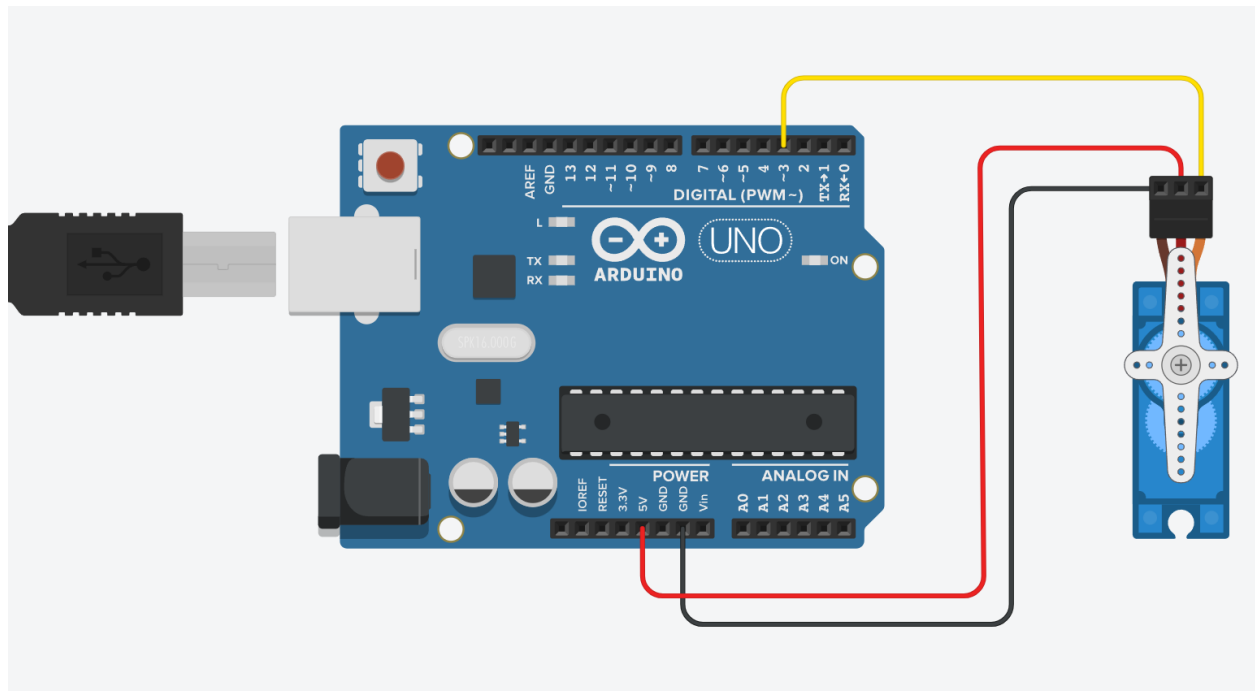
Co-funded by the  
Erasmus+ Programme  
of the European Union



## Circuit 5:

### Circuit title: Simple movement of a servo motor

Circuit description: Using digital pin 3 of Arduino we are going to position the servo motor in 90, 180 and 0 degrees angle. The servo motor is connected to 5V and GND for power and digital pin 3 in PWM mode to control the position.



## Circuit 5

```
#include <Servo.h>           //The servo library

Servo myServo;               //We need to create a servo object

void setup()
{
  myServo.attach(3);         //Configure the servo pin
  myServo.write(0);           //Set servo at 0 degrees angle
}

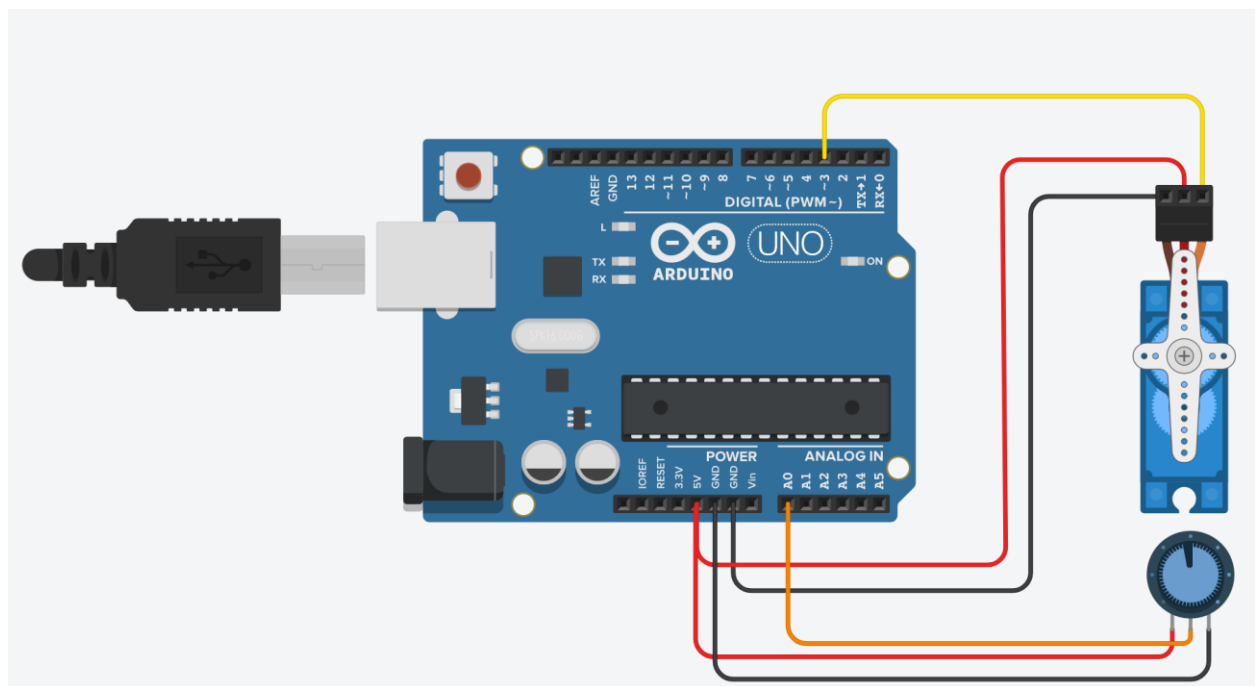
void loop()
{
  myServo.write(90);          //set servo at 90 angle
  delay(1000);                //wait for 1 second
  myServo.write(180);         //set servo at 180 angle (max)
  delay(1000);                //wait for 1 second
  myServo.write(0);           //set servo at 0 angle (min)
  delay(1000);                //wait for 1 second
}
```



## Circuit 6:

### Circuit title: Controlling a servo motor using a potentiometer

Circuit description: In this circuit a potentiometer connected to A0 pin is used to control the servo motor position. The servo connections are the same that in the previous circuit and the potentiometer is connected to 5V ground and A0 analog pin of the Arduino board.



**Circuit 6**

In this example the servo position is controlled by the potentiometer. To translate the potentiometer position to servo position the function map is used:

<code>map(value, from_min , from_max, to_min, to_max)</code>	Map a value from the [from_min, from_max] scale to the [to_min, to_max] scale.
--------------------------------------------------------------	--------------------------------------------------------------------------------

```
#include <Servo.h>;
```

```
Servo myServo;
```

```
//Creates the servo object
```

```
void setup()
```

```
{
```

```
  myServo.attach(3);
```

```
//Attach the servo to pin number 3
```

```
  myServo.write(0);
```

```
//position servo in 0 position
```

```
}
```



```
void loop()
{
  int position = map(analogRead(0),0,1023,0,175); //map the position of the potentiometer
  myServo.write(position);                       //to a angle of the servo and store the
}                                                 //mapped value in position variable
                                              //before sending it to the servo
```

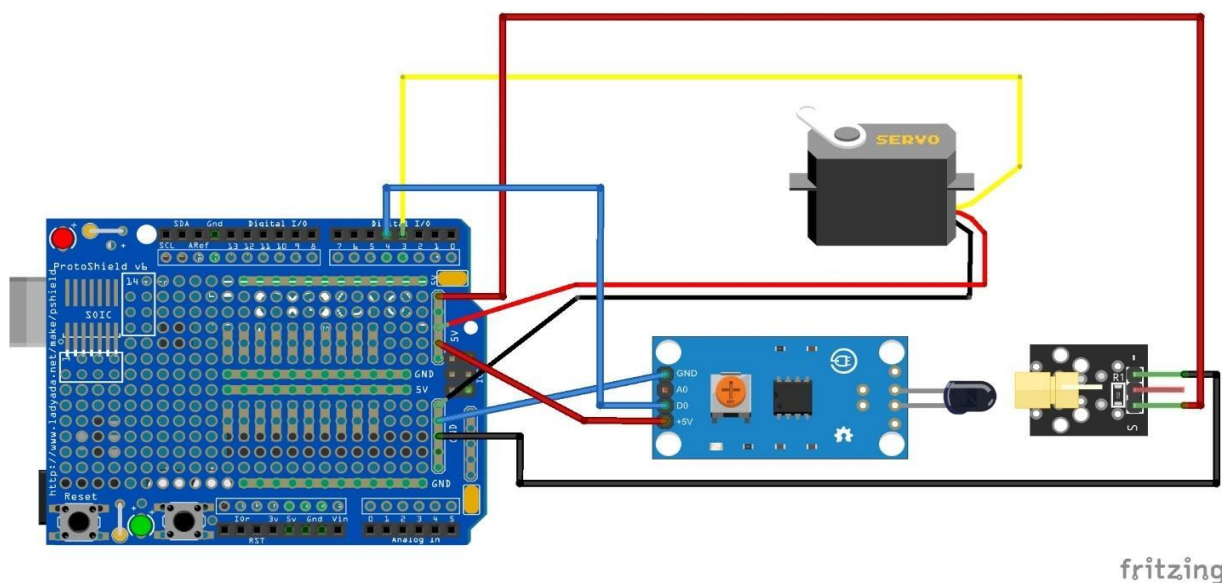
### Circuit 7:

#### Circuit title: Laser detector-controlled car barrier

Circuit description: We are going to simulate a car barrier using a servo. To detect the card a LASER light emitter and a light detector are used. In order to facilitate the connections a proto shield is used.

For this project we are going to use:

- 1 Laser emitter module
- 1 Light detector module
- 1 Servo motor SG90
- 1 Proto shield
- Cables
- Support board
- Barrier
- Car
- Arduino Uno board



fritzing

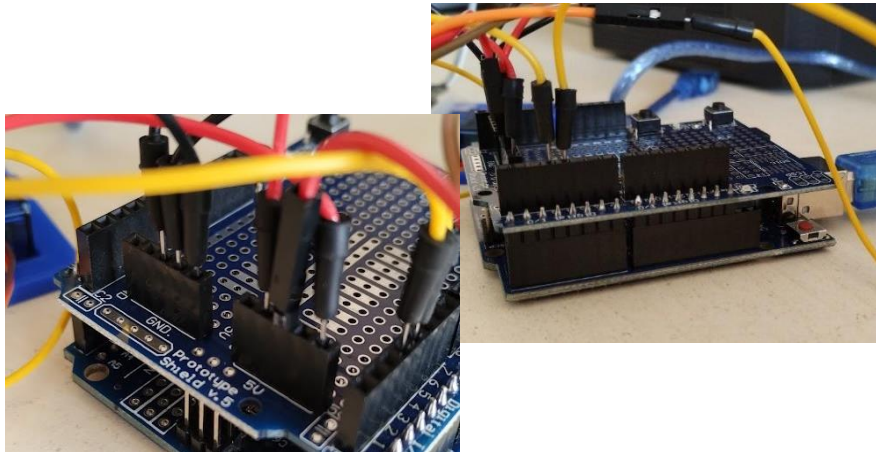
### Circuit 7



Co-funded by the  
Erasmus+ Programme  
of the European Union



Connect the proto shield over the Arduino board. Please note that there are two headers for 5V and GND that are very convenient to connect the power cables of the sensors and the servo motor.



Connect the – and S pin of the LASER module to the GND and 5V headers of the Proto Shield:



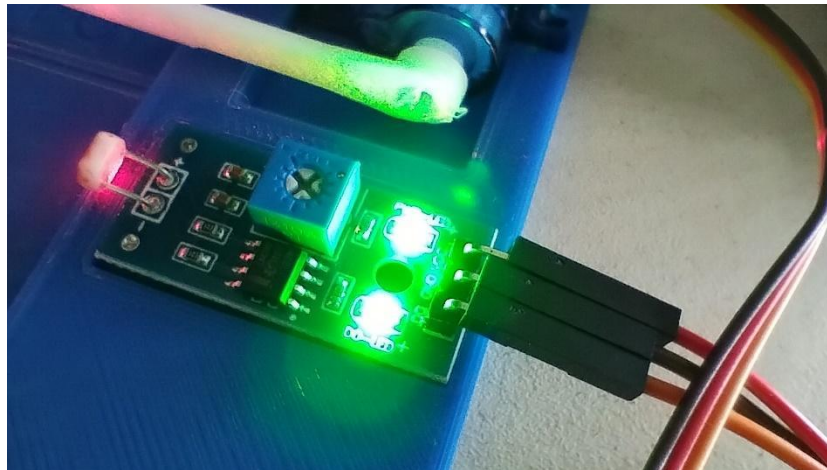
Connect the VCC, GND and module to the corresponding

LASER Emitter Module	
S	5V
Middle pin	Not Connected
-	GND

D0 pins of the light detector pins of the proto shield:



Co-funded by the  
Erasmus+ Programme  
of the European Union



Light Detector Module	
D0	Digital Pin 4
GND	GND
VCC	5V

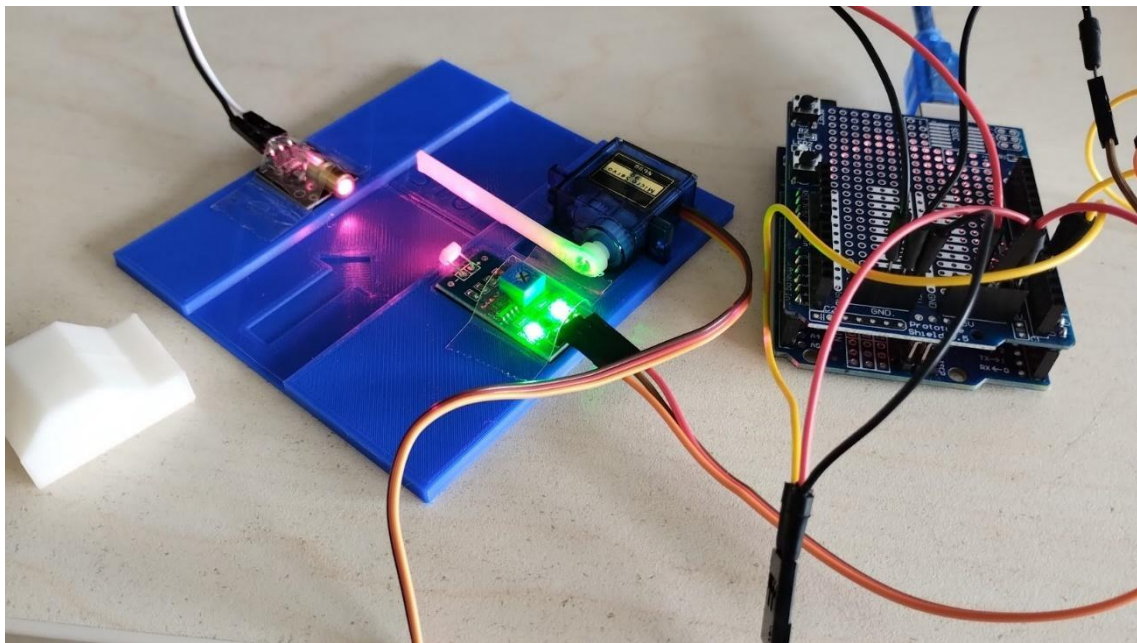
Turn on Arduino, point the LASER to the light sensor and adjust the sensibility of the light using a screwdriver on the blue potentiometer of the light sensor module. The left light should only turn on when the LASER hits on the sensor as in the image above.

Connect the Servo pins to power and data pin of the proto shield:

Servo motor	
Brown	GND
Red	5V
Yellow	Digital Pin ~3

Attach the barrier to the servo motor. You may need to adjust the angle of the barrier if the servo motor is not on the correct position.

Put all the elements in the support board and fix them with adhesive tape or blue tack to make sure they stay in place.



**Figure 5 Servo controlled barrier**

### Controlling a Servo Motor with Arduino

To control a servo motor with an Arduino, we can use the Servo library, which makes the process easy and convenient. The library provides functions for attaching the servo to a pin, setting the position, and moving the servo smoothly to the desired angle.

To avoid false object detection the program will read the sensor each 250ms. The program will consider that there is a car waiting only when 4 readings in a row detect an obstacle between the LASER and the light sensor.

```
#include <Servo.h>
```

```
Servo myServo;  
int detections;
```

```
int sensorPin = 4;           //Light sensor pin  
int servoPin = 3;            //Servo Motor pin. Must be a PWM pin  
int open = 0;                //Position of servo on opened state  
int close = 128;             //Position of servo on closed state  
int threshold = 4;           //Number of row reads to be considered as detection  
int waitToPass = 4000;       //Time to wait for the car to pass
```

```
void setup() {  
  myServo.attach(servoPin);  //Initialize the servo motor  
  myServo.write(close);      //Set barrier on close position  
  pinMode(sensorPin, INPUT); //Set sensor pin as input pin  
  detections = 0;            //Initialize the number of object detection  
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
void loop() {  
  
  if (digitalRead(sensorPin)==1){          //If there is something detected  
    detections = min ( threshold, detections +1); //Increment number of detections by one  
                                              //until reaching threshold  
  } else {                                  //If no object is detected  
    detections = max ( 0, detections -1);    //Decrement number of detections by one  
                                              //until reaching 0  
  }  
  delay (250);                             //Four reads by second  
  
  if (detections == threshold){            //If threshold level of detection is reached  
    myServo.write(open);                  //Open barrier  
    delay (waitToPass);                   //Wait for the car to pass  
  }  
  
  if (detections == 0){                    //If no object is detected for threshold reads  
    myServo.write(close);                 //Close the barrier  
  }  
}
```





Co-funded by the  
Erasmus+ Programme  
of the European Union



## Sensors Module and Training KIT

The aim of this module is to explain how sensors work in cooperation with Arduino. Several types of Arduino-based Sensors will be presented while applications including them will be developed, in order to be used by teachers of secondary vocational education.

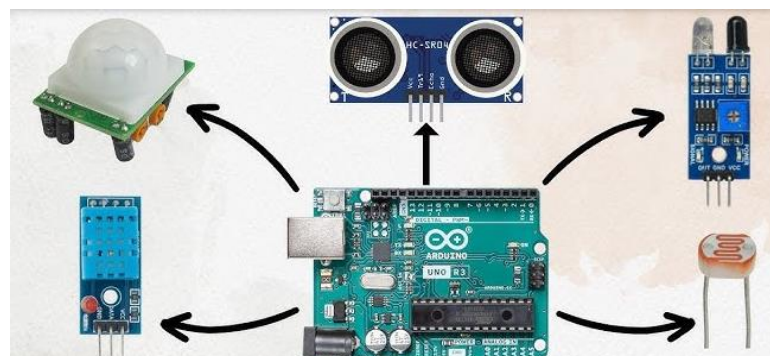
### About sensors

A sensor is a device with which the Arduino board interacts with the environment. A sensor actually receives as an input a signal and responds giving as an output an electrical signal.

To be more specific, sensors receive different kinds of signals i.e. physical, chemical or biological and respond converting them into an electric signal, in the form of current or voltage or charge. We could also say that a sensor is a translator that converts a non-electrical value to an electrical value.

There are sensors of different types based on the applications, the input signal, the conversion mechanism and the material used in sensor characteristics such as cost, accuracy or range.

We can find them everywhere as our world is full of different types of sensors and their simple or more complex applications: in our offices, gardens, shopping malls, homes, cars, toys etc. That is why it is critical to understand the way they operate and their many possibilities.



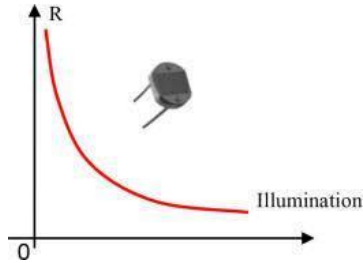


Co-funded by the  
Erasmus+ Programme  
of the European Union



## 1. Photoresistor

### 1.1 About photoresistor



A photoresistor is a light-controlled variable resistor. A high intensity of light incident on the surface will cause a lower resistance, whereas a lower light intensity will cause higher resistance. The most common usage is as a light and dark activate switch. It is also called LDR (Light Dependent Resistor).

Let's see how a photoresistor reacts with light.

### 1.2 Circuit 1

**Circuit title:** Luminance detection

**Circuit description:** The circuit detects the brightness of the room, if it falls below the set limit, it turns on the led

**What you will need:**

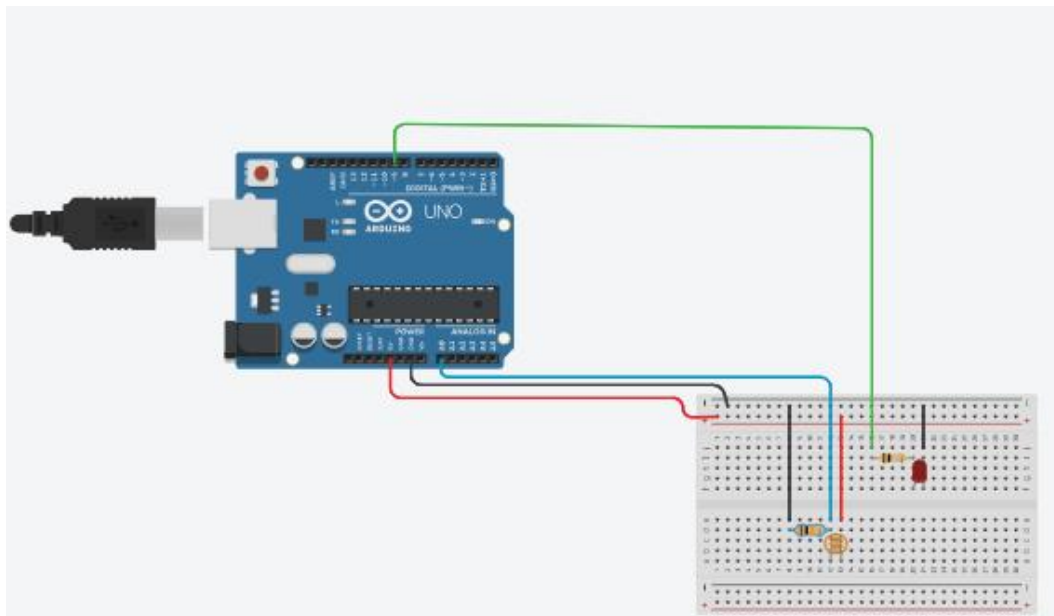
- a) Breadboard
- b) Photoresistor
- c) Resistor  $220\Omega^*$ ,  $10K\Omega^{**}$
- d) Led 3mm
- e) Arduino

\* The resistor of  $220\Omega$  is the standard protection resistance of LED at 5 volts.

\*\* The  $10K\Omega$  resistor is used to protect the Arduino from overcurrent when the photoresistor's value decreases significantly.

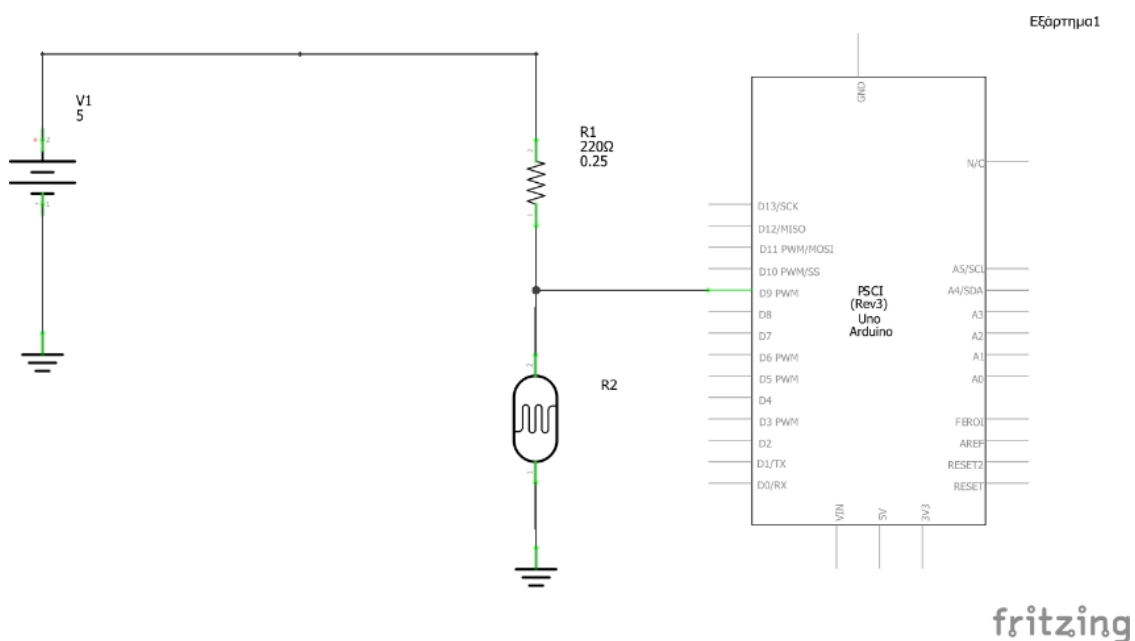


Co-funded by the  
Erasmus+ Programme  
of the European Union



The LED connected to pin 9 is set as an output. When pin 9 is HIGH, the LED will emit light; otherwise, it will remain off.

With the photoresistor and the 10K $\Omega$  resistor, we build a voltage divider. The value that will be sent to the processor depends on the intensity of the light on the sensor.





### 1.3 The Code

```
1  const int photoantistasi = A0; // Photoresistor at Arduino analog pin A0
2  const int ledPin=9;    // Led pin at Arduino pin 9
3
4  //Variables
5  int timi ;            // Store value from photoresistor
6
7  void setup(){
8    pinMode(ledPin, OUTPUT); // Set ledPin - 9 pin as an output
9    pinMode(photoantistasi , INPUT); // Set photoantistasi - A0 pin as an input (optional)
10   Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
11
12   }
13 void loop(){
14   value = analogRead(photoantistasi );
15
16   //You can change the value "70."
17   if (timi > 70){
18     digitalWrite(ledPin, LOW); //Turn led off
19   }
20   else{
21     digitalWrite(ledPin, HIGH); //Turn led on
22   }
23
24   delay(500); //Small delay
25 }
```

The program reads the analog input. If the value exceeds '70', the LED will turn on; otherwise, it will turn off. Each program loop includes a delay of 500 milliseconds (using the command 'delay(500)') to give the sensor time to take measurements.

#### tips



You can modify your code and, through the serial port, monitor the sensor's values on your computer.



```
26 const int photoantistasi = A0; // Photoresistor at Arduino analog pin A0
27 const int ledPin=9;    // Led pin at Arduino pin 9
28
29 //Variables
30 int timi ;           // Store value from photoresistor
31
32 void setup(){
33   pinMode(ledPin, OUTPUT); // Set ledPin - 9 pin as an output
34   pinMode(photoantistasi , INPUT); // Set pResistor - A0 pin as an input (optional)
35   Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
36
37 }
38 void loop(){
39   value = analogRead(photoantistasi );
40
41   //You can change the value "25."
42   if (timi> 70){
43     digitalWrite(ledPin, LOW); //Turn led off
44   }
45   else{
46     digitalWrite(ledPin, HIGH); //Turn led on
47   }
48
49   delay(500); //Small delay
50   Serial.println(timi); // print value(after each value of the variable change line on the screen).
```



## 2. Motion sensor

The motion sensor works by detecting changes in infrared radiation or in the presence of heat and movement within its coverage area. The most common type of motion sensor is a passive infrared (PIR) sensor, which detects changes in the infrared radiation emitted by objects in its field of view.

### 2.1 About PIR sensor

The infrared motion detection sensor (Pyroelectric InfraRed Sensors) is a sensor that allows us to detect the presence of a moving living organism in a specific area.

All living things emit infrared radiation, which suitable sensors can easily detect.

The PIR sensor consists of two areas that detect infrared radiation.

When a living organism enters the first area, a positive potential difference is created in the electronic circuit of the sensor.

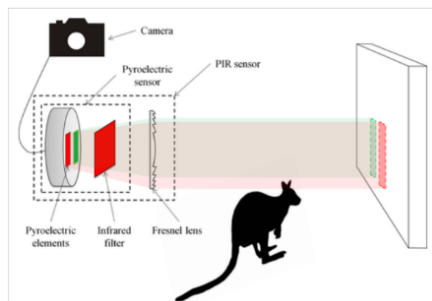
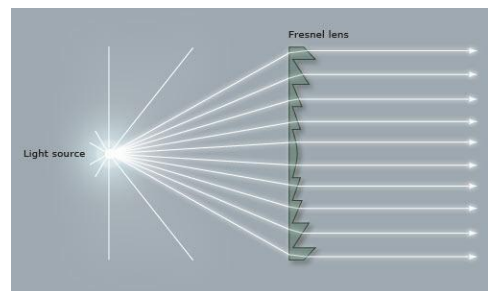
When it leaves the second area, a negative potential difference is created.

This voltage triggers the electronic circuit of the sensor and sends a logical '1' to the processor (existence of movement in the space). The problem with this function is that it only detects a small area, the thin zone between the two detection areas.

To increase the area range, the sensor is covered by a Fresnel lens.

Augustin Fresnel discovered the Fresnel lens and found application in nautical lighthouses.

Its property is that it concentrates the light of a source in a specific direction to a particular level.



Here, we use it the other way around. As it can be seen in the picture.

The infrared radiation present in a space converges on the sensor and thus increases the area it can detect



Co-funded by the  
Erasmus+ Programme  
of the European Union



## 2.2 Circuit 2:

**Circuit title:** Motion detector

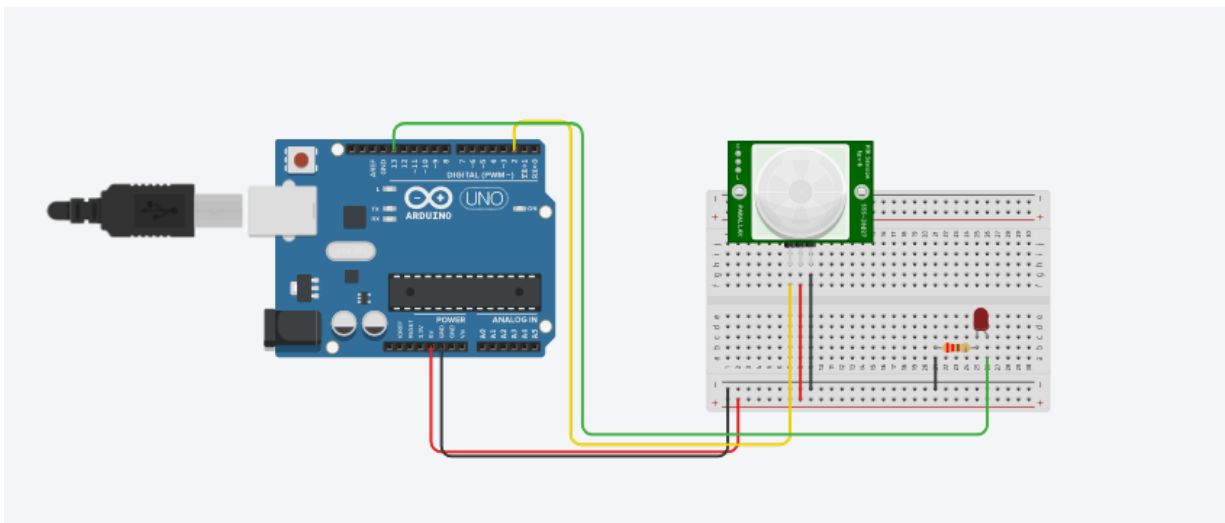
**Circuit description:** The circuit monitors the space. If motion is detected, it activates the led

**What you will need:**

- a) Breadboard
- b) PIR sensor
- c) Resistor 220 $\Omega$ \*, 10K $\Omega$  \*\*
- d) Led 3mm
- e) Arduino

\* The resistor of 220 $\Omega$  is the standard protection resistance of LED at 5 volts.

\*\* The 10K $\Omega$  resistor is used to protect the Arduino from overcurrent when the photoresistor's value decreases significantly.



Supply the sensor with a voltage of 5 volts. The pin signal is connected to pin 2 of the Arduino. The LED is connected to pin 13 of the Arduino.

When the sensor detects motion, it sends bit '1' and the program lights the led

*The PIR sensor has three pins: the power pin, which is connected to a 5-volt power supply; the ground pin, which is connected to the GR pin; and the signal pin. The signal pin outputs a value of 0 when there is no movement detected, and a value of 1 when the sensor detects movement.*



## 2.3 The Code

```
int led = 13;           // the pin that the LED is attached to
int sensor = 2;         // the pin that the sensor is attached to
int state = LOW;        // by default, no motion detected
int val = 0;            // variable to store the sensor status (value)

void setup() {
  pinMode(led, OUTPUT); // initialize LED as an output
  pinMode(sensor, INPUT); // initialize sensor as an input
  pinMode(buzzer, OUTPUT);
  tone(buzzer, 1000, 2000);
  Serial.begin(9600);    // initialize serial
}

void loop(){
  val = digitalRead(sensor); // read sensor value
  if (val == HIGH) {         // check if the sensor is HIGH
    digitalWrite(led, HIGH); // turn LED ON
    delay(100);              // delay 100 milliseconds
    if (state == LOW) {
      Serial.println("Motion detected!");
      state = HIGH;          // update variable state to HIGH
    }
  }
  else {
    digitalWrite(led, LOW);  // turn LED OFF
    noTone(buzzer);
    delay(200);              // delay 200 milliseconds

    if (state == HIGH){
      Serial.println("Motion stopped!");
      state = LOW;           // update variable state to LOW
    }
  }
}
```

The program reads the input 2 of Arduino. If the value is '1', the LED will turn on, and through the serial port, the "Motion detected !" is displayed; otherwise, the LED will turn off, and the "Motion stopped " will be displayed.

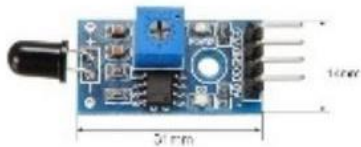


### 3. Flame sensor

An important topic in building automation is fire detection. Two types of sensors are used: smoke detection and flame sensors. Usually, when the smoke sensor is activated, we have an alarm, while if the flame sensor is activated, the extinguishing system starts.

There are multiple ways to detect a fire, like detecting temperature change, smoke detection, etc. In all of these, detecting temperature change would be more accurate since some fires won't even have detectable smoke.

#### 3.1 About flame sensor

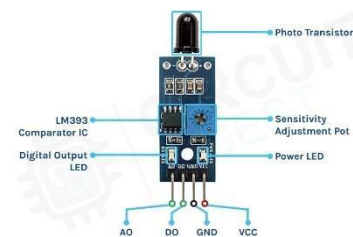


The IR photodiode detects IR radiation from any warm body. It then compares this value to a specified value.

We can display the irradiance value on the sensor's analogue output, or once the irradiance reaches a threshold value, the sensor will change its digital output accordingly.

The flame sensor module has only very few components, which include an IR photodiode, an LM393 comparator IC, and some passive components.

The power LED will light up when the module is powered and the DO LED will turn off, when a flame is detected. The sensitivity can be adjusted with the trimmer resistor onboard.



#### 3.2 Circuit 3

**Circuit title:** Flame detector

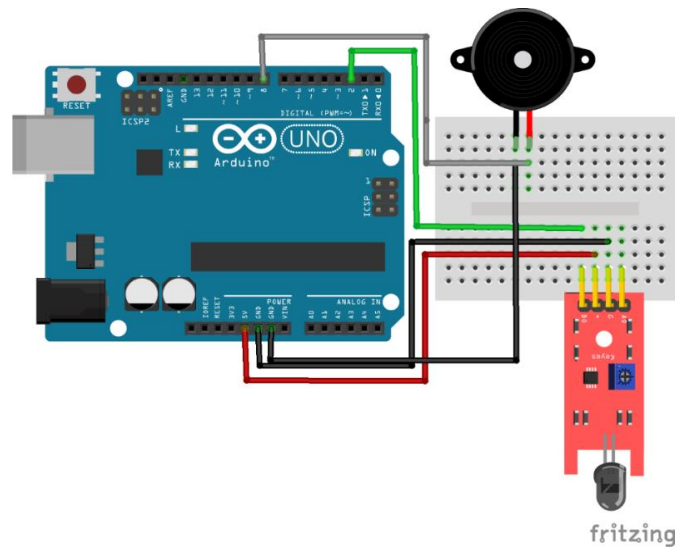
**Circuit description:** The circuit monitors the area. If the flame is detected, activate the buzzer and the alarm sounds.

**What you will need:**

- a) Breadboard
- b) Flame sensor
- c) Passive buzzer
- d) Arduino



Co-funded by the  
Erasmus+ Programme  
of the European Union



Supply the sensor with a voltage of 5 volts. The digital pin signal is connected to pin 2 of the Arduino. The buzzer is connected to pin 13 of the Arduino.

When a flame is detected, the sensor sends a logical "1" to the Arduino input, and you activate, via output 8, the buzzer with a sound frequency of 1KHz

### 3.3 The Code

```
int flameSens=2; //flameSens to arduino pin 2
int buzzerPin = 8; //buzzer to arduino pin 8
void setup(){
  pinMode(flameSens, INPUT); //initialize Flame sensor output pin connected pin as input
  pinMode(buzzerPin, OUTPUT); // initialize digital pin buzzerPin as an output
  Serial.begin(9600); // initialize serial communication @ 9600 baud
}
void loop(){
  if (digitalRead(flameSens) == 1 )
  {
    tone(buzzerPin, 1000); //Send 1KHz sound signal
    Serial.println("** Warning!!!! Fire detected!!! **");
  }
  else
  {
    digitalWrite(LED_BUILTIN, LOW); // Led OFF
    noTone(buzzerPin); //stop sound
    Serial.println("No Fire detected");
  }
  delay(100);
}
```

#### tips



Using the tone() function, you can create different sounds. Try generating sound frequencies of 2400 Hz and 2900 Hz with a 1000-second delay between them.



## 4. Soil moisture sensor

Soil moisture sensors are popular in agriculture, landscaping, and gardening. Moisture sensing technologies provide benefits to growers by saving them money and time spent on manual watering systems.

They can help farmers and gardeners check moisture levels quickly, precisely and affordably. With this, you can easily determine when to water or irrigate your field.

Soil analysis using a sensor reduces uncertainty by providing real-time data, allowing you to make more informed decisions about how to manage resources such as water.

### 4.1 About soil moisture sensor



The soil moisture sensor operates straightforwardly. The sensor includes a fork-shaped probe with two exposed conductors inserted into the soil or wherever the moisture content is to be measured.

These act as variable resistors (similar to potentiometers) whose resistance varies with the soil's moisture content. This resistance varies inversely with soil moisture.

In addition, the sensor includes an electronic module that connects the probe to the Arduino.

The module generates an output voltage based on the probe's resistance, available at an Analog Output (AO) pin.

The same signal is fed to an LM393 High Precision Comparator, which digitises it and makes it available at a Digital Output (DO) pin.

The module includes a potentiometer for adjusting the sensitivity of the digital output (DO).

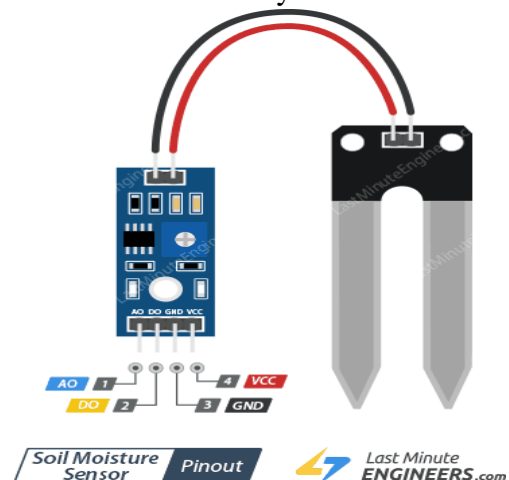
The soil moisture sensor only requires four pins to connect.

AO (Analog Output) generates an analogue output voltage proportional to the soil moisture level.

**DO (Digital Output)** DO becomes LOW when the moisture level exceeds the threshold value (as set by the potentiometer), and HIGH otherwise.

**VCC** supplies power to the sensor. It is recommended that the sensor be powered from 3.3V to 5V. Please keep in mind that the analogue output will vary depending on the voltage supplied to the sensor.

**GND** is the ground pin





Co-funded by the  
Erasmus+ Programme  
of the European Union



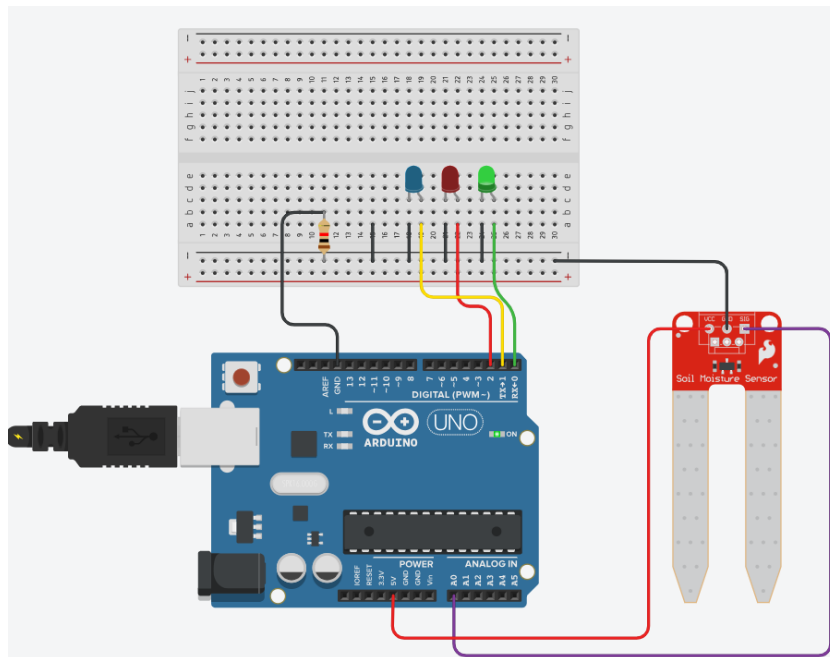
## 4.2 Circuit 4

**Circuit title:** Soil moisture detector

**Circuit description:** The circuit measures soil moisture as a percentage of a given value. If the moisture is between 0-30%, the red light turns on; from 30%-60%, the blue light turns on, and above 60%, the green light turns on.

**What you will need:**

- a) Breadboard
- b) Soil moisture sensor
- c) Green, blue, red led
- d) a  $220\Omega$  resistor is necessary to protect the LED
- e) Arduino



**Note:** At the start of the program, the sensor must detect the maximum moisture level so it can accurately calculate percentages throughout the program cycle.

## 4.3 The Code

```
// Moisture Sensor Arduino Code
int greenLight = 9; //Defining pins
int blueLight = 8;
int redLight = 10;
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
float maximumMoistureLevel; //The max moisture level and current moisture
                                levels will be needed for percentage calculations
float currentMoistureLevel; // = Moisture level

void setup() {
    pinMode(8, OUTPUT); //initiate pin 8 as output
    pinMode(9, OUTPUT); //initiate pin 9 as output
    pinMode(10, OUTPUT); //initiate pin 10 as output
    pinMode(A1, INPUT); //A1 is the pin used for the Soil Moisture Sensor
    delay(100);
    maximumMoistureLevel = analogRead(A1)
    digitalWrite(greenLight, HIGH); //all the leds turn on for 2sec to show that the program has been
                                    initiated.

    digitalWrite(blueLight, HIGH);
    digitalWrite(redLight, HIGH); ;
    delay(100);
    digitalWrite(greenLight, LOW);
    digitalWrite(blueLight, LOW);
    digitalWrite(redLight, LOW);
    Serial.begin(9600);
}

void loop() {
    if (maximumMoistureLevel/currentMoistureLevel <= 0.3) //if the moisture level below 30%
    {
        digitalWrite(greenLight, LOW);
        digitalWrite(blueLight, LOW);
        digitalWrite(redLight, HIGH); //Switch red light on, but don't sound the buzzer
    }
    else if (maximumMoistureLevel/currentMoistureLevel <= 0.8 &&
             maximumMoistureLevel/currentMoistureLevel > 0.3) //if the moisture level is between 30 and
             60%
    {
        digitalWrite(greenLight, LOW);
        digitalWrite(blueLight, HIGH); //Just switch yellow light on
        digitalWrite(redLight, LOW);
    }
    else //Otherwise the moisture level is above 60%, and therefore it's good enough
    {
        digitalWrite(greenLight, HIGH); //Switch green light on
        digitalWrite(blueLight, LOW);
        digitalWrite(redLight, LOW);
    }
    currentMoistureLevel = analogRead(A1); //renewal of measurements

    delay(500); //Short delay to not overload the program
    Serial.println("maximumMoistureLevel"); //Just so you can see the max moisture level as a reading
                                            between 0-1023

    Serial.println(maximumMoistureLevel);
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
Serial.println("currentMoistureLevel");//Just so you can see the moisture level as a reading  
between 0-1023
```

```
Serial.println(currentMoistureLevel);  
}
```

### tips



The humidity sensor has a disadvantage: because it is constantly powered in a humid environment, it has a short lifespan. Therefore, you should only operate the sensor when you need to take a measurement.



## 5. Active fire protection (AFP)

### 5.1 About Active fire protection (AFP)

Active Fire Protection (AFP) refers to a set of systems that require some form of action to function effectively in the event of a fire. These actions can be manually operated, such as using a fire extinguisher, or automated, like a sprinkler system. Irrelevant of the method, some action is necessary for these systems to work.

A typical active fire protection system detects the fire, sounds the alarm, and then activates the automatic extinguishing system.

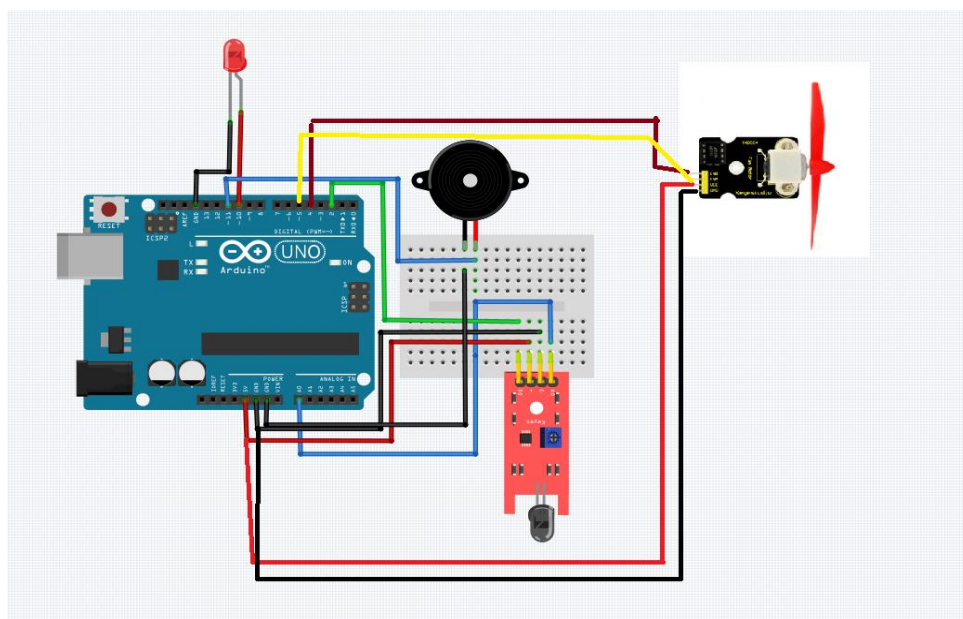
### 5.2 Circuit 5

**Circuit title:** Active fire protection (AFP)

**Circuit description:** The circuit detects fire using a flame sensor. When a flame is detected, both an audible alarm and a visual alert are activated, and the alarm will sound (buzzer + LED) four times. If the flame remains, the fan is turned on for 4 seconds to extinguish it. Should the flame continue to be present, the process repeats itself.

**What you will need:**

- a) Breadboard
- b) flame sensor
- c) Red led
- d) a 220Ω resistor is necessary to protect the LED
- e) passive buzzer module
- f) module fan
- g) Arduino
- h) tea light





Co-funded by the  
Erasmus+ Programme  
of the European Union



### Pinout circuit

Pin 11 □ buzzer

Pin 10 □ Red led

Pin4 □ INA

Pin 5 □ INB

**Note:** The new component in the circuit is the fan module, which has four pins: Vcc, Gnd, INA, and INB. A voltage of 5 volts is connected to Vcc, while the ground is connected to Gnd. The INA and INB pins are used for motor control. When INA is low and INB is low, the motor stops. When INA is low and INB is high, the motor turns clockwise.



Co-funded by the  
Erasmus+ Programme  
of the European Union



### 5.3 The Code

```
int flameSens=A0; //flamesensor to arduino pin 2
int buzzerPin = 11; //buzzer to arduino pin 8
int ledPin=10; //LED to arduino pin 8
int INA=4; // motor shelld INA pin to arduino pin4
int INB=5; // motor shelld INB pin to arduino pin5
int flame = 0; // variable to store the sensor status (value)

void setup(){
  pinMode(flameSens, INPUT); //initialize Flame sensor output pin as an input to arduino
  pinMode(buzzerPin, OUTPUT); // initialize buzzer pin as an output
  pinMode(ledPin, OUTPUT); // initialize led pin as an output
  Serial.begin(9600); // initialize serial communication @ 9600 baud
  pinMode(INA, OUTPUT); // initialize led pin as an output
  pinMode(INB, OUTPUT); // initialize led pin as an output
}

void loop() {
  flame=analogRead(flameSens);
  Serial.println("flame arxh");
  Serial.println(flame); //sensor measurement display

  if(flame <100)// read sensor value
  {
    for(int i=1; i<=4; i++) // alarm sounds 4 times
    {
      digitalWrite(ledPin,HIGH); // turn LED on
      tone(buzzerPin,2400); //buzzer sounds at 2400 KHz
      delay (1000);
      digitalWrite(ledPin,LOW); // turn LED off
      tone(buzzerPin,2900); //buzzer sounds at 2900 KHz
      delay (1000);
      Serial.println("flame loop");
      Serial.println(flame);
    }
    digitalWrite(INA,LOW); // turn fan on clockwise
    digitalWrite(INB,HIGH); // turn fan on clockwise
    Serial.println("flame moter");
    Serial.println(flame);
    delay(4000);
  }
  digitalWrite(INA,LOW); // turn fan on clockwise
  digitalWrite(INB,LOW); // turn fan on clockwise
  noTone(buzzerPin);
  Serial.println("flametelos"); //sensor measurement display
  Serial.println(flame);
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



## Operation

When a flame is detected, the alarm will sound (buzzer + LED) four times, and then the fan will turn on for 4 seconds.

## 6. Automatic plant watering system

Using an automatic plant watering system is an efficient way to conserve water while taking care of your garden. This system delivers the exact amount of water needed for plants, in contrast to hand watering or using a sprinkler system, which can easily result in overwatering or underwatering.

### 6.1 Circuit title: Automatic plant watering system

**Circuit description:** This circuit measures the soil moisture level. If the moisture level falls below a predetermined threshold, the valve opens, allowing water to flow to the plant, and the green LED turn on. When the moisture level exceeds the specified threshold, the valve closes to stop the watering, and the blue LED illuminates.

#### What you will need:

- a) Breadboard
- b) moisture sensor (with module )
- c) Led green, led blue
- d) 220 $\Omega$  resistor is necessary to protect the LED.
- e) solenoid valve
- f) small water tank
- g)Arduino
- h) small pot with plant

#### Pinout circuit

Pin 8  $\square$  led, green

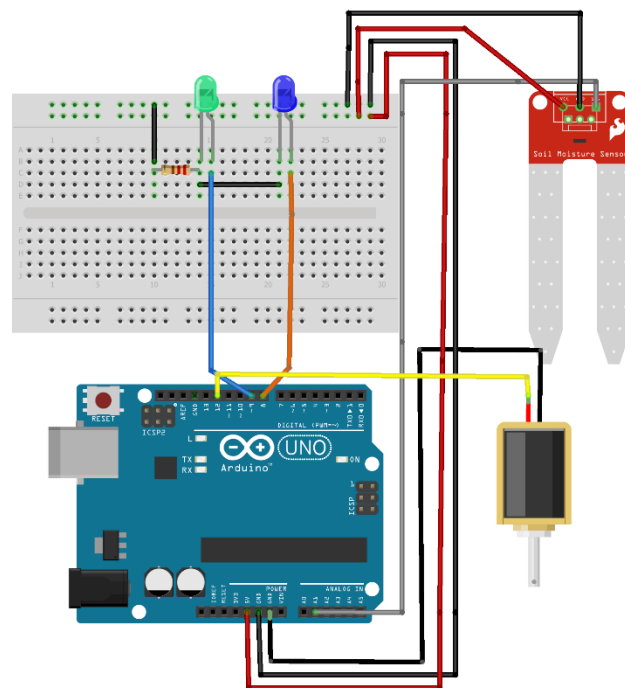
Pin 9  $\square$ led blue

A1  $\square$ analogue input moisture sensor

Pin 12  $\square$  valve



Co-funded by the  
Erasmus+ Programme  
of the European Union



fritzing

### 6.3 The Code

```
// Moisture Sensor Arduino Code
```

```
int greenLight = 9; //Defining pins
int blueLight = 8; //Defining pins
int moistureLevel=A1; //Defining pins
int valvePin=10; //Defining pins
int maxMoistureLevel; //The max moisture level
int minMoistureLevel; //The max moisture level
int currentMoistureLevel; // current moisture level

void setup() {
  pinMode(blueLight,OUTPUT); //initiate pin as output
  pinMode(greenLight,OUTPUT); //initiate pin 9 as output
  pinMode(moistureLevel, INPUT); //A1 is the pin used for the Soil Moisture
  Sensor
  pinMode(valvePin, OUTPUT); //initiate pin 9 as output
  Serial.begin(9600); // initialize serial communication @ 9600 baud:
}
void loop() {
  maxMoistureLevel=700; //set the variable to 700
  minMoistureLevel=400; //set the variable to 400
  currentMoistureLevel=analogRead(moistureLevel); //Assign the variable the
  current value
  if (currentMoistureLevel>1000) //if the moisture level below 1000 green and
  blue lights turn on
  {
    digitalWrite(greenLight,HIGH); //Switch light on
    digitalWrite(blueLight,HIGH); //Switch light on
    digitalWrite(valvePin,LOW); //Switch light off
  }
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
else
if((currentMoistureLevel>minMoistureLevel)&&(currentMoistureLevel<1000))//if the
moisture level is in between 30 and 60%
{
    digitalWrite(greenLight, HIGH);
    digitalWrite(blueLight, LOW);// led light on
    delay(3000);
    digitalWrite(valvePin, HIGH);// open the valve
    delay(500);
    digitalWrite(valvePin, LOW);// close the valve
    delay(300);
    digitalWrite(greenLight, LOW);// led light off
}
if (currentMoistureLevel>maxMoistureLevel){//if the moisture level below 30%
    digitalWrite(blueLight,HIGH);
}

delay(500); //Short delay
Serial.println("MoistureLevel");
Serial.println(currentMoistureLevel);
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



## 7. Security light

Security lights are electric lights, usually outside a building, that switch on when someone or something moves near them. Burglars try to break into the house in the early hours, but neighbours can spot them when security lights come on. This system should be more efficient by adding a brightness sensor; the light will illuminate only when the surrounding light is low, conserving energy.

### 7.1 Circuit title: Security light

**Circuit description:** This circuit features two sensors: a motion sensor (PIR) and a brightness sensor. When the monitoring area's brightness level is low and motion is detected, the light will turn on for three seconds.

#### What you will need:

- a) Breadboard
- b) photoresistor
- c) module Led
- d) 10K resistor
- e) sensor PIR
- g) Arduino

#### Pinout circuit

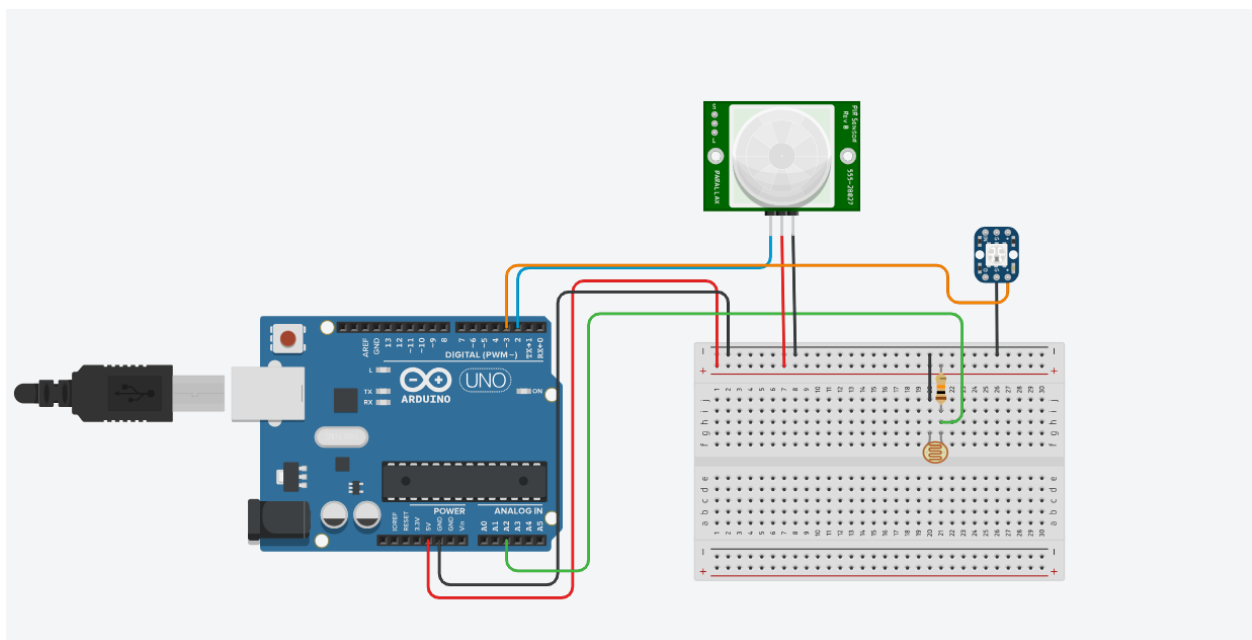
Pin 2  $\square$  PIR input

Pin A2  $\square$  input brightness sensor

Pin 3  $\square$  Led



Co-funded by the  
Erasmus+ Programme  
of the European Union



### 7.3 The Code

```
//Constants
const int pResistor = A2;    // Photoresistor at Arduino analog pin A2
const int ledPin=3;          // Led pin at Arduino pin 9

//Variables
int value;                   // Store value from photoresistor (0-1023)

void setup(){
  pinMode(ledPin, OUTPUT);   // Set ledPin - 9 pin as an output
  pinMode(pResistor, INPUT); // Set pResistor - A0 pin as an input (optional)
  Serial.begin(9600);        //rate at which arduino communicates with laptop
}

void loop(){
  value = analogRead(pResistor); //reads the analog data from the sensor

  if (value < 700){
    digitalWrite(ledPin, LOW); //Turn led off
  }
  else{
    digitalWrite(ledPin, HIGH); //Turn led on
  }

  delay(500);                //Small delay
  Serial.println(value);      //prints on the serial monitor
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



## 8. Application

All sensor projects can be applied to the corresponding construction. The pinout for this construction is identical to that of the Individual project.

### Pinout circuit

Pin A0 □ flame sensor

Pin 2 □ PIR output

Pin 3 □ Led module

Pin4 □ INA motor

Pin 5 □ INB motor

Pin 8 □ led, green

Pin 9 □ led blue

Pin 10 □ Red led

Pin 11 □ buzzer

Pin 12 □ valve

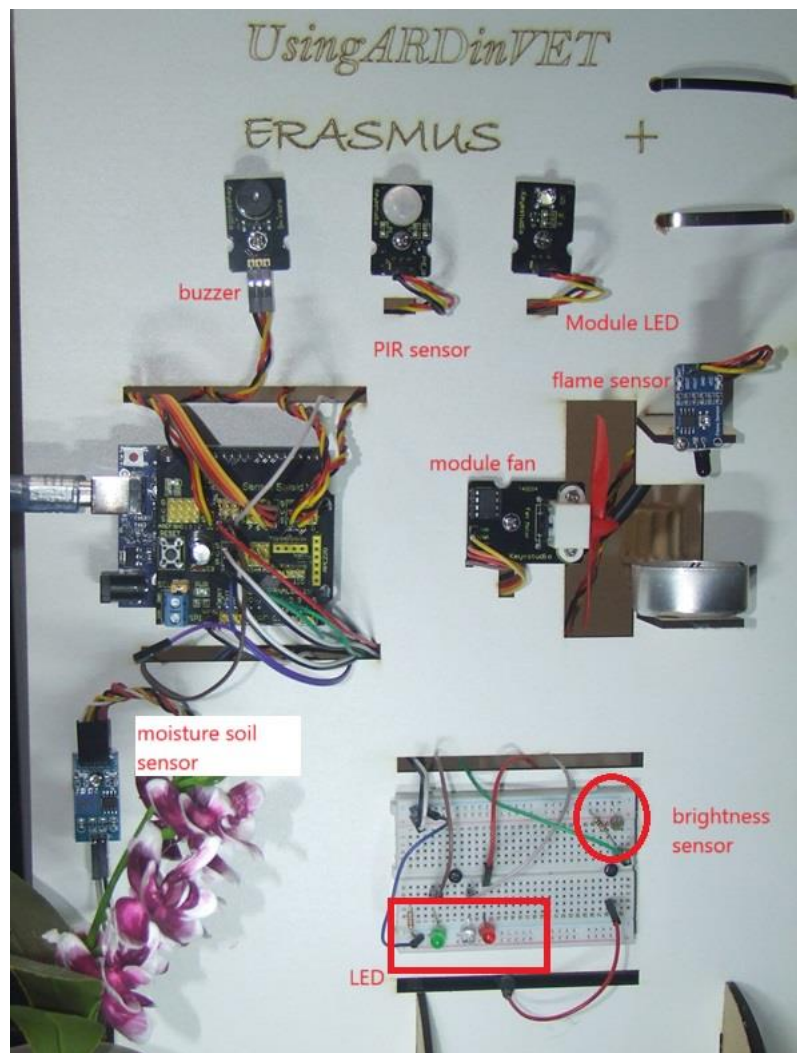
Pin A0 □analogue input flame sensor

Pin A1 □analogue input moisture sensor

Pin A2 □analogue input brightness sensor



Co-funded by the  
Erasmus+ Programme  
of the European Union



With the following program, you can run projects: **Automatic plant watering system, active fire protection, and security light at the same time**, without needing to load their programs individually.

```
// complete program
int sensorPir = 2;
int ledPin=3;
int INA=4;
int INB=5;
int blueLight = 8;
int greenLight = 9;
int redLedPin=10;
int buzzerPin = 11;
int valvePin=10;
int flameSens=A0;
int moistureLevel=A1;
int sensorPhotores = A2;
int maxMoistureLevel;
int minMoistureLevel;
level
int currentMoistureLevel;
level

//Sensor PIR at Arduino pin 2
// Led pin at Arduino pin 9
// motor shelld INA pin to arduino pin4
// motor shelld INB pin to arduino pin5
//Defining pins
//Defining pins
//Defining pins
//Defining pins
//flamesensor to arduino pin A0
//moisture sensor to arduino pin A1
// Photoresistor at Arduino analog pin A2
//variable for the max moisture level
//variable for the min moisture

// variable for the current moisture
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
int valuePhoto; //variable to store value from
photoresistor (0-1023)
int valuePir; // variable to store value from sensor
PIR (0-1023)
int flame = 0; // variable to store the sensor status
(value)

void setup() {
    pinMode(ledPin, OUTPUT); // Set ledPin as an output
    pinMode(sensorPhotores, INPUT); // Set sensorPhotores pin as an input
    pinMode(sensorPir, INPUT); // Set sensorPhotores pin as an input
    pinMode(flameSens, INPUT); //initialize Flame sensor output pin as an
input to arduino.
    pinMode(buzzerPin, OUTPUT); // initialize buzzer pin as an output
    pinMode(redLedPin, OUTPUT); // initialize redled pin as an output
    pinMode(INA, OUTPUT); // initialize INA fan pin as an output
    pinMode(INB, OUTPUT); // initialize INB fan pin as an output
    pinMode(blueLight, OUTPUT); //initiate blue light pin as output
    pinMode(greenLight, OUTPUT); //initiate green light pin as output
    pinMode(moistureLevel, INPUT); //A1 is the pin used for the Soil Moisture
Sensor
    pinMode(valvePin, OUTPUT); //initiate pin 9 as output
    maxMoistureLevel=700; //set the variable to 700
    minMoistureLevel=400; //set the variable to 400
    Serial.begin(9600); // initialize serial communication @ 9600
    baud:
}
void securityLight(){ //method for security light
    digitalWrite(ledPin, HIGH); // turn LED on
    delay(5000);
    digitalWrite(ledPin, LOW); // turn LED off
}
void flameSens1(){ //method for fire
    for(int i=1; i<=4; i++) // alarm sound 4 times
    {
        digitalWrite(redLedPin, HIGH); // turn LED on
        tone(buzzerPin, 2400); //buzzer sounds at 2400 KHz
        delay(1000);
        digitalWrite(redLedPin, LOW); // turn LED off
        tone(buzzerPin, 2900); //buzzer sounds at 2900 KHz
        delay(1000);
        Serial.println("flame loop"); //display variable 'flame'
        Serial.println(flame);
    }
    digitalWrite(INA, LOW); // turn fan on clockwise
    digitalWrite(INB, HIGH); // turn fan on clockwise
    noTone(buzzerPin); // stop buzzer
    delay(4000);
    digitalWrite(INA, LOW); // turn fan off
    digitalWrite(INB, LOW); // turn fan off
}
void watering(){ //method for watering
    if (currentMoistureLevel>1000) //if the moisture sensor it is not
on the ground
    {
        digitalWrite(greenLight, HIGH); //Switch light on, shows the sensor
is not on the ground
        digitalWrite(blueLight, HIGH); //Switch light onon, shows the
sensor is not on the ground
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



```
        digitalWrite(redLedPin, LOW);          //Switch light off on, shows the
sensor is not on the ground
    }
    else
if((currentMoistureLevel>minMoistureLevel)&&(currentMoistureLevel<1000))//if the
moisture level is below the min level
    {
        digitalWrite(greenLight, HIGH);        // led light on
        digitalWrite(blueLight, LOW);          // led light off
        delay(3000);
        digitalWrite(redLedPin, HIGH);         // open the valve
        delay(500);
        digitalWrite(redLedPin, LOW);          // close the valve
        delay(300);
        digitalWrite(greenLight, LOW);         // led light off
    }
    if (currentMoistureLevel>maxMoistureLevel){ //if the
moisture level is above the min level
        digitalWrite(blueLight, HIGH);        // led light on
    }

    Serial.println("MoistureLevel");           // display the moisture
    Serial.println(currentMoistureLevel);
}

void loop() {
    valuePhoto = analogRead(sensorPhotores);   //Assign the variable the
current value
    valuePir=digitalRead(sensorPir);           //Assign the variable the
current value
    flame=analogRead(flameSens);               //Assign the variable the
current value
    currentMoistureLevel=analogRead(moistureLevel); //Assign the variable the
current value
    if (valuePhoto > 800&& valuePir==HIGH)      //if there is darkness and
movement
    {
        securityLight();
    }
    if(flame <450)                             //if there is fire
    {
        flameSens1();
    }

    if((currentMoistureLevel>minMoistureLevel)&&(currentMoistureLevel<1000))//if the
moisture level is below the min level and //if the moisture sensor is on the
ground
    {
        watering();
    }
    Serial.println("valuePhoto");              //display the variable 'valuePhoto'
    Serial.println(valuePhoto);                //display the variable 'valuePhoto'
    Serial.println("flame");                   //display the variable 'valuePhoto'
    Serial.println(flame);                     //display the variable 'valuePhoto'
    delay(500);
}
```



Co-funded by the  
Erasmus+ Programme  
of the European Union



# Annexex



Co-funded by the  
Erasmus+ Programme  
of the European Union



For more info,  
please visit our project website:

[www.ardinvet.com](http://www.ardinvet.com)



Co-funded by the  
Erasmus+ Programme  
of the European Union



**“This project is Funded by the  
Erasmus+ Program of the European Union.  
However, European Commission cannot be  
held responsible for any use which may be  
made of the information contained therein”**



Co-funded by the  
Erasmus+ Programme  
of the European Union



## **BIBLIOGRAPHY:**

1. Arduino: The ultimate Arduino guide for beginners, including Arduino programming, Arduino cookbook, tips, tricks, and more! Craig Newport;
2. Introduction to microcontrollers and programmable logic controllers - Author Viorel-Constantin Petre, Publisher: Matrixrom Publishing House;
3. Arduino: Getting Started With Arduino and Basic Programming With Projects (Advanced Methods to Learn Arduino Programming) - Ernest Leclerc;
4. Arduino Programming – Editura Nelly B.L. International Consulting LTD., March-2020.