Software libraries

The purpose of a library is to capture the core functions needed in an application, and to put them in an independent library file. Then allow any application to use them in its own code. The libraries define what is called "objects" with defined "methods".

How does a library do this? Object "classes" with methods or functions Classes can be thought of a something like, for example, a car. This has many "methods", steering, speed, braking, etc. Any car has this. So if we can create an anonymous "car`' and use its methods we could for example create a race, with two, three or more cars. Controlling each separately.

Doing this in software.
We need a formal way to define classes, and create instances of them in our code. We do this in two files we put in the common library, then call them up in your specific code. The two files have extensions ".h" (header) and ".cpp" (C++ code). The outline of the library is in the ".h" file, and the detailed functions (known as methods in the jargon) - what the objet can do - is in the ".cpp" file.

Let's look at a real library.
This is called "Flash". This is a very simple library to blink an LED. It is written in two files, `Flash.h` and `Flash.cpp`. Later we will see how our own Arduino sketch code uses the library.

Here is Flash.h

```
// Flash.h

#ifndef Flash_h                         // prevent including twice
#define Flash_h

#include <Arduino.h>                    // Arduino library stuff

class Flash {                           // class name

      public:                           // public functions
            Flash();                // constructor

            void begin(int pin);        // assign pin
            void run(int on, int off);  // actual LED flash function

      private:
            int _pin;               // private copy of pin
};

#endif
```

1. The whole code is surrounded by

```
#ifndef Flash_h                         // prevent including twice
#define Flash_h
```

and...

```
#endif
```

This is precaution, we don't want, even by accident, the library to be "included' more than once in our sketch. So we check if an environment variable `"Flash_h" exists.` And if not we define it. Then at the end of the header file we end the "if" statement.

2. In between this check, comes the body of the header, which is an outline of the library that the computer uses to validate, that means check, the functions called up in your sketch do actually exist and that you have used them correctly.

3. Now the Arduino IDE provides a set of standard functions, like `digitalWrite(pin, state)` in a hidden library of its own. This is automatically visible to your sketch, but not to your library. So we have to include it or make it accessible to your library.

```
#include <Arduino.h>                    // Arduino library stuff
```

4. Then we define what is known as a "class", this is what the LED can do.

```
class Flash {                           // class name
```

...what the class can do

```
};
```

(don't ever forget the ending ";").

5. And the contents of this class are the outline of the actual functions

```
      public:                           // public functions
            Flash();                // constructor

            void begin(int pin);        // assign pin
            void run(int on, int off);  // actual LED flash function

      private:
            int _pin;               // private copy of pin
```

These are divided into two sections, "public" and "private". The public functions are visible

and can be used in your sketch code, the private ones are for use just in the ".cpp" function code file.

Now here we have a first function.

```
Flash();
```

which is known as the "constructor", this is used to create an object (the LED) that can be used in your code. You'll see how later.

Then the other two functions.

```
void begin(int pin);          // assign pin
void run(int on, int off);    // actual LED flash function
```

Which "begin" your code, initialising things, here it is the "pin" the LED is connected to. And "run" which launches the flashing of the LED for an "on" time and "off" time.

6. Then come any private functions (none here) or variables

```
int _pin;               // private copy of pin
```

_pin is an internal, private copy of the external "pin" number given by the begin function and is used in the .cpp file.

Here is Flash.cpp

```
//Flash.cpp

#include <Flash.h>                        // header with prototypes & variables

Flash::Flash() {                    // constructor

}

void Flash::begin(int pin) {              // assign pin
      pinMode(pin, OUTPUT);               // set pin mode
      _pin = pin;                   // copy to private variable
}

void Flash::run(int on, int off) {        // code for running led flash
      digitalWrite(_pin, HIGH);
      delay(on);
      digitalWrite(_pin, LOW);
      delay(off);
```

```
}
```

From the top.

1. The .cpp code needs access to the .h code outline, so it is "included" here.

```
#include <Flash.h>                        // header with prototypes & variables
```

2. Then each function needed is written. To show they are part of the library, called "Flash" each function is preceded by "Flash::"

```
Flash::Flash() {                    // constructor

}

void Flash::begin(int pin) {              // assign pin
      pinMode(pin, OUTPUT);               // set pin mode
      _pin = pin;                   // copy to private variable
}

void Flash::run(int on, int off) {        // code for running led flash
      digitalWrite(_pin, HIGH);
      delay(on);
      digitalWrite(_pin, LOW);
      delay(off);
}
```

The first allows us to construct an object with the functions (methods) of the library - as we shall see in our sketch code later.

The other two handle the initialisation of the pin the LED is connected to, and the actual blink routine.

Libraries folder
After the .h and .cpp file have been written, as plain text files. They are saved in the Arduino/ libraries folder in a new folder with the same name, Flash.

Your code
Now to use the library, here is the sketch

```
// HELLO_LIB

#include "Flash.h"            // include library

#define LED 13                // on board LED pin
```

```
Flash led;                    // create class object called "led"

void setup() {
  led.begin(LED);             // assign pin
}

void loop() {
  led.run(100, 2000);         // run to flash the LED, 0.1/2sec
}
```

1. The first, rather obvious, thing to do is to include the library you want to use in your sketch!

```
#include "Flash.h"            // include library
```

2. Next we need to create an object to use, led will now have all the functions (methods) of the library Flash

```
Flash led;                    // create class object called "led"
```

3. Then we write the normal setup and loop functions.. calling up the library functions

```
void setup() {
  led.begin(LED);             // assign pin
}

void loop() {
  led.run(100, 2000);         // run to flash the LED, 0.1/2sec
}
```

The library function for the object led is first in setup, where the LED pin number is passed in. Then in the loop the work is carried out, as the run library function is called with two parameters the on time and the off time in milliseconds.

That's it.