

# Introduction to *Arduino Uno*

# Intended Learning Outcomes

At the end of the lesson, you should be able to:

1. Identify the parts of Arduino Uno.
2. Know the components of Arduino Uno.
3. Learn how to program to read inputs and to produce outputs in response to that input.
4. Create a simple program using the Arduino programming language using push-button.
5. Create a running lights effect in response to a button press or a toggle switch using the Arduino Uno Board.

# What is Arduino?

- Arduino is an open-source electronics platform based on easy-to-use hardware and software.
- Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online.
- You can tell your board what to do by sending a set of instructions to the microcontroller on the board.
- To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

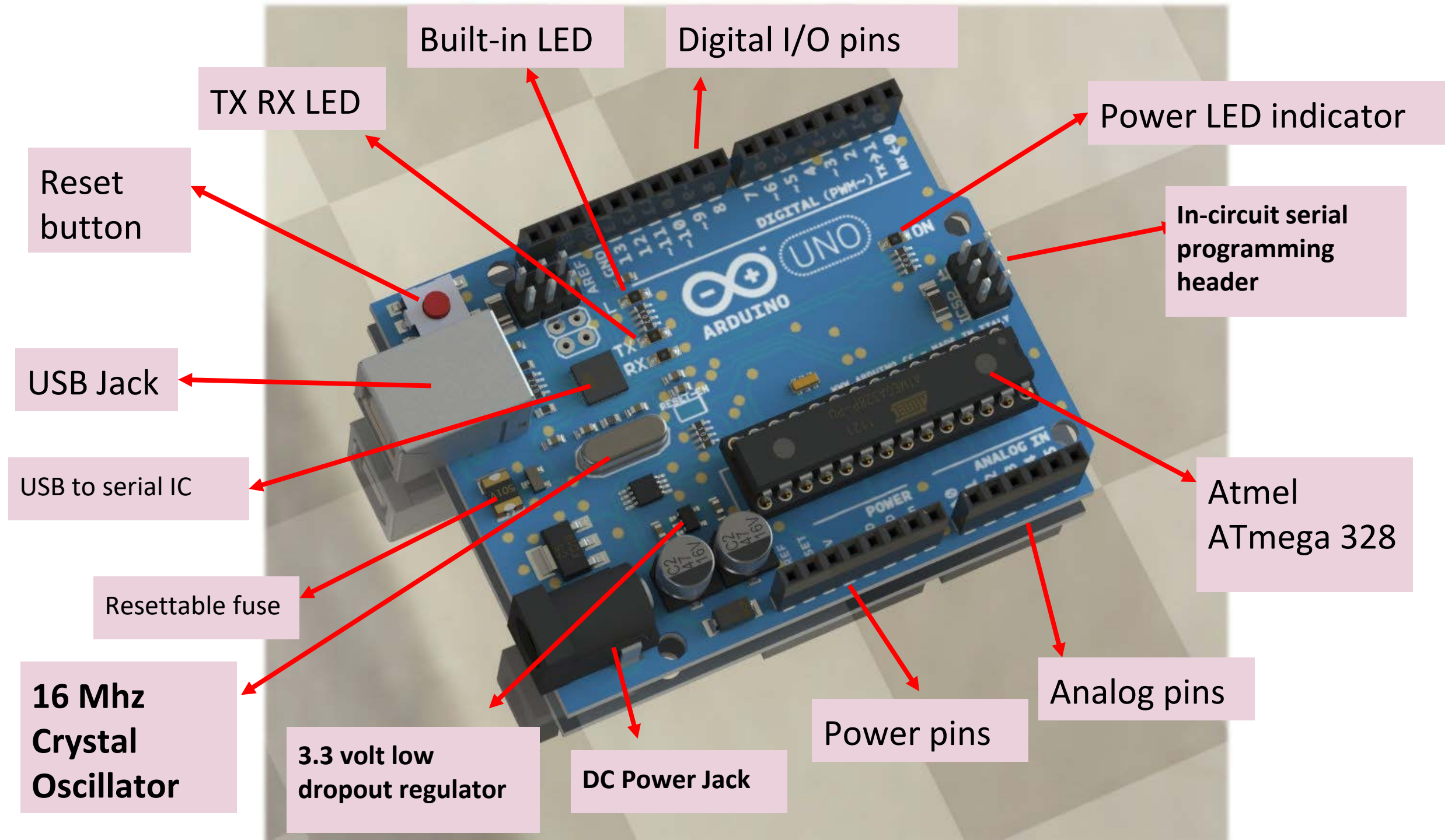


# What is Arduino?

- 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.
- As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments.
- All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The [software](#), too, is open-source, and it is growing through the contributions of users worldwide.

# Arduino Uno board provides the user with:

- 6 analog input pins
- 14 digital I/O pins of which 6 of them can also be used for PWM outputs
- a power jack
- a USB port
- an ICSP header
- a reset button
- a small LED connected to digital pin 13
- 16MHz crystal oscillator



# Component Explanations

1. **Digital I/O pins** – input and output pins (0-13) of which 6 of them (3, 5, 6, 9, 10 and 11) also provide PWM (Pulse Width Modulation) output by using the `analogWrite()` function. Pins (0 (RX) and 1 (TX)) are also used to transmit and receive serial data.
2. **Power LED indicator** – LED that lights up when the board is connected to a power source.
3. **ICSP Header** – pins for “In-Circuit Serial Programming” which is another method of programming.
4. **ATmega328 chip** – 8-bit microcontroller that processes the sketch you programmed. This is the soul of the board. It helps the board receive information from users and transmit command signals to devices that are connected to the Arduino board.
5. **Analog input pins** – pins (A0-A5) that take-in analog values to be converted to be represented with a number range 0-1023 through an Analog to Digital Converter (ADC).

# Component Explanations

6. **Power Pins** – pins that can be used to supply a circuit with values VIN (voltage from DC Jack), 3.3V and 5V.
7. **DC Power Jack** – where the power source (AC-to-DC adapter or battery) should be connected. It is limited to input values between 6-20V but recommended to be around 7-12V.
8. 3.3 volt low dropout regulator - **used** to derive lower output voltages from a main supply or battery.
9. **16 Mhz Crystal Oscillator** (aa-suh-lei-tr) – clock that has a frequency of 16MHz
10. **Resettable fuse** - **used** to protect against overcurrent faults in electronic circuits.



# Component Explanations

- 11. USB to serial IC** - The **USB serial** converter **chip** is just a solution to allow PC software to allow application software like the **Arduino** IDE to talk to a device via a software comm port interface, but using the **USB** bus to send the data.
- 12. *USB port/jack*** – allows the user to connect with a USB cable the board to a PC to upload sketches or provide a voltage supply to the board. This is also used for serial communication through the serial monitor from the Arduino software.
- 13. *Reset Button*** – a button that is pressed whenever you need to restart the sketch programmed in the board.
- 14. TX RX LED** - **TX** and **RX** - for Transmit and Receive
- 15. *Built-in LED*** – in order to gain access or control of this pin, you have to change the configuration of pin 13 where it is connected to.

# Microcontroller

- Microcontroller is called a computer on a chip. It has its own memory, CPU, and RAM inbuilt on a single chip. It has pins that can perform digital and analog operations
- Basically, a microprocessor is the brain of the computer while a microcontroller is a muscle with brain and it can perform many tasks on its own (standalone).
- The microcontroller has a solid state memory which can be programmed as many times as you need to perform the function with the pins ,they are called General Purpose Input Output pins or GPIO pins
- They are available with various architectures such as 8bit ,16bit,32bit, and 64 bit

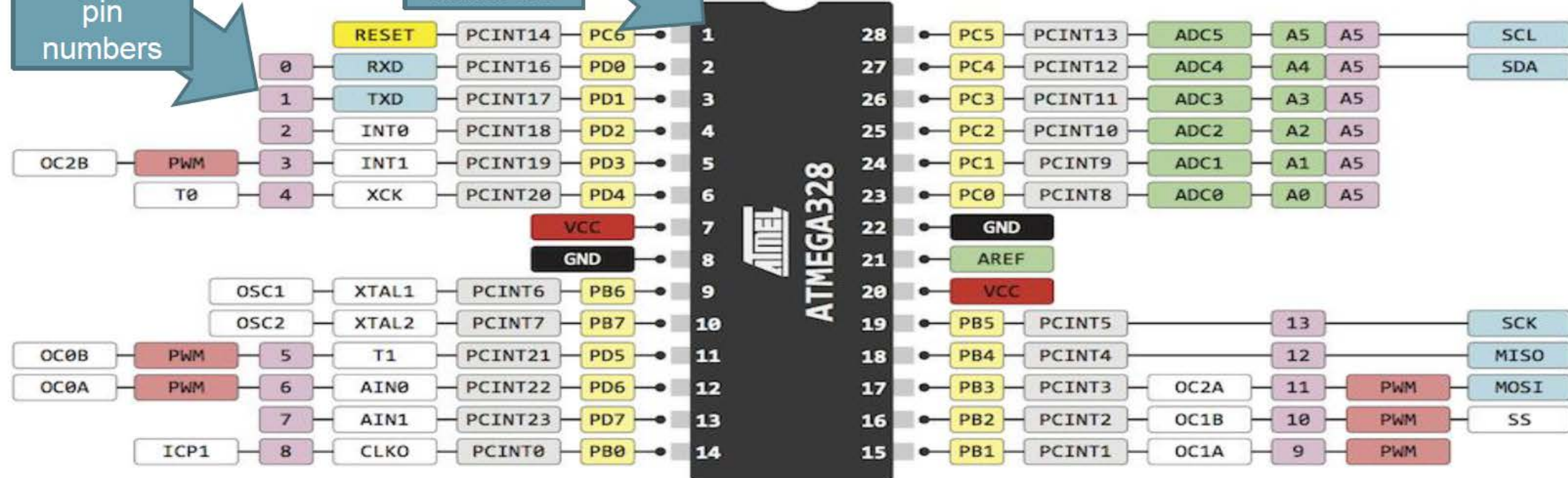
THE  
DEFINITIVE  
**ATMEGA328**  
&Arduino  
PINOUT DIAGRAM

# Atmega 328 microcontroller description



physical  
pin  
numbers

Adruino  
pin  
numbers

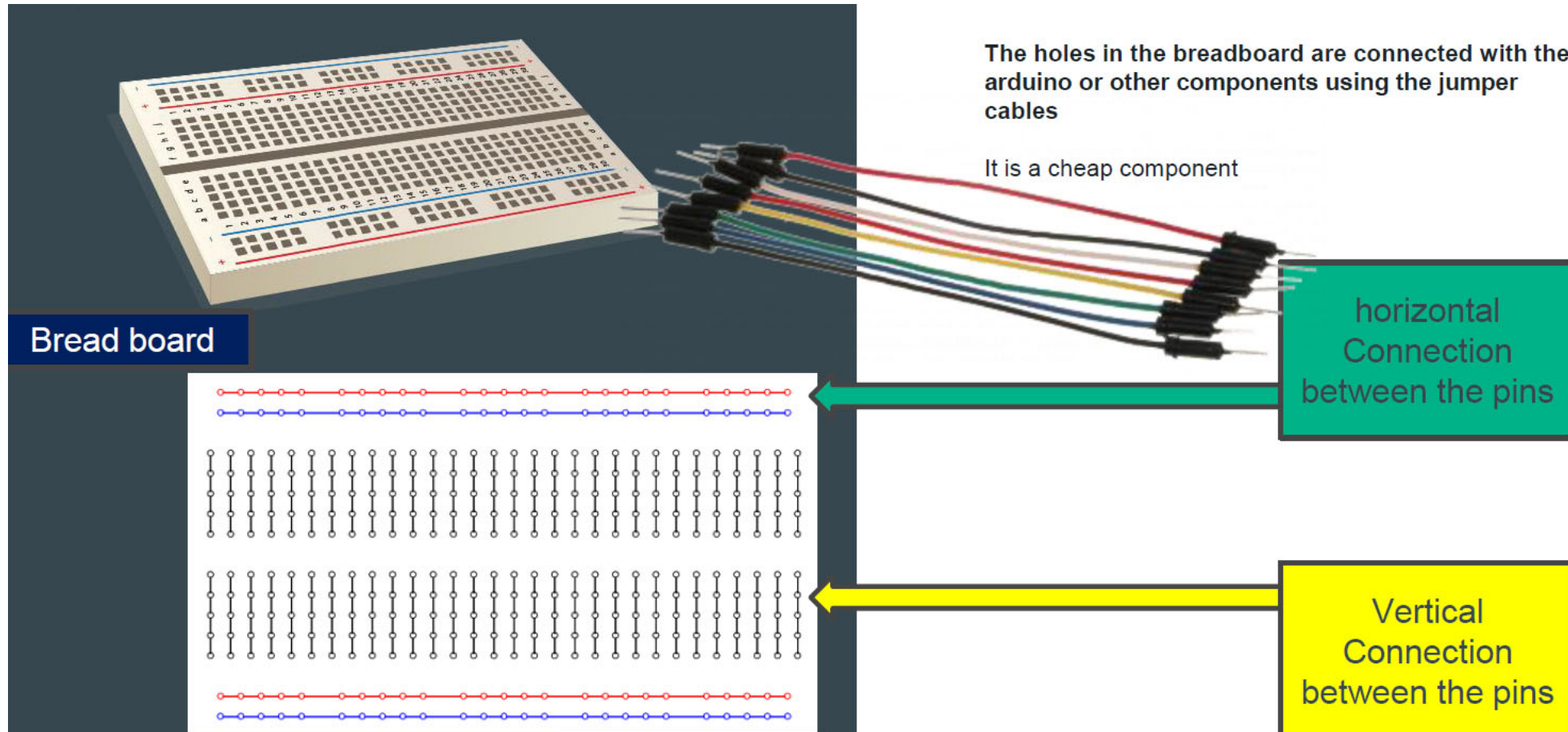


Arduino software for the first  
timers

# Arduino Uno and breadboard wiring

The breadboard is a type of prototyping tool that is used for temporary connections of electronic components.

It is the cheapest and most easy way to connect the components

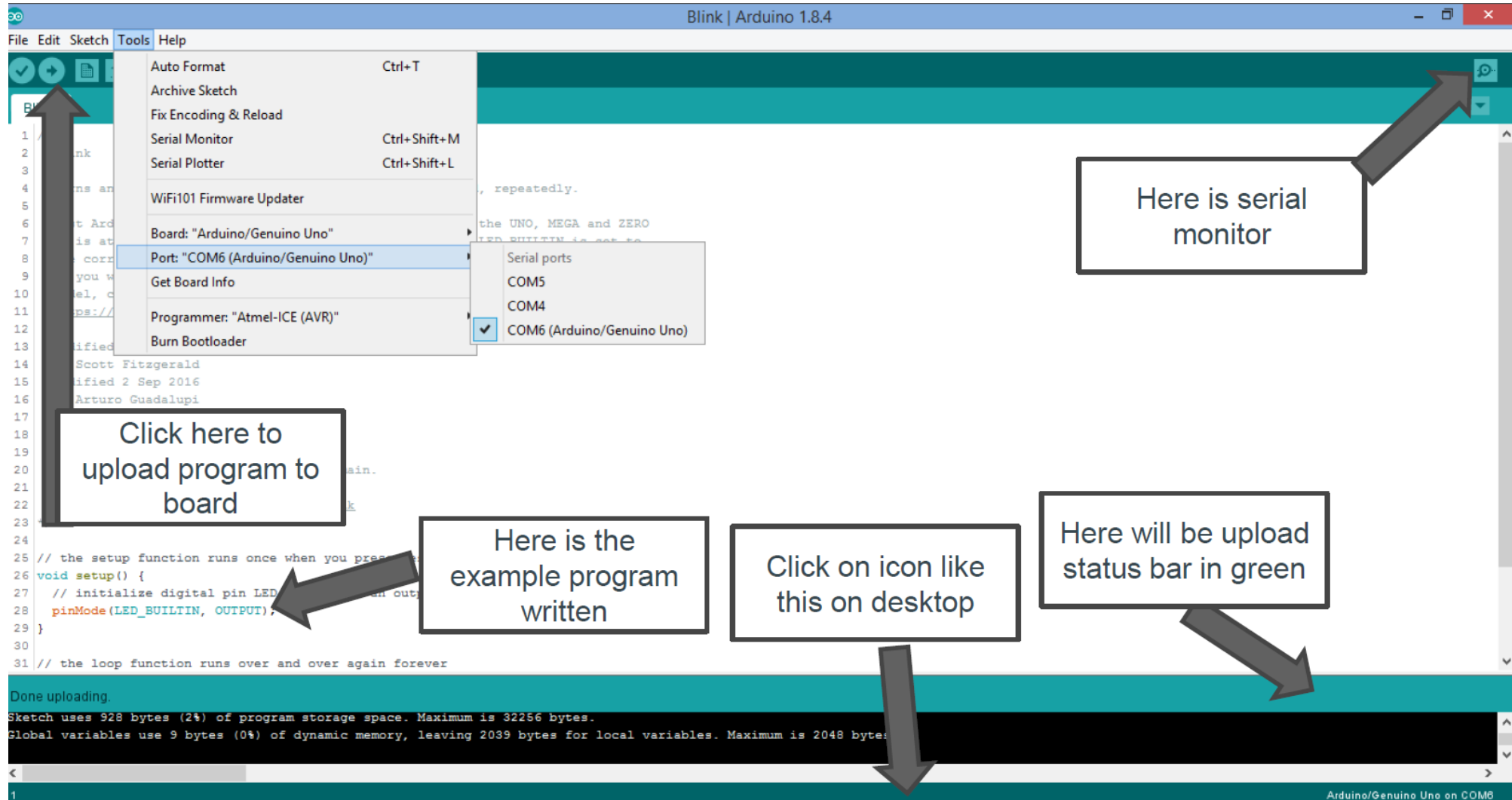


# Let's get started with programming

- The program to run the Arduino board is written in Arduino programming language.
- It has a simple syntax of mainly three steps
  - Declare the pin to use
  - Set it input or output
  - Select its use
  - And put it for continuous use in the void loop and for single-use in void setup
- Check out: <http://arduino.cc/en/Guide/HomePage> (Links to an external site.)
  1. Download & install the Arduino environment (IDE)
  2. Connect the board to your computer via the USB cable
  3. If needed, install the drivers
  4. Launch the Arduino IDE
  5. Select your board
  6. Select your serial port
  7. Open the blink example
  8. Upload the program



# Arduino Environment



# How to blink Led (hello world)

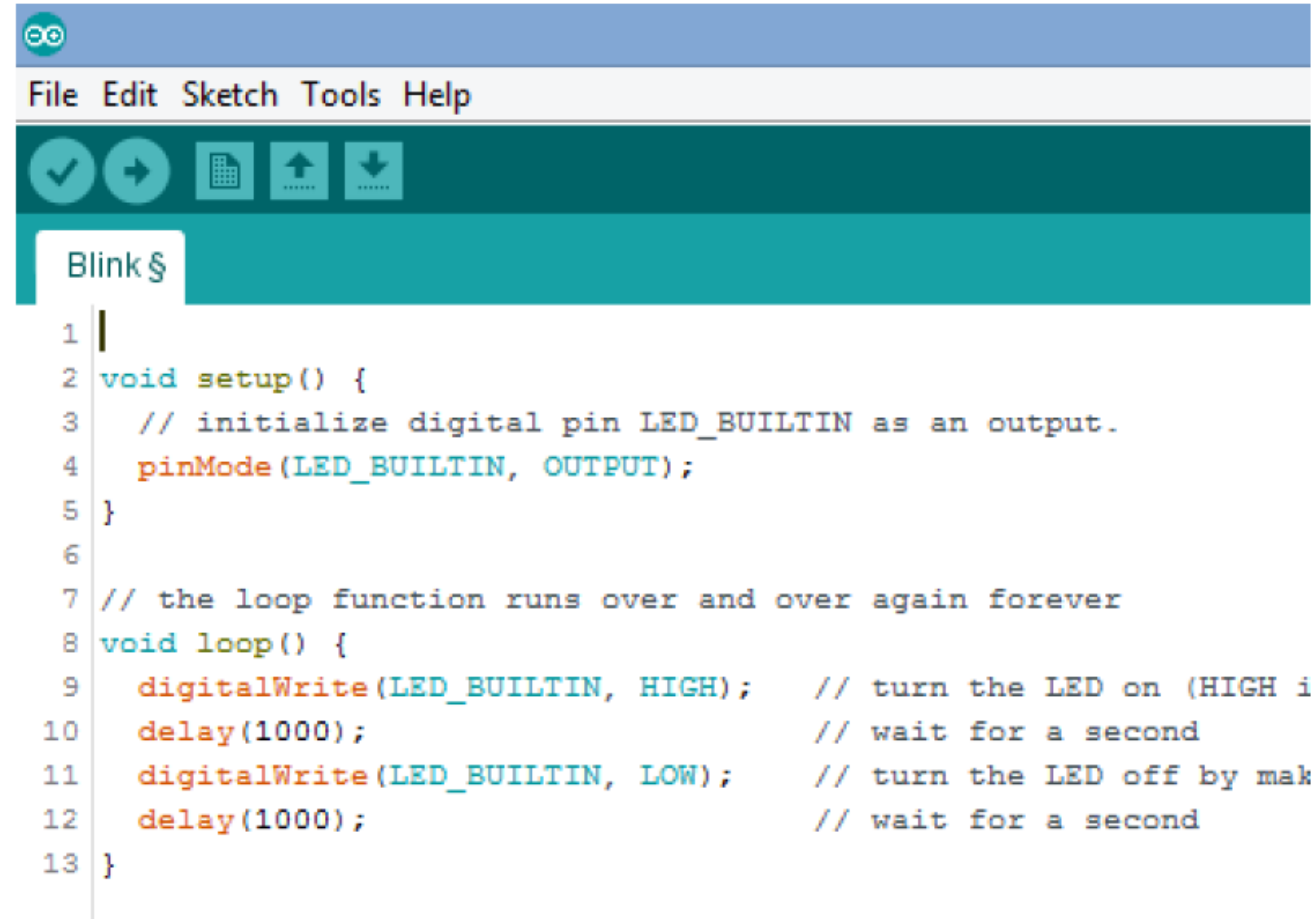
## The code

Test the board by going to file ,select example

Select basics and select blink

Click on arrow and if all good you will see the program uploading and the light will blink on board

## Sample

A screenshot of the Arduino IDE interface. The title bar says "Blink \$". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for a checkmark, a right arrow, a grid, an up arrow, and a down arrow. The main text area contains the following C++ code:

```
1 |
2 void setup() {
3   // initialize digital pin LED_BUILTIN as an output.
4   pinMode(LED_BUILTIN, OUTPUT);
5 }
6
7 // the loop function runs over and over again forever
8 void loop() {
9   digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is
10  delay(1000);                        // wait for a second
11  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by mak
12  delay(1000);                        // wait for a second
13 }
```

Anything else written is the description of the program

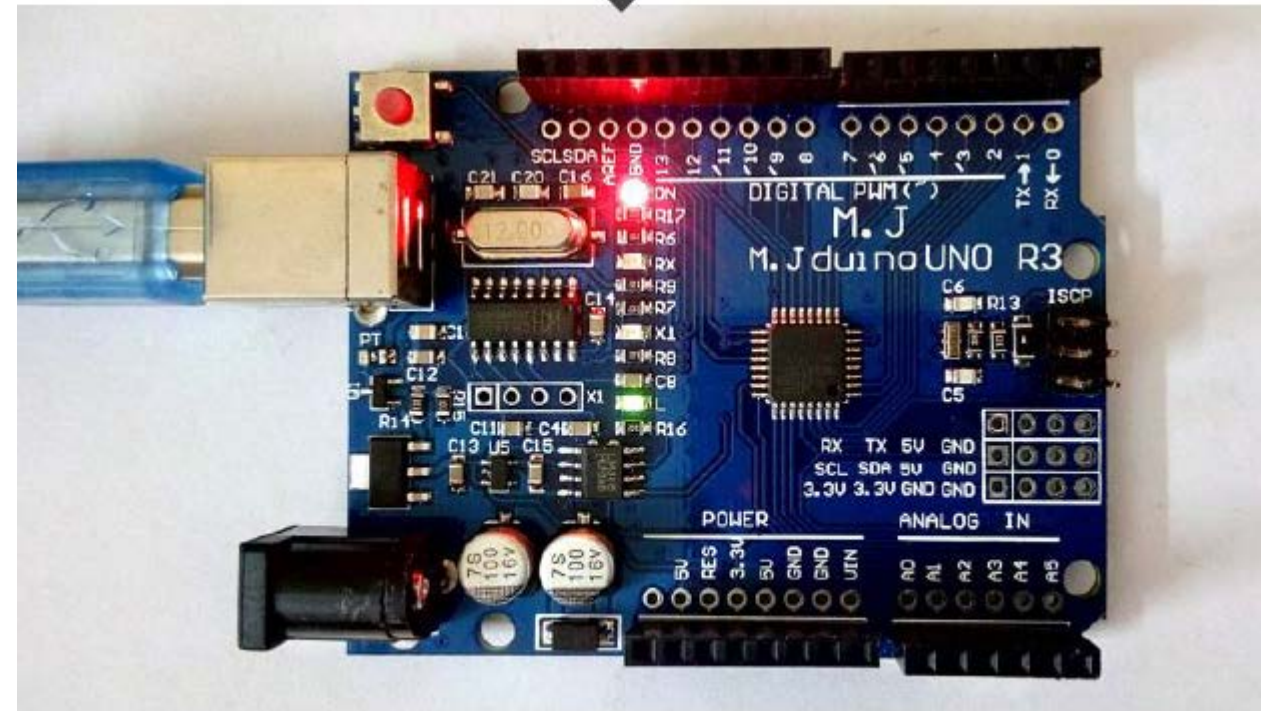


# Output

## Code meaning

- The code is a three-step program
  - Define the pin in void setup
  - Set it output
  - In the void, loop set the pin to digital write = high and wait for 1 second
  - Digital write = low
  - Wait for 1 second
  - Run this in the loop

Led on pin 13 will blink



# Programming

# setup()

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each powerup or reset of the Arduino board.

## Example Code

```
int buttonPin = 3;
```

```
void setup() {  
  Serial.begin(9600);  
  pinMode(buttonPin, INPUT);  
}
```

```
void loop() {  
  // ...  
}
```

# loop()

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Example Code

```
int buttonPin = 3;
```

```
// setup initializes serial and the button pin
```

```
void setup() {  
  Serial.begin(9600);  
  pinMode(buttonPin, INPUT);  
}
```

```
// loop checks the button pin each time,
```

```
// and will send serial if it is pressed
```

```
void loop() {  
  if (digitalRead(buttonPin) == HIGH) {  
    Serial.write('H');  
  }  
  else {  
    Serial.write('L');  
  }  
  
  delay(1000);  
}
```

# digitalWrite()

- Write a HIGH or a LOW value to a digital pin.
- If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

## Syntax

`digitalWrite(pin, value)`

# Example Code (digitalWrite())

The code makes the digital pin 13 an OUTPUT and toggles it by alternating between HIGH and LOW at one second pace.

```
void setup() {  
  pinMode(13, OUTPUT); // sets the digital pin 13 as output  
}
```

```
void loop() {  
  digitalWrite(13, HIGH); // sets the digital pin 13 on  
  delay(1000);           // waits for a second  
  digitalWrite(13, LOW); // sets the digital pin 13 off  
  delay(1000);           // waits for a second  
}
```

# digitalRead()

Reads the value from a specified digital pin, either HIGH or LOW.

## Syntax

```
digitalRead(pin)
```

## Parameters

pin: the Arduino pin number you want to read

## Returns

HIGH or LOW

# Example Code (digitalRead())

Sets pin 13 to the same value as pin 7, declared as an input.

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT); // sets the digital pin 13 as output
  pinMode(inPin, INPUT);   // sets the digital pin 7 as input
}

void loop() {
  val = digitalRead(inPin); // read the input pin
  digitalWrite(ledPin, val); // sets the LED to the button's value
}
```



# analogRead()

Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023.

On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit.

## Syntax

```
analogRead(pin)
```

## Parameters

pin: the name of the analog input pin to read from (A0 to A5 on most boards, A0 to A6 on MKR boards, A0 to A7 on the Mini and Nano, A0 to A15 on the Mega).

## Returns

The analog reading on the pin. Although it is limited to the resolution of the analog to digital converter (0-1023 for 10 bits or 0-4095 for 12 bits). Data type: int.

# analogWrite()

Writes an analog value (PWM wave) to a pin. 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.

## Syntax

```
analogWrite(pin, value)
```

## Parameters

pin: the Arduino pin to write to. Allowed data types: int.

value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

## Example Code

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

```
int ledPin = 9;    // LED connected to digital pin 9
int analogPin = 3; // potentiometer connected to analog pin 3
int val = 0;       // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT); // sets the pin as output
}

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

# Variables

A **variable** is a place to store a piece of data. It has a name, a value, and a type. For example, this statement (called a declaration):

```
int pin = 13;
```

creates a variable whose name is pin, whose value is 13, and whose type is int. Later on in the program, you can refer to this variable by its name, at which point its value will be looked up and used. For example, in this statement:

```
pinMode(pin, OUTPUT);
```

it is the value of pin (13) that will be passed to the pinMode() function. In this case, you don't actually need to use a variable, this statement would work just as well:

```
pinMode(13, OUTPUT);
```

# Variable Scope

It refers to the part of your program in which the variable can be used. This is determined by where you declare it. For example, if you want to be able to use a variable anywhere in your program, you can declare at the top of your code. This is called a global variable; here's an example:

```
int pin = 13;
void setup()
{
  pinMode(pin, OUTPUT);
}
void loop()
{
  digitalWrite(pin, HIGH);
}
```

As you can see, pin is used in both the setup() and loop() functions. Both functions are referring to the same variable, so that changing it one will affect the value

# Variable Scope

```
int pin = 13;
void setup()
{
  pin = 12;
  pinMode(pin, OUTPUT);
}
void loop()
{
  digitalWrite(pin, HIGH);
}
```

Here, the digitalWrite() function called from loop() will be passed a value of 12, since that's the value that was assigned to the variable in the setup() function.

# Variable Scope

If you only need to use a variable in a single function, you can declare it there, in which case its scope will be limited to that function. For example:

```
void setup()
{
  int pin = 13;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, HIGH);
}
```

In this case, the variable pin can only be used inside the setup() function. If you try to do something like this:

```
void loop()
{
  digitalWrite(pin, LOW); // wrong: pin is not in
  scope here.
}
```

# Examples

# Analog Read Serial

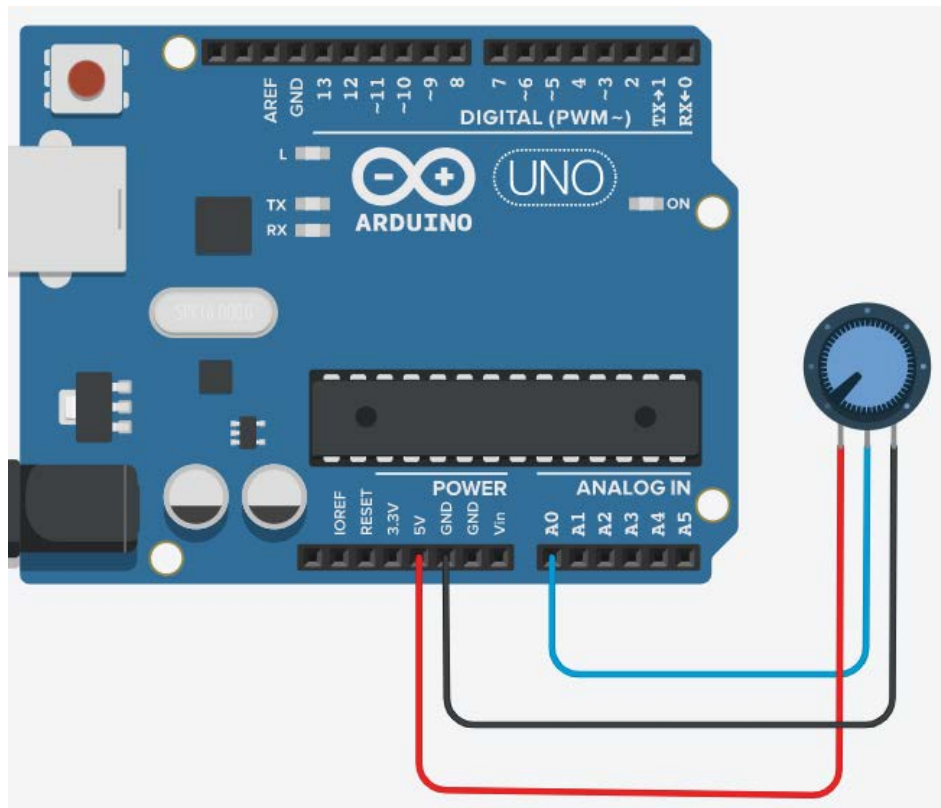
This example shows you how to read analog input from the physical world using a potentiometer.

A **potentiometer** is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your board, it is possible to measure the amount of resistance produced by a potentiometer (or *pot* for short) as an analog value. In this example you will monitor the state of your potentiometer after establishing serial communication between your Arduino and your computer running the Arduino Software (IDE).

## Hardware Required

- Arduino Board
- 10k ohm Potentiometer





Serial Monitor

```

839
839
921
921
921
921
921
982
982
982
982
1023
1023
1023
1023

```

```
/*
```

## AnalogReadSerial

Reads an analog input on pin 0, 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.

This example code is in the public domain.

<http://www.arduino.cc/en/Tutorial/AnalogReadSerial>

```
*/
```

```

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

```

```

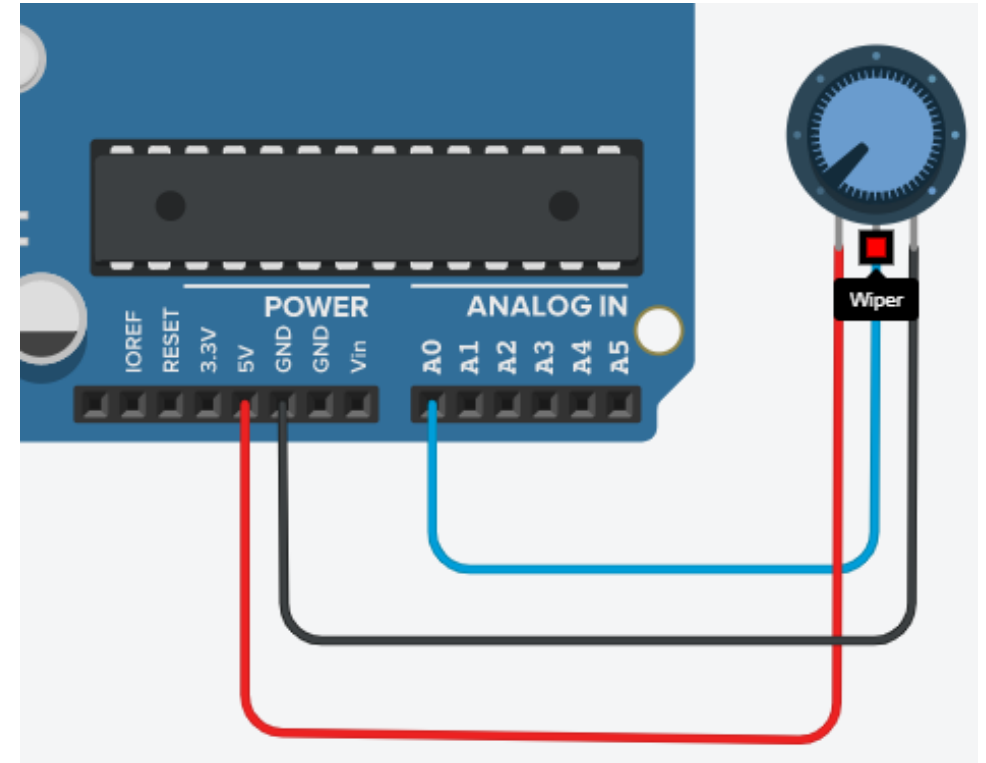
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);    // delay in between reads for stability
}

```

# Explanation

By turning the shaft of the potentiometer (puh·ten·shee·**aa**·muh·tr), you change the amount of resistance on either side of the wiper, which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10k ohm), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the *analog voltage* that you're reading as an input.

The Arduino boards have a circuit inside called an *analog-to-digital converter or ADC* that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, [`analogRead\(\)`](#) returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.



## Code

In the sketch below, the only thing that you do in the setup function is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the command:

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

Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an int datatype) coming in from your potentiometer:

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

Finally, you need to print this information to your serial monitor window. You can do this with the command `Serial.println()` in your last line of code:

```
Serial.println(sensorValue)
```

Now, when you open your Serial Monitor in the Arduino Software (IDE) (by clicking the icon that looks like a lens, on the right, in the green top bar or using the keyboard shortcut Ctrl+Shift+M), you should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

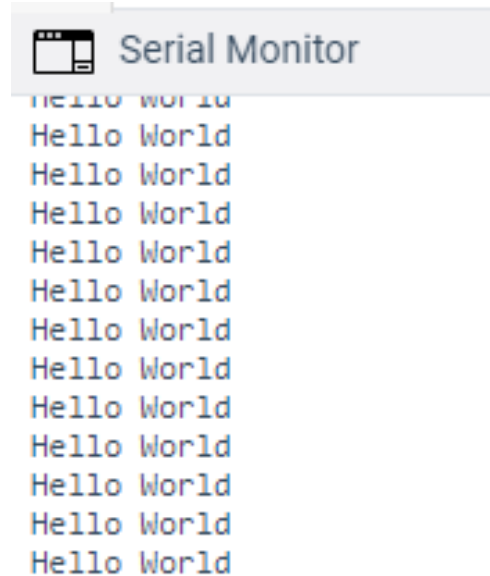
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

# Printing Hello World

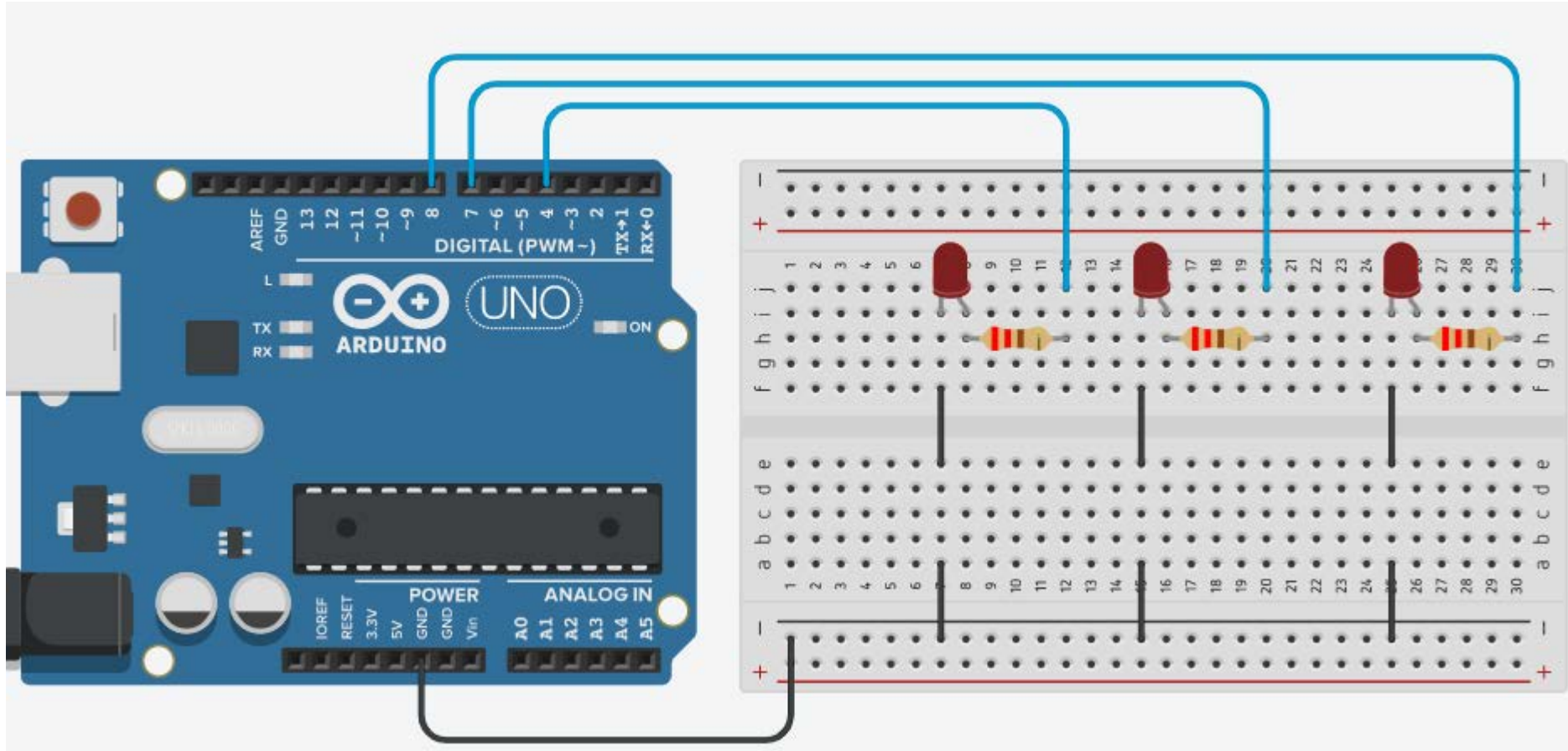
```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {

  Serial.println("Hello World");
  delay(100);          // delay in between reads for stability
}
```



# Turn on, LED light

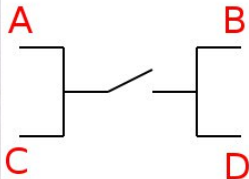
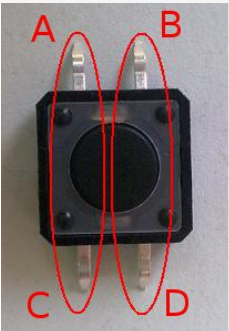
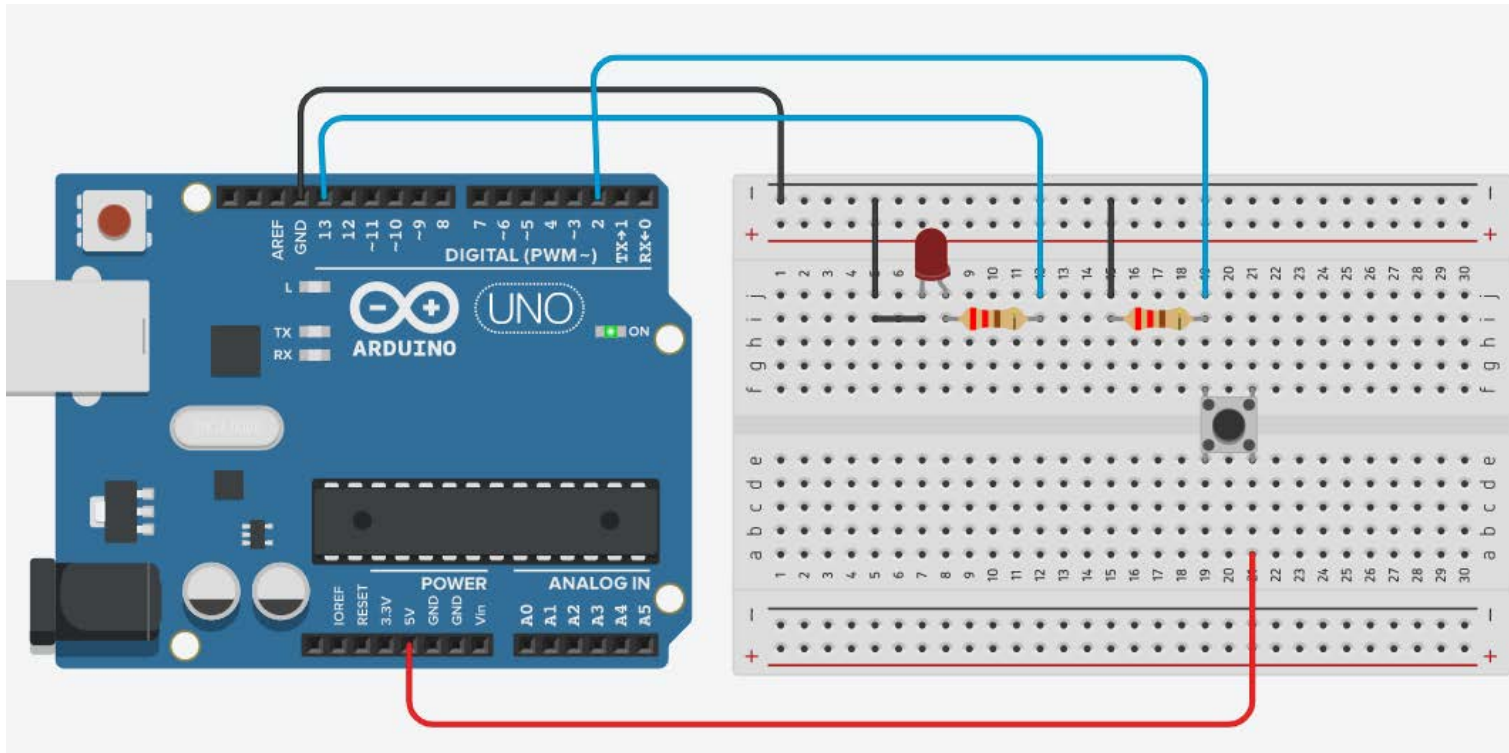


```
int pin4 = 4, pin7 = 7, pin8 = 8;
```

```
void setup()  
{  
  pinMode(pin4, OUTPUT);  
  pinMode(pin7, OUTPUT);  
  pinMode(pin8, OUTPUT);  
}
```

```
void loop()  
{  
  digitalWrite(pin4, HIGH);  
  digitalWrite(pin7, HIGH);  
  digitalWrite(pin8, HIGH);  
  delay(500);  
  
  digitalWrite(pin4, LOW);  
  digitalWrite(pin7, LOW);  
  digitalWrite(pin8, LOW);  
  delay(500);  
}
```

# Turn On, LED light with a Push Button



```
int inputStatus = 0;
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(2, INPUT);
}

void loop()
{
  inputStatus = digitalRead(2);

  if (inputStatus == HIGH) {
    digitalWrite(13, HIGH);
    delay(100); // Wait for 100 millisecond(s)
  }
  else {
    digitalWrite(13, LOW);
    delay(100); // Wait for 100 millisecond(s)
  }
}
```