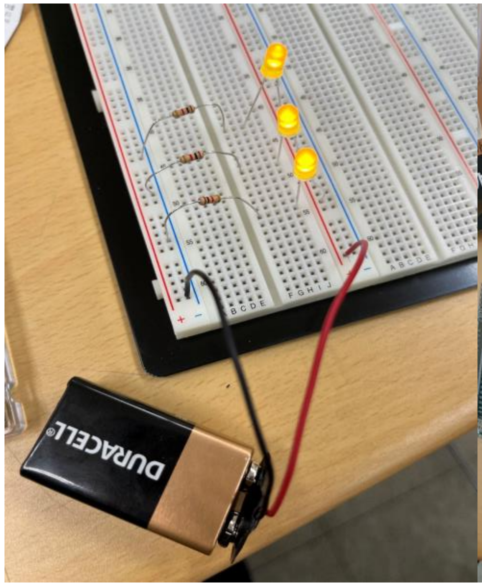
LECTURE 4
Arduino Programming and Applications

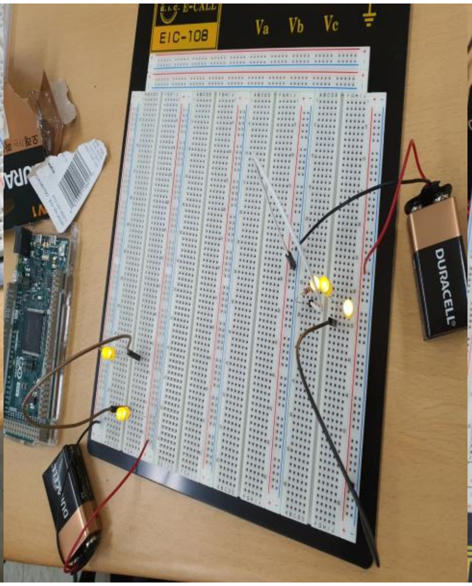
Instructor: Osman Gul, Ph.D. Candidate
Department of Mechanical Engineering
Korea Advanced Institute of Science and Technology (KAIST)

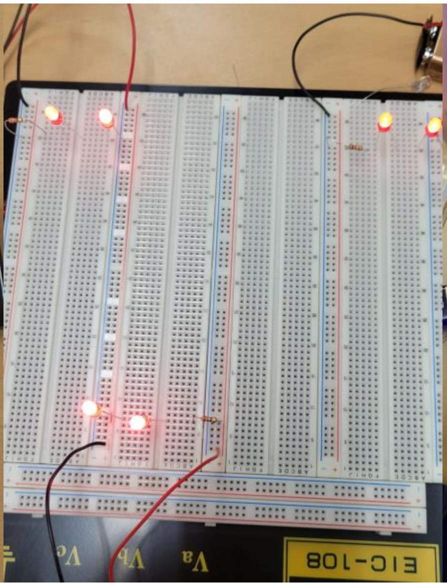
Syllabus:

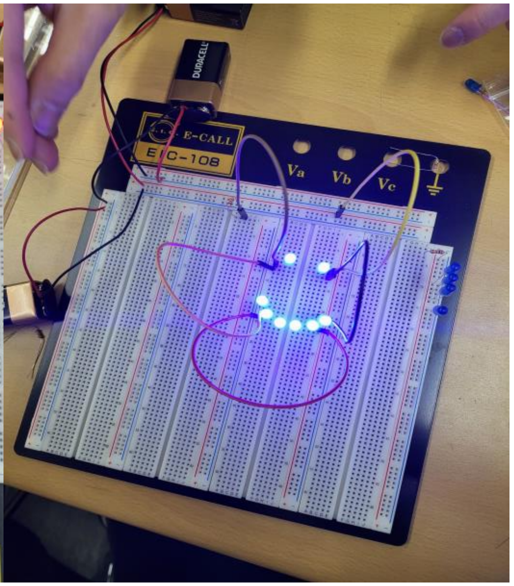
Week	Date& Time*	Content
1		No Class (Course Add/Drop Period)
2	March 6 (Mon) 20-22	Orientation and Course Overview
3	March 13 (Mon) 20-22	Sensor Fundamentals
4	March 20 (Mon) 20-22	Arduino Fundamentals
5	March 27 (Mon) 20-22	Arduino Programming and Applications
6	April 3 (Mon) 20-22	Force Sensor and Application: Control Multiple LEDs
7	April 10 (Mon) 20-22	Light Sensor and Application: Solar Tracker
8	April 17	No Class (Midterm)
9	April 24 (Mon) 20-22	Humidity and Temperature Sensor and Application: Temperature Controlled Fan
10	May 1 (Mon) 20-22	Gas Sensor and Application: Smoke Detector
11	May 8 (Mon) 20-22	Sound Sensor and Application: Control LED by Clapping
12	May 15 (Mon) 20-22	Accelerometer Sensor and Application: Ping Pong Game
13	May 22 (Mon) 20-22	Ultrasonic Sensor and Application: Flappy Bird Game
14	May 29 (Mon) 20-22	Course Wrap-up
15	June 5	No Class (Final)
16	June 12	No Class (Final)

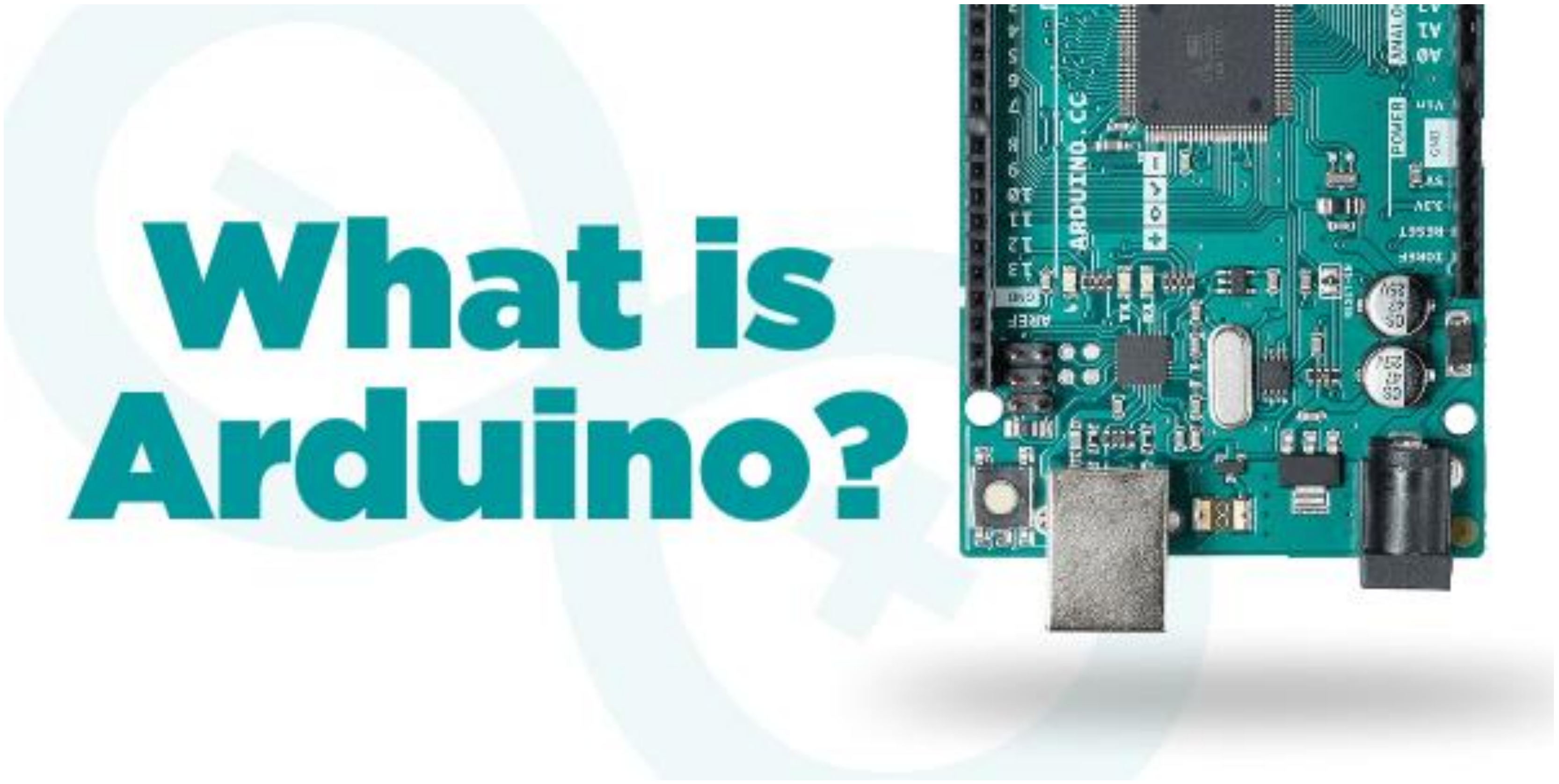
Last Week:


Team #1


Team #2

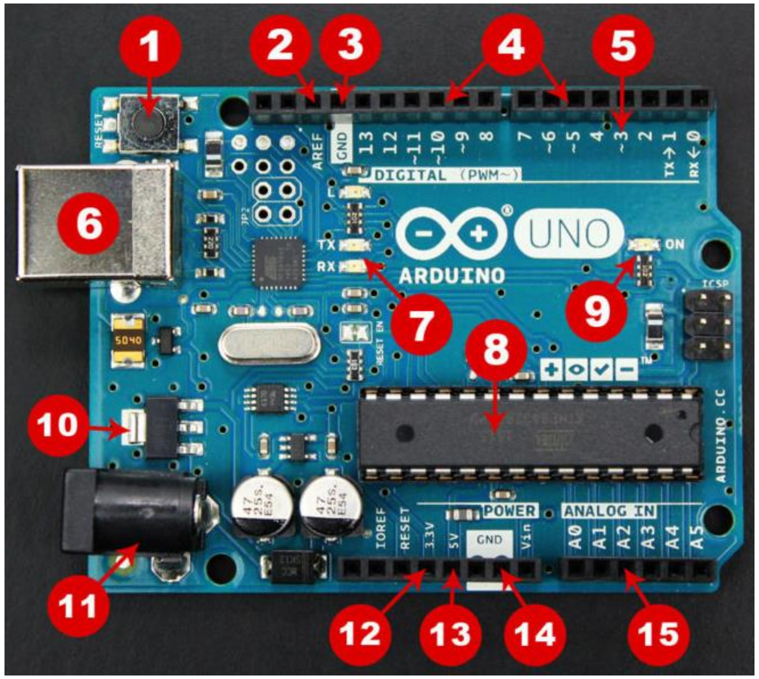

Team #3


Team #4



https://robu.in/

Arduino Uno:



1.Reset Button – This will restart any code that is loaded to the Arduino board

2.AREF – Stands for "Analog Reference" and is used to set an external reference voltage

3.Ground Pin – There are a few ground pins on the Arduino and they all work the same

4.Digital Input/Output – Pins 0-13 can be used for digital input or output

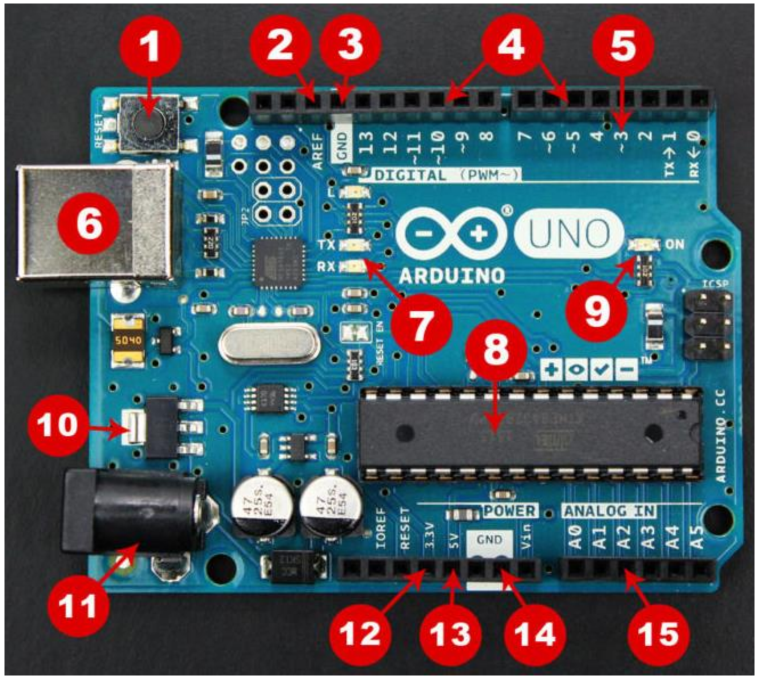
5.PWM – The pins marked with the (~) symbol can simulate analog output

6.USB Connection – Used for powering up your Arduino and uploading sketches

7.TX/RX – Transmit and receive data indication LEDs

8.ATmega Microcontroller – This is the brains and is where the programs are stored

Arduino Uno:



9. Power LED Indicator – This LED lights up anytime the board is plugged in a power source

10. Voltage Regulator – This controls the amount of voltage going into the Arduino board

11. DC Power Barrel Jack – This is used for powering your Arduino with a power supply

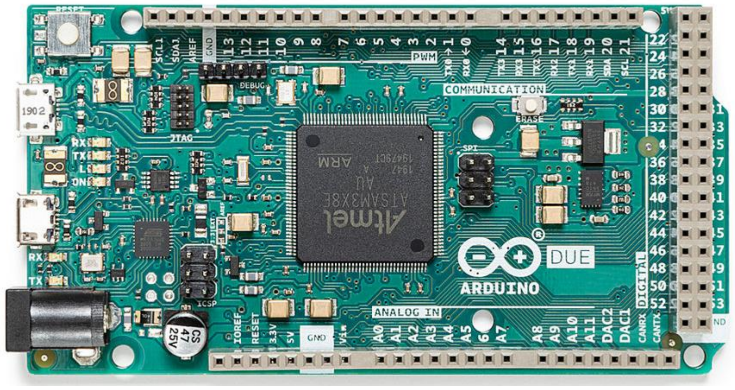
12. 3.3V Pin – This pin supplies 3.3 volts of power to your projects

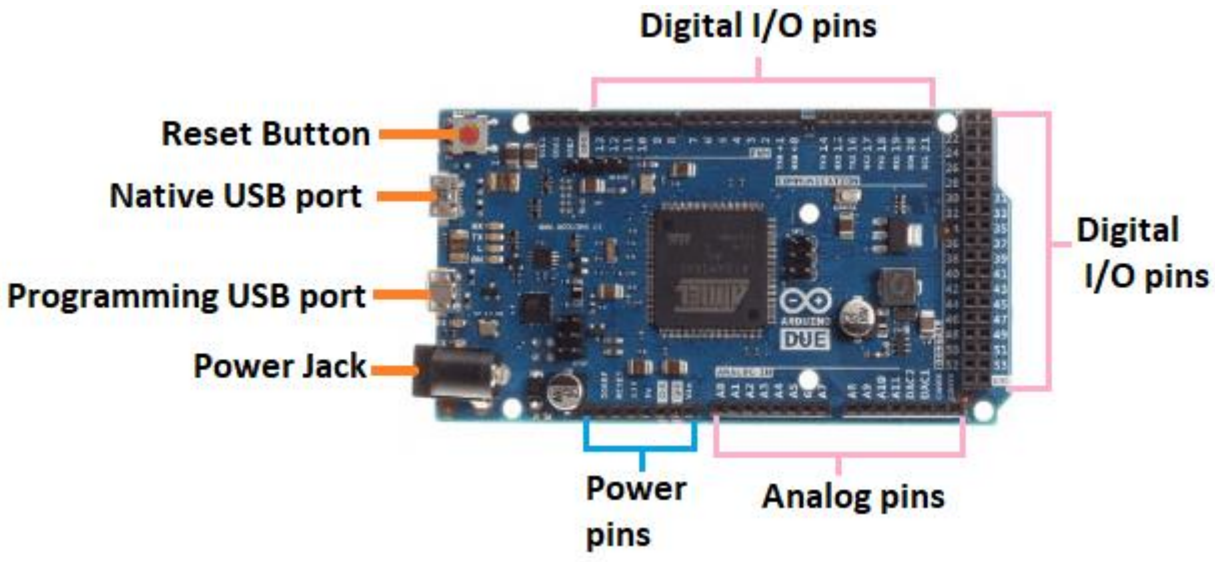
13. 5V Pin – This pin supplies 5 volts of power to your projects

14. Ground Pins – There are a few ground pins on the Arduino and they all work the same

15. Analog Pins – These pins can read the signal from an analog sensor and convert it to digital

Arduino Due:

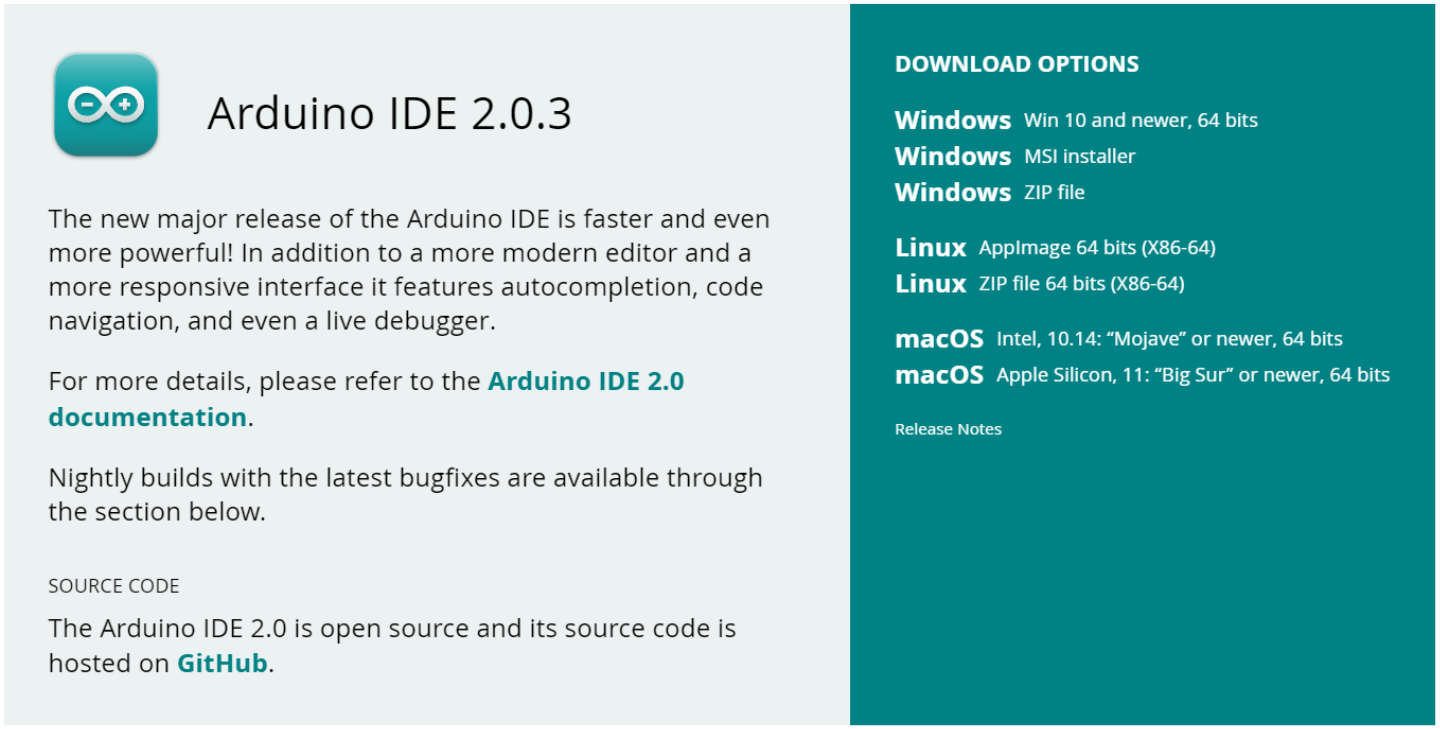




Digital I/O pins
Reset Button
Native USB port
Programming USB port
Power Jack
Digital I/O pins
Power pins
Analog pins

Installation of the Arduino IDE:

https://www.arduino.cc/en/software



Arduino IDE 2.0.3

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

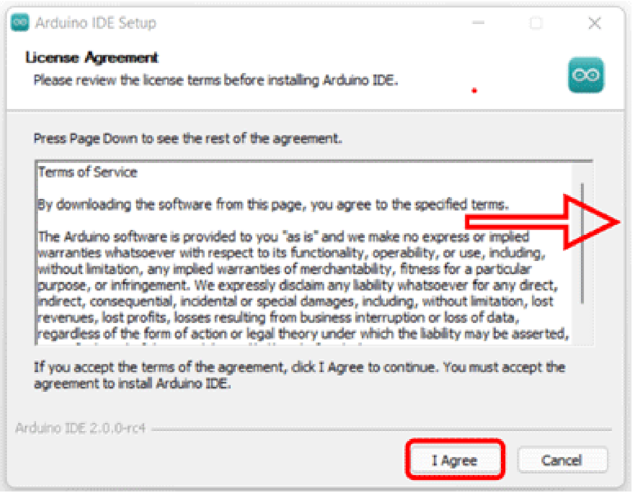
DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file

Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.14: "Mojave" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

Release Notes


Arduino IDE Setup

License Agreement

Please review the license terms before installing Arduino IDE.

Press Page Down to see the rest of the agreement.

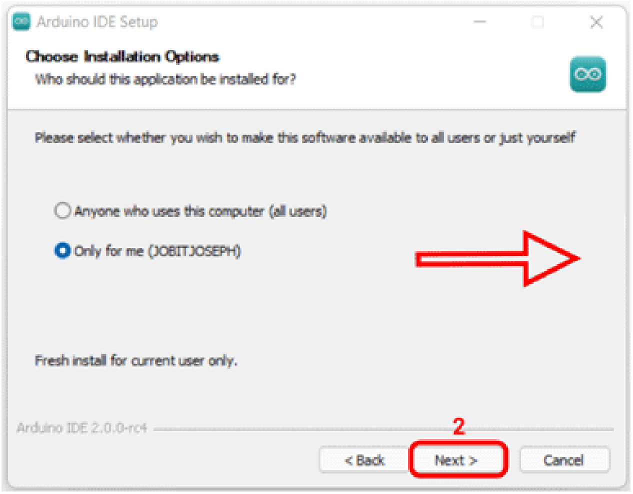
Terms of Service

By downloading the software from this page, you agree to the specified terms.

The Arduino software is provided to you "as is" and we make no express or implied warranties whatsoever with respect to its functionality, operability, or use, including, without limitation, any implied warranties of merchantability, fitness for a particular purpose, or infringement. We expressly disclaim any liability whatsoever for any direct, indirect, consequential, incidental or special damages, including, without limitation, lost revenues, lost profits, losses resulting from business interruption or loss of data, regardless of the form of action or legal theory under which the liability may be asserted,

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Arduino IDE.

Arduino IDE 2.0.0-rc4


Arduino IDE Setup

Choose Installation Options

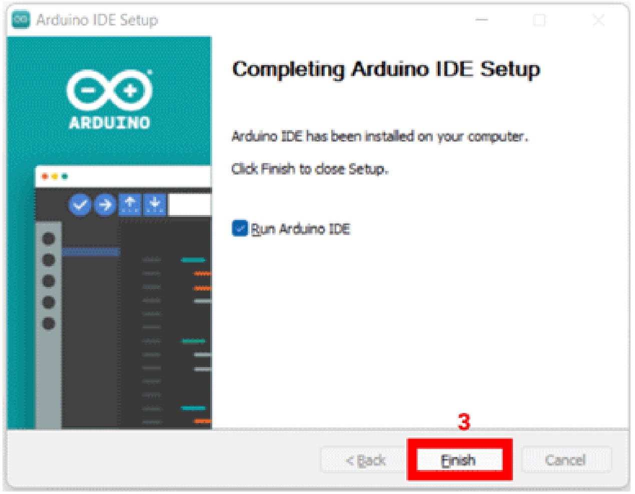
Who should this application be installed for?

Please select whether you wish to make this software available to all users or just yourself

Anyone who uses this computer (all users) [✓ Only for me (JOBITJOSEPH)]

Fresh install for current user only.

Arduino IDE 2.0.0-rc4


Arduino IDE Setup

Completing Arduino IDE Setup

Arduino IDE has been installed on your computer.

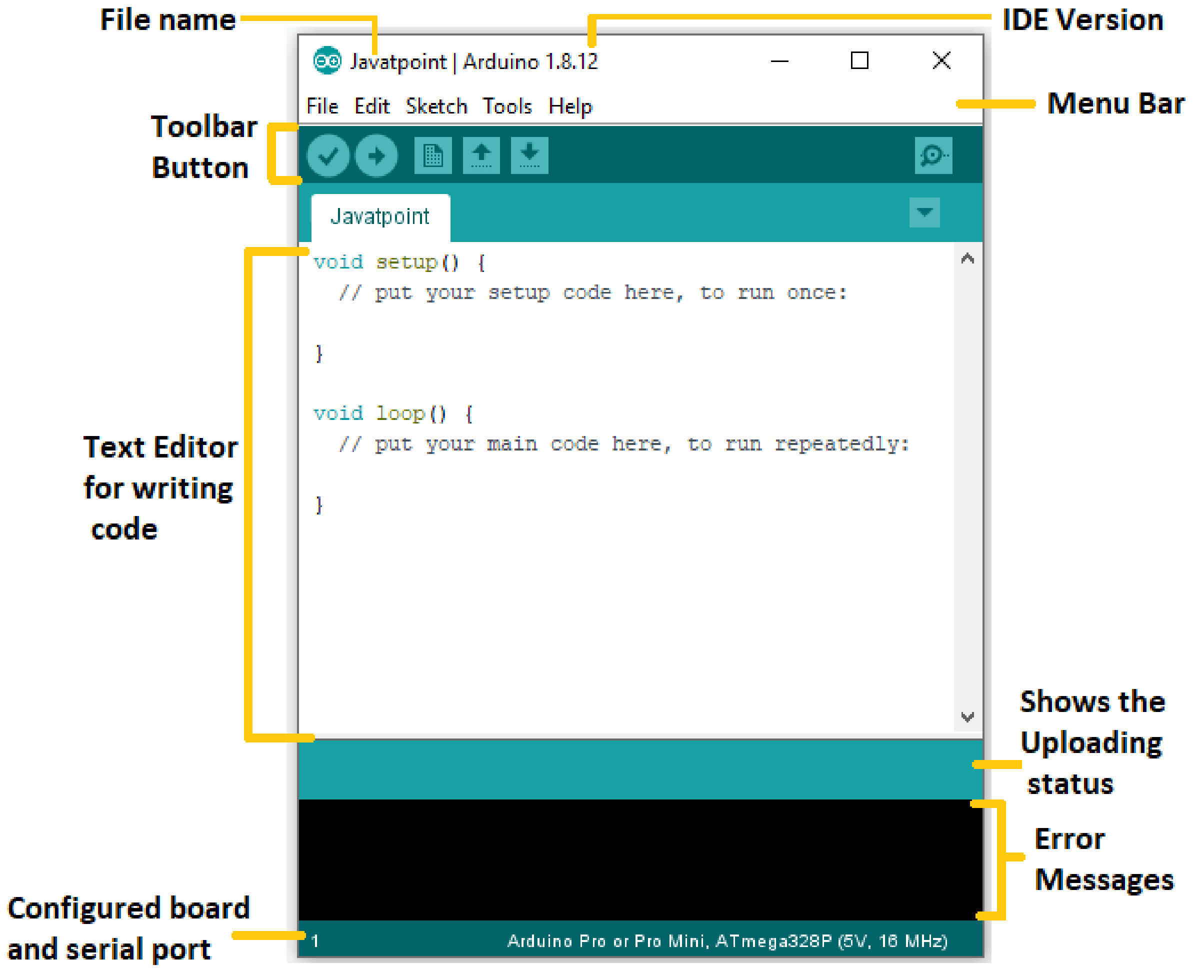
Click Finish to close Setup.

[✓ Run Arduino IDE]

https://www.arduino.cc/en/software

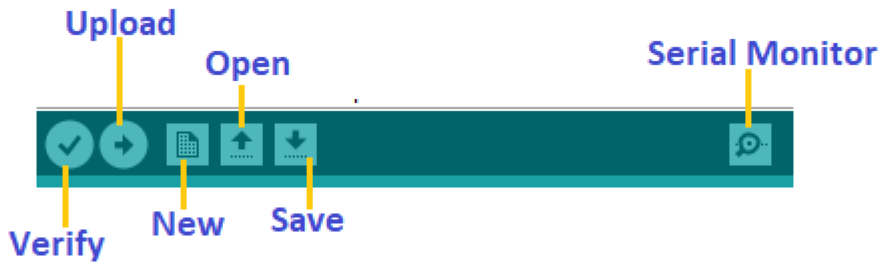
•Download the installer for your operating system from the above-given link.
•Once the download is done open the .exe file.
•Agree to the license agreement and select if the IDE should be installed to all users or not and click on "Next" to continue.
•Select the location in which you want the IDE to install if you want to change the location or keep it default and click on "Install"
•Wait for the installer to finish installation, and click on "Close".

Arduino IDE:

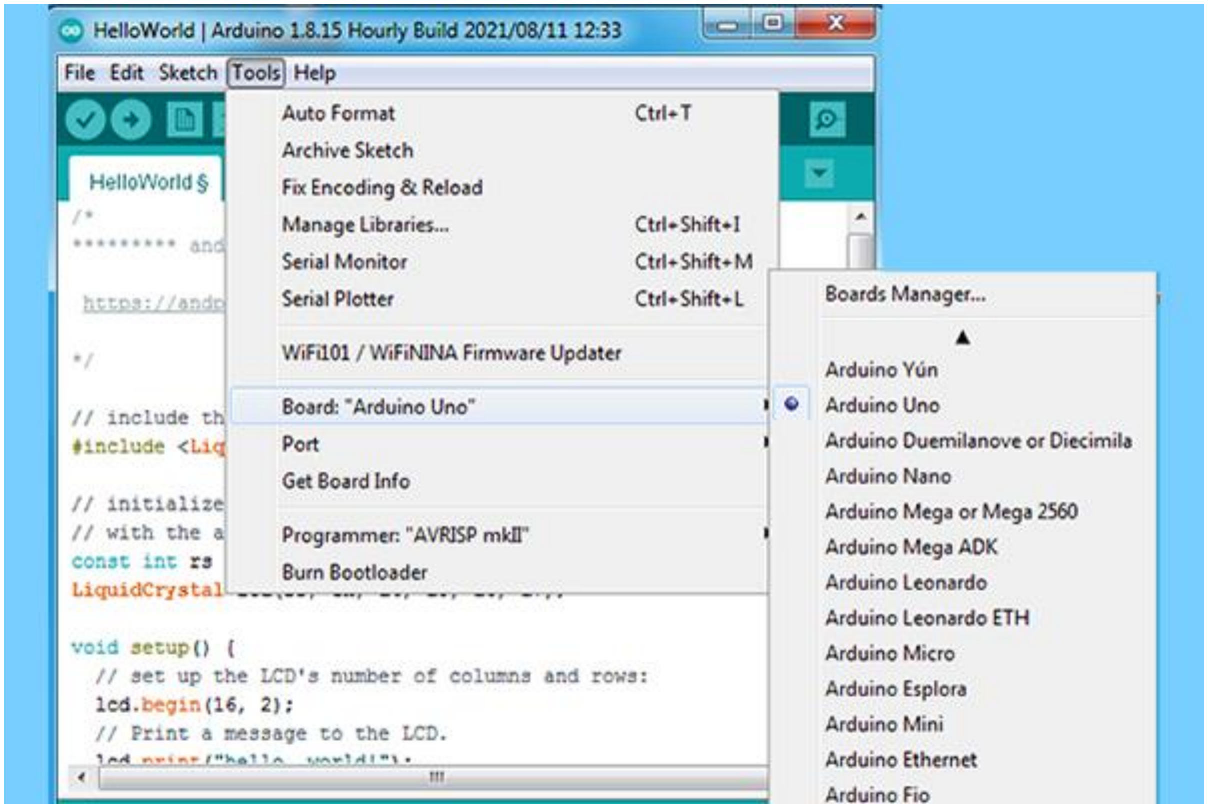

File name
IDE Version
Javatpoint | Arduino 1.8.12
File Edit Sketch Tools Help
Menu Bar
Toolbar Button
Javatpoint
Text Editor for writing code
Shows the Uploading status
Error Messages
Configured board and serial port
1
Arduino Pro or Pro Mini, ATmega328P (5V, 16 MHz)

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

}

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

}
```

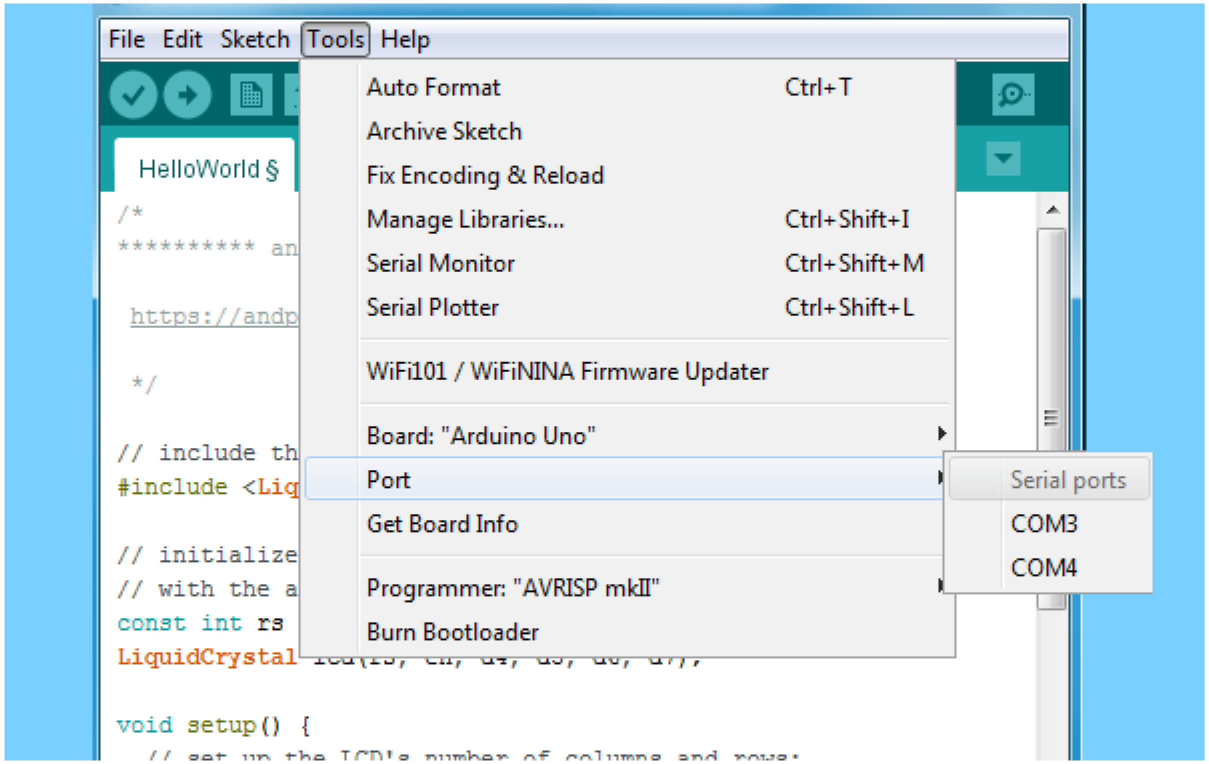
Toolbar:



1.Verify: this button use to review the code, or make sure that is free from mistakes.
2.Upload: this button is use to upload the code on the Arduino board.
3.New: this button use to create new project, or sketch (sketch is the file of the code).
4.Open: is use when you want to open the sketch from sketchbook.
5.Save: save the current sketch in the sketchbook.
6.Serial monitor: showing the data which have been sent from Arduino.

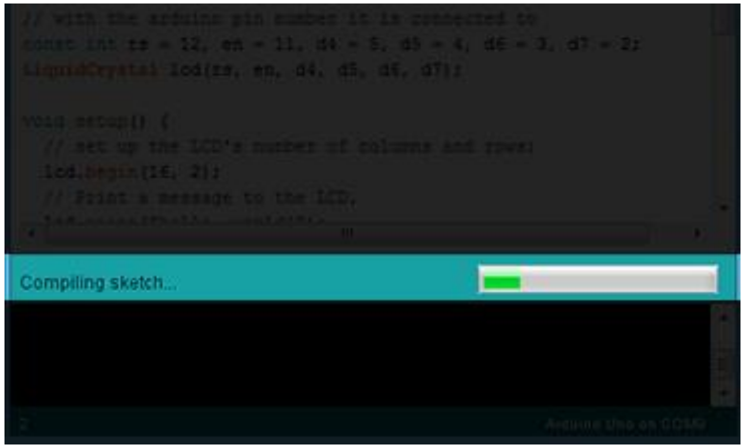

HelloWorld | Arduino 1.8.15 Hourly Build 2021/08/11 12:33
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino Yún
Arduino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo
Arduino Leonardo ETH
Arduino Micro
Arduino Esplora
Arduino Mini
Arduino Ethernet
Arduino Fio

```
/*
********* and
 https://andp
*/

// include th
#include <Liq

// initialize
// with the a
const int rs
LiquidCrystal

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
```


File Edit Sketch Tools Help
HelloWorld §
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM3
COM4

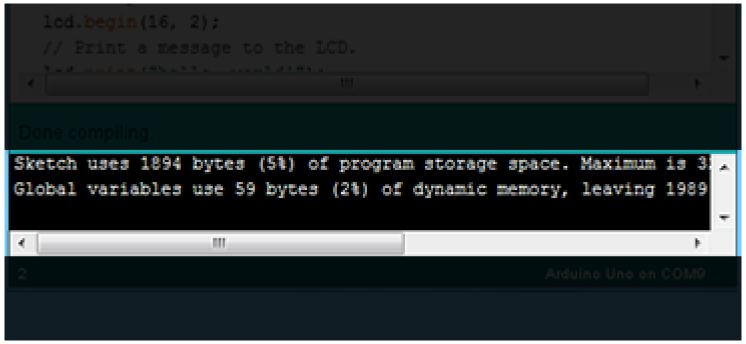
```
/*
********** an
 https://andp
 */

// include th
#include <Liq

// initialize
// with the a
const int rs
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
```

- First thing: in the menu we click on **"Tools"**, then we click on **"Board"** and we select **Arduino board** which you are using.
- Second: in the menu we click on **"Tools"** again, we click on **"Port"** and we select **Serial port** that we connected Arduino board with.
- Third: in **"Code editor"** we write the programming code, then we click on **"Verify"** to verify it correctness.
- Fourth: we click on **"Upload"** to upload the code on the Arduino board.



```
// with the arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
```

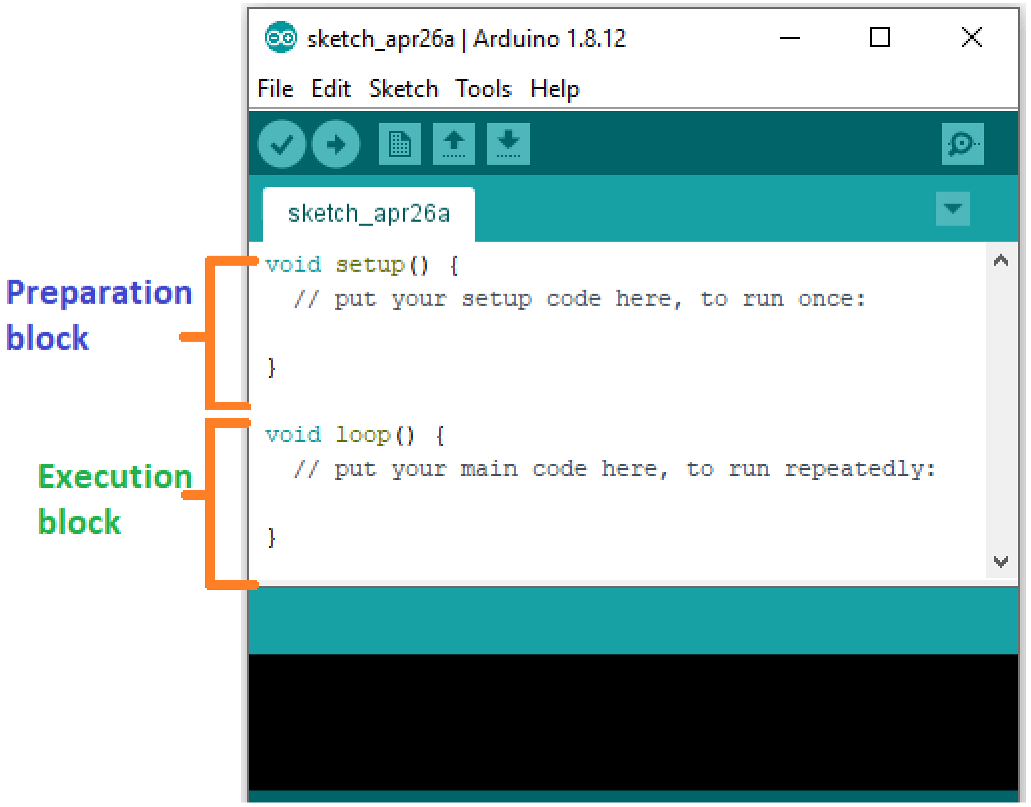
Compiling sketch...



```
  lcd.begin(16, 2);
  // Print a message to the LCD.
```

Done compiling.

```
Sketch uses 1894 bytes (5%) of program storage space. Maximum is 3
Global variables use 59 bytes (2%) of dynamic memory, leaving 1989
```



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

}

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

}
```

Brackets

There are two types of brackets used in the Arduino coding, which are listed below:

- Parentheses ( )
- Curly Brackets { }

Parentheses ( )

The parentheses brackets are the group of the arguments, such as method, function, or a code statement. These are also used to group the math equations.

Curly Brackets { }

The statements in the code are enclosed in the curly brackets. We always require closed curly brackets to match the open curly bracket in the code or sketch.

Open curly bracket- ' **{** '

Closed curly bracket - ' **}** '

Line Comment

There are two types of line comments, which are listed below:

- Single line comment
- Multi-line comment

// Single line comment

The text that is written after the two forward slashes are considered as a single line comment. The compiler ignores the code written after the two forward slashes. The comment will not be displayed in the output. Such text is specified for a better understanding of the code or for the explanation of any code statement.

The // (two forward slashes) are also used to ignore some extra lines of code without deleting it.

/ * Multi - line comment */

The Multi-line comment is written to group the information for clear understanding. It starts with the single forward slash and an asterisk symbol (**/ ***). It also ends with the **/ ***. It is commonly used to write the larger text. It is a comment, which is also ignored by the compiler.

The coding screen is divided into two blocks. The **setup** is considered as the preparation block, while the **loop** is considered as the execution block. It is shown below:

```
void setup ( )
{
Coding statement 1;
Coding statement 2;
.
.
.
Coding statement n;
}
void loop ( )
{
Coding statement 1;
Coding statement 2;
.
.
.
Coding statement n;
}
```

What is Setup? What type of code is written in the setup block?

It contains an initial part of the code to be executed. The **pin modes, libraries**, **variables**, etc., are initialized in the setup section. It is executed only **once** during the uploading of the program and after reset or power up of the Arduino board.
Zero setup () resides at the top of each sketch. As soon as the program starts running, the code inside the curly bracket is executed in the setup and it executes only once.

What is Loop? What type of code is written in the Loop block?

The loop contains statements that are executed repeatedly. The section of code inside the curly brackets is repeated depending on the value of variables.

Time in Arduino

The time in Arduino programming is measured in a millisecond.

Where, 1 sec = 1000 milliseconds

We can adjust the timing according to the milliseconds.

For example, for a 5-second delay, the time displayed will be 5000 milliseconds.

Time in Arduino

Example:

Let's consider a simple LED blink example.

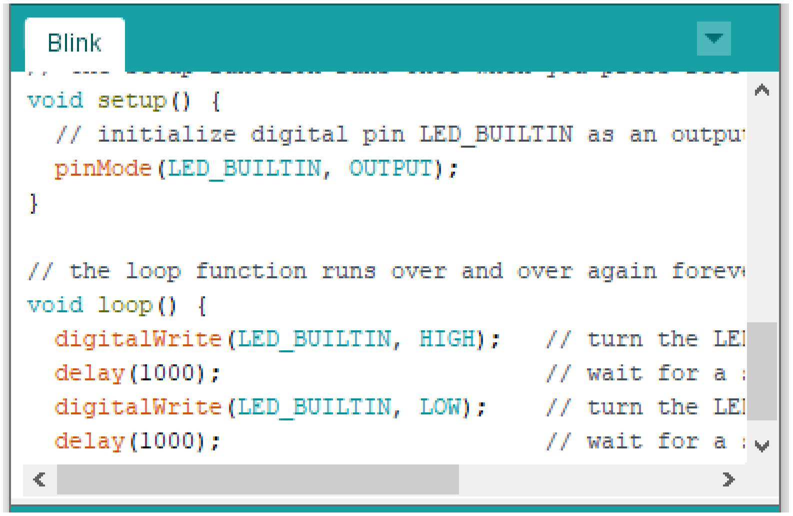
The steps to open such example are:

Click on the **File** button, which is present on the menu bar.

Click on the **Examples**.

Click on the **Basics** option and click on the **Blink**

The example will reopen in a new window, as shown below:



```
void setup() {
  // initialize digital pin LED_BUILTIN as an outpu
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forev
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LE
  delay(1000);                       // wait for a
  digitalWrite(LED_BUILTIN, LOW);    // turn the LE
  delay(1000);                       // wait for a
```

•The void setup () would include pinMode as the main function.

pinMode ( )

The specific pin number is set as the INPUT or OUTPUT in the pinMode () function.

The Syntax is: **pinMode (pin, mode)**

Where,

pin: It is the pin number. We can select the pin number according to the requirements.

Mode: We can set the mode as INPUT or OUTPUT according to the corresponding pin number.

Let' understand the pinMode with an example.

Example: We want to set the 12 pin number as the output pin.

Code:

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

Why is it recommended to set the mode of pins as OUTPUT?

The OUTPUT mode of a specific pin number provides a considerable amount of current to other circuits, which is enough to run a sensor or to light the LED brightly. The output state of a pin is considered as the low-impedance state.

The high current and short circuit of a pin can damage the ATmel chip. So, it is recommended to set the mode as OUTPUT.

Can we set the pinMode as INPUT?

The digitalWrite () will disable the LOW during the INPUT mode. The output pin will be considered as HIGH.

We can use the INPUT mode to use the external pull-down resistor. We are required to set the pinMode as INPUT_PULLUP. It is used to reverse the nature of the INPUT mode.

The sufficient amount of current is provided by the pull-up mode to dimly light an LED, which is connected to the pin in the INPUT mode. If the LED is working dimly, it means this condition is working out.

Due to this, it is recommended to set the pin in OUTPUT mode.

•The void loop () would include **digitalWrite( )** and **delay ( )** as the main function.

digitalWrite( )

The digitalWrite ( ) function is used to set the value of a pin as HIGH or LOW.
Where,
HIGH: It sets the value of the voltage. For the 5V board, it will set the value of 5V, while for 3.3V, it will set the value of 3.3V.
LOW: It sets the value = 0 (GND).
If we do not set the pinMode as OUTPUT, the LED may light dim.
The syntax is: **digitalWrite(pin, value HIGH/LOW)**
pin: We can specify the pin number or the declared variable.

Let's understand with an example.

Example:

```
digitalWrite (13, HIGH);
digitalWrite (13, LOW);
```

The HIGH will ON the LED and LOW will OFF the LED connected to pin number 13.

What is the difference between digitalRead () and digitalWrite ()?
The digitalRead () function will read the HIGH/LOW value from the digital pin, and the digitalWrite () function is used to set the HIGH/LOW value of the digital pin.

delay ( )
The delay () function is a blocking function to pause a program from doing a task during the specified duration in milliseconds.
For example, - delay (2000)
Where, 1 sec = 1000millisecond
Hence, it will provide a delay of 2 seconds.

Code:
```
digitalWrite (13, HIGH);
delay (2000);
digitalWrite (13, LOW);
delay (1000);
```

Here, the LED connected to pin number 13 will be ON for 2 seconds and OFF for 1 second. The task will repeatedly execute as it is in the void loop ().

We can set the duration according to our choice or project requirements.

Example: To light the LED connected to pin number 13. We want to ON the LED for 4 seconds and OFF the LED for 1.5 seconds.

Code:

```
void setup ()
{
pinMode ( 13, OUTPUT);  // to set the OUTPUT mode of pin number 13.
}
void loop ()
{
digitalWrite (13, HIGH);
delay (4000);  // 4 seconds = 4 x 1000 milliseconds
digitalWrite (13, LOW);
delay (1500);  // 1.5 seconds = 1.5 x 1000 milliseconds
}
```

Arduino Serial | Serial.begin()

Serial Communication

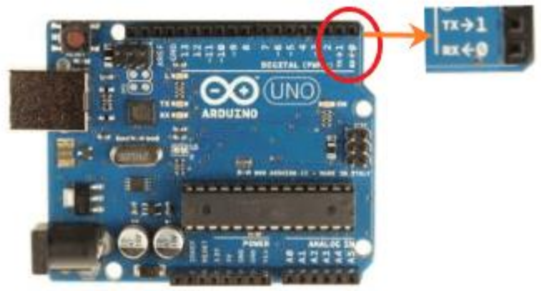
The serial communication is a simple scheme that uses the **UART** (Universal Asynchronous Receiver/Transmitter) on the Microcontroller. It uses,

5V for logic 1 (high)
0V for logic 0 (low)

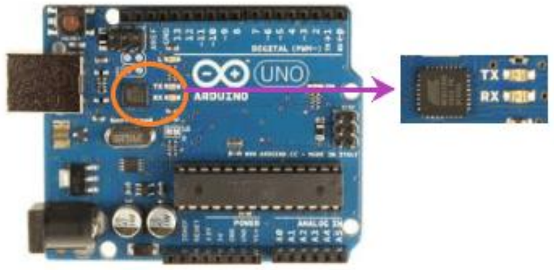
For a 3.3V board, it uses
3V for logic 1 (high)
0V for logic 0 (low)

Every message sent on the UART is in the form of 8 bits or 1 byte, where **1 byte = 8 bits.** The messages sent to the computer from Arduino are **sent from PIN 1 of the Arduino board, called Tx (Transmitter)**. The messages being sent to the Arduino from the computer are **received on PIN 0, called Rx** (**Receiver**).

These two pins on the Arduino UNO board look like the below image:



When we initialize the pins for serial communication in our code, we cannot use these two pins (Rx and Tx) for any purpose. The Tx and Rx pins are also connected directly to the computer.
The pins are connected to the serial Tx and Rx chip, which acts as a serial to USB translator.
It acts as a medium for the computer to talk to the Microcontroller.
The chip on the board looks like the below image:



The object can include any number of data members (information) and member functions (to call actions).

The **Serial.begin( )** is a part of the serial object in the Arduino. It tells the serial object to perform initialization steps to send and receive data on the Rx and Tx (pins 1 and 0).

Serial.begin ( )

The serial.begin( ) *sets the baud rate for serial data communication*. The **baud** rate signifies **the data rate in bits per second.**

The default baud rate in Arduino is **9600 bps (bits per second**). We can specify other baud rates as well, such as 4800, 14400, 38400, 28800, etc.

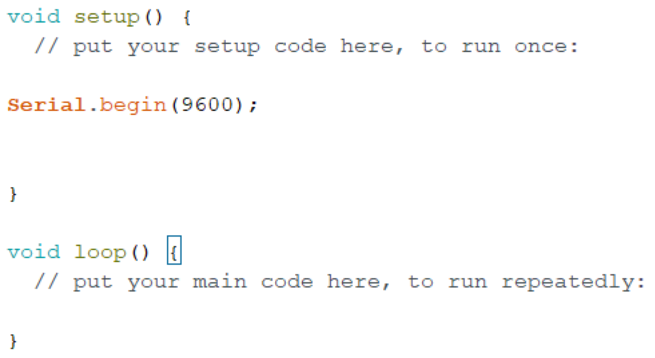
The Serial.begin( ) is declared in two formats, which are shown below:

•begin(speed)
•begin(speed, config)

Where,

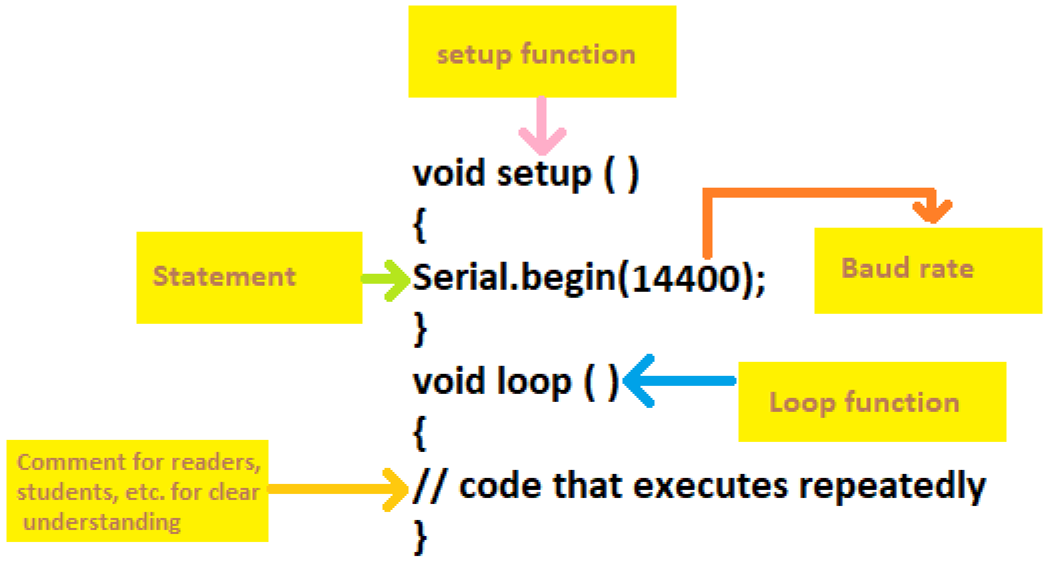
serial: It signifies the serial port object.
speed: It signifies the baud rate or bps (bits per second) rate. It allows ***long*** data types.
config: It sets the stop, parity, and data bits.

```
void setup ( )
{
Serial.begin(4800);
}
void loop ( )
{
}
```



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

Serial.begin(9600);


}

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

}
```

- ✓ The serial.begin (4800) open the serial port and set the bits per rate to 4800. The messages in Arduino are interchanged with the serial monitor at a rate of 4800 bits per second.

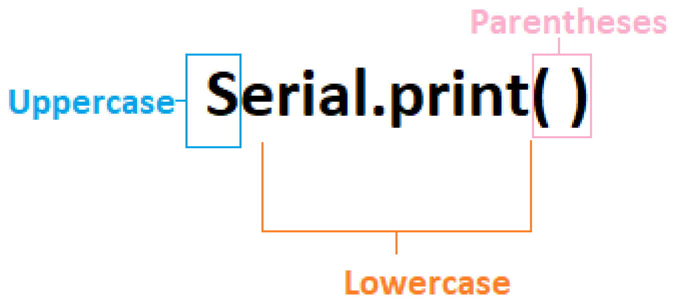


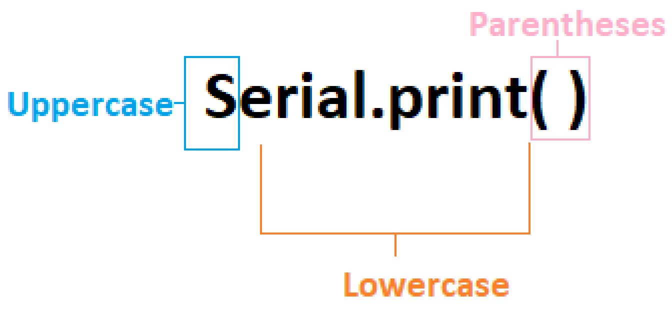
```
void setup ( )
{
Serial.begin(14400);
}
void loop ( )
{
// code that executes repeatedly
}
```

Serial.print ( )

The serial.print ( ) in Arduino prints the data to the serial port. The printed data is stored in the ASCII (American Standard Code for Information Interchange) format, which is a human-readable text.
Each digit of a number is printed using the ASCII characters.
The printed data will be visible in the **serial monitor**, which is present on the right corner on the toolbar.
The Serial.print( ) is declared in two formats, which are shown below:

- print(value)
- print(value, format)





Where,

serial: It signifies the serial port object.

print: The print ( ) returns the specified number of bytes written.

value: It signifies the value to print, which includes any data type value.

format: It consists of number base, such as OCT (Octal), BIN (Binary), HEX (Hexadecimal), etc. for the integral data types. It also specifies the number of decimal places.

Serial.print(value)

The serial.print ( ) accepts the number using the ASCII character per digit and value up to two decimal places for floating point numbers.

```
Serial.print(15.452732)
Output: 15.45
```

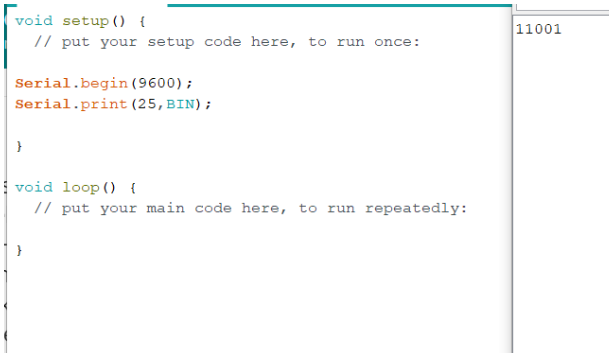
It sends bytes to the printer as a single character. In Arduino, the strings and characters using the Serial.print( ) are sent as it is.

```
Serial.print("Hello Arduino")
Output: "Hello Arduino"
```

Serial.print(value, format)

It specifies the base format and gives the output according to the specified format. It includes the formats Octal -OCT (base 8), Binary-BIN (base 2), Decimal-DEC (base 10), and Hexadecimal-HEX (base 16).
Let's understand by few examples.

Serial.print(25, BIN)
Output:
11001
It converts the decimal number 25 to binary number 11001.



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

Serial.begin(9600);
Serial.print(25,BIN);

}

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

}
```

```
11001
```

Serial.print(58, HEX)
Output:
3A
It converts the decimal number 58 to hexadecimal number 3A.

```
Serial.print(58, OCT)
```

Output: 72

It converts the decimal number 58 to octal number 72.

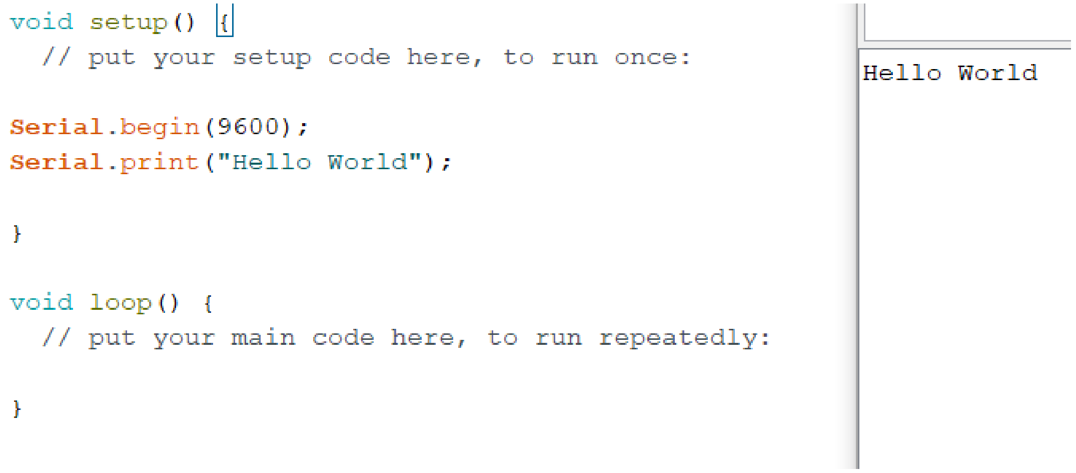
```
Serial.print(25, DEC)
```

Output:

25

The conversion is from decimal to decimal. So, the output will be the same.

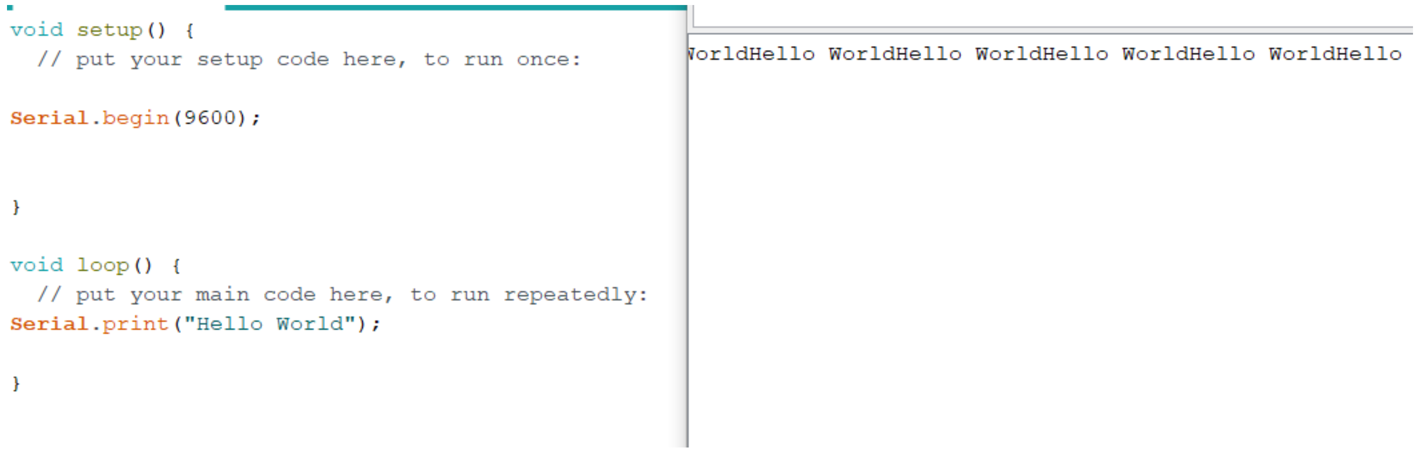
```
void setup() {
  // put your setup code here, to run once:

Serial.begin(9600);
Serial.print("Hello World");

}

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

}
```



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

Serial.begin(9600);
Serial.print("Hello World");

}

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

}
```

```
Hello World
```

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

Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
Serial.print("Hello World");

}
```



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

Serial.begin(9600);


}

void loop() {
  // put your main code here, to run repeatedly:
Serial.print("Hello World");

}
```

```
JorldHello WorldHello WorldHello WorldHello WorldHello
```

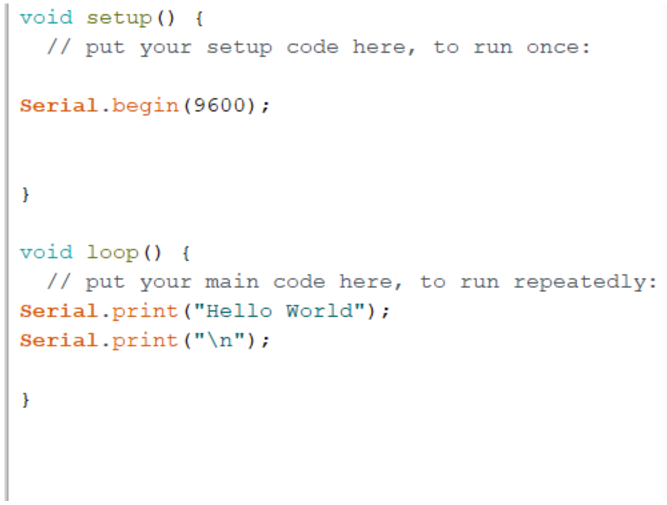
Printing a Tab space

We can also print the tab in the output.
Let's consider the code below:

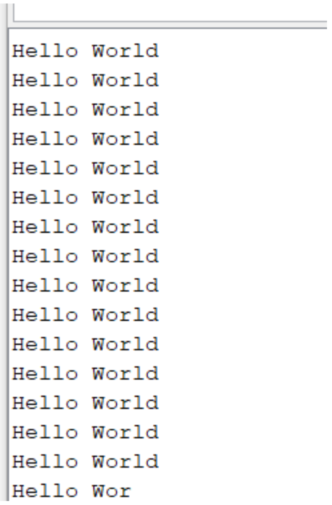
```
void setup ( )
{
Serial.begin ( 4800);
}
void loop ( )
{
Serial.print(" Hello Arduino" );
Serial.print(" \ t ");
}
```

Here, Serial.print(" \ t ") is used to print the tab in the output program.

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

Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.print("Hello World");
Serial.print("\n");

}
```



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

Serial.begin(9600);


}

void loop() {
  // put your main code here, to run repeatedly:
Serial.print("Hello World");
Serial.print("\n");

}
```



```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello Wor
```

Serial.println ( )

The Serial.println ( ) means print line, which sends the string followed by the carriage return ('\r' or ASCII 13) and newline ('\n' or ASCII 10) characters. It has a similar effect as pressing the Enter or Return key on the keyboard when typing with the Text Editor.

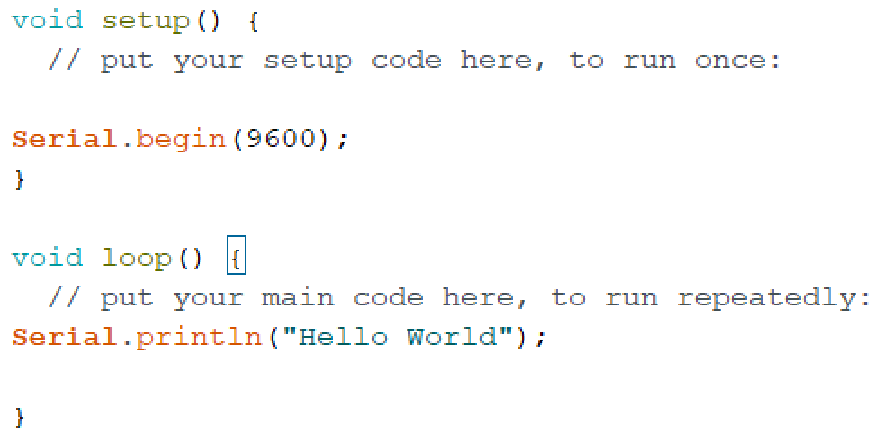
The Serial.println( ) is also declared in two formats, which are shown below:

- println(value)
- println(value, format)

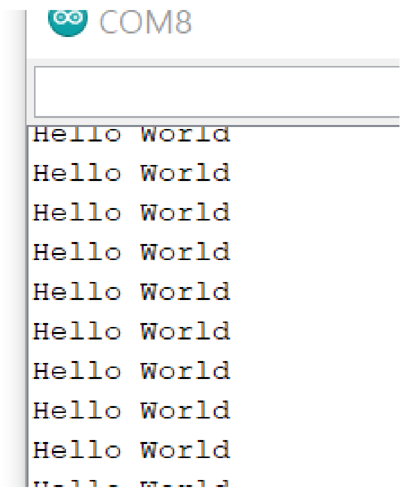
What is the difference between Serial.print( ) and Serial.println( )?

The text written inside the open and closed parentheses in the Serial.println( ) moves in a new line. With the help of Serial.print( ) and Serial.println( ), we can figure the order and execution of certain things in our code.

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

Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.println("Hello World");
}
```



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

Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.println("Hello World");

}
```



COM8

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

```
void setup ( )
{
Serial.begin ( 4800);
}
void loop ( )
{
Serial.print(" Hello");
delay(1000);
Serial.println("Arduino");  // It will print Arduino followed by a new line.
delay ( 1500);  // delay of 1.5 seconds between each printed line.
}
```

Click on the **Upload** button-> **Serial monitor** for the output.
In the output, the word **Hello** will appear followed by the word **Arduino** 1 second later. After 1.5 second, another line will be printed.

```
void setup()
{
  Serial.begin(9600);
}
void loop() {
  int a = 5; // initialization of values to the variables a and b
  int b = 4;
  int c;
  c = myAddfunction(a, b); // c will now contains the value 9
  Serial.println(c); // to print the resulted value
  delay(1000); // time delay of 1 second or 1000 milliseconds
}
int myAddfunction(int i, int j)
{
  int sum;
  sum = i + j;
  return sum;
}
```

- ✓ we will create a function that determines if a number is even or odd.

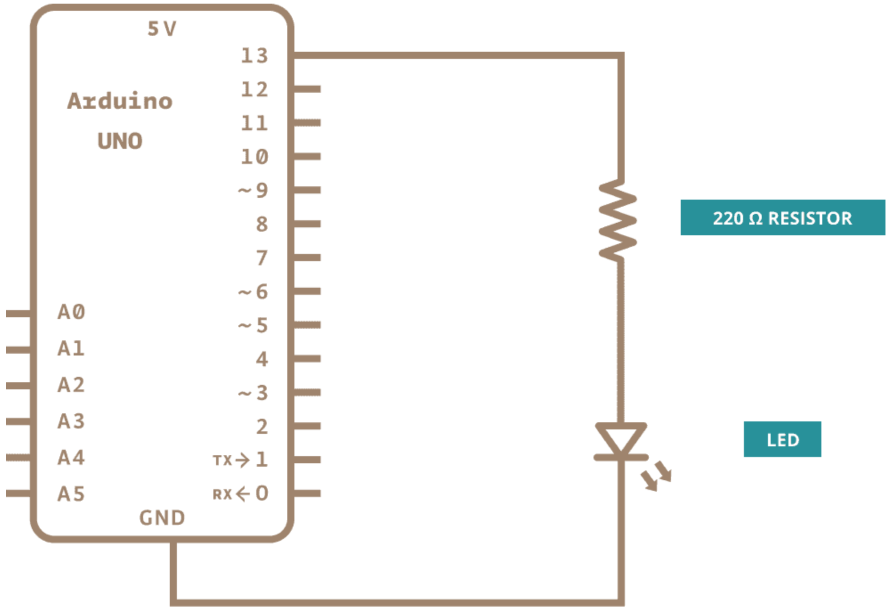
```
int a= 0;
int b;
void setup()
{
 Serial.begin(9600);
}
void loop()
{
b = Evenfunction(a); // we can store the function return value in variable b
 Serial.print(a);
 Serial.print(" : "); // to separate even or odd text
 if (b==1)
 {
  Serial.println( " Number is even");
 }
  else
  {
   Serial.println("Number is odd");
  }

   a++; // the function will increment and will again run
           delay(1000);
}

 int Evenfunction(int d)
            {
            if (d% 2==0)
            {
             return 1;
            }
            else
            {
             return 0;
            }
           }
```

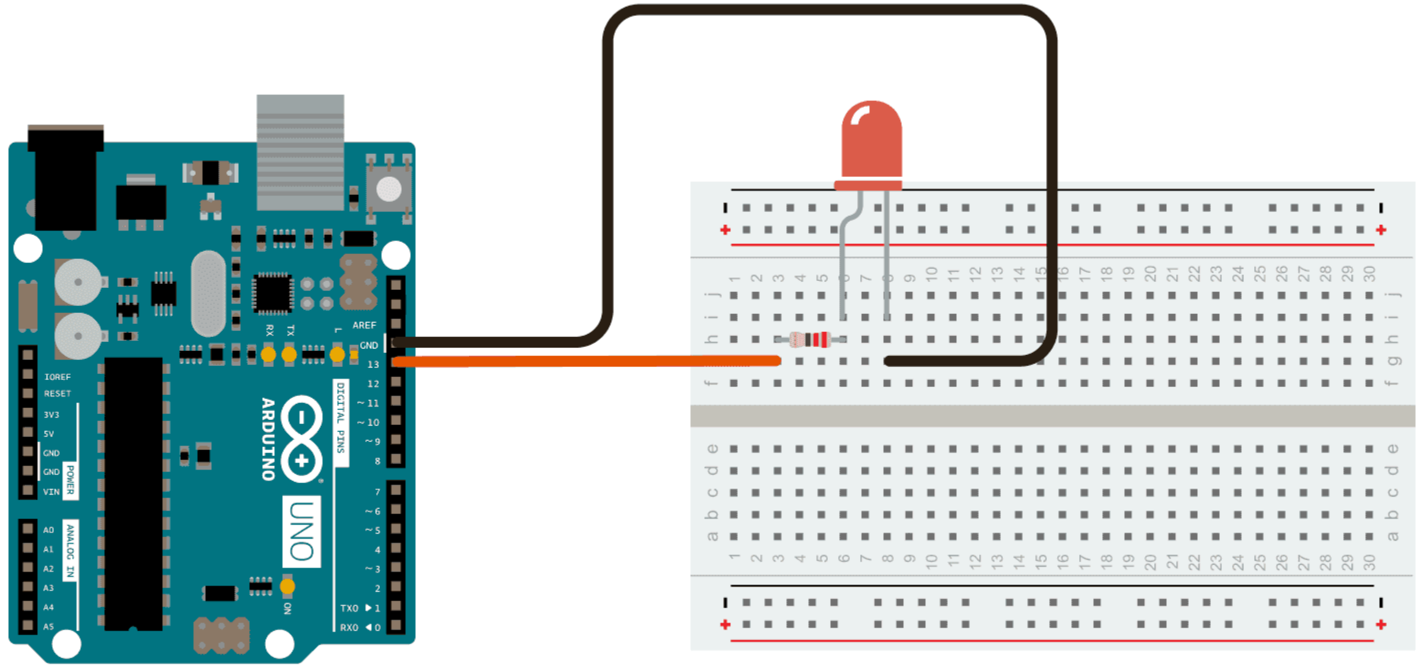
Blinking LED:

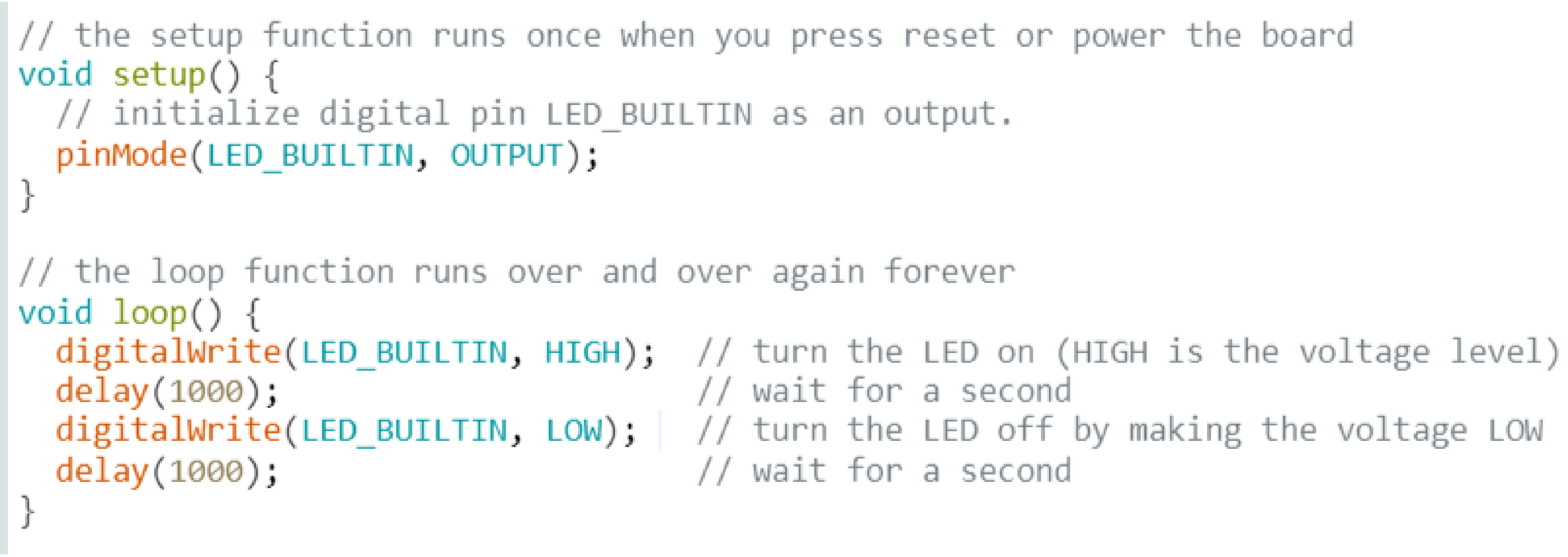
Components Required

You will need the following components:

- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × LED
- 1 × 330Ω Resistor
- 2 × Jumper







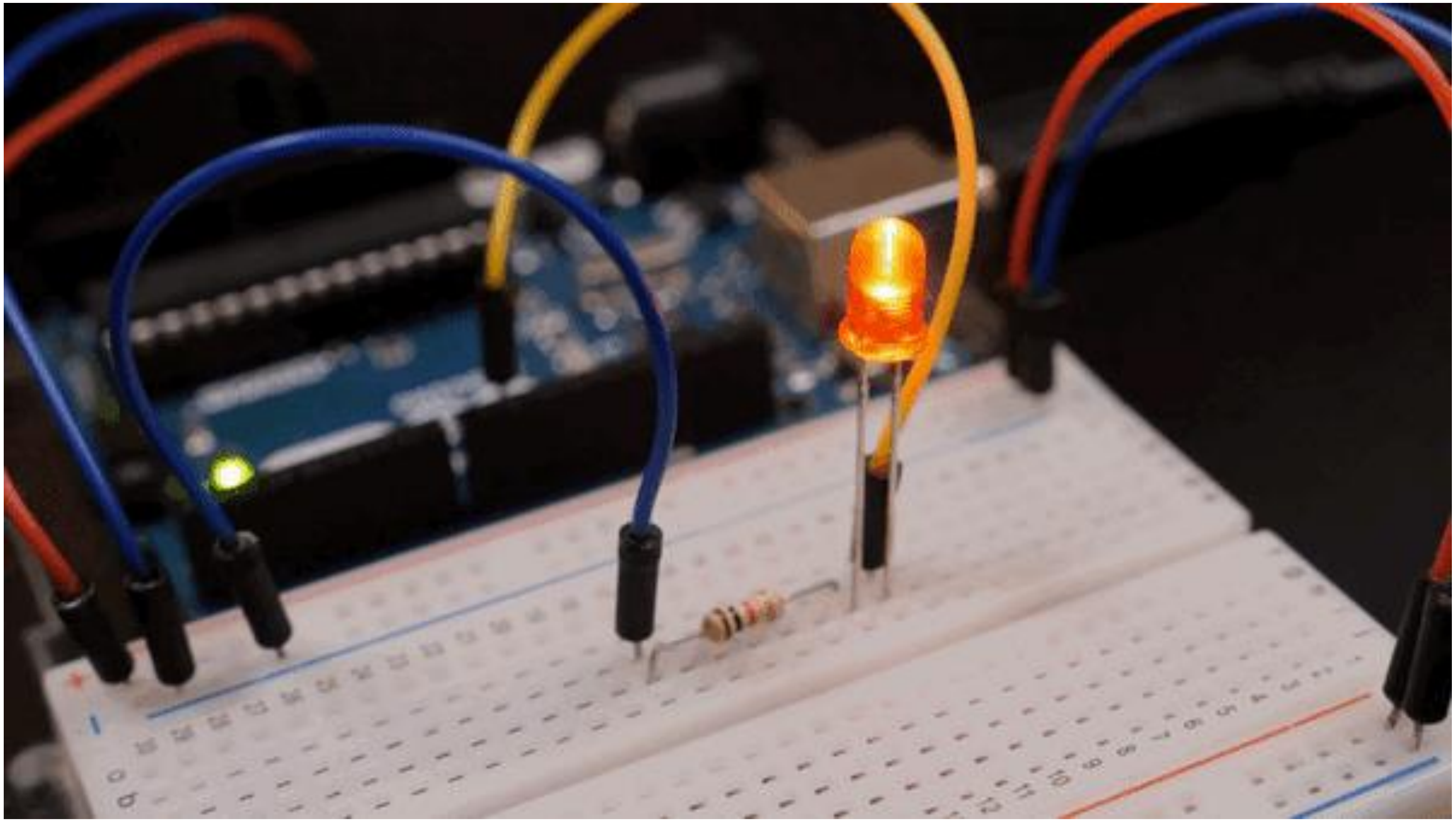
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

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

```
void setup ()
{
pinMode ( 13, OUTPUT);  // to set the OUTPUT mode of pin number 13.
}
void loop ()
{
digitalWrite (13, HIGH);
delay(1000);  // 1 second = 1 x 1000 milliseconds
digitalWrite (13, LOW);
delay(500);  // 0.5 second = 0.5 x 1000 milliseconds
}
```

```
int led = 13;
void setup() {

pinMode(led, OUTPUT);

}

void loop() {

digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
}
```

```
int led = 7;
int del = 1000;

void setup() {

pinMode(led, OUTPUT);

}
void loop() {

digitalWrite(led, HIGH);
delay(del);
digitalWrite(led, LOW);
delay(del);

}
```

Fading LED



- ✓ This example shows how to fade an LED on pin 13 using the analogWrite() function.
- ✓ The analogWrite() function uses PWM, so if you want to change the pin you're using, be sure to use another PWM capable pin. On most Arduinos, the PWM pins are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.
- ✓ AnalogWrite uses pulse width modulation (PWM), turning a digital pin on and off very quickly with different ratio between on and off, to create a fading effect.
- ✓ Connect the **anode** (the longer, positive leg) of your LED to digital output pin 13 on your board through a 220 ohm resistor. Connect the **cathode** (the shorter, negative leg) directly to ground.

Fading LED

```
int brightness = 0;

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

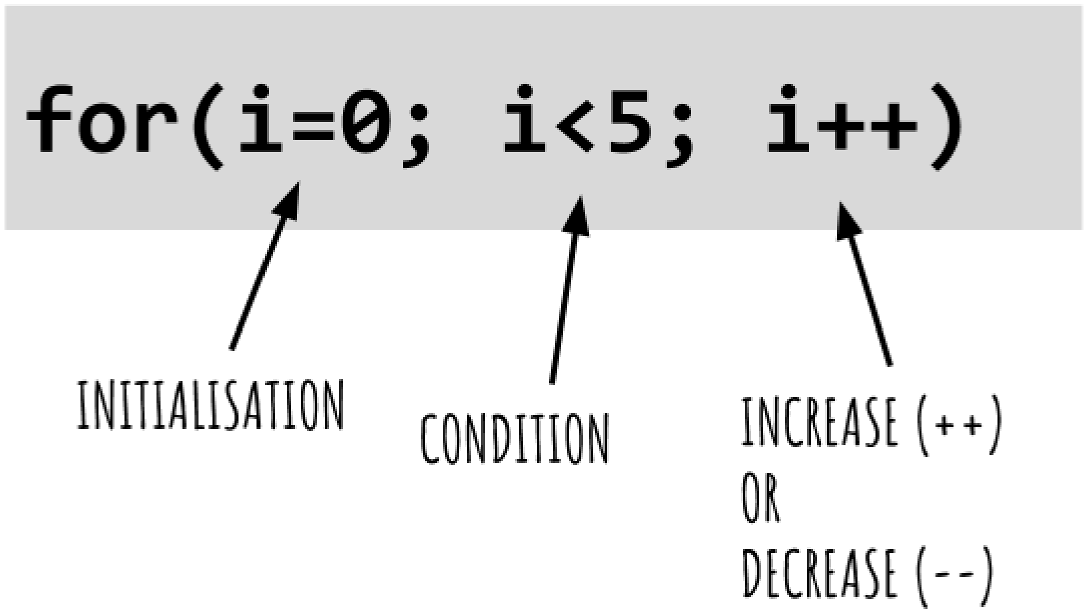
- ✓ The main body of the program starts out by creating a variable called brightness and sets it equal to zero, then inside the setup() pin 13 is initialized as an output.

- ✓ The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what PWM value to write.

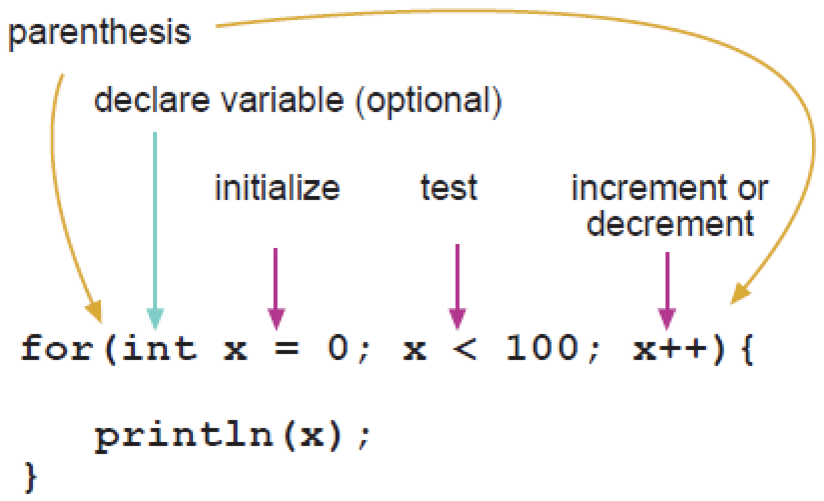
- ✓ In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the sketch below, the PWM value is set using a variable called brightness. Each time through the loop, it increases by the value of the variable fadeAmount.

- ✓ If brightness is at either extreme of its value (either 0 or 255), then fadeAmount is changed to its negative. In other words, if fadeAmount is 5, then it is set to -5. If it's -5, then it's set to 5. The next time through the loop, this change causes brightness to change direction as well.

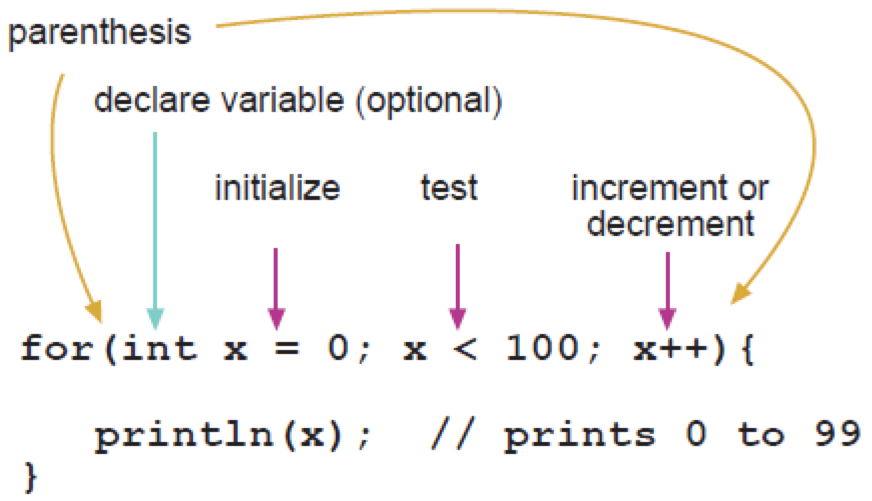
- ✓ analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the fading effect.



```
for(i=0; i<5; i++)
```



```
for(int x = 0; x < 100; x++){
   println(x);
}
```



```
for(int x = 0; x < 100; x++){
   println(x);  // prints 0 to 99
}
```

```
void loop()
{
  for (brightness = 0; brightness <= 255; brightness += 5) {
    analogWrite(13, brightness);
    delay(30); // Wait for 30 millisecond(s)
  }
  for (brightness = 255; brightness >= 0; brightness -= 5) {
    analogWrite(13, brightness);
    delay(30); // Wait for 30 millisecond(s)
  }
}
```

✓ The program's loop uses two for loops to count up from 0 to 255 by increments of 5. The analogWrite() function takes two arguments: the Arduino pin number (13 in our case), and a value between 0 (off) and 255 (all the way on).

```
int brightness = 0;

void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  for (brightness = 0; brightness <= 255; brightness += 5) {
    analogWrite(13, brightness);
    delay(30); // Wait for 30 millisecond(s)
  }
  for (brightness = 255; brightness >= 0; brightness -= 5) {
    analogWrite(13, brightness);
    delay(30); // Wait for 30 millisecond(s)
  }
}
```