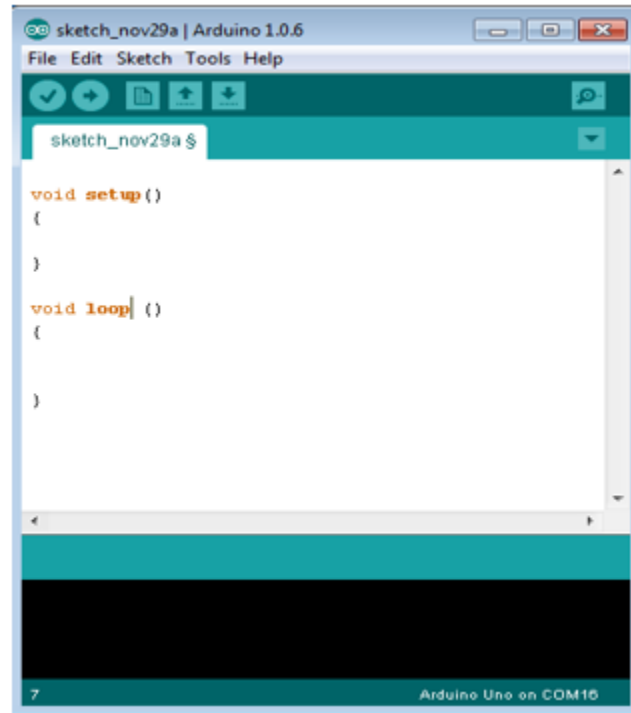
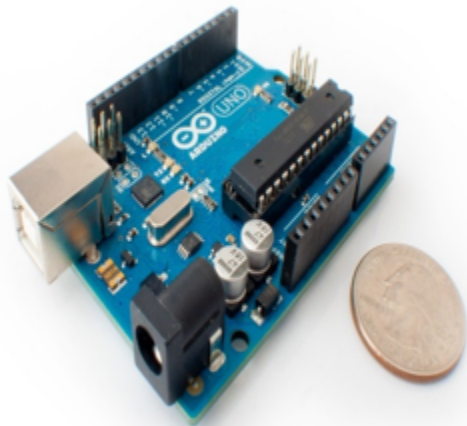


Arduino - Overview

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.



Board Types

Various kinds of Arduino boards are available depending on different microcontrollers used. However, all Arduino boards have one thing in common: they are programmed through the Arduino IDE.

The differences are based on the number of inputs and outputs (the number of sensors, LEDs, and buttons you can use on a single board), speed, operating voltage, form factor etc. Some boards are designed to be embedded and have no programming interface (hardware), which you would need to buy separately. Some can run directly from a 3.7V battery, others need at least 5V.

Here is a list of different Arduino boards available.

Arduino boards based on ATMEGA328 microcontroller

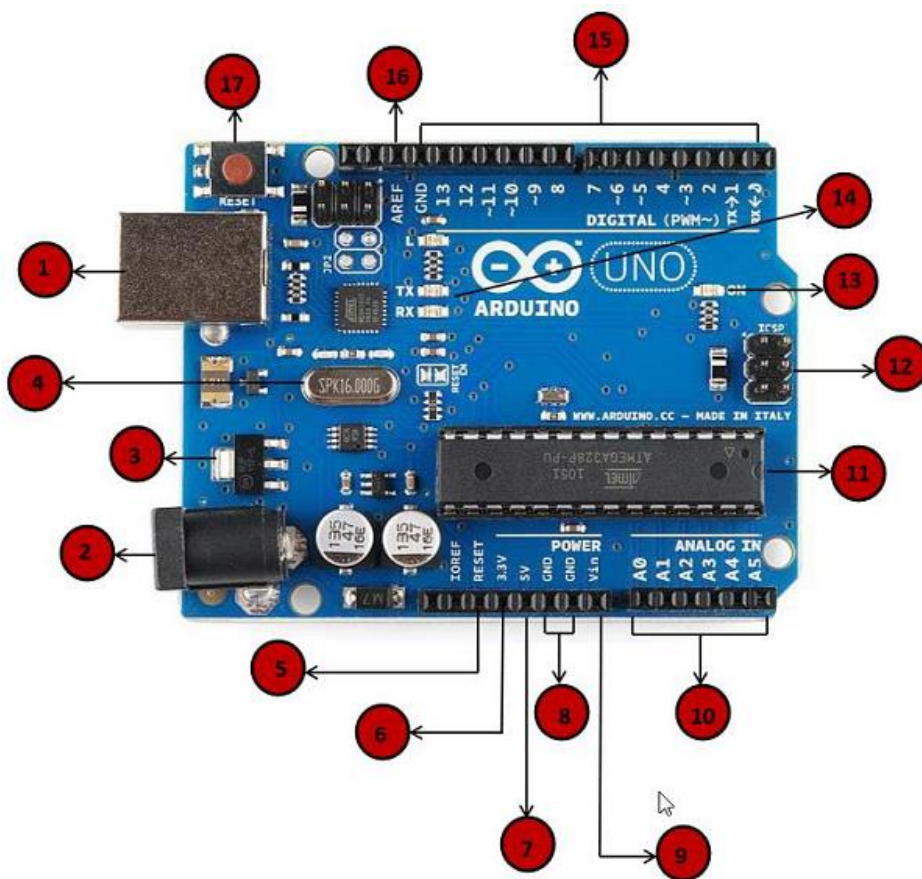
Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Uno R3	5V	16MHz	14	6	6	1	USB via ATmega16U2
Arduino Uno R3 SMD	5V	16MHz	14	6	6	1	USB via ATmega16U2
Red Board	5V	16MHz	14	6	6	1	USB via FTDI
Arduino Pro 3.3v/8 MHz	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
Arduino Pro 5V/16MHz	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino mini 05	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro mini 3.3v/8mhz	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro mini 5v/16mhz	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Ethernet	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino Fio	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
LilyPad Arduino 328 main board	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
LilyPad Arduino simple board	3.3V	8MHz	9	4	5	0	FTDI-Compatible Header










Arduino boards based on ATMEGA32u4 microcontroller





Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Leonardo	5V	16MHz	20	12	7	1	Native USB
Pro micro 5V/16MHz	5V	16MHz	14	6	6	1	Native USB
Pro micro 3.3V/8MHz	5V	16MHz	14	6	6	1	Native USB
LilyPad Arduino USB	3.3V	8MHz	14	6	6	1	Native USB

Arduino - Board Description

In this chapter, we will learn about the different components on the Arduino board. We will study the Arduino UNO board because it is the most popular board in the Arduino board family. In addition, it is the best board to get started with electronics and coding. Some boards look a bit different from the one given below, but most Arduinos have majority of these components in common.



	<p>Power USB</p> <p>Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).</p>
	<p>Power (Barrel Jack)</p> <p>Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).</p>
	<p>Voltage Regulator</p> <p>The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.</p>
	<p>Crystal Oscillator</p> <p>The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.</p>
	<p>Arduino Reset</p> <p>You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).</p>
	<p>Pins (3.3, 5, GND, Vin)</p> <ul style="list-style-type: none"> • 3.3V (6) – Supply 3.3 output volt • 5V (7) – Supply 5 output volt • Most of the components used with Arduino board works fine with 3.3 volt and 5 volt. • GND (8)(Ground) – There are several GND pins on the Arduino, any of which can be used to ground your circuit. • Vin (9) – This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.
	<p>Analog pins</p> <p>The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.</p>
	<p>Main microcontroller</p> <p>Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.</p>
	<p>ICSP pin</p> <p>Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.</p>

	<p>Power LED indicator</p> <p>This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.</p>
	<p>TX and RX LEDs</p> <p>On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.</p>
	<p>Digital I/O</p> <p>The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.</p>
	<p>AREF</p> <p>AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.</p>

Arduino - Installation

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, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in figure 1

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the figure 2



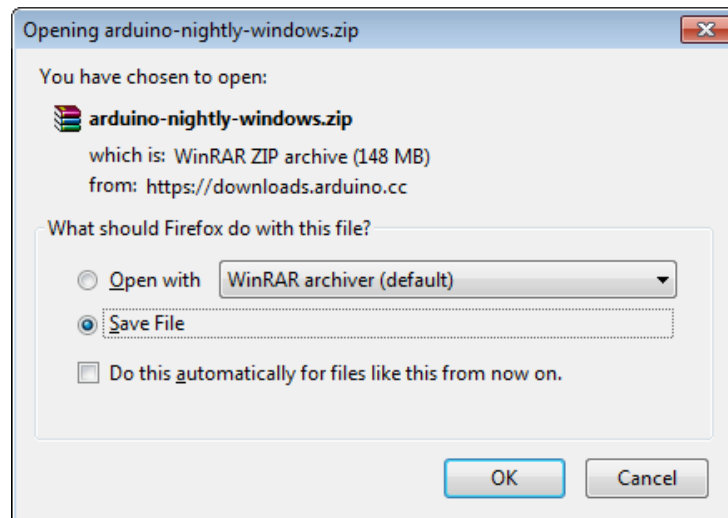
Figure 1



Figure 2

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.



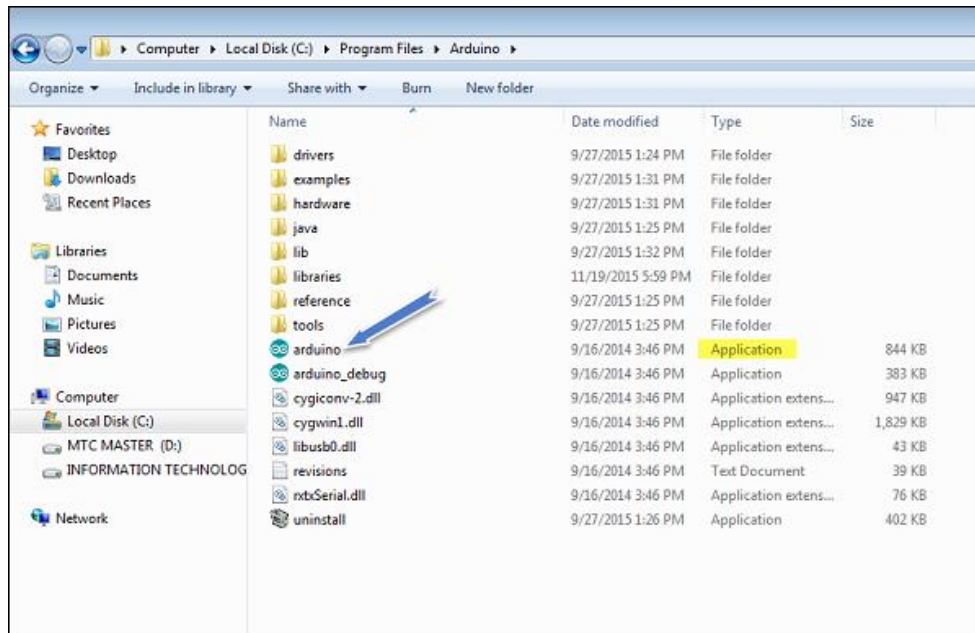
Step 3 – Power up your board.

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.

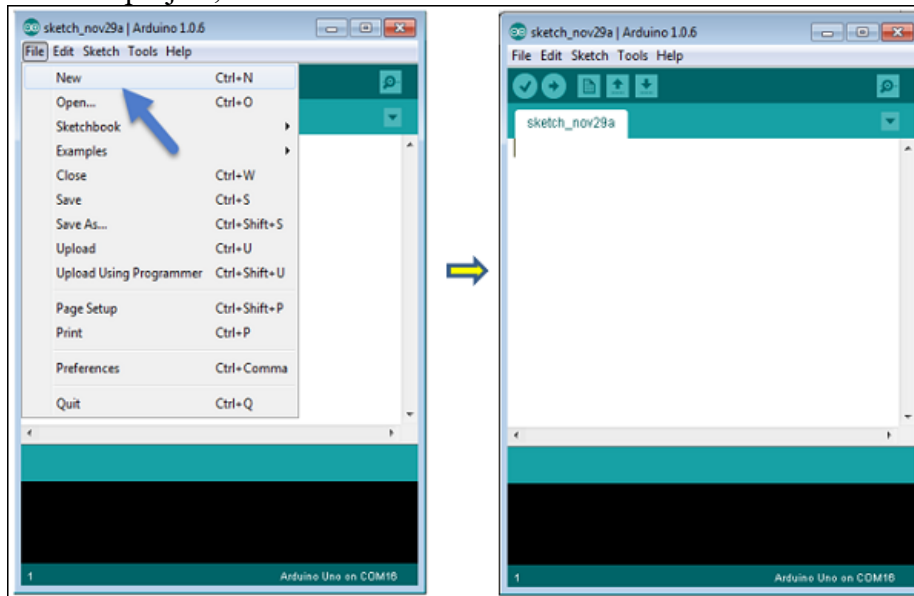


Step 5 – Open your first project.

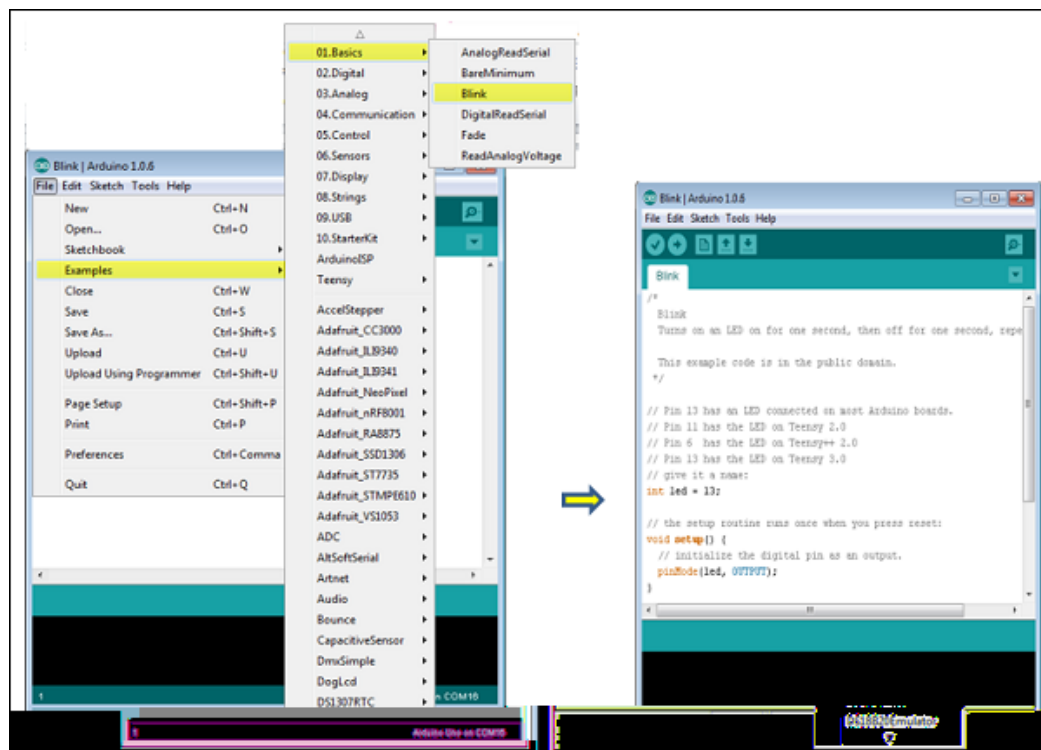
Once the software starts, you have two options –

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

To create a new project, select File → New.



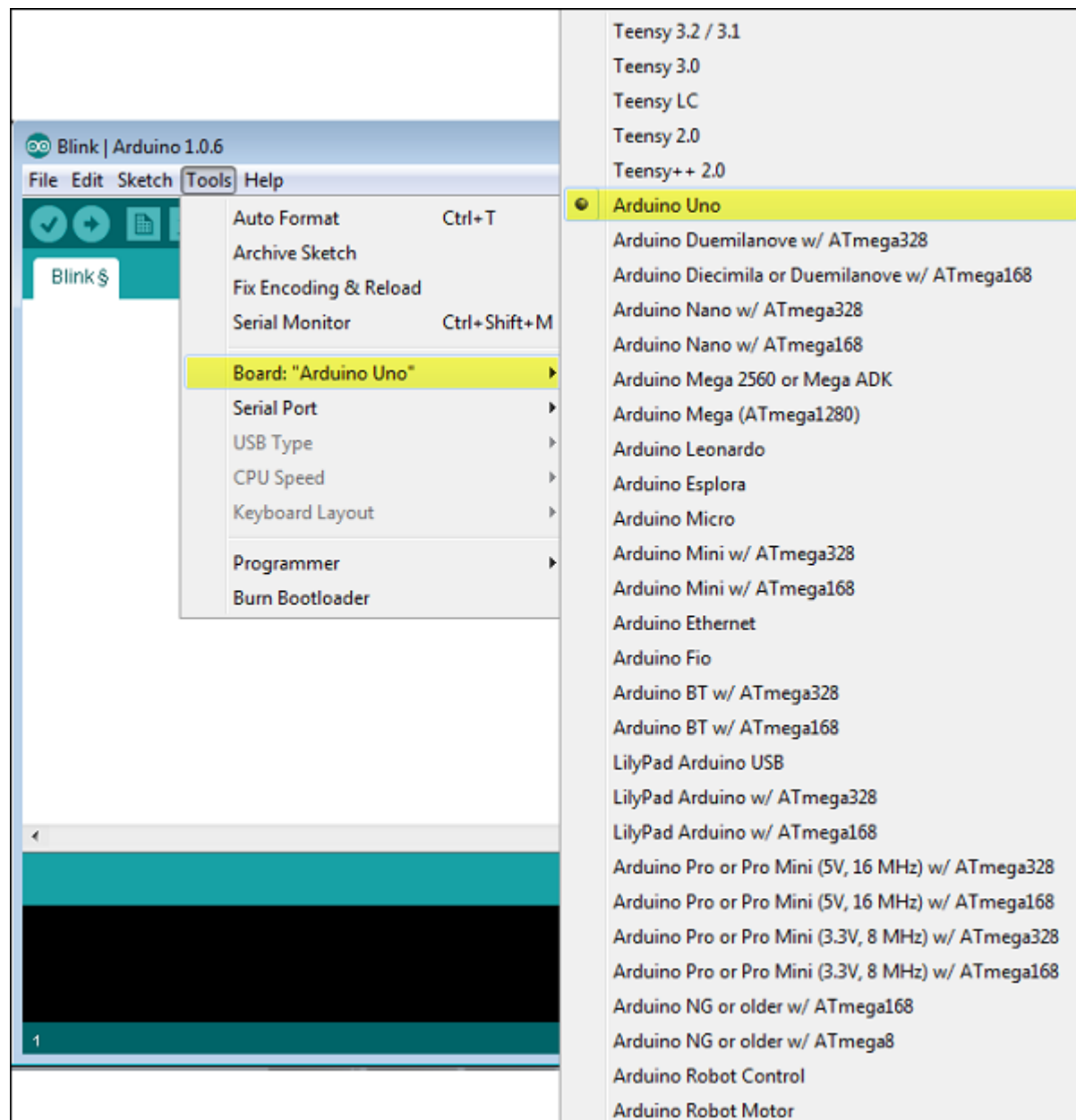
To open an existing project example, select File → Example → Basics → Blink.



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.

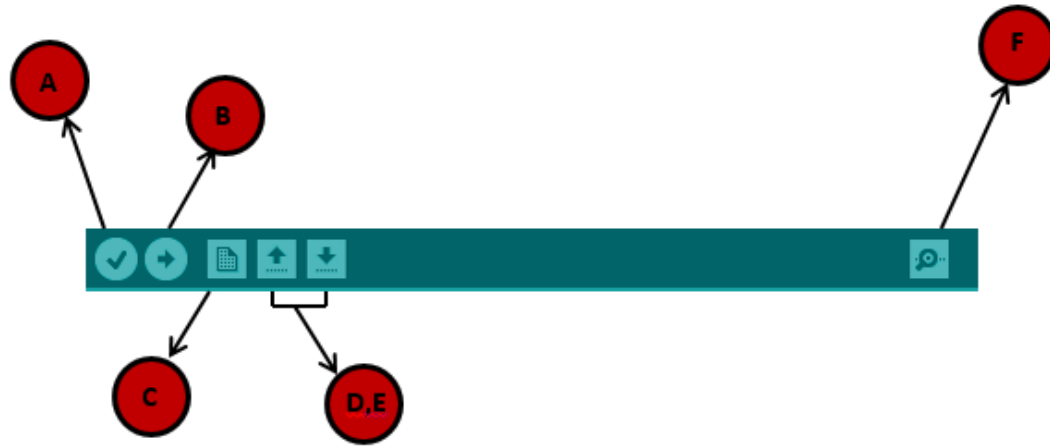
Go to Tools → Board and select your board.



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 8 – Upload the program to your board.

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 sketch.

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.

Arduino - Program Structure

The Arduino software is open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

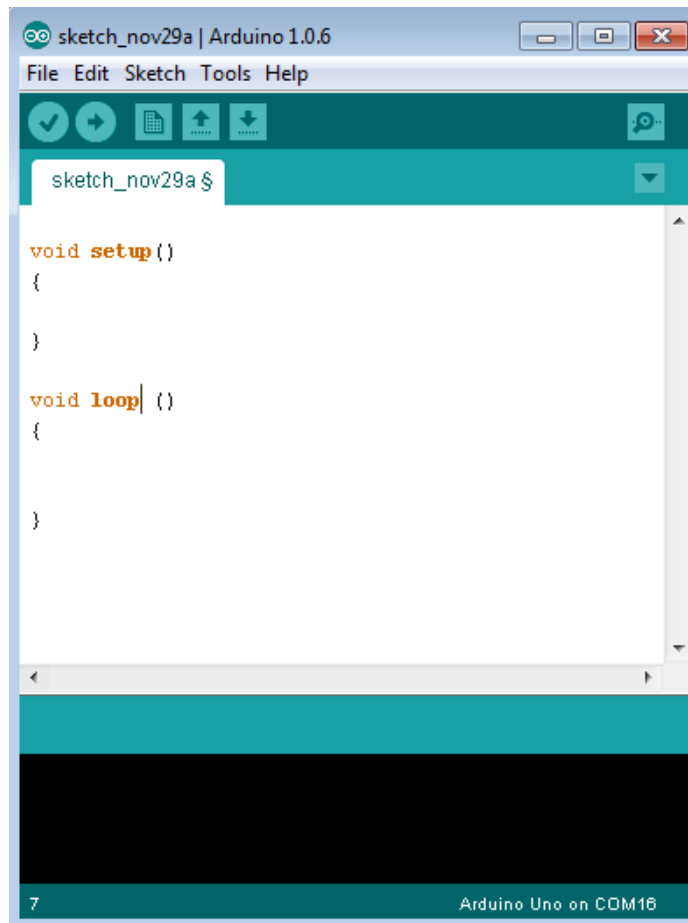
Sketch – The first new terminology is the Arduino program called “**sketch**”.

Structure

Arduino programs can be divided in three main parts: **Structure**, **Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions –

- Setup() function
- Loop() function



Void setup () {

}

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

INPUT – -

OUTPUT – -

RETURN – -

Void Loop () {

}

PURPOSE – 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.

INPUT – -

OUTPUT – -

RETURN – -

Arduino - Data Types

void	Boolean	char	Unsigned char	byte	int	Unsigned int	word
long	Unsigned long	short	float	double	array	String-char array	String-object

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

Boolean

A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte of memory.

`boolean val = false ; // declaration of variable with type boolean and initialize it with false`

`boolean state = true ; // declaration of variable with type boolean and initialize it with true`

Char

A data type that takes up one byte of memory that stores a character value. Character literals are written in single quotes like this: 'A' and for multiple characters, strings use double quotes: "ABC".

However, characters are stored as numbers. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

`Char chr_a = 'a' ; // declaration of variable with type char and initialize it with character a`

`Char chr_c = 97 ; // declaration of variable with type char and initialize it with character 97`

unsigned char

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

`Unsigned Char chr_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y`

byte

A byte stores an 8-bit unsigned number, from 0 to 255.

`byte m = 25 ;//declaration of variable with type byte and initialize it with 25`

int

Integers are the primary data-type for number storage. int stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^{15} and a maximum value of $(2^{15}) - 1$).

The int size varies from board to board. On the Arduino Due, for example, an int stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^{31} and a maximum value of $(2^{31}) - 1$).

`int counter = 32 ;// declaration of variable with type int and initialize it with 32`

Unsigned int

Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 ($2^{16}) - 1$). The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 ($2^{32} - 1$).

`Unsigned int counter = 60 ; // declaration of variable with
type unsigned int and initialize it with 60`

Word

On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

`word w = 1000 ;//declaration of variable with type word and initialize it with 1000`

Arduino - Variables & Constants

What is Variable Scope?

Variables in C programming language, which Arduino uses, have a property called scope. A scope is a region of the program and there are three places where variables can be declared. They are –

- Inside a function or a block, which is called **local variables**.
- In the definition of function parameters, which is called **formal parameters**.
- Outside of all functions, which is called **global variables**.

Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by the statements that are inside that function or block of code. Local variables are not known to function outside their own. Following is the example using local variables –

```
Void setup () {
```

```
}
```

```
Void loop () {
```

```
  int x , y ;
```

```
  int z ; Local variable declaration
```

```
  x = 0;
```

```
  y = 0; actual initialization
```

```
  z = 10;
```

```
}
```

Global Variables

Global variables are defined outside of all the functions, usually at the top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

The following example uses global and local variables –

```
Int T , S ;
```

```
float c = 0 ; Global variable declaration
```

```
Void setup () {
```

```
}
```

```
Void loop () {
```

```
  int x , y ;
```

```
  int z ; Local variable declaration
```

```
  x = 0;
```

```
  y = 0; actual initialization
```

```
  z = 10;
```

```
}
```

Arduino - Operators

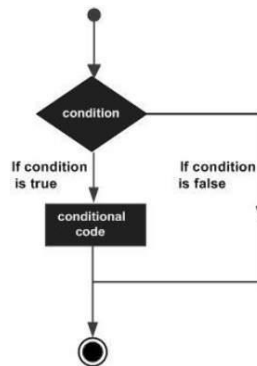
An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Comparison Operators
- Boolean Operators
- Bitwise Operators
- Compound Operators
- For more details use link https://www.tutorialspoint.com/arduino/arduino_quick_guide.htm

Arduino - Control Statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program. It should be along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages



Control Statements are elements in Source Code that control the flow of program execution. They are -

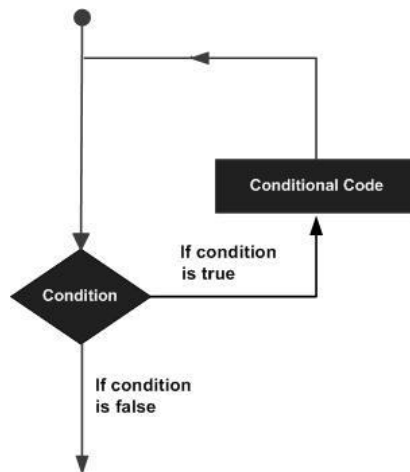
S.NO.	Control Statement & Description
1	<u>If statement</u> It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.
2	<u>If ...else statement</u> An if statement can be followed by an optional else statement, which executes when the expression is false.
3	<u>If...else if ...else statement</u> The if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.
4	<u>switch case statement</u> Similar to the if statements, switch...case controls the flow of programs by allowing the programmers to specify different codes that should be executed in various conditions.
5	<u>Conditional Operator ? :</u> The conditional operator ? : is the only ternary operator in C

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 –

.NO.	Loop & Description
1	<u>while loop</u> while loops 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.
2	<u>do...while loop</u> The do...while loop is similar to the while loop. In the while loop, the loop-continuation condition is tested at the beginning of the loop before performed the body of the loop.
3	<u>for loop</u> A for loop executes statements a predetermined number of times. The control expression for the loop is initialized, tested and manipulated entirely within the for loop parentheses.
4	<u>Nested Loop</u> C language allows you to use one loop inside another loop. The following example illustrates the concept.
5	<u>Infinite loop</u> It is the loop having no terminating condition, so the loop becomes infinite.



Arduino - Functions

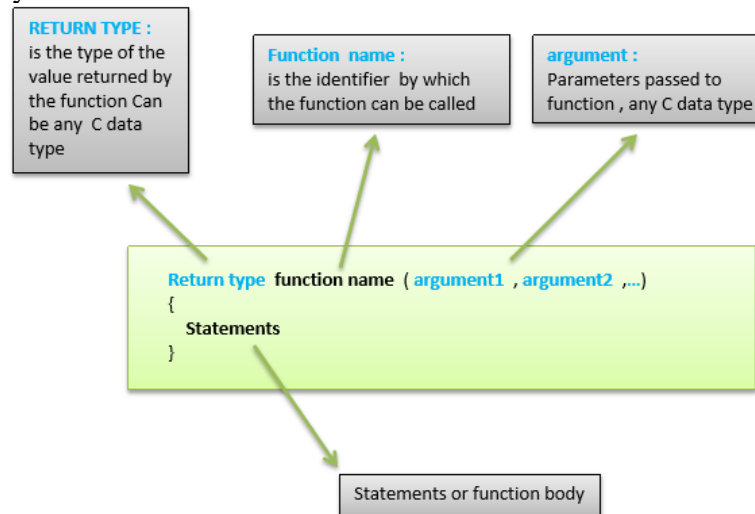
Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages –

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought about and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.

There are two required functions in an Arduino sketch or a program i.e. setup () and loop(). Other functions must be created outside the brackets of these two functions.

The most common syntax to define a function is



Function Declaration

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways –

The first way is just writing the part of the function called **a function prototype** above the loop function, which consists of –

- Function return type
- Function name
- Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon (;).

The following example shows the demonstration of the function declaration using the first method.

```
int sum_func (int x, int y) // function declaration {
    int z = 0;
    z = x+y ;
    return z; // return the value
}
```

```
void setup () {
    Statements // group of statements
}
```

```
Void loop () {
    int result = 0 ;
    result = Sum_func (5,6) ; // function call
}
```

The second part, which is called the function definition or declaration, must be declared below the loop function, which consists of –

- Function return type
- Function name
- Function argument type, here you must add the argument name
- The function body (statements inside the function executing when the function is called)

The following example demonstrates the declaration of function using the second method.

```
int sum_func (int , int ) ; // function prototype
```

```
void setup () {
    Statements // group of statements
}
```

```
Void loop () {
    int result = 0 ;
    result = Sum_func (5,6) ; // function call
}
```

```
int sum_func (int x, int y) // function declaration {
    int z = 0;
    z = x+y ;
    return z; // return the value
}
```

The second method just declares the function above the loop function.

Arduino - Strings

Strings are used to store text. They can be used to display text on an LCD or in the Arduino IDE Serial Monitor window. Strings are also useful for storing the user input. For example, the characters that a user types on a keypad connected to the Arduino.

There are two types of strings in Arduino programming –

- Arrays of characters, which are the same as the strings used in C programming.
- The Arduino String, which lets us use a string object in a sketch.

String Character Arrays

The first type of string that we will learn is the string that is a series of characters of the type `char`. An array is; a consecutive series of the same type of variable stored in memory. A string is an array of **char** variables.

A string is a special array that has one extra element at the end of the string, which always has the value of 0 (zero). This is known as a "null terminated string".

String Character Array Example

This example will show how to make a string and print it to the serial monitor window.

```
void setup() {  
  char my_str[6]; // an array big enough for a 5 character string  
  Serial.begin(9600);  
  my_str[0] = 'H'; // the string consists of 5 characters  
  my_str[1] = 'e';  
  my_str[2] = 'l';  
  my_str[3] = 'l';  
  my_str[4] = 'o';  
  my_str[5] = 0; // 6th array element is a null terminator  
  Serial.println(my_str);  
}
```

```
void loop() {
```

```
}
```

The following example shows what a string is made up of; a character array with printable characters and 0 as the last element of the array to show that this is where the string ends. The string can be printed out to the Arduino IDE Serial Monitor window by using **Serial.println()** and passing the name of the string.

This same example can be written in a more convenient way as shown below

```
void setup() {  
  char my_str[] = "Hello";  
  Serial.begin(9600);  
  Serial.println(my_str);  
}  
void loop() {  
  
}
```

In this sketch, the compiler calculates the size of the string array and also automatically null terminates the string with a zero. An array that is six elements long and consists of five characters followed by a zero is created exactly the same way as in the previous sketch.

Manipulating String Arrays

We can alter a string array within a sketch as shown in the following sketch.
Example

```
void setup() {  
  char like[] = "I like coffee and cake"; // create a string  
  Serial.begin(9600);  
  // (1) print the string  
  Serial.println(like);  
  // (2) delete part of the string  
  like[13] = 0;  
  Serial.println(like);  
  // (3) substitute a word into the string  
  like[13] = ' '; // replace the null terminator with a space  
  like[18] = 't'; // insert the new word  
  like[19] = 'e';  
  like[20] = 'a';  
  like[21] = 0; // terminate the string  
  Serial.println(like);  
}  
  
void loop() {  
  
}
```

Result

```
I like coffee and cake  
I like coffee  
I like coffee and tea
```

For more details use link https://www.tutorialspoint.com/arduino/arduino_quick_guide.htm

Arduino - Time

Arduino provides four different time manipulation functions. They are -

S.No.	Function & Description
1	<u>delay () function</u> The way the delay() function works is pretty simple. It accepts a single integer (or number) argument. This number represents the time (measured in milliseconds).
2	<u>delayMicroseconds () function</u> The delayMicroseconds() function accepts a single integer (or number) argument. There are a thousand microseconds in a millisecond, and a million microseconds in a second.
3	<u>millis () function</u> This function is used to return the number of milliseconds at the time, the Arduino board begins running the current program.
4	<u>micros () function</u> The micros() function returns the number of microseconds from the time, the Arduino board begins running the current program. This number overflows i.e. goes back to zero after approximately 70 minutes.

Arduino - Arrays

For more details use link https://www.tutorialspoint.com/arduino/arduino_quick_guide.htm

Arduino - I/O Functions

The pins on the Arduino board can be configured as either inputs or outputs. We will explain the functioning of the pins in those modes. It is important to note that a majority of Arduino analog pins, may be configured, and used, in exactly the same manner as digital pins.

Pins Configured as INPUT

Arduino pins are by default configured as inputs, so they do not need to be explicitly declared as inputs with `pinMode()` when you are using them as inputs. Pins configured this way are said to be in a high-impedance state. Input pins make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 megaohm in front of the pin.

This means that it takes very little current to switch the input pin from one state to another. This makes the pins useful for such tasks as implementing a capacitive touch sensor or reading an LED as a photodiode.

Pins configured as `pinMode(pin, INPUT)` with nothing connected to them, or with wires connected to them that are not connected to other circuits, report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

Pull-up Resistors

Pull-up resistors are often useful to steer an input pin to a known state if no input is present. This can be done by adding a pull-up resistor (to +5V), or a pull-down resistor (resistor to ground) on the input. A 10K resistor is a good value for a pull-up or pull-down resistor.

Using Built-in Pull-up Resistor with Pins Configured as Input

There are 20,000 pull-up resistors built into the Atmega chip that can be accessed from software. These built-in pull-up resistors are accessed by setting the `pinMode()` as `INPUT_PULLUP`. This effectively inverts the behavior of the `INPUT` mode, where `HIGH` means the sensor is OFF and `LOW` means the sensor is ON. The value of this pull-up depends on the microcontroller used. On most AVR-based boards, the value is guaranteed to be between 20k Ω and 50k Ω . On the Arduino Due, it is between 50k Ω and 150k Ω . For the exact value, consult the datasheet of the microcontroller on your board.

When connecting a sensor to a pin configured with `INPUT_PULLUP`, the other end should be connected to the ground. In case of a simple switch, this causes the pin to read `HIGH` when the switch is open and `LOW` when the switch is pressed. The pull-up resistors provide enough current to light an LED dimly connected to a pin configured as an input. If LEDs in a project seem to be working, but very dimly, this is likely what is going on.

Same registers (internal chip memory locations) that control whether a pin is `HIGH` or `LOW` control the pull-up resistors. Consequently, a pin that is configured to have pull-up resistors turned on when the pin is in `INPUT` mode, will have the pin configured as `HIGH` if the pin is then switched to an `OUTPUT` mode with `pinMode()`. This works in the other direction as well, and an output pin that is left in a `HIGH` state will have the pull-up resistor set if switched to an input with `pinMode()`.

```
pinMode(3,INPUT) ; // set pin to input without using built in pull up resistor  
pinMode(5,INPUT_PULLUP) ; // set pin to input using built in pull up resistor
```

Pins Configured as OUTPUT

Pins configured as `OUTPUT` with `pinMode()` are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (do not forget the series resistor), or run many sensors but not enough current to run relays, solenoids, or motors.

Attempting to run high current devices from the output pins, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often, this results in a "dead" pin in the microcontroller but the remaining chips still function adequately. For this reason, it is a good idea to connect the `OUTPUT` pins to other devices through 470 Ω or 1k resistors, unless maximum current drawn from the pins is required for a particular application.

pinMode() Function

The pinMode() function is used to configure a specific pin to behave either as an input or an output. It is possible to enable the internal pull-up resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pull-ups.

pinMode() Function Syntax

```
Void setup () {
```

```
    pinMode (pin , mode);
```

```
}
```

pin – the number of the pin whose mode you wish to set

mode – INPUT, OUTPUT, or INPUT_PULLUP.

Example

```
int button = 5 ; // button connected to pin 5
```

```
int LED = 6; // LED connected to pin 6
```

```
void setup () {
```

```
    pinMode(button , INPUT_PULLUP);
```

```
    // set the digital pin as input with pull-up resistor
```

```
    pinMode(button , OUTPUT); // set the digital pin as output
```

```
}
```

```
void setup () {
```

```
    If (digitalRead(button) == LOW) // if button pressed {
```

```
        digitalWrite(LED,HIGH); // turn on led
```

```
        delay(500); // delay for 500 ms
```

```
        digitalWrite(LED,LOW); // turn off led
```

```
        delay(500); // delay for 500 ms
```

```
    }
```

```
}
```

digitalWrite() Function

The digitalWrite() function is used to 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. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.

digitalWrite() Function Syntax

```
Void loop() {
```

```
    digitalWrite (pin ,value);
```

```
}
```

- pin – the number of the pin whose mode you wish to set
- value – HIGH, or LOW.

Example

```
int LED = 6; // LED connected to pin 6
```

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

```
void setup () {  
  digitalWrite(LED,HIGH); // turn on led  
  delay(500); // delay for 500 ms  
  digitalWrite(LED,LOW); // turn off led  
  delay(500); // delay for 500 ms  
}
```

analogRead() function

Arduino is able to detect whether there is a voltage applied to one of its pins and report it through the `digitalRead()` function. There is a difference between an on/off sensor (which detects the presence of an object) and an analog sensor, whose value continuously changes. In order to read this type of sensor, we need a different type of pin.

In the lower-right part of the Arduino board, you will see six pins marked “Analog In”. These special pins not only tell whether there is a voltage applied to them, but also its value. By using the `analogRead()` function, we can read the voltage applied to one of the pins.

This function returns a number between 0 and 1023, which represents voltages between 0 and 5 volts. For example, if there is a voltage of 2.5 V applied to pin number 0, `analogRead(0)` returns 512.

analogRead() function Syntax

```
analogRead(pin);
```

pin – the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Example

```
int analogPin = 3; // potentiometer wiper (middle terminal)  
  // connected to analog pin 3  
int val = 0; // variable to store the value read
```

```
void setup() {  
  Serial.begin(9600); // setup serial  
}
```

```
void loop() {  
  val = analogRead(analogPin); // read the input pin  
  Serial.println(val); // debug value  
}
```

Arduino - Pulse Width Modulation

Pulse Width Modulation or PWM is a common technique used to vary the width of the pulses in a pulse-train. PWM has many applications such as controlling servos and speed controllers, limiting the effective power of motors and LEDs.

analogWrite() Function

The `analogWrite()` function writes an analog value (PWM wave) to a pin. It can be used to light a LED at varying brightness or drive a motor at various speeds. After a call of the `analogWrite()` function, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` or a call to `digitalRead()` or `digitalWrite()` on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support `analogWrite()` on pins 9, 10, and 11.



The Arduino Due supports `analogWrite()` on pins 2 through 13, and pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.

analogWrite() Function Syntax

`analogWrite (pin , value) ;` value – the duty cycle: between 0 (always off) and 255 (always on)

Example

```
int ledPin = 9; // LED connected to digital pin 9
int analogPin = 3; // potentiometer connected to analog pin 3
int val = 0; // variable to store the read value
void setup() {
  pinMode(ledPin, OUTPUT); // sets the pin as output
}
void loop() {
  val = analogRead(analogPin); // read the input pin
  analogWrite(ledPin, (val / 4)); // analogRead values go from 0 to 1023,
  // analogWrite values from 0 to 255
}
```

while(!serial)

```
void setup() {  
  //Initialize serial and wait for port to open:  
  Serial.begin(9600);  
  while (!Serial) {  
    ; // wait for serial port to connect. Needed for native USB  
  }  
}
```

```
void loop() {  
  //proceed normally  
}
```