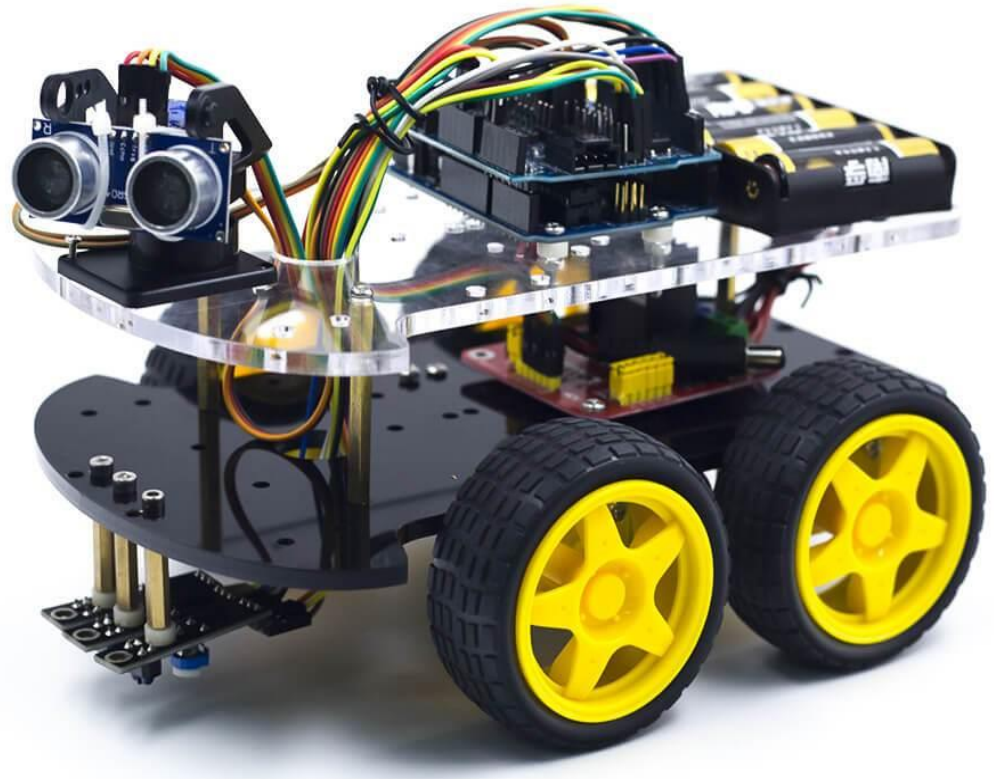
NSDC – Junior Skills Championship Mobile robotics - Round 3



Wednesday, August 18, 2021

Round 3 Webinar content

Day 1	Arduino C programming: Introduction to Arduino IDE, Interfacing IO devices
Day 2	Sensor Programming: PIR Sensor, IR Sensor, Ultrasound, Motor & Driver Interfacing
Day 3	Line follower bot experiment & Obstacle avoidance bot
Day 4	Cliff detection & Wall following Bot

Round 3 - Instructions and important information

- Round 3 will be a Skill Test subjective type:
  - Electronics & programming
  - 3 level skill set mapping
- Participants will be able to download the Problem Statement at the scheduled time of Round 3 Competition
- The Problem Statement will contain tasks to perform and instructions to submit the solutions
- No submission will be accepted after the scheduled time of the competition
- Device for attempting competition
  - Laptop/Desktop is recommended
  - Please ensure that your device is connected to a stable internet during the time of competition.
- All future communication shall be done over your registered email address. Please regularly check your emails

Agenda – Day 1

- Arduino Platform & C programming
- Arduino IDE
  - Arduino IDE installation
  - Arduino IDE tool
- Arduino Programming
  - Setup & Loop functions
  - Analog and Digital functions
  - Data types
  - Operators
  - Internal Pull-Up Resistor
  - Arduino Libraries
  - Interfacing I/O devices
  - Motors

Arduino Input / Output Pins

- Top to bottom rows of the board
- Holes in the board which we can stick wires in
- Holes are connected to the chips through traces on-board
- 14 digital I/O pins on top [0–13]
- High – 5Volts, Low – 0 volts, Max current – 40mA
- 6 Analog input pins on the bottom [A0 – A5]
- Power output pins on the bottom [5V, 3.3V]

Arduino IDE

- Arduino programs are written in the Arduino Integrated Development Environment (IDE)
- Arduino IDE is a special software running on your system that allows you to write sketches (synonym for program in Arduino language)
- The Arduino programming language is based on a very simple hardware programming language called processing, which is similar to the C language

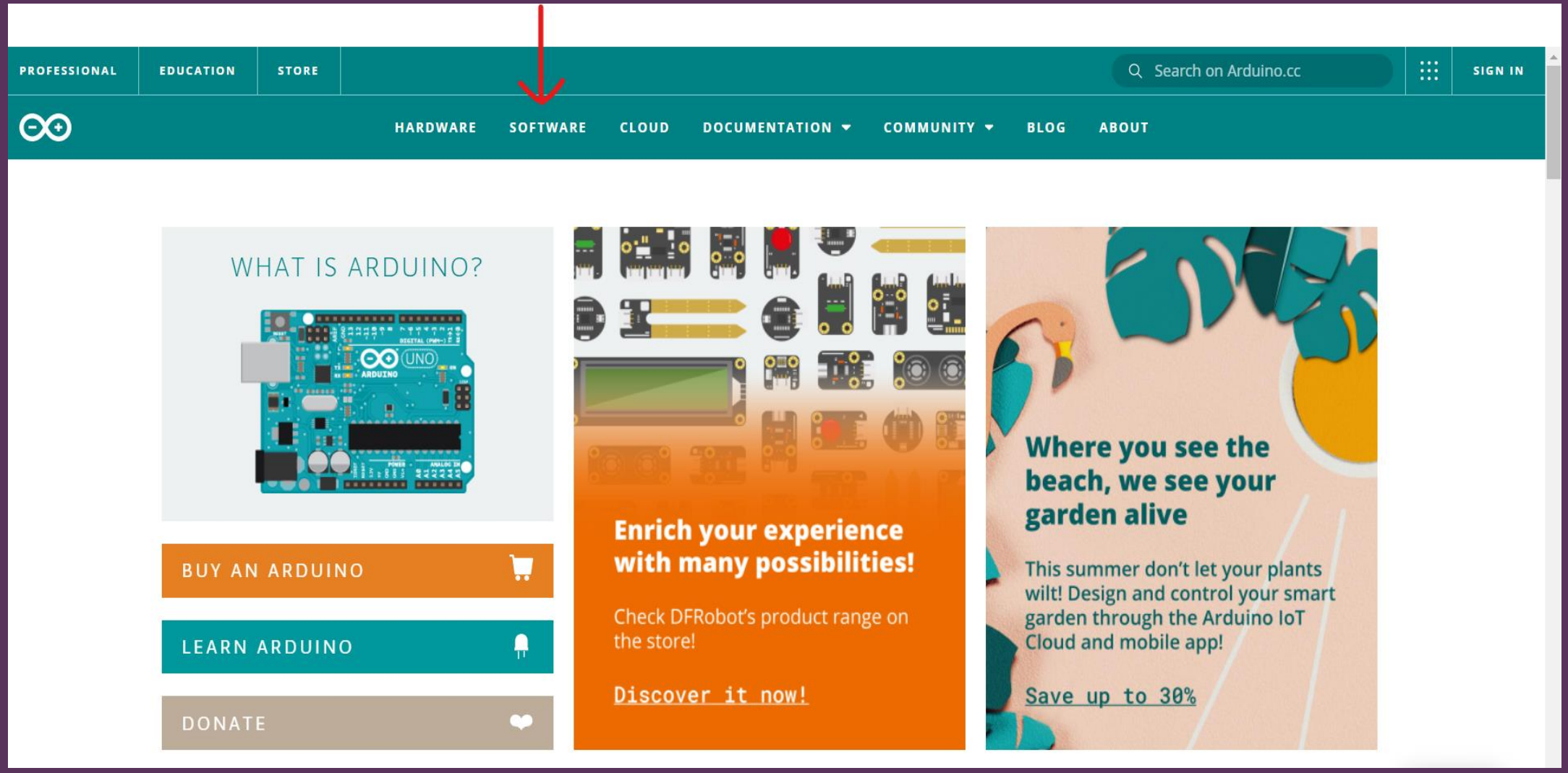
Arduino IDE Installation

HOLOWORLD

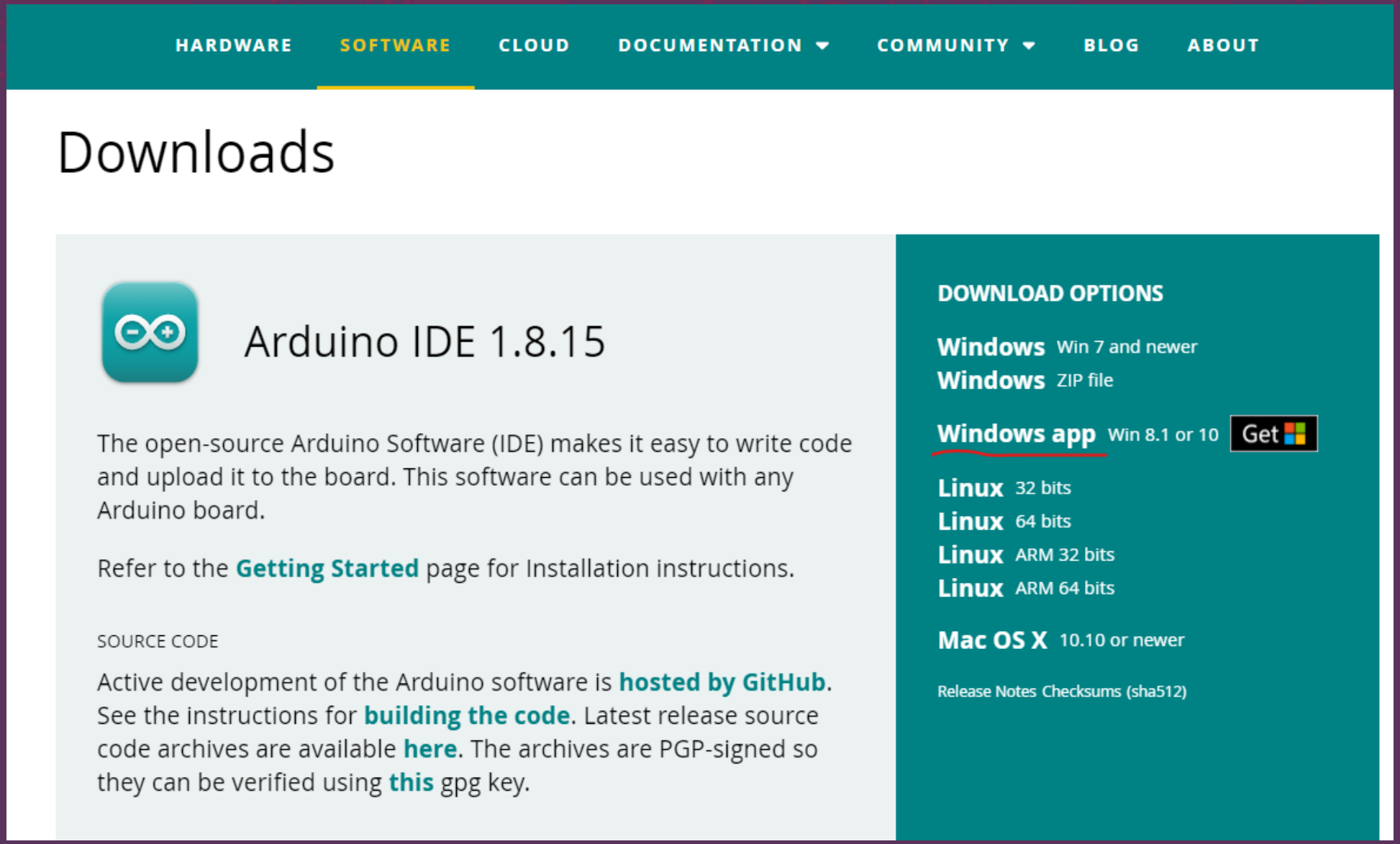
- The first step in programming the Arduino board is downloading and installing the Arduino IDE.
- The open-source Arduino IDE runs on Windows, Mac OS X, and Linux.
- Download the Arduino software (depending on your OS) from the official website and follow the instructions to install and the updated version on going is ARDUINO 1.8.13

Installing Arduino IDE

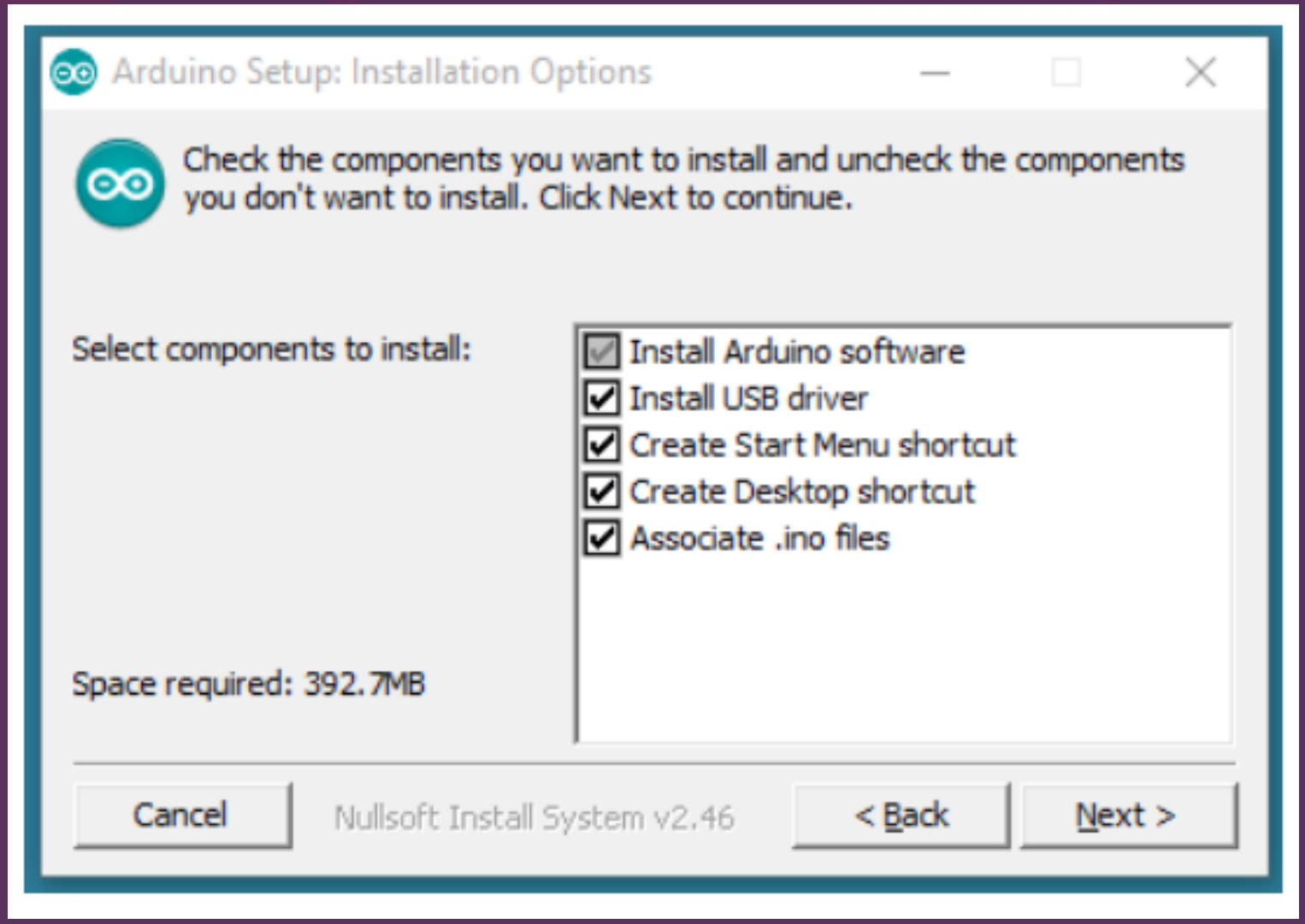
- Head over to arduino.cc
- Go to the Software tab and click on it

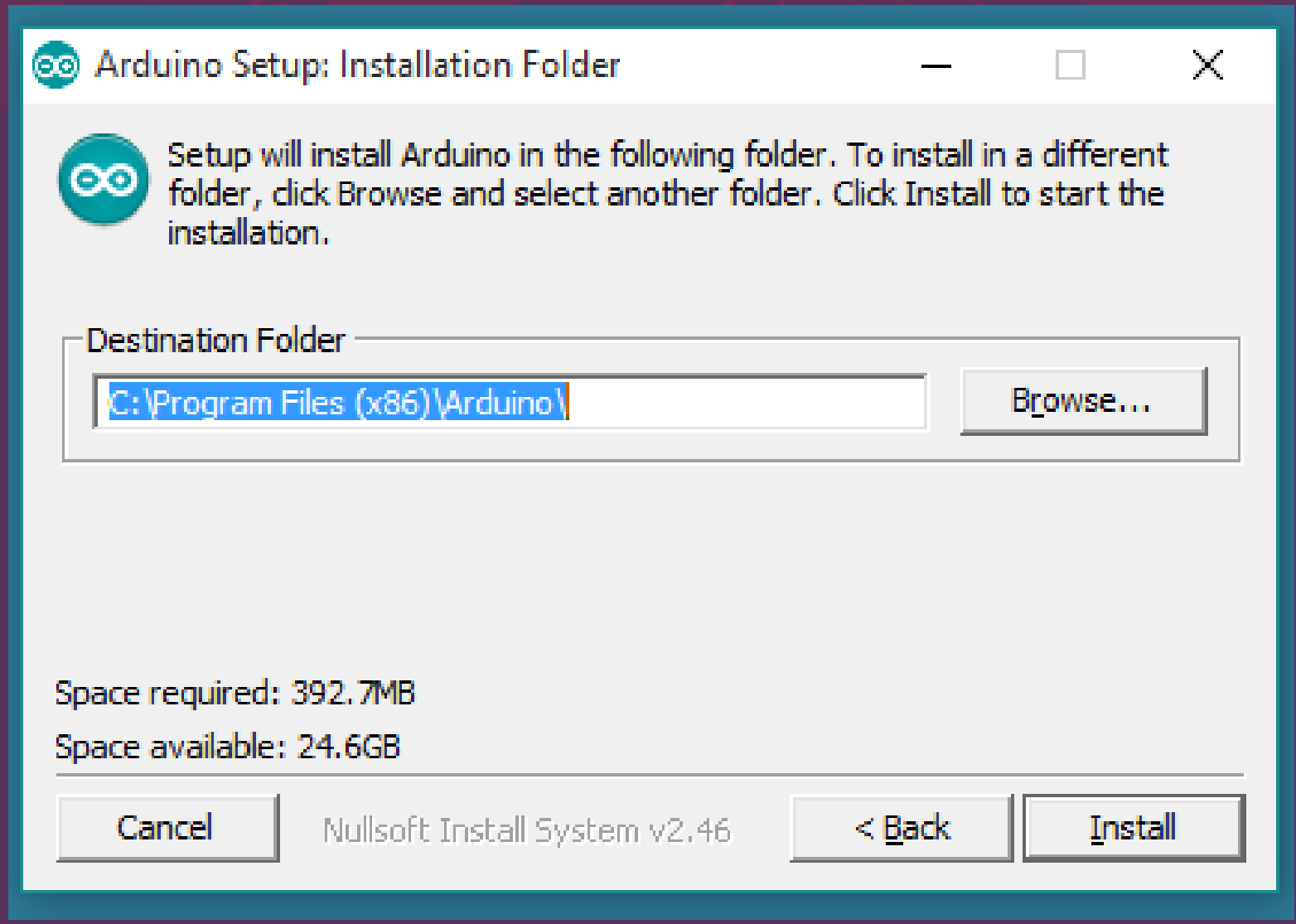


- Scroll down and click on the download option for your operating system

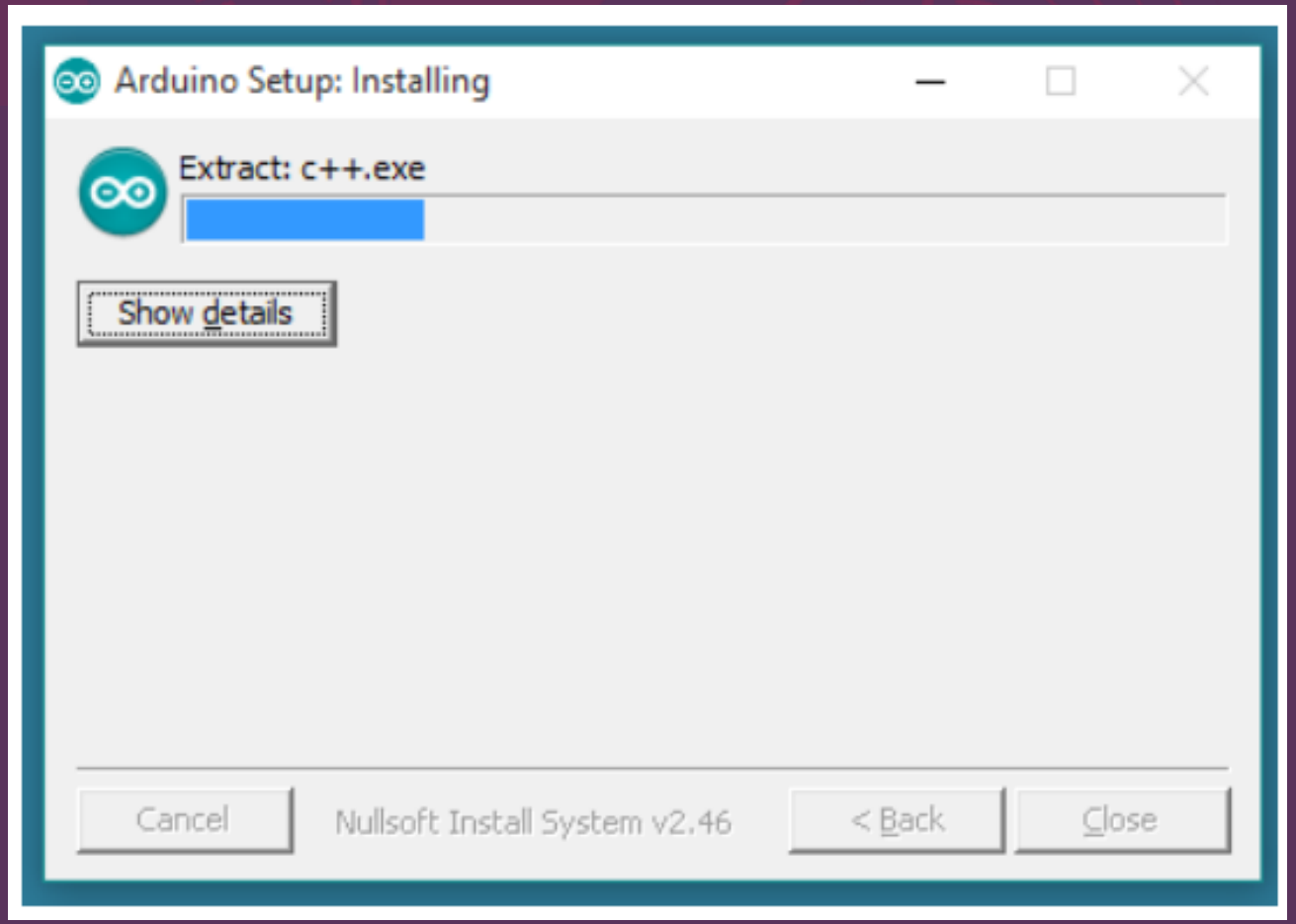


- When the download finishes, proceed with the installation and allow the driver installation process when you get a warning from the operating system.



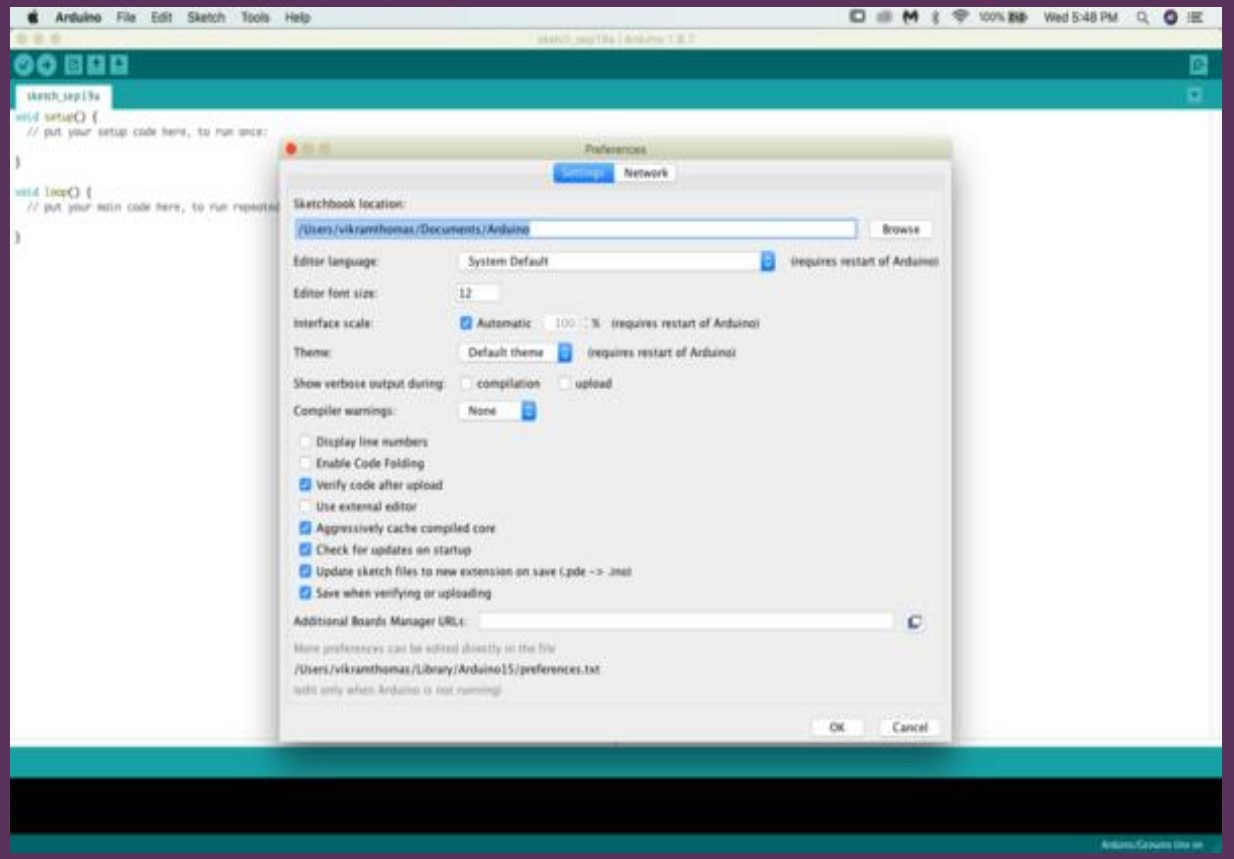

Arduino Setup: Installation Folder
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
C:\Program Files (x86)\Arduino\
Browse...
Space required: 392.7MB
Space available: 24.6GB
Cancel
Nullsoft Install System v2.46
< Back
Install

- Choose the installation directory (we suggest to keep the default one)

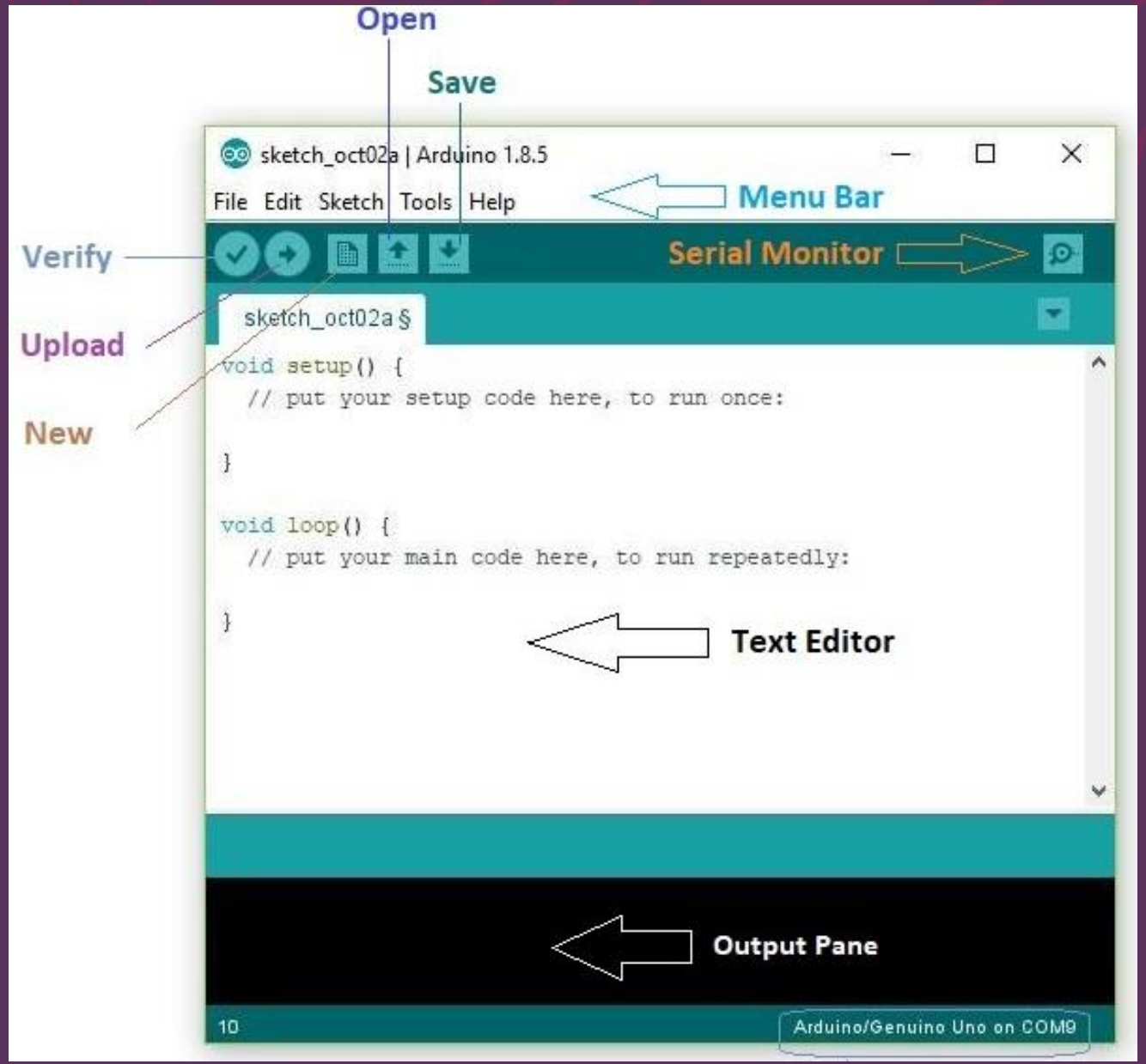


- The process will extract and install all the required files to execute properly the Arduino Software (IDE)

- Once you install the IDE the first thing to do is set up your preferences.
- You can do that by heading over to the preferences section under the Arduino tab.
- Over here you can control various different settings like where you want your sketches to be saved, font size, and a lot more.



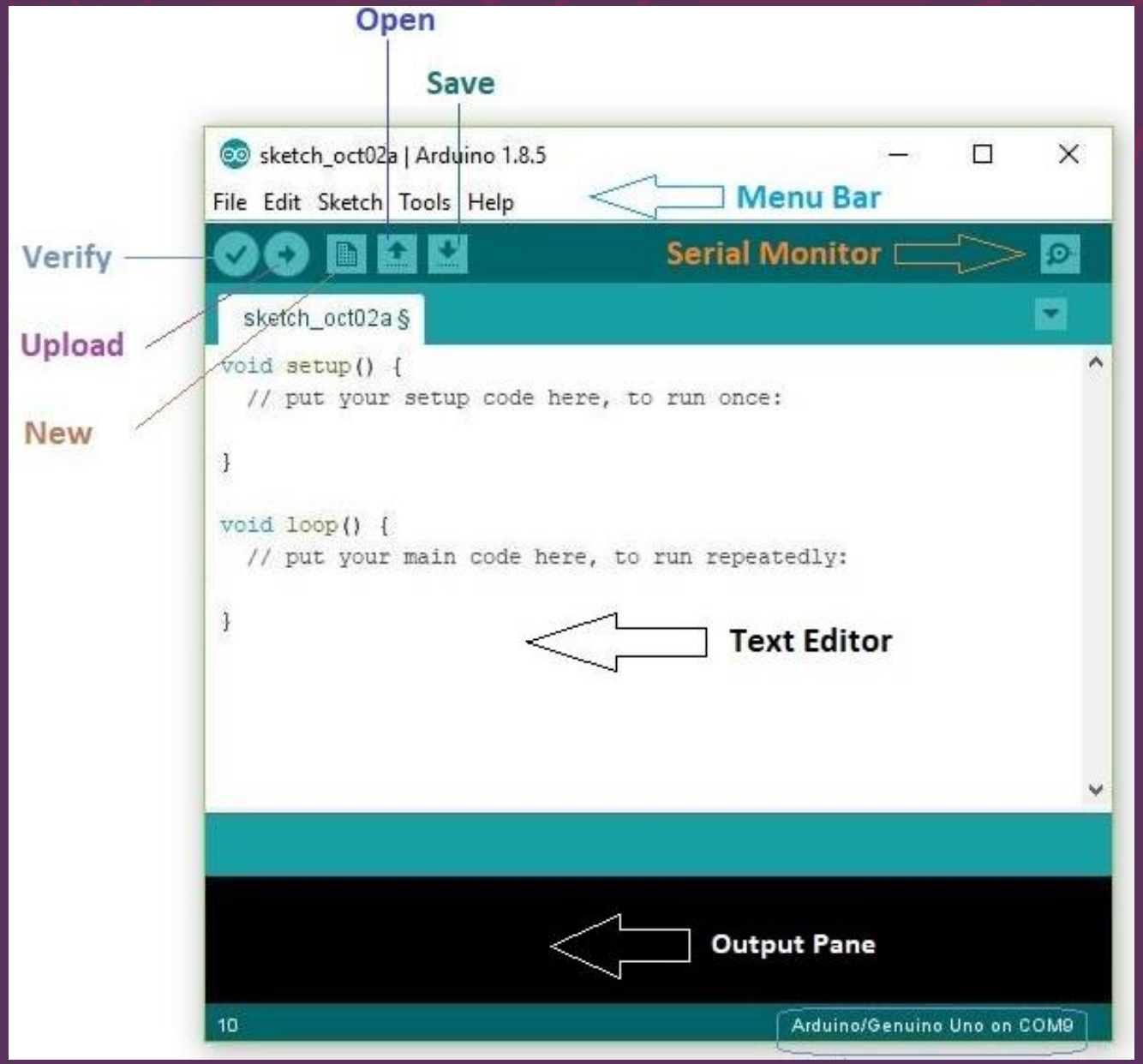
Arduino IDE



Once the software has been installed on your computer, go ahead and open it up.

Menu Bar: Gives you access to the tools needed for creating and saving Arduino sketches.

Verify Button: Compiles your code and checks for errors in spelling or syntax.

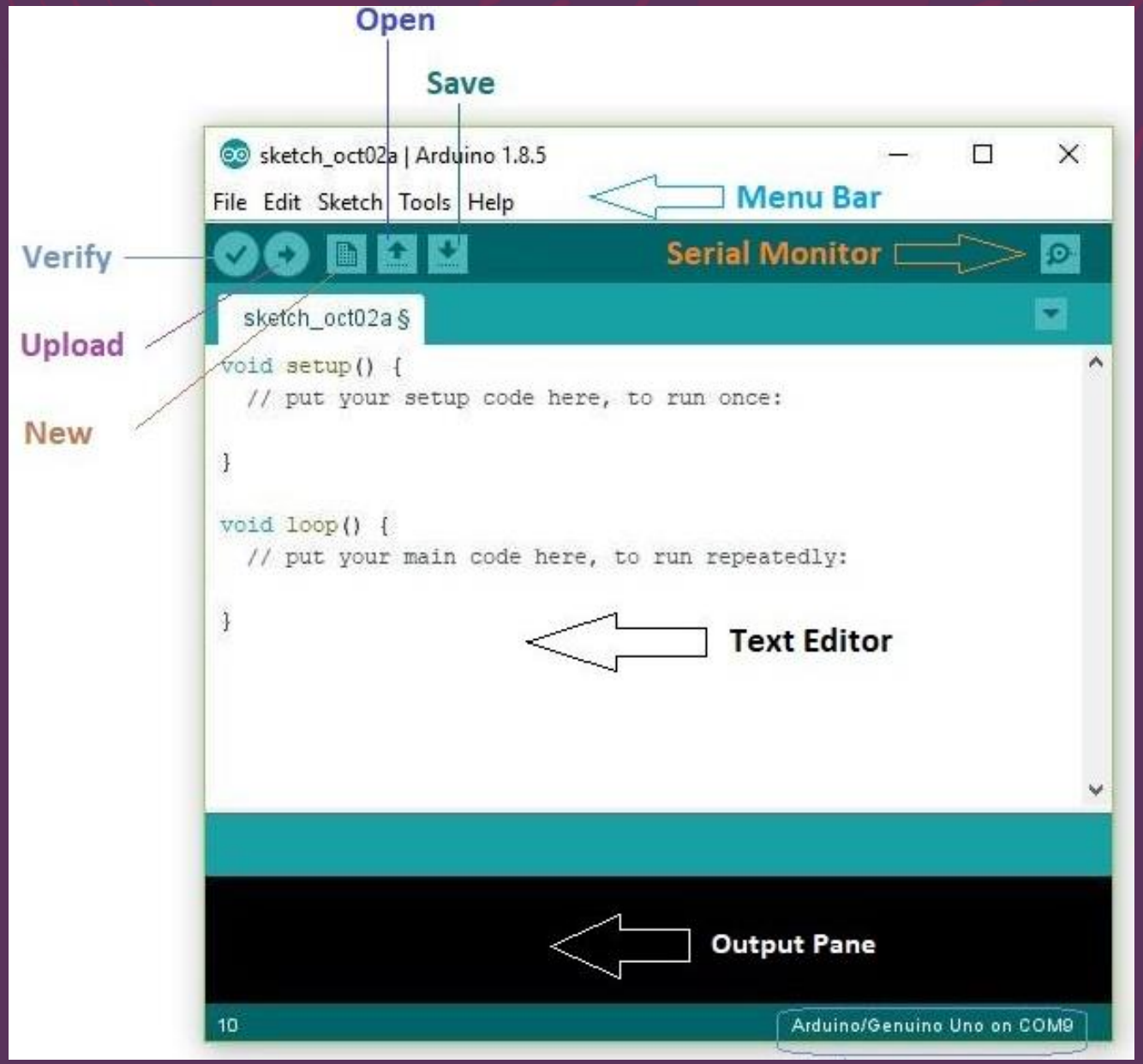


Upload Button

- The second button is the upload button, and it is used for uploading your code to the arduino board.

- The shortcut key for this button is 'cntrl + u'. When you use this button, you will normally see two LED's light up on your board, the TX and RX.

- These LED's light up when there is information being passed between the board and the IDE.



New Editor Button

- The new editor button opens up a new code editing window that you can use instead of the current one. You can use the shortcut key ‘cntrl + n’.

Open-File Button

- The open file button is used for opening a pre-existing file. You can use the shortcut key ‘cntrl + o’ for opening a file.

Save Button

- The save button can be used for saving your sketch. You can use the shortcut key ‘cntrl + s’

Arduino IDE

HOLOWORLD

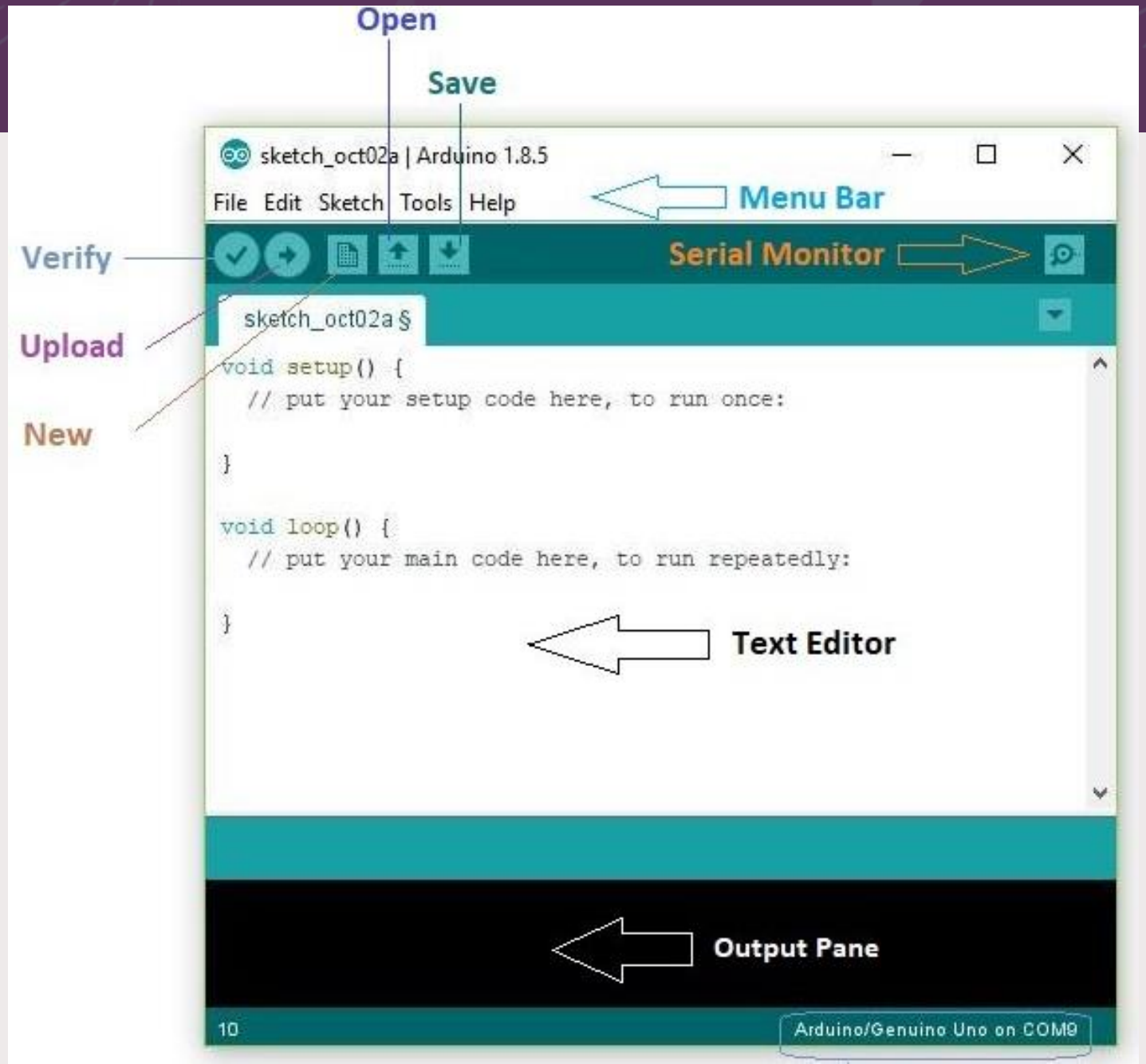
Serial Monitor: When the board is connected, this will display the serial information of your Arduino

Code Area: This area is where you compose the code of the sketch that tells the board what to do.

Message Area: This area tells you the status on saving, code compiling, errors and more.

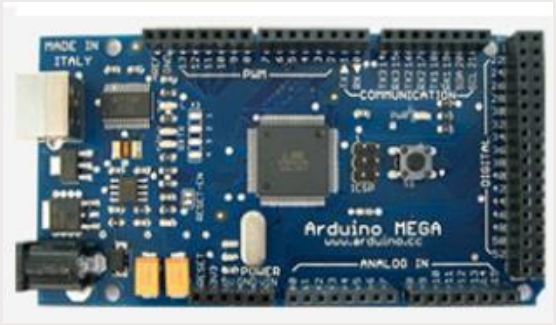
Text Console: Shows the details of an error messages, size of the program that was compiled and additional info.

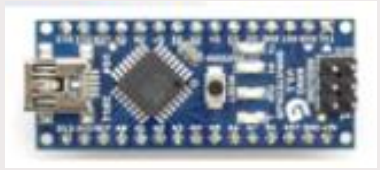
Board and Serial Port: Tells you what board is being used and what serial port it's connected to.

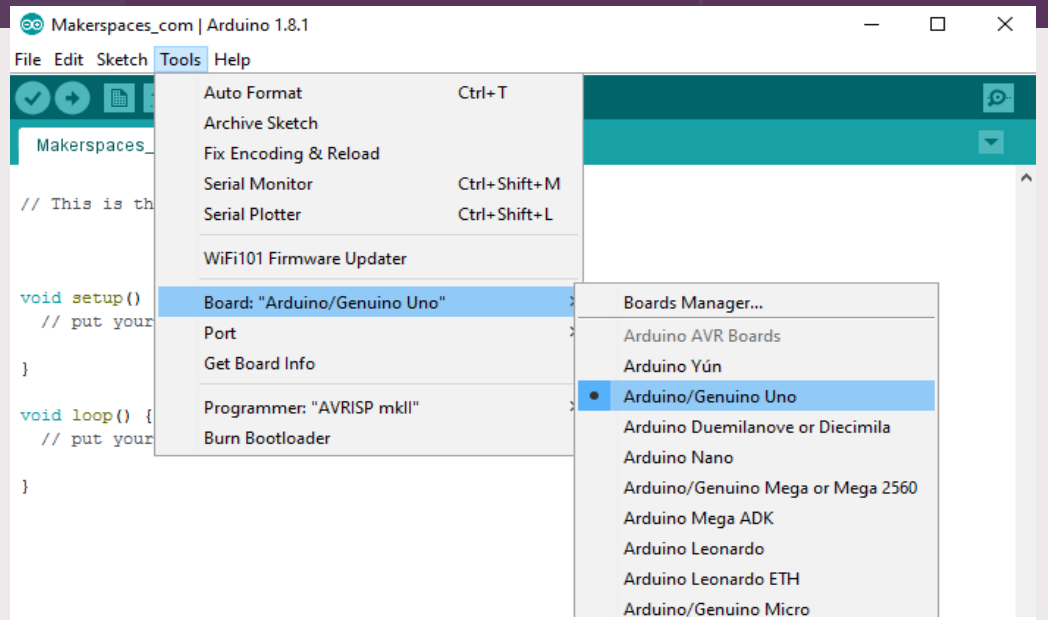


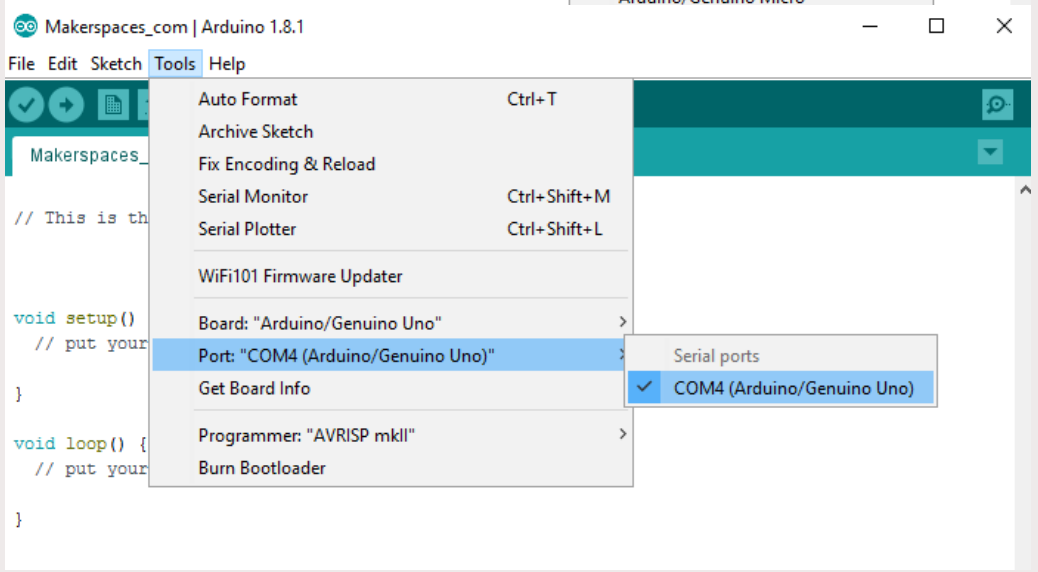
Connect Your Different Arduino boards

- At this point you are ready to connect your Arduino to your computer.
- Plug one end of the USB cable to the Arduino Uno and then the other end of the USB to your computer's USB port.
- Once the board is connected, you will need to go to Tools then Board then finally select Arduino Uno (for example)
- Next, you have to tell the Arduino which port you are using on your computer.
- To select the port, go to Tools then Port then select the port that says Arduino.









Pull-up resistor

Consider a circuit, which is a simple normally-open push-button on a breadboard. It has been wired so that one side is tied to +5V and the other side is connected to Pin 12 of an Arduino Uno.

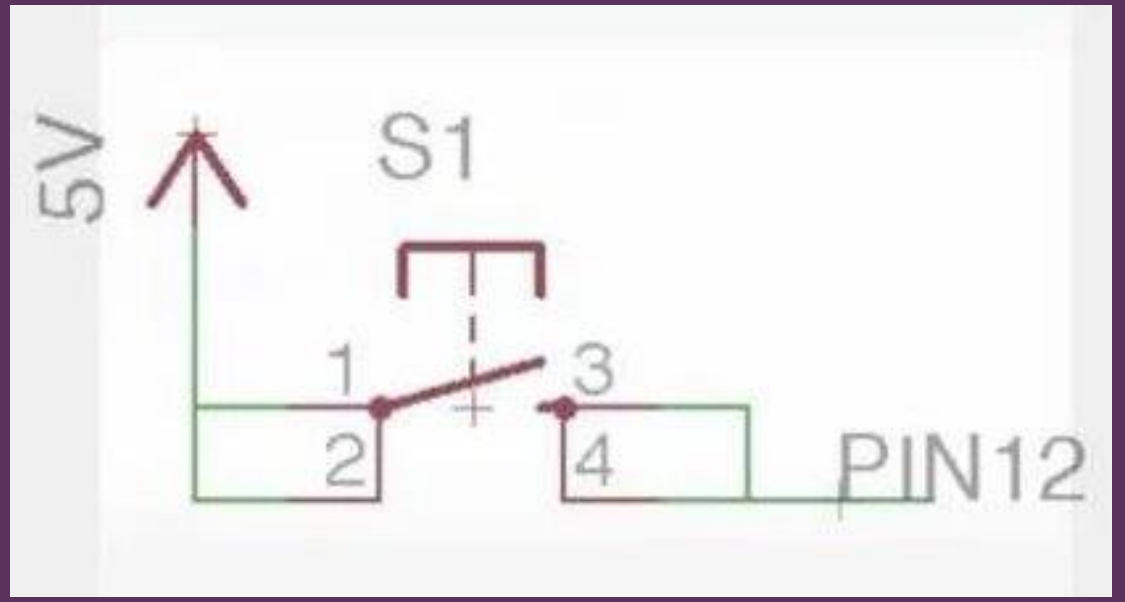
The code for this switch is very simple. The LED on Pin 13 should light up when Pin 12 is HIGH and be off when Pin 12 is LOW. The expected behavior is that the LED will be OFF whenever the button is not pushed and ON when the button is pushed

- The fix for floating pins is to "pull them up" to a known value when the switch is unpressed.
- This is done with a Pull-Up resistor.

Setup Function	**void setup**()
	{
	pinMode(13, **OUTPUT**);
	// Use Built-In LED for Indication
	pinMode(12, **INPUT**);
	// Push-Button On Bread Board
	}

Loop Function	**void loop**()
	{
	bool buttonState = digitalRead(12);
	// store current state of pin 12
	digitalWrite(13, buttonState);
	}

- This is done with a Pull-Up resistor, as illustrated in the following schematic.
- Now when nothing is connected, current cannot flow through the resistor.
- So, the Voltage on both legs will be the same.
- This means the same point where the resistor connects to the switch and Pin 12 will be forced to sit at 5V.
- When the button is pressed, that same point will drop to 0V as current starts flowing through the resistor (ohm's law).



- All that needs to be done is turn the Arduino Internal Pull-Up resistor on and you get the previous schematic, When using any kind of “open” inputs with an Arduino such as switches, push buttons, reed relays, and some sensors a pull-up resistor is needed for reliable operation.

- These resistors hold the I/O pin at a known value until the switch forces the I/O pin to a different known value. On each board there are Arduino Internal Pull-Up resistors built-in, they just need to be turned on in the sketch, usually in setup().

Without pull-up resistor

Setup Function	**void setup**()
	{
	pinMode(13, **OUTPUT**);
	// Use Built-In LED for Indication
	pinMode(12, **INPUT**);
	// Push-Button On Bread Board
	}

Loop Function	**void loop**()
	{
	bool buttonState = digitalRead(12);
	// store current state of pin 12
	digitalWrite(13, buttonState);
	}

With pull-up resistor

Setup Function	**void setup**()
	{
	pinMode(13, **OUTPUT**);
	// Use Built-In LED for Indication
	/* INPUT_PULLUP enables the Arduino Internal Pull-Up Resistor */
	pinMode(12, INPUT_PULLUP);
	// Push-Button On Bread Board
	}

Loop Function	**void loop**()
	{
	bool buttonState = digitalRead(12);
	// store current state of pin 12
	digitalWrite(13, buttonState);
	}

Arduino Programming

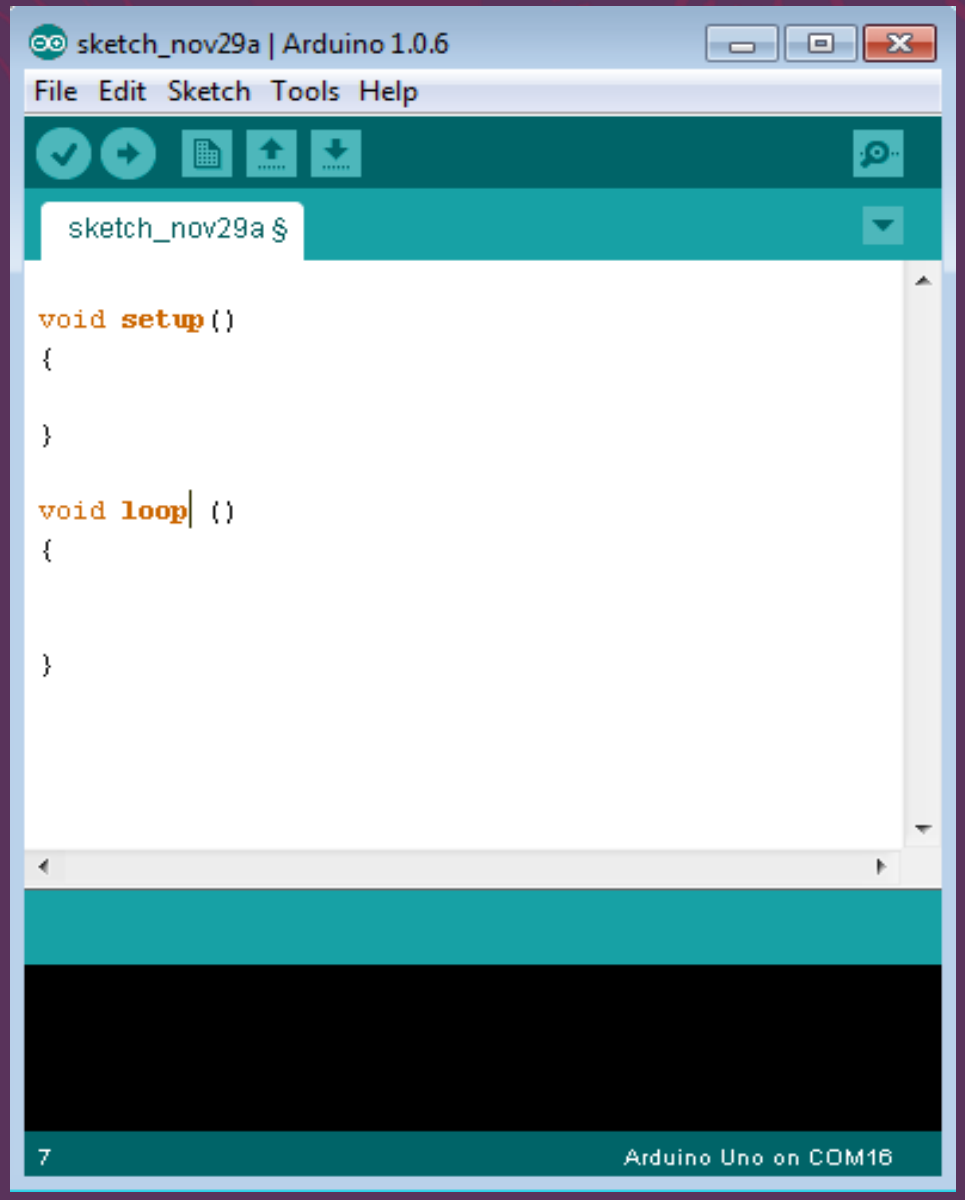
Block Programs

Script Programs

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

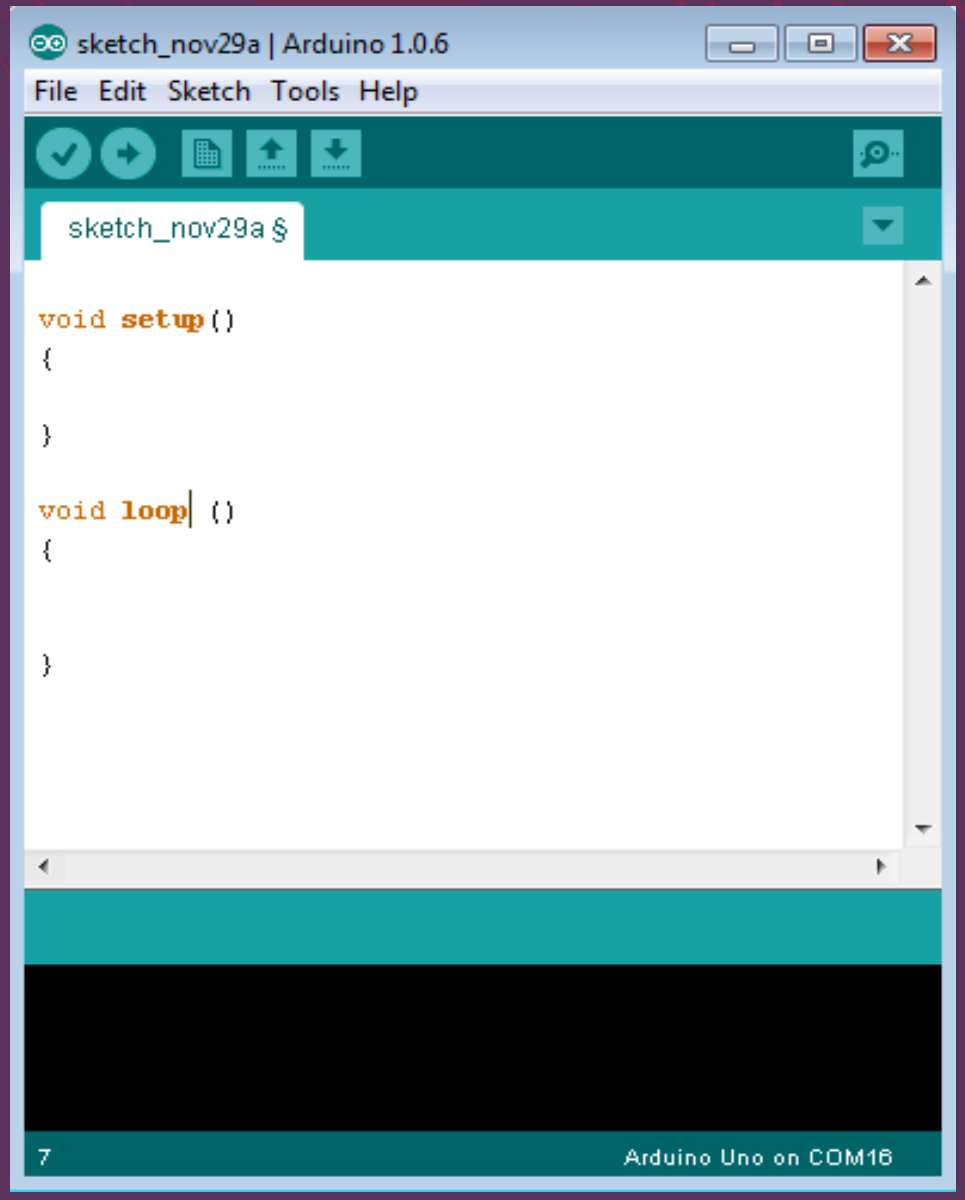
Program Structure

- The structure of Arduino program is pretty simple. Arduino programs have a minimum of 2 blocks,
- Preparation & Execution
- Each block has a set of statements enclosed in curly braces:
- Here, setup ( ) is the preparation block and loop ( ) is an execution block.
- The setup function is the first to execute when the program is executed, and this function is called only once.
- The setup function is used to initialize the pin modes and start serial communication.
- The setup function has to be included even if there are no statements to execute.



Program Structure

- After the setup ( ) function is executed, the execution block runs next.
- The execution block hosts statements like reading inputs, triggering outputs, checking conditions etc.
- As the name suggests, the loop( ) function executes the set of statements (enclosed in curly braces) repeatedly.



Setup Function	**void setup ( )** **{** **pinMode (pin_number, OUTPUT);** **// set the 'pin-number' as output** **pinMode (pin_number, INPUT);** **// set the 'pin-number' as output** **}**
Loop Function	**void loop ( )** **{** **digitalWrite (pin_number, HIGH);** **// turns ON the component connected to 'pin-number'** **delay (1000);** **// wait for 1 sec** **digitalWrite (pin_number, LOW);** **// turns OFF the component connected to 'pin-number'** **delay (1000);** **//wait for 1sec** **}**

Major Arduino Functions

Each of the digital pins on the Arduino boards can be used as an input or output using the below 3 functions

- pinMode()
- digitalWrite()
- digitalRead()

pinMode()

- Configures the specified pin to behave either as an input or an output. Additionally, the INPUT mode explicitly disables/enables the internal pullups.

Syntax: pinMode(pin, mode)

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.
- If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin.
- Syntax: digitalWrite(pin, value)

digitalRead()

- Reads the value from a specified digital pin, either HIGH or LOW.
- Syntax: digitalRead(pin)

Setup Function	**void setup ( )** **{** **pinMode (pin_number, OUTPUT);** **// set the 'pin-number' as output** **pinMode (pin_number, INPUT);** **// set the 'pin-number' as output** **}**
Loop Function	**void loop ( )** **{** **digitalWrite (pin_number, HIGH);** **// turns ON the component connected to 'pin-number'** **delay (1000);** **// wait for 1 sec** **digitalWrite (pin_number, LOW);** **// turns OFF the component connected to 'pin-number'** **delay (1000);** **//wait for 1sec** **}**

Each of the Analog pins on the Arduino boards can be used as an input or output, using the below 3 functions:

1.pinMode()
2.analogWrite()
3.analogRead()

pinMode()

- Configures the specified pin to behave either as an input or an output.
- Syntax: pinMode(pin, mode)

analogWrite()

- Writes an analog value (PWM wave) to a pin.
- Can be used to light a LED at varying brightness 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 another function
- The analogWrite function has nothing to do with the analog pins or the analogRead function.
- Syntax: analogWrite(pin, 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.
- Syntax: `analogRead(pin, value)`

Serial Functions

a) Serial.begin(baud_rate)

- `baud_rate` : The baud rate that will be used for serial communication. Can be 4800, 9600, 14400, 19200, etc.
- This function is used to define the baud rate that will be used for serial communication. For communicating with specific devices, the device baud rate needs to be used.

b) Serial.available()

- This function is used to get the number of bytes available for reading from the serial port.
- if(Serial.available())
  If data available at serial port, take action.

c) Serial.print(value)

- This function is used to print data to a serial port in a form that is human readable (character, strings, numbers).
- `Example` Serial.print(“Hi 1234”)

  Hi 1234
- `Example` Serial.println(“Hi”)

  Serial.println("1234")

  Hi

  1234

d) Serial.read()

- This function returns a character that was received on the Rx pin of Arduino.
- `Example` char read_byte

  read_byte = Serial.read()

  Byte of data read is stored in read_byte.

e) Serial.write(value), Serial.write(string), Serial.write(buff, length)

- `value` : value to be sent as a single byte.
- `string` : string to be sent as a series of bytes.
- `buff` : array of data to be sent as bytes.
- `length` : number of bytes to be sent.
- This function writes data in binary form to the serial port. Data is sent in form of bytes or series of bytes.
- `Example` Serial.write(100)
  Serial.write("Hello")

Serial Monitor Example programs

- Arduino program to print "Hello World" message in the "Serial Monitor"

Setup Function	void setup()
	{
	Serial.begin(9600);
	Serial.println("Hello, world!");
	}

Loop Function	void loop()
	{
	}

begin(9600)

- This starts serial communication, so **that the Arduino can send out commands through the USB connection**.
- The value 9600 is called the 'baud rate' of the connection.
- This is how fast the data is to be sent.

Data types.

Data type is an attribute associated with a piece of data that tells a computer system how to interpret its value. Understanding data types ensures that data is collected in the preferred format and the value of each property is as expected.

Integer (int)

- It is the most common numeric data type used to store numbers without a fractional component (3707, 0, 707).

Floating Point (float)

- It is also a numeric data type used to store numbers that may have a fractional component (707.07, 0.7, 707.00).
- It can store decimal values upto 6 to 7 demicals places.

const int

- is **a constant pointer to integer**. This means that the variable being declared is a constant pointer pointing to an integer. Effectively, this implies that the pointer shouldn't point to some other address
- The difference between int and const int is that **int is read/write while const int is read-only**.

Decimal point values

- Similar to float decimcal can stores fractional numbers.
- Sufficient for storing 15 decimal digits

Setup function
Declaring I/O
Variables etc.
	const int LED = 15;
	void setup ( )
	{
	pinMode (LED, OUTPUT);
	//Declaring pin 15 as output pin
	}

Loop function to
Execute the program
for infinity time
	void loop( )
	// The loop function runs again and again
	{
	digitalWrite (LED, HIGH);
	//Turn ON the LED
	
	delay(1000); //Wait for 1sec
	
	digitalRead (LED, LOW);
	// Turn off the LED
	
	delay(1000); // Wait for 1sec
	}

Other common data types:

- Char – character type data ("a" "b" "#" "$" "5" "7" etc)

- String – store many character type data values

  ("hello" , "23408987" , "@pple" , "@#$%^" , "This is an example")

- Boolean – store only either true or false value

Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Comparison Operators
- Boolean Operators

Comparision Operators

Operator name	Operator simple	Description	Example
equal to	==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true
not equal to	!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true
less than	<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true
greater than	>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true
less than or equal to	<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true
greater than or equal to	>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true

```
void loop () {
  int a = 9,b = 4
  bool c = false;
  if(a == b)
    c = true;
  else
    c = false;
if(a != b)
    c = true;
  else
    c = false;
  if(a < b)
    c = true;
  else
    c = false;
  if(a > b)
    c = true;
  else
    c = false;
  if(a <= b)
    c = true;
  else
    c = false;
  if(a >= b)
    c = true;
  else
    c = false;
}
```

Result

```
c = false
c = true
c = false
c = true
c = false
c = false
```

Boolean Operators

Operator name	Operator simple	Description	Example
and	&&	Called Logical AND operator. If both the operands are non-zero then then condition becomes true.	(A && B) is true
or	||	Called Logical OR Operator. If any of the two operands is non-zero then then condition becomes true.	(A || B) is true
not	!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false

Example

```
void loop () {
   int a = 9,b = 4
   bool c = false;
   if((a > b)&& (b < a))
      c = true;
   else
      c = false;

   if((a == b)|| (b < a))
      c = true;
   else
      c = false;

   if( !(a == b)&& (b < a))
      c = true;
   else
      c = false;
}
```

Result

```
c = false
c = true
c = false
c = true
c = false
c = false
```

Arithmetic Operators

Operator name	Operator simple	Description	Example
assignment operator	=	Stores the value to the right of the equal sign in the variable to the left of the equal sign.	A = B
addition	+	Adds two operands	A + B will give 30
subtraction	-	Subtracts second operand from the first	A - B will give -10
multiplication	*	Multiply both operands	A * B will give 200
division	/	Divide numerator by denominator	B / A will give 2
modulo	%	Modulus Operator and remainder of after an integer division	B % A will give 0

Example

Example

```
void loop () {
    int a = 9,b = 4,c;
    c = a + b;
    c = a - b;
    c = a * b;
    c = a / b;
    c = a % b;
}
```

Result

```
a + b = 13
a - b = 5
a * b = 36
a / b = 2
Remainder when a divided by b = 1
```

Let us do some Coding

Program to add two number:

Setup Function	void setup()
	{
	Serial.begin(9600);
	}

Loop Function	void loop()
	{
	int a = 100;
	int b = 20;
	int sum = a + b;
	
	Serial.print(a);
	Serial.print(" + ");
	Serial.print(b);
	Serial.print(" = ");
	Serial.println(sum);
	delay(1000);
	}

Program to divide two numbers

Setup Function	void setup()
	{
	Serial.begin(9600);
	}

Loop Function	void loop()
	{
	float a = 100;
	float b = 7;
	float result = a / b;
	
	Serial.print(a);
	Serial.print(" / ");
	Serial.print(b);
	Serial.print(" = ");
	Serial.println(result);
	delay(1000);
	}

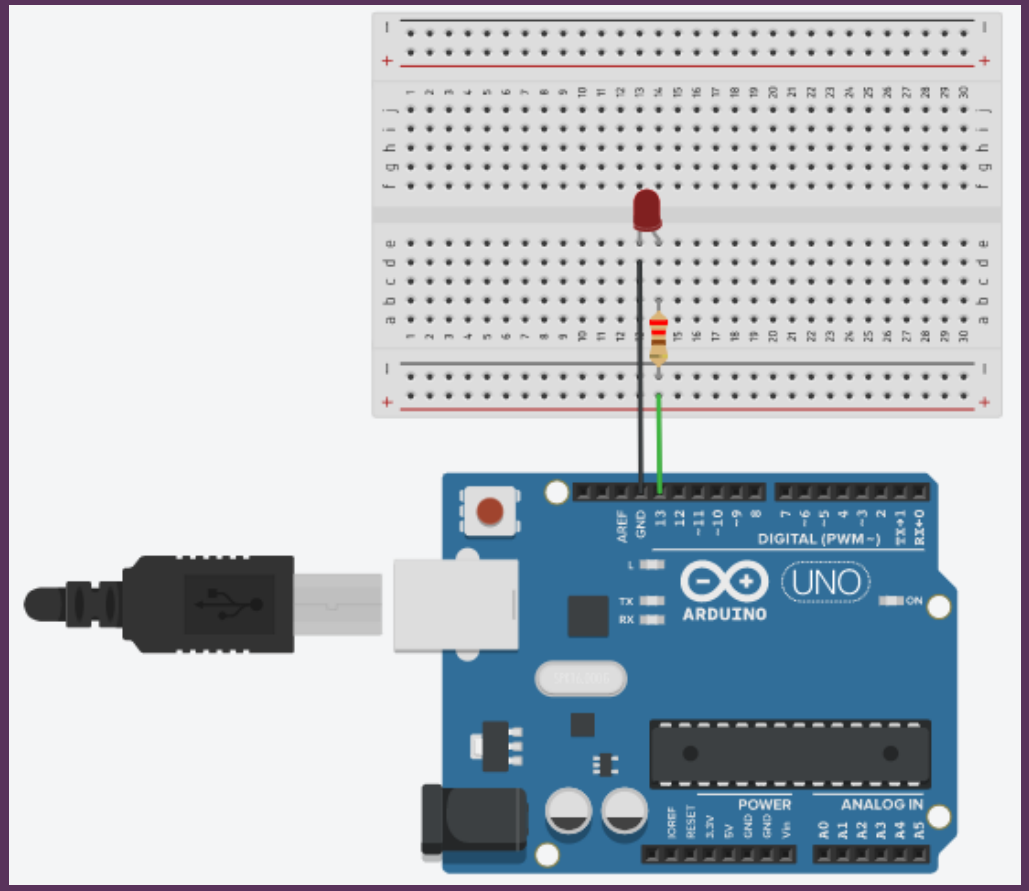
Exercise: Find area of circle

Setup Function	void setup()
	{
	Serial.begin(9600);
	}

Loop Function	void loop()
	{
	// calculate the area of a circle with radius of 9.2
	float radius = 9.2;
	float pi = 3.14;
	float area = pi * radius * radius;
	Serial.print("Area of circle is: ");
	// print area to 4 decimal places
	Serial.println(area, 4);
	}

- Radius = 9.2
- Pi = 3.14
- Area = pi * radius * radius

LED Blinking



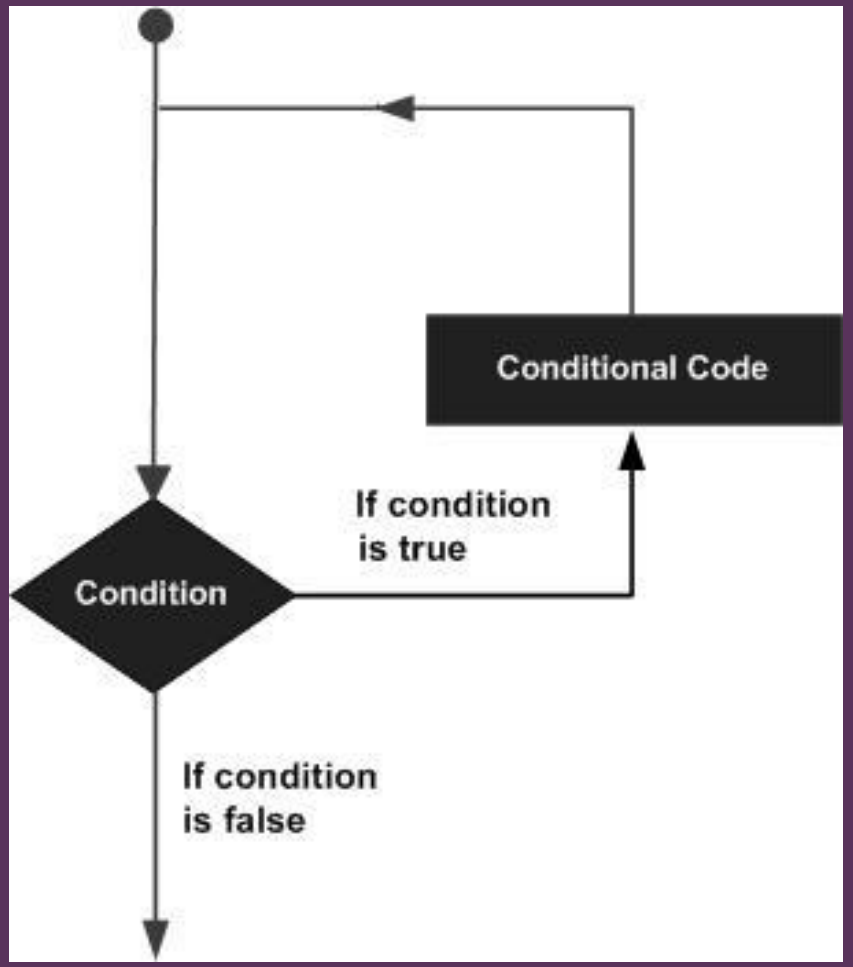
Setup function
Declaring I/O
Variables etc.
	void setup ( )
	{
	pinMode (13, OUTPUT);
	//Declaring pin 13 as output pin
	
	}

Loop function to
Execute the program
for infinity time
	void loop( )
	// The loop function runs again and again
	{
	digitalWrite (13, HIGH);
	//Turn ON the LED
	
	delay(1000); //Wait for 1sec
	
	digitalWrite (13, LOW);
	// Turn off the LED
	
	delay(1000); // Wait for 1sec
	}

Arduino - Loops

- Programming languages provide various control structures that allow for more complicated execution paths.

- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages



for statements

- Description
- The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop.
- The for statement is useful for any repetitive operation and is often used in combination with arrays to operate on collections of data/pins.
- There are three parts to the for loop header:

```
for (initialization; condition; increment) {
    //statement(s);
}
```

- The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again.
- When the condition becomes false, the loop ends.

Initializing the variables	`const int pinLed = 10;` `// LED is attached to pin 10`

Setup function
Declaring I/O,
Pinmode
Variables etc.
	`void setup() {`
	
	`// set pin 10 as an output pin`
	`pinMode(pinLed , OUTPUT);`
	
	`// turn the LED off at beginning`
	`digitalWrite(pinLed , LOW);`
	
	`//start serial connection`
	`Serial.begin(9600);`
	`}`

Loop function to
Execute the program
for infinity time
	`void loop() {`
	`for (int i = 0; i < 3; i++) {`
	`// turn the LED on`
	`digitalWrite(pinLed , HIGH);`
	`// wait for 1 second`
	`delay(1000);`
	`// turn the LED off`
	`digitalWrite(pinLed , LOW);`
	`// wait for 1 second`
	`delay(1000);`
	`} // closing the for loop()`
	`}`

- The for loop will run three times through the code.
- That is because we declared an integer variable called **i.**
- If **i** is less than three the code between the curly brackets is executed.
- Furthermore, if **i** is three the code is not executed and everything in the for loop() is ignored.

If statement in Arduino

- The conditional statement is one which is used often when using sensors with Arduino.
- An if statement must have a test within the parentheses (….) that can result in being true or false.

The **if statement** evaluates the test inside the parenthesis().

- If the test is evaluated to be true, statements inside the brackets are executed.
- If the test is evaluated to be false, statements inside the brackets are not executed.

Conditions in an if statement: the conditional part if the **if** statement includes the boolean expression, which can be true or false.

Eg. You may want to increase the led brightness by 5 points up to 255.

Fading LED

Initializing the variables	int brightness = 0;
	// Brightness of LED is initially set to 0
	
	int fade = 5;
	// By how many points the LED should fade
	const int led = 10;

Setup function
Declaring I/O, Pinmode
Variables etc.
	void setup()
	{
	pinMode(led, OUTPUT);
	//pin 10 is set as output pin
	}

Loop function to
Execute the program
for infinity time
	void loop()
	{
	analogWrite(led, brightness);
	// set the brightness of LED
	
	brightness = brightness + fade;
	//Increase the brightness of LED by 5 points
	
	if (brightness >= 0 || brightness <= 255)
	// check the level of brightness
	
	{
	fade ++;
	}
	delay(30); // Wait for 30 milliseconds
	}

if … else statement

- Most of the time you will also want to code a part that is executed when the statement is False. This can be done by using an else statement after the if statement.
- You could use only an if statement in some cases but the if … else statement allows greater control over the flow of a code than the basic if statement.
- An else clause (if at all exists) will be executed if the condition in the if statement is false.
- You could use several **if** statements until a true test is encountered. Again, when the true test is found, the code written in that block will run.
- You can have an unlimited amount of **else….if** conditions in a sketch.

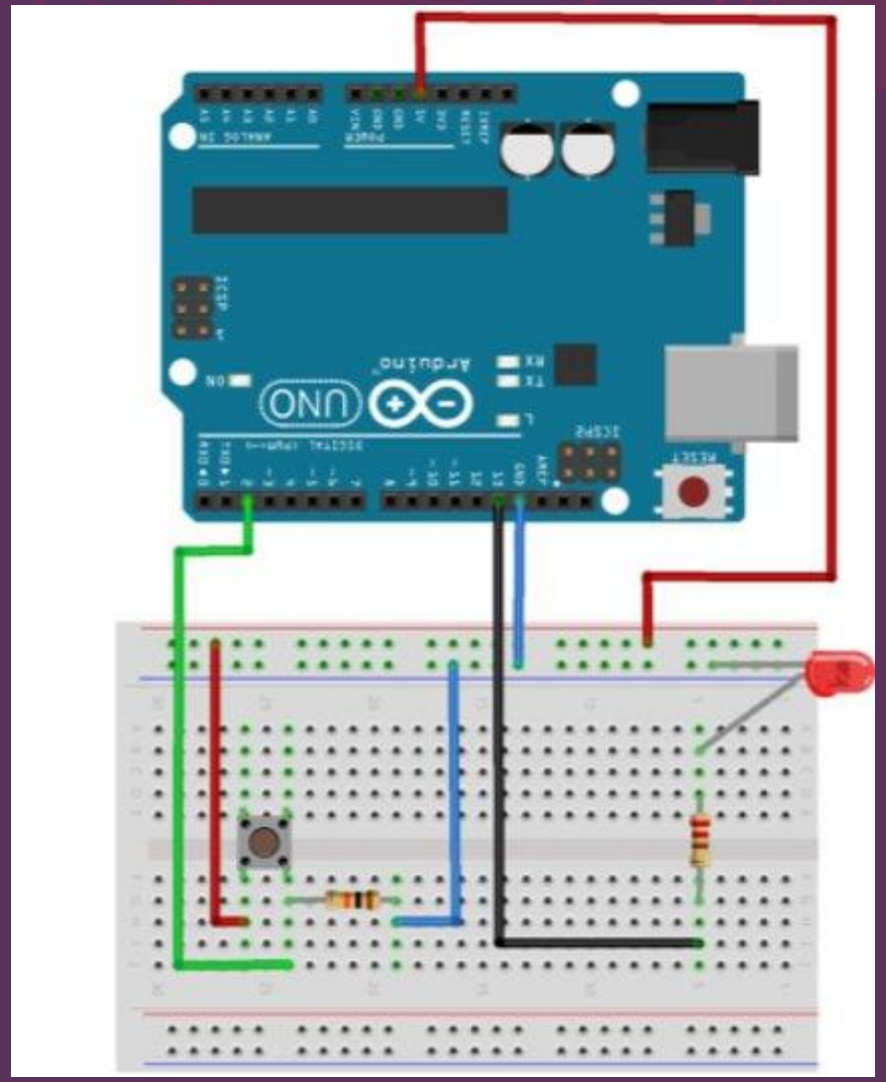
LED with Switch

- A switch is an electrical component that completes a circuit when pushed and breaks the circuit when released. In this project, we will be using a small pushbutton switch to control an LED.

  - Arduino UNO R3
  - Breadboard
  - Connecting wire
  - LED –red
  - 1kΩ and 220Ω resistor
  - Push to ON switch

- In order to use a switch, we have to first load the file called “Button” which can be found here:

- **File > Examples > Digital > Button**



```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

while loops

- A `while` loop will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false.
- Something must change the tested variable, or the while loop will never exit.
- This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

while loop Syntax:	while(expression)
	{
	Block of statements;
	}

- condition: a boolean expression that evaluates to true or false.

Example Code

```
var = 0;
while (var < 200) {
  // do something repetitive 200 times
  var++;
}
```

do… while loop

- The do… while loop works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

do…while loop Syntax:	do
	{
	Block of statements;
	}
	while (expression);

- condition: a boolean expression that evaluates to true or false.

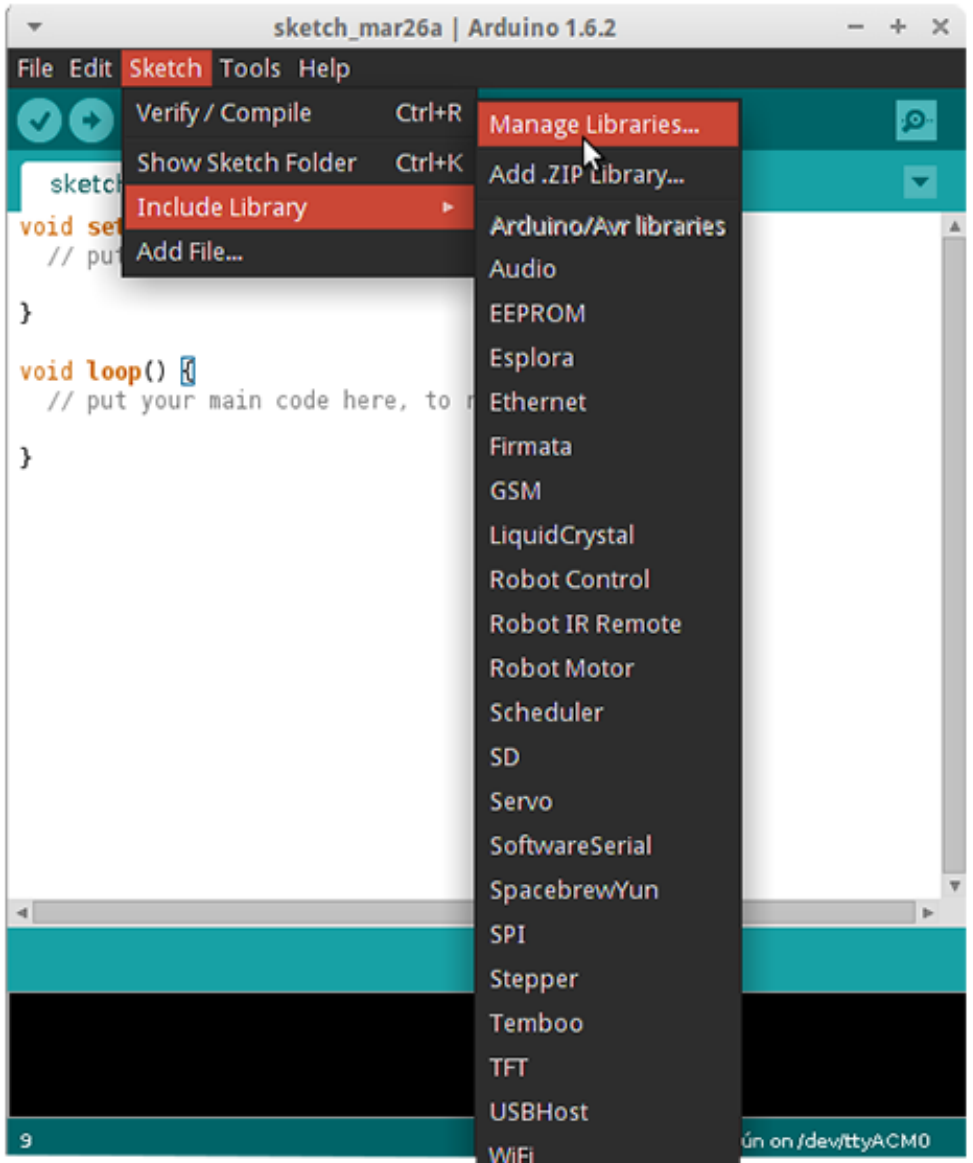
Example Code

```
int x = 0;
do {
  delay(50);            // wait for sensors to stabilize
  x = readSensors();    // check the sensors
} while (x < 100);
```

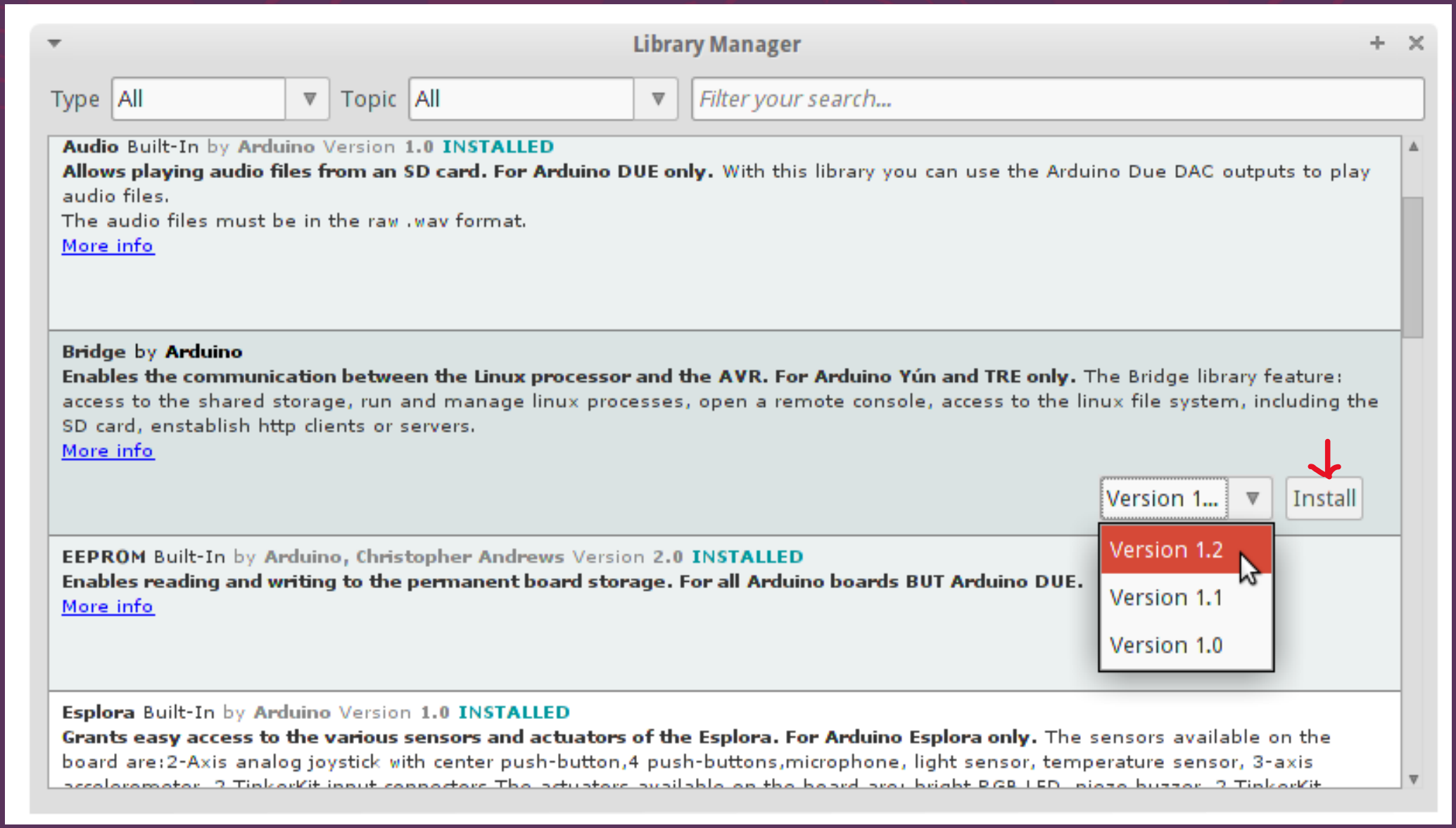
Adding libraries to Arduino

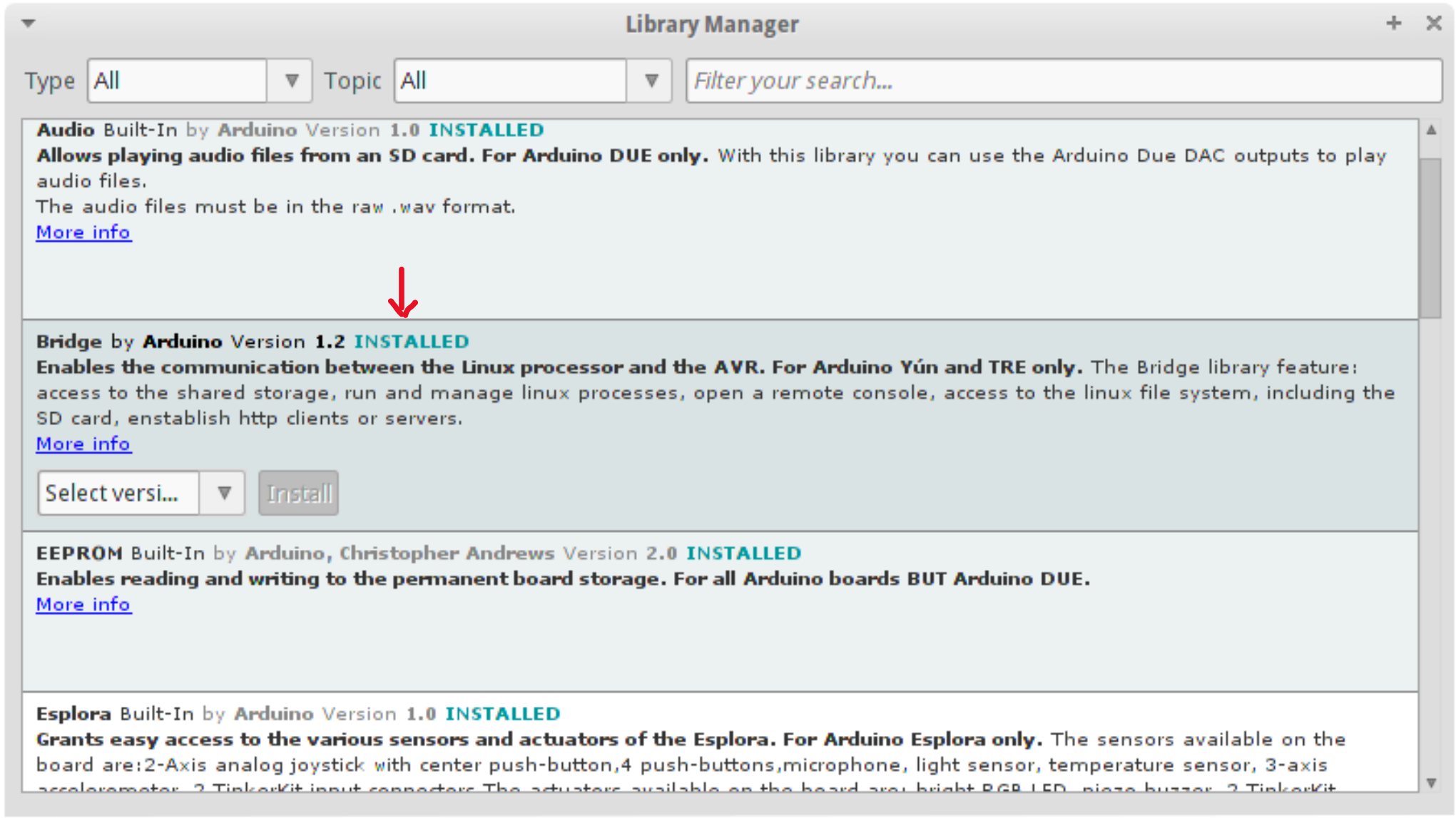
- Libraries are **files written in C or C++** which provide your sketches with extra functionality (e.g., the ability to control an LED matrix, or read an encoder, etc.).
- To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.6.2).
- Open the IDE and click to the "Sketch" menu and then *Include Library* > *Manage Libraries*.

Sketch > Include Library > Manage Libraries



- Then the Library Manager will open, and you will find a list of libraries that are already installed or ready for installation.

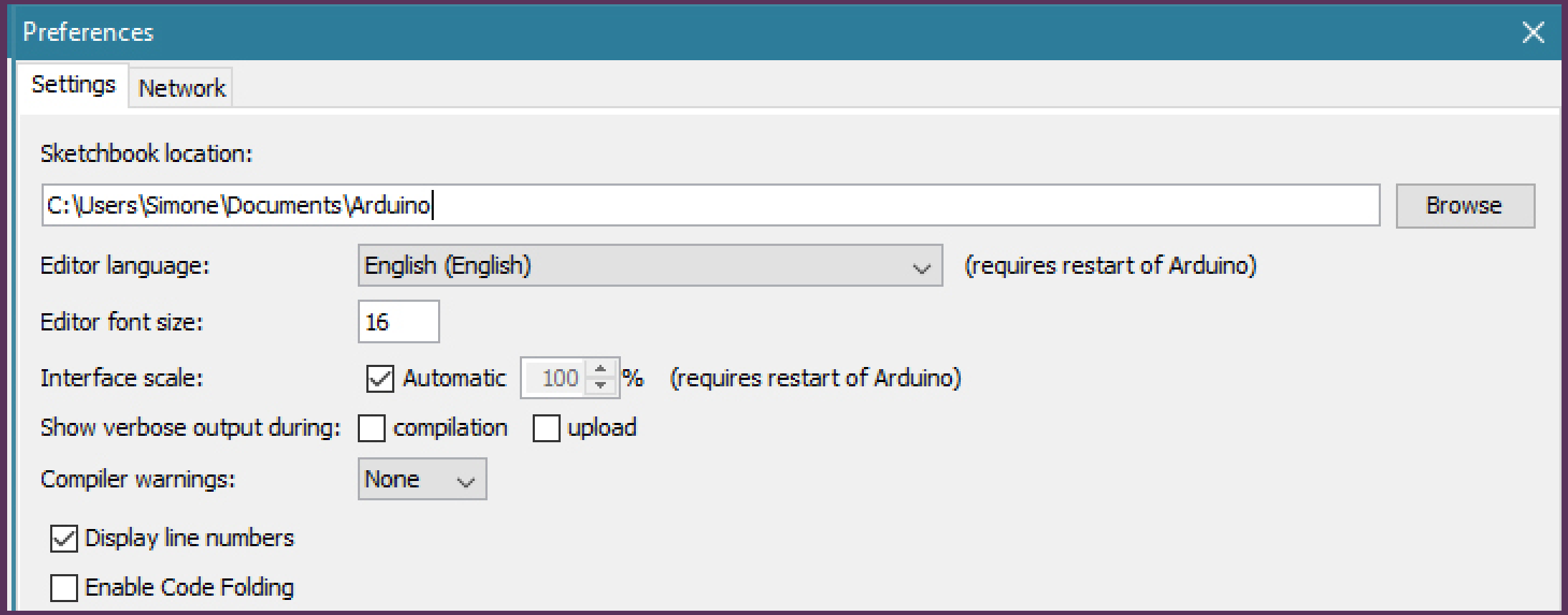

Library Manager
Type All
Topic All
Filter your search...
Audio Built-In by Arduino Version 1.0 INSTALLED
Allows playing audio files from an SD card. For Arduino DUE only. With this library you can use the Arduino Due DAC outputs to play audio files.
The audio files must be in the raw .wav format.
More info
Bridge by Arduino
Enables the communication between the Linux processor and the AVR. For Arduino Yún and TRE only. The Bridge library feature: access to the shared storage, run and manage linux processes, open a remote console, access to the linux file system, including the SD card, enstablish http clients or servers.
More info
Version 1...
Install
Version 1.2
Version 1.1
Version 1.0
EEPROM Built-In by Arduino, Christopher Andrews Version 2.0 INSTALLED
Enables reading and writing to the permanent board storage. For all Arduino boards BUT Arduino DUE.
More info
Esplora Built-In by Arduino Version 1.0 INSTALLED
Grants easy access to the various sensors and actuators of the Esplora. For Arduino Esplora only. The sensors available on the board are:2-Axis analog joystick with center push-button,4 push-buttons,microphone, light sensor, temperature sensor, 3-axis


Library Manager
Type All
Topic All
Filter your search...
Audio Built-In by Arduino Version 1.0 INSTALLED
Allows playing audio files from an SD card. For Arduino DUE only. With this library you can use the Arduino Due DAC outputs to play audio files.
The audio files must be in the raw .wav format.
More info
Bridge by Arduino Version 1.2 INSTALLED
Enables the communication between the Linux processor and the AVR. For Arduino Yún and TRE only. The Bridge library feature: access to the shared storage, run and manage linux processes, open a remote console, access to the linux file system, including the SD card, enstablish http clients or servers.
More info
Select versi...
Install
EEPROM Built-In by Arduino, Christopher Andrews Version 2.0 INSTALLED
Enables reading and writing to the permanent board storage. For all Arduino boards BUT Arduino DUE.
More info
Esplora Built-In by Arduino Version 1.0 INSTALLED
Grants easy access to the various sensors and actuators of the Esplora. For Arduino Esplora only. The sensors available on the board are:2-Axis analog joystick with center push-button,4 push-buttons,microphone, light sensor, temperature sensor, 3-axis

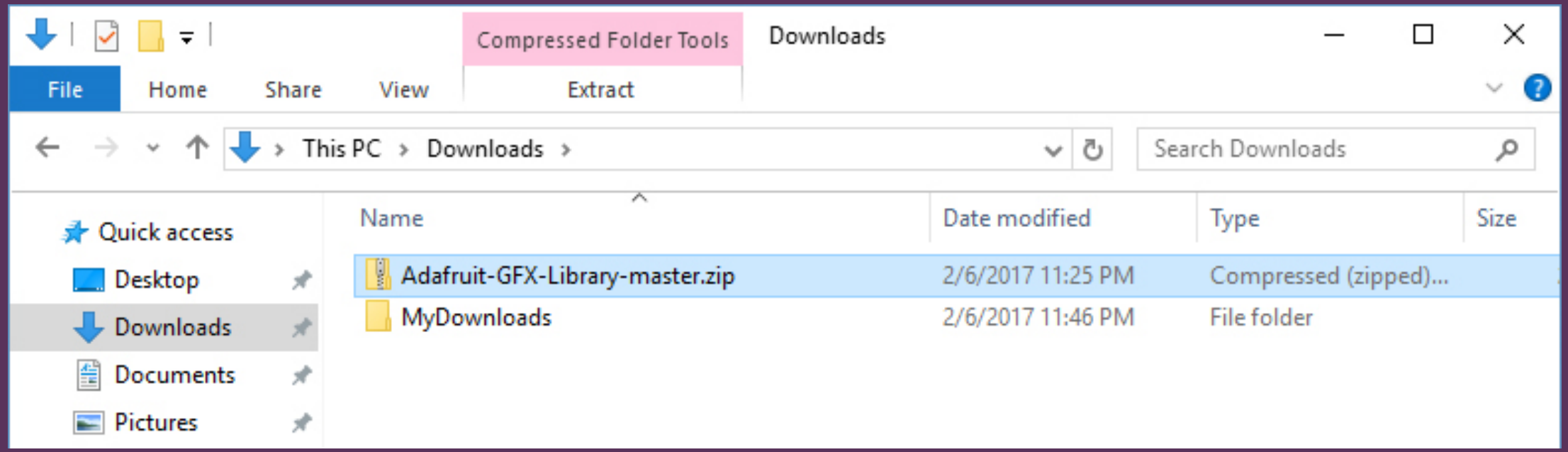
Manual installation

- When you want to add a library manually, you need to download it as a ZIP file, expand it and put in the proper directory.
- The ZIP file contains all you need, including usage examples if the author has provided them.
- The library manager is designed to install this ZIP file automatically as explained in the former topic, but there are cases where you may want to perform the installation process manually and put the library in the *libraries* folder of your sketchbook by yourself.

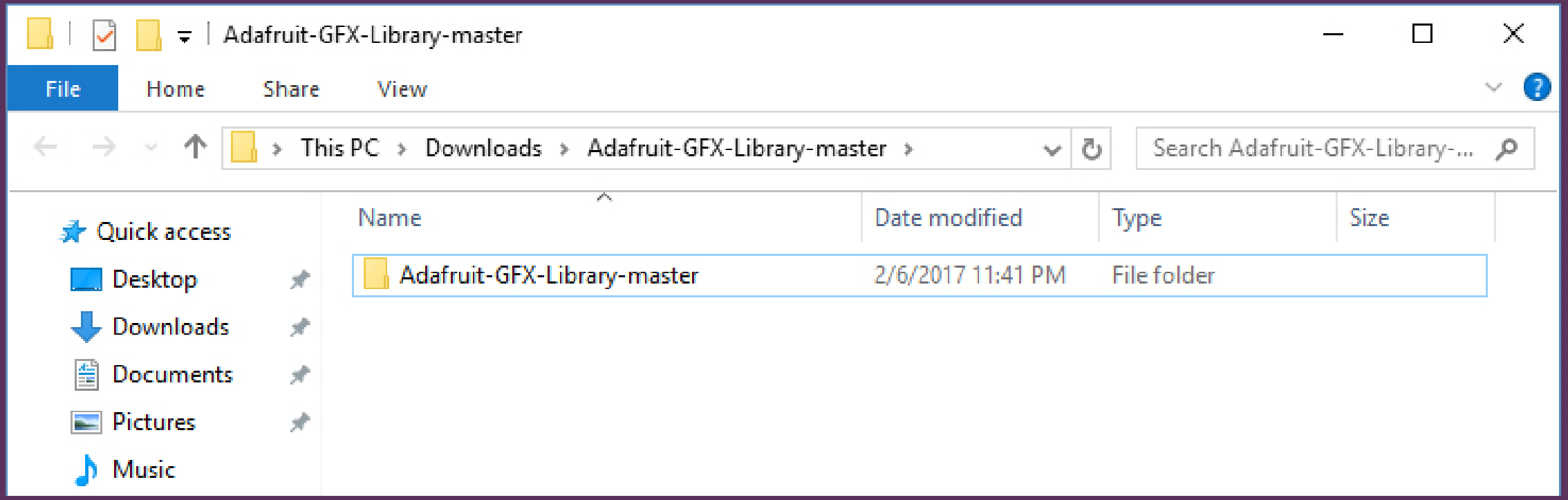
- You can find or change the location of your sketchbook folder at *File* > *Preferences* > *Sketchbook* location.



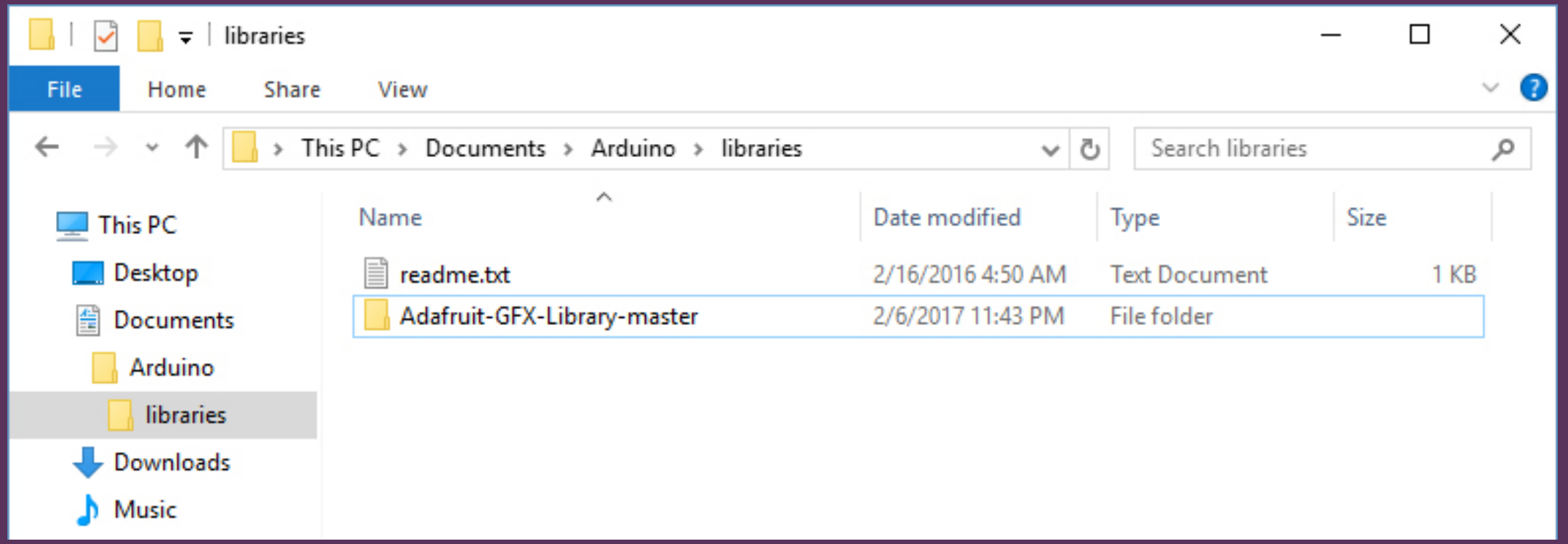
Go to the directory where you have downloaded the ZIP file of the library



Extract the ZIP file with all its folder structure in a temporary folder, then select the main folder, that should have the library name

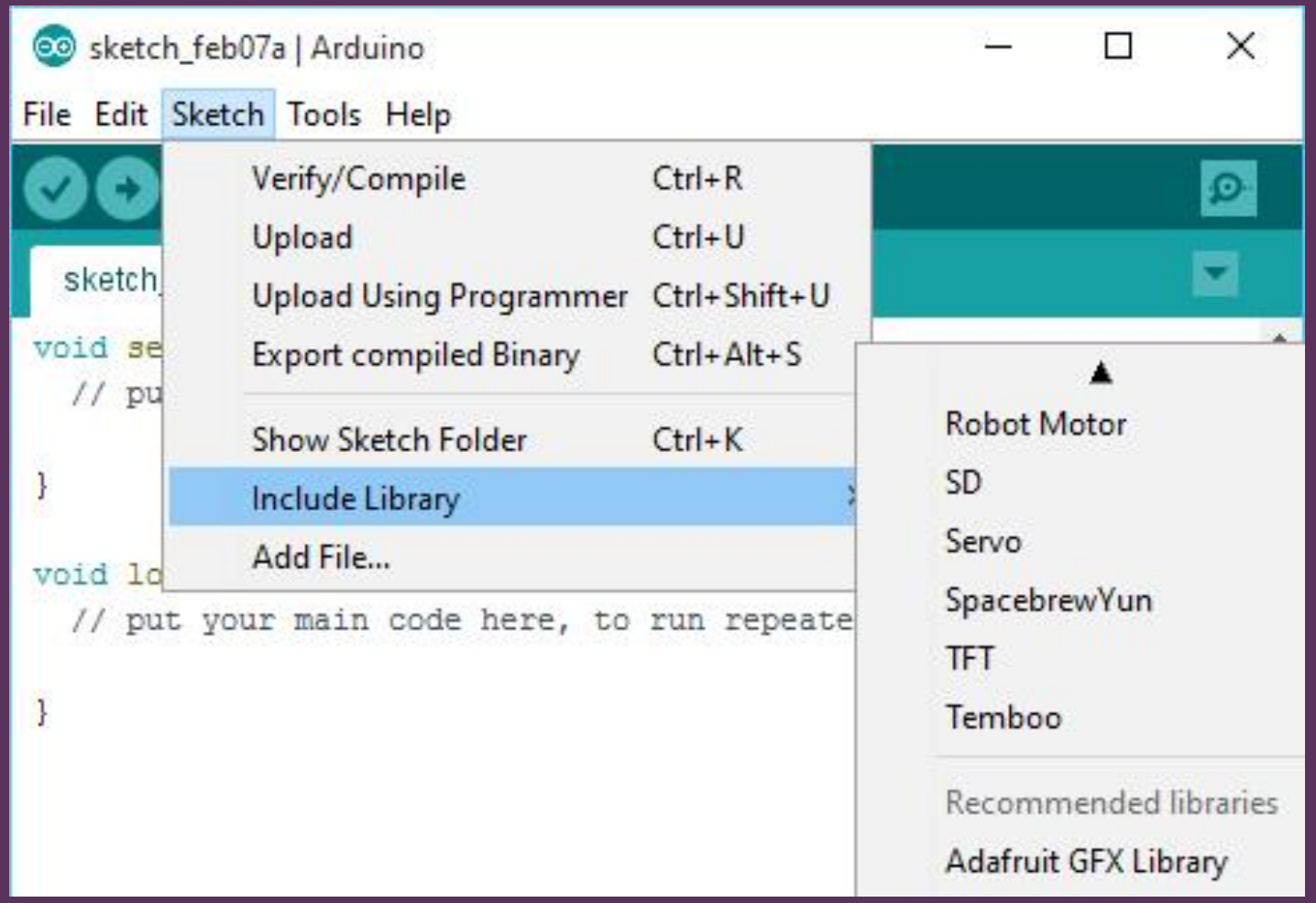


Copy it in the “libraries” folder inside your sketchbook.



Start the Arduino Software (IDE), go to *Sketch > Include Library*.

Verify that the library you just added is available in the list.



Interfacing I/O Devices

- This section explains how to interface many different input and output devices to the microcontroller. I hope you all understood the BASIC code examples of the Arduino board.
- In addition to the BASIC programming concepts let us move to the peripheral interface.

- The interfacing circuits can be spited into four subsections:
  - Introduction to ‘standard’ interfacing circuits
  - Output Device Interfacing
  - Input Device Interfacing
  - Advanced Component Interfacing

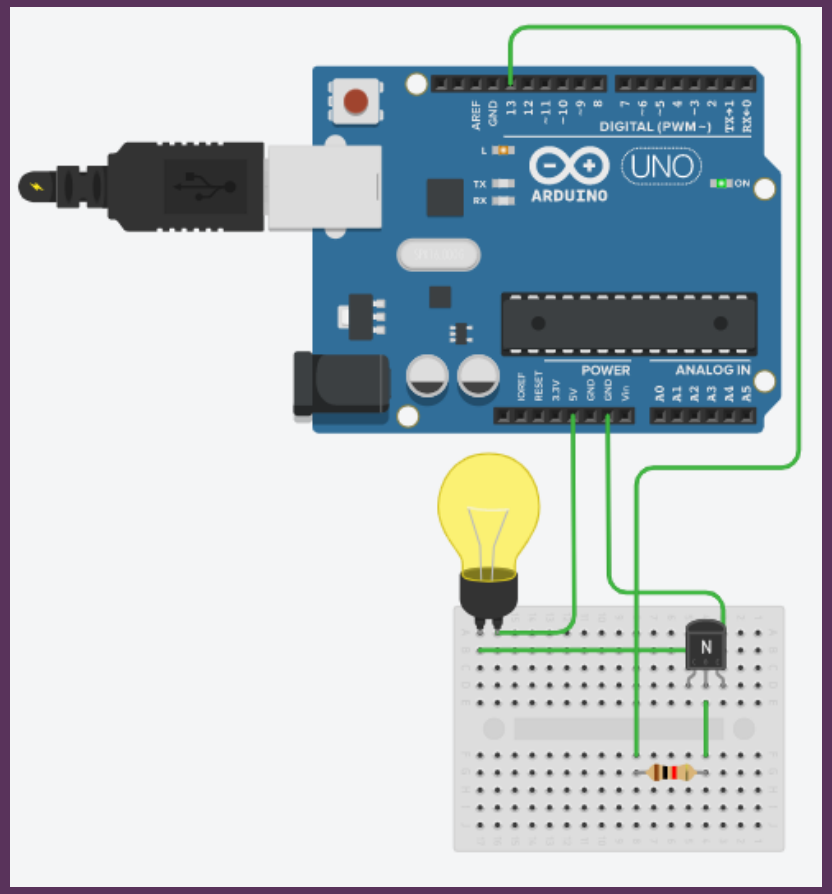
Standard Interfacing Circuits

- *The Standard Transistor Interfacing Circuit*

- Many components such as relays, solenoids, high power LEDs, buzzers and others require more drive current and/or higher voltages than the microcontroller outputs can handle.
- One way around this problem is to use the microcontroller to drive a transistor, which in turn controls the load.
- Many output devices will require a transistor switching circuit and commonly used NPN transistor is called the BC54x and can switch currents up to 800mA.
- This is the transistor used in almost all the circuits in microcontroller.

Aim: To interface the Signal LAMP interface with Microcontroller

- Arduino Uno, Bulb, Transistor NPN, Resistor 1K ohms, Breadboard

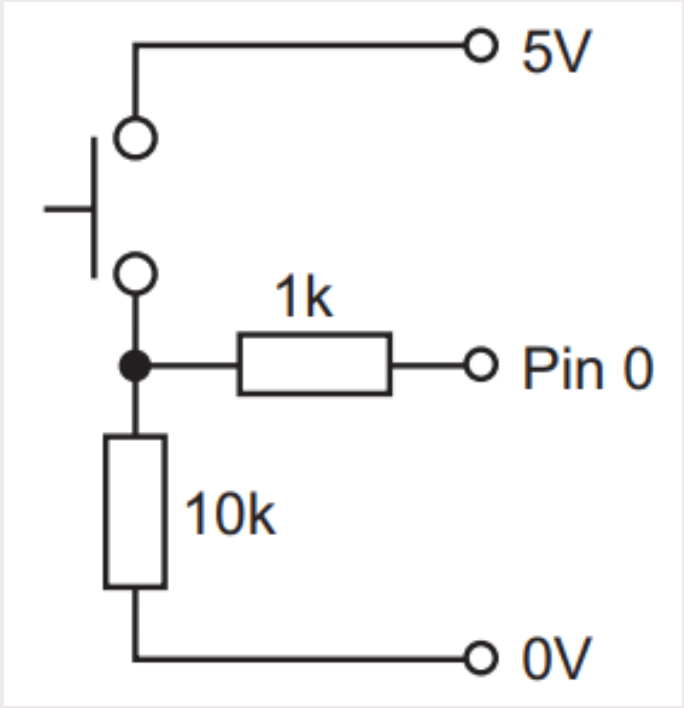


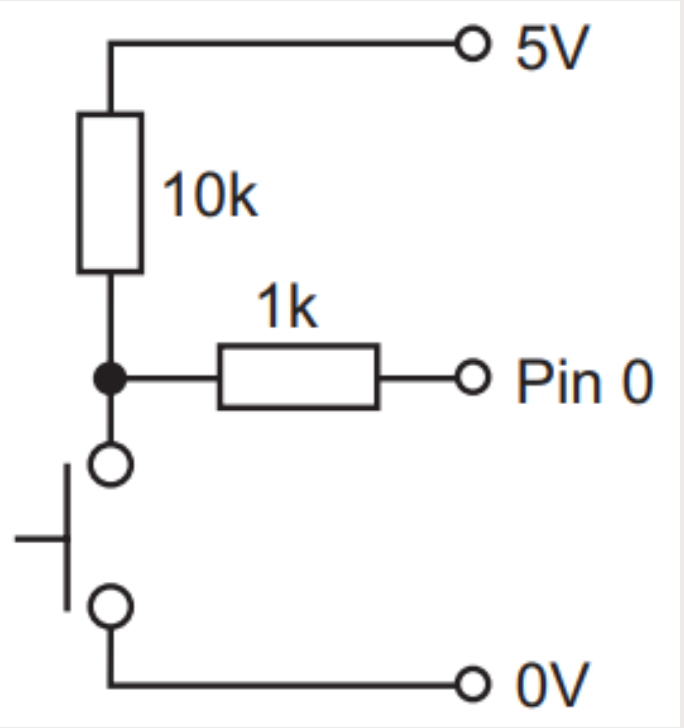

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Arduino interfacing with I/O devices

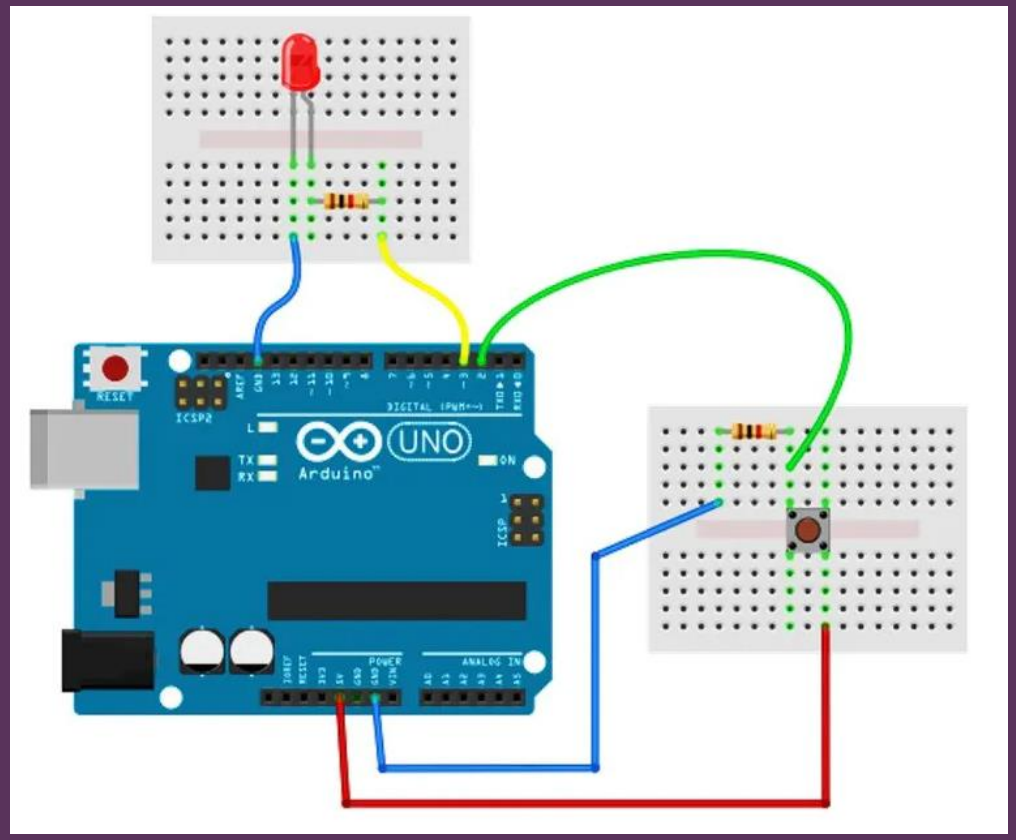
Input Device - Switches

There are a large variety of switches available, but the majority all have two ‘contacts’ which are either ‘open’ (off) or ‘closed’ (on). The two circuits shown below can be used with almost all switches





LED and a Push Button

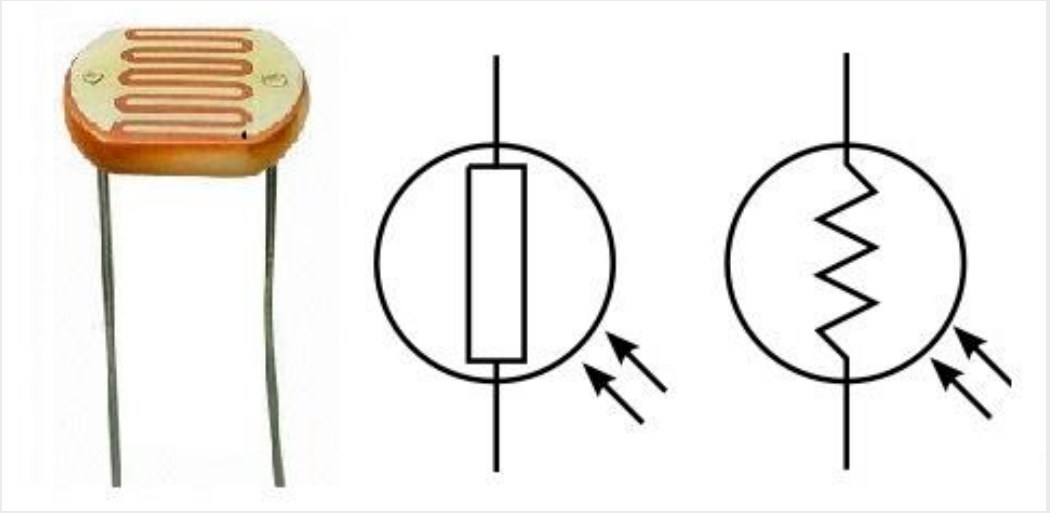


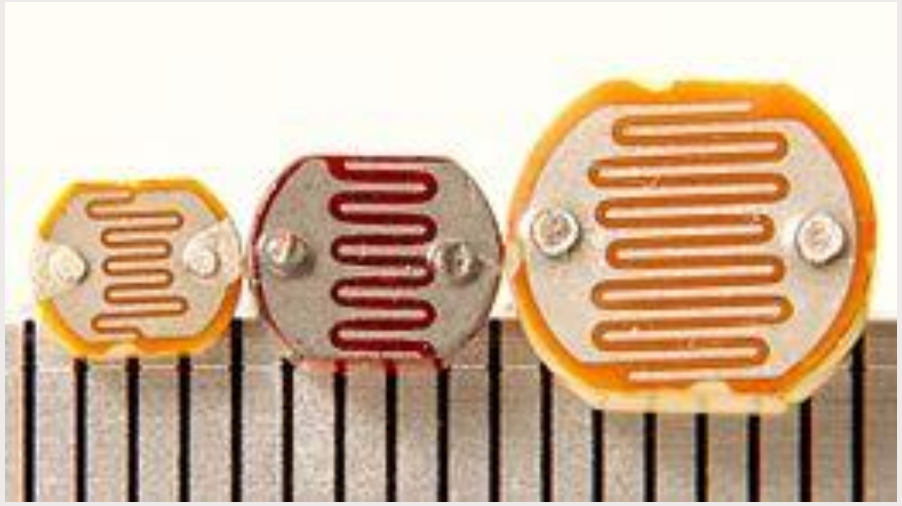

```
const int led = 3;
const int btn = 2;
int btn_state = 0;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(btn, INPUT);
 }
void loop()
{
  btn_state = digitalRead(btn);
  if(btn_state == HIGH){
  digitalWrite(led, HIGH);
  delay(100); // Wait for 1000 millisecond(s)
  } else {
  digitalWrite(led, LOW);
  delay(100); // Wait for 1000 millisecond(s)
  }
}
```

Arduino interfacing with I/O devices

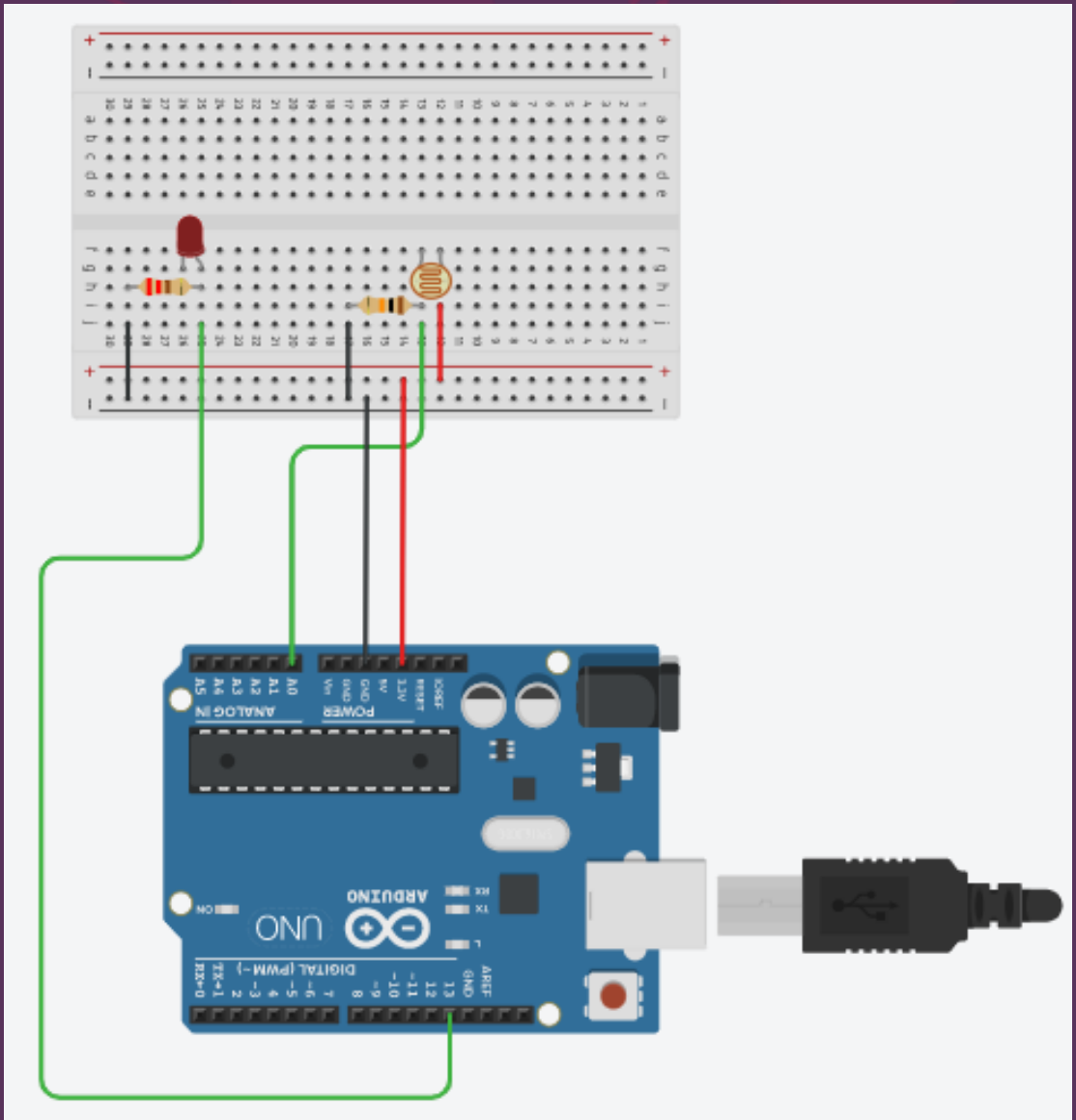
Light Dependant Resistor (LDR)

- A Light Dependant Resistor (LDR) is a resistor that changes in value according to the light falling on it
- Light Dependant Resistor (LDR) are light- controlled variable resistors
- A commonly used device, the ORP-12, has a high resistance in the dark, and a low resistance in the light.





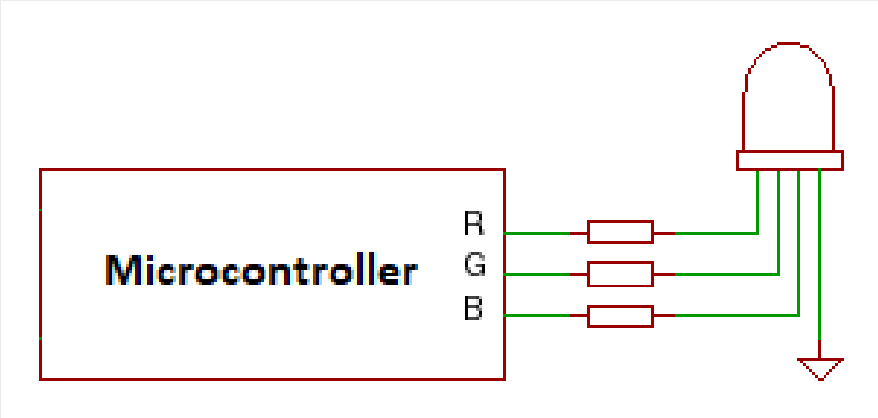
LDR Interfacing

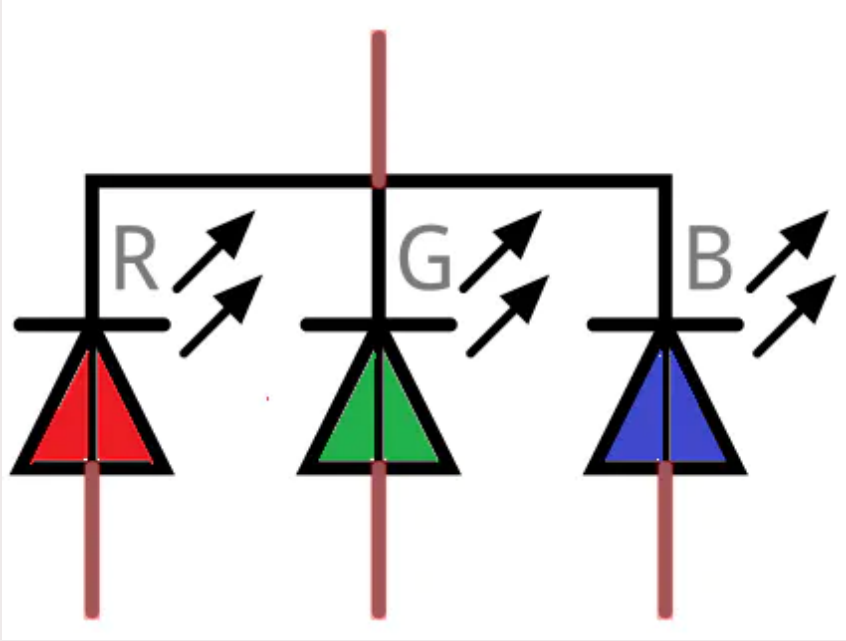


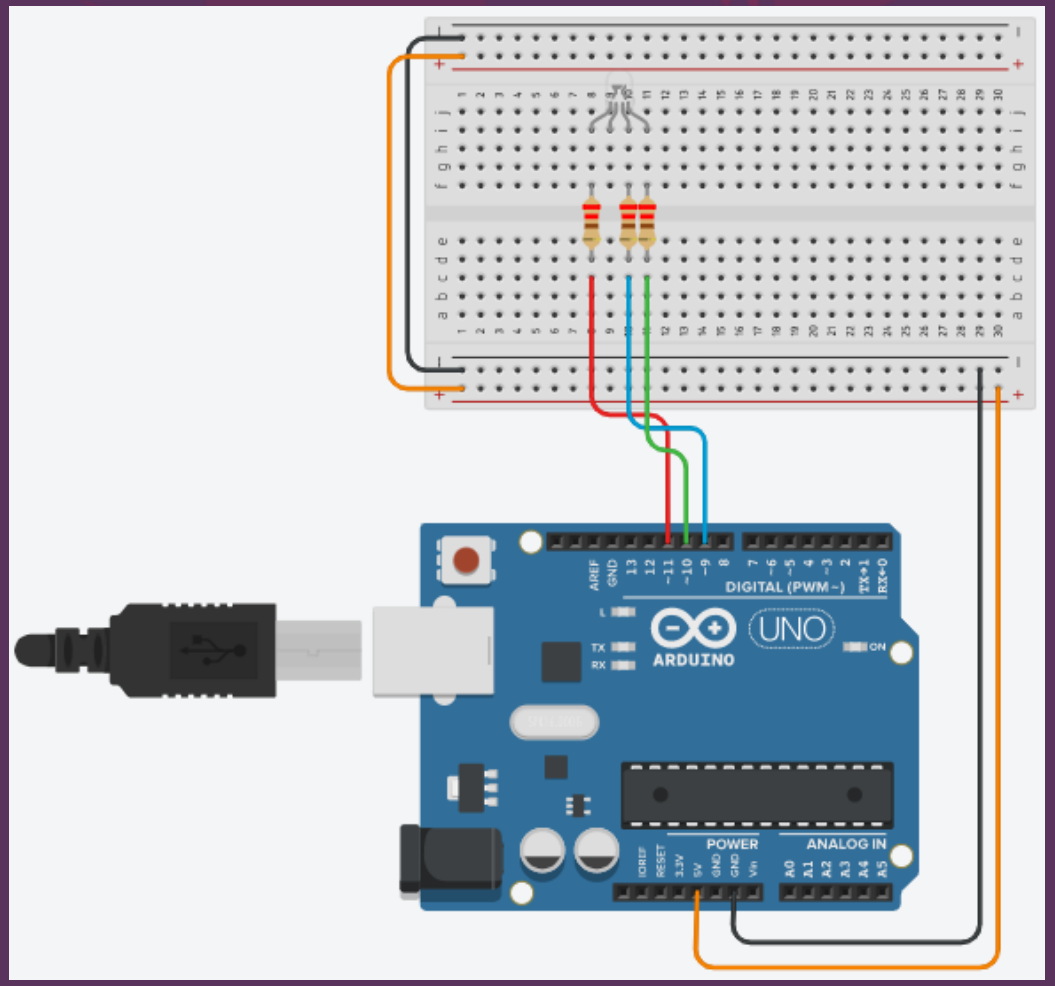
```
const int ledPin = 13;
const int ldrPin = A0;

void setup() {
Serial.begin(9600);
pinMode(ledPin, OUTPUT);
pinMode(ldrPin, INPUT);
}

void loop() {
int ldrStatus = analogRead(ldrPin);
if (ldrStatus <= 200) {
digitalWrite(ledPin, HIGH);
Serial.print("Its DARK, Turn on the LED : ");
Serial.println(ldrStatus);
} else {
digitalWrite(ledPin, LOW);
Serial.print("Its BRIGHT, Turn off the LED : ");
Serial.println(ldrStatus);
    }
}
```

Arduino interfacing with I/O devices

- The RGB LEDs consists of three different LED's, from the name you can guess that these LEDs are red, green and blue
- We can obtain many other colors by mixing up these colors
- The Arduino microcontroller has an Analog pins varying from 0 to 255 which will help us in obtaining different colors for RGB LED








```
int red_pin= 11;
int green_pin = 10;
int blue_pin = 9;
void setup() {
  pinMode(red_pin, OUTPUT);
  pinMode(green_pin, OUTPUT);
  pinMode(blue_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);
 }
void RGB_color(int red_value,
  int green_value, int blue_value)
 {
  analogWrite(red_pin, red_value);
  analogWrite(green_pin, green_value);
  analogWrite(blue_pin, blue_value);
}
```

Color variations

```
int red_pin= 11;
int green_pin = 10;
int blue_pin = 9;
void setup() {
  pinMode(red_pin, OUTPUT);
  pinMode(green_pin, OUTPUT);
  pinMode(blue_pin, OUTPUT);
}
void loop() {
  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_value,
  int green_value, int blue_value)
 {
  analogWrite(red_pin, red_value);
  analogWrite(green_pin, green_value);
  analogWrite(blue_pin, blue_value);
}
```

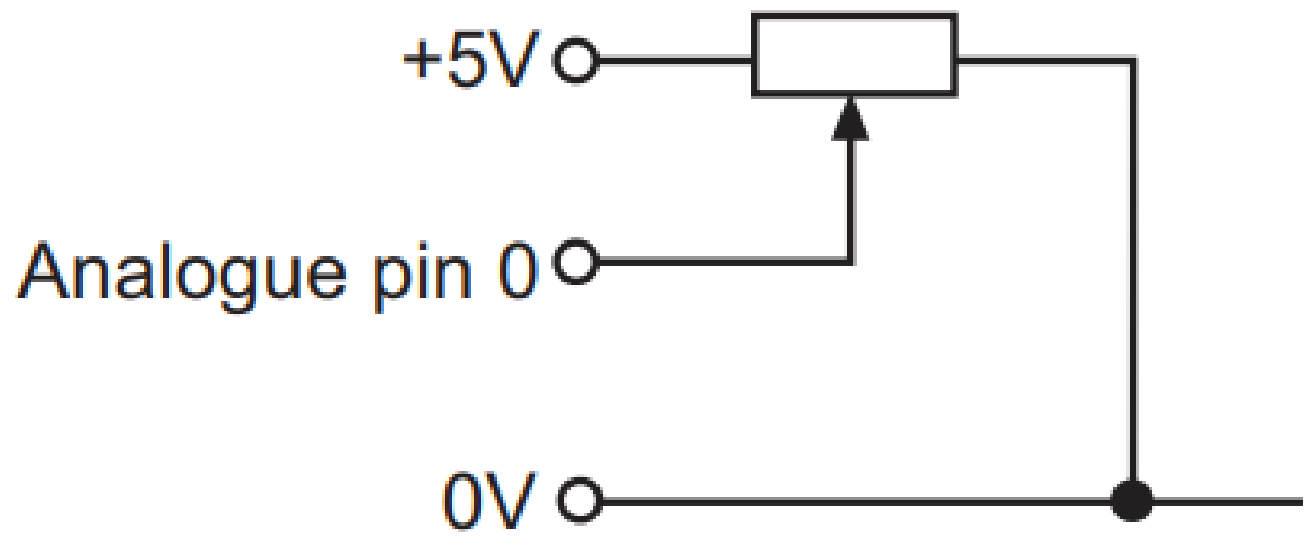
Arduino interfacing with I/O devices

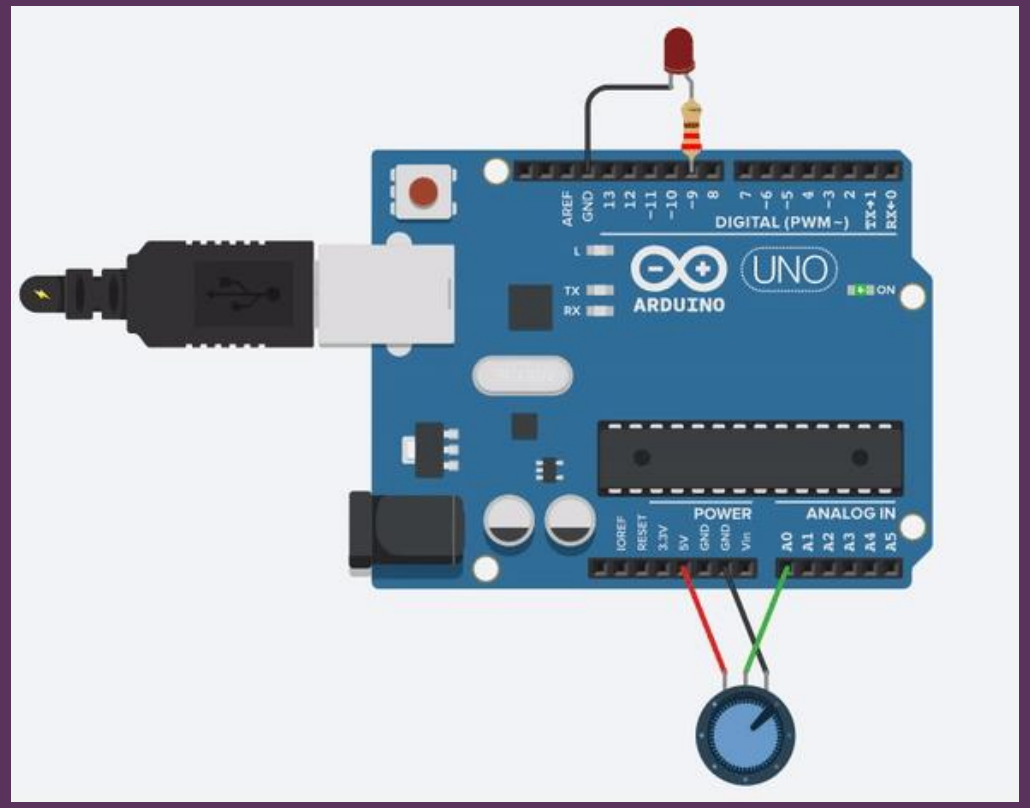
Input Device - Potentiometer

A potentiometer (or 'variable resistor') has a spindle that can be moved to change the resistance value of the potentiometer.

This can be used to measure rotational or linear movement

The Analog pins used to measure the value of the resistance by carrying out an Analog to Digital Conversion. The value of the resistance is given a 'value' between 0 and 1023 which is then stored in a variable.






```
int potPin = A0;
// select the input pin for the potentiometer
int ledPin = 9;
// select the pin for the LED
int val = 0;
// variable to store the value coming from the sensor

void setup() {
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT);
    pinMode(potPin, INPUT);
}
void loop() {
    val = analogRead(potPin)/4;

    // read the value from the sensor
    analogWrite(ledPin, val);

    // stop the program for some time
    Serial.println(val);
}
```

Motors and Arduino

Now we will interface different types of motors with the Arduino board (UNO) and show you how to connect the motor and drive it from your board.

There are three different type of motors –

- DC motor
- Servo motor
- Stepper motor

A **DC motor (Direct Current motor)** is the most common type of motor. DC motors normally have just two leads, one positive and one negative. If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.

Warning – Do not drive the motor directly from Arduino board pins. This may damage the board. Use a driver Circuit or an IC.

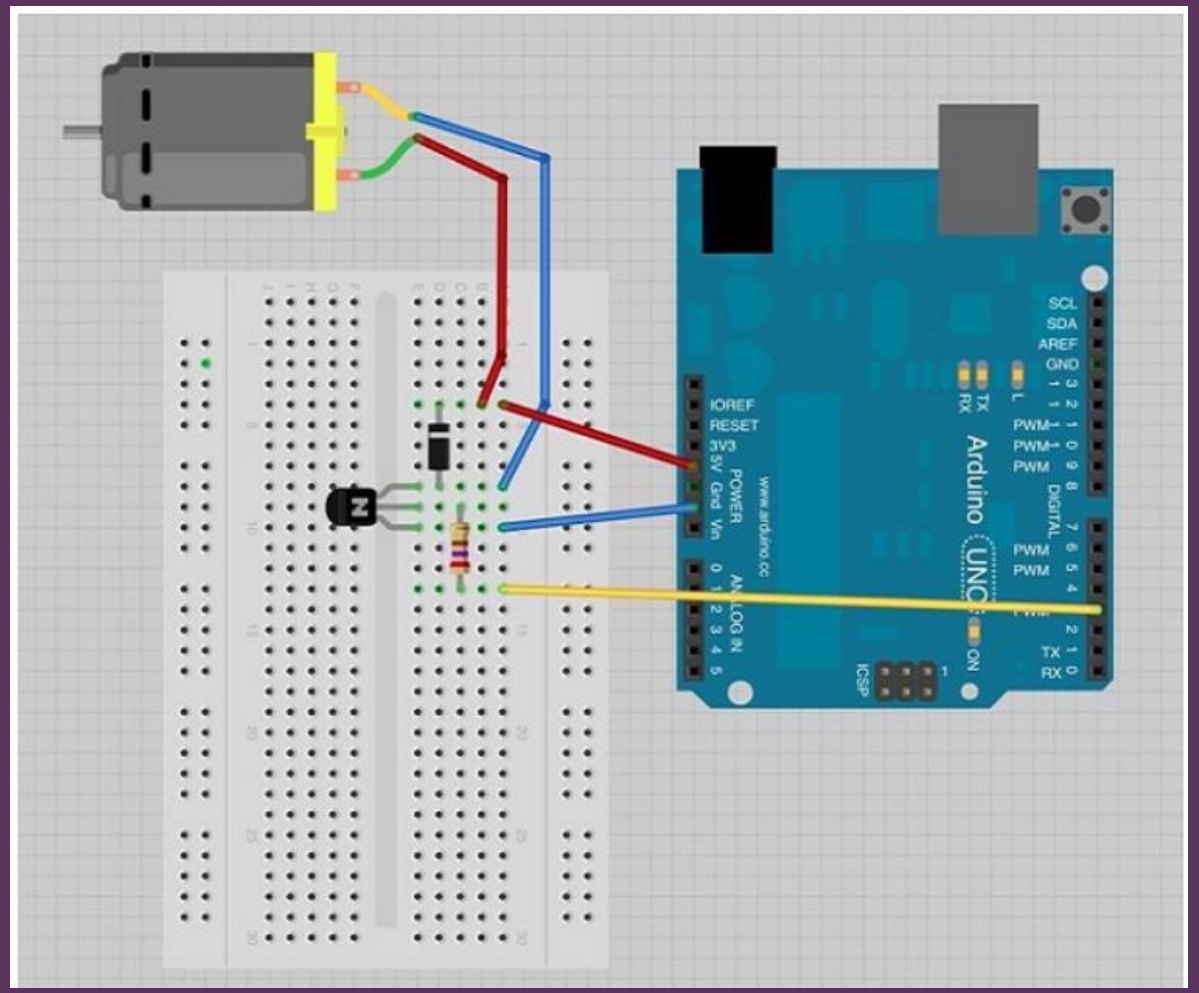
We will divide this chapter into three parts –

- Just make your motor spin
- Control motor speed

You will need the following components –

- 1x Arduino UNO board
- 1x PN2222 Transistor
- 1x Small 6V DC Motor
- 1x 1N4001 diode
- 1x 270 Ω Resistor

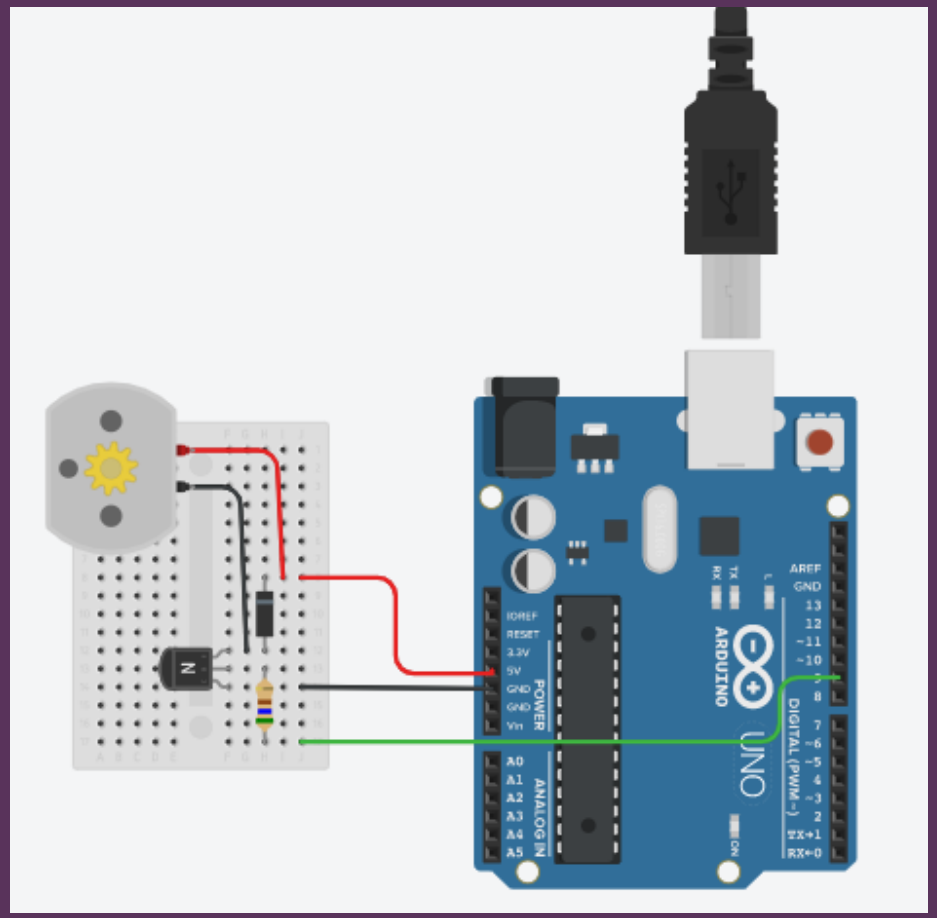
Spin the motor



Spin ControlArduino Code

```
int motorPin = 3;

void setup() {
PinMode(motorPin, OUTPUT);
}

void loop() {
   digitalWrite(motorPin, HIGH);
}
```

Control the speed of motor

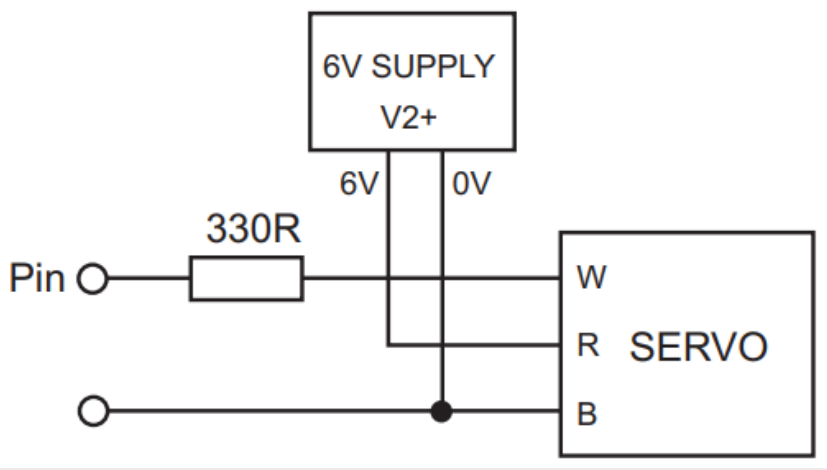


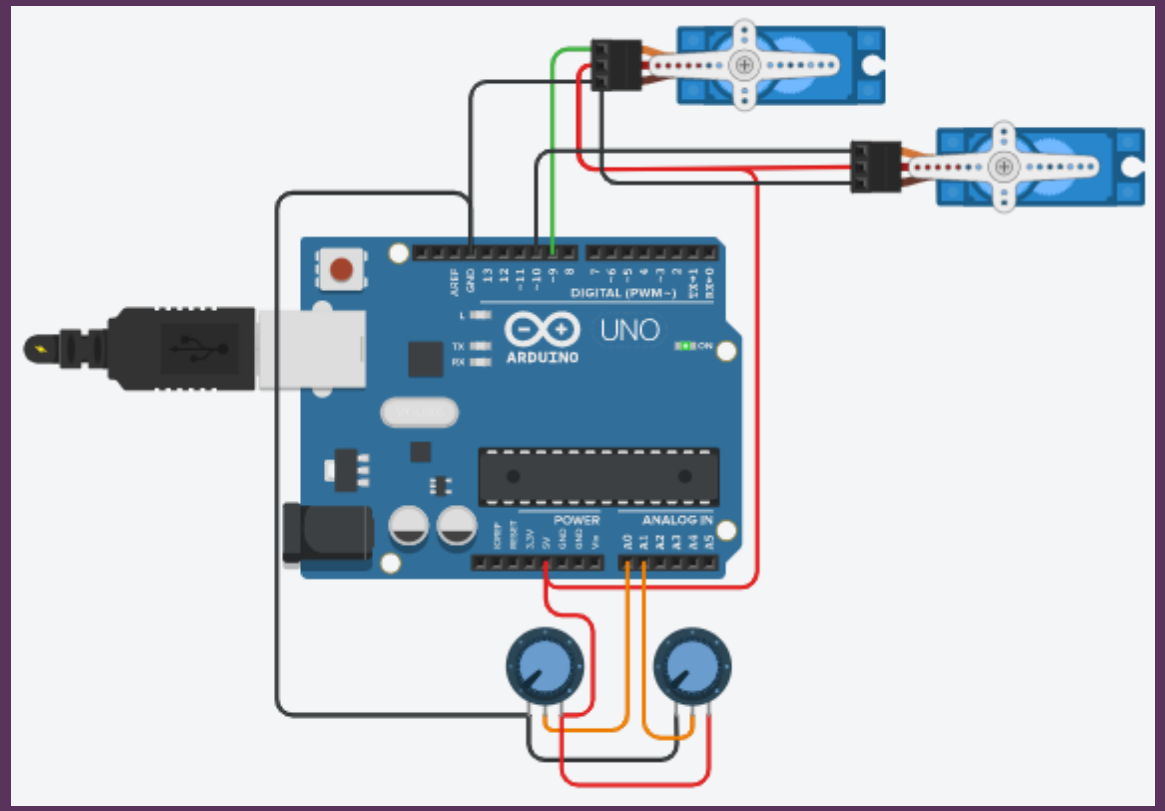

```
int motorPin = 9;
void setup(){
 pinMode(motorPin, OUTPUT);
}
void loop() {
 for(int mValue = 0 ; mValue <= 255; mValue +=5){
 analogWrite(motorPin, mValue);
 delay(30);
 }
 for(int mValue = 255 ; mValue >= 0; mValue -=5){
 analogWrite(motorPin, mValue);
 delay(30);
 }
}
```

Arduino interfacing with motors

Control Servo motor

- Servos are used in most industrial robots and planes to control the mechanism
- They are accurate devices that always rotate the same amount for a given signal, and so are ideal for use in many automated machines
- A typical servo has just three connection wires, normally red, black and white (or yellow)
- The red wire is the 5V supply, the black wire is the 0V supply, and the white (or yellow) wire is for the positioning signal






```
#include <Servo.h>

Servo servo_9;

Servo servo_10;

void setup()
{
  pinMode(A0, INPUT);
  servo_9.attach(9, 500, 2500);

  pinMode(A1, INPUT);
  servo_10.attach(10, 500, 2500);

}

void loop()
{
  servo_9.write(map(analogRead(A0), 0, 1023, 0, 180));
  servo_10.write(map(analogRead(A1), 0, 1023, 0, 180));
  delay(10);
  // Delay a little bit to improve simulation performance
}
```

- A Stepper Motor or a step motor is a brushless, synchronous motor, which divides a full rotation into a number of steps. Unlike a brushless DC motor, which rotates continuously when a fixed DC voltage is applied to it, a step motor rotates in discrete step angles.
- The Stepper Motors therefore are manufactured with steps per revolution of 12, 24, 72, 144, 180, and 200, resulting in stepping angles of 30, 15, 5, 2.5, 2, and 1.8 degrees per step. The stepper motor can be controlled with or without feedback.
- Imagine a motor on an RC airplane. The motor spins very fast in one direction or another. You can vary the speed with the amount of power given to the motor, but you cannot tell the propeller to stop at a specific position.
- Now imagine a printer. There are lots of moving parts inside a printer, including motors. One such motor acts as the paper feed, spinning rollers that move the piece of paper as ink is being printed on it. This motor needs to be able to move the paper an exact distance to be able to print the next line of text or the next line of an image.

Plan for Day 2 :

- L293D Motor Driver
- L298N Motor Driver
- Ultrasonic sensor
- IR sesnor
- PIR sensor