Part 1

# lesson 5

## Blink and Add Libraries

## Overview

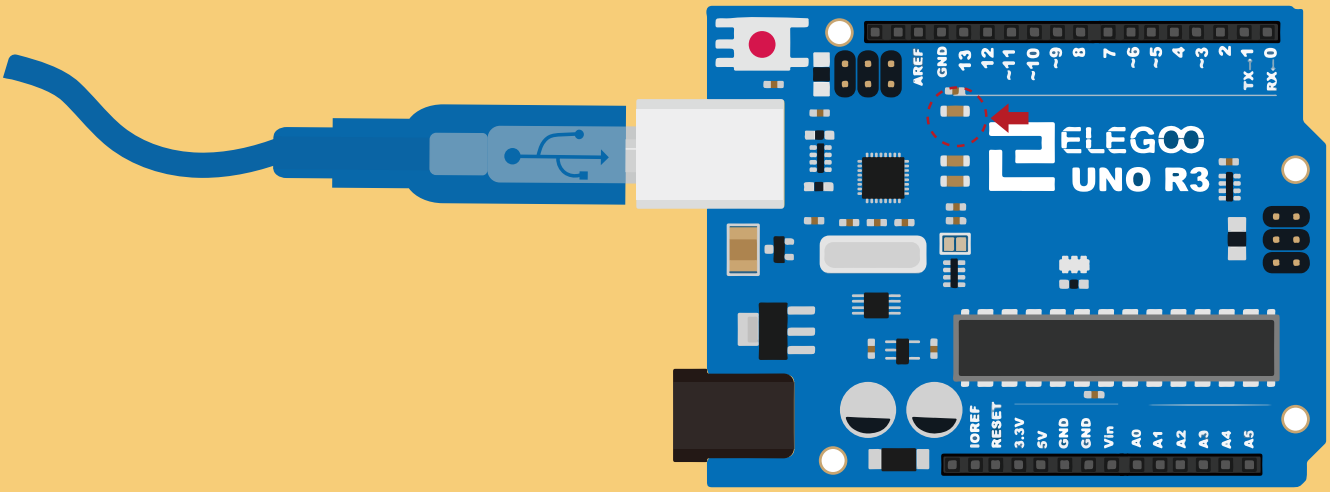
In this lesson, you will learn how to program your UNO R3 controller board to blink the Arduino' s built-in LED, and how to download programs by basic steps.
In addition, we need to learn how to add libraries for which, we can use library functions in future learning to expand Arduino functions more easily.

## Component Required

(1) x Elegoo Uno R3

## Principle

The UNO R3 board has rows of connectors along both sides that are used to connect to several electronic devices and plug-in 'shields' that extend its capability.
It also has a single LED that you can control from your sketches. This LED is built onto the UNO R3 board and is often referred to as the 'L' LED as this is how it is labeled on the board.



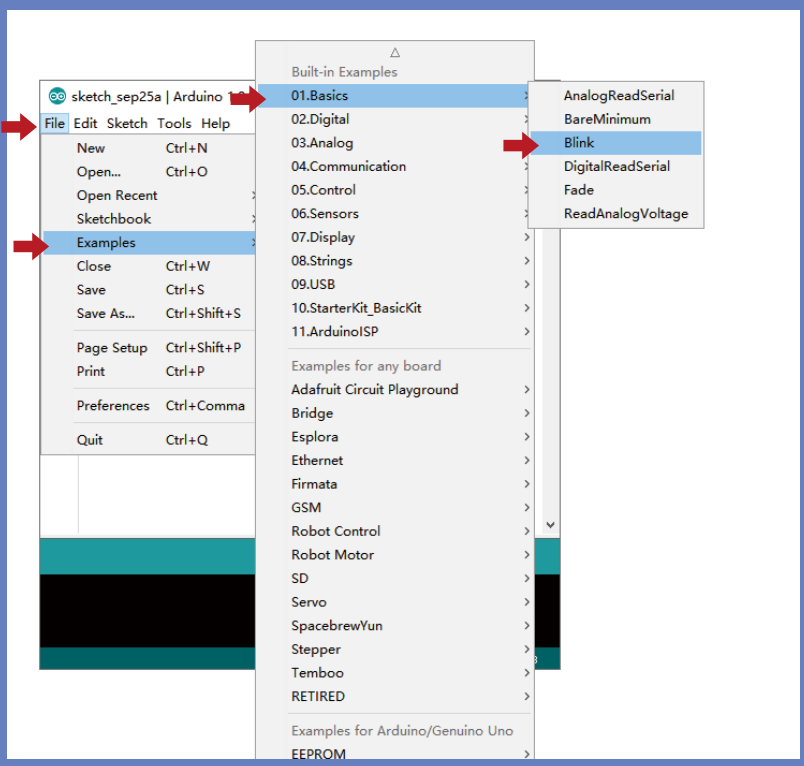

You may find that your UNO R3 board's 'L' LED already blinks when you connect it to a USB plug. This is because the boards are generally shipped with the 'Blink' sketch pre-installed.

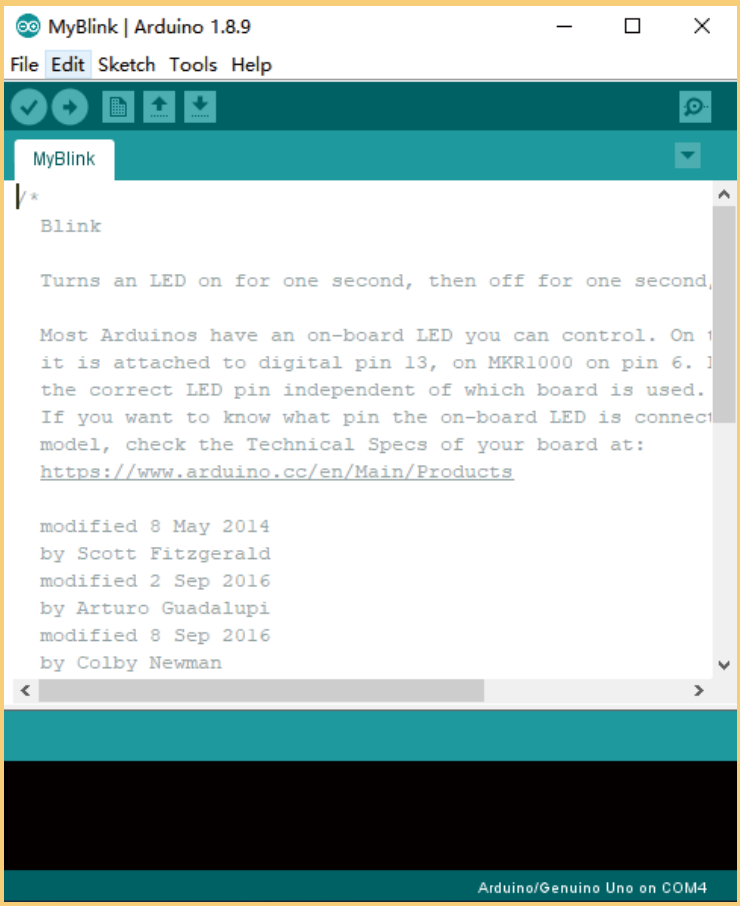
In this lesson, we will reprogram the UNO R3 board with our own Blink sketch and then change the rate at which it blinks.

In Lesson , you set up your Arduino IDE and made sure that you could find the right serial port for it to connect to your UNO R3 board. The time has now come to put that connection to the test and program your UNO R3 board.

The Arduino IDE includes a large collection of example sketches that you can load up and use. This includes an example sketch for making the 'L' LED blink.

Load the 'Blink' sketch that you will find in the IDE's menu system under File -> Examples -> 01.Basics.

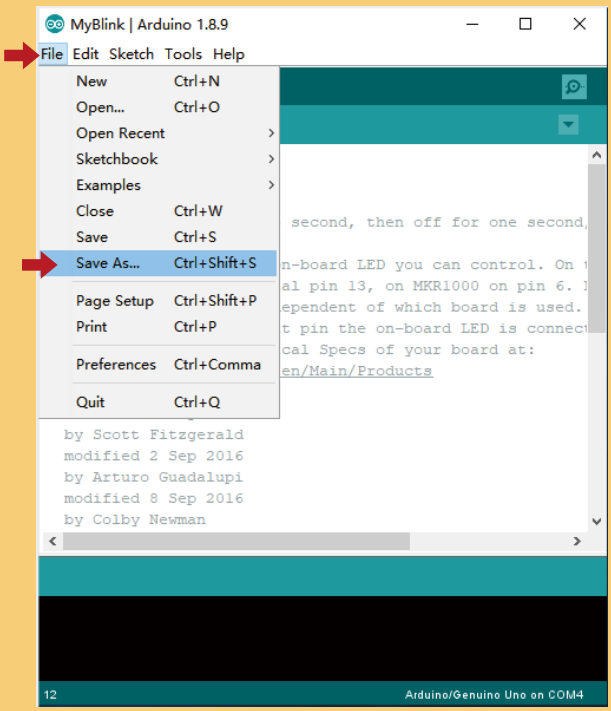


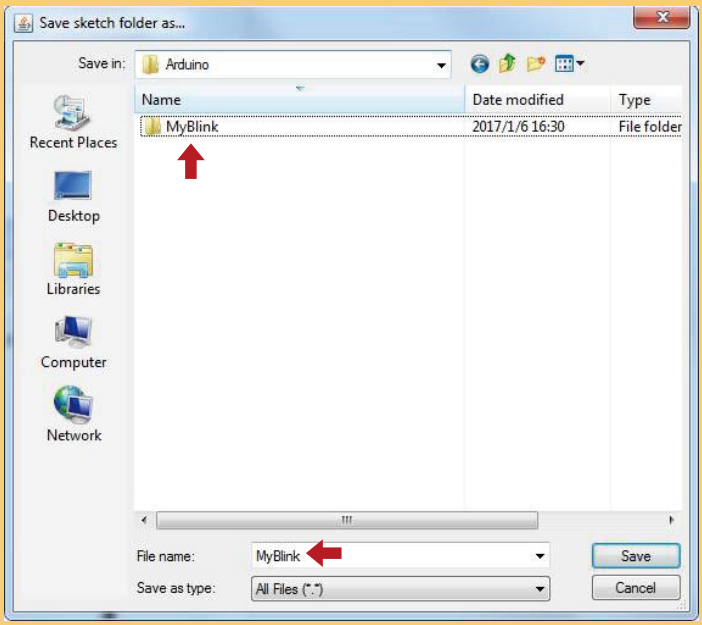




When the sketch window opens, enlarge it so that you can see the entire sketch in the window.

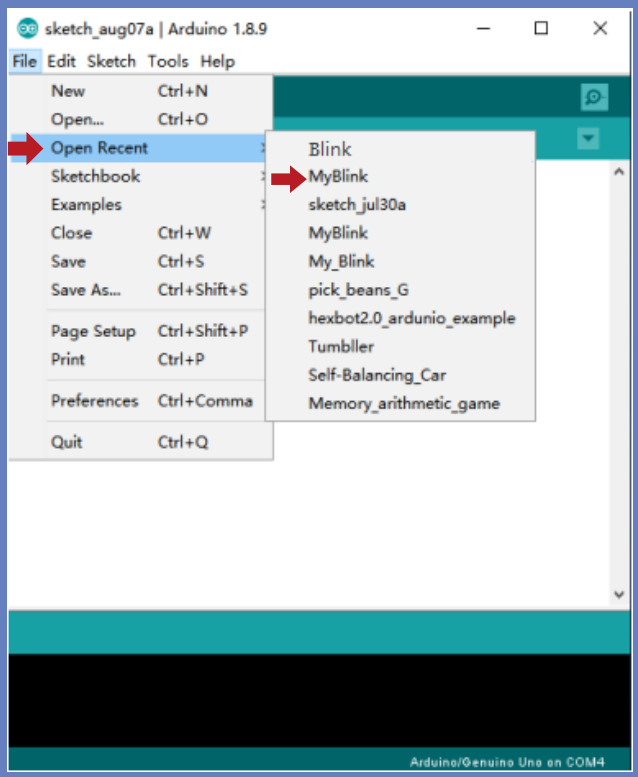
The example sketches included with the Arduino IDE are 'read-only'.
That is, you can upload them to an UNO R3 board, but if you change them, you cannot save them as the same file.

Since we are going to change this sketch, the first thing you need to do is save your own copy.
From the File menu on the Arduino IDE, select 'Save As.'
and then save the sketch with the name 'MyBlink'.

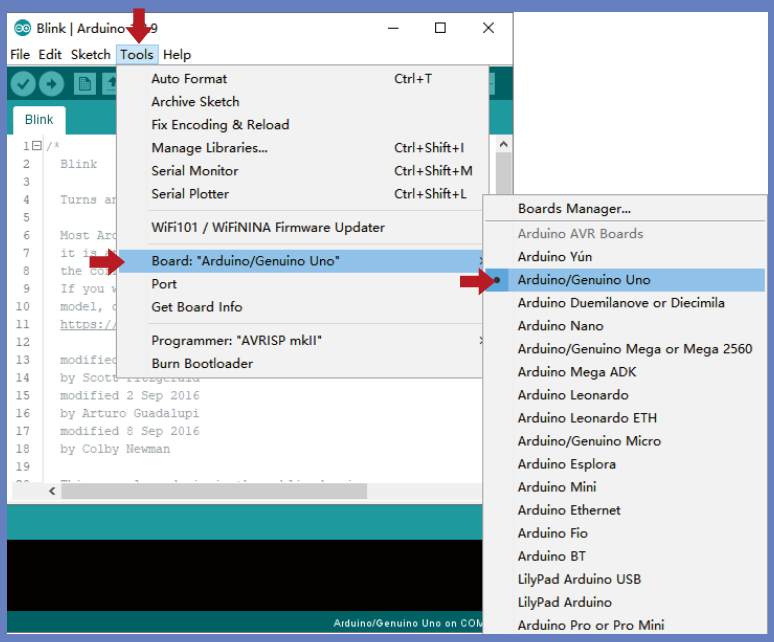


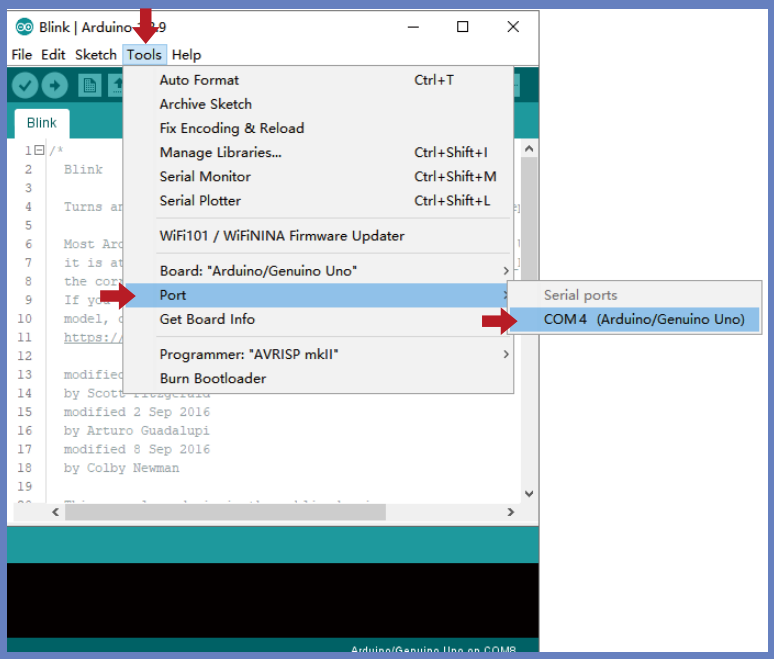




You have saved your copy of 'Blink' in your sketchbook. This means that if you ever want to find it again, you can just open it using the File -> Open Recent.



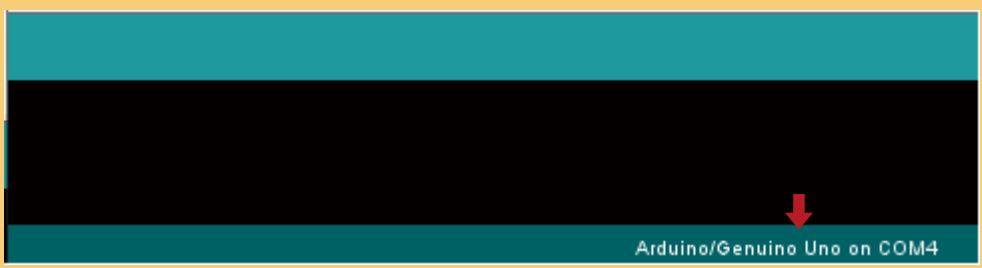
Attach your Arduino board to your computer with the USB cable and check that the 'Board Type' and 'Serial Port' are set correctly.



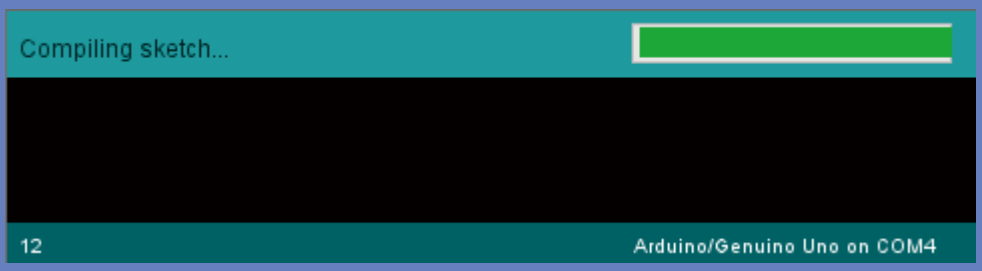

Note: The Board Type and Serial Port here are not necessarily the same as shown in picture.
If you are using 2560, then you will have to choose Mega 2560 as the Board Type, other choices can be made in the same manner.
And the Serial Port displayed for everyone is different, despite COM 4 chosen here, it could be COM3 or COM5 on your computer.
A right COM port is supposed to be COMX (arduino XXX), which is by the certification criteria.

The Arduino IDE will show you the current settings for board at the bottom of the window.

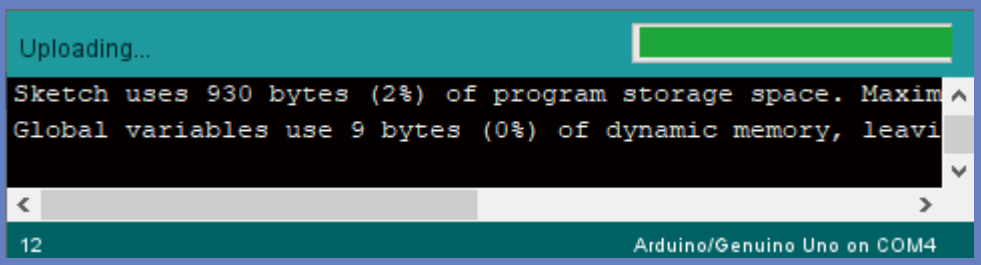


Click on the 'Upload' button. The second button from the left on the toolbar.

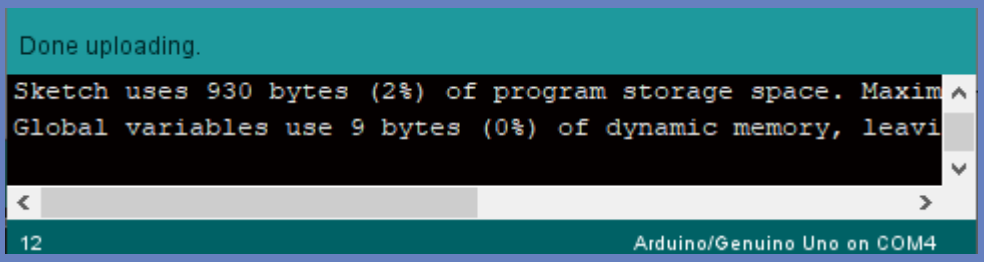

If you watch the status area of the IDE, you will see a progress bar and a series of messages. At first, it will say 'Compiling Sketch...'. This converts the sketch into a format suitable for uploading to the board.




Next, the status will change to 'Uploading'. At this point, the LEDs on the Arduino should start to flicker as the sketch is transferred.




Finally, the staus will change to 'Done'.




The other message tells us that the sketch is using 930 bytes of the 32,256 bytes available.
After the 'Compiling Sketch..' stage you could get the following error message:


It means that your board is not connected at all, or the drivers have not been installed (if necessary) or that the wrong serial port is selected.
If you encounter this, go back to Lesson 3 and check your installation.
Once the upload has completed, the board should restart and start blinking.

**Code :**

Note that a huge part of this sketch is composed of comments. These are not actual program instructions; Rather, they just explain how the program works.
They are there for your benefit.

**The Sketch start with**

```
/*
Blink
Turns an LED on for one second, then off for one second, repeatedly.
...
This example code is in the public domain. http://www.arduino.cc/en/Tutorial/Blink
*/
// the setup function runs once when you press reset or power the board
```

//
**[Further Syntax]**
**Description**

Line comments are lines in the program that are used to inform yourself or others about the way the program works. They are ignored by the compiler, and not exported to the processor, so they don' t take up any space in the microcontroller' s flash memory. Comments' only purpose is to help you understand (or remember), or to inform others about how your program works.

A single line comment begins with **//** (two adjacent slashes). This comment ends automatically at the end of a line. Whatever follows **//** till the end of a line will be ignored by the compiler.

/**/
**[Further Syntax]**
**Description**

The beginning of a block comment or a multi-line comment is marked by the symbol /* and the symbol */ marks its end. This type of comment is called so as this can extend over more than one line; Once the compiler reads the /* it ignores whatever follows until it encounters a */.

**The first block of code is :**

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

In this case, there is just one command there, which, as the comment states tells the Arduino board that we are going to use the LED pin as an output.

setup()
**[Sketch]**
**Description**

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

**{ ...}**
**[Further Syntax]**
**Description**

Curly braces (also referred to as just "braces" or as "curly brackets") are a major part of the C++ programming language. They are used in several different constructs, outlined below, and this can sometimes be confusing for beginners.

An opening curly brace **{ must always be followed by a closing curly brace }**. This is a condition that is often referred to as the braces being balanced. The Arduino IDE (Integrated Development Environment) includes a convenient feature to check the balance of curly braces. Just select a brace, or even click the insertion point immediately following a brace, and its logical companion will be highlighted.

Unbalanced braces can often lead to cryptic, impenetrable compiler errors that can sometimes be hard to track down in a large program. Because of their varied usages, braces are also incredibly important to the syntax of a program and moving a brace one or two lines will often dramatically affect the meaning of a program.

It is also mandatory for a sketch to have a 'loop' function.

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

Loop : Unlike the 'setup' function that only runs once, oppositely, the loop function runs constantly.

digitalWrite(Pin num , HIGH/LOW):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 for HIGH, 0V (ground) for LOW.

delay(ms):Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second).

ms : the number of milliseconds to pause.(range:0~4394967295).

Inside the loop function, the commands first of all turn the LED pin on (HIGH),
then 'delay' for 1000 milliseconds (1 second), then turn the LED pin off and pause for another second.

You are now going to make your LED blink faster. As you might have guessed, the key to this lies in changing the parameter in () for the 'delay' command.

```
30 // the loop function runs over and over again forever
31 void loop() {
32   digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the volt
33   delay(500)                         // wait for a second
34   digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the vo
35   delay(500)                         // wait for a second
36 }
```

This delay period is in milliseconds, so if you want the LED to blink twice as fast, change the value from 1000 to 500. This would then pause for half a second each delay rather than a whole second.

Upload the sketch again and you should see the LED start to blink more quickly.

## Installing Additional Arduino Libraries

Once you are comfortable with the Arduino software and using the built-in functions, you may want to extend the ability of your Arduino with additional libraries.

## What are Libraries?

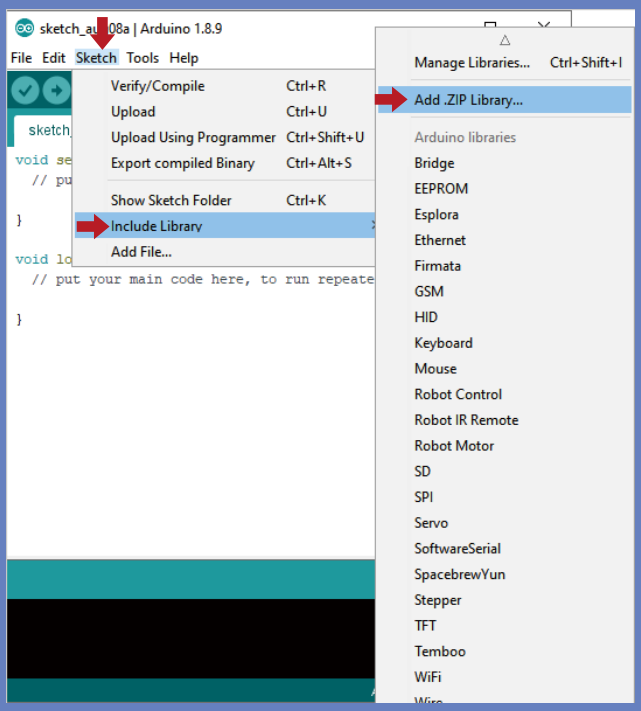
Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the built-in LiquidCrystal library makes it easy to talk to character LCD displays.
There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are listed in the reference. To use the additional libraries, you will need to install them.

## How to Install a Library

### 1) Importing a .zip Library

Libraries are often distributed as a ZIP file or folder. The name of the folder is the name of the library. Inside the folder will be a .cpp file, a .h file and often a keywords.txt file, examples folder, and other files required by the library. Starting with version 1.0.5, you can install 3rd party libraries in the IDE. Do not unzip the downloaded library, leave it as is.

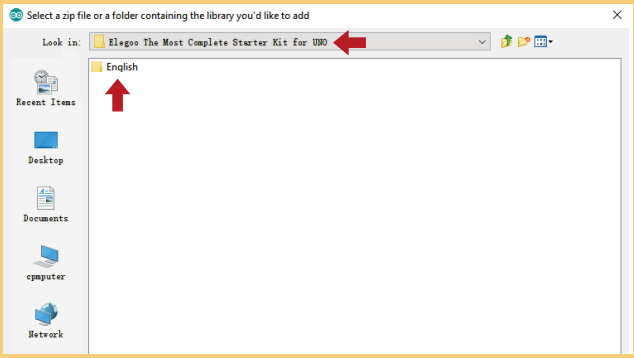
In the Arduino IDE, navigate to Sketch > Include Library. At the top of the drop down list, select the option to "Add .ZIP Library".

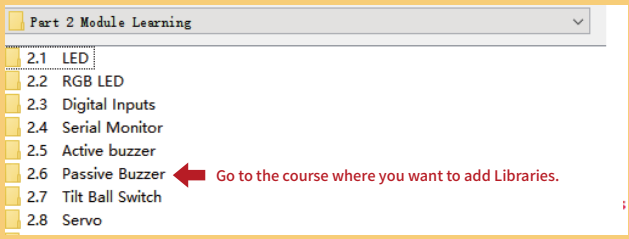


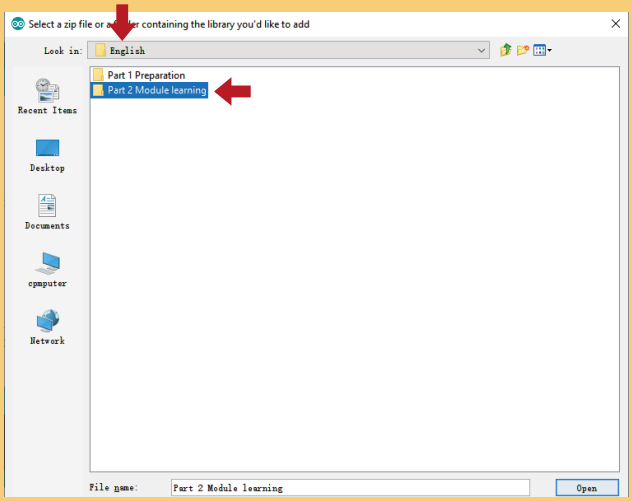

You will be prompted to select the library you would like to add. Navigate to the .zip file's location and open it.

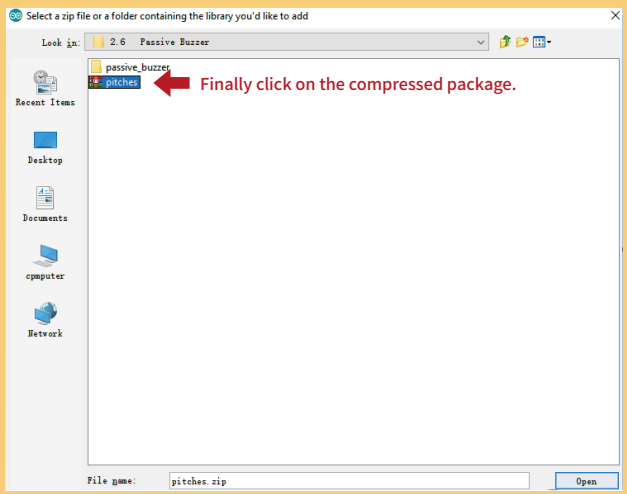
If the library is used in each course folder, it will be provided with the corresponding library compression package.

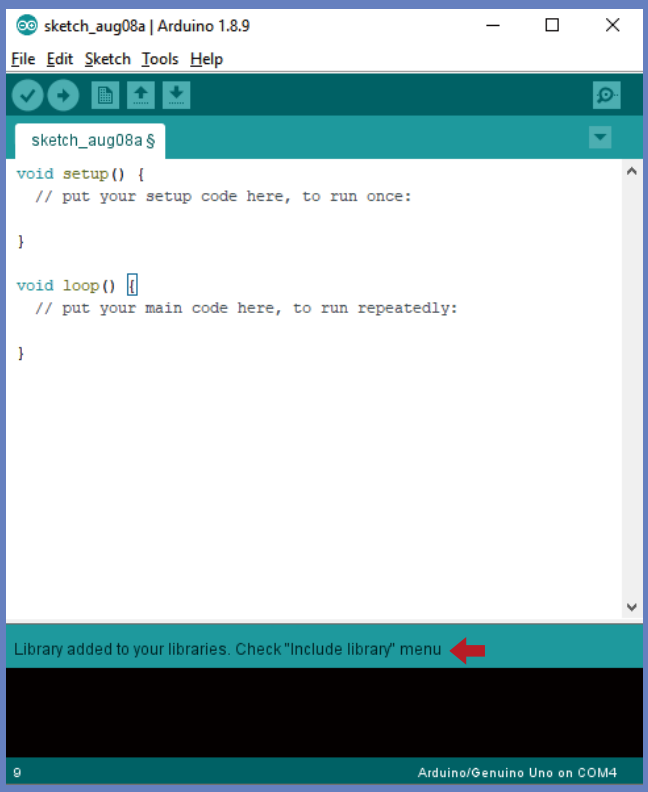
Take Lesson 11 as an example



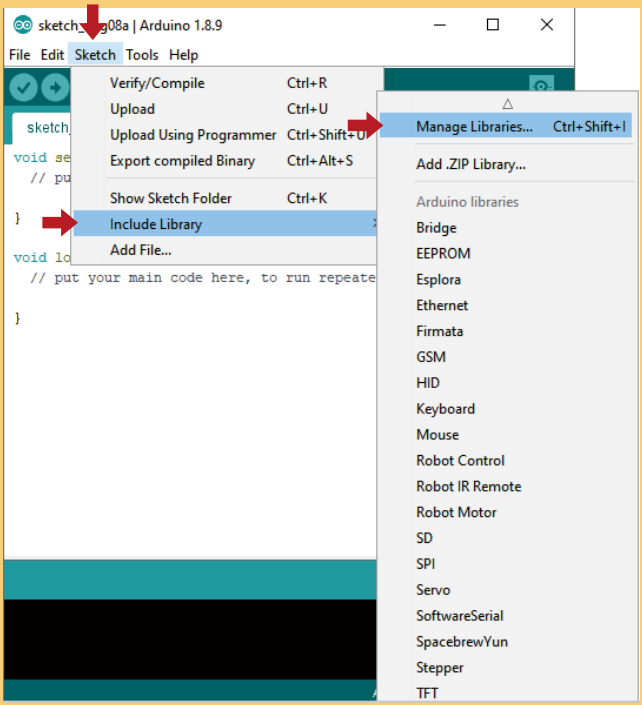



Return to the Sketch > Import Library menu. You should now see the library at the bottom of the drop-down menu. It is ready to be used in your sketch. The zip file has been expanded in the libraries folder in your Arduino sketches directory.

NB: the Library will be available to use in sketches, but examples for the library will not be exposed in the File > Examples until after the IDE has restarted.

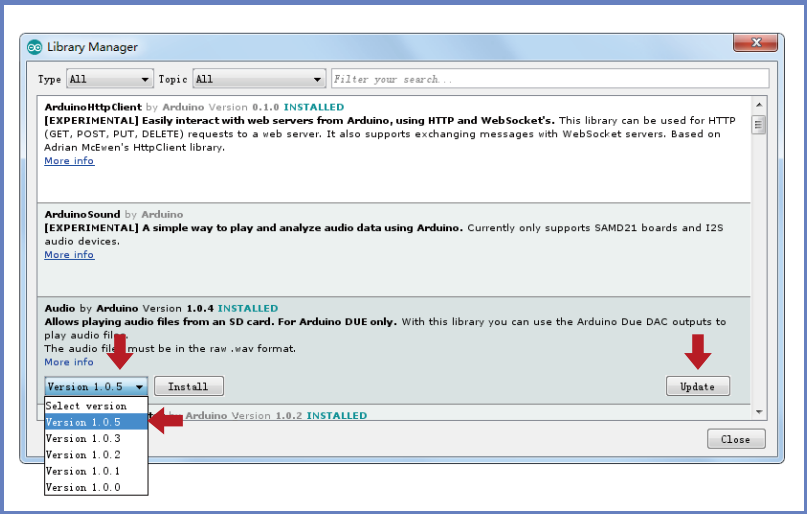
## 2) Using the Library Manager

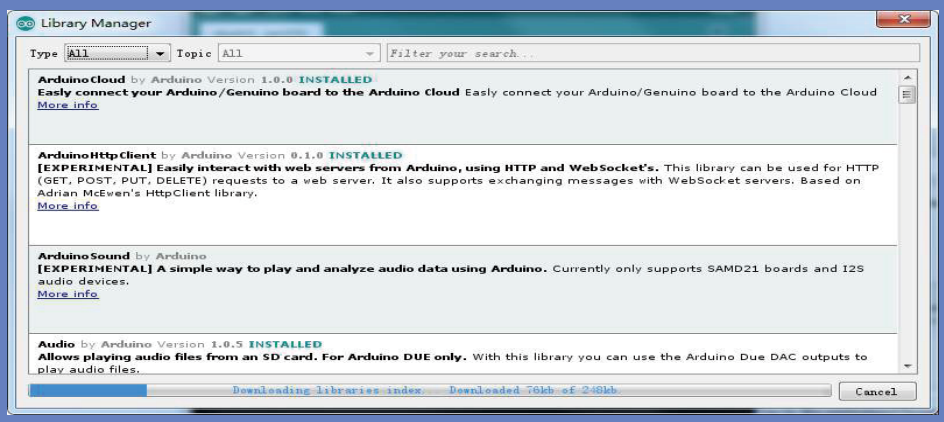
To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.8.9). Open the IDE and click to the "Sketch" menu and then Include Library > Manage Libraries.




Then the library manager will open and you will find a list of libraries that are already installed or ready for installation. In this example we will install the **Audio** library. Scroll the list to find it, then select the version of the library you want to install. Sometimes only one version of the library is available. If the version selection menu does not appear, don't worry: it is normal.

There are times you have to be patient with it, just as shown in the figure. Please refresh it and wait.




Finally click on install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an Installed tag should appear next to the Bridge library. You can close the library manager.


You can now find the new library available in the Include Library menu. If you want to add your own library, open a new issue on Github.
Those two are the most common approaches. MAC and Linux systems can be handled likewise.
The manual installation to be introduced below as an alternative may be seldom used and users with no needs may skip it.

### 3) Manual installation

To install the library, first quit the Arduino application. Then uncompress the ZIP file containing the library. For example, if you're installing a library called "ArduinoParty", uncompress ArduinoParty.zip. It should contain a folder called ArduinoParty, with files like ArduinoParty.cpp and ArduinoParty.h inside. (If the .cpp and .h files aren't in a folder, you'll need to create one. In this case, you'd make a folder called "ArduinoParty" and move into it all the files that were in the ZIP file, like ArduinoParty.cpp and ArduinoParty.h).

Drag the ArduinoParty folder into this folder (your libraries folder). Under Windows, it will likely be called "My Documents\Arduino\libraries". For Mac users, it will likely be called "Documents/Arduino/libraries". On Linux, it will be the "libraries" folder in your sketchbook.

Your Arduino library folder should now look like this (on Windows):
My Documents\Arduino\libraries\ArduinoParty\ArduinoParty.cpp
My Documents\Arduino\libraries\ArduinoParty\ArduinoParty.h
My Documents\Arduino\libraries\ArduinoParty\examples
or like this (on Mac and Linux):
Documents/Arduino/libraries/ArduinoParty/ArduinoParty.cpp
Documents/Arduino/libraries/ArduinoParty/ArduinoParty.h
Documents/Arduino/libraries/ArduinoParty/examples
....

There may be more files than just the .cpp and .h files, just make sure they're all there. (The library won't work if you put the .cpp and .h files directly into the libraries folder or if they're nested in an extra folder. For example: Documents\Arduino\libraries\ArduinoParty.cpp and Documents\Arduino\libraries\ArduinoParty\ArduinoParty\ArduinoParty.cpp won't work).

Restart the Arduino application. Make sure the new library appears in the Sketch->Import Library menu item of the software. That's it! You've installed a library !