SOFTWARE DISCRIPTION

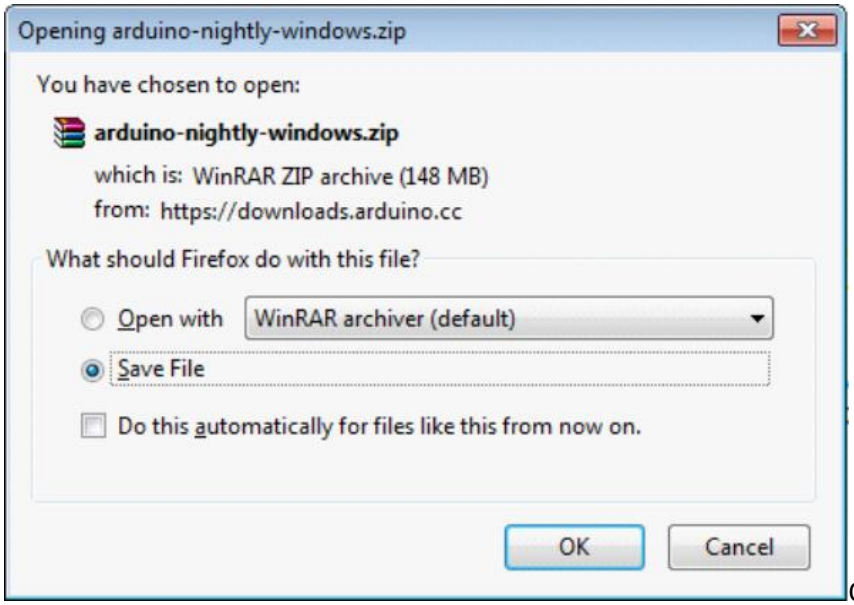
Arduino ide is the software used to write-compile-upload program to arduino.Its a open source software

- **Procedure to Install Arduino Software (IDE)**

Step 1 – First we must have an Arduino board and a USB cable. In case we use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, we will need a standard USB cable (A plug to B plug), the kind we would connect to a USB printer as shown in Fig. 3.1.5.1 (a). In case we use Arduino Nano, we will need an A to Mini-B cable instead as shown in Fig. 3.1.5.1 (b).

Fig 3.1.5.1 (b) A to Mini-B Cable

Step 2 – Download Arduino IDE Software.



One can get different versions of Arduino IDE from the Download page on the Arduino Official website. We must select our software, which is compatible with our operating system (Windows, IOS, or Linux). Unzip the file, after downloading it completely.

Fig. 3.1.5.2 Downloading Arduino IDE

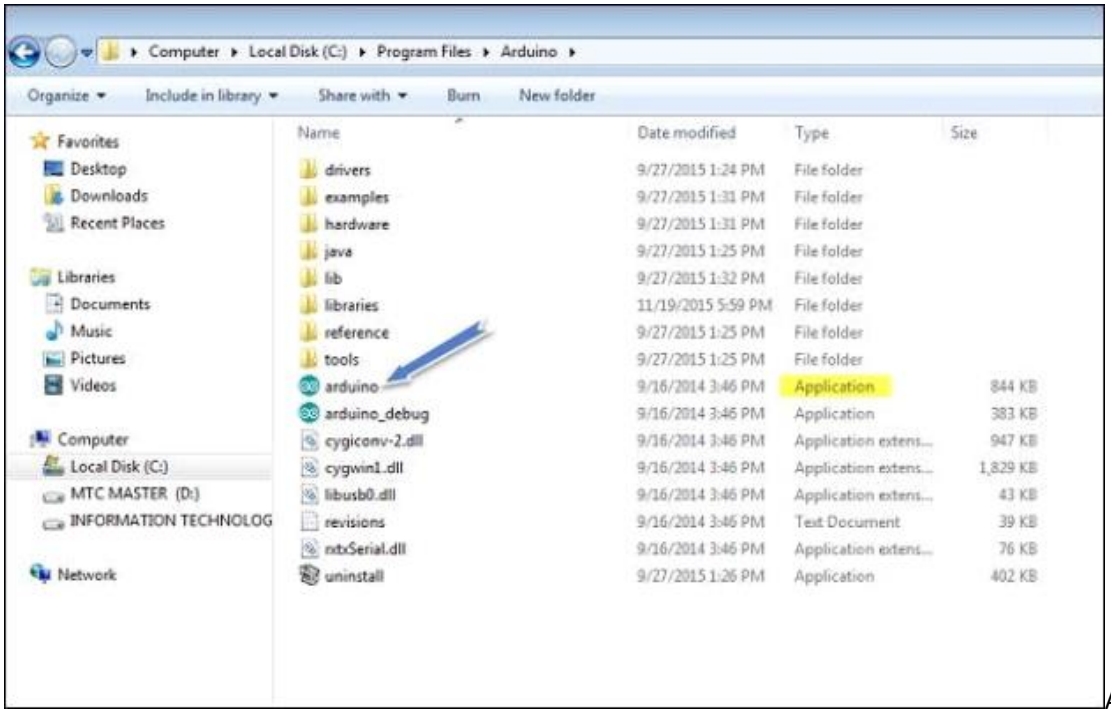
Step 3 – Power up your board.

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 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.

Fig. 3.1.5.4 Launching Arduino IDE

Step 5 – Open your first project.

Once the software starts, we have two options:

- Create a new project.

To create a new project, select File →**New,** as shown in Fig. 3.1.5.5 (a).

- Open an existing project example.

To open an existing project example, select File → Example → Basics → Blink, as shown in Fig. 3.1.5.5 (b).

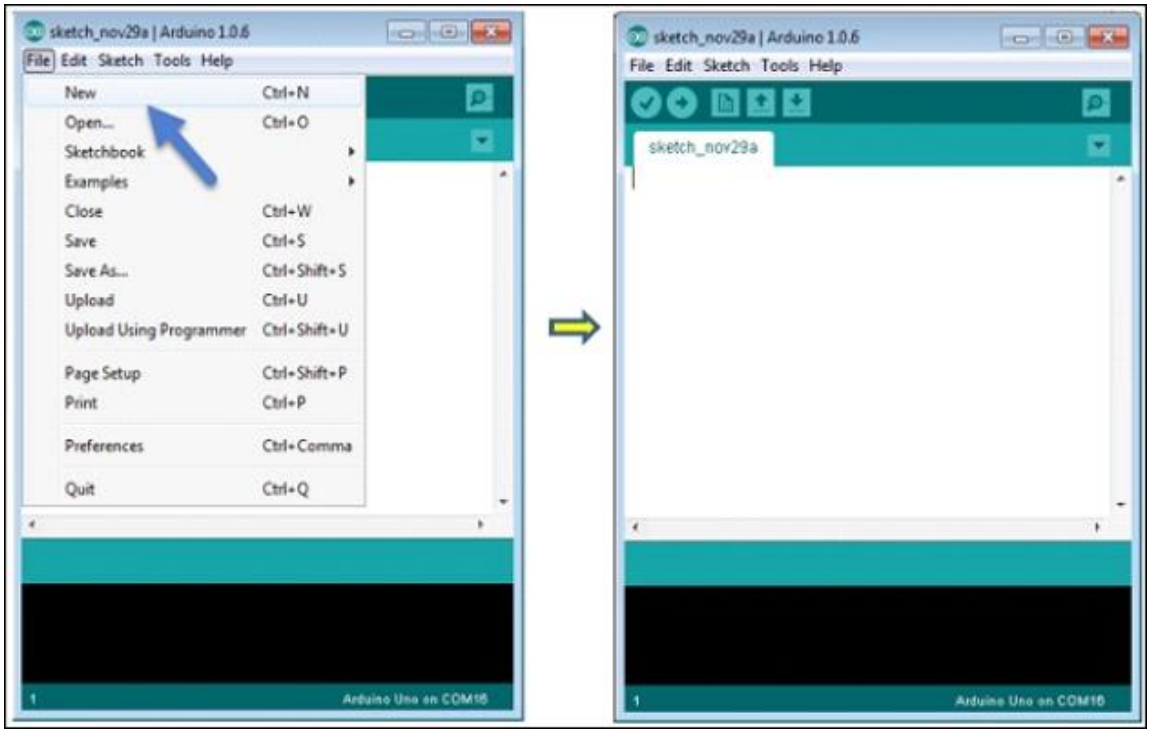


Fig. 3.1.5.5 (a) Creating a New Project

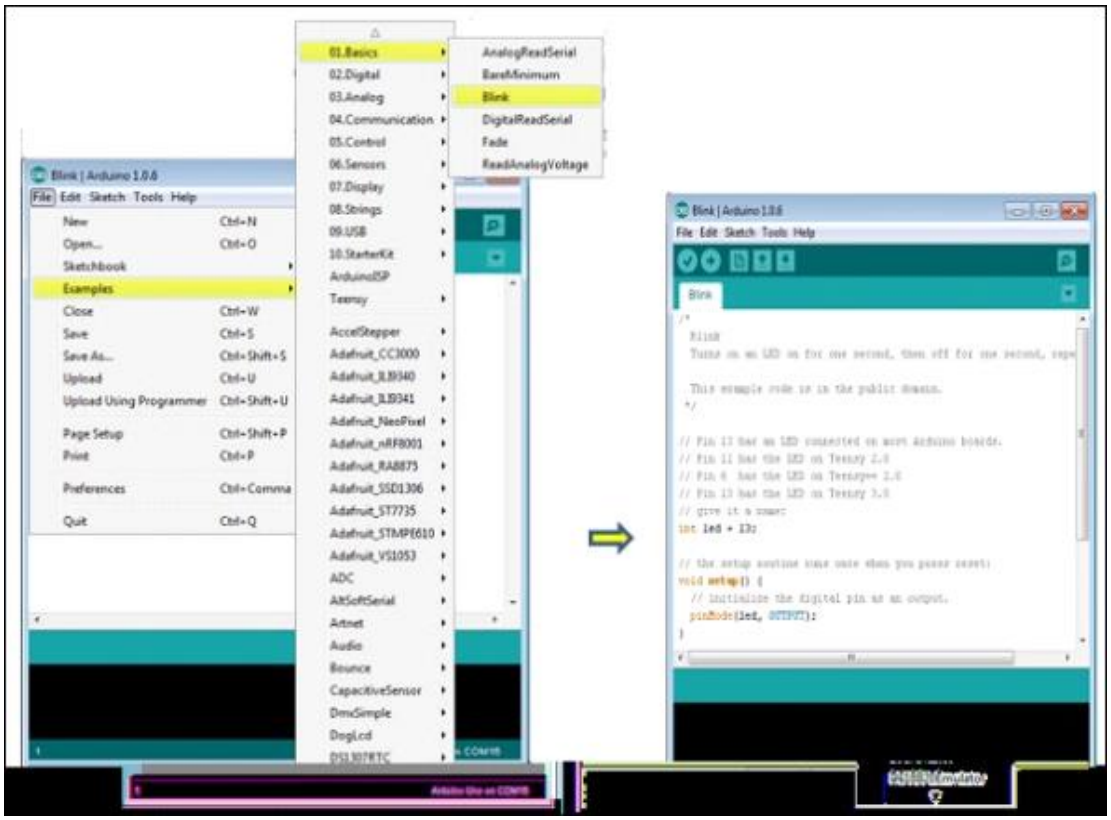


Fig. 3.1.5.5 (b) Opening an Existing Project

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

Step 6 – Select the respective Arduino board.

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

Go to Tools → Board and select the board.

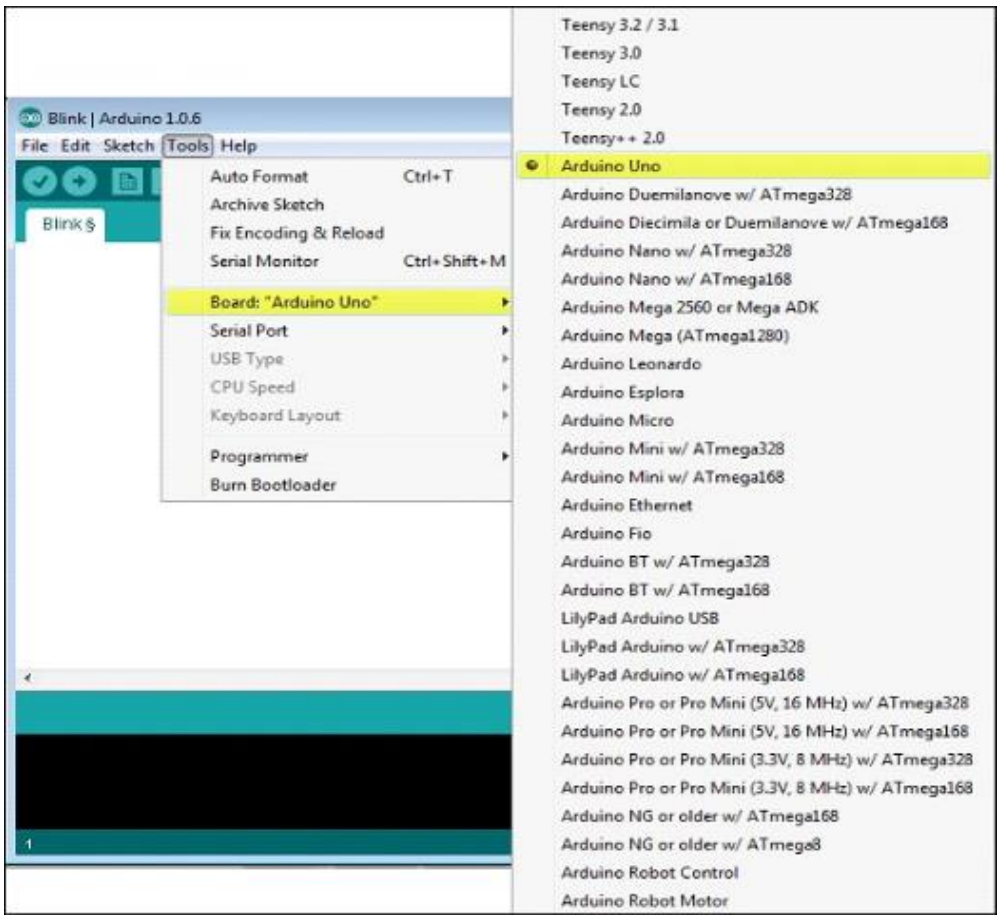


Fig. 3.1.5.6 Selecting the Arduino Board

Step 7 – Select the 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.

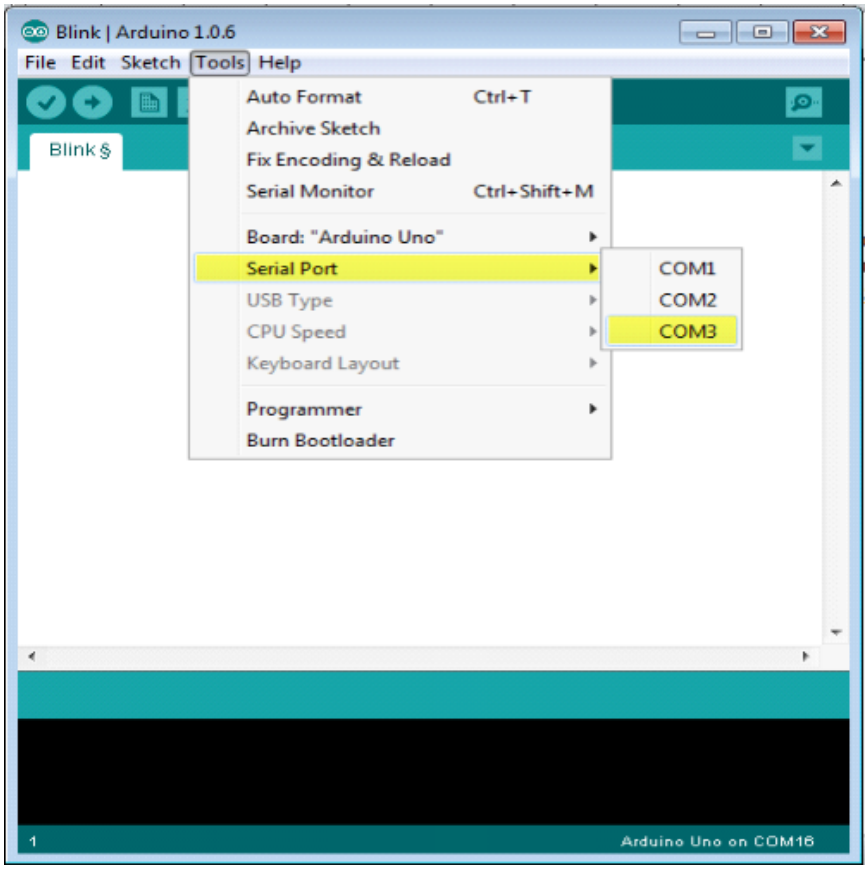
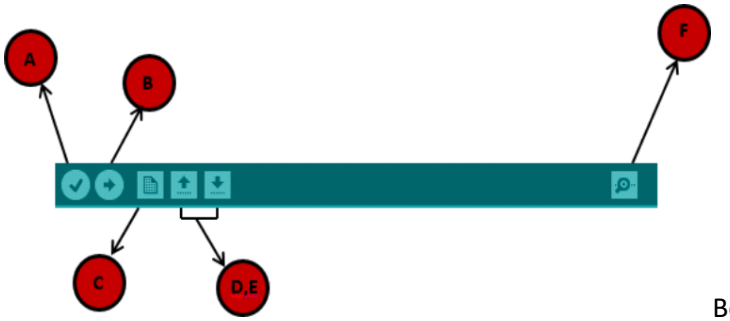


Fig. 3.1.4.7 Selecting the Serial Port

Step 8 – Upload the program to the board.



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

Fig. 3.1.4.8 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 send and receive the serial data from the board.

Now, simply click the "Upload" button in the environment. Wait few seconds; we 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.

- **Program Structure**

In this section, we will study in depth, the Arduino program structure and we will learn more new terminologies used in the Arduino world. 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. Arduino programs are called sketch.

Arduino programs can be divided in three main parts: **Structure, Values** (variables and constants), and **Functions**. Let us learn about the Arduino software program, step by step, and how to 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

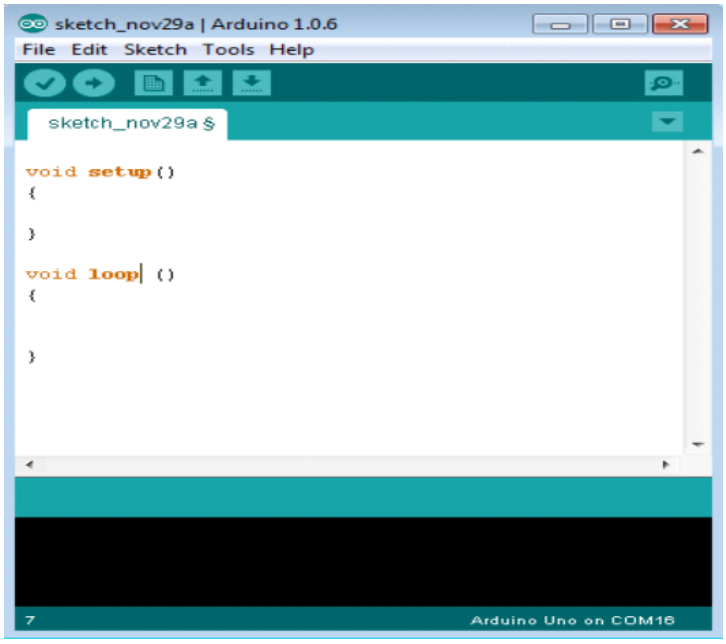


Fig. 3.1.6 Structure of a Sketch

The **setup ()** function is called when a sketch starts. It is used 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.

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 our program to change and respond. It actively controls the Arduino board.

- **Programming using Embedded C**

C is a high-level programming language intended for system programming. Embedded C is an extension that provides support for developing efficient programs for embedded devices. Yet, it is not a part of the C language. In our Internship program, we employed Embedded C programs to write sketches to be dumped on Arduino Uno.

- **Introduction to Embedded C**

Embedded C programming language is an extension to the traditional C programming language that is used in embedded systems. The embedded C programming language uses the same syntax and semantics as the C programming language.

The only extension in the Embedded C language from normal C Programming Language is the I/O Hardware Addressing, fixed-point arithmetic operations, accessing address spaces, etc.

- **Basic Structure of Embedded C Program:**

The embedded C program has a structure similar to C programming. The five layers of Embedded C programming structure are:

- Comments
- Pre-processor directives
- Global declaration
- Local declaration
- Main function()

The whole code follows the below outline. This is the basic structure of the embedded c program. Each code has a similar outline. Now let us learn about each of this layer in detail.

Outline of an Embedded C code is as shown below:

- Multiline Comments Denoted using /*……*/
- Single Line Comments Denoted using //
- Pre-processor Directives #include<…> or #define
- Global Variables Accessible anywhere in the program
- Function Declarations Declaring Function
- Main Function Main Function, execution begins here

7. {
   - Local Variables Variables confined to main function
   - Function Calls Calling other Functions
   - Infinite Loop Like while (1) or for (;;)
   - Statements . . . . .
12. ….
13. ….
14. }
15. Function Definitions Defining the Functions
16. {
    - Local Variables Local Variables confined to this Function
    - Statements . . . . .
19. ….
20. ….
21. }

- **Comment Section:** Comments are simple readable text, written in code to make it more understandable to the reader. Usually comments are written in // or /* */.

Example: //Test program

- **Pre-processor Directives Section:** The Pre-Processor directives tell the compiler which files to look in to find the symbols that are not present in the program.

For Example, in 8051 Keil compiler we use,

- #include<reg51.h>

- **Global Declaration Section:** The global variables and the user-defined functions are declared in this part of the code. They can be accessed from anywhere.

```
1.  void delay (int);
```

- **Local Declaration Section:** These variables are declared in the respective functions and cannot be used outside the main function.
- **Main Function Section:** Every C programs need to have the main function. So does an embedded C program. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket. Both the declaration and execution part are inside the curly braces.

Example:

```
1.  void main(void) // Main Function 2.     {
3.  P1 = 0x00;
4.  while(1)
5.  {
6.  P1 = 0xFF;
7.  delay(1000);
8.  P1 = 0x00;
9.  delay(1000);
10. }
11. }
```

- **Function Definition Section:** The function is defined in this section.

- **Arduino Code libraries**
- **Library Structure**

- A library is a folder comprised with C++ (.cpp) code files and C++ (.h) header files.

- The .h file describes the structure of the library and declares all its variables and functions.
- The .cpp file holds the function implementation.
- **Importing Libraries**

The first thing to do is to find the library we want to use out of the many libraries available online. After downloading it to our computer, we just need to open Arduino IDE and click on Sketch > Include Library > Manage Libraries. We can then select the library we want to import. Once the process is complete the library will be available in the sketch menu.

- **Arduino Code Explanation**

Arduino code is written in C++ with an addition of special methods and functions. C++ is a human-readable programming language. When we create a sketch (the name given to Arduino code files), it is processed and compiled to machine language.

- **Code Structure**

The basic concepts which one should know to write a program on Arduino IDE are discussed below:

- **Libraries**

In Arduino, much like other leading programming platforms, there are built-in libraries that provide basic functionality. In addition, its possible to import other libraries and expand the Arduino board capabilities and features. These libraries are roughly divided into libraries that interact with a specific component or those that implement new functions.

To import a new library, we need to go to Sketch > Import Library

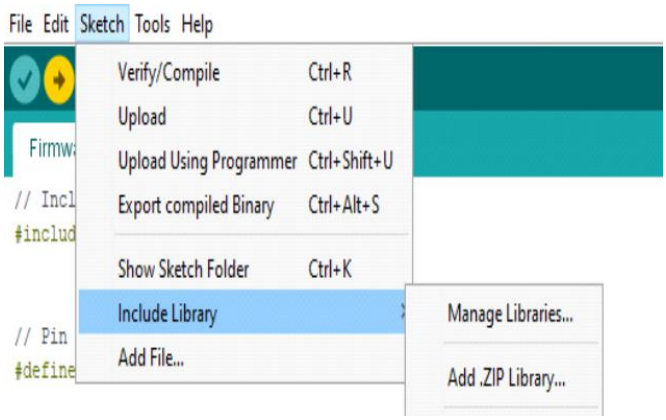


Fig. 3.2.2.1 Including Libraries on IDE

In addition, at the top of our file, we have to use #include to include external libraries. We can also create custom libraries to use in isolated sketches.

- **Pin Definitions**

To use the Arduino pins, we need to define which pin is being used and its functionality. A convenient way to define the used pins is by using:

#define pinName pinNumber.

The functionality is either input or output and is defined using the pinMode () method in the setup section.

- **Declarations**

- **Variables**

Whenever were using Arduino, we need to declare global variables and instances to be used later on. In a nutshell, a variable allows us to name and store a value to be used in the future. For example, we would store data acquired from a sensor in order to use it later. To declare a variable we simply define its type, name and initial value. Its worth mentioning that declaring global variables isnt an absolute necessity. However, its advisable to declare variables to make it easy to utilize our values further down the line.

- **Instances**

In software programming, a class is a collection of functions and variables that are kept together in one place. Each class has a special function known as a constructor, which is used to create an instance of the class. In order to use the functions of the class, we need to declare an instance for it.

- **Setup()**

Every Arduino sketch must have a setup function. This function defines the initial state of the Arduino upon boot and runs only once.

Here well define the following:

- Pin functionality using the pinMode function
- Initial state of pins
- Initialize classes
- Initialize variables
- Code logic

- **Loop()**

The loop function is also a must for every Arduino sketch and executes once setup () is complete. It is the main function and as its name hints, it runs in a loop over and over again. The loop describes the main logic of our circuit.

- **Code Logic**

The basic Arduino code logic is an if-then structure and can be divided into 4 blocks:

- **Setup** - will usually be written in the setup section of the Arduino code, and performs things that need to be done only once, such as sensor calibration.

- **Input** - at the beginning of the loop, read the inputs. These values will be used as conditions (if) such as the ambient light reading from an LDR using analogRead ().
- **Manipulate Data** - this section is used to transform the data into a more convenient form or perform calculations. For instance, the AnalogRead () gives a reading of 0-1023 which can be mapped to a range of 0-255 to be used for PWM. (See analogWrite ())
- **Output** - this section defines the final outcome of the logic (then) according to the data calculated in the previous step. Looking at an example of the LDR and PWM, turns on an LED only when the ambient light level goes below a certain threshold.

- **From Software to Hardware**

There is a lot to be said of Arduinos software capabilities, but its important to remember that the platform is comprised of both software and hardware. The two work in tandem to run a complex operating system.

Code → Compile → Upload → Run

At the core of Arduino, is the ability to compile and run the code.

After writing the code in the IDE we need to upload it to the Arduino. Clicking the Upload button (the right-facing arrow icon), will compile the code and upload it if it passed compilation. Once the upload is complete, the program will start running automatically.