# Code Libraries

A library is a file containing some related set of functions that we might want to reuse over multiple projects. This saves us the time and energy of re-writing that code. A library may contain a set of useful math functions, or the code that lets the Arduino interface with some specific device, or some code that interfaces with the serial port on the computer. A properly written and maintained library can make the process of building new projects far easier. The most common use of libraries in Arduino prototyping is using the manufacturer provided libraries for the devices you might buy.  This is far easier than writing this code yourself, but it's easy for this part of the code to become a 'black box' that you don't understand.

In the activity, you wrote and expanded code in the sketch that drove the 7 segment display. In this lecture, we'll examine the process of turning that sketch code into a reusable library, and importing and running that library to complete a sketch that offers the exact same functionality, but with a more elegant and manageable code structure.

An Arduino Library typically consists of two files:

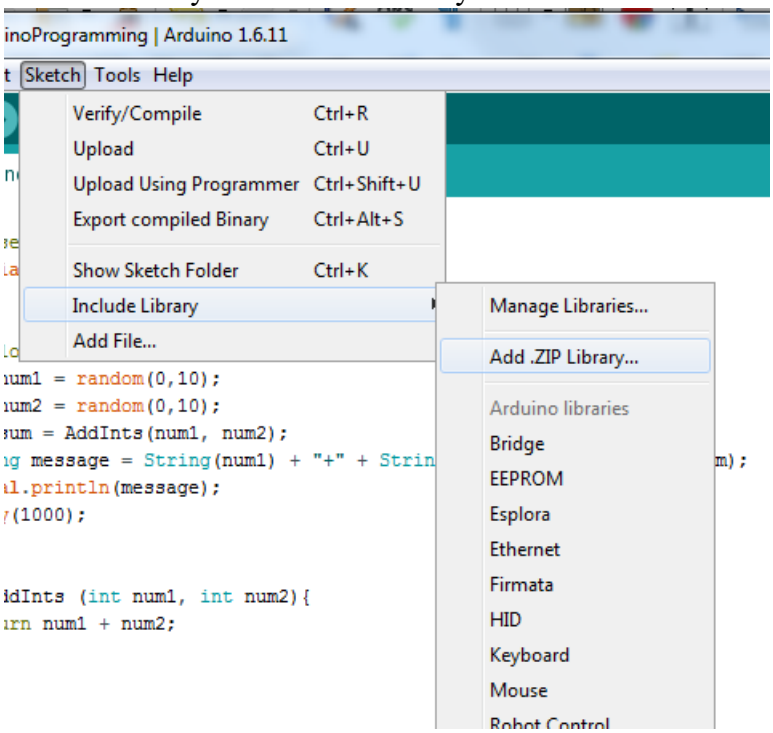
| **Header File (.h)** | **Class File (.cpp)** |
|---|---|
| This file contains a list of all the functions and properties that the library contains. This makes it easy for us to be able to know at a glance *what* the library does, without concerning ourselves with the details of *how* the library does it. For instance, we can look at the header file and know that our library has a function called average that expects a list of numbers and return the average value. What we don't know, is if the function calculates this value by adding all the numbers and dividing by the numbers of terms (median), or finding the most common number in the list (mode), or the middle value (mean), etc. To know that, we'd need to look in the class file. | This is the file that contains the actual code that implements the functions described in the header file. Its worth noting that library files are written in a *different* language than sketches. Sketches are written in a dialect of the language C, while libraries are written in the language C++. Because of this, actually writing your own library is outside the scope of this course. However, since using other people's libraries is such a common task in Arduino development, it's worth understanding them. |

## Importing A Library:

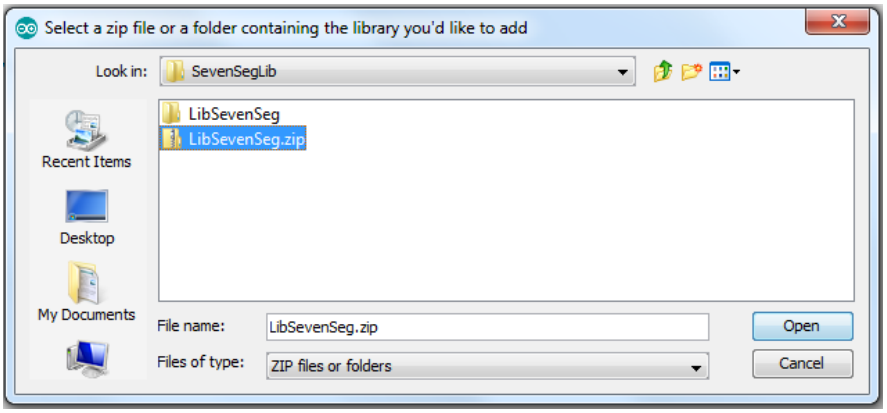
The first step in using a library is to download it to your machine. Typically you can download them from a manufacturer's website, or through an open-source developers code repository. There are also some more sophicated code management tools that can be used to retreive libraries. For this example, however, the library will be included in the source files as if you have already downloaded it in the form of a compressed (zip) folder. For the rest of the course, your libraries will be pre-imported for you before the class begins.

Next, we need to import the library into the Arduino IDE. This will extract the libraries into the folder where the Arduino stores all the libraries. This will make them available for any Ardiuno sketches that are written on this computer.

Click on the sketch menu, and the select
Include Library --> Add .ZIP library




This will open up a file dialog. You should find the library zip file at:
2.5 – 7SegmentDisplay/code/SevenSegLib/LibSevenSeg.zip



## Including A Library:

In the previous step, we imported the library so that it was available for the Arduino IDE to use. In this step, we will include the library into a sketch. This means the library will be compiled into the final binary file that we send to the Arduino. We only include libararies we intend to use, otherwise our binary file size would be problematic. We include the library with the following line at the top of the sketch:

```
#include <SevenSeg.h>
```

## Using A Libary:

Now that the libary is imported and included, we're ready to actually use it in our sketch. First, we create a variable that represents the library, tihs step is known as 'instantiating' the library.

```
SevenSeg sevenSeg;
```

This line can look a little confusing, but its a very common pattern. This is a variable declaration, so the first word is the type of the variable, and the second word is the name of the variable. In this instance, the type of the variable is the libary "SevenSeg" and the name of the variable we have unimaginatively assigned to "sevenSeg"with a lowercase "s".

With the variable that represents the library defined, we can now acccess the functions of the library by using the following syntax:

```
sevenSeg.setup();
```

This is how we call the setup function of the SevenSeg library. This is the code that puts the 7 segment display in a state where it is ready to display information; which is why we call it inside the sketches setup function.

```
sevenSeg.display(num, 2000);
```

Finally, we call the display word of the 7 segment display which shows the 4-digit number represented by num for 2000 cycles.

While there is some initial complexity in migrating your code into a library; there are some definite advantages in terms of the ease with which you can re-use that same code across multiple projects, and the way that the sketch itself becomes fairly trivial and easy to understand once you've burried the complexity of the lower level code in a library. This is an example of the value of code abstraction.