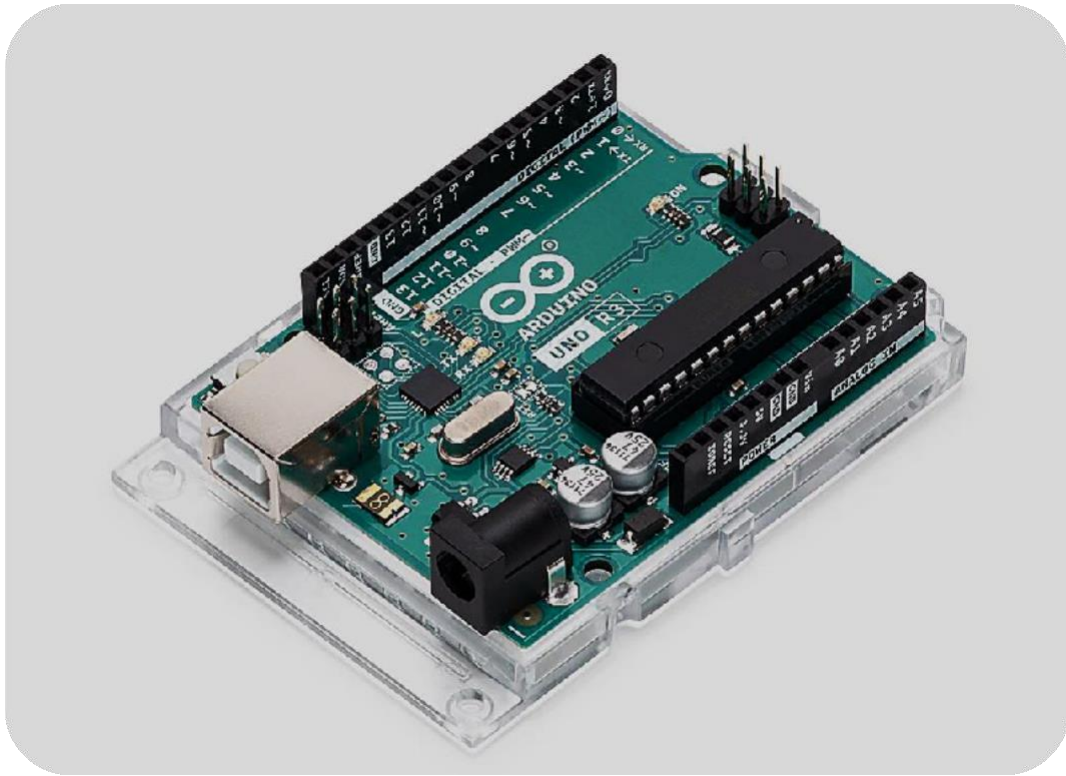


INTRODUCTION TO ARDUINO

Over the years, Arduino has been the brain behind thousands of projects, ranging from everyday objects to complex scientific instruments. A global community of makers, including students, hobbyists, artists, programmers, and professionals, has gathered around this open-source platform.



Arduino UNO R3 Board

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. Once it reached a broader community, the Arduino board began to evolve to adapt to new needs and challenges, diversifying from simple 8-bit boards to products for IoT applications, wearable devices, 3D printing, and embedded environments.

WHAT IS ARDUINO?

Arduino is a hardware and software-based development platform designed for creating interactive projects. Arduino boards contain an Atmel AVR microcontroller and various electronic components for circuit connections. Arduino is a completely open-source platform.

It is programmed using a wiring-based programming language and the Arduino IDE, which is a processing-based software development environment. The Arduino programming language is almost identical to the C programming language, making it easy to write code with a basic understanding of C++.



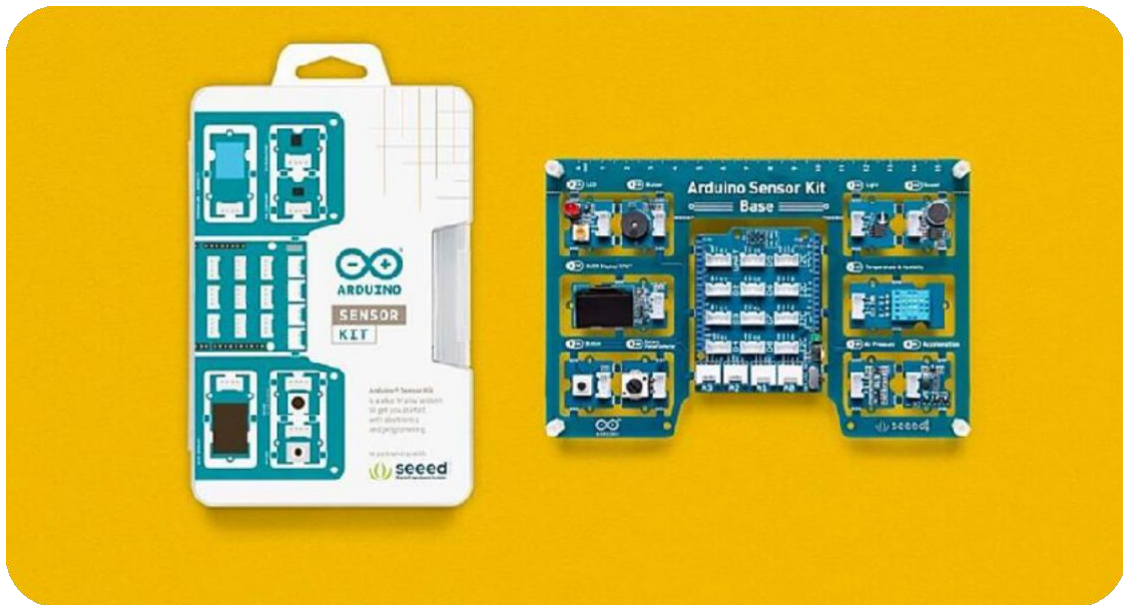
Avrupa Birliği tarafından
ortak finanse edilmektedir

By sending a set of instructions to the microcontroller on the board, you can tell your Arduino what to do. To do this, you use the Arduino programming language (based on Wiring) and the Arduino Software (IDE) which is based on Processing. Once the software is complete, the code is easily transferred to the board via a USB cable.

Why Arduino?

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy to use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux.

Teachers and students can use it to create low-cost scientific instruments, demonstrate principles of chemistry and physics, or get started with programming and robotics. Arduino is an essential tool for learning new things. Anyone—children, hobbyists, artists, or programmers—can start tinkering by simply following step-by-step instructions from a kit or by sharing ideas online with other members of the Arduino community.



Arduino Sensor Kit

There are many microcontrollers and microcontroller platforms available for physical computing. Arduino takes the complex details of microcontroller programming and combines them into an easy-to-use package. Arduino also simplifies the process of working with microcontrollers but offers several advantages over other systems for teachers, students, and interested hobbyists:

- **Inexpensive** – Arduino boards are relatively cheap compared to other microcontroller platforms. The cheapest version of the Arduino module can be assembled by hand, and even pre-assembled Arduino modules cost less than \$50.
- **Cross-Platform** – Arduino Software (IDE) works on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.



Avrupa Birliği tarafından
ortak finanse edilmektedir

- **Simple, Clear Programming Environment** – Arduino Software (IDE) is easy to use for beginners but flexible enough for advanced users to benefit from as well.
- **Open Source and Extensible Software** – The Arduino software is published as open-source tools that can be extended by experienced programmers. The language can be expanded through C++ libraries, and those who wish to understand the technical details can transition from Arduino to the underlying AVR C programming language. Similarly, you can directly incorporate AVR-C code into your Arduino programs if you wish.

GETTING STARTED WITH ARDUINO

Introduction to Hardware, Software Tools, and the Arduino API:

Since its inception in 2005, the Arduino platform has become one of the most recognized brands in the field of electronics and embedded design.

So, what are the building blocks of Arduino? What is a "board," how do I write code for it, and what tools do I need to create my own project? The aim of this guide is to provide you with an overview of Arduino projects.

Arduino Hardware

Over the years, Arduino has released hundreds of hardware designs in many shapes and forms. These include:

Classic Arduino Family

			
Arduino UNO R4 Minima	Arduino UNO R4 WiFi	Arduino UNO R3	Arduino Leonardo
			
Arduino UNO Mini Limited Edition	Arduino Micro	Arduino Zero	Arduino UNO WiFi Rev2



Avrupa Birliği tarafından
ortak finanse edilmektedir

Arduino Nano Family

The Nano Family is a series of boards with a small footprint, packed with features. These boards include a variety of embedded sensors, such as temperature/humidity, pressure, motion, microphone, and more. They can also be programmed with MicroPython and support Machine Learning.

Arduino MKR Family

The MKR Family consists of a series of boards, shields, and carriers that can be combined to create amazing projects without any additional circuitry. Each board (except MKR Zero) is equipped with a radio module that enables Wi-Fi®, Bluetooth®, LoRa®, Sigfox®, and NB-IoT communication. All boards in the family are based on the Arm® Cortex®-M0 32-bit SAMD21 low-power processor and are equipped with a crypto chip for secure communication.

The MKR Family shields and carriers are designed to expand the board's functionalities: including environmental sensors, GPS, Ethernet, motor control, and RGB matrices.

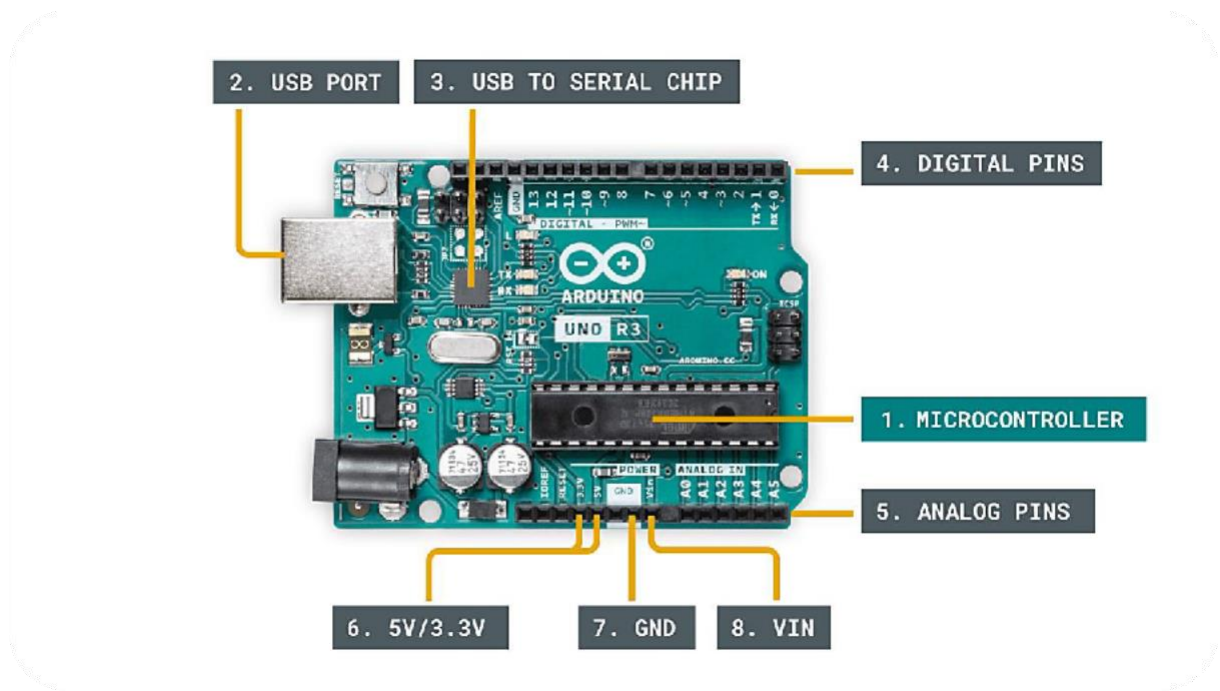
Arduino Mega Family

In the Mega family, you will find boards aimed at projects that require high processing power and GPIO pins.

Anatomy of the Arduino Uno R3 Board

While all Arduino boards are different, there are a few fundamental components that can be found in nearly every Arduino. Let's take a look at the image below:





Basic Components of an Arduino Board

1. Microcontroller - This is the brain of an Arduino and the component into which we load programs. Think of it as a small computer designed to execute only a specific number of tasks. It has an Atmel Atmega 328P microcontroller.

- **2. USB Port** - Used to connect your Arduino board to your computer.
- **3. USB to Serial Chip** - The USB to Serial chip is an important component because it helps convert data from a computer to the embedded microcontroller. This is what allows you to program the Arduino board from your computer.
- **4. Digital Pins** - Pins that use digital logic (0, 1, or LOW/HIGH). They are typically used for switches and turning an LED on and off.
- **5. Analog Pins** - Pins that can read analog values with a resolution of 10 bits (0-1023).
- **6. 5V / 3.3V Pins** - These pins are used to supply power to external components.
- **7. GND** - Also known as ground, negative, or just -, this is used to complete a circuit at an electrical level of 0 volts.
- **8. VIN** - Stands for Voltage Input, where you can connect external power sources.

You will find many more components depending on the Arduino board. The items listed above are generally found on any Arduino board.

Basic Operations



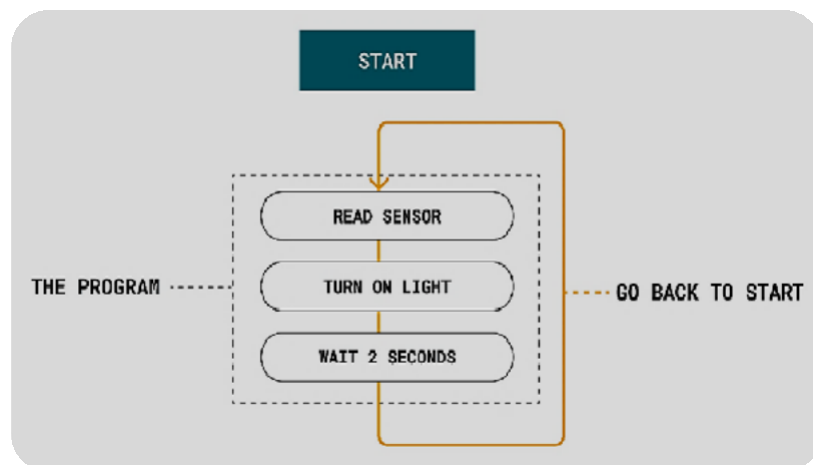
Avrupa Birliği tarafından
ortak finanse edilmektedir

Most Arduino boards are designed to have a single program running on the microcontroller. This program can be designed to perform a single action, like blinking an LED. It can also be designed to execute hundreds of actions in a loop. The scope varies from one program to another.

The program loaded onto the microcontroller will start running as soon as power is supplied. Each program has a function called "loop." Within the loop function, you can do things like:

- Read a sensor.
- Turn on a light.
- Check if a condition is met.
- All of the above.

The speed of a program is incredibly fast unless we tell it to slow down. It depends on the size of the program and how long it takes the microcontroller to execute it, but it is generally in microseconds (millionths of a second).



The Basic Operation of Arduino

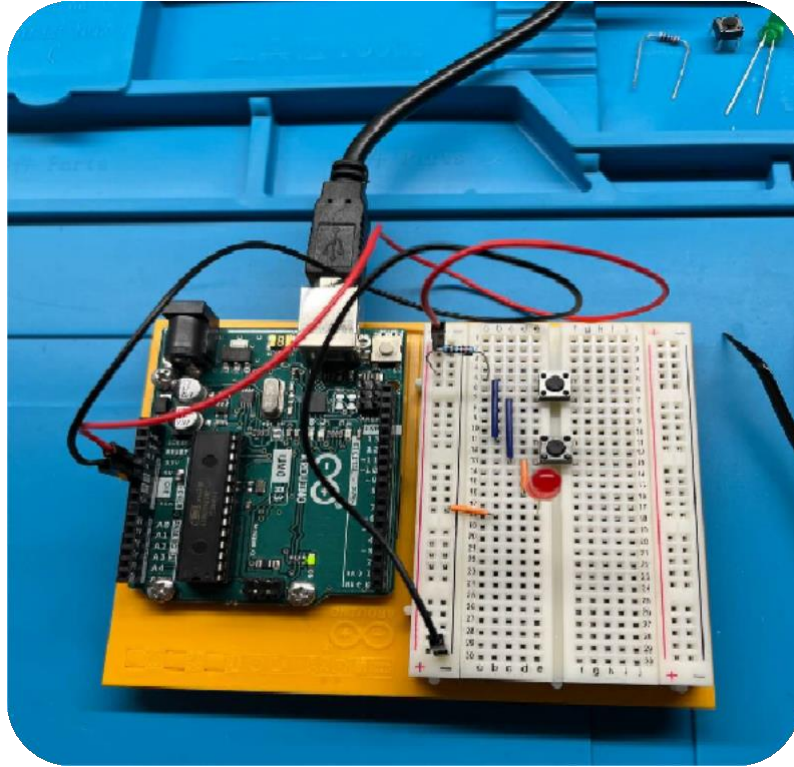
Circuit Basics

Circuits consist of at least one active electronic component and conductive materials like wires to allow current to flow. When working with Arduino, you will most often be building a circuit for your project.

A simple example of a circuit is an LED circuit. A wire connects a pin on the Arduino to a resistor (to protect the LED from high current), and finally to an LED, which is connected to the ground pin (GND). When the pin is set to HIGH, the microcontroller on the Arduino board allows an electric current to flow through the circuit, turning the LED on. When the pin is set to LOW, the LED turns off because no electric current is flowing through the circuit.

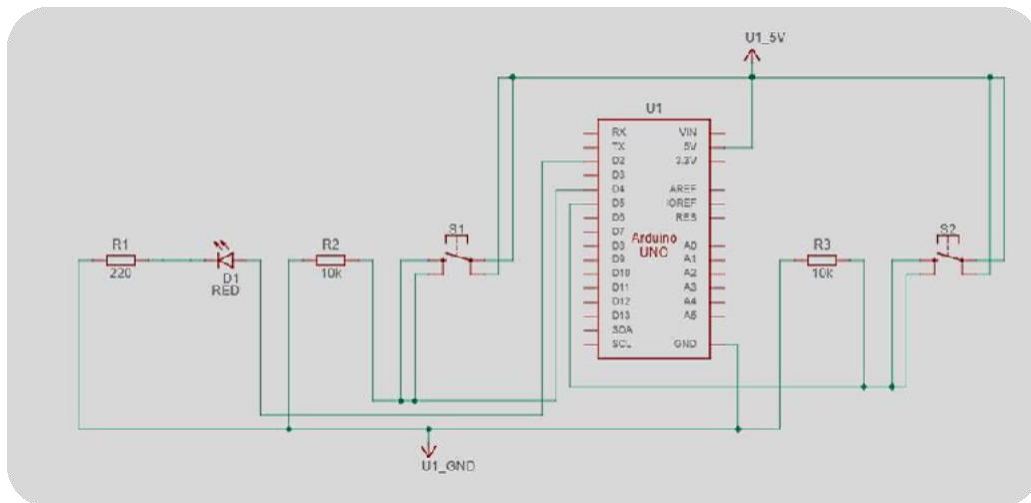


Avrupa Birliği tarafından
ortak finanse edilmektedir



LED Circuit with Arduino

Circuits are typically represented as diagrams that are plans for your circuit. The image below shows the schematic representation of the same circuit shown in the image above.



Schematic of an LED Circuit with Arduino

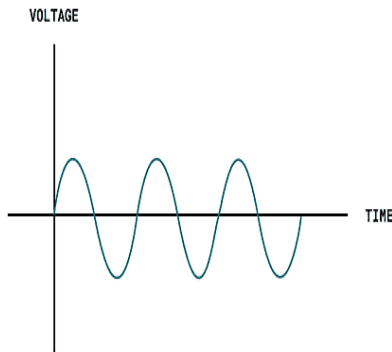
Electronic Signals

All communication between electronic components is facilitated by electronic signals. There are two main types of electronic signals: analog and digital.

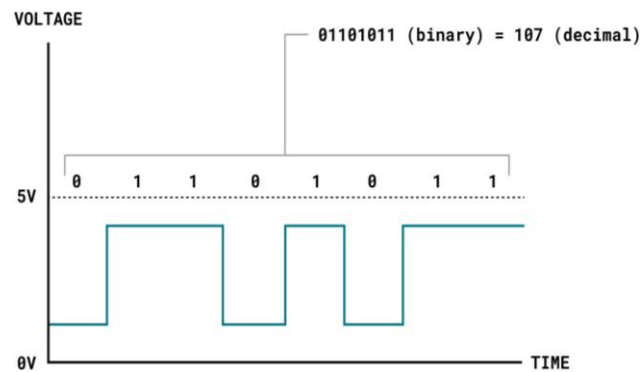


Avrupa Birliği tarafından
ortak finanse edilmektedir

Analog Signal



Digital Signal



Fundamentals of Analog Signal

An analog signal is typically dependent on a range. On an Arduino, this range is usually 0-5V or 0-3.3V. For example, if we use a potentiometer (an analog component used to change the resistance in a circuit), we can manually adjust this range (0-5V). In the program, this is represented in a range of 0-1023, which corresponds to a 10-bit resolution.

If we write an analog signal using Pulse Width Modulation (PWM), we can use a range of 0-255 since we are using an 8-bit resolution.

Fundamentals of Digital Signal

A digital signal works a bit differently; it represents only two binary states (0 or 1) that are read in the program as high (HIGH) or low (LOW) states. This is the most common type of signal in modern technology.

You can easily read and write digital signals on an Arduino, which is useful for reading button states or turning something on and off. Digital signals may seem very basic (only 0 or 1), but they are actually much more sophisticated. For example, we can create a sequence by rapidly sending a high or low state several times. This is known as a binary sequence or bit stream.

Let's take a look at two binary sequences:

- 1- 101101
- 2- 101110001110011

In decimal format, it is as follows:

- 1- 45
- 2- 23667



Avrupa Birliği tarafından
ortak finanse edilmektedir

This is a clever way to send large amounts of data from one point to another by quickly transmitting high and low signals. We use Serial Communication Protocols to interpret the data coming from the signals.

Sensors and Actuators

When working with Arduino, it is important to understand sensors, actuators, and the differences between them.

What is a Sensor?

A sensor, in its simplest definition, is used to perceive its environment, meaning it records a physical parameter, such as temperature, and converts it into an electronic signal. Sensors can also take the form of a simple button: when a state changes (for example, when we press a button), the electronic signal transitions from low to high (from 0 to 1).

There are many types of sensors and various ways to record data from them. Perhaps the easiest to use is the analog sensor, which transmits a range of values by varying the voltage input supplied to an Arduino analog pin (usually between 0-5 volts). This simply gives you a range of 0-1023 (10-bit resolution).

In many cases where we use a library, all we need is a single line of code:

```
1 sensorValue = sensor.read();
```

What is an Actuator?

An actuator, in simple terms, is used to initiate or change a physical state. Some examples include:

- A light (for example, an LED).
- A motor.
- A switch.

Actuators convert electrical signals into radiant energy (light) or mechanical energy (movement).

How actuators are controlled really depends on the type of components we have. The simplest way is to simply turn something on or off, while a more advanced method involves controlling the amount of voltage supplied to a component (such as the speed of a motor).

Common functions used to control actuators include `digitalWrite()` and `analogWrite()`.



Avrupa Birliği tarafından
ortak finanse edilmektedir

```
1 digitalWrite(LED, HIGH); //turn on an LED
2 digitalWrite(LED, LOW);  //turn off an LED

3 analogWrite(motor, 255); //set a motor to maximum capacity
4 analogWrite(motor, 25);  //set a motor to 10% of its capacity
```

Input and Output

Sensors and actuators are commonly referred to as inputs and outputs. When we write a program, it is common to create conditions that check the status of a sensor and decide whether to trigger an action.

A basic example of this is a button and an LED. We can write a condition that checks whether a button is pressed, turns on the LED, and turns it off when the button is not pressed. In an Arduino program, this looks like:

```
1 int buttonState = digitalRead(buttonPin); //read and store the
                                           button state (0 or 1)
2
3 if(buttonState == HIGH){                //check if state is high (button
                                           is pressed)
4     digitalWrite(LED, HIGH);            //turn on LED
5 } else {
6     digitalWrite(LED, LOW);             //turn off LED
7 }
```

Serial Communication Protocols There are several serial communication protocols that use the digital signals mentioned above to send data.

The most common ones are UART, SPI, and I²C. The UART protocol is used, among other things, to send data between a computer and an Arduino board, such as when loading a new program or reading data directly from an Arduino.

SPI and I²C protocols are used for communication between both internal and external components. Communication is carried out by something called a serial bus, which is connected to a specific pin on the Arduino.

Using the I²C protocol, we can connect multiple sensors on the same pin and accurately receive data. Each device has an address that we need to specify in our program when making data requests.

Memory

The "standard" Arduino typically has two types of memory: SRAM and Flash memory.



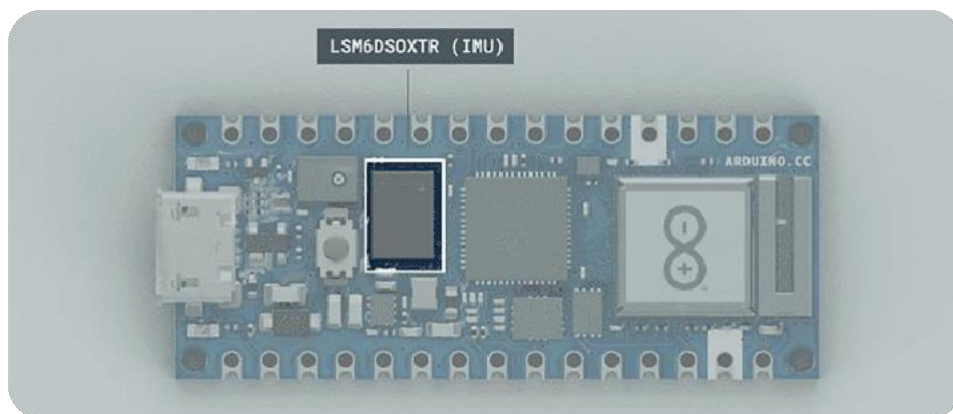
Avrupa Birliği tarafından
ortak finanse edilmektedir

SRAM (Static Random Access Memory) is used to store values such as the state of a variable (e.g., a Boolean). This memory is reset when power is turned off.

Flash memory is primarily used to store the main program or instructions for the microcontroller. This memory is not erased when power is turned off, allowing the microcontroller instructions to execute as soon as the board is powered.

The amount of memory available on an Arduino varies from board to board. For example, the Arduino UNO has 32kB of Flash / 2kB of SRAM, while the Nano 33 IoT has 256kB of Flash / 32kB of SRAM.

Embedded Sensors



An IMU (Inertial Measurement Unit) on the Nano RP2040 Connect board.

Many new Arduino boards are equipped with embedded sensors. For example, the Nano 33 BLE Sense has 7 embedded sensors but is only 45x18 mm in size (about the size of a thumb). All of these connect via the I²C protocol, as mentioned above, and each has a unique address.

Arduino API

The Arduino API, also known as the "Arduino Programming Language," consists of various functions, variables, and structures based on the C/C++ language.

Main Components

The Arduino API can be divided into three main sections: functions, variables, and structures:

- **Functions:** Used to control the Arduino board and perform calculations. For example, to read from or write to a digital pin, match a value, or use serial communication.
- **Variables:** Arduino constants, data types, and conversions. For example, int, boolean, array.
- **Structures:** Elements of Arduino (C++) code, such as:



Avrupa Birliği tarafından
ortak finanse edilmektedir

- Sketch (loop(), setup())
- Control structures (if, else, while, for)
- Arithmetic operators (multiplication, addition, subtraction)
- Comparison operators, such as == (equals), != (not equal), > (greater than).

The Arduino API can be defined as a simplified version of the C++ programming language that includes many additions for controlling Arduino hardware.

Program Structure

The absolute minimum requirement for an Arduino program is the use of two functions: void setup() and void loop(). The term "void" indicates that nothing is returned during execution.

- **void setup()** - This function is executed once when the Arduino is powered on. Here, we define things like a pin's mode (input or output), the baud rate for serial communication, or the initialization of a library.

- **void loop()** - Here, we write the code that we want to run repeatedly, such as turning a light on or off based on an input, or reading a sensor every X seconds.

The functions mentioned above are always required in an Arduino sketch, but of course, you can add several more functions that are useful for longer programs.

Sketch

In an Arduino project, a program is referred to as a "sketch." A sketch is a file that contains your program code. It has a .ino extension and is always stored in a folder with the same name.

The folder can also contain other files, such as header files, that can be included in your sketch.

Example Sketch

Below is an example of a standard Arduino sketch that includes some popular Arduino programming elements.

```
1 /*
2 This is a comment at the top of a program,
3 it will not be recognized as code. Very good
4 to add an explanation of what your code does
5 here.
6
7 This sketch shows how to read a value from a
8 sensor connected to pin A1, print it out in
9 the Serial Monitor, and turn on an LED connected
10 to pin number 2 if a conditional is met.
11 */
```



Avrupa Birliği tarafından
ortak finanse edilmektedir

```

12
13 int sensorPin = A1; //define pin A1 (analog pin)
14 int ledPin = 2; //define pin 2 (digital pin)
15 int sensorValue; //create variable for storing readings
16
17 //void setup is for configurations on start up
18 void setup() {
19     Serial.begin(9600); //initialize serial communication
20     pinMode(ledPin, OUTPUT); //define ledPin as an output
21 }
22
23 void loop() {
24     sensorValue = analogRead(sensorPin); // do a sensor reading
25
26     Serial.print("Sensor value is: "); //print a message to
                                         the serial monitor
27     Serial.println(sensorValue); //print the value to the
                                         serial monitor
28
29     //check if sensorValue is below 200
30     if(sensorValue < 200) {
31         digitalWrite(ledPin, HIGH); //if it is, turn on the
                                         LED on pin 2.
32     }
33     //if sensorValue is above 200, turn off the LED
34     else{
35         digitalWrite(ledPin, LOW);
36     }
37 }

```

Libraries

Arduino libraries are an extension of the standard Arduino API and consist of thousands of libraries contributed by both official and community sources. Libraries simplify the use of complex code, such as reading a specific sensor, controlling a motor, or connecting to the Internet. Instead of writing all that code yourself, you can load a library, add it to the top of your code, and use any of the available functions from the library. All Arduino libraries are open source and can be used freely by anyone.

To use a library, you need to include it at the top of your code, as shown in the example below:

```
1 #include <Library.h>
```

Arduino Software Tools

Another integral part of the Arduino ecosystem is the software tools. Commonly known as the Arduino IDE, it is an integrated development environment. But what does this really



Avrupa Birliği tarafından
ortak finanse edilmektedir

mean? You need to write a program to program your board, compile this program into machine code, and finally upload the new program to your board. The Arduino IDE simplifies all of this, from writing the first line of code to running it on the microcontroller of the Arduino board. It is a program or application that you can download (or use an online version) to manage all your code development. In the past, this was a complex process requiring a good set of knowledge in electronics and computer science. Now, with the help of the Arduino IDE, anyone can learn how to do it.

Currently, there are three versions of the Arduino IDE available:

- Arduino IDE 1.8.x (classic)
- Arduino IDE 2 (new)
- Arduino Cloud Editor (online)

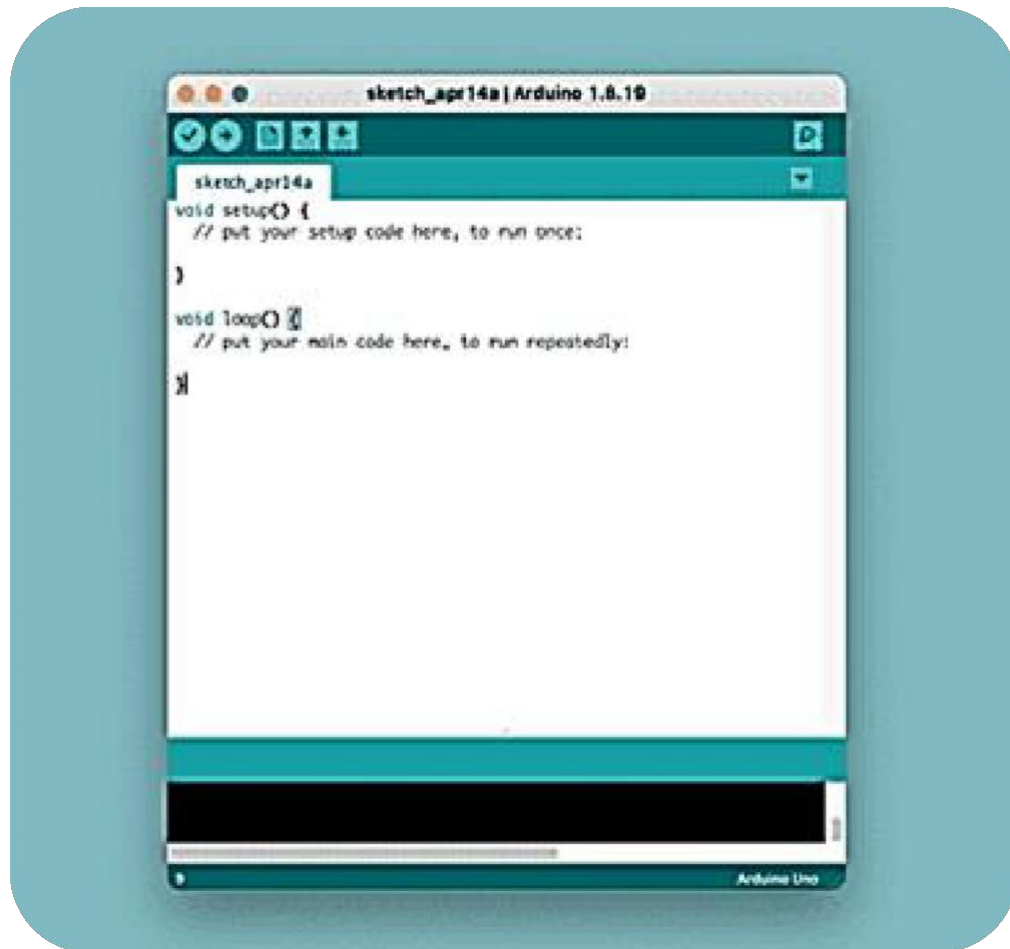
Typical Workflow To upload code to the Arduino board using the IDE, the following steps are generally taken:

1. Set up your board – this means installing the correct “package” for your board. You cannot use your board without the package. The installation is done directly in the IDE and is a quick and easy process.
2. Create a new sketch – the sketch is your main program file. Here, we write a series of instructions that we want to execute on the microcontroller.
3. Compile your sketch – the code we wrote does not appear exactly as it will when uploaded to our Arduino: compiling the code means we check for errors and convert it into a binary file (1s and 0s). If something fails, you will see the error in the console.
4. Upload your sketch – once the compilation is successful, the code can be uploaded to your board. At this step, we physically connect the board to the computer and select the correct serial port.
5. Serial Monitor (optional) – for most Arduino projects, it is important to know what is happening on your board. The Serial Monitor tool, available in all IDEs, allows data to be sent from your board to your computer.

6. Arduino IDE 1.8.x



Avrupa Birliği tarafından
ortak finanse edilmektedir



Classic Arduino IDE

Today, the Arduino IDE 1.8.X, or "Java IDE," is considered the "old" editor and was released during the early days of Arduino's launch.

Arduino IDE2

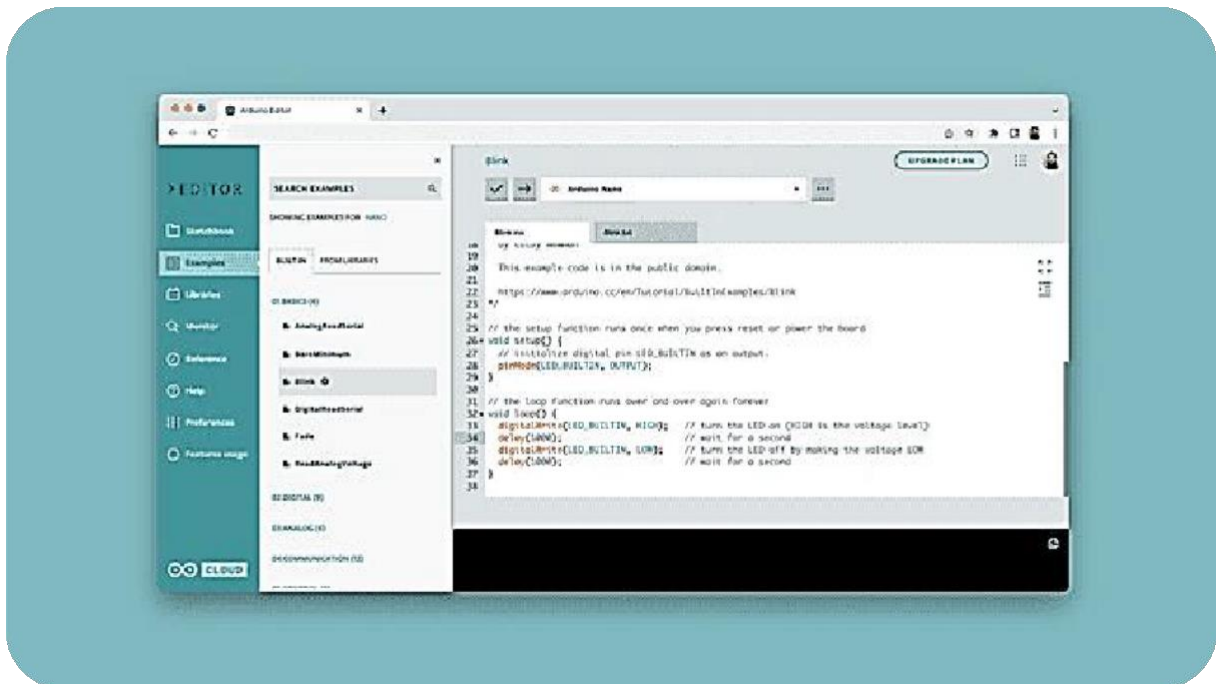


Avrupa Birliği tarafından
ortak finanse edilmektedir



New Arduino IDE

Cloud Editor



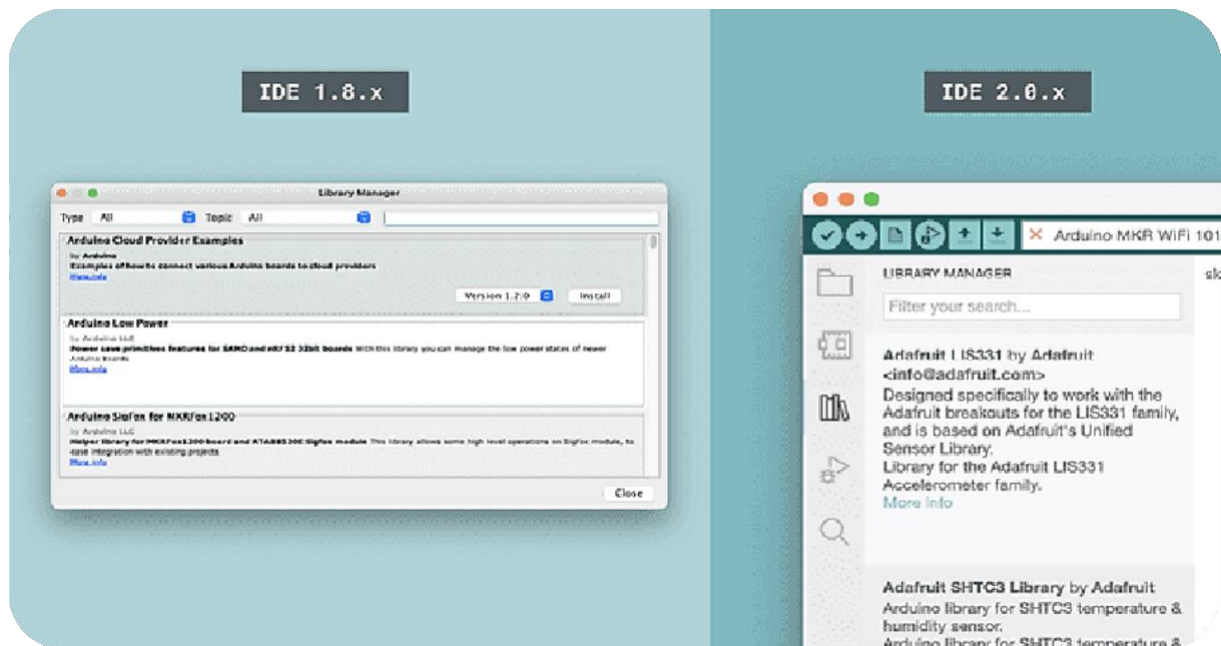
Cloud Editor

Arduino **Cloud Editor** is an online IDE that is part of the Arduino Cloud package. Functionally similar, this editor is fully web-based with online storage among other features. You will need to register an Arduino account to use the Cloud Editor.



Avrupa Birliği tarafından
ortak finanse edilmektedir

Library Manager



Library Manager in IDE 1.8.x and IDE 2

Each version of the IDE includes a library manager for installing Arduino software libraries. Thousands of libraries, both official and contributed, can be downloaded directly. Code examples for each library become available during the download.

Using Arduino Software (IDE) The offline IDE makes it easy to write code and upload it to the board without an internet connection. We recommend this software for users with weak or no internet connectivity. This software can be used with any Arduino board.

Currently, there are two versions of the Arduino IDE: IDE 1.xx and IDE 2.x. IDE 2.x is a new major release that is faster and more powerful than IDE 1.xx. In addition to being a more modern editor with a more responsive interface, it includes advanced features to assist users in coding and debugging.

The following steps can guide you when using the offline IDE (you can choose either IDE 1.xx or IDE 2.x):

1. Download and install the Arduino Software IDE: • Arduino IDE 1.xx (Windows, Mac OS, Linux, Portable IDE for Windows and Linux, ChromeOS). • Arduino IDE 2.x
2. Connect your Arduino board to your device.
3. Open the Arduino Software (IDE).

The Arduino Integrated Development Environment - or Arduino Software (IDE) - connects to Arduino boards to upload programs and communicate with them. Programs written using Arduino Software (IDE) are called sketches. These sketches are written in a text editor and saved with the .ino file extension.



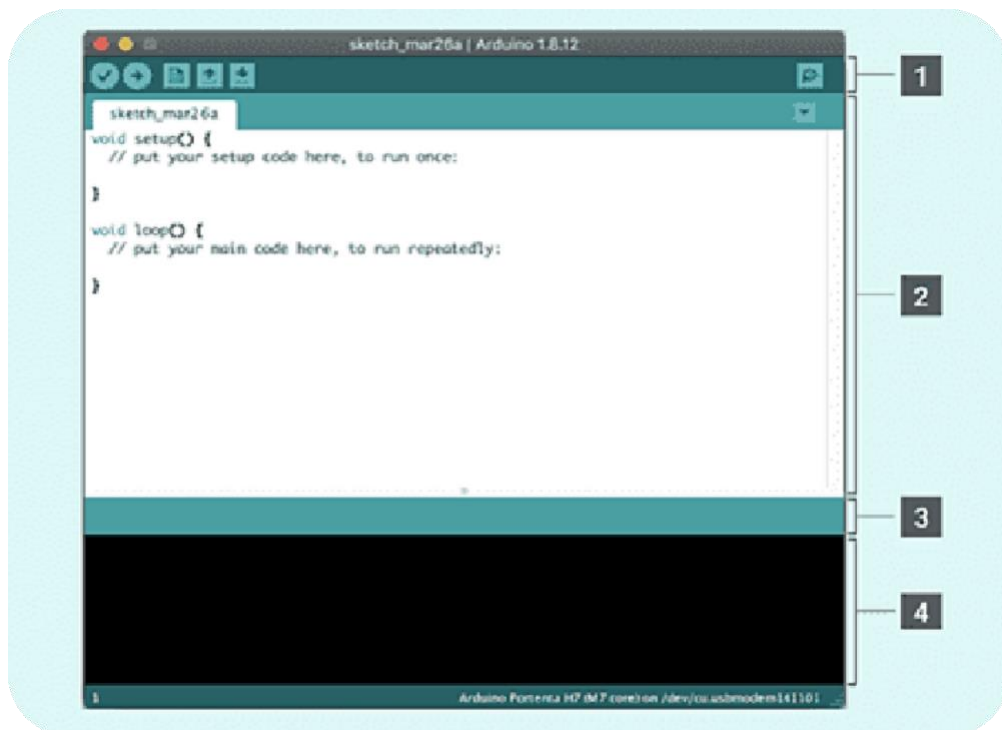
Avrupa Birliği tarafından
ortak finanse edilmektedir

Using Offline IDE 1.xx

The editor consists of four main areas:

1. A Toolbar that contains buttons for common functions and a series of menus. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.
2. A message area that provides feedback during saving and exporting and also displays errors.
3. A text editor where you will write your code.
4. A text console that displays the text output generated by the Arduino Software (IDE), including full error messages and other information.

In the lower right corner of the window, the configured board and serial port are displayed.



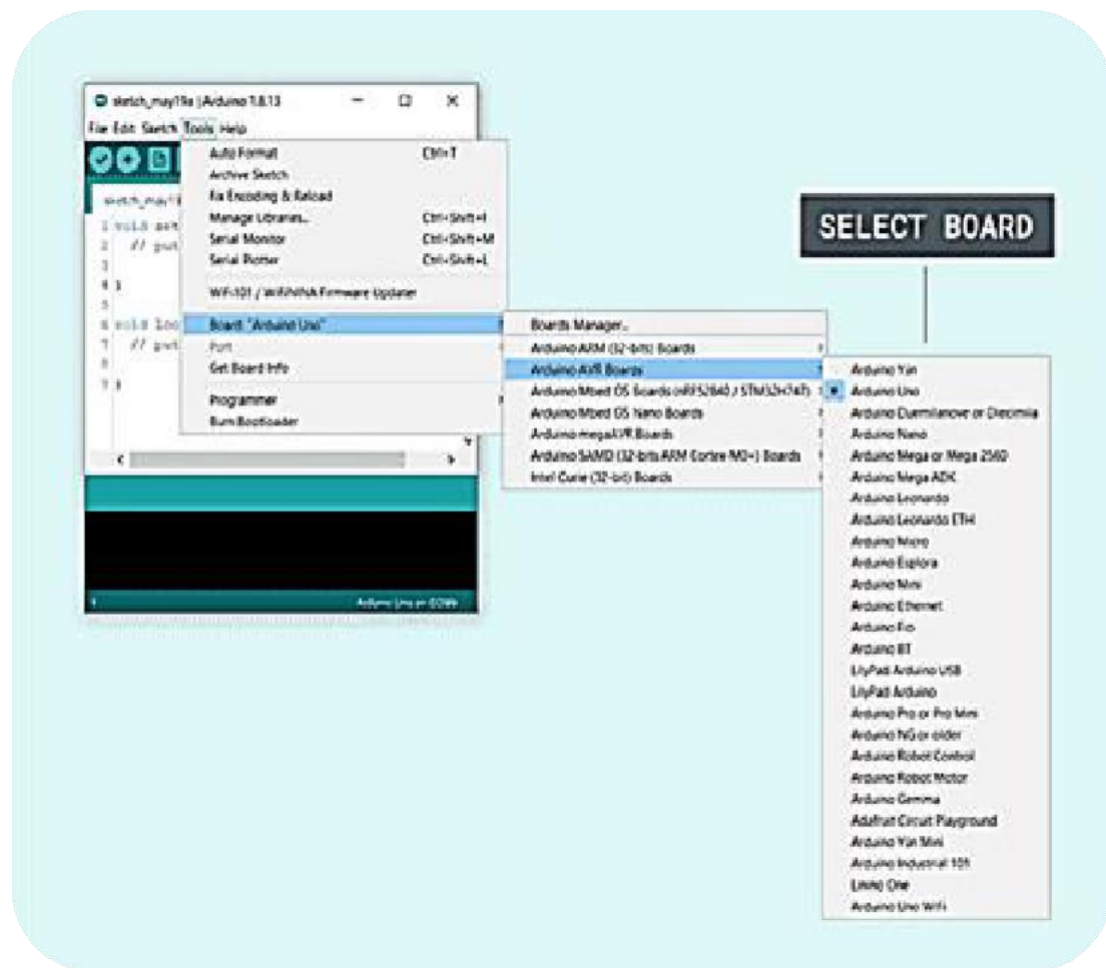
Arduino Software IDE

Now that everything is ready, let's try to make your board blink!

5. Connect your Arduino or Genuino board to your computer.
6. Now you need to select the correct core and board. This is done by going to Tools > Board > Arduino AVR Boards > Board. Make sure you select the board you are using. If you can't find your board, you can add it from Tools > Board > Board Manager.

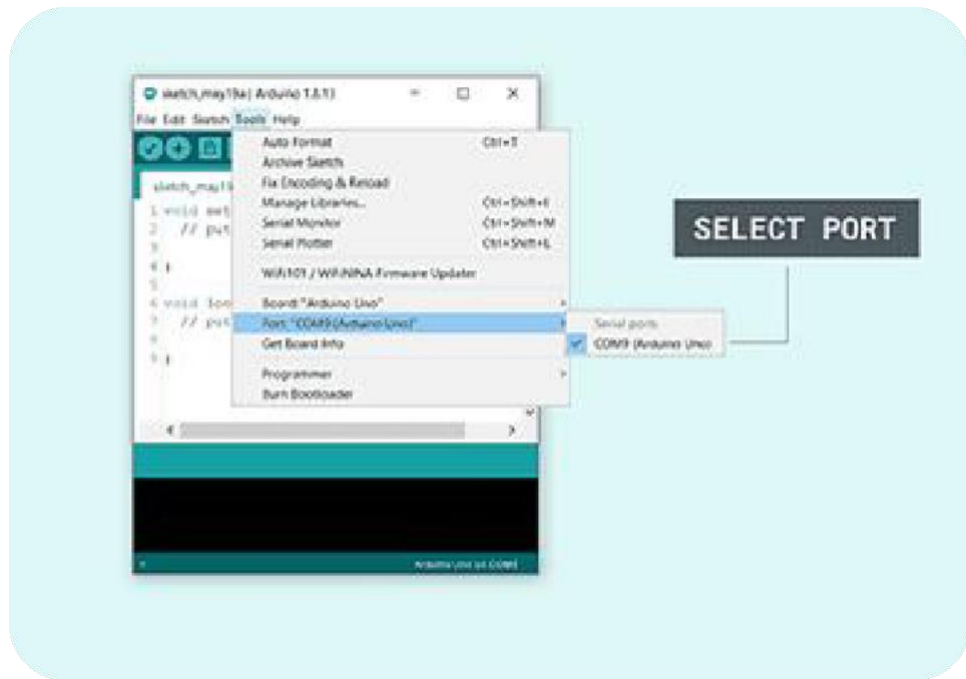


Avrupa Birliği tarafından
ortak finanse edilmektedir



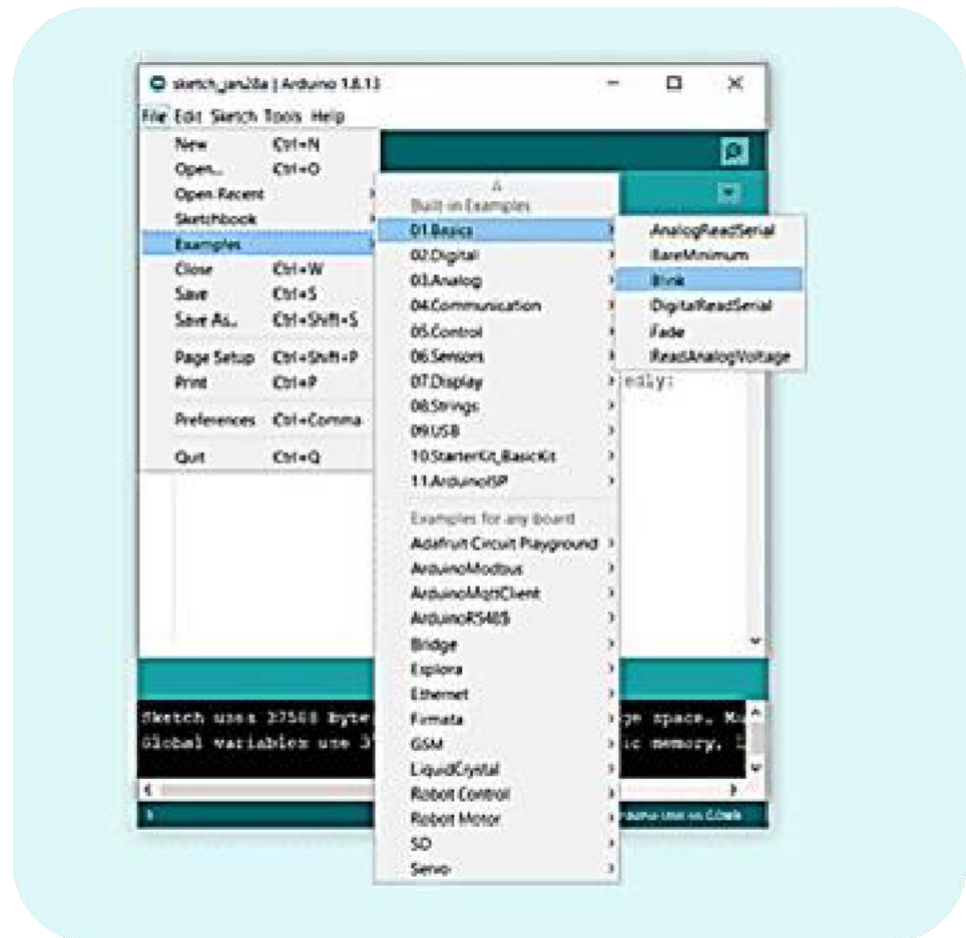
A Board Selection

7. Now, let's make sure your board is recognized by the computer by selecting the port. This is done simply by going to Tools > Port, where you can select your board from the list.



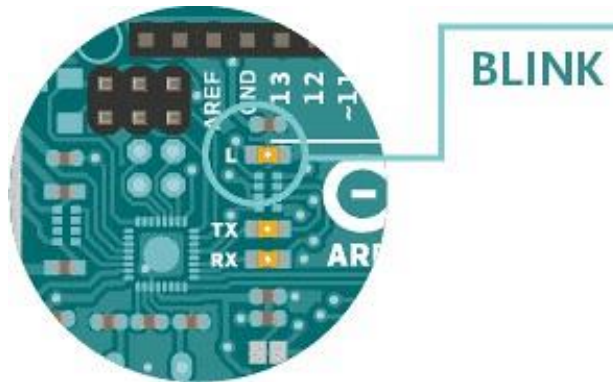
Port selection

8. Let's try an example: Go to File > Examples > 01.Basics > Blink.



Opening an example

9. To upload this to your board, simply click the arrow in the upper left corner. This process takes a few seconds, and it's important not to disconnect the board during this time. If the upload is successful, you will see a message saying 'Upload complete' in the lower output area.
10. After the upload is complete, you should see the yellow LED with an L next to it blinking on your board. You can adjust the blinking speed by changing the delay number in parentheses to 100, then re-upload the Blink sketch. The LED should now blink much faster.



Congratulations! You have successfully programmed your board to blink its built-in LED

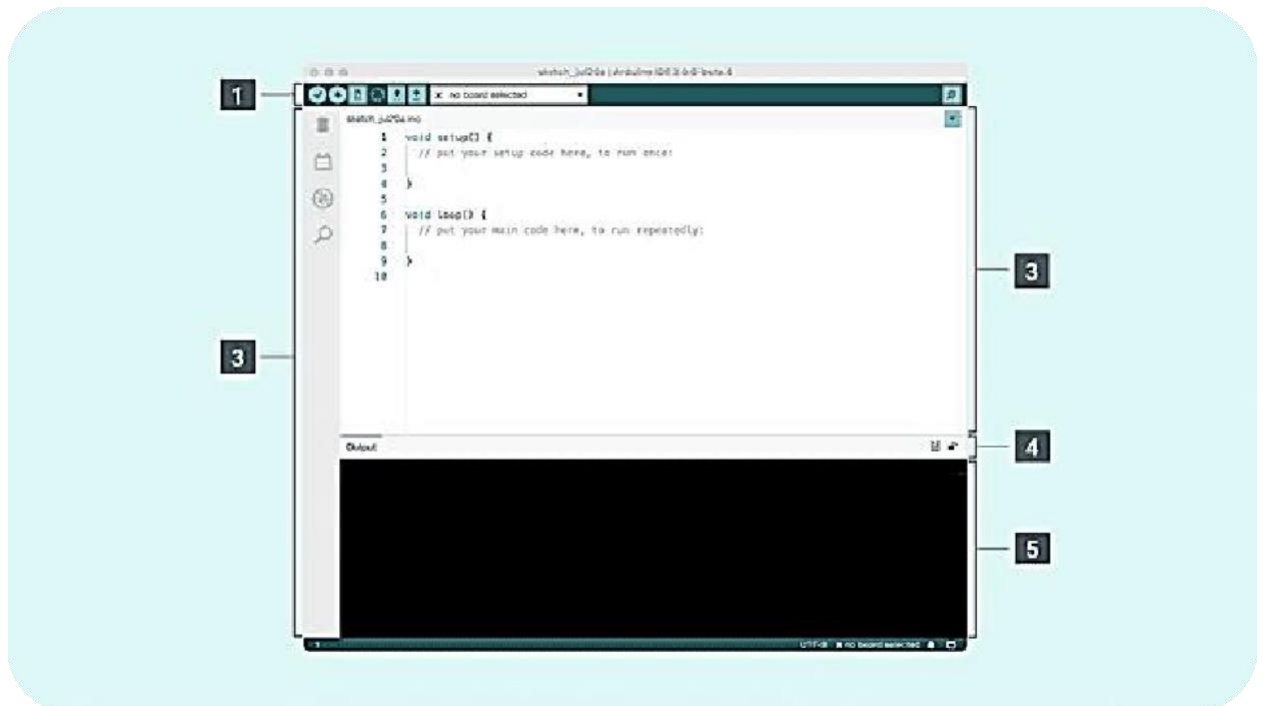
Using the Offline IDE 2.x

The editor consists of four main areas:

1. A toolbar containing buttons for common functions and a set of menus. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, select your board and port, and open the serial monitor.
2. A sidebar for regularly used tools. It provides quick access to board managers, libraries, board debugging, and a search and replace tool.
3. A text editor for writing your code.
4. Console controls provide control over the output in the console.
5. A text console displays the text output generated by the Arduino Software (IDE), including full error messages and other information. The configured board and serial port are displayed in the bottom right corner of the window.



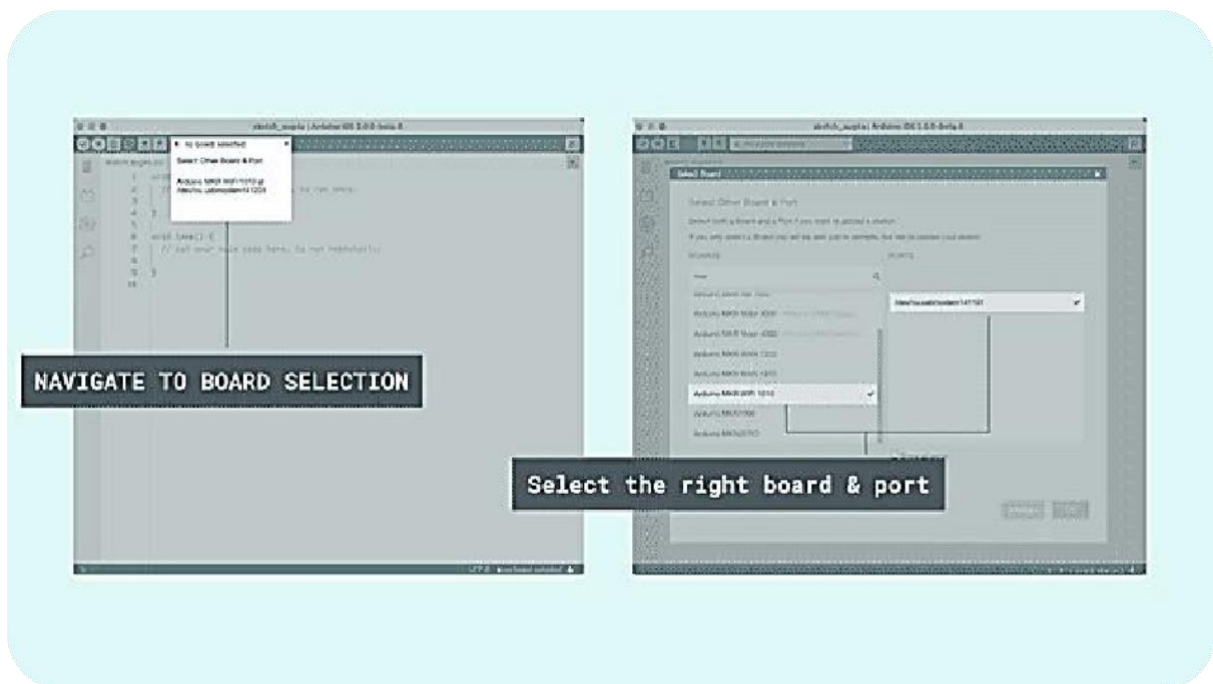
Avrupa Birliği tarafından
ortak finanse edilmektedir



Arduino Software IDE

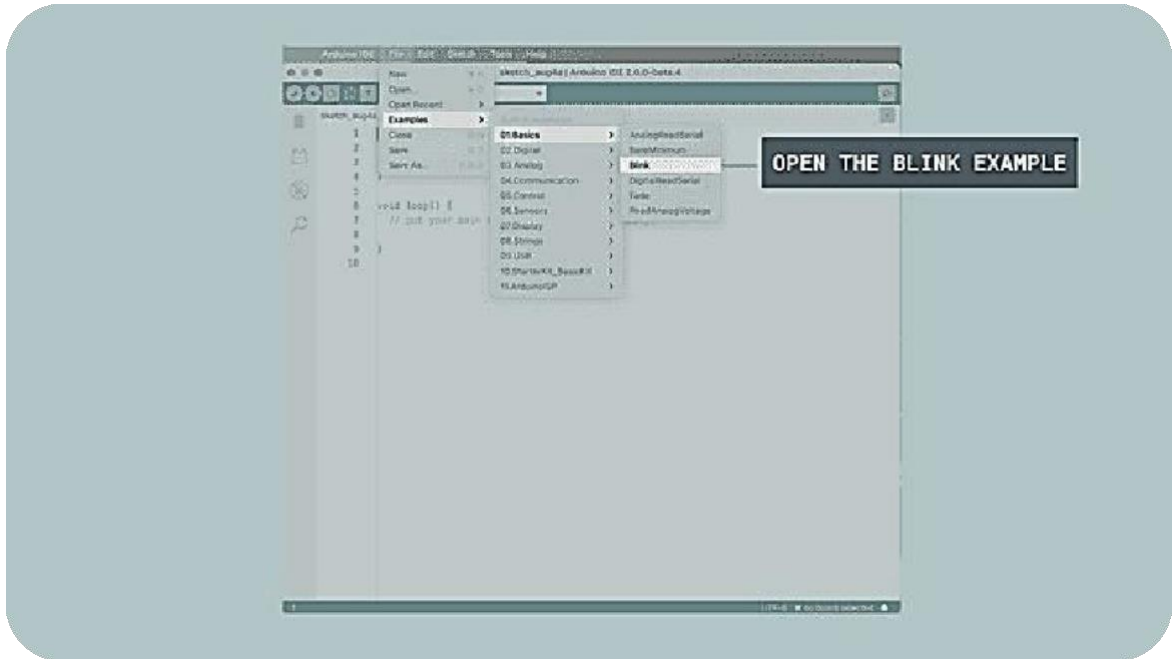
Now that everything is ready, let's try to make your board blink!

1. Connect your Arduino or Genuino board to your computer.
2. Now you need to select the correct board and port. This is done from the toolbar. Make sure to select the board you are using. If you can't find your board, you can add it from the board manager in the sidebar.



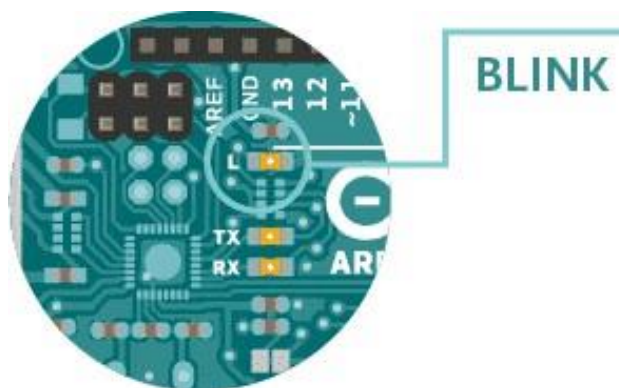
A board and port selection

3. Let's try an example: Go to File > Examples > 01.Basics > Blink.



Opening an example

4. To upload this to your board, simply click the arrow in the upper left corner. This process takes a few seconds, and it's important not to disconnect the board during this time. If the upload is successful, you will see a 'Done uploading' message in the lower output area.
5. Once the upload is complete, you should see the yellow LED with the letter L next to it start to blink on your board. You can adjust the blinking speed by changing the delay number in parentheses to 100 and re-uploading the Blink sketch. The LED should now blink much faster.



Congratulations! You have successfully programmed your board to blink the built-in LED!

In addition to this, there is a free online website called Tinkercad, where we can easily code and simulate Arduino or electronic circuits.

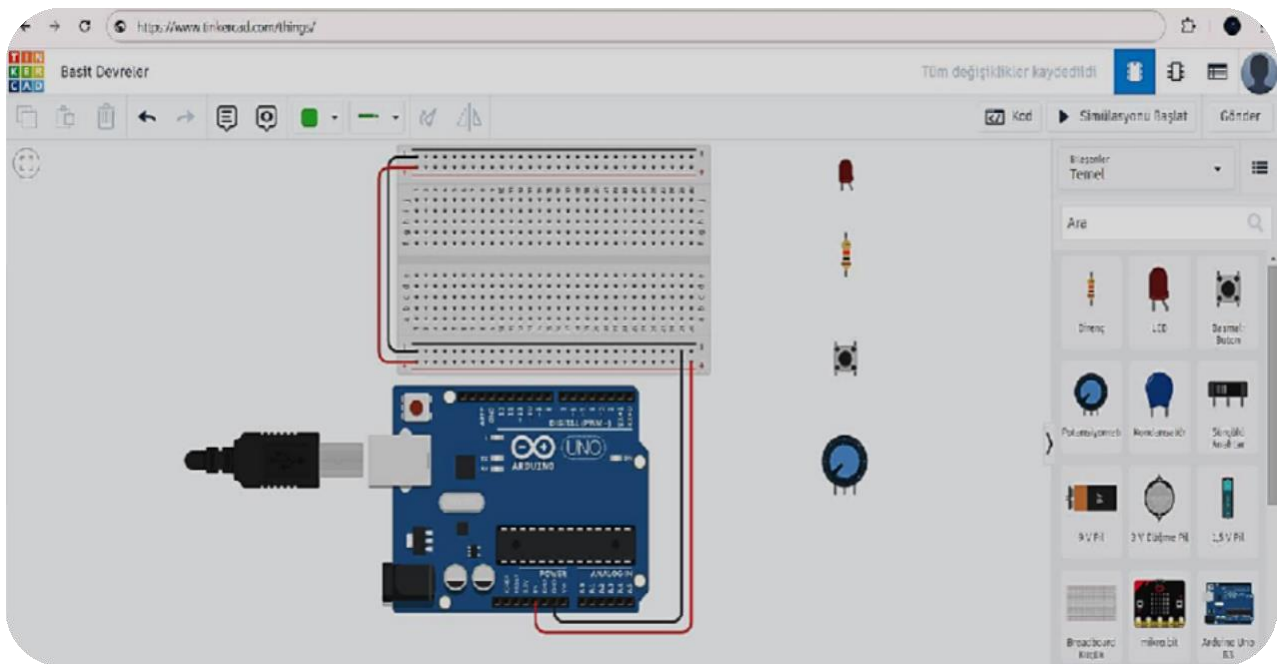


Avrupa Birliği tarafından
ortak finanse edilmektedir

Tinkercad

Tinkercad is a free online educational application by Autodesk that allows us to do 3D design, electronic circuits, and coding. There is no need to download a program to use Tinkercad, as it works in the cloud, enabling us to access and use our projects from anywhere with an internet connection, such as a PC.

The Tinkercad Circuits section is a simple yet visually impressive online simulation program that can be used to demonstrate basic electronic circuit setups. In this section, fundamental electronics and Arduino applications can be conducted using simulations. This way, we can see how robotic applications work without using any physical circuit components or boards. Additionally, it helps prevent potential errors, thereby avoiding financial losses.



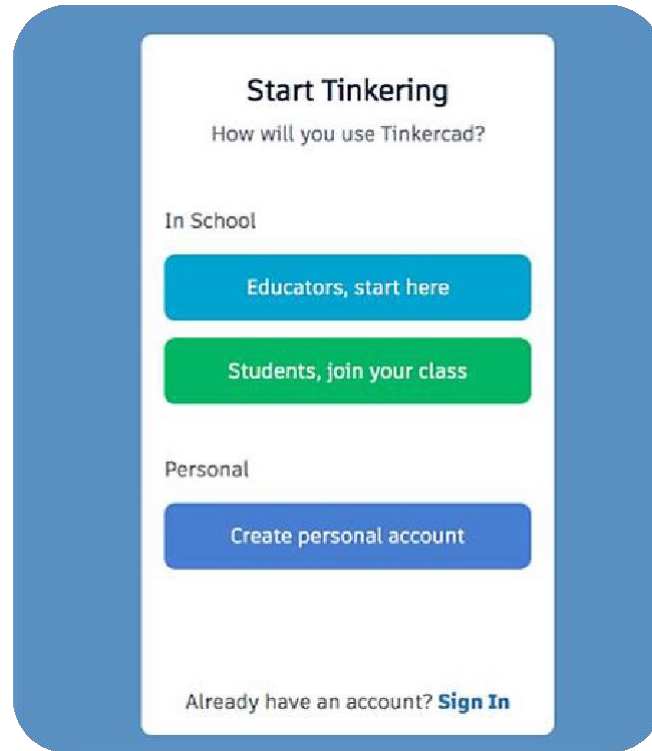
Tinkercad Circuits

Signing Up and Logging into Tinkercad

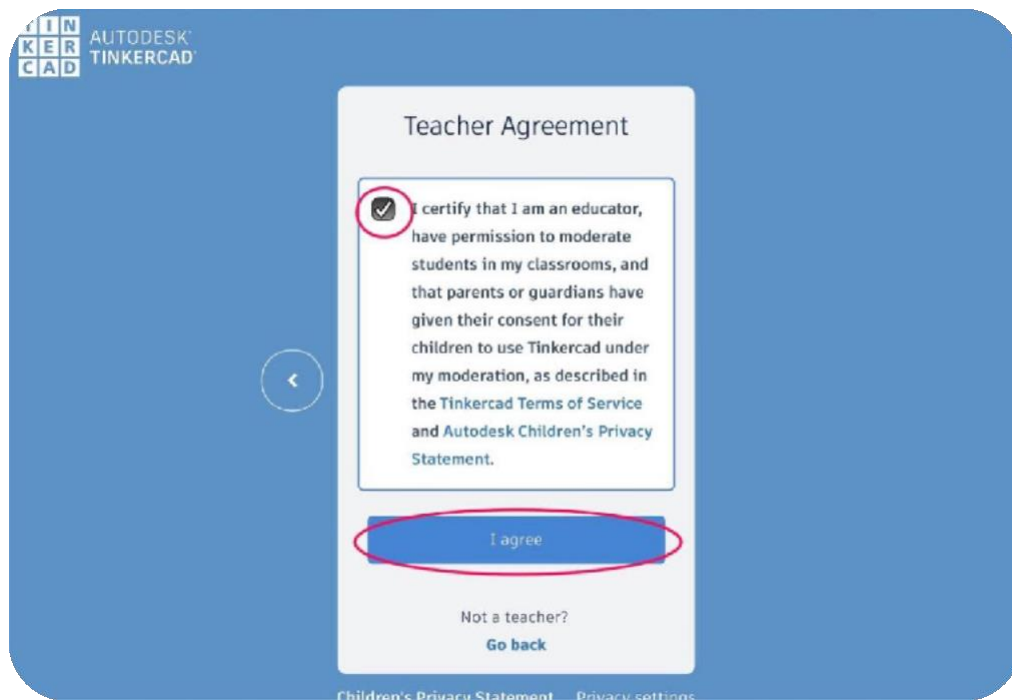
To create designs or simulations with Circuits on Tinkercad, you need to register on the platform. After entering the website at <https://www.tinkercad.com>, you can join the system by selecting the 'Join Now' option.

After selecting 'Join Now,' three options will appear on the screen, as shown below.



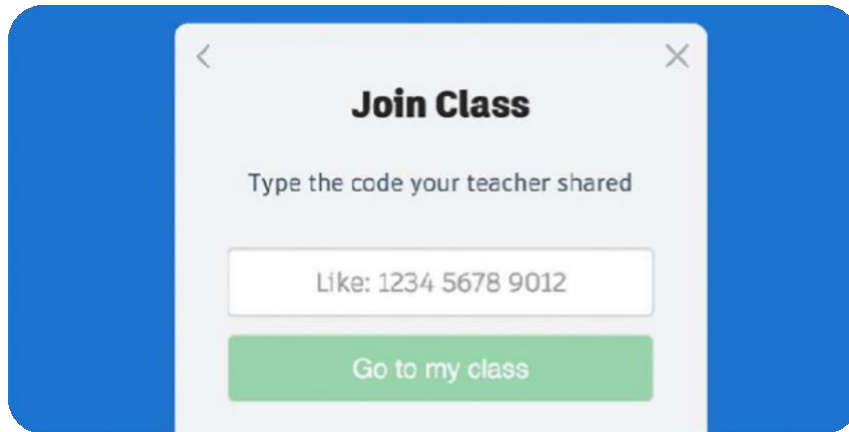


Educators, start here: With this option, you can create a teacher account by accepting the Teacher Agreement when registering for the system.



Classes can be defined through the account created by the teacher, allowing students to log in using a code without registering in the system.

Students can join a Class: With this option, students can log in without registering in the system by using the class codes created for them with the teacher's account.

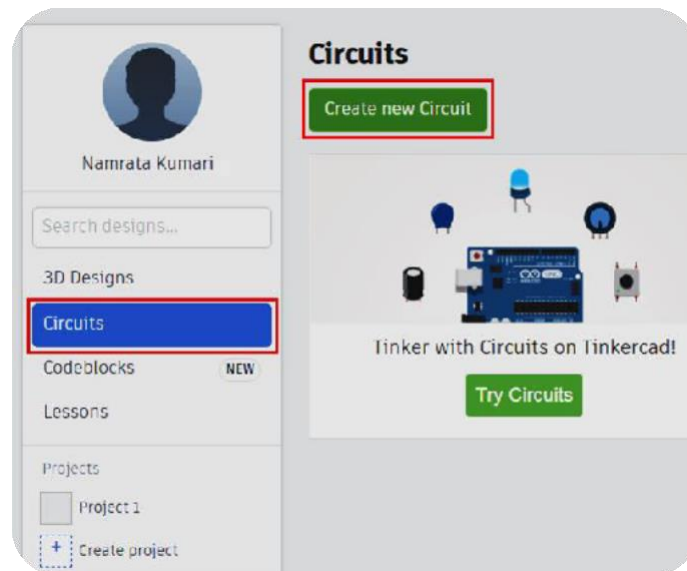


Create a Personal Account: This option allows you to create a standard account on the system. If a class account will not be created, this option can be selected.

When you want to log in to the system from a different device or at a different time, you can use the "Log In" option on the homepage to access the system with your existing account.









After logging into the system, the Tinkercad application can be used. To create electronic simulations using the Circuits feature, click the +Create button on the left side of the opened page and select the Circuits option.







Creating a New Circuit

After selecting the Circuits option, you will reach the simulation page to create new circuits. The sections on the simulation screen are as follows.

	Rotates the selected component 90 degrees.
	Deletes the selected component.
	Undoes the last action.
	Redoes the undone action.
	Adds a comment to the simulation.
	Hides or shows the comments on the simulation.

Tinkercad shortcuts

	Kod	The code writing tab opens.
	Simülasyonu Başlat	The circuit is run.
	Dışa Aktar	The completed work is downloaded to the computer.
	Paylaş	Allows the simulation to be shared with other users.

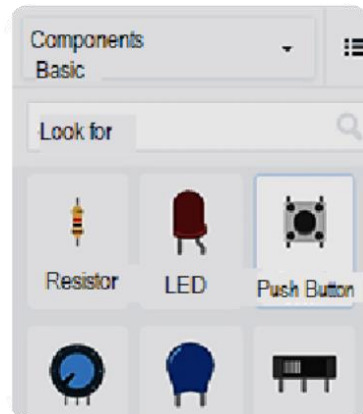
Simulation tools

To name the circuit page, you can click on the text next to the Tinkercad icon in the upper left corner of the page to change the project name.



Adding Components

To add components to the design screen, you can drag the desired component from the menu on the right and position it in the center of the screen. Basic components are displayed when the page is opened.



Avrupa Birliği tarafından
ortak finanse edilmektedir

When a component is dragged onto the scene, it becomes usable, and it can also be named using the box in the upper right corner.

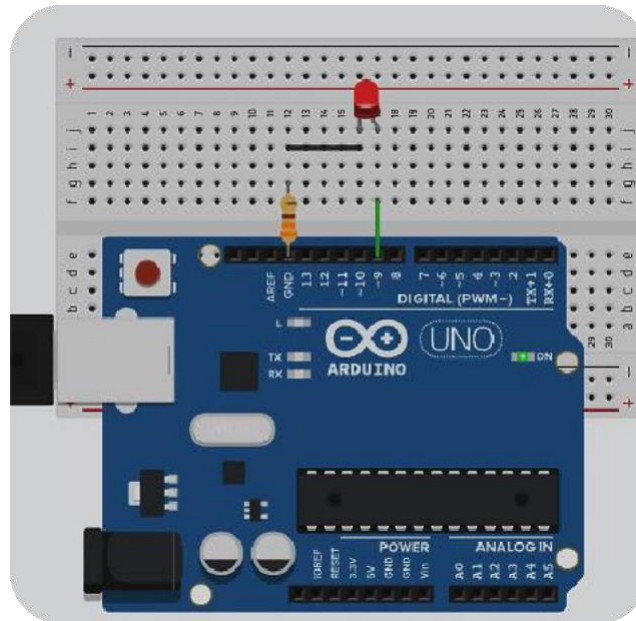


Connecting Components

When designing real electronic circuits, components can be connected using soldering in everyday life, or they can be connected to each other using jumper wires without the need for soldering on a breadboard. In the simulation, breadboards and connection wires should be used to establish connections. To connect components, you can create a wire by using the left mouse button on the component's leg, and the created wire can be positioned by moving the mouse to the desired location.

Creating Circuit Connections

When creating a circuit as shown, the shorter leg of the LED should be connected to the GND pin of the Arduino through a 330 ohm resistor, while the longer leg can be connected to any of the Arduino's digital pins. In this case, the coding should be done according to the pin to which the longer leg of the LED is connected.

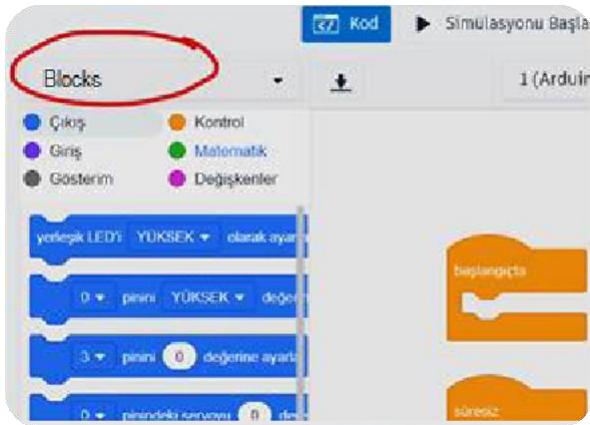


Creating Code Blocks

After establishing the circuit connections, it is necessary to create the corresponding code. The Arduino board will operate based on the written code. Therefore, the code must be created according to the established connections. To create code blocks, simply click on the Code tab on the page. After this action, the code blocks panel will open. Additionally, coding can also be done in text format instead of using blocks. To do this, after opening the code section, select the Text option instead of Blocks, as shown in the figure below.



Avrupa Birliği tarafından
ortak finanse edilmektedir



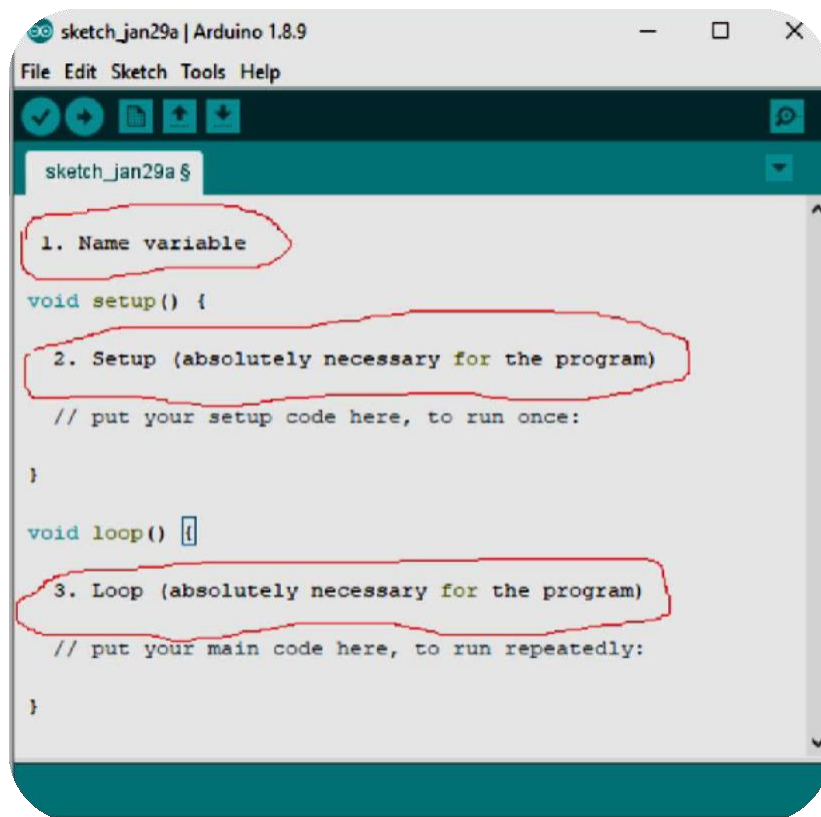
Programming with Blocks



Programming with Text

Introduction to Arduino Programming

With the development environment, Arduino programs can be written, compiled, and uploaded to the boards. Arduino programming uses a language structure based on C/C++/Java. Thanks to libraries, there is no need to delve into hardware levels, and there is case sensitivity in code writing. An Arduino program is called a "sketch." A sketch can be divided into three sections.



1. Name Variable (Variable Definition - Optional):

In the first section, the elements of the program are named. This part is optional.



Avrupa Birliği tarafından
ortak finanse edilmektedir

2. Setup (Setup - Absolutely Necessary for the Program):

The code in the setup section is executed only once. Input and output pins must be defined.

Output Pins: The pin must provide voltage. For example: turning on an LED.

Input Pins: The board must read voltage. For example: activating a switch.

3. Loop (Loop - Absolutely Necessary for the Program):

This loop section will be continuously repeated by the board. The project runs, and when it reaches the end, it starts over and continues like this.

Some Rules Related to Coding

- Commands can be written on the same line or one below the other. However, for the sake of program readability, it is beneficial to write them on separate lines.
- A semicolon (;) is placed at the end of commands.
- If there are libraries to be used, they are included in the program at the beginning with the #include command.
- Turkish characters should not be used. However, they can be used in comment lines (since they are not included in the compilation process).
- There is case sensitivity in code writing.

Semicolon “;”

The semicolon is used to terminate a command. Forgetting to end a command with a semicolon will result in a compiler error.

Example: `int a = 13;`

{ } (Curly Braces):

The main uses of curly braces are for functions, loops, and conditional statements. An opening curly brace { must always be followed by a closing curly brace }.

Example:

```
void setup () { .....  
.....  
}
```



Avrupa Birliği tarafından
ortak finanse edilmektedir

“//” Symbol:

The “//” symbol can be written anywhere in the program and the line where it is written is considered a comment line. Comment lines serve as reminders or notes while writing the program. The compiler ignores the line with the “//” symbol.

Another method used for comments is the “/” and “/” symbols. After placing the “/” symbol in the desired location within the program, necessary notes or explanations can be written. The comment section is then concluded with the “/” symbol. The compiler does not consider any text written between these markers; they are solely for the user’s reference. This method is preferred for writing multiple comment lines.

Example:

```
// This command supplies 5V power to a pin.  
/* This command supplies 5V power to a pin. Then, LOW sends 0V. */
```

Arduino Applications and Commands

Below are Arduino applications along with their respective command lines. This demonstrates the definitions of the codes, how to use them, and their effects on the circuit.

• void setup()

This function is called when a sketch begins. It is used to initialize variables, pin modes, start using libraries, etc. The setup() function will run only once each time the Arduino board is powered on or reset.

Example Code:

```
1 int buttonPin = 3;  
2  
3 void setup() {  
4     Serial.begin(9600);  
5     pinMode(buttonPin, INPUT);  
6 }  
7  
8 void loop() {  
9     // ...  
10 }
```

void loop()

The loop() function does exactly what its name suggests; it repeatedly loops, allowing your program to change and respond.



Avrupa Birliği tarafından
ortak finanse edilmektedir

Example Code:

```
1 int buttonPin = 3;
2
3 // setup initializes serial and the button pin
4 void setup() {
5     Serial.begin(9600);
6     pinMode(buttonPin, INPUT);
7 }
8
9 // loop checks the button pin each time,
10 // and will send serial if it is pressed
11 void loop() {
12     if (digitalRead(buttonPin) == HIGH) {
13         Serial.write('H');
14     }
15     else {
16         Serial.write('L');
17     }
18
19     delay(1000);
20 }
```

digitalWrite:

This command allows us to provide either 0 V or 5 V (or 3.3 V) to our pins. For example, if a pin is set as OUTPUT using `pinMode()`, we can control the voltage as follows:

- `digitalWrite(pinNo, HIGH)` ile pine 5V verilir.
- `digitalWrite(pinNo, LOW)` ile pine 0V verilir.
- `digitalWrite(LED, HIGH)` // powers the pin associated with the variable `LED`.
- `digitalWrite(motor, LOW)` // stops the motor associated with the variable `motor`.

digitalRead:

- This command allows us to read incoming voltage values between 0 and 5 V.
- In digital terms, 5 V is equal to 1, while 0 V is equal to 0.

Usage: - `digitalRead(variable);`

-`DigitalRead(bluetooth);` //Reads the values coming from the Bluetooth module.

-`DigitalRead(potansiyometre);` // Reads the values coming from the potentiometer.



Avrupa Birliği tarafından
ortak finanse edilmektedir

analogWrite:

- This command allows us to provide values between 0 and 5 V. The incoming and outgoing electric current ranges from 0 to 255.
- The advantage of this command is that it enables us to adjust the speed of certain components or control their position.

For example, it can be used to: Drive a motor at slow, medium, or high speeds, Adjust the position of a servo motor.

Usage: AnalogWrite(variable, LOW)

```
-AnalogWrite(motor,255) // We provided 5V to the motor.
```

```
-AnalogWrite(motor,204) // We provided 4V to the motor.
```

AnalogRead:

– This command allows us to read values between 0-5V.

- **Usage:** - analogRead(variable);

```
-analogRead(blueetooth); //Reads data coming from Bluetooth.
```

```
-analogRead(potansiyometre); // Reads data coming from the potentiometer.
```

pinMode:

The pinMode() function is used to configure a specific pin to behave as either an input or an output.

Usage: pinMode(pin, mode)

Parameters:

- **pin:** Arduino pin number.
- **mode:** INPUT, OUTPUT, or INPUT_PULLUP.

```
pinMode(3, OUTPUT); // Sets digital pin 3 as an output
```

```
pinMode(8, INPUT); // Sets digital pin 8 as an input
```

delay(millisecons):

This function causes the program to wait for a specified amount of time. By entering 1000 in the parentheses, it provides a delay of 1000 milliseconds, which equals 1 second.

Example: delay(1000);

Serial.begin(baud_rate):

This command initializes serial communication. You can send or receive text, information, or



Avrupa Birliği tarafından
ortak finanse edilmektedir

numbers via this serial communication. The typical baud rate is 9600.

Usage: Serial.begin(baud_rate);

```
-Serial.begin(9600); // Starts serial communication at 9600 baud rate
```

Serial.print():

This command allows you to print data received from sensors to the screen.

Usage: Serial.print(variable);

```
-Serial.print("Hello World");
```

```
-Serial.print(potansiyometre); // Prints potentiometer data to the screen in-line
```

```
-Serial.println(potansiyometre); // Prints potentiometer data to the screen on a new line
```

❖ for :

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is typically used to increase and terminate the loop. The for statement is useful for any repetitive task and is often used in conjunction with arrays to work on data/pin collections.

Usage: for (initial value; condition; increment) {

```
// statements to execute;
```

```
}
```

Example:

```
for(x=0;x<100;x=x+2)
```

```
{
```

```
Serial.println(x); //Even numbers less than 100 will be printed on the screen.
```

```
}
```

❖ If - Else Statement

The if statement checks a condition, and if the condition is "true," it executes the statement or block of statements associated with it. If the condition is "false," the else statement is executed.

Usage:



Avrupa Birliği tarafından
ortak finanse edilmektedir

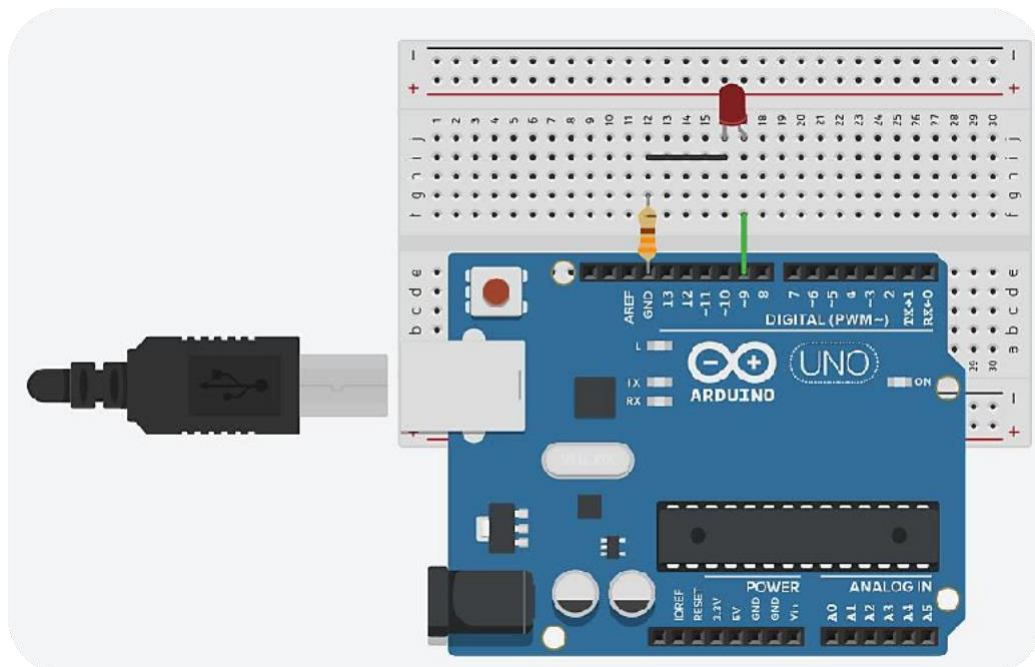

```
if (condition) {  
  // actions to be performed if the condition is true  
}  
condition: a boolean expression (i.e., it can be true or false).  
Else {  
  // actions to be performed if the condition is false  
}
```

Note: The `if` statement can be used alone without the `else`.

Arduino Applications

❖ *Application 1: Digital Output (LED Blinking)*

Using Arduino Uno to turn on an LED for 1 second and then turn it off for 1 second.



Materials: 1 x Arduino Uno R3, 1 x Kırmızı LED ,1 x 330 Ω direnç, Breadbord

Connect the negative (-) leg of the LED to the GND pin on the Arduino, and connect the positive (+) leg of the LED through a resistor to digital pin 9 on the Arduino. The purpose of



Avrupa Birliği tarafından
ortak finanse edilmektedir

the resistor is to limit the current flowing through the LED. Without the resistor, the LED is likely to burn out.

Code Example:

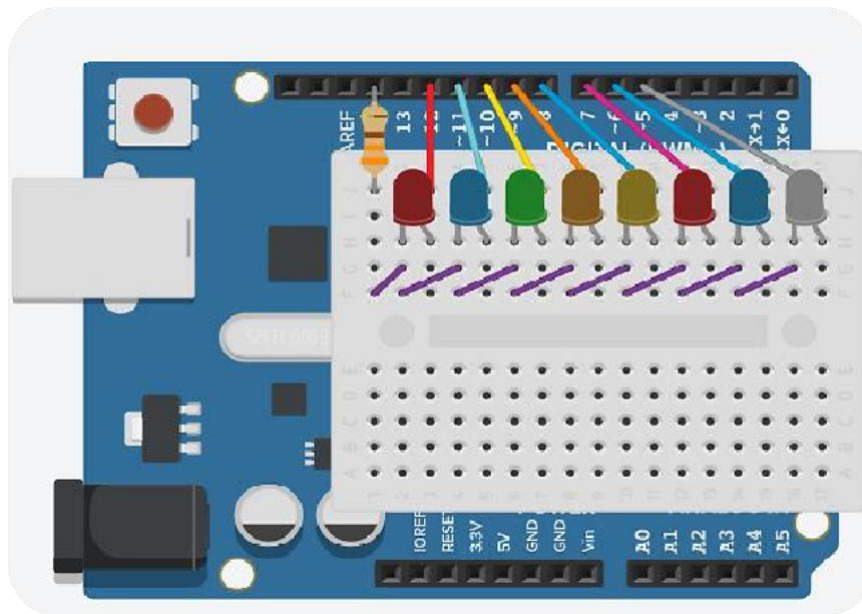
```
void setup()
{
  pinMode(9, OUTPUT); // Set pin 9 as an output
}

void loop()
{
  digitalWrite(9, HIGH); // Send 5V to pin 9 (LED ON)
  delay(1000); // Wait for 1000 milliseconds (1 second)

  digitalWrite(9, LOW); // Send 0V to pin 9 (LED OFF)
  delay(1000); // Wait for 1000 milliseconds (1 second)
}
```

❖ Application 2

Lighting 8 LEDs in sequence with Arduino and turning them off all at once. This process continues indefinitely.



Materials: 1 x Arduino Uno R3, 8 x LED ,1 x 330 Ω direnç, Breadbord

The negative (–) terminals of the LEDs are connected together and linked to the GND pin through a resistor, while the positive terminals of the LEDs are connected to the digital pins 12, 11, 10, 9, 8, 7, 6, and 5 of the Arduino in sequence.

Code Example:

```
void setup()
{
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
}
void loop()
{
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);
  digitalWrite(11, LOW);
  digitalWrite(12, LOW);
  delay(1000);

  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(6, HIGH);
  delay(1000);
  digitalWrite(7, HIGH);
  delay(1000);
  digitalWrite(8, HIGH);
  delay(1000);
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(10, HIGH);
  delay(1000);
  digitalWrite(11, HIGH);
  delay(1000);
  digitalWrite(12, HIGH);
  delay(1000);
}
```



Avrupa Birliği tarafından
ortak finanse edilmektedir

❖ *Application 3*

The program codes in Application 2 can be written much shorter by using loops. In particular, for applications where the number of repetitions is known, a for loop can be preferred. By using the for loop, the program in Application 2 can be shortened as follows.

Code Example:

```
int i = 0;

void setup()
{
  for(i = 5;i<=12;i++){
    pinMode(i,OUTPUT);
  }
}

void loop()
{
  for(i = 5;i<=12;i++){
    digitalWrite(i,LOW);
  }
  delay(1000);

  for(i = 5;i<=12;i++){
    digitalWrite(i,HIGH);
    delay(1000);
  }
}
```

❖ *Application 4*

In the circuit of Application 2, the LEDs will light up in sequence until the end, and then they will turn off in sequence from the end back to the beginning.

Code Example:

```
int i = 0;

void setup()
{
  for(i = 5;i<=12;i++){
    pinMode(i,OUTPUT);
  }
}

void loop()
{
  for(i = 5;i<=12;i++)
```



Avrupa Birliği tarafından
ortak finanse edilmektedir

```

{
  digitalWrite(i,HIGH);
  delay(500);
}
for(i = 12;i>=5;i--)
{
  digitalWrite(i,LOW);
  delay(500);
}
}

```

❖ *Application 5*

In the circuit of Application 2, the LEDs will light up in sequence from the outside to the inside and then turn off in sequence from the inside to the outside.

Code Example:

```

int i = 0;

void setup()
{
  for(i = 5;i<=12;i++){
    pinMode(i,OUTPUT);
  }
}

void loop()
{
  for(i = 0;i<4;i++)
  {
    digitalWrite(5+i,HIGH);
    digitalWrite(12-i,HIGH);
    delay(500);
  }
  for(i = 3;i>=0;i--)
  {
    digitalWrite(5+i,LOW);
    digitalWrite(12-i,LOW);
    delay(500);
  }
}

```

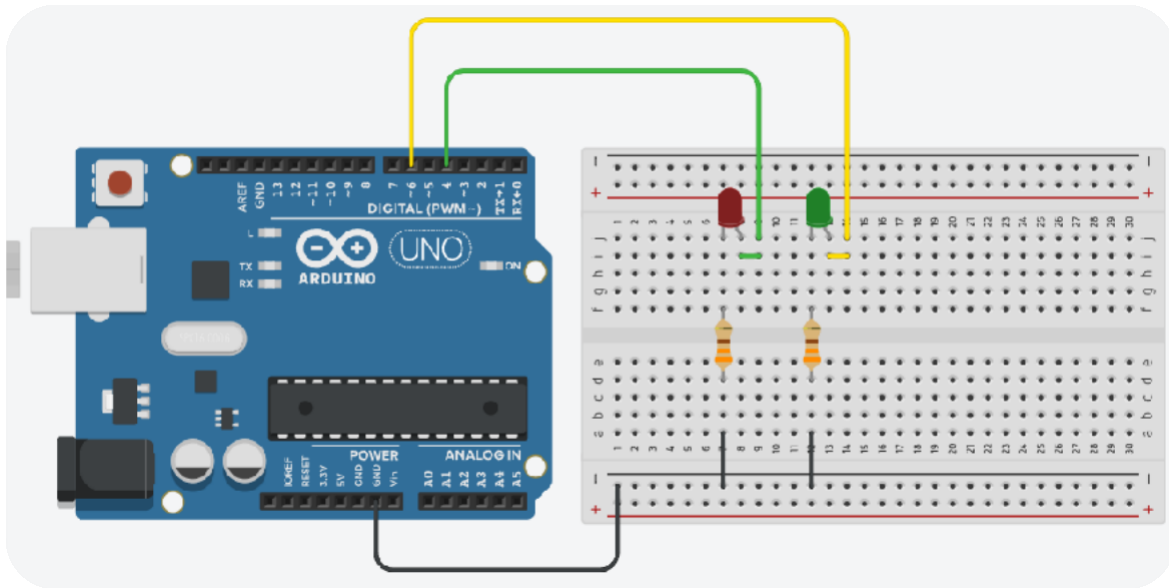
❖ *Application 6*

Flip-Flop:

Lighting and turning off 2 LEDs in sequence with Arduino. (The LEDs will not be on or off at the same time.) The delay will be 1 second.



Avrupa Birliği tarafından
ortak finanse edilmektedir



Materials: 1x Arduino Uno R3, 1x Red LED, 1x Green LED, 2x 330 Ω resistors, Breadboard

Code Example:

```
/* Flip-Flop */

void setup() {

  pinMode(4, OUTPUT); // Red LED pin set as digital output
  pinMode(6, OUTPUT); // Green LED pin set as digital output
}

void loop(){

  digitalWrite(4, HIGH); // Turn on the red LED
  digitalWrite(6, LOW); // Turn off the green LED
  delay(1000); // Wait for 1000 milliseconds

  digitalWrite(4, LOW); // Turn off the red LED
  digitalWrite(6, HIGH); // Turn on the green LED
  delay(1000); // Wait for 1000 milliseconds
}
```

❖ Application 7

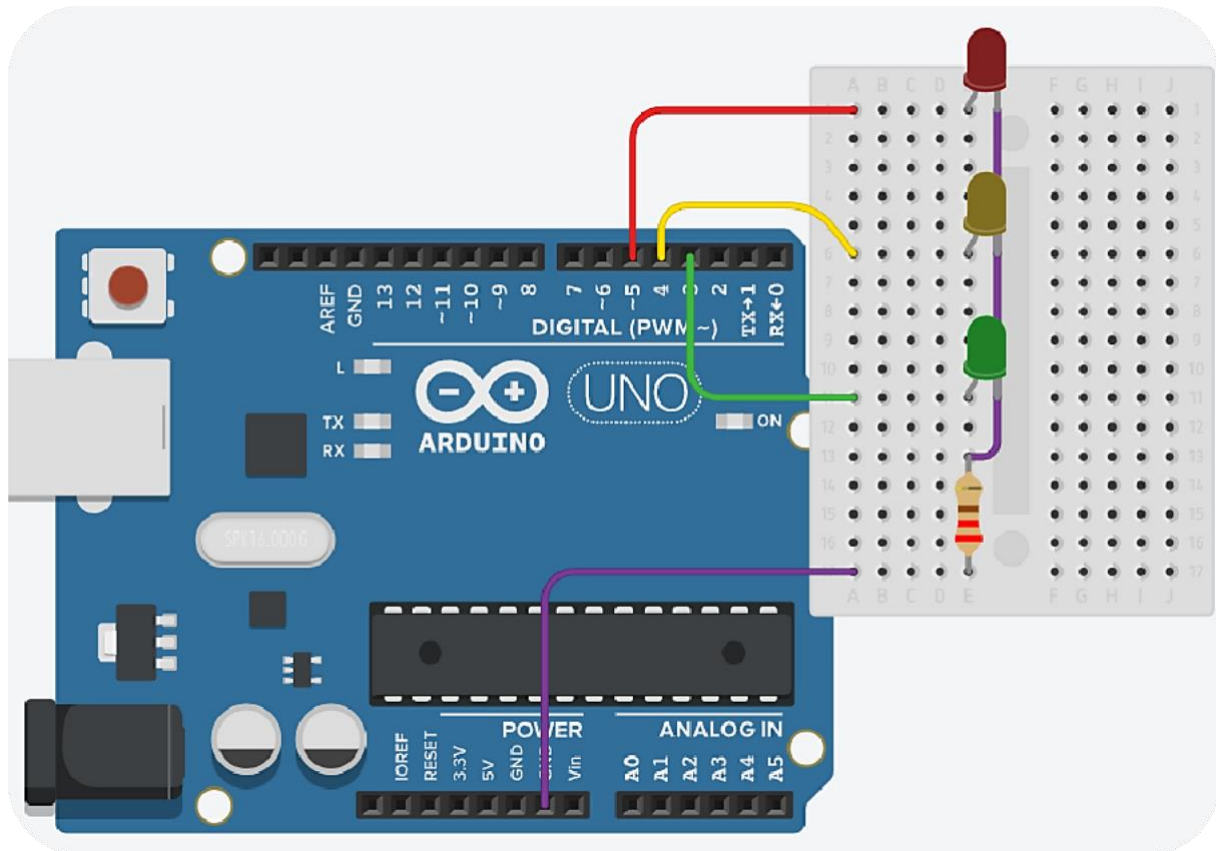
Traffic Light:

Circuit Description: In this project, we will set up a traffic light system. There are 3 LEDs in different colors (green, yellow, and red).



Avrupa Birliği tarafından
ortak finanse edilmektedir

This code simulates the functioning of a traffic light system. The red LED stays on for 8 seconds, then both red and yellow LEDs light up for 2 seconds, the green LED lights up for 5 seconds, and finally, the yellow LED lights up for 2 seconds. The program then repeats itself.



Materials: 1x Arduino Uno R3, 1x Red LED, 1x Green LED, 1x Yellow, LED 1x 220 Ω resistors, Breadboard mini

Code Example:

```
/* Traffic Light*/

int kirmizi_LED = 5;
int sari_LED = 4;
int yesil_LED = 3;

void setup() { // Set the pins as output
  pinMode(kirmizi_LED, OUTPUT);
  pinMode(sari_LED, OUTPUT);
  pinMode(yesil_LED, OUTPUT);
}

void loop() {
  digitalWrite(kirmizi_LED, HIGH); // Turn on the red LED for 8 seconds
  digitalWrite(sari_LED, LOW);
  digitalWrite(yesil_LED, LOW);
```

```
    delay(8000);

    digitalWrite(kirmizi_LED, HIGH); //Turn on both red and yellow LEDs for 2
seconds
    digitalWrite(sari_LED, HIGH);
    digitalWrite(yesil_LED, LOW);
    delay(2000);

    digitalWrite(kirmizi_LED, LOW); // Turn on the green LED for 5 seconds
    digitalWrite(sari_LED, LOW);
    digitalWrite(yesil_LED, HIGH);
    delay(5000);

    digitalWrite(kirmizi_LED, LOW); // Turn on the yellow LED for 2 seconds
    digitalWrite(sari_LED, HIGH);
    digitalWrite(yesil_LED, LOW);
    delay(2000);
}
```

❖ *Application 8*

RGB LED Application:

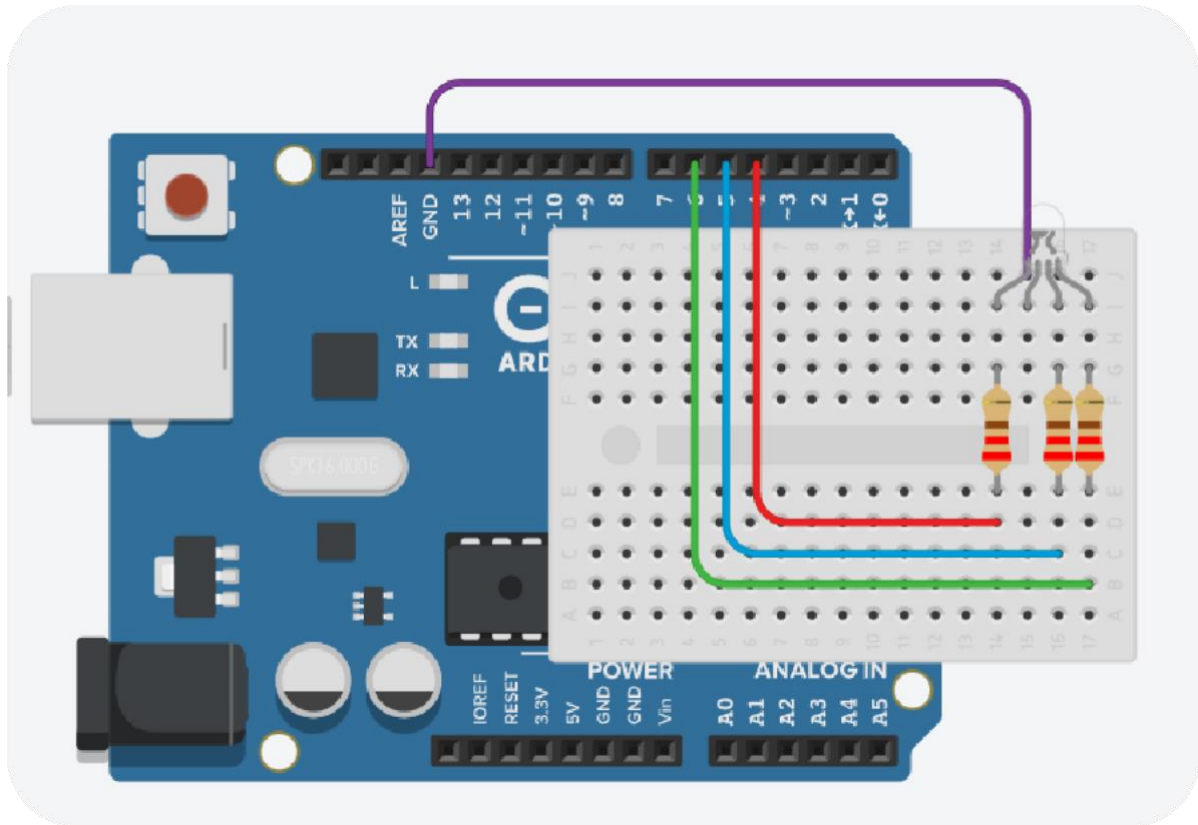
An RGB LED is a single package containing three LEDs that are commonly connected. It is possible to digitally control the light intensity of these three colors. Additionally, using PWM (Pulse Width Modulation) technique, desired colors can be achieved.

We will turn on and off each color of the RGB LED at 1-second intervals. To achieve white light, all LEDs need to be energized simultaneously.

Materials: 1x Arduino Uno R3, 1x RGB LED, 3x 220 Ω resistors, Breadboard mini



Avrupa Birliği tarafından
ortak finanse edilmektedir



Code Example:

```
const int Kirmizi_Led=4; // Assigned the value 4 to Red_Led
const int Mavi_Led=5; // Assigned the value 5 to Blue_Led
const int Yesil_Led=6; // Assigned the value 6 to Green_Led

void setup()
{
  pinMode(Mavi_Led,OUTPUT);
  pinMode(Yesil_Led,OUTPUT);
  pinMode(Kirmizi_Led,OUTPUT);
}

void loop()
{
  digitalWrite(Mavi_Led, LOW); // Red_Led on, others off
  digitalWrite(Yesil_Led, LOW);
  digitalWrite(Kirmizi_Led, HIGH);
  delay(1000);

  digitalWrite(Mavi_Led, LOW ); // Green_Led on
  digitalWrite(Yesil_Led, HIGH);
  digitalWrite(Kirmizi_Led, LOW );
  delay(1000);

  digitalWrite(Mavi_Led, HIGH); // Blue_Led on
```

```

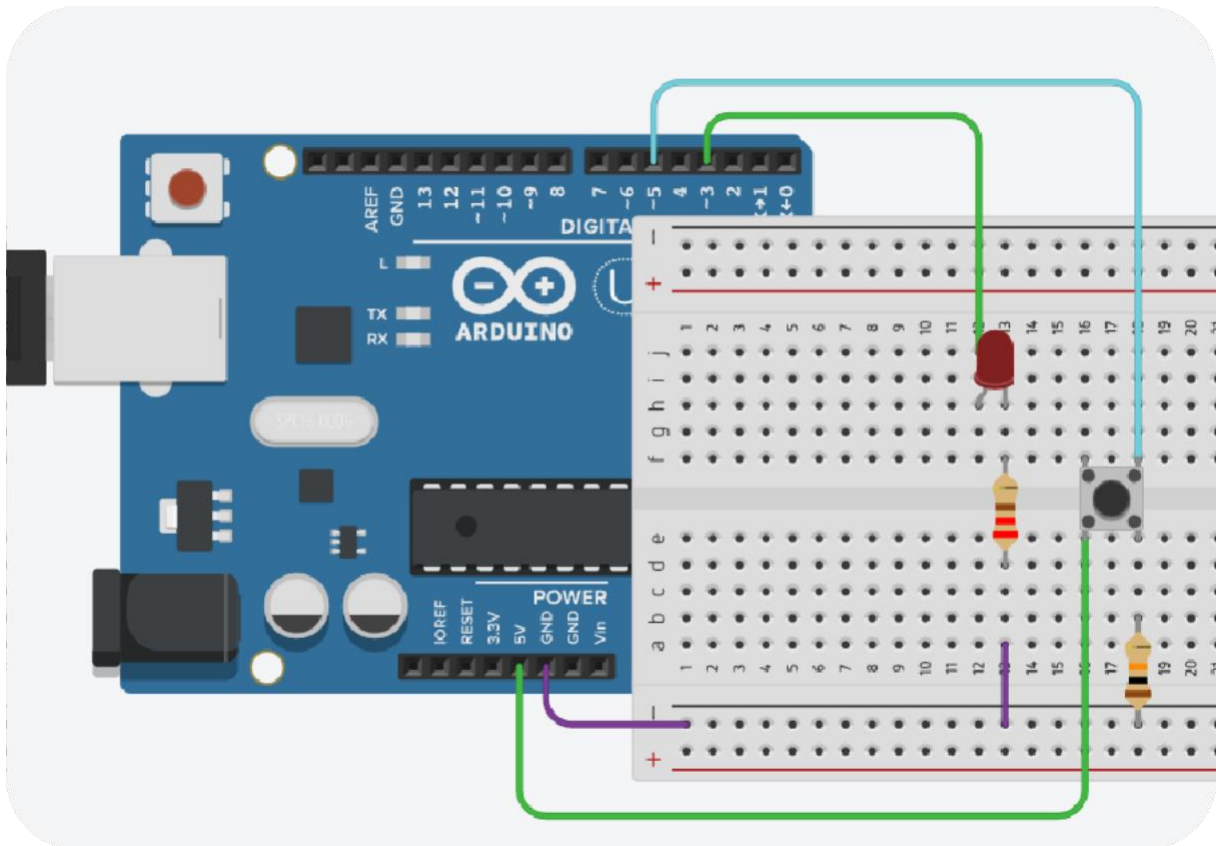
digitalWrite(Yesil_Led, LOW);
digitalWrite(Kirmizi_Led, LOW);
delay(1000);

// Turning on all three LEDs to create white light
digitalWrite(Mavi_Led, HIGH);
digitalWrite(Yesil_Led, HIGH);
digitalWrite(Kirmizi_Led, HIGH);
delay(1000);
}

```

❖ Application 9

When we press a button, the LED will light up, and when we release the button, the LED will turn off. The negative (–) leg of the LED is connected to the GND pin of the Arduino board through a resistor, and the positive (+) leg is connected to pin 3. The button output is connected to pin 5, and the 5V and GND pins are connected to the button. This ensures that when the button is not pressed, pin 5 receives GND.



Materials: 1x Arduino Uno R3, 1x LED, 1x 220 Ω resistor, 1x 10 k Ω resistor, 1x button, Breadboard.

Code Example:

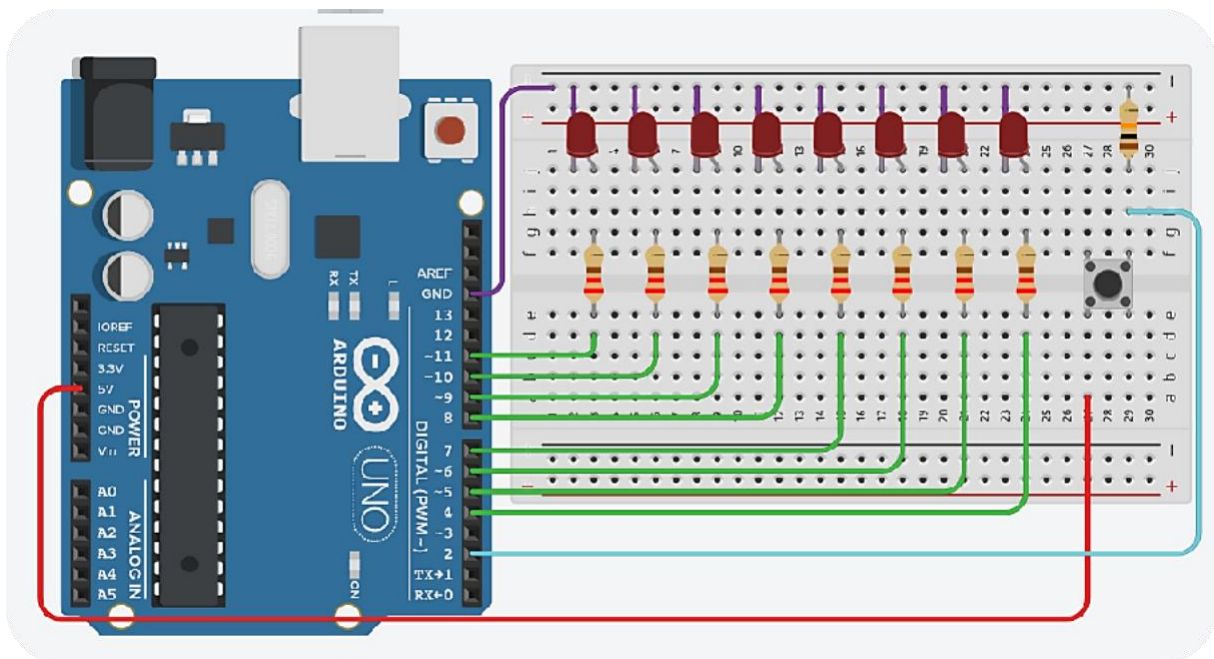
```
int buton=5;
int Led=3;

void setup()
{
  pinMode(Led, OUTPUT);
  pinMode(buton, INPUT);
}

void loop()
{
  if (digitalRead(5)==HIGH){
    digitalWrite(Led,HIGH);
  }
  else {digitalWrite(Led,LOW);
  }
}
```

❖ Application 10

A circuit and code that lights up 8 LEDs sequentially each time the button connected to the Arduino is pressed.



Materials: 1x Arduino Uno R3, 8x LEDs, 8x 220 Ω resistors, 1x 10 k Ω resistor, Button, Breadboard

The negative (-) terminals of the LEDs are connected together and connected to the GND pin. The positive terminals of the LEDs are connected through resistors to the Arduino's digital pins in the order of 11-10-9-8-7-6-5-4. The button output is connected to pin 2, and the 5V and GND pins are also connected to the button.

Code Example-1:

```
int led = 3;

void setup(){
  pinMode(2, INPUT);
  for(int i = 4;i<=11;i++)
    pinMode(i, OUTPUT);
}

void loop(){
  if(digitalRead(2)==1) {
    led++;
    delay(50); //To prevent any vibrations that may occur with the
               button.
  }
  digitalWrite(led-1,LOW);
  digitalWrite(led,HIGH);

  if(led == 12) led = 4;
}
```

Note: In button applications, adding a wait time after pressing the button can prevent any vibrations from occurring. Another way to prevent vibrations is to wait for the button to be released after it has been pressed before performing any action. This can be implemented using a loop. The following code demonstrates this.

Code Example-2:

```
int led = 3;

void setup() {
  pinMode(2, INPUT);
  for(int i = 4;i<=11;i++)
    pinMode(i, OUTPUT);
}

void loop() {
  if(digitalRead(2)==1)
  {
```



Avrupa Birliği tarafından
ortak finanse edilmektedir


```

while(digitalRead(2)==1); /* This command makes the program wait
                           at this line until the button is
                           released. */

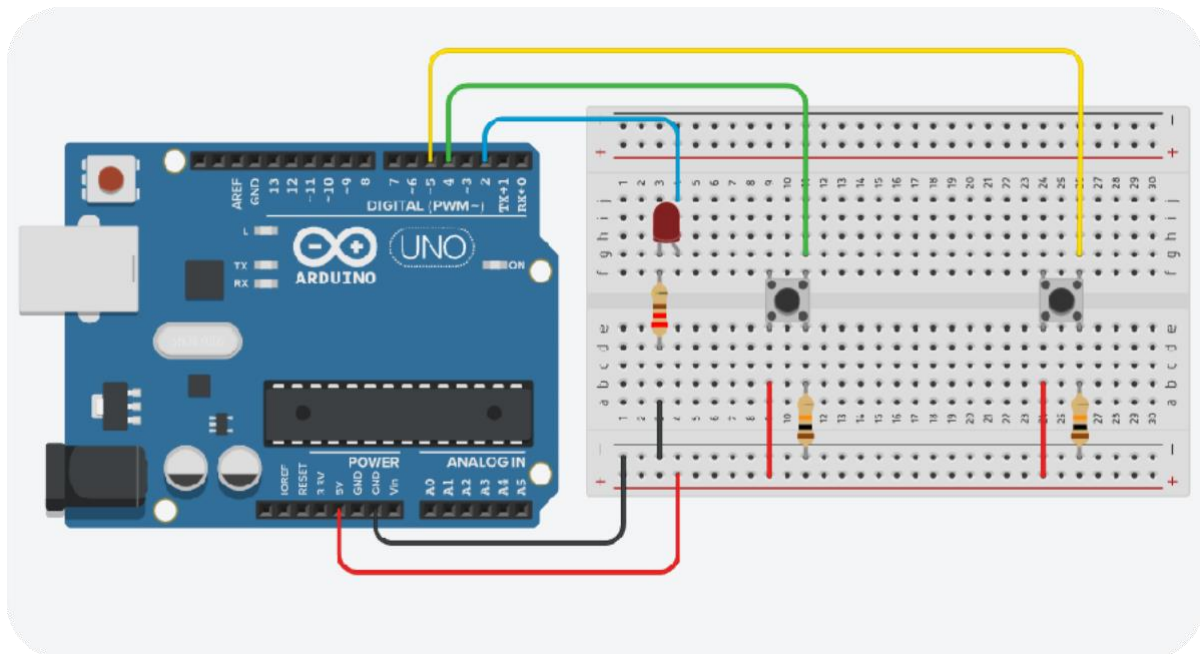
led++;
}
digitalWrite(led-1,LOW);
digitalWrite(led,HIGH);

if(led == 12) led = 4;
}

```

❖ Application 11

Controlling the LED with two buttons: One button turns the LED on, while the other button turns it off.



Materials: 1x Arduino Uno R3, 1x LEDs, 1x 220 Ω resistors, 2x 10 k Ω resistor, 2x Button, Breadboard

Code Example:

```

int start=0;
int stop=0;

void setup(){
  pinMode(2, OUTPUT);
  pinMode(5, INPUT);
  pinMode(4, INPUT);
}

```

```

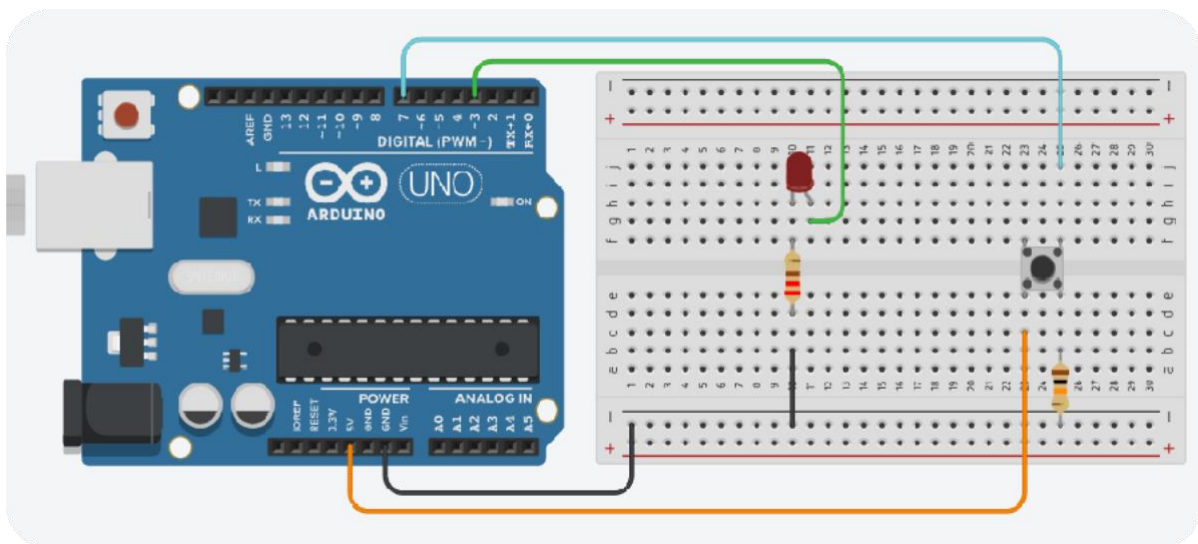
void loop(){
  start = digitalRead(5); //5 nolu pinden butona bak
  if (start == 1) //start eğer 1 ise
  {
    digitalWrite(2, HIGH); //2.pini yak
  }
  stop = digitalRead(4); //4 nolu pinden butona bak.
  if (stop == 1) //stop eğer 1 ise
  {
    digitalWrite(2, LOW); //2.pini söndür
  }
}

```

❖ Application 12

Controlling the LED with a single button-1:

Pressing the same button alternately turns the LED on and off.



Materials: 1x Arduino Uno R3, 1x LEDs, 1x 220 Ω resistors, 1x 10 k Ω resistor, 1x Button, Breadboard

Code Example-1:

This code controls an LED with a button. When the button is pressed, the LED turns on, and when the button is released, the LED turns off. Here's the translation of your description:

```

int buton=0;

void setup(){
  pinMode(3, OUTPUT);
  pinMode(7, INPUT);
}

void loop(){

```

```

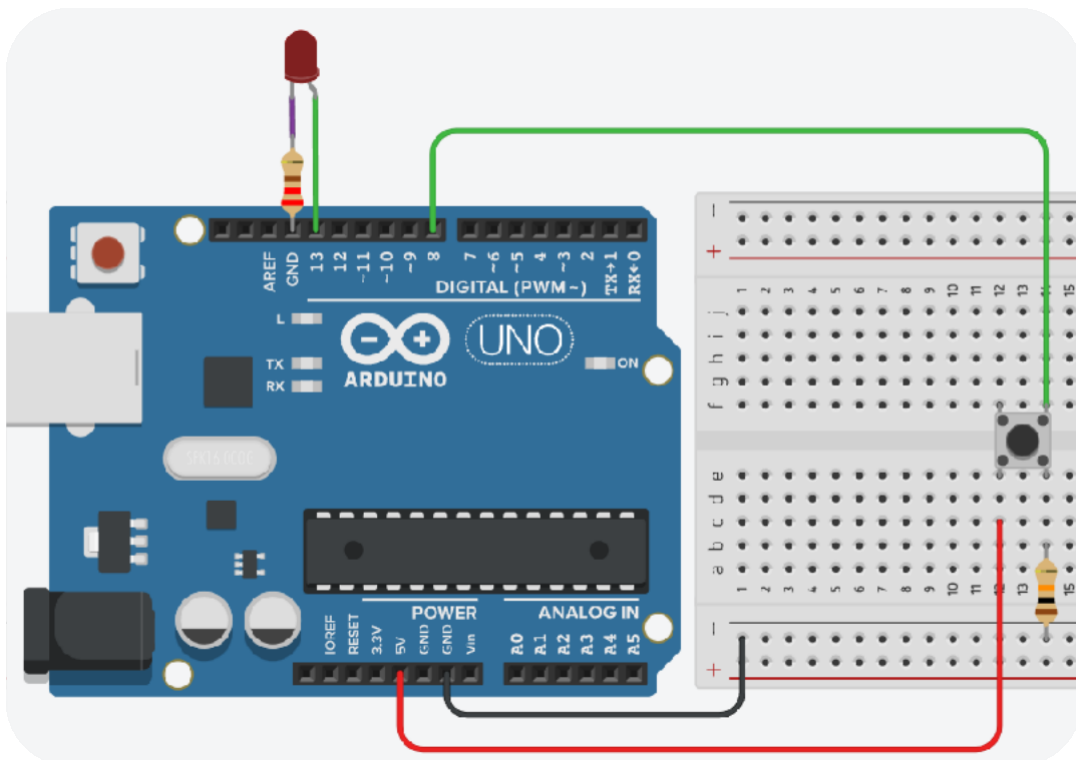
while (buton==0){
  buton = digitalRead(7);
}
digitalWrite(3, HIGH);
while (buton==1){
  buton = digitalRead(7);
}
buton=0;

while(buton==0){
  buton = digitalRead(7);
}
digitalWrite(3, LOW);

while (buton==1){
  buton = digitalRead(7);
}
buton=0;
}

```

Controlling the LED with a single button-2:



Materials: 1x Arduino Uno R3, 1x LEDs, 1x 220 Ω resistors, 1x 10 k Ω resistor, 1x Button, Breadboard

Code Example-2:

This code controls an LED connected to pin 13 based on the state of a button connected to pin 8. The LED turns on when the button is pressed and turns off when the button is released. Here's a brief breakdown of how it works:

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(8, INPUT);
}

void loop()
{
  while (digitalRead(8)==0)
  {}
  while (digitalRead(8)==1)
  {
    digitalWrite(13, HIGH);
  }

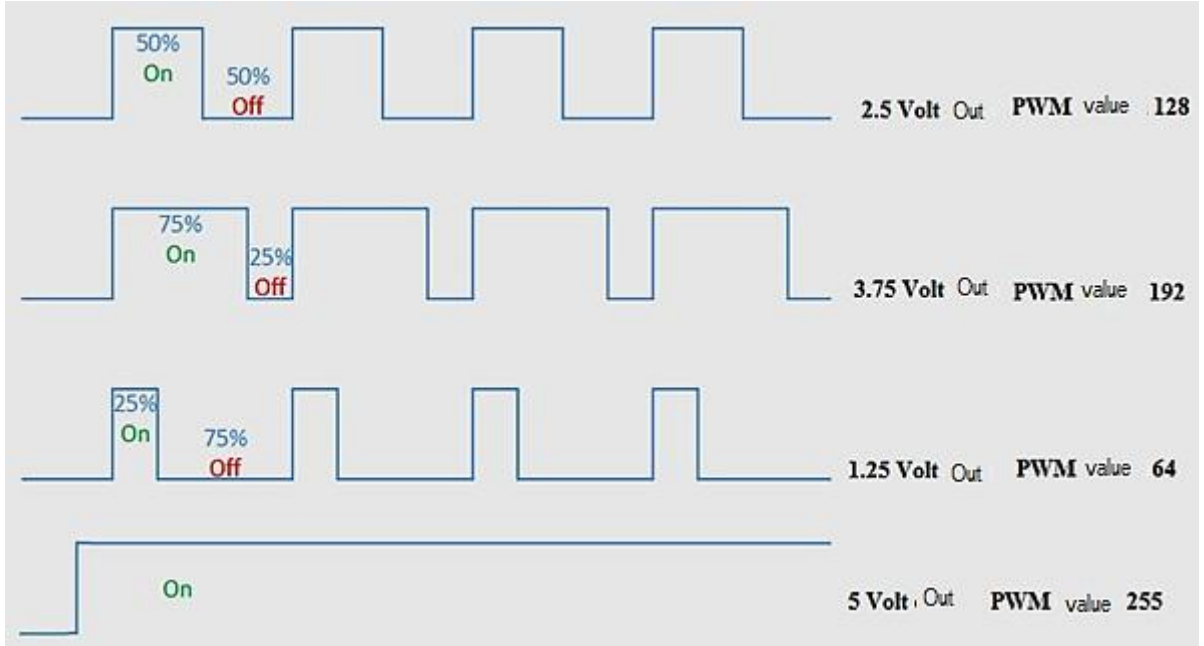
  while (digitalRead(8)==0)
  {}
  while (digitalRead(8)==1)
  {
    digitalWrite(13, LOW);
  }
}
```

Analog Output RGB LED Application

Arduino Analog Output: The pins on Arduino boards marked with a “~” provide PWM (Pulse Width Modulation) output. PWM output sends the output voltage at 5 volts for a certain period and at 0 volts for another period, continuously and rapidly. Below is an example of a PWM signal.



Avrupa Birliği tarafından
ortak finanse edilmektedir



PWM Output Signals

Arduino UNO'nun PWM çıkışı: 8 bittir. $2^8=256$ ile Arduino ile çıkışa gönderebileceğimiz en büyük değerdir. Çıkıştan 5 Volt almak için 255, 0 Volt almak için 0 göndeririz. Bu değerlerden PWM çıkışının çözünürlüğünü tespit ederiz. $5/255=0.019$ değerini elde ederiz. Bu, Arduino UNO ile 0.019 Volt'luk değişikliği algılayabileceğimiz anlamına gelir. Arduino UNO'da kullanılan ATmega328P işlemcisi 6 adet PWM çıkışı vardır. Bu çıkışların frekansı işlemci içerisindeki zamanlayıcılar ile ayarlanabilir. Aşağıdaki tabloda hangi zamanlayıcının hangi pinler tarafından kullanıldığı görülmektedir.

Timer 0	5 and 6 numbered pins
Timer 1	9 and 10 numbered pins
Timer 2	3 and 11 numbered pins

❖ Application 13

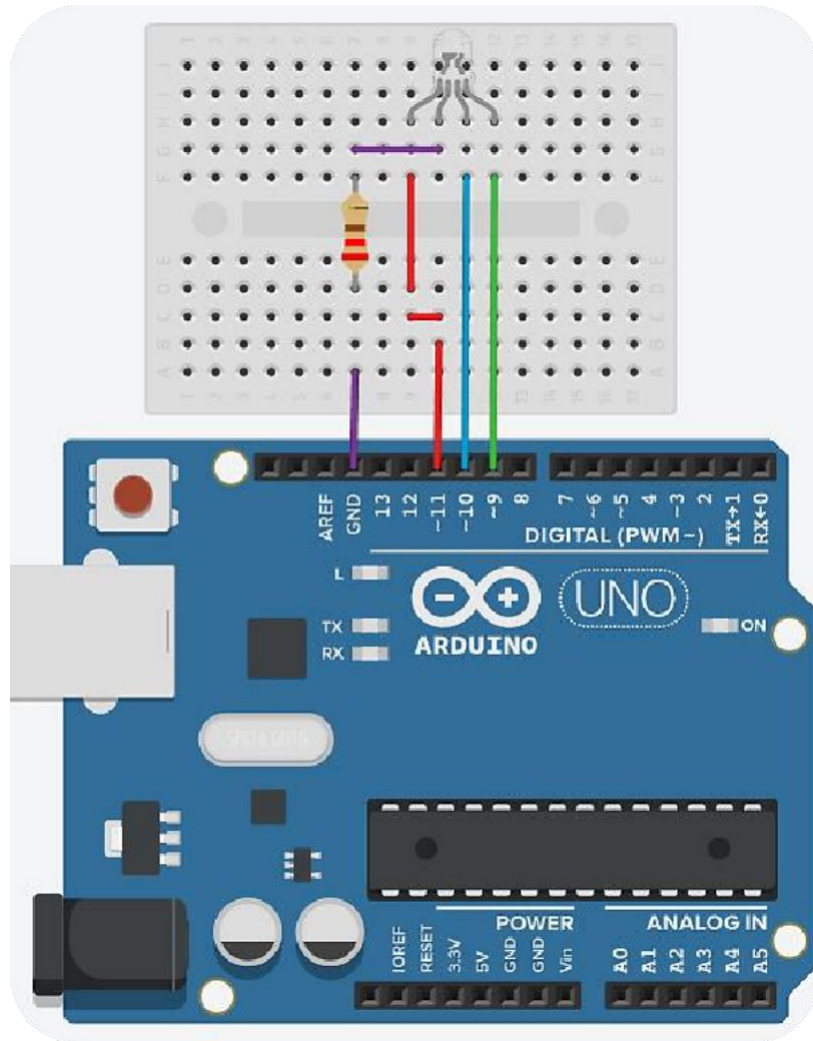
Let's light the RGB LED in sequence: red, green, and blue.

Connect Arduino's PWM output pins 9, 10, and 11 to the red, green, and blue pins of the RGB LED, respectively. The GND pin should be connected to the negative (-) pin of the LED.

Materials: 1x Arduino Uno R3, 1x RGB LED, 1x 220 Ω resistor, Mini Breadboard



Avrupa Birliği tarafından
ortak finanse edilmektedir



Code Example:

```
void setup()
{
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
}

void loop()
{
  analogWrite(9,255);
  analogWrite(10,0);
  analogWrite(11,0);
  delay(500);

  analogWrite(9,0);
  analogWrite(10,255);
  analogWrite(11,0);
  delay(500);
}
```



```
analogWrite(9,0);
analogWrite(10,0);
analogWrite(11,255);
delay(500);
}
```

❖ *Application 14*

Lighting the RGB LED in random colors at 1-second intervals. Apply this to the circuit from Application 11.

Code Example:

```
int kirmizi;
int yesil;
int mavi;

void setup()
{
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  randomSeed(analogRead(A0));
}

void loop()
{
  kirmizi = random(255);
  yesil = random(255);
  mavi = random(255);
  analogWrite(9,yesil);
  analogWrite(10,mavi);
  analogWrite(11,kirmizi);
  delay(500);
}
```

Used commands:

randomSeed(analogRead(A0)); Initializes the random number generator. We connect it to an unused analog input so that this generator does not produce the same values repeatedly. Unused analog inputs continuously generate random values. This ensures that the value produced by random is constantly different. **kirmizi = random(255);** Generates a random value between 0 and 255 and assigns this value to the variable kirmizi.

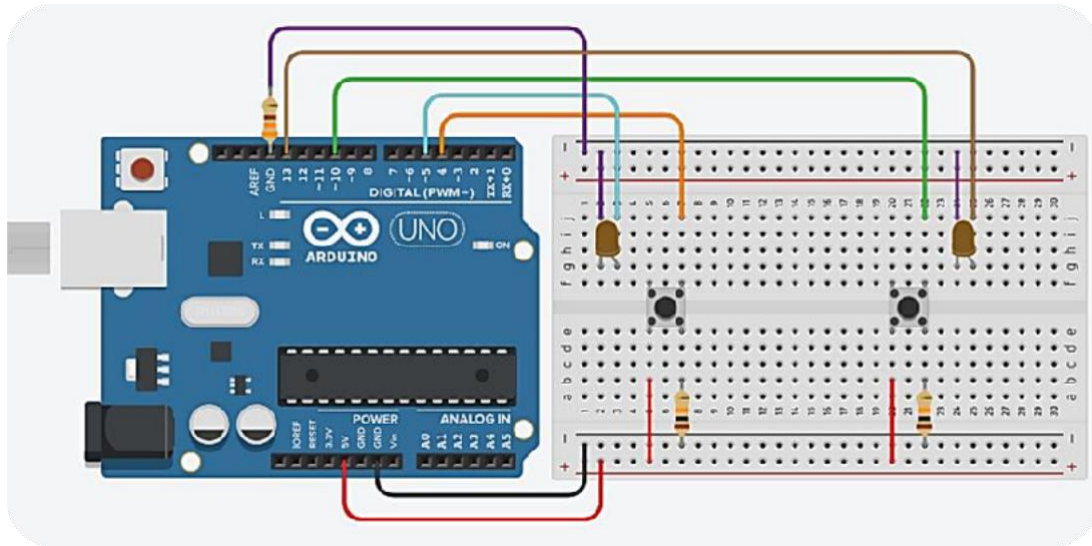


Avrupa Birliği tarafından
ortak finanse edilmektedir

❖ Application 15

Making a Right-Left Vehicle Signal Light Using Buttons:

Pressing the right button turns on the right signal, while pressing the left button turns on the left signal.



Materials: 1x Arduino Uno R3, 2x LEDs, 1x 220 Ω resistor, 2x 10 k Ω resistors, 2x Buttons, Breadboard

Code Example:

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, INPUT);
  pinMode(10, INPUT);
}

void loop()
{
  if (digitalRead(4)==1)
  {
    digitalWrite(5, HIGH);
    delay(500);
    digitalWrite(5, LOW);
    delay(500);
  }

  if (digitalRead(10)==1)
  {
    digitalWrite(13, HIGH);
    delay(500);
```

```
digitalWrite(13, LOW);  
delay(500);  
}  
  
}
```

Analog Input Potentiometer (Adjustable Resistor) Applications:

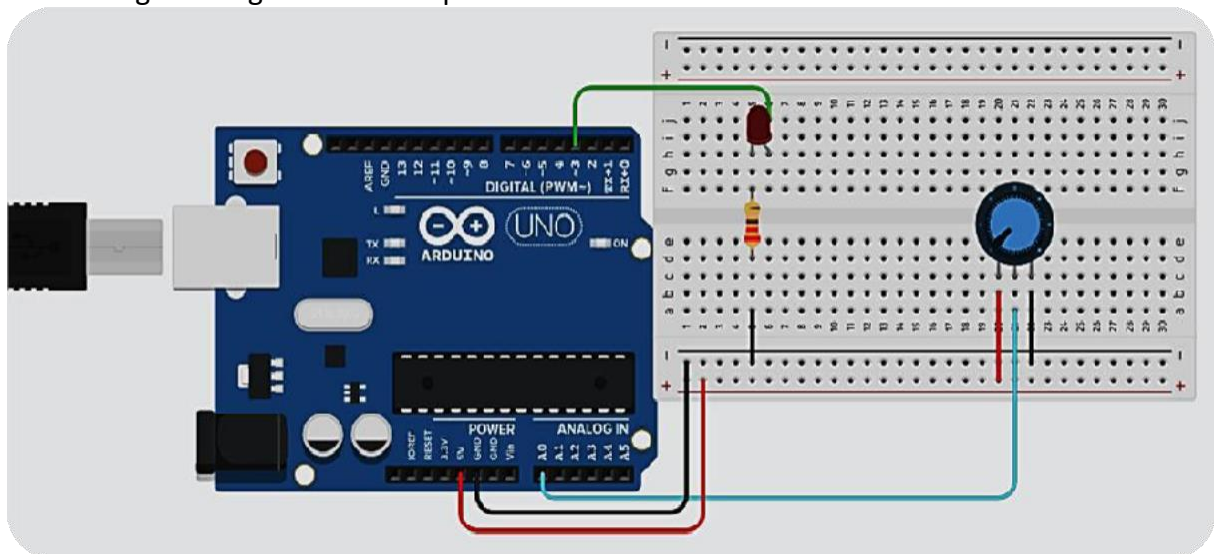
Potentiometer: The Arduino Uno has 6 analog inputs with 10-bit resolution, ranging from A0 to A5. These analog inputs can also be used digitally. When used digitally, they must be defined using the pinMode command. When defining with the pinMode command, these pins should be assigned numbers just like digital pins. The numbers for these pins are as follows:

A0	14
A1	15
A2	16
A3	17
A4	18
A5	19

Since analog inputs can be represented in 10 bits, $2^{10}=1024$ is obtained. Since 5 volts can be applied to the Arduino inputs, reading 1024 from the analog input indicates that 5 volts have been applied. The sensitivity of the analog input can read a precision of $5/1024=0.00488$ volts.

❖ Application 16

Controlling LED brightness with a potentiometer.



Materials: 1x Arduino Uno R3, 1x LED, 1x 220 Ω direnç, 1x Pot, Breadbord



Avrupa Birliği tarafından
ortak finanse edilmektedir

Code Example:

```
int pot=A0;
int led=3;

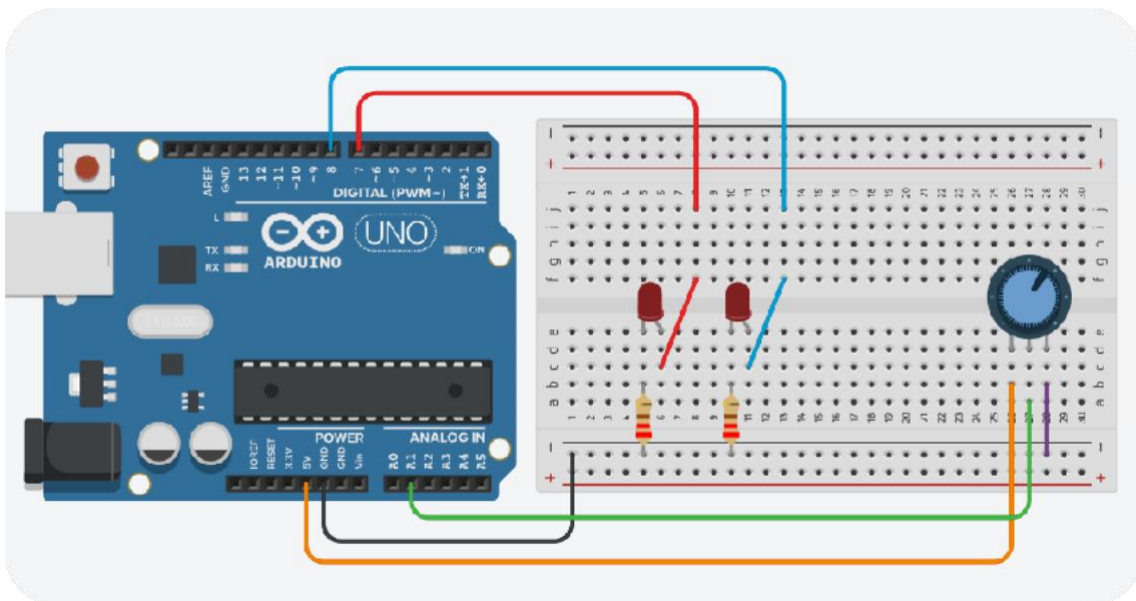
void setup()
{
  Serial.begin(9600);
  pinMode(pot, INPUT);
  pinMode(led, OUTPUT);
}

void loop()
{
  int potdeger= analogRead(pot);
  potdeger=map(potdeger,0,1023,0,255);
  analogWrite(led, potdeger);
  delay(10);
  Serial.println(potdeger);
}
```

❖ Application 17

If the potentiometer connected to the A1 input is below 2.5 volts, the LED connected to pin 7 will light up; if it is above 2.5 volts, the LED connected to pin 8 will light up.

The middle pin of the potentiometer is connected to the Arduino's analog input A1. One of the other two pins of the potentiometer is connected to the 5V pin, and the other pin is connected to the GND pin. The negative (-) terminal of the LEDs is connected to GND through a resistor, while the positive terminals are connected to pins 7 and 8, respectively.



Malzemeler: 1x Arduino Uno R3, 2x LED, 2x 220 Ω direnç, 1x Pot, Breadbord



Avrupa Birliği tarafından
ortak finanse edilmektedir

Code Example:

```
int pot;

void setup()
{
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
}

void loop()
{
  pot = analogRead(A1);
  pot = map(pot,0,1024,0,255);
  if(pot >= 127)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,LOW);
  }
  if(pot < 127)
  {
    digitalWrite(8,HIGH);
    digitalWrite(7,LOW);
  }
}
```



Avrupa Birliği tarafından
ortak finanse edilmektedir