**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\*\*\*\*\****

BEST PHOTO of The project

**“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”**

***COORDINATOR:***

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

***PARTNERS:***

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

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

**Instituto De Educaiıón Secundaria "María Molıner" (SEGOVIA, Spain)**

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

****

**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.

-Creating project team and tools.

-Project Website.

-Training Kits and Set.

-UsingARDinVET 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 | **??????** |
| 4 | Keypad Module and Training KIT | **??????** |
| 5 | Dot Matrix Display Module and Traning KIT | **??????** |
| 6 | Motor Module and Training KIT | **??????** |
| 7 | Sensor Module And Training KIT | **??????** |
|  | Annexex | **??????** |

**PROJECT MODULES and KITS of UsingARDinVET**

**Erasmus+ KA210-VET**

**Small-scale partnerships in vocational**

**education and training**

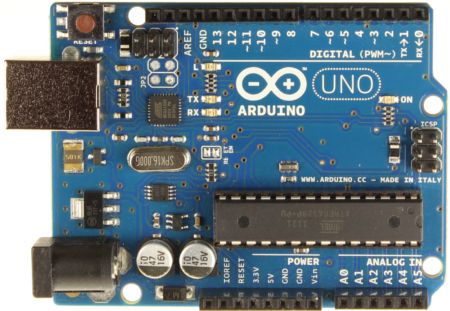
**Project Title: “Using Arduinos in Vocational Training”**

**Project Acronym: “UsingARDinVET”**

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

**INTRODUCTION to ARDUINOS**

**(BASICS of ARDUINOS)**



**INTRODUCING the ARDUINO**

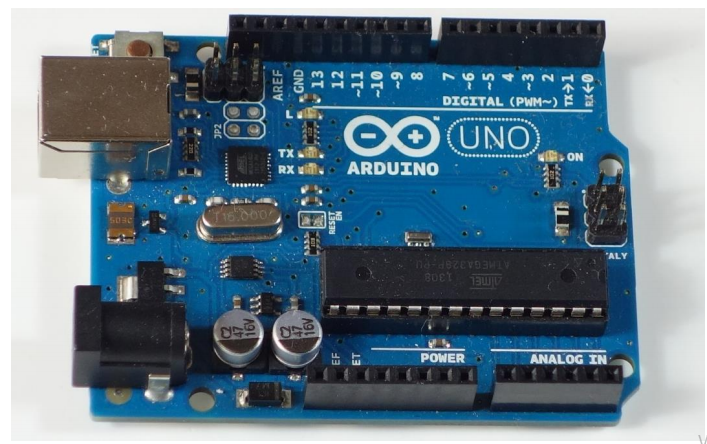
Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the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

On te 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.

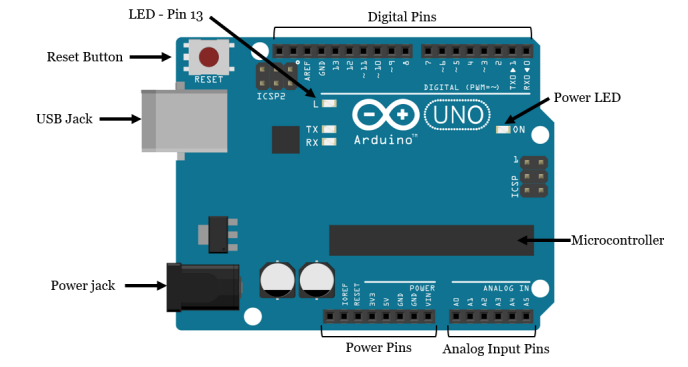
**What is an Arduino?**

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 are information being sent between the computer and the Arduino.

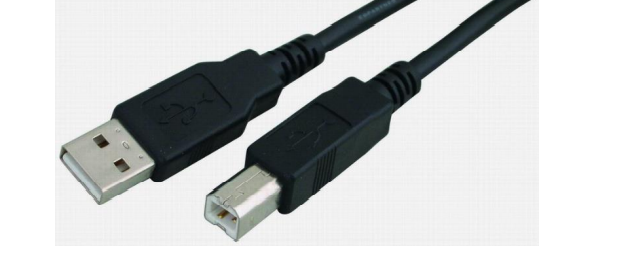
• **Analog pins:** the analog pins are labeled from A0 to A5 and are often used to read analog sensors. They can read different amounts of voltage between 0 and 5V. Additionally, they can also be used as digital output/input pins like the digital pins.

• **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.

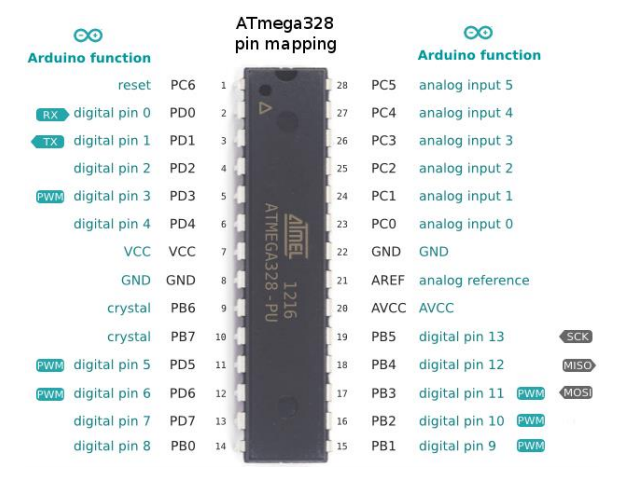
****

• **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 buton.. 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 MİNİ**

|  |  |
| --- | --- |
| 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**

|  |  |
| --- | --- |
| 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 Leonarda. 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.

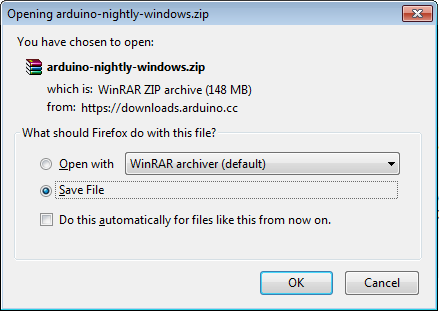
To install the Arduino IDE for Windows, we have to follow instructions.

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

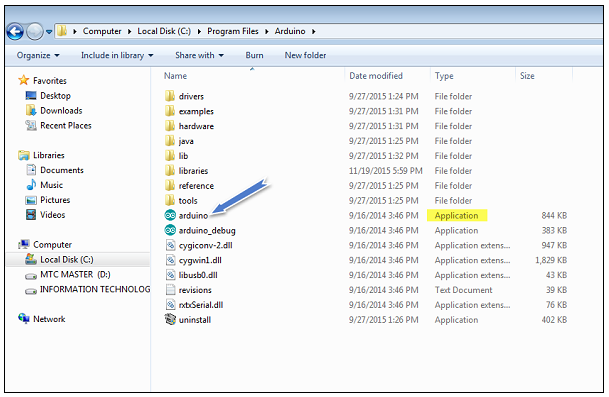
To download the Arduino IDE, please click on the following link: https://www.arduino.cc/en/Main/Software.



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



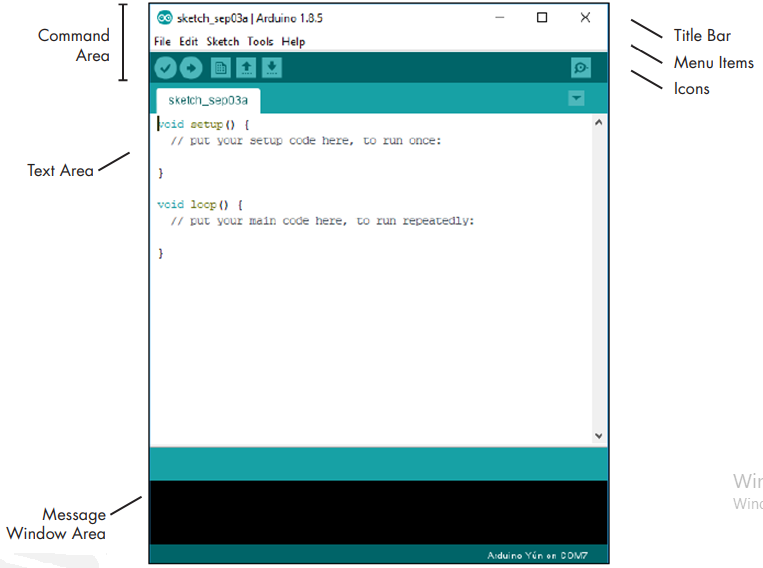
Inside the folder, we can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE. Then, simply follow the installation wizard to install the Arduino IDE.



**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:

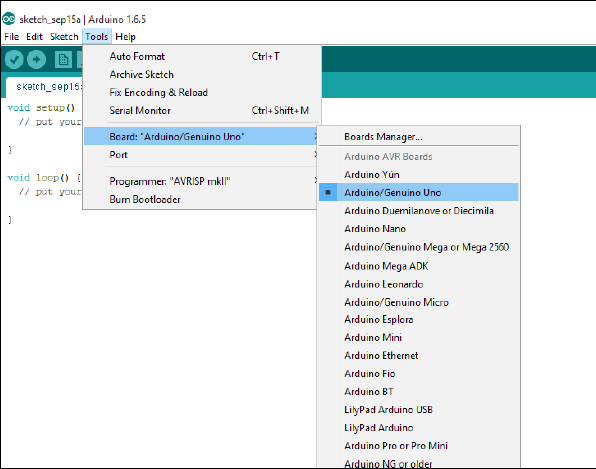
|  |  |
| --- | --- |
|  | **Verify (Compile):** Click this to check that the Arduino sketch is valid and doesn’t contain any programming mistakes. |
|  | **New:** Click this to open a new blank sketch in a new window. |
|  | **Open:** Open Click this to open a saved sketch. |
|  | **Save:** Click this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

**Connecting your Arduino**

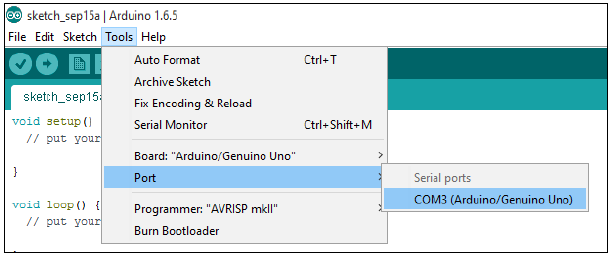
Connect your Arduino UNO to your computer via USB.

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**

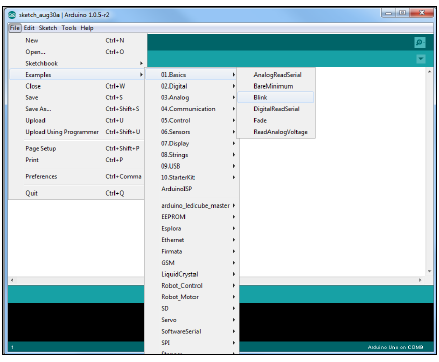
To show you how to upload code to your Arduino board, we’ll show you a simple

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.

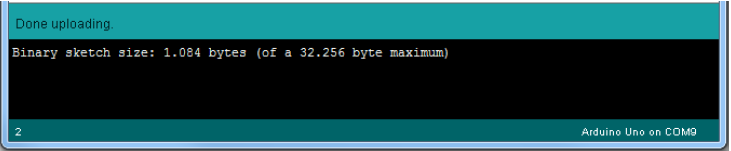
2. Go to **File Examples ** **01.Basics ** **Blink**



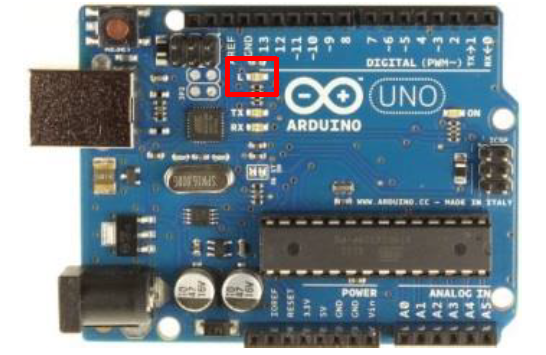
By default, the Arduino IDE comes pre-configured for the Arduino UNO. Click the **Upload** button and wait a few seconds.

****

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

****

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.

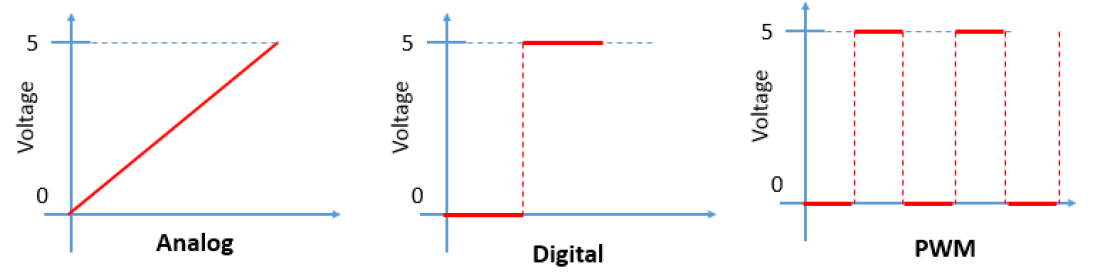
**Difference between digital, analog and PWM**

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.



**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.

**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**

|  |  |
| --- | --- |
| Image result for direnç resimleri | Image result for buton  resimleri |
| Image result for led  resimleri | Image result for jumper resimleri |

**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.

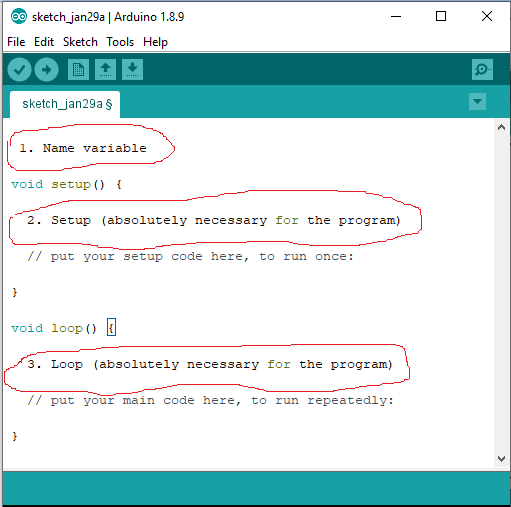
5. Wire it up. Connect your hardware, circuitry, and other items to the Arduino board.

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.



**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.

[**;**](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\SemiColon.html)**(Semicolon):**

[;](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\SemiColon.html) (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;

[**{}**](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\Braces.html)**(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) **}**

[**//**](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\Comments.html)**(Single line comment) and** [**/\* \*/**](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\Comments.html)**(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**](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\Define.html)**:**

#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](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\Const.html) 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**](file:///C:\Program%20Files%20(x86)\Arduino\reference\www.arduino.cc\en\Reference\Include.html)**:**

#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](https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/scope) that govern other variables. This, and the pitfalls of using [#define](https://www.arduino.cc/reference/en/language/structure/further-syntax/define), makes the const keyword a superior method for defining constants and is preferred over using [#define](https://www.arduino.cc/reference/en/language/structure/further-syntax/define).

### Example:

const float pi = 3.14;

### Notes:

[#define](https://www.arduino.cc/reference/en/language/structure/further-syntax/define)**or**const: You can use either const or [#define](https://www.arduino.cc/reference/en/language/structure/further-syntax/define) for creating numeric or string constants. For [arrays](https://www.arduino.cc/reference/en/language/variables/data-types/array), you will need to use const. In general const is preferred over [#define](https://www.arduino.cc/reference/en/language/structure/further-syntax/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 symble** | **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()](https://www.arduino.cc/en/Reference/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()](https://www.arduino.cc/en/Reference/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 dalay() 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](http://arduino.cc/en/Tutorial/PWM)) 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](https://www.arduino.cc/reference/en/language/structure/control-structure/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](https://www.arduino.cc/reference/en/language/variables/constants/constants), the statement block, and the increment is executed, then the condition is tested again. When the condition becomes [false](https://www.arduino.cc/reference/en/language/variables/constants/constants), the loop ends.  
increment: executed each time through the loop when condition is [true](https://www.arduino.cc/reference/en/language/variables/constants/constants).

### 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**](https://www.arduino.cc/reference/en/language/structure/control-structure/if) statements, [switch/ case](https://www.arduino.cc/reference/en/language/structure/control-structure/switchcase) 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**](https://www.arduino.cc/reference/en/language/structure/control-structure/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;

}

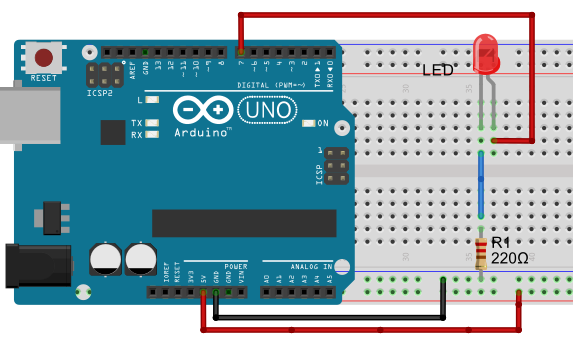
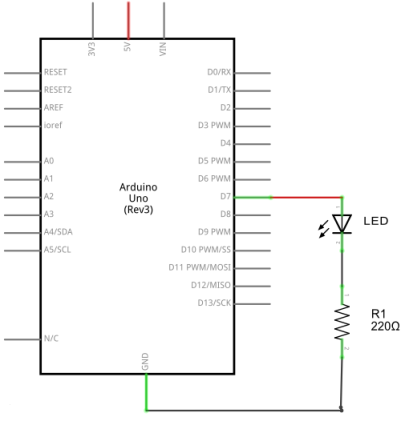
**var:** a variable whose value to compare with various cases. Allowed data types: int, char.  
**label1, label2:** constants. Allowed data types: int, char.

**Note:**

Arduino circuits are shown in two ways;

1-) Breadboard view

2-) Schematic view

  1-) Breadboard view 2-) Schematic view

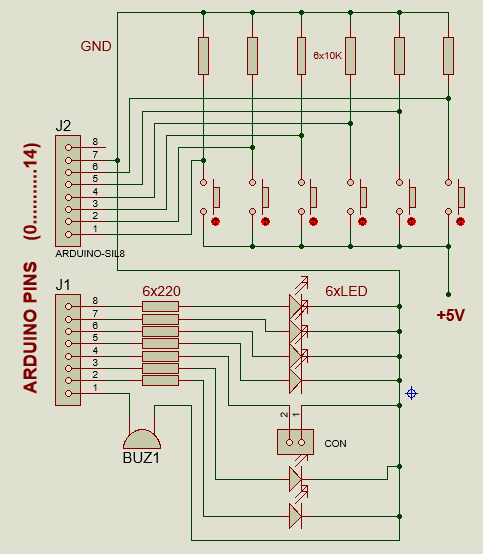
We will use Breadboard view which is used more.

Enough! Let’s make something!

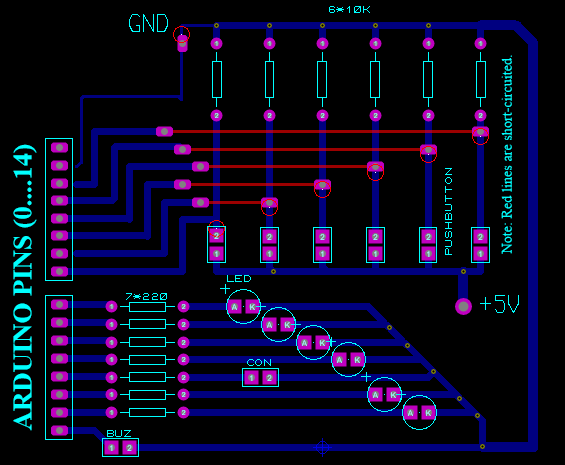
**Circuits of Arduino Button and LED Module Kit**

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 maket 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



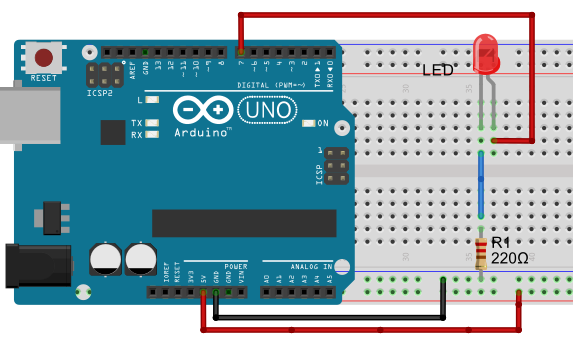
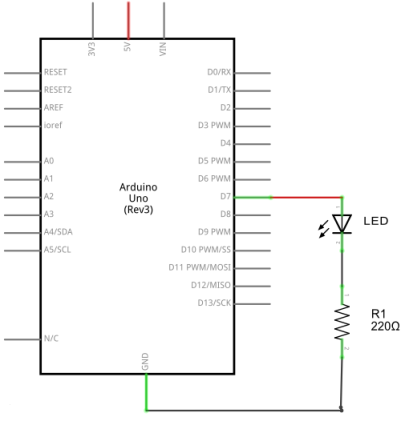
**Figure:** PCB Schema of Button and LED Module Kit

**Sample Arduio 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

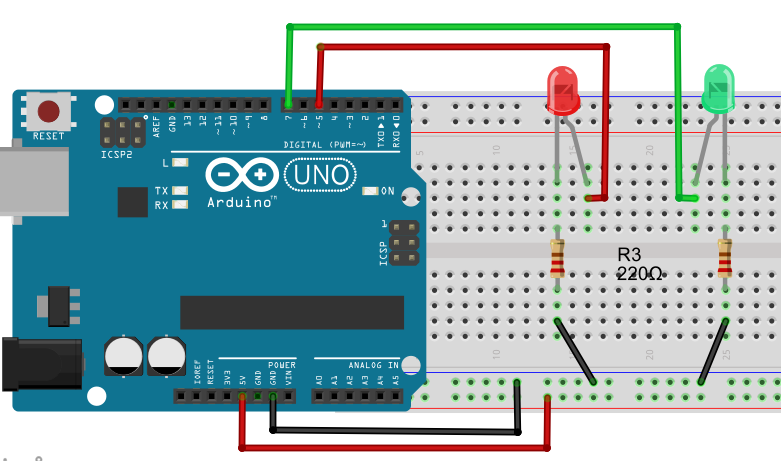
**delay(1000);**  // stopping the program for 1000 milliseconds

**}**

**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;** // Pin where the green LED is attached

**int redLED=7;** // Pin where the red LED is attached

**void setup() {**

**pinMode(greenLED, OUTPUT);** // green LED pin is initialised as OUTPUT

**pinMode(redLED, OUTPUT**); // red LED pin is initialised as OUTPUT

}

**void loop(){**

**digitalWrite(greenLED, HIGH);** // switch on green LED

**digitalWrite(redLED, LOW);** // switch off red LED

**pause(2000);**

**digitalWrite(greenLED, LOW);** // switch off green LED

**digitalWrite(redLED, HIGH);** // switch on red LED

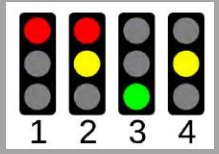
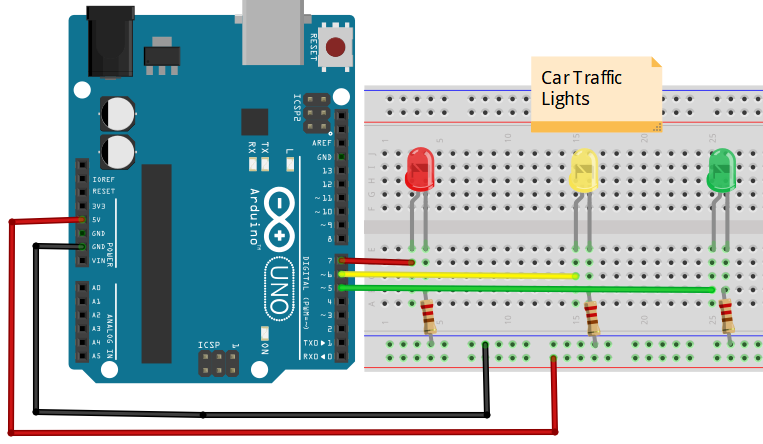
**pause(2000);**

**}**

**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() {** // here, we are initializing our pins as outputs

**pinMode(redLED, OUTPUT);**

**pinMode(yellowLED, OUTPUT);**

**pinMode(greenLED, OUTPUT);**

**}**

**void loop() {**

**digitalWrite(redLED, HIGH);** // redLED is ON for 9 seconds

**digitalWrite(yellowLED, LOW);**

**digitalWrite(greenLED, LOW);**

**delay(9000);**

**digitalWrite(redLED, HIGH);** // redLED and yellowLED are ON for 2seconds

**digitalWrite(yellowLED, HIGH);**

**digitalWrite(greenLED, LOW);**

**delay(2000);**

**digitalWrite(redLED, LOW);** // greenLED is ON for 9 seconds

**digitalWrite(yellowLED, LOW);**

**digitalWrite(greenLED, HIGH);**

**delay(9000);**

**digitalWrite(redLED, LOW);** // Again, yellowLED is ON for 2seconds

**digitalWrite(yellowLED, HIGH);**

**digitalWrite(greenLED, LOW);**

**delay(2000);**

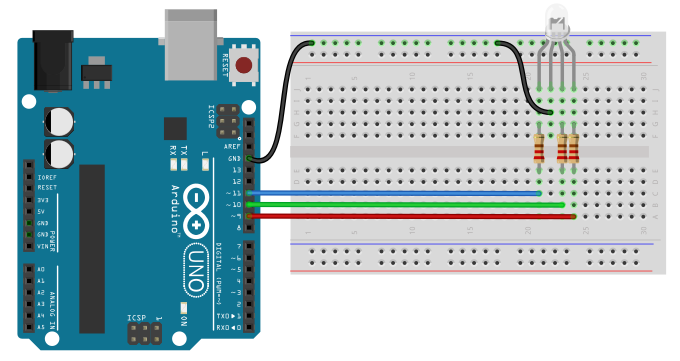
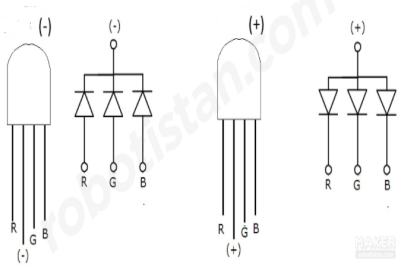
/\* The loop starts again \*/

**}**

**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);**

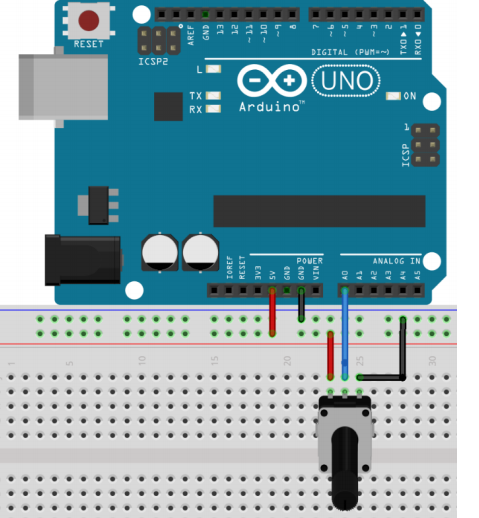
// We display the white color by activating all the leds. **digitalWrite(BlueLed, HIGH); digitalWrite(GreenLed, HIGH); digitalWrite(RedLed, HIGH); delay(1000); }**

# **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:

**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);**

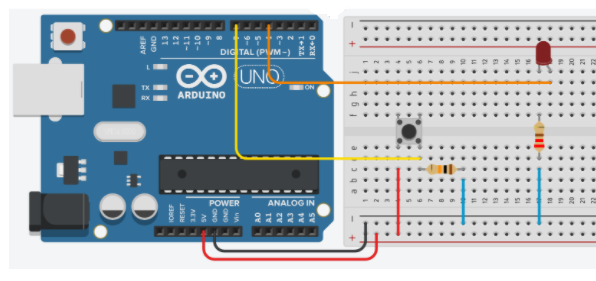
**delay(1000);** // delay between reads for stability

**}**

# **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,

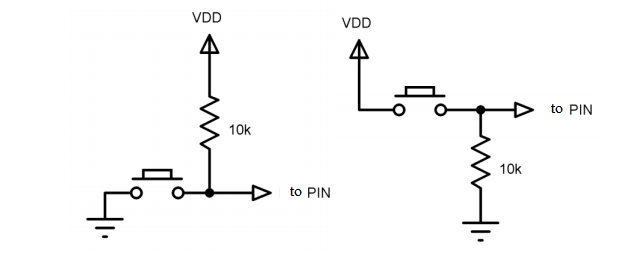
**digitalWrite(4, HIGH);**        // Led is ON,

**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 differents ways. Pull\_Up connection and Pull-down connection.

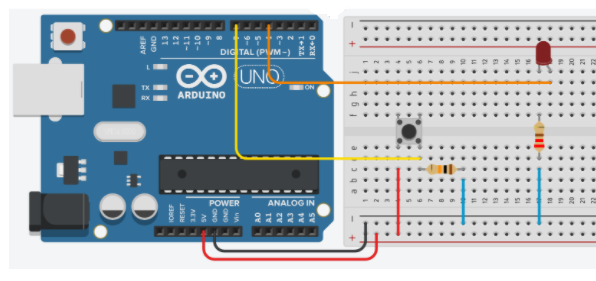


**Figure:** Switches or Butons that can be used for the IF…THEN command

# **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);**

**}**

**void loop()**

**{**

**if (digitalRead(buton) == HIGH)**    // The button is ON,

**digitalWrite(led, HIGH);**   // Led is ON

**else**                                // If not,

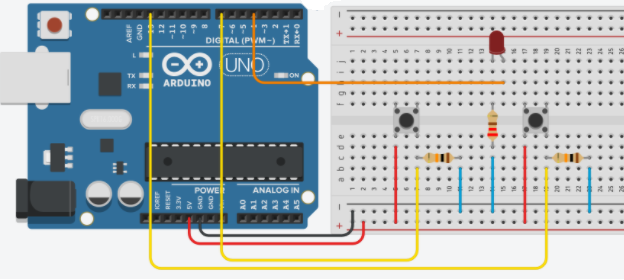
**digitalWrite(led, LOW);**    // Led is OFF.

**}**

# **Circuit 8:**

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

**Circuit Explanation:**  One buton turns on the Led, another buton 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**

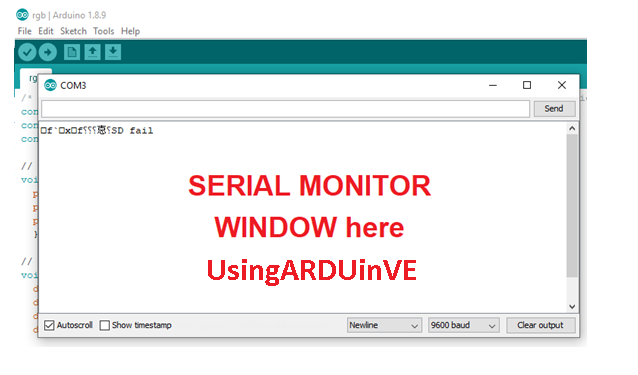
The Arduino IDE has a feature that can be a great help in debugging sketches or controlling Arduino from your computer's keyboard.

**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()](https://www.arduino.cc/reference/en/language/functions/communication/serial/print).

Serial.println(val);   
 Serial.println(val, format); Serial.printle(13, BIN);  gives "1101", binary value of 15 on erial 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.

# **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()**

**{**

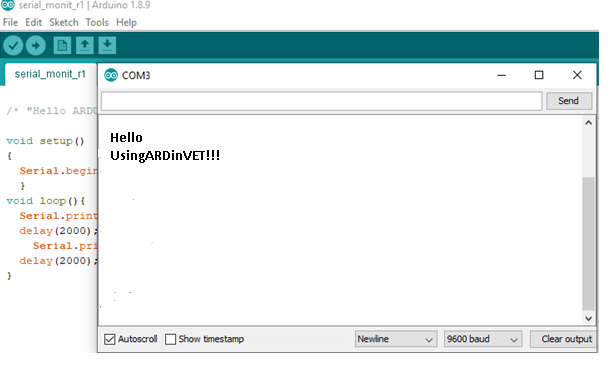
**Serial.println(" HELLO ");**

**delay(2000);**

**Serial.println(" UsingARDinVET!!!");**

**delay(2000);**

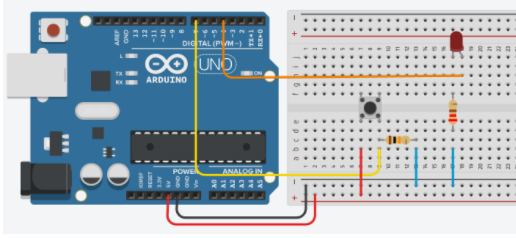
**}**



# **Circuit 10:**

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

**Circuit Explanation:** If the buton is pressed, "Led is ligthing, LED=ON" message is on the serial monitor and the Led is lighting. If 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.

**{**

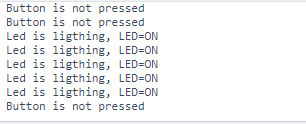
**digitalWrite(4, LOW);**

**Serial.println("Button is not pressed");**

**delay(1000);**

**}**

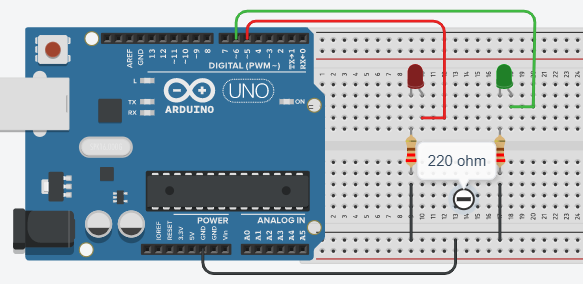
**}** //The view of the serial monitor is seen below.



# **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 nd 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 ("------------------------------"); }**

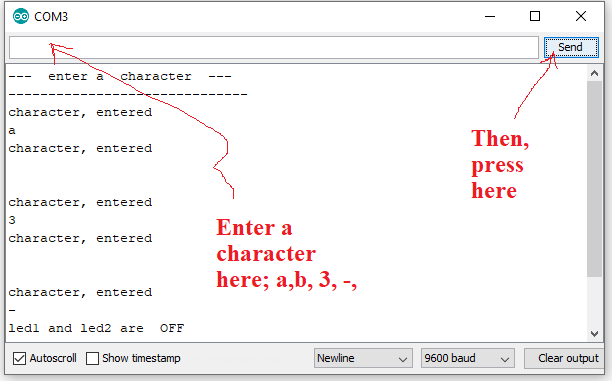
**void loop() { “ if (Serial.available()>0) { character=Serial.read(); Serial.println ("character, entered "); Serial.println (character);**

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

**if (character=='a') digitalWrite (led1, 1);**

**if (character=='3') digitalWrite (led2, 1);**

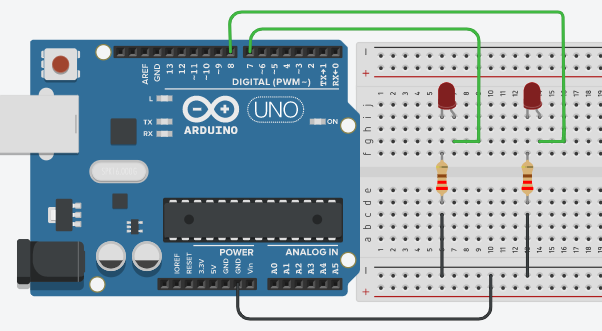
**if (character=='-') { digitalWrite(led1,0); digitalWrite(led2,0); Serial.println ("led1 and led2 are OFF "); } } }**



# **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);**

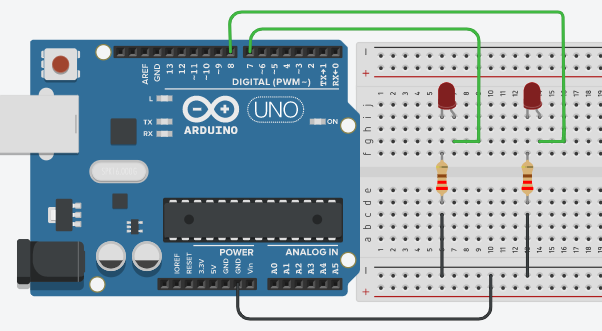
**digitalWrite(led1, 0); digitalWrite(led2, 0); delay(1000); } }**

# **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() {**

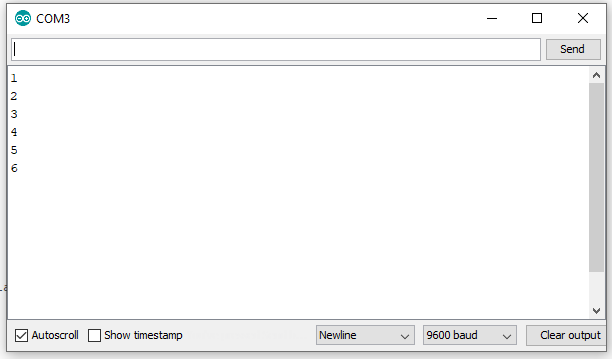
**for(int i=1; i<=6; i++)** // the value of i = 1, 2,3, 4,5, 6

**{**

**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.

**}**

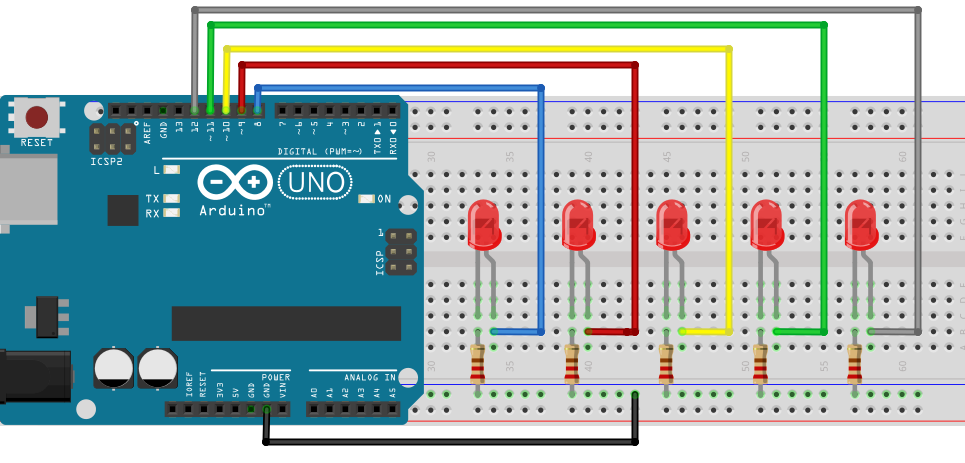
**NOTE:** To restart, the reset button must be pressed.



# **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:**

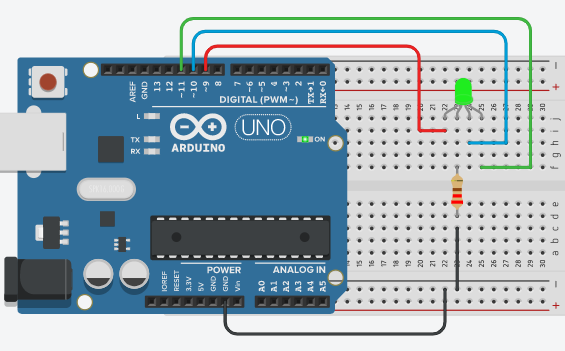
**Circuit title:** " Producing random colors with RGB led, Using the switch/case command”

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

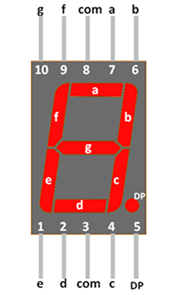
**case 3:** //No colour **digitalWrite (R, 0); digitalWrite (G, 0); digitalWrite (B, 0); delay(dly); break; } }**

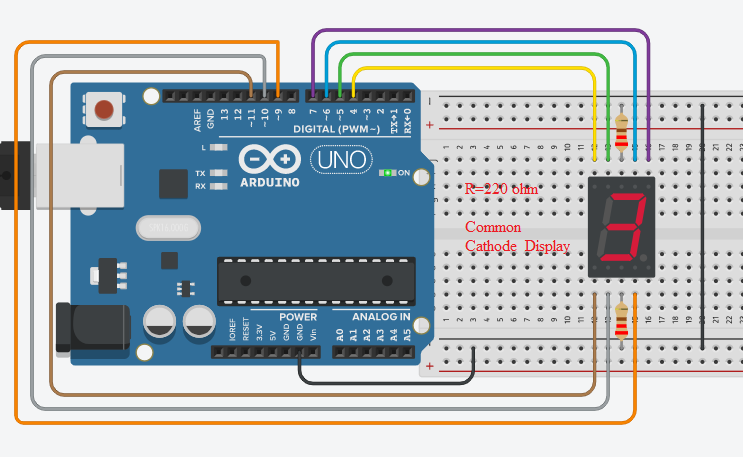
# **Circuit 16:**

**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-Cathoode 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; } } }**

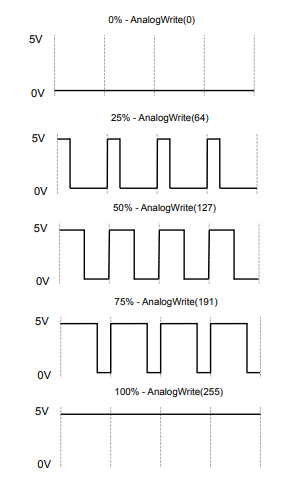
# **Circuit 17:**

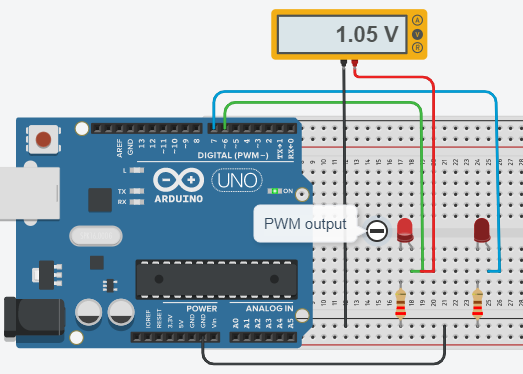
**Circuit title:** " Using the PWN Technique”

**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:



****

**/\* Learnning 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.

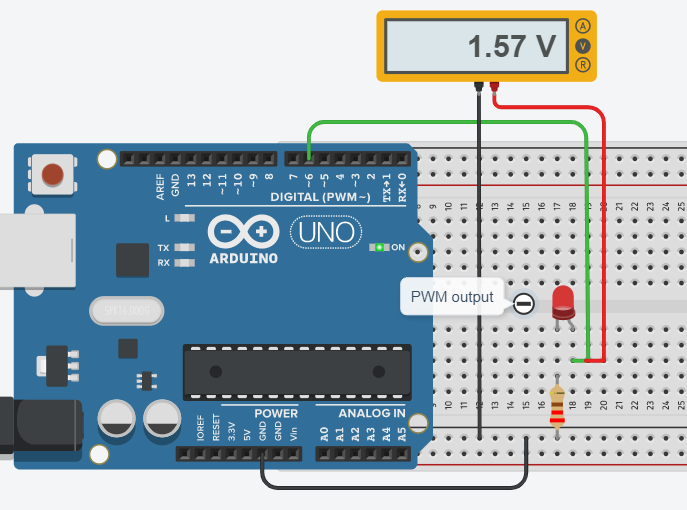
**analogWrite(led2,60); //** The value of 60 is sent to Led2 pin, i.e 1.05 V.

**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);**

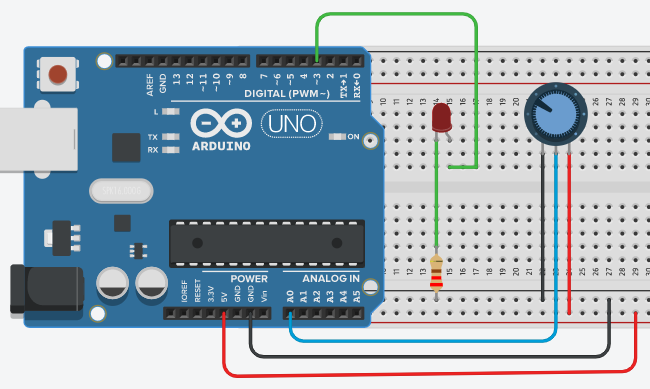
**analogWrite(led1, a);**

**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:** " Fotentiometer 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

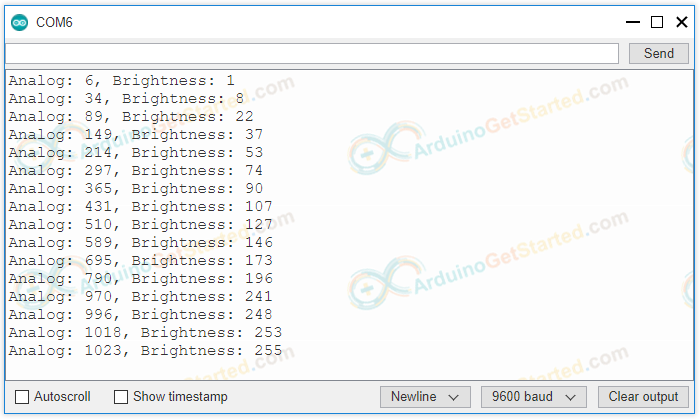
**Serial.print("Analog: ");**

**Serial.print(analogValue);**

**Serial.print(" Brightness: ");**

**Serial.println(brightness);**

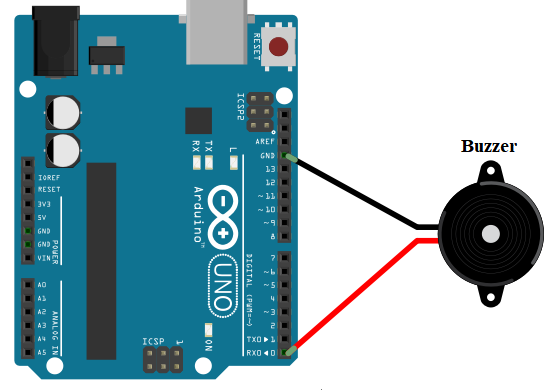
**delay(100);** // Serial Monitor screen i below **}**



# **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);**

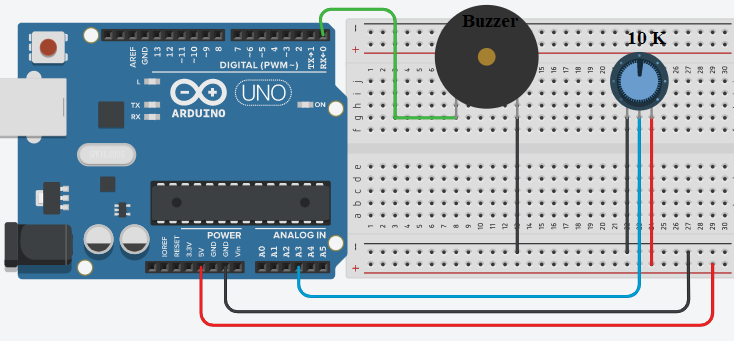
**digitalWrite(buzzer\_pin, LOW);**

**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;**

**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

**}**