Chapter 1

ARDUINO

(Working with A PC)

Learning Outcomes

When you have completed this programme you will be able to:

- Have an overview knowledge on Arduino
- Download and Install the Arduino IDE
- Perform a Demo Test (Arduino Board Check)
- Understand the basic interface on the Arduino IDE
- Perform Blink: First Arduino Code
- Perform Blink LED Project

Introduction

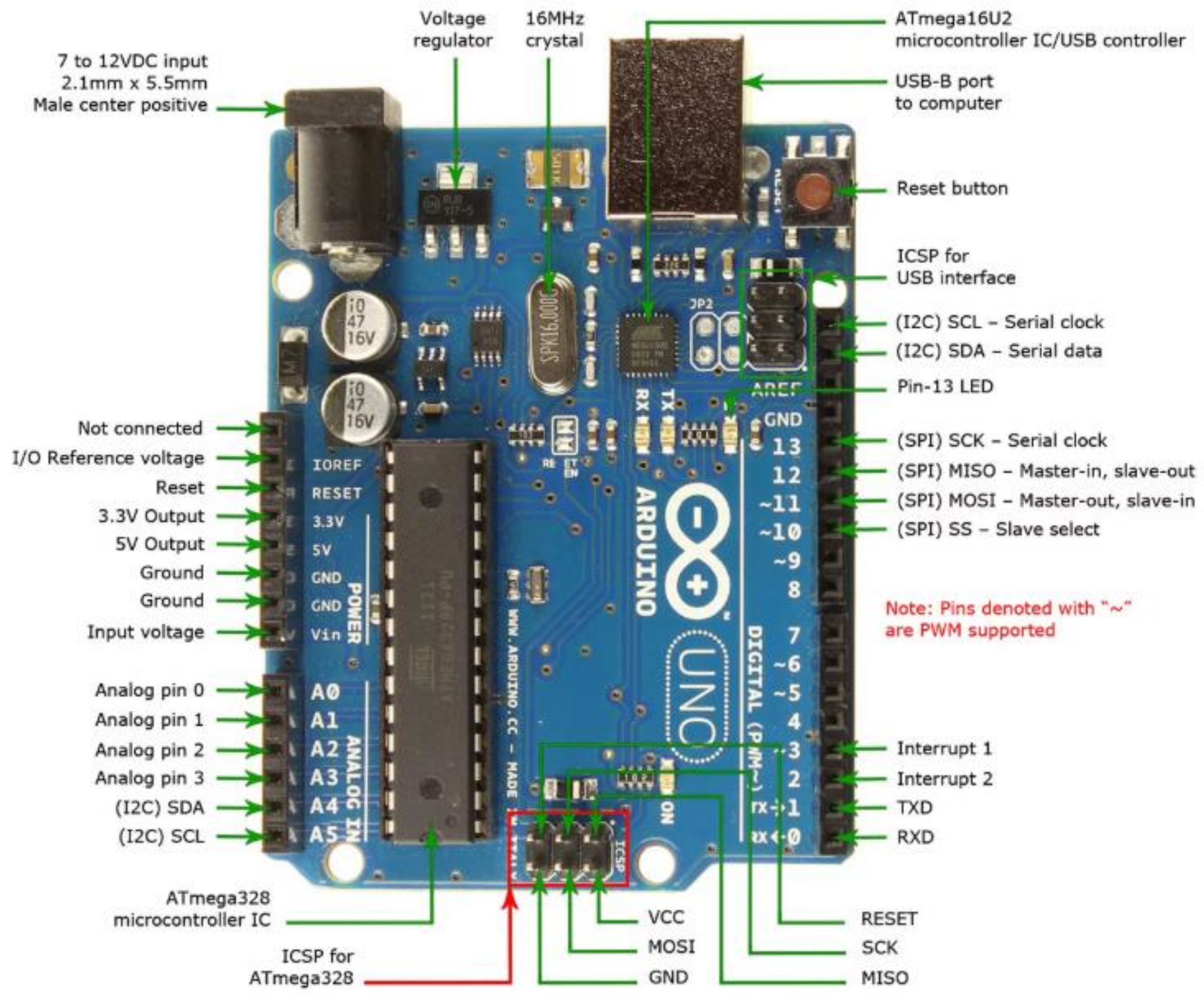
In this programme we shall be discussing in detail the **Arduino IDE and Module** where IDE stands for Integrated Development Environment.

The Arduino IDE is the official native platform by Arduino.cc which runs on both C and C++ programming languages which is used for writing, compiling and uploading functional codes into your Arduino module.
Arduino is an open-source prototyping 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. Awesome right?

To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. It is intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments."

Your imagination is your limit in building with Arduino, so let us not waste too much time talking about how powerful this module can be.

Firstly, we will take a look at the schematics of an Arduino Uno, which is the most common type of Arduino board.



Arduino Uno

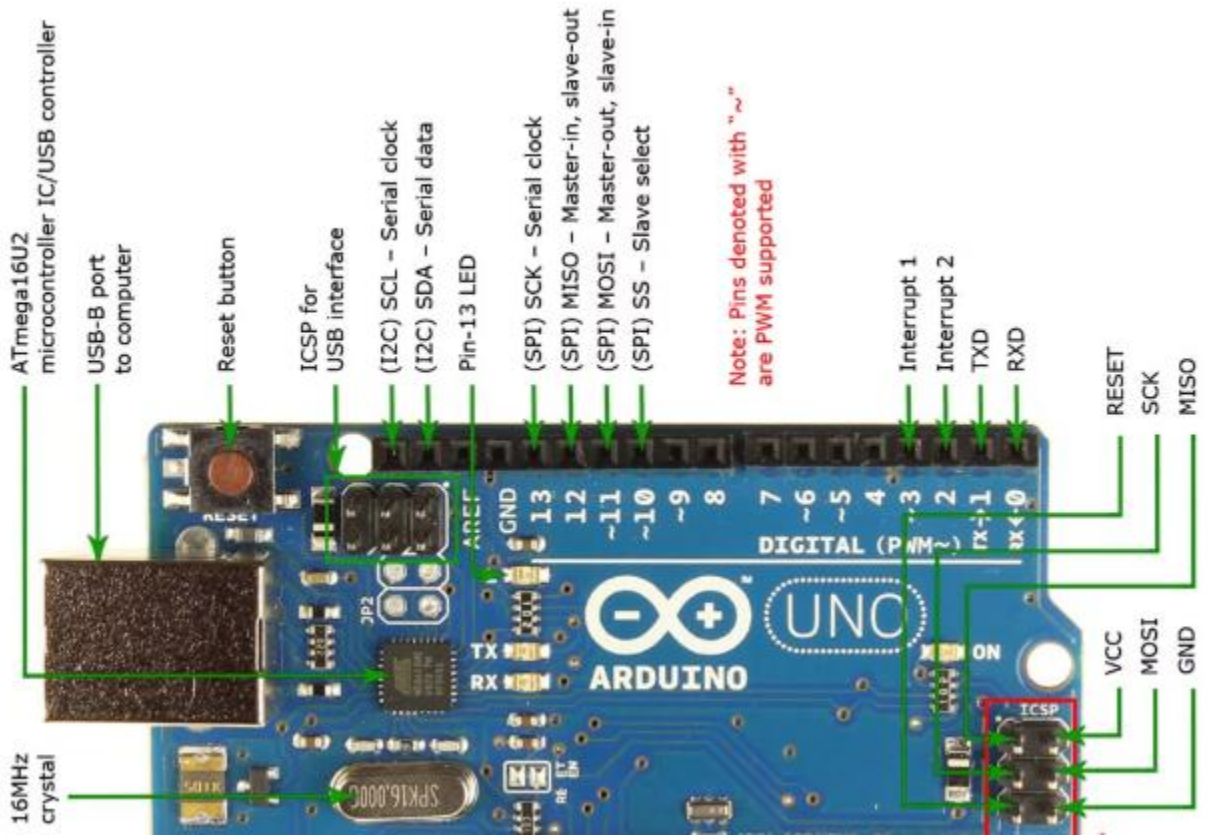
I understand that it looks like a city's aerial perspective, but do not panic I will explain the important labelled features for adequate comprehension.

Overview of Board

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet).

A data sheet, data sheet, or spec sheet is a document that summarizes the performance and other technical characteristics of a product, machine and component, or software in sufficient detail to enable design engineers to understand the component's role in the overall system.

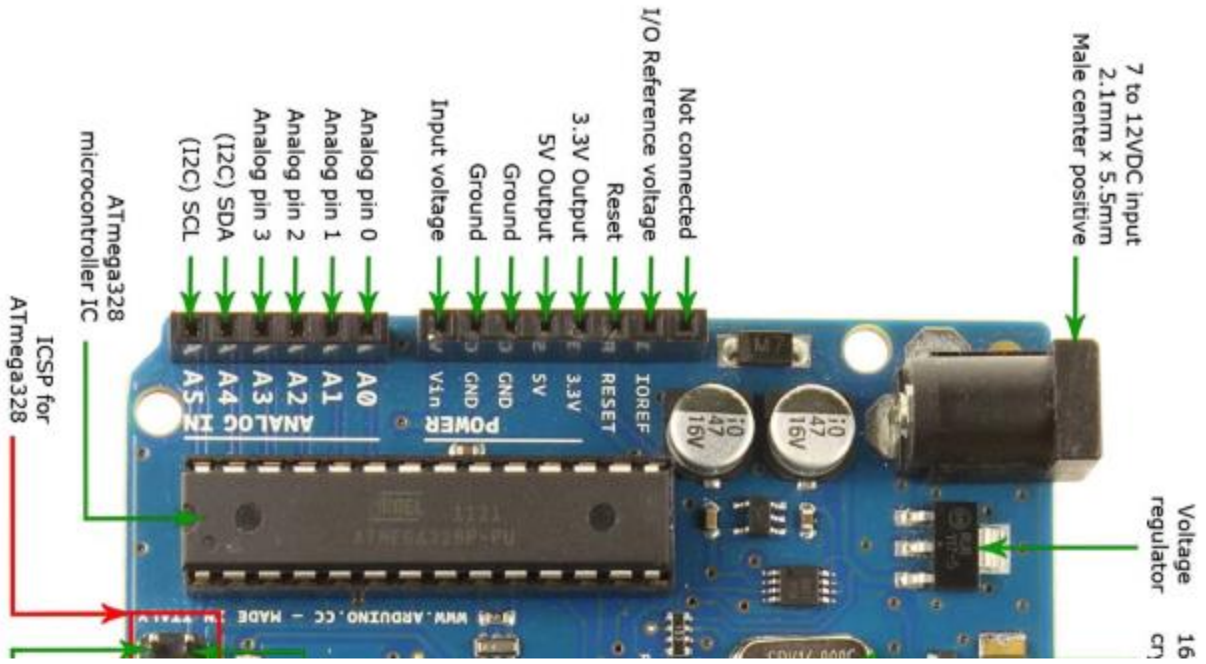
So let's have a look from the right side;



- We can see where the label says "DIGITAL (PWM)", this digital pins which are 14 in number (0-13) are both input and output pins depending on your code.
- The USB-B port to the computer, well you guessed right it is the port that connects your board to your PC via a cable to upload your working, compiled codes. The USB-B port can also be used to power the Arduino.

- The reset button, just like restarting your PC, the reset button functions the same way to restart the Arduino.It implies the program memory ROM is set to the starting position or address.Your code starts from the beginning, and hardware resets.This is particularly useful when an error is found in the code execution.
- And finally GND, implies ground which is the reference point for the circuit connection.

Moving on to the other half;



- Now the voltage regulator on this board functions to keep the voltage needed by the board within that range of value.
- The DC input (7 to 12V) is used to power the board.
- Down to the analog pins which are 6 in number (A0-A5) are input pins to which analog devices/components are connected to.
- ATMEGA328P is high performance, low power controller from Microchip. It is an 8-bit microcontroller, in simplicity, meaning it can process 8 bits of data at a time.

Note: The Arduino has an inbuilt analog to digital converter (ADC).

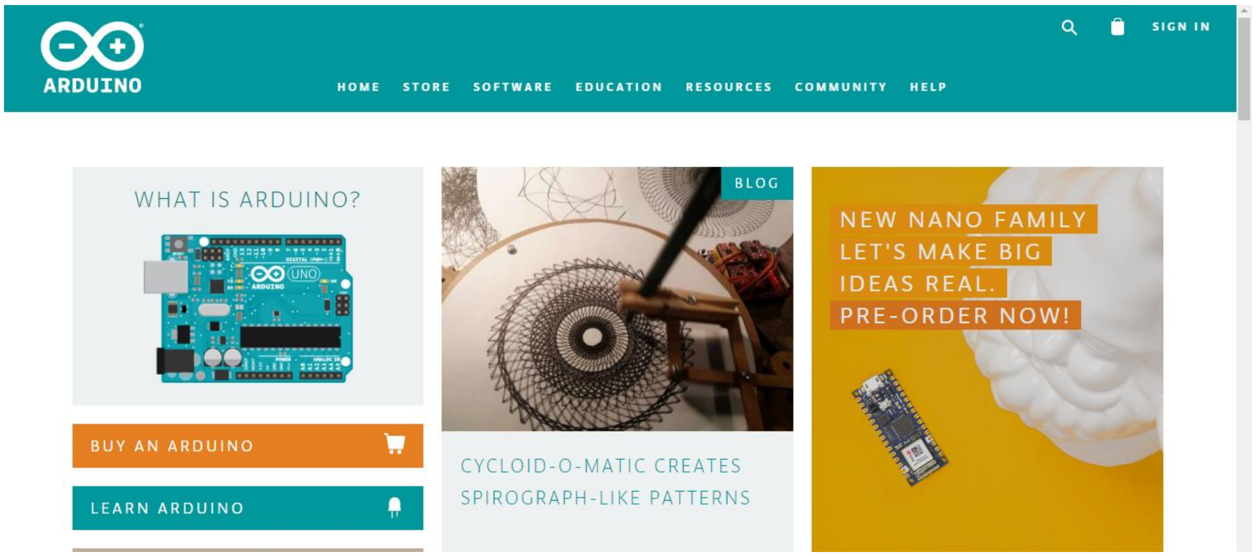
Impressive right? Now let us move on from the board and take a look on the IDE, how to download it and most importantly, how to use it.

Arduino IDE

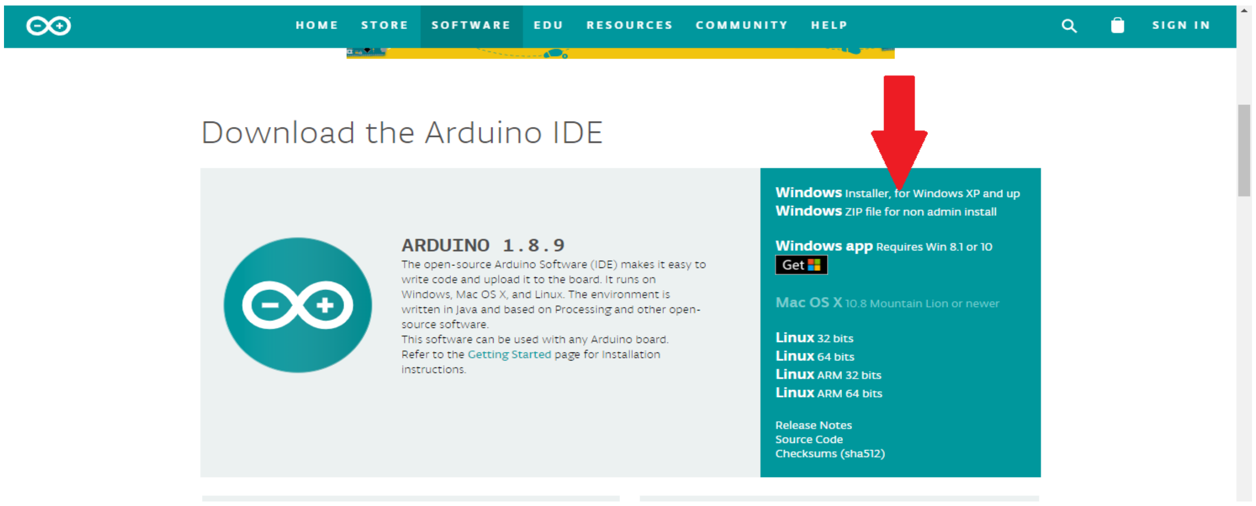
For programming our Arduino, we will need the Arduino IDE (Integrated Development Environment).

The official Arduino IDE can be downloaded from http://Arduino.cc/en/Main/Software

- Click on the link given above, you should see a page like this. While on this page, I would strongly advise you take a look around and see all what Arduino has to offer, who knows you might pick up some valuable information.😉



- Now move your cursor to software section on the main menu, you should see a drop down menu, click on the download option. This will then take you to a page that looks like this. Click the link where the arrow is pointed describing your system type, for me i would be clicking Windows Installer for Windows XP and up.



Download the Arduino IDE

ARDUINO 1.8.9

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

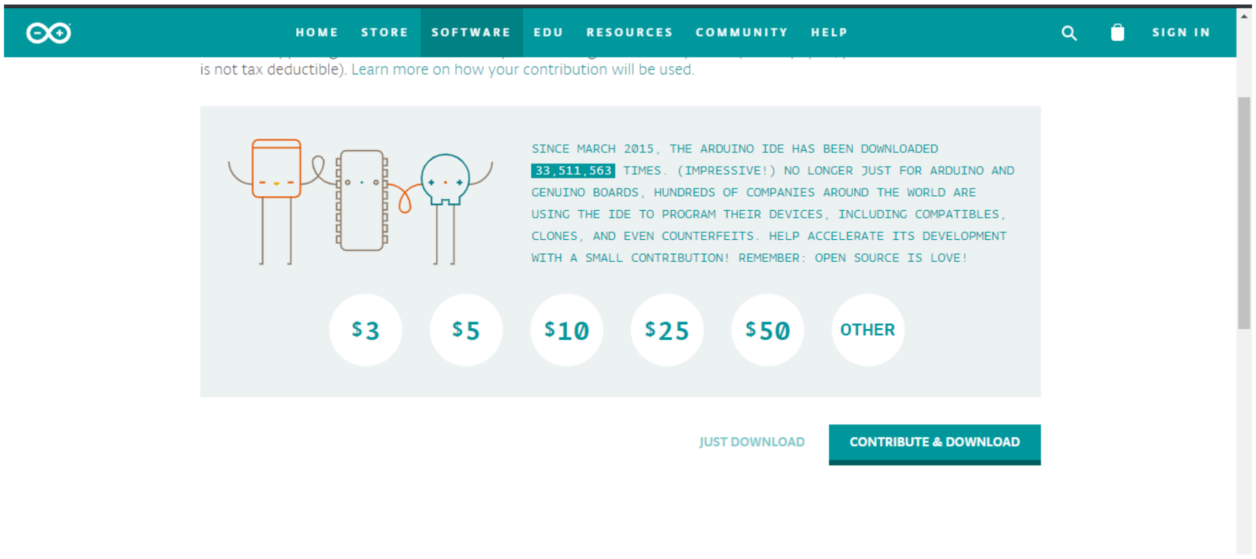
Windows app Requires Win 8.1 or 10
Get

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

- Finally it will take you to a new page. Click "just download" and wait for your download to complete.



is not tax deductible). Learn more on how your contribution will be used.

SINCE MARCH 2015, THE ARDUINO IDE HAS BEEN DOWNLOADED 33,511,563 TIMES. (IMPRESSIVE!) NO LONGER JUST FOR ARDUINO AND GENUINO BOARDS, HUNDREDS OF COMPANIES AROUND THE WORLD ARE USING THE IDE TO PROGRAM THEIR DEVICES, INCLUDING COMPATIBLES, CLONES, AND EVEN COUNTERFEITS. HELP ACCELERATE ITS DEVELOPMENT WITH A SMALL CONTRIBUTION! REMEMBER: OPEN SOURCE IS LOVE!

$3 $5 $10 $25 $50 OTHER

JUST DOWNLOAD CONTRIBUTE & DOWNLOAD

Moving on after downloading

Installation

We will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

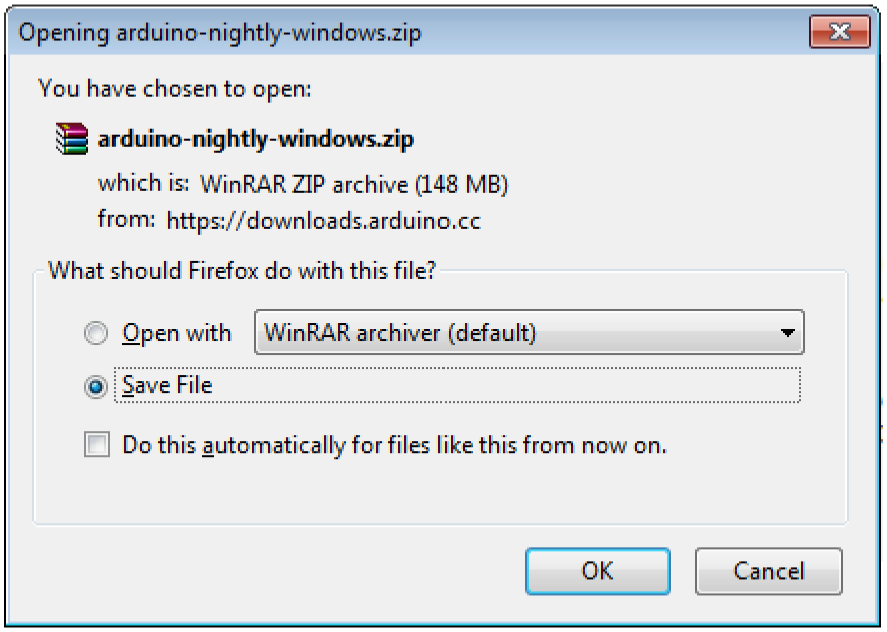
Step 1 – First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

Arduino Board Check

When you plug in your Arduino for the first time, you'll see a green light (with 'ON' written next to it - this is the power LED) and an orange light that blinks (with 'L' written next to it). This is the default 'Blink' program, it turns the internal LED on for a second, then turns it off for a second, repeating forever.

Step 2 – Download Arduino IDE Software.

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.



Opening arduino-nightly-windows.zip

You have chosen to open:

arduino-nightly-windows.zip

which is: WinRAR ZIP archive (148 MB)

from: https://downloads.arduino.cc

What should Firefox do with this file?

[ ] Open with WinRAR archiver (default)

[✓ Save File]

[ ] Do this automatically for files like this from now on.

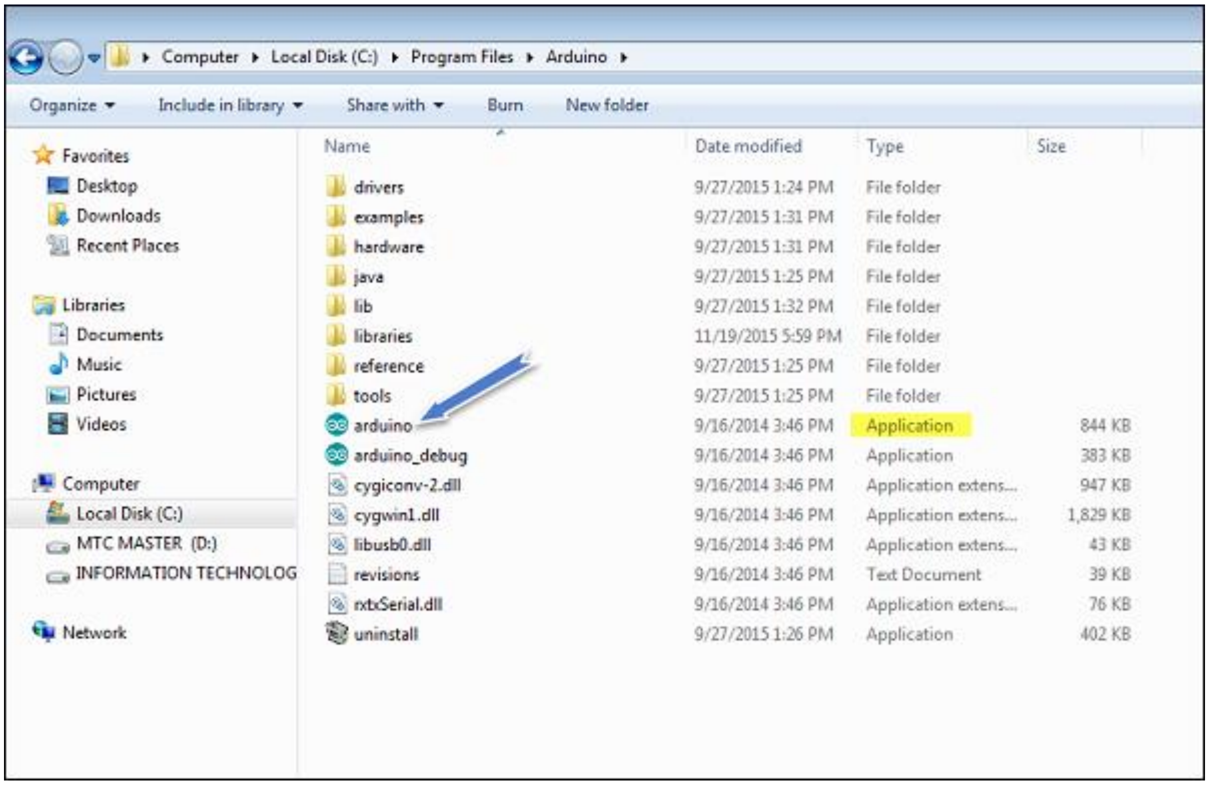
Step 3 – Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

Step 4 – Launch Arduino IDE.

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.



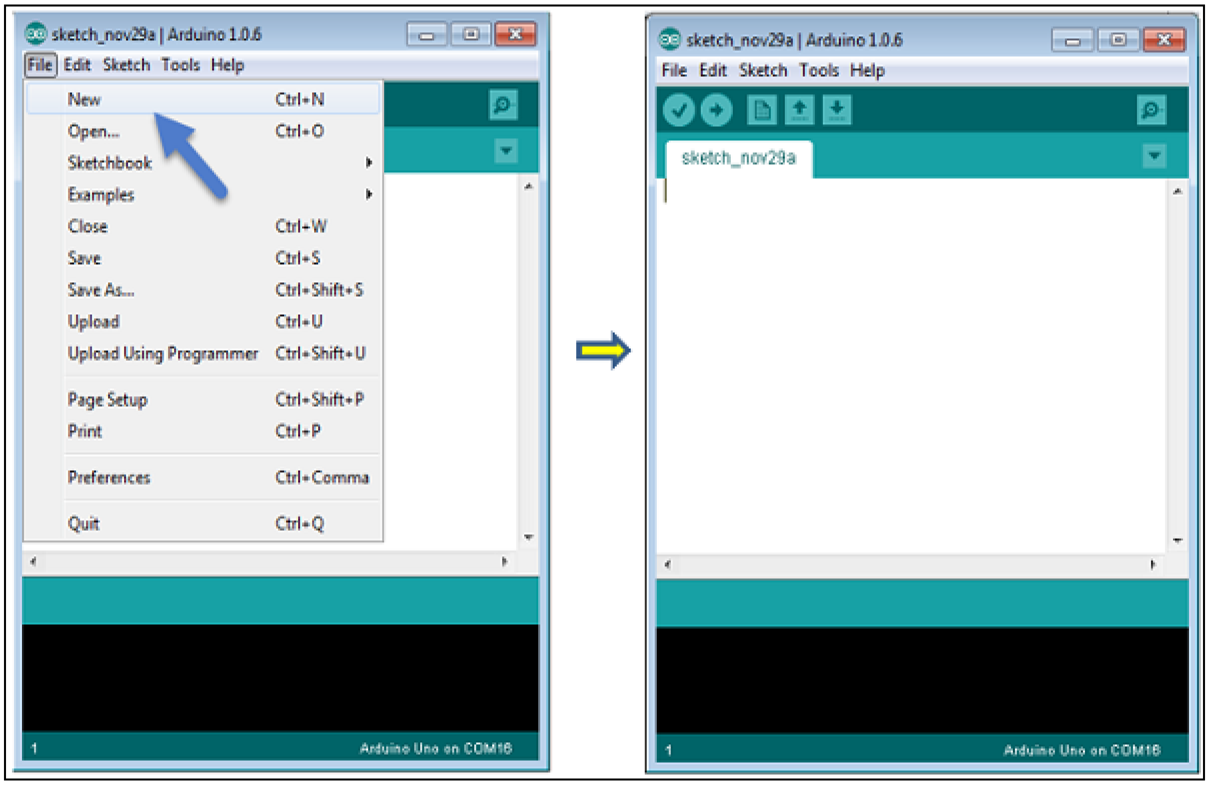
Name	Date modified	Type	Size
drivers	9/27/2015 1:24 PM	File folder	
examples	9/27/2015 1:31 PM	File folder	
hardware	9/27/2015 1:31 PM	File folder	
java	9/27/2015 1:25 PM	File folder	
lib	9/27/2015 1:32 PM	File folder	
libraries	11/19/2015 5:59 PM	File folder	
reference	9/27/2015 1:25 PM	File folder	
tools	9/27/2015 1:25 PM	File folder	
arduino	9/16/2014 3:46 PM	Application	844 KB
arduino_debug	9/16/2014 3:46 PM	Application	383 KB
cygiconv-2.dll	9/16/2014 3:46 PM	Application extens...	947 KB
cygwin1.dll	9/16/2014 3:46 PM	Application extens...	1,829 KB
libusb0.dll	9/16/2014 3:46 PM	Application extens...	43 KB
revisions	9/16/2014 3:46 PM	Text Document	39 KB
rxtxSerial.dll	9/16/2014 3:46 PM	Application extens...	76 KB
uninstall	9/27/2015 1:26 PM	Application	402 KB

Step 5 – Open your first project.

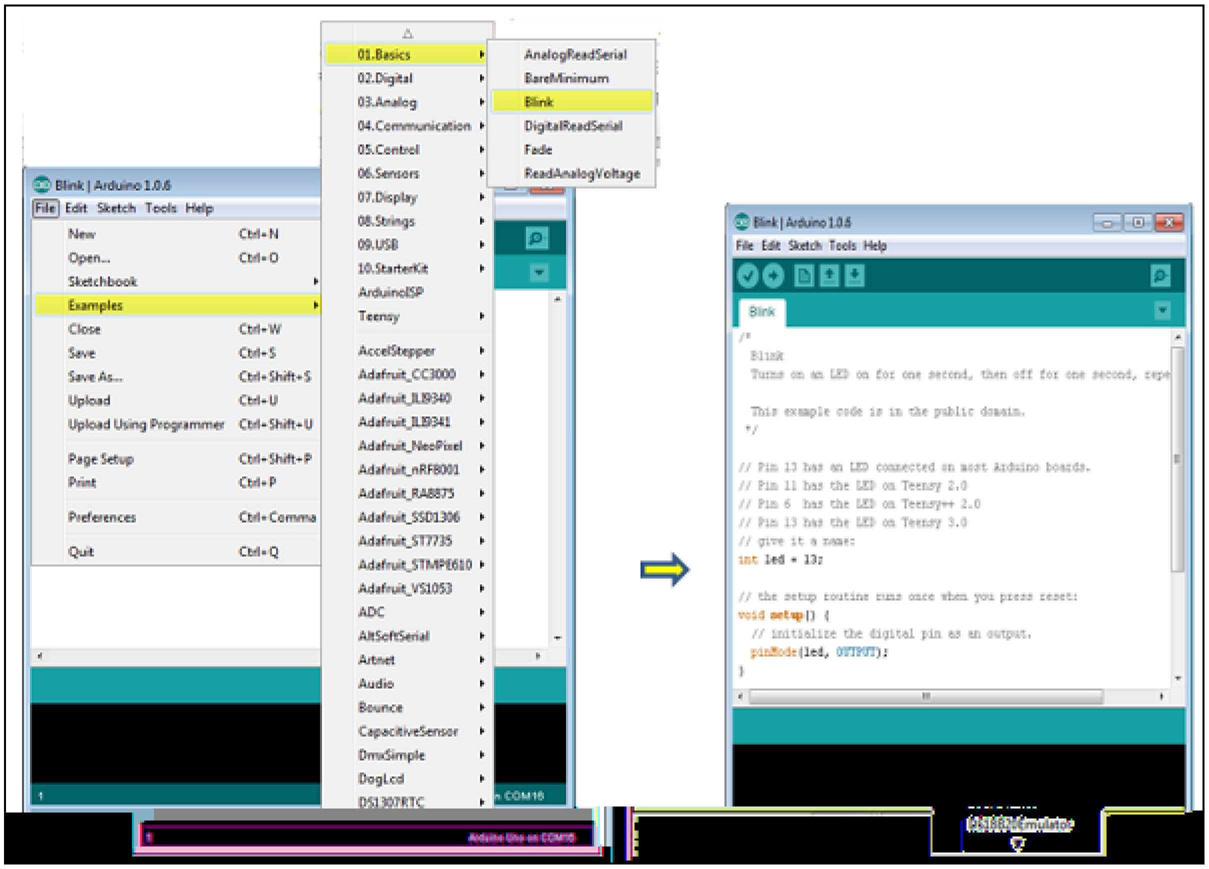
Once the software starts, you have two options –

- Create a new project.
- Open an existing project example.

To create a new project, select File → **New**.



To open an existing project example, select File → Example → Basics → Blink.



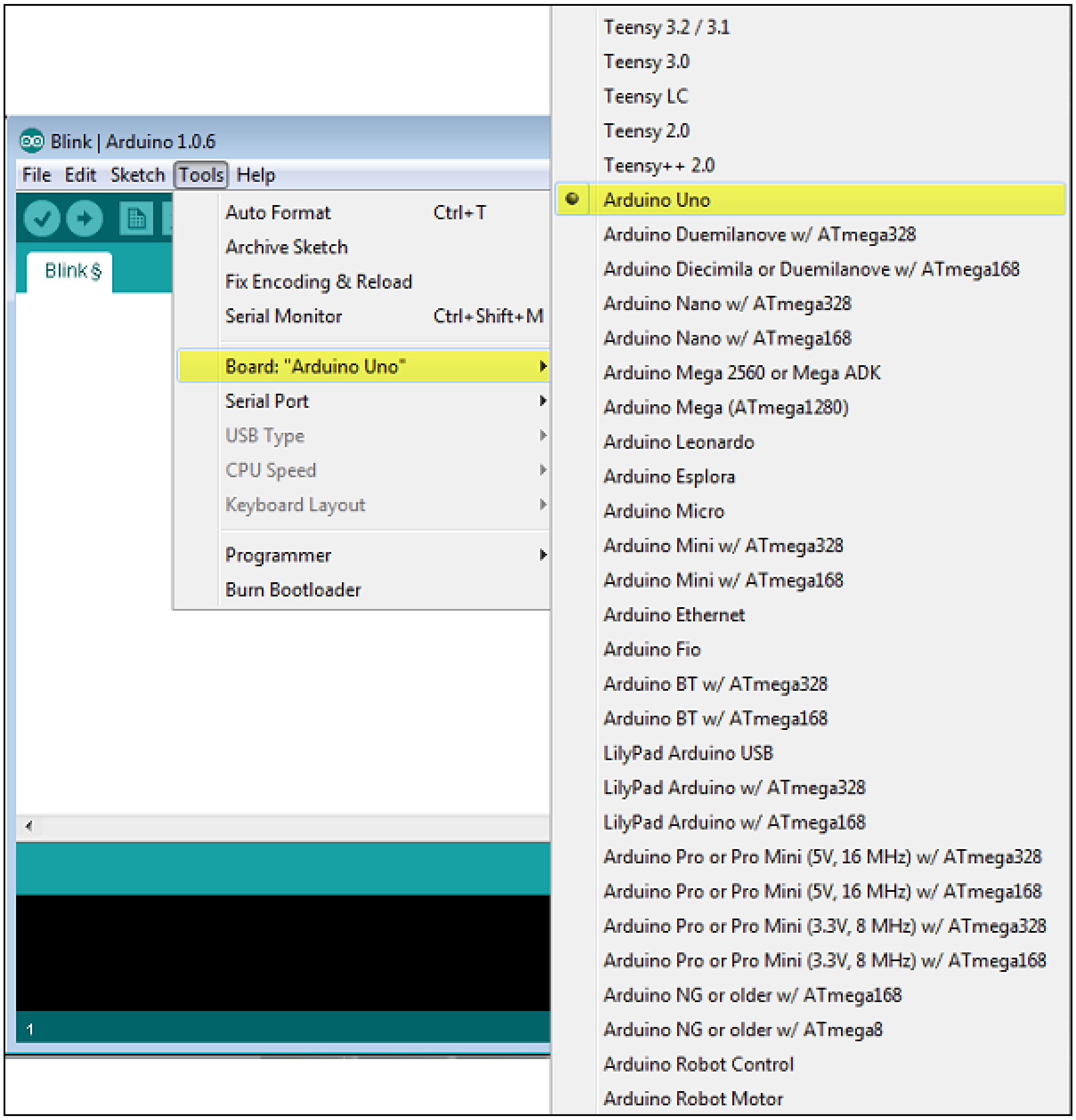
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6  has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other examples from the list.

Step 6 – Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

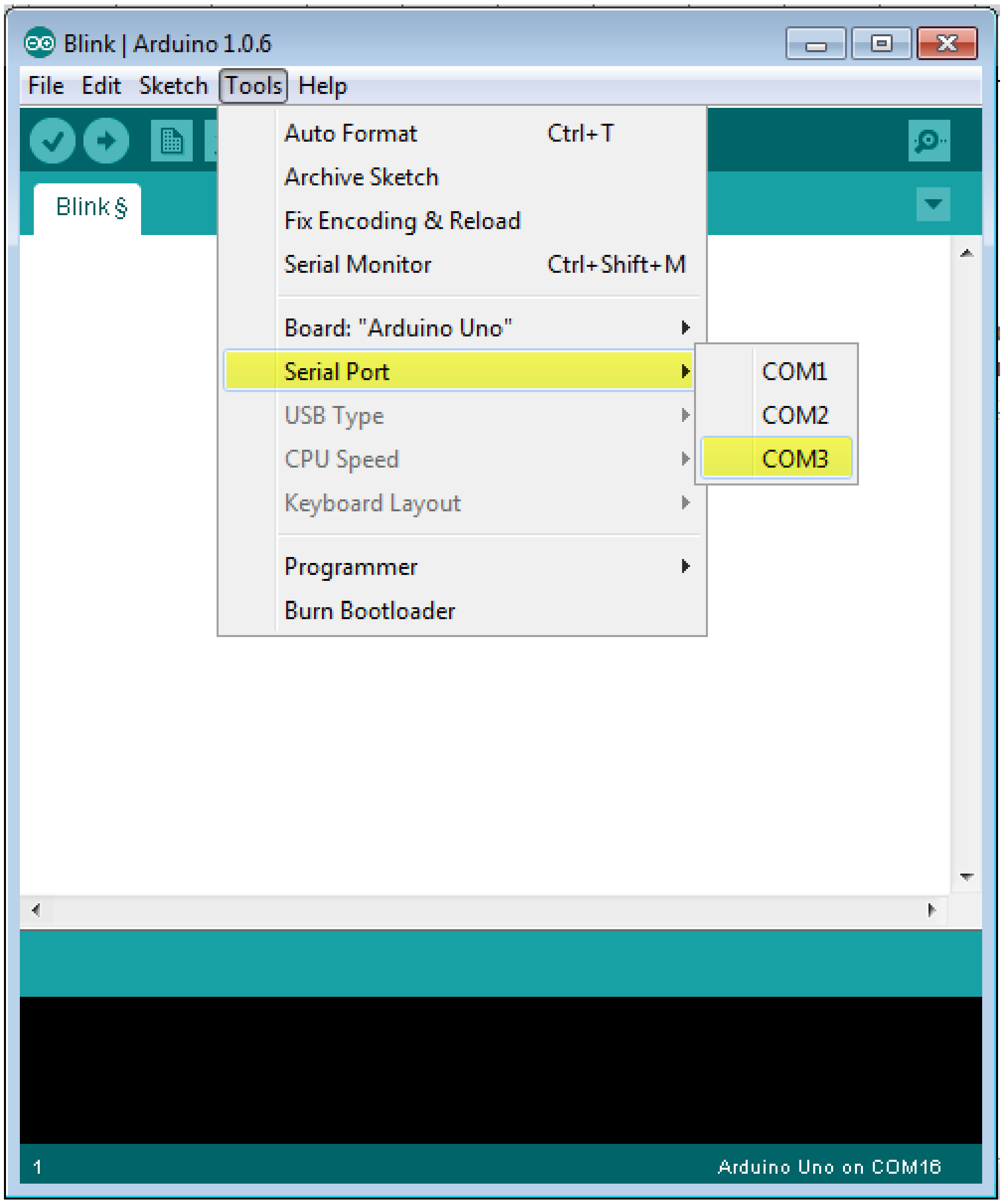
Go to Tools → Board and select your board.


Blink | Arduino 1.0.6
File Edit Sketch Tools Help
Blink §
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Arduino Uno"
Serial Port
USB Type
CPU Speed
Keyboard Layout
Programmer
Burn Bootloader
Teensy 3.2 / 3.1
Teensy 3.0
Teensy LC
Teensy 2.0
Teensy++ 2.0
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Esplora
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Arduino Robot Control
Arduino Robot Motor
1

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using.

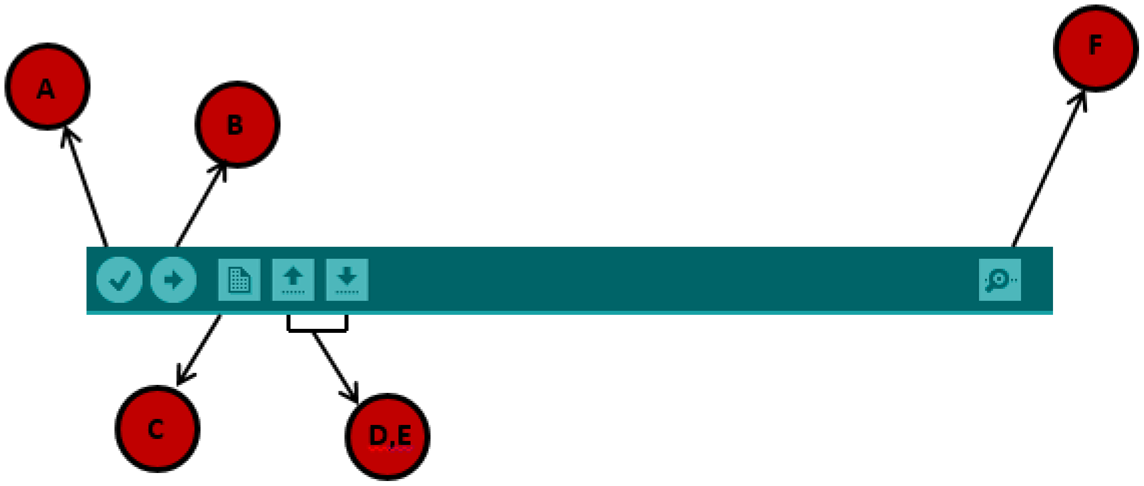
Step 7 – Select your serial port.

Select the serial device of the Arduino board. Go to **Tools → Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 8 – Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A – Used to check if there is any compilation error.

B – Used to upload a program to the Arduino board.

C – Shortcut used to create a new sketch.

D – Used to directly open one of the example sketches.

E – Used to save your sketch.

F – Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "**Done uploading.**" will appear in the status bar.

Note – If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

Now let's move on to performing our first Arduino project. We will be performing a blink LED light project. I can tell you are as eager as I am 😉.

Blink LED Project

LEDs are small, powerful lights that are used in many different applications. To start, we will work on blinking an LED, the Hello World of microcontrollers. It is as simple as turning a light on and off. Establishing this important baseline will give you a solid foundation as we work towards experiments that are more complex.

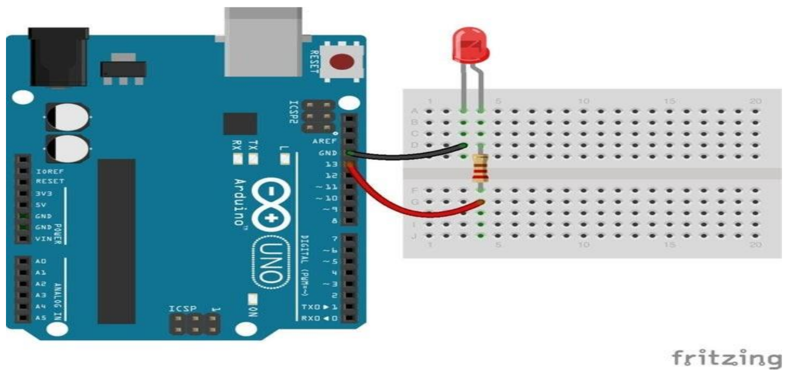
Components Required

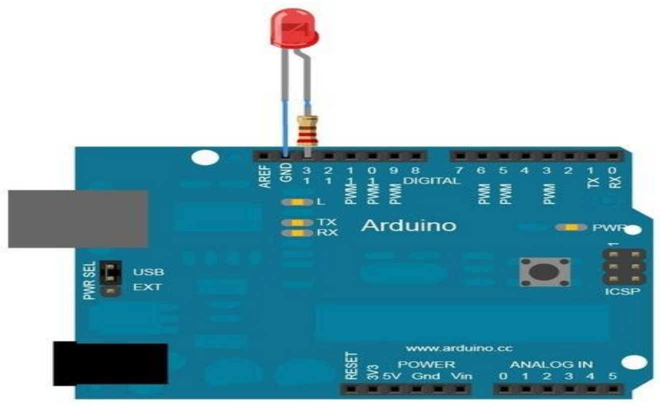
For this project, you will need the following components:

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

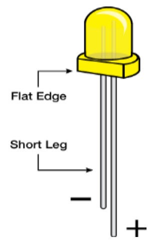
Procedure

Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below.

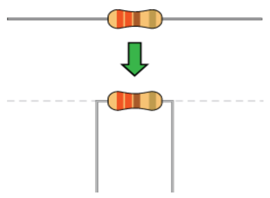




Note − To find out the polarity of an LED, look at it closely. The shorter of the two legs, towards the flat edge of the bulb indicates the negative terminal.

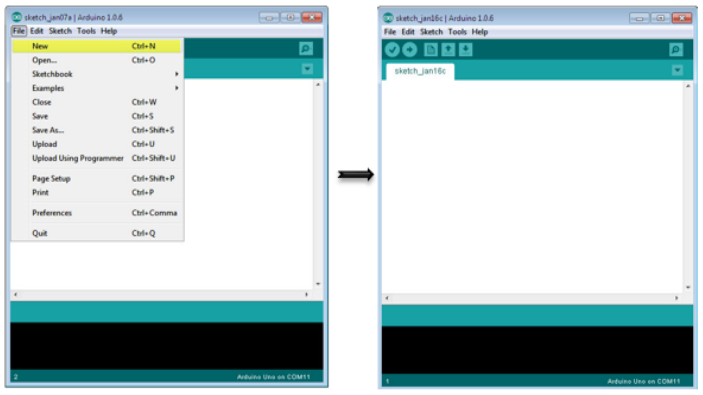


Components like resistors need to have their terminals bent into 90° angles in order to fit the breadboard sockets properly. You can also cut the terminals shorter.



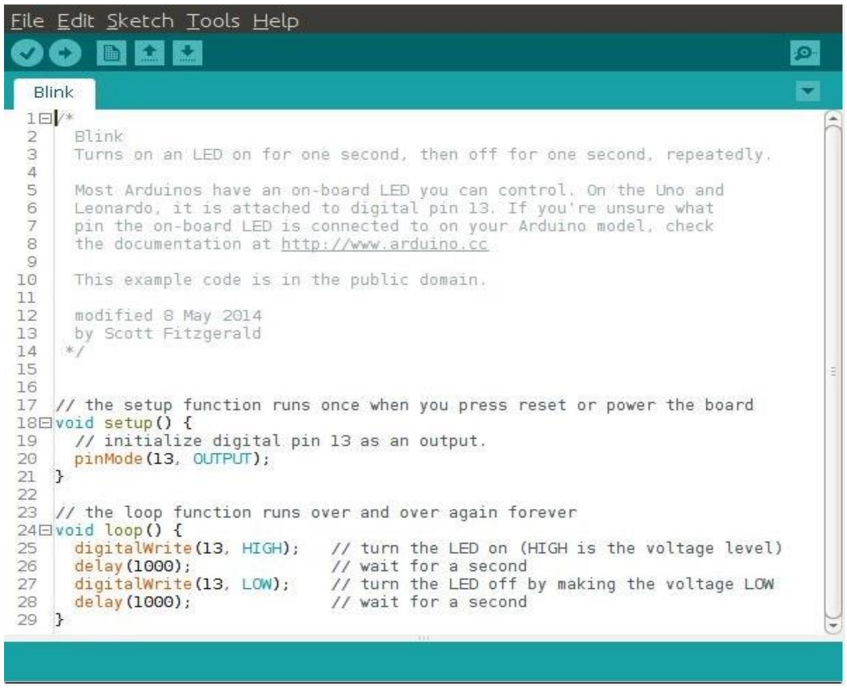
Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the new sketch File by clicking New.



Arduino Code

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
*/
// the setup function runs once when you press reset or power the board
        void setup() {  // initialize digital pin 13 as an output.
        pinMode(13, OUTPUT);
        }
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```



```
/*
 Blink
 Turns on an LED on for one second, then off for one second, repeatedly.

 Most Arduinos have an on-board LED you can control. On the Uno and
 Leonardo, it is attached to digital pin 13. If you're unsure what
 pin the on-board LED is connected to on your Arduino model, check
 the documentation at http://www.arduino.cc

 This example code is in the public domain.

 modified 8 May 2014
 by Scott Fitzgerald
 */


// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Code to Note

· **'void setup(){'**

This is the setup routine, it runs only once, every time the Arduino is started up. (void is a data type, it means no information is returned by the setup, more on this later. In the setup(), that only runs once when the program is started, we set pin 13 as an output.

· **'void loop(){'**

The code between the curved brackets is executed after the setup is finished, and will repeat forever (at least until you restart the Arduino, or upload another program). In the loop(), we make the output of the led HIGH (5v), wait 1,000ms, make it LOW (0v) and wait for another second. This loop will be repeated forever (at least until you restart the Arduino, or upload another program)

· /*this is a comment */ this is not

· this is not a comment // this is a comment

· every statement ends with a semicolon (“**;**”)

· ***pinMode(13, OUTPUT);*** or ***pinMode(13,1);*** sets the pin 13 as an output

· ***digitalWrite(13, state);*** sets a pin 13 on high (5v) or low (0v). State can be HIGH or LOW, 1 or 0, true or false.

· ***delay(time);*** waits for a given amount of time, in milliseconds. i.e. delay(1000) – 1sec

Upload

Using the steps for uploading above, upload the code unto the Arduino Board

Result

You should see your LED turn on and off. If the required output is not seen, make sure you have assembled the circuit correctly, and verified and uploaded the code to your board.

Chapter 2
ARDUINO BASIC CONCEPT AND PROGRAM NOTATION

Learning Outcomes

When you have completed this module, you will:

- Understand the nature of programming language used on Arduino IDE.
- Know and understand the basic Functions and Variables and how they are used.
- Understand the Structure of the Arduino IDE.
- Understand Data types and how to use them

Introduction

We already discussed a little about Arduino and even performed a simple project in the previous module. Let us do a quick recap on this, shall we? So we said the Arduino is an open-source prototyping platform and also said it runs on C and C++. The C programming language is a non-object-oriented language, while C++ is a semi object-oriented language.

What does Object-Oriented mean? Object-oriented programming (OOP) can simply be explained as programming in which programs are organized around objects, rather than functions and logic. Now for each programming language, there's something known as syntax.

What comes to your mind when you hear the word syntax? Let us take a wild guess on that. I would definitely think Error? Well if you thought the same, you are not exactly wrong, but that is not the syntax we are talking about here. Let me give you an example. So In your school, there are rules made to guide every member of the student body and to maintain orderliness right? Now imagine what happens if those rules are not in place, chaos! The same applies to a programming language, I mean without any particular rule for structure or whatsoever you could imagine how it could turn out right? Just typing whatever you feel, in any order, you feel is appropriate.

Not to waste too much time imagining the outcome of such, a Syntax in programming can simply be defined as rules guiding the characters used in a particular programming language. In other words, it means using character structure that a computer can interpret. In essence, trying to execute a command without proper syntax generates "syntax error", usually causing the program to fail.
Let us look at some basic syntax, shall we?

Syntax

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

}

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

}
```

Let us begin by running through the sample code above;

- **The "#include" statement:** This line is used to add Libraries to the code. Now you might ask, what is a library? A library in programming can be explained as modules that are stored in object format. An example of such is *#include<LiquidCrystal.h>.* What this particular line of code does is import the function for LCD.
- **Comments:** Comments are useful in programming to help explain lines of code for easy understanding when it is revisited or being worked on or even seen by someone else. The "**/**/**" (backslash double asterisk and backslash)is used to write multiple line comments, while the "**//**" (double backslash)this is used to write a single comment line.
- **The semicolon:** This is just like a full-stop in written English. It tells the compiler that you have finished with a line of code and it moves on to the next.
- **Curly braces:** These are very important as they are used to enclose further instructions carried out by a function. For instance, the compiler will generate an error if you forget to close a curly bracket.

Now let us go further, shall we?

Functions

This, perhaps is probably not the first time we are hearing the word functions, but what does it mean in terms of programming?

Functions can be explained as "self contained" pieces of code, often encapsulated that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result. Functions are designed such that once written, it can be used over and over and over again. Take for instance we have the following instructions which could be a function

boilYam
1. Peel yam
2. Rinse yam
3. Place in pot
4. Add water
5. Boil
6. Drain yam

This set of simple instructions could be encapsulated in a function that we call boilYam. Every time we want to carry out all those instructions we just type boilYam and all the instructions are executed.

In Arduino, there are quite a number of functions that are so often used that they have been built into the IDE. These functions are grouped into 14 which are;

1. Digital I/O
2. Math
3. Random numbers
4. Analog I/O
5. Zero, due & MKR family
6. Trigonometry
7. Bits and Bytes
8. Advanced I/O
9. Characters
10. External interrupts

11. Interrupts
12. Time
13. Communication
14. Usb

Let us talk about five of these functions groups we would be making use of in this textbook. We would then have full understanding of what each of these function group is all about. Fasten your seatbelts this is about to get really interesting.

Digital I/O: Just as the group name suggests, this is a function group that encompasses functions that read and write from the digital pins on the Arduino Uno. Common examples of these functions are the "digitalRead() and "digiatlWrite()"

Example

```
void loop() {

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

 digitalWrite(ledPin, val);  // sets the LED to the button's value

}
```

Math: This function group holds the functions used in mathematical computation. For example the "sqrt" function Calculates the square root of a number, written as sqrt(x). Few other examples are abs(), constrain() , map(), max(), min() , pow() , sq().

Random numbers: This function group contains only two functions, the "random()" and the "randomSeed()"

- **random()**

  The random function generates pseudo-random numbers (values that are statistically random, derived from a known starting point and is typically repeated over and over).

Syntax

random(max)

random(min, max)

Parameters definition

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

max: upper bound of the random value, exclusive.

- **randomSeed()**

The randomSeed() initializes the pseudo-random number generator, causing it to start at an unknown point in its random sequence. This sequence, while very long, and random, is always the same.

It is important to note that to get different values generated by the random() function, you have to use randomSeed() to initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.

Syntax

randomSeed(seed)

Parameter definition

seed: number to initialize the pseudo-random sequence.

Example

The code generates a pseudo-random number and sends the generated number to the serial port.

```
long randNumber;
```

```
void setup() {

  Serial.begin(9600);

  randomSeed(analogRead(0));

}



void loop() {

  randNumber = random(300);

  Serial.println(randNumber);

  delay(50);

}
```

Analog I/O: It is very easy to assume this is the opposite of the digital I/O function group, but I assure you, there is more to this than that assumption. This function group has 3 functions in it namely: analogRead(), analogReference(), and analogWrite(). Let us take a look at what each of these function does shall we?

- **analogRead():** This reads values 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 per 1024 units or, 0.0049 volts (4.9 mV) per unit. Straightforward right?

  Syntax

  analogRead(pin)

  Parameters

pin: the name of the analog input pin to read from (A0 to A5).

Data type: int (integers).

- **analogWrite():** Writes an analog value to a pin. This can be used to light an LED at varying brightnesses or drive a motor at various speeds.

  Syntax

  analogWrite(pin, value)

  Parameters

  pin: the Arduino pin to write to.

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

- **analogReference():** Configures the reference voltage used for analog input (i.e. the value used as the top of the input range).The default external analog reference is 5 volts and/or 3.3 volts while the default internal analog reference is 1.1V.

  Syntax

  analogReference(type)

  Parameters

  type: which type of reference to use (see list of options in the description).

Time: The name of this function group give a clear idea of what each function in this group does. There are 4 functions in this group namely: delay(), delayMicroseconds(), micros(), millis(). Let us take a look at what each of these function does shall we?

- **delay()**

What this function does is puts the program on hold for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second). Therefore to pause the program for one second it is then set to 1000.

Syntax

delay(ms)

Parameters

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

Example

The code pauses the program for one second before toggling the output pin.

```
int ledPin = 13;             // LED connected to digital pin 13



void setup() {

  pinMode(ledPin, OUTPUT);    // sets the digital pin as output

}

void loop() {

  digitalWrite(ledPin, HIGH); // sets the LED on

  delay(1000);                // waits for a second

  digitalWrite(ledPin, LOW);  // sets the LED off
  delay(1000);          // waits for a second

}
```

Having looked at what functions are and focused on some function groups immediate to our learning, we can move further into understanding what variables are and the types.

Variables

Variables can be explained as short descriptive names used as storage location. In other words, variable is a value that can change, depending on conditions or on information passed to the program. Just like functions, there are various groups on the Arduino IDE which encompases Data type and constants. These variable groups are;

1. Constants
2. Conversion
3. Data type
4. Variable scope & qualifiers
5. Utilities

Just like the function groups, we will talk about variable groups we are going to be making use of throughout this textbook.

Constants: Just as the name implies, they are values that never change. In other words, a value that cannot be altered during normal program execution. In Arduino language, constants are predefined expressions. They are used to make the programs easier to read. They are classified constants in groups: Floating Point Constants, Integer Constants, HIGH & LOW, (INPUT, OUTPUT & INPUT_PULLUP), LED_BUILTIN, True or False. Let us take a look at what each of these constants mean shall we?

Integer Constants

Integer constants are numbers that are used directly in a sketch, like 123. Normally, integer constants are treated as base 10 (decimal) integers, but special notation (formatters) may be used to enter numbers in other bases. Let us take a look below;

- **Decimal (base 10):** This is the common-sense math with which you are acquainted. Constants without other prefixes are assumed to be in decimal format.

  Example

  ```
  n = 101;  // same as 101 decimal ((1 * 10^2) + (0 * 10^1) + 1)
  ```

- **Binary (base 2):** Only the characters 0 and 1 are valid. To write this, character must lead with “B”.

  Example

  ```
  n = B101; // same as 5 decimal ((1 * 2^2) + (0 * 2^1) + 1)
  ```

  Note: The binary formatter only works on bytes (8 bits) between 0 (B0) and 255 (B11111111). If it is convenient to input an int (16 bits) in binary form you can do it a two-step procedure such as:

  ```
  myInt = (B11001100 * 256) + B10101010;  // B11001100 is the high byte`
  ```

- **Octal (base 8):** Only the characters 0 through 7 are valid. Octal values are indicated by the prefix "0" (zero).

  Example

  ```
  n = 0101; // same as 65 decimal ((1 * 8^2) + (0 * 8^1) + 1)
  ```

- **Hexadecimal (base 16)**

  Valid characters are 0 through 9 and letters A through F; A has the value 10, B is 11, up to F, which is 15. Hex values are indicated by the prefix "0x". Note that A-F may be upper (A-F) or lower case (a-f).

  Example

  ```
  n = 0x101;  // same as 257 decimal ((1 * 16^2) + (0 * 16^1) + 1)
  ```

Floating Point Constant

This is similar to integer constants, floating point constants are used to make code more readable. Floating point constants are swapped at compile time for the value to which the expression evaluates.

Example

```
n = 0.005;  // 0.005 is a floating point constant
```

HIGH & LOW

When reading or writing to a digital pin there are only two possible values a pin can take/be-set-to: HIGH and LOW.

- **HIGH:**The meaning of HIGH (in reference to a pin) is somewhat different depending on whether a pin is set to an INPUT or OUTPUT. When a pin is configured as an INPUT with pinMode(), and read with digitalRead(), the Arduino will report HIGH if:
  - a voltage greater than 3.0V is present at the pin
  - a voltage greater than 2.0V volts is present at the pin

  When a pin is configured to OUTPUT with pinMode(), and set to HIGH with digitalWrite(), the pin is at:

  - 5 volts
  - 3.3 volts

  In this state it can source current, e.g. light an LED that is connected through a series resistor to ground.

- **LOW**

The meaning of LOW also has a different meaning depending on whether a pin is set to INPUT or OUTPUT. When a pin is configured as an INPUT with pinMode(), and read with digitalRead(), the Arduino (ATmega) will report LOW if:

- a voltage less than 1.5V is present at the pin
- a voltage less than 1.0V (Approx) is present at the pin

When a pin is configured to OUTPUT with pinMode(), and set to LOW with digitalWrite(), the pin is at 0 volts. In this state it can sink current, e.g. light an LED that is connected through a series resistor to +5 volts (or +3.3 volts).

True or False

These are logic statements used to represent truth and falsity in the Arduino language.

False: False is defined as 0 (zero) logically.

True: True is defined as 1 logically, but also has a wider definition. Any integer which is non-zero is true, in a Boolean sense. So -1, 2 and -200 are all defined as true, too, in a Boolean sense.

Note: The true and false constants are typed in lowercase unlike HIGH, LOW, INPUT, and OUTPUT.

LED_BUILTIN

Arduino Uno board has a pin connected to an on-board LED in series with a resistor. The constant LED_BUILTIN is the number of the pin to which the on-board LED is connected. This LED is connected to digital pin 13.

Data Types: What are Data types? Basically, data types are attributes of data which tells the compiler how to use the data. There are a couple of data types, but we will focus on a few which are:

- **String():** This can be simply defined as an array of characters. In Arduino, this data type “string()” can be used for integer and floating point unit, but is used to represent text rather than numbers.

  Syntax

  String(val)

  String(val, base)

  String(val, decimalPlaces)

  Let us look at a few examples of how a string is used;

  String thisString = String(13); // this gives you the String "13"

  String thisString = String(13, HEX); // this gives you the String "D", which is the hexadecimal representation of the decimal value 13.

  String thisString = String(13, BIN); // this gives you the String "1101", which is the binary representation of 13.

- **Boolean and Bool:** Boolean is a non-standard type alias for bool defined by Arduino. It’s recommended to instead use the standard type bool, which is identical. A bool holds one of two values, true or false. (Each bool variable occupies one byte of memory.)

  Syntax

  bool var = val;

  Example

  bool running = false;

- **Int:** Int, short for Integers, are your primary data-type for number storage.

Syntax

int var = val;

var: variable name.

val: the value you assign to that variable.

- **Char:** This is basically a data type used to store a character value. Character literals are written in single quotes, like this: 'A' (for multiple characters - strings - use double quotes: "ABC").

  Syntax

  char var = val;

  var: variable name.

  val: the value to assign to that variable.

  Example

  char myChar = 'A';

  char myChar = 65;

- **Float:** This data type is majorly used for floating-point numbers, a number that has a decimal point. Let me show you an example;

  Example

  Since we already established that the Arduino IDE is based on the C and C++ the example I will show you would be based on this. Let's do this!

We will be comparing the data types float and int in order to understand this;

For float

```
#include <iostream>

using namespace std;

int main()

{

   float m = 23.88;

   cout<<m; // this gives an output of “23.88”

}
```

For int

```
#include <iostream>

using namespace std;

int main()

{

   int n = 23.88;

   cout<<n; // this gives an output of “23” neglecting the decimals

}
```

That is how “float” work. Let’s take a look at double shall we?

- **Double:** This is also used for floating-point numbers but in this case it is a double precision floating point number. It is mostly used interchangeably with float, but they are different.

The difference is double has two times the precision of float. In general a double has 15 decimal digits of precision, while float has 7.

- **Void:** The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

  Example

  The code shows how to use void.

```
// actions are performed in the functions "setup" and "loop"

// but  no information is reported to the larger program



void setup() {

  // ...

}



void loop() {

  // ...

}
```

Moving on, let us take a look at arrays;

- **Arrays:** The term array probably doesn’t sound so important to you, but it definitely is, in fact, it's importance can not be overestimated. What is an array? An array is simply a collection of variables that are accessed with an index number. Let me paint a picture in your mind. Think of a book shelf with 3 horizontal rows labelled A, B, and C, where each row has a total of 5 books. Now row A has books numbered 1-5, row B has 6-10, row C has 11-15. Now that can be said to be 3 different arrays, and can be written like this;

int A[5] = {1,2,3,4,5}; int B[5] = {6,7,8,9,10}; and int C[5] = {11,12,13,14,15};

An array could also be words for example: an array of animals

int A[5] = {cat,dog,ram,chicken,rat};

Parameters

int: This specifies the data type

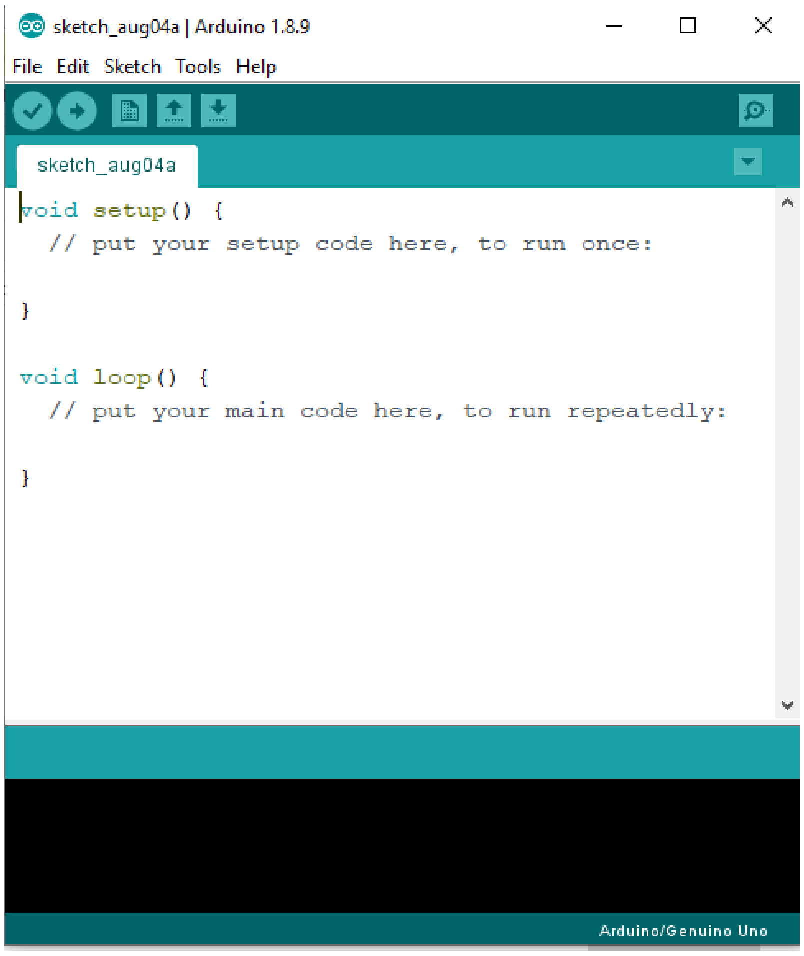
Letters (A, B, and C): The name of the array,

[5] (boxed bracket with number): This tells the number of element in the array

{}: This is used to house the elements which are separated by commas (,).

Moving on let us talk about the structure of the Arduino IDE.

Structure



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

}

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

}
```

Void 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.

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.

Having finished this module, I now believe we understand the structure of Arduino IDE, Functions used in Arduino IDE, and Data types

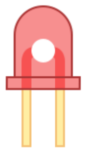
Chapter 3
RGB LED

Learning Objectives

When you have completed this module, you will:

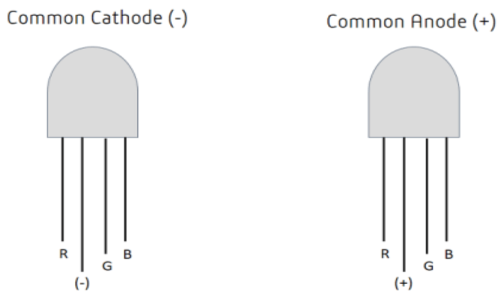
- Understand the configuration of an RGB LED.
- Know how to integrate an RGB LED with Arduino.
-

Introduction



LEDs! We have come across this in module 5 of the beginner track, and we stressed on how important they are, pointing out some applications in traffic lights, digital clocks, remote controls, televisions, and even billboards. We also pointed out how it differs from an incandescent bulb as they do not have filaments that can burn out and how they do not get hot. Basically, they are very powerful when you look at their areas of applications. Moving further, we are going to be discussing the RGB LED in this module.

The RGB LED from a distance looks exactly like your regular LED, but there is more to it to differentiate these two. Having a second look at it, you will notice it has four legs. Inside the RGB LED, there are actually three LEDs, one red, one green, and one blue. Now a smart question to ask is; since there are 4 legs and 3 colours of LED all mapped to one colour each, what about the last leg? Well the last leg of the LED is called the common, that is, cathode or anode, depending on the type of LED it is (common anode or common cathode).



It is 100% possible to generate other pigments of color from a single RGB LED by mixing these 3 colours. We mix colors just like mixing paint on a palette. How does this work? By controlling the brightness of each of the individual LEDs you can mix pretty much any color you want. In this single LED each colour (red, green, and blue) is tied to one leg for control to enable mixing of colours as earlier said.

Of course there is more than one way to go about this, a harder way would be to use different value resistors which is a lot of work! But the Arduino, being an interesting module has an **analogWrite** function that you can use with pins marked with “a ~” to output a variable amount of power to the appropriate LEDs.

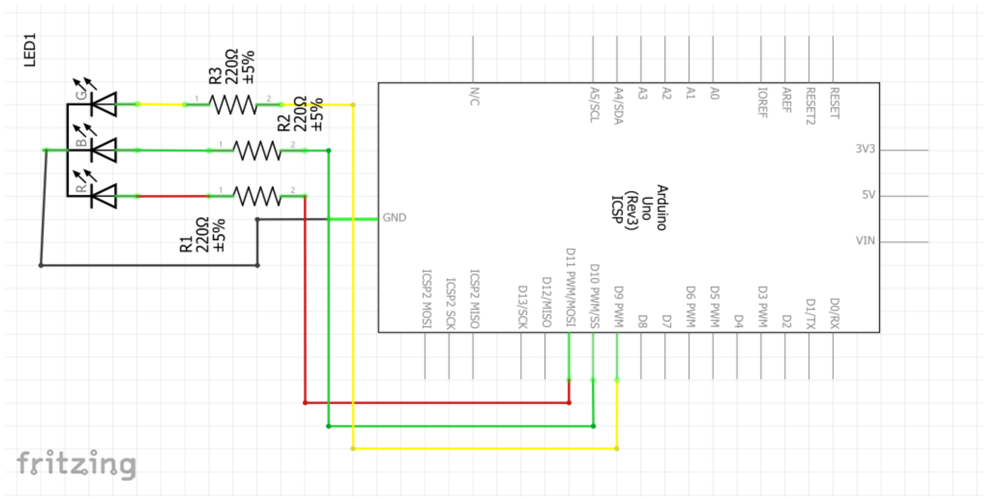
Let us perform a simple project to demonstrate this, shall we?

Components

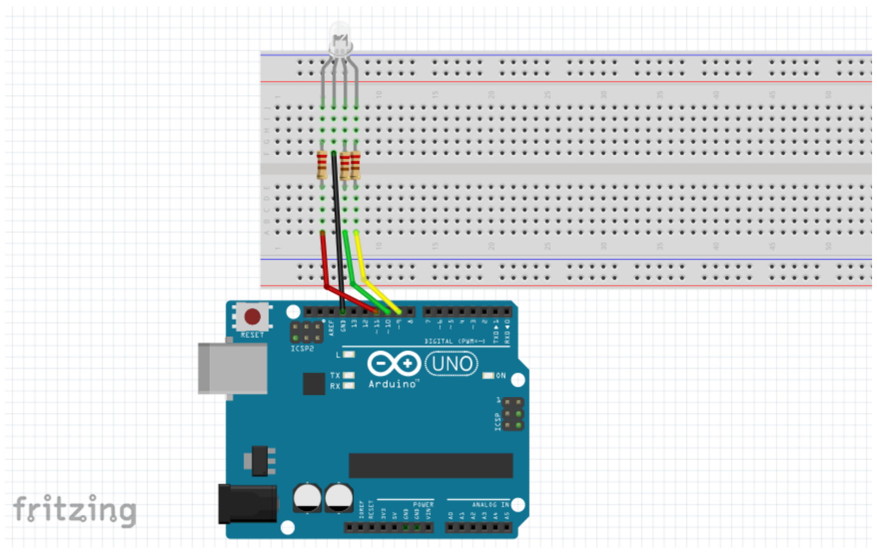
To build this project, we will require some electronic components which are;

- Arduino UNO
- 3 Resistors (220Ω)
- RGB LED
- Connecting wires
- Breadboard
- And lastly, a PC (personal computer) with the Arduino IDE installed to write the Arduino code

Schematics



Connection



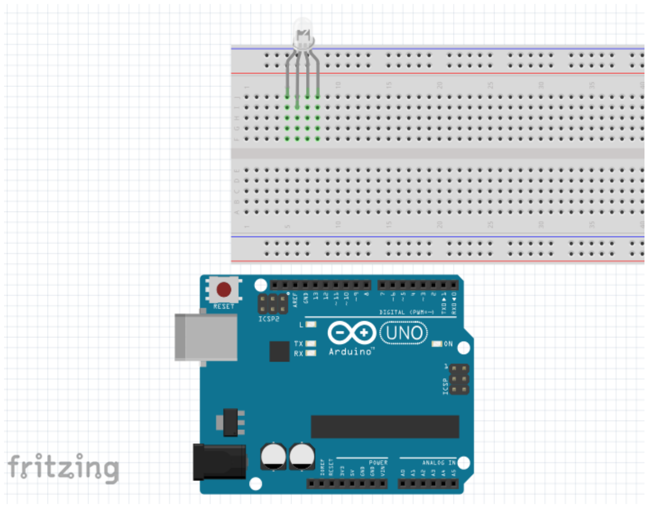
In order to achieve the connection shown above, we have to follow certain steps.

1. The first thing we are going to be doing placing our LED on the breadboard and making sure it is connected properly. For this project we are going to be making use of a common cathode RGB LED.

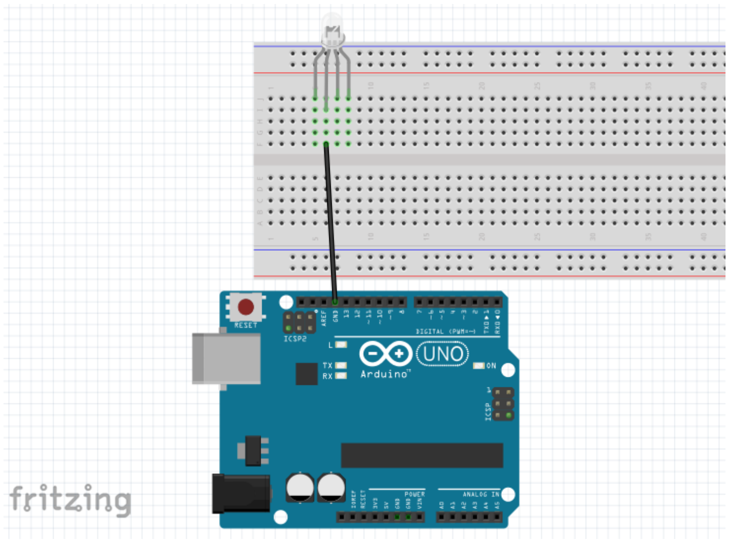
   A smart question would be; how do you differentiate a common cathode from a common anode since they both look exactly the same? The best way to identify or distinguish a common anode from a common cathode using a multimeter.

How does this work?

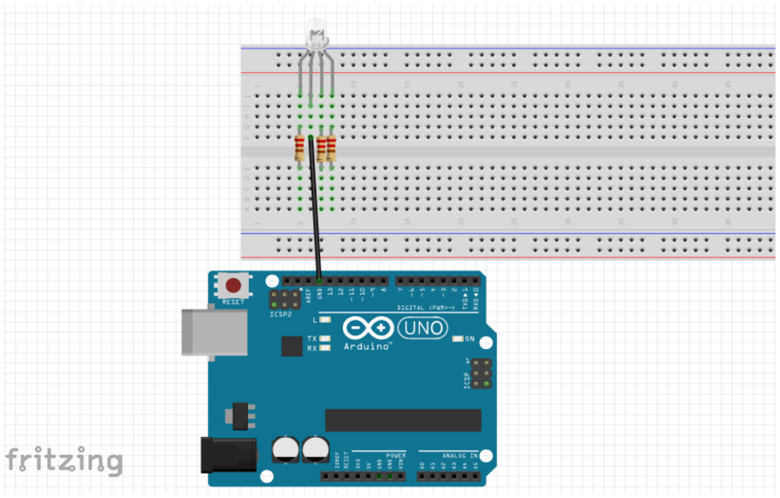
- Put your multimeter in continuity mode.
- Place the red connector on the longest leg of LED and the black on any of the other LED legs.
- If the LED lights up, then it is a common anode.
- Quite the opposite, if the LED lights up when the black connector is on the longest leg of the LED and the red connector on any of the other legs, it is a common cathode.



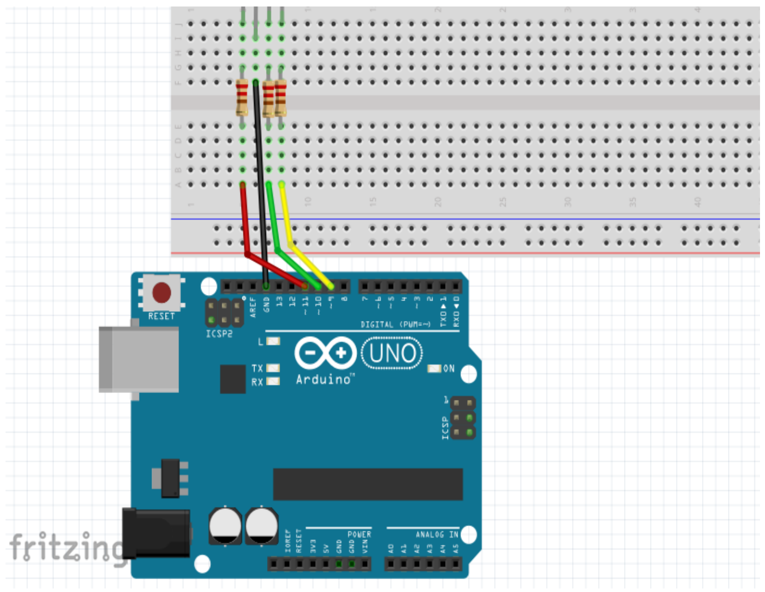
2. Now the next step is to connect the (common cathode) longest leg to the ground pin on the Arduino.



3. The next obvious step would be connecting the remaining 3 legs to a 220Ω resistor each. Like this;



4. We would be connecting the end legs of these resistors to pins 9, 10, 11 starting from the right.



Code

After properly connecting the circuit, the next step is to type and upload code to the Arduino board to enable the circuit work. The code we will be writing will help us display different colours for 10 seconds each before changing.

Let us get to it;

As usual, the first thing to do is define the pins that are going to be used

```
int red_light_pin= 11;

int green_light_pin = 10;

int blue_light_pin = 9;
```

Setting up the LEDs as Output pins.

```
void setup() {

  pinMode(red_light_pin, OUTPUT);

  pinMode(green_light_pin, OUTPUT);
  pinMode(blue_light_pin, OUTPUT);

}

void loop() {

  RGB_color(255, 0, 0); // Red

  delay(1000);

  RGB_color(0, 255, 0); // Green

  delay(1000);

  RGB_color(0, 0, 255); // Blue

  delay(1000);

  RGB_color(255, 255, 125); // Raspberry

  delay(1000);

  RGB_color(0, 255, 255); // Cyan

  delay(1000);

  RGB_color(255, 0, 255); // Magenta

  delay(1000);

  RGB_color(255, 255, 0); // Yellow

  delay(1000);

  RGB_color(255, 255, 255); // White

  delay(1000);
```

```
}

void RGB_color(int red_light_value, int green_light_value, int blue_light_value)

 {

  analogWrite(red_light_pin, red_light_value);

  analogWrite(green_light_pin, green_light_value);

  analogWrite(blue_light_pin, blue_light_value);

}
```

After uploading the code to the Arduino module, the program should start running immediately, and you will catch a glimpse of how powerful the RGB LED could be.