Arduino and The Way of the Ninja (For Absolute Beginners)

Table of Contents

Introduction
Chapter One: The Hardware
Chapter Two: The Software
Chapter Three: Making Projects
Chapter Four: More Projects
Chapter Five: Putting It All Together and Next Steps
About the Author
Connect with Jun Cecilio

Acknowledgements

This ebook is for all hobbyists and hobbyist at heart. May this be an inspiration for those who want to start and develop a productive activity involving microcontrollers. May it impart the courage and daring to tackle new skills and new projects. To boldly go where you have never gone before.

Thanks to the Arduino developers for providing open source hardware and software for engineers, technicians, hobbyists to address the need for a cheap, easily understandable microcontroller and its accompanying developer platform (IDE).

And lastly, thanks to my mother for the encouragement.

Introduction

"A ninja must always be calm, patient." If you are a true beginner in Arduino, then this ebook is for you. Written by a hobbyist for other hobbyist who want to start in microcontrollers. This is a quick breeze on the important learning points the beginner must master like a ninja to be able to move on to more complex projects on their own. .Along the way, it is hoped the reader also finds enjoyment in the assimilation of knowledge and imbibing the ninja spirit as applied to knowledge building.

In the first line of the previous paragraph, the above ninja characteristics are those that must be possessed when studying and mastering any new skill or knowledge. And in learning Arduino, both the hardware and the software aspect, one needs to be calm in spirit and patient in the approach to learning. This ebook is developed just for absolute beginners. Those without prior know how of Arduino or even microcontroller programming. The examples and instructions are made as simple as possible with lots of images for the reader to quickly start new projects to control devices and to prepare for more complex ones later. The reader will find no complicated circuitry or program codes in the software examples to aid in the understanding of core concepts. The training is stripped to its bare essential so as to let the reader enjoy the process of learning and not be confused or demotivated by any theory or concepts more appropriate for non-beginners. The approach is also methodical to clearly imprint the basics which is the foundation for a good understanding of Arduino.

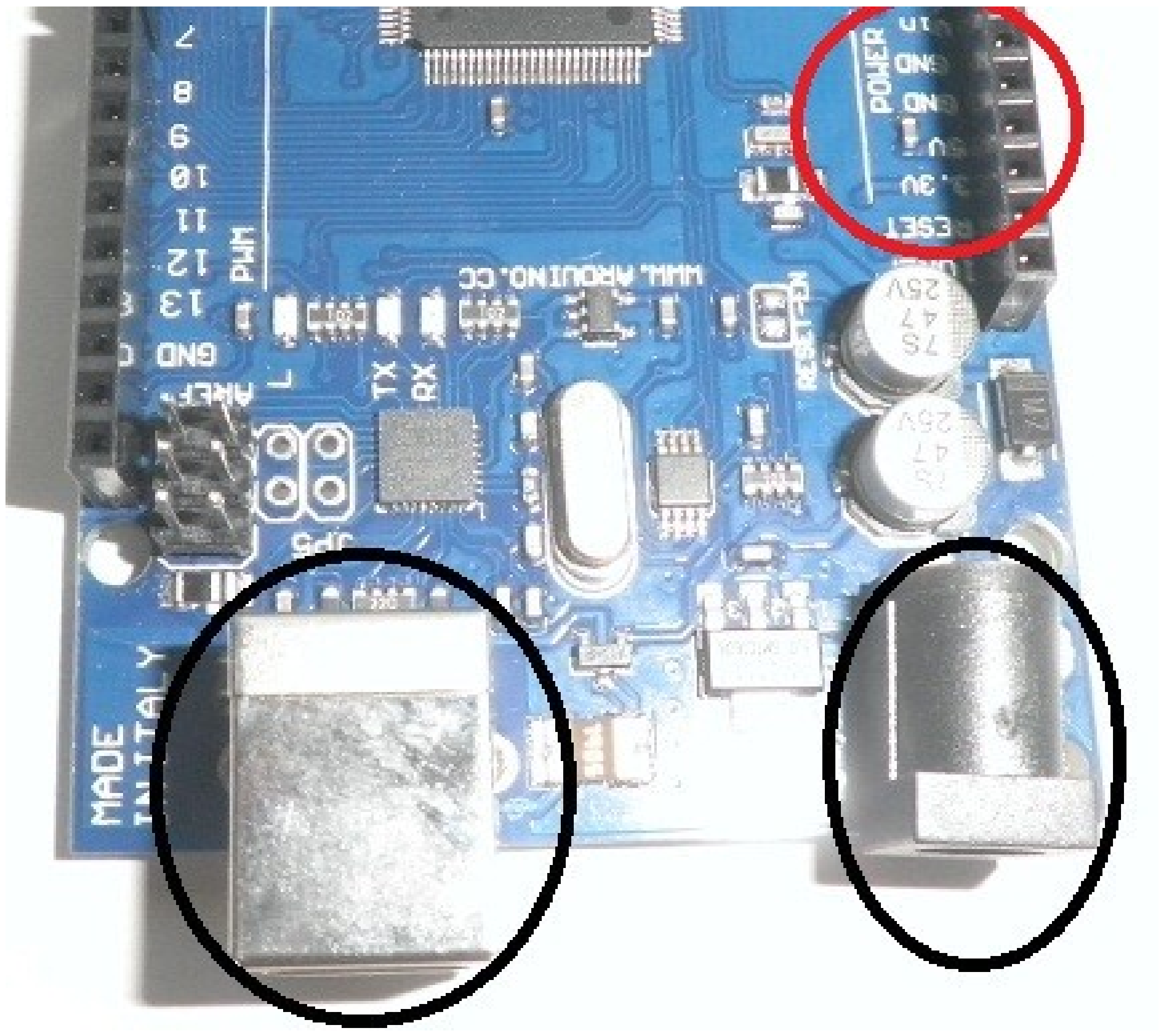
And so without further ado, the ninja Arduino mastery begins...

Chapter One: The Hardware

Weapons Savvy. "As the ninja is weapons savvy, so you must be in the knowledge and use of the Arduino hardware as your primary weapon/tool to make projects."

Compose your mind and be mentally calm as you tackle this first chapter on Arduino. First thing the reader must understand is that the Arduino hardware is open source, meaning the hardware schematics and components are freely available. One can browse the internet to download the circuit diagrams of common Arduino boards (e.g. UNO). The components can be bought from electronics shops or online. Of course making your own boards entails more than a beginner's know how and this book caters to pure beginners only. To be a ninja in the Arduino hardware aspect one also has to be calm and prepared to take advantage of the above fact. This is the actual starting point of the reader's learning and a calm mind and spirit of preparedness shall aid greatly in a quicker understanding of the concepts.

The picture below is for an Arduino ATMEGA2560 board. Do not worry if it is not the same as the board you are going to use or is using. It is the understanding of the basic hardware features that you are after and all Arduino boards share the same basic configuration. The atmega2560 is more than a beginner's board. It is more powerful and has more features. Even if you are only using an UNO board, whatever is discussed here applies also to UNO. And for any Arduino board for that matter.



The rectangular socket circled on the lower left part of the image is the USB port socket. In some Arduino boards, the serial port is smallerand may be more than one. Verify your model in the official Arduino website. The one circled on the lower right is the power plug socket or external jack.

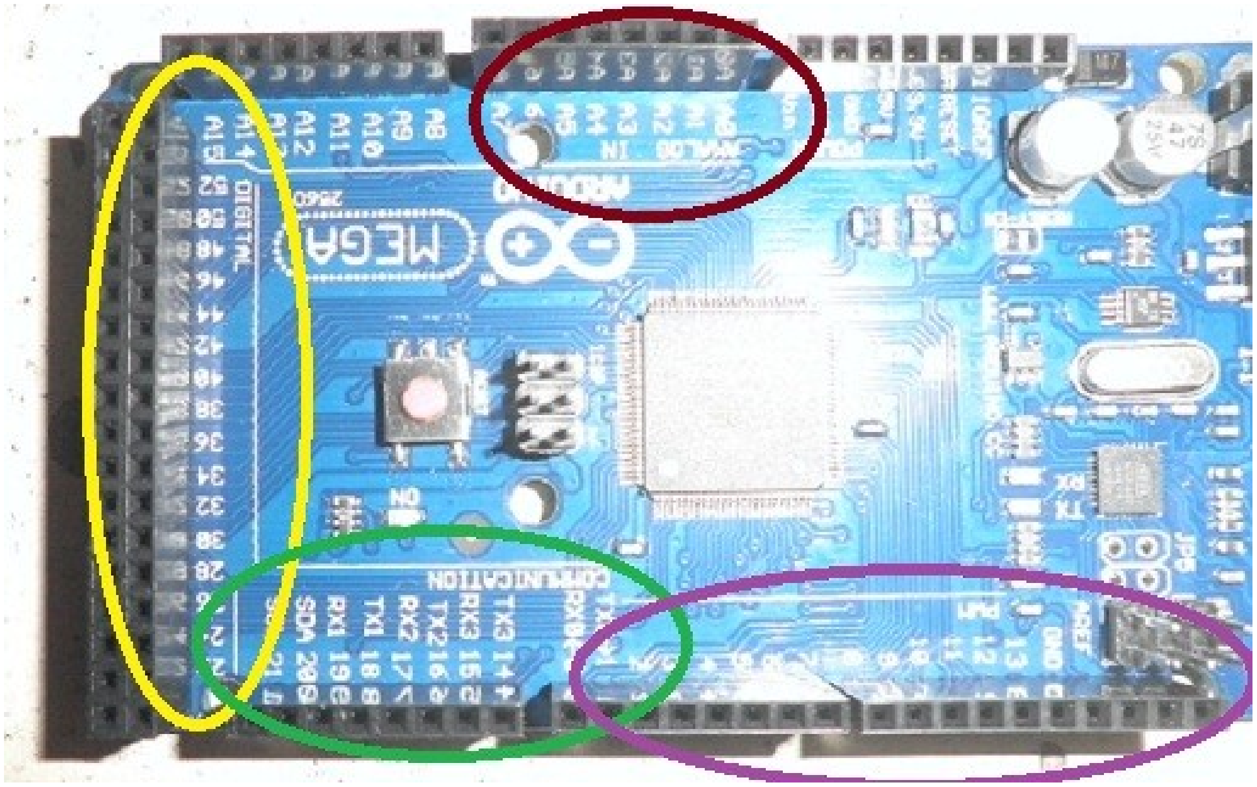
You can power the Arduino board in three ways. First, the external jack is used if you want to power the board independent of the PC. Second, the Arduino board can be powered via a USB cable connected to the Arduino USB port, and the other end connected to a PC or laptop. When you buy an Arduino board, it usually comes with a USB cable which you use to connect to a PC when you are uploading sketches (programs) to your board.

If you are going to use the external jack, you can use input voltage of 7-12 volts. It is recommended you use at least 9volts to make sure the board will function properly. On the upper right portion circled in red is the power section of the board itself. On any Arduino board, the different sections of the controller which can interact with the outside world are all labelled. You can look at your own board to confirm.

In this example, the word "power" is printed on the board itself. You can also see the ground(GND), 5V and 3.3Vtest points or jumpering point, Vref, etc... The terminal block to the right next to the labels(the plastic socket with many holes in it) is where you connect if you want to tap any voltage(5V, 3.3V) or ground(GND). You do this when you want the Arduino board to power the external device you are controlling. Of course you can also independently power the device separate from the Arduino board. If your external device uses up larger currents or a different voltage from what is available from the board, better to use a separate power set up for the device.

The third and final way to power the Arduino is if you need to power the Arduino board from a battery. You can connect the battery positive to the terminal next to the Vin label in the power section. Then connect the battery negative to the terminal next to the GND(ground) label. That's it for the power section of the board.

Below is a picture showing the other sections of the board a beginner must familiarize and understand.



The section circled in brown on the top portion of the image is the analog input section. Look closely at your own Arduino board and locate the section with the label "Analog In." The number of inputs in the terminal block depends on the particular board you are using. As a pure beginner, do not be overly focused yet on understanding and mastering how to utilize the analog port. The basic way to initially understand and familiarize with Arduino is to know how to turn on and off external devices using digital signals. That will be touched on in the digital port section.

The terminal block area encircled in violet at the lower right portion of the image is the Pulse Width Modulation (PWM) port. This port is mostly utilized in applications where you need to control power delivered to an external device (e.g. motor) or even in communications. Again it is a topic not usually for beginners. After you have mastered the digital port then you can begin baby steps in the use of the PWM port.

The section inside the green circle, middle bottom of the image, is the communication port. The Arduino can communicate with a PC, another Arduino or even another microcontroller. To do that one uses the Communication port. As you gain more experience and expertise, you can start trying simple communication programs with your PC or another controller.

The last section encircled in yellow, left most circle on the image) is the digital input/output port. This is what beginners must master first before moving on to more complex project using the other ports. The digital terminals can be configured as input or output as will be discussed in the software chapters that will follow. This is the port you will use for simple control of external devices which is what microcontrollers (including Arduino) are made for.

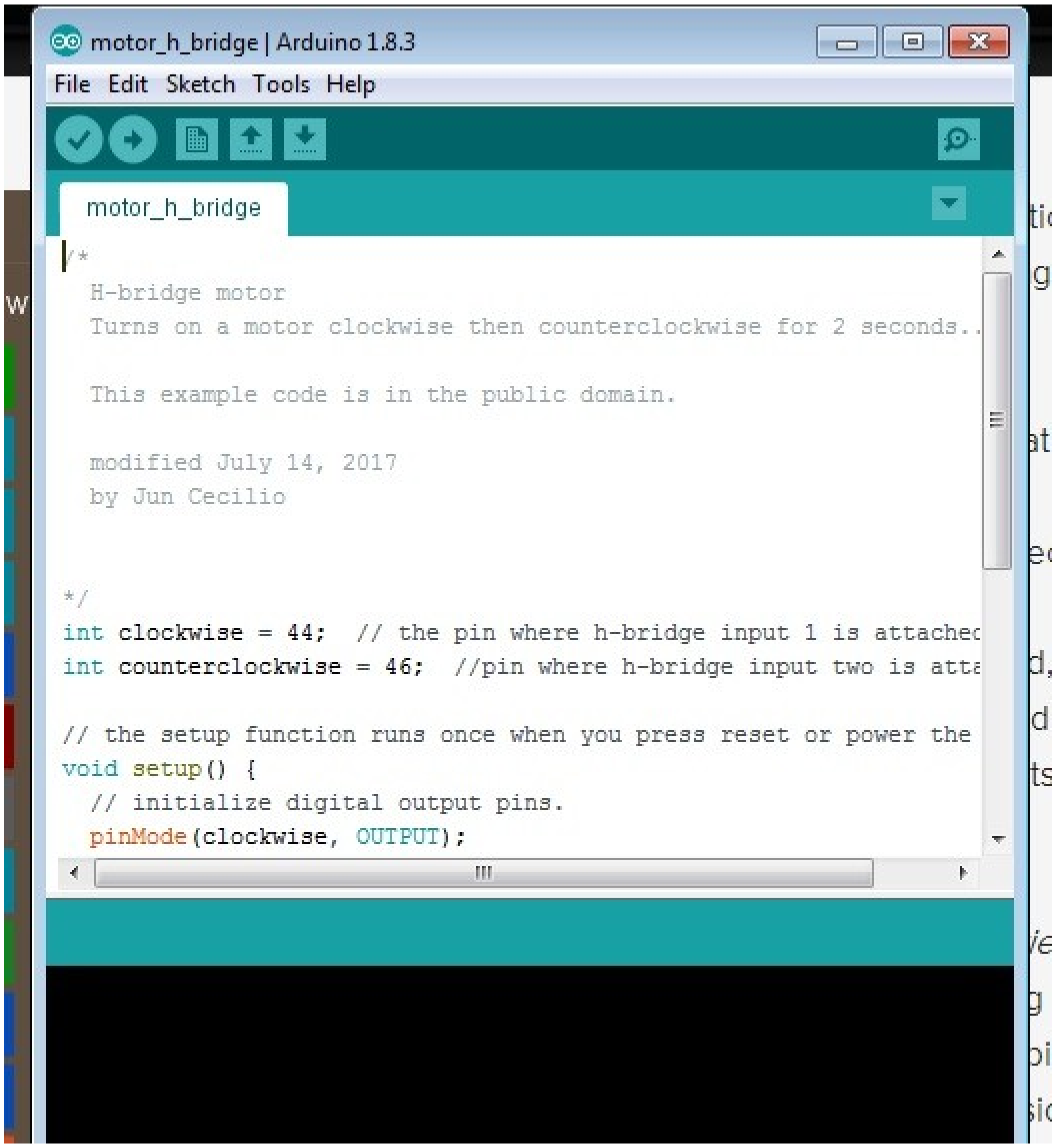
That's it! You are finished with the hardware portion for beginners. Quick as a ninja isn't it? And that is how this ebook is presented as you will find out in the rest of this ebook Now you are primed for the next section which will expose you to the software side of Arduino.

Chapter Two: The Software

Stealth and Camouflage. "As the ninja is a master of stealth and camouflage, so will the software provide you with the stealth to make your controller perform amazing tasks via instructions camouflaged in deceptively simple codes."

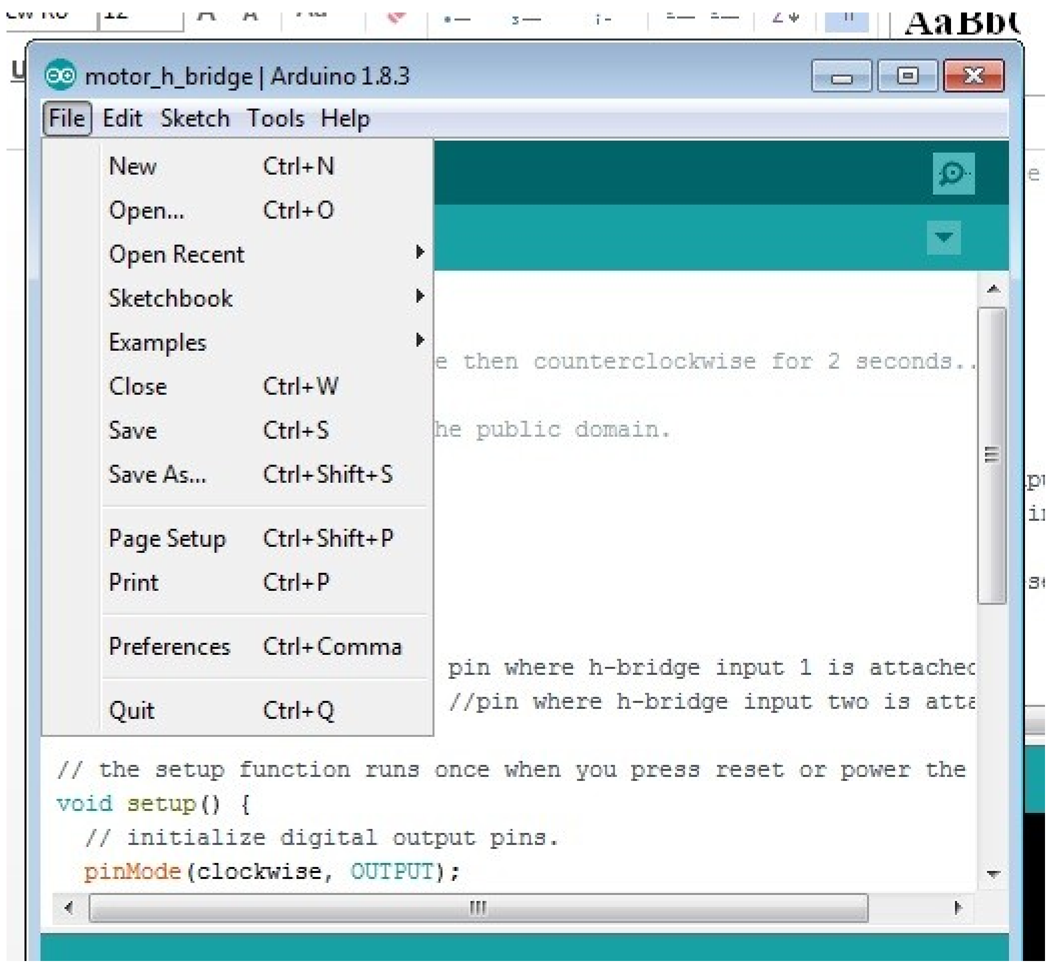
The hardware is what people see but it camouflages the software that provides the stealth that makes the hardware perform amazing functions. Be patient in looking for the simplest software solutions as usually they are the best. Now the first thing you need to do is to download the Arduino Software IDE. Go to the official Arduino website (the one with arduino.cc when you google it). Go to the "software" menu on top and download the Arduino IDE. Be careful not to choose the WEB IDE version as you cannot code offline if that is what you downloaded. Just follow the step by step instruction there to download the correct version for your PC operating system. When you begin the installation after downloading, at some point you will be prompted to install drivers for your PC. Choose yes or the affirmative to avoid any trouble once you connect your Arduino board via USB cable, to the USB port of your PC. To avoid any mistake, carefully read the options the download program gives out before clicking anything

If everything goes well, after finishing installation you will see the light blue Arduino icon on the programs list of your computer. Click it once to launch the Arduino IDE. You will see something like the picture below on the screen of your computer. As of this writing, version 1.8.4 is the latest Arduino IDE version. The next discussion involves the menu items in the IDE beginners will most often use. As you finish this ebook, you will gain the ability and agility to study the other more advanced menu items even on your own, which is also a goal of this book.



```
/*
  H-bridge motor
  Turns on a motor clockwise then counterclockwise for 2 seconds..

  This example code is in the public domain.

  modified July 14, 2017
  by Jun Cecilio


*/
int clockwise = 44;  // the pin where h-bridge input 1 is attachec
int counterclockwise = 46;  //pin where h-bridge input two is atta

// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital output pins.
  pinMode(clockwise, OUTPUT);
```

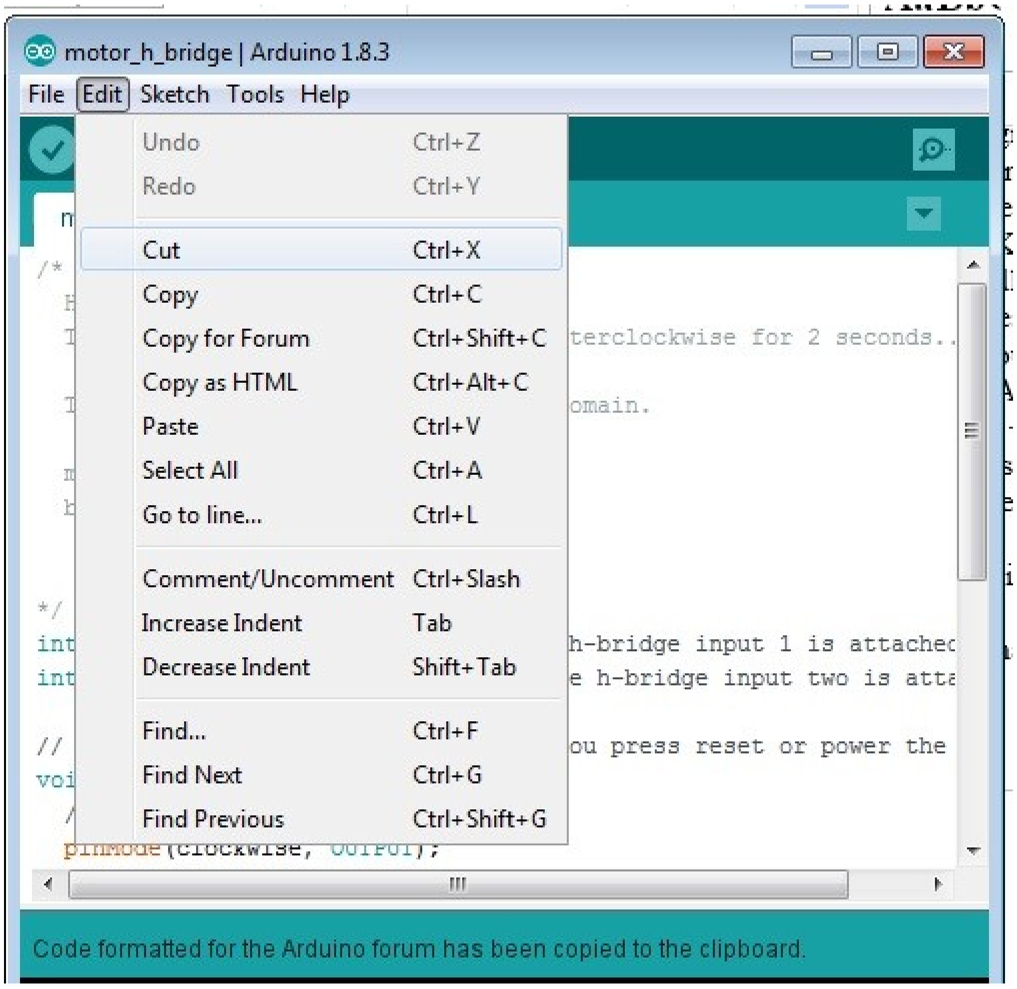
The first menu on the top is the FILE menu. Click it once and you will see the submenus under File. They work very similar to Microsoft applications. So if you are familiar with them, then Arduino menus will be a breeze for you. But we will still discuss the features you will most use when making sketches.



```
e then counterclockwise for 2 seconds..
he public domain.
 pin where h-bridge input 1 is attachec
 //pin where h-bridge input two is atta
// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital output pins.
  pinMode(clockwise, OUTPUT);
```

The **New** (CTRL+N is the shortcut key) submenu opens a new file screen. It will be a blank semi-formatted template where you can start your sketch (program). In Arduino speak, programs are called sketches so do not be confused with the term. **Open** (CTRL+O) shortcut key) will let you open any previous sketch you may have made. **Open Recent** will list the recent sketches you opened which you may want to reopen. **Sketchbook** will show you the sketches you have already created and saved. **Examples** will list the pre-installed sketches by Arduino programmers which you may want to use as references or as templates also. When you have time, you may want to browse the examples. They are a good source of snatches of codes you can reuse for your sketches. **Close** (CTRL+W) will close the current sketch or window you are working on. **Save** (CTRL+S) will save the current sketch you are working on. **Save As** (CTRL+SHIFT+S) will let you save under a different file name the current sketch you are working on. **Page Setup** (CTRL+SHIFT+P) lets you format the page in portrait or landscape. **Print** (CTRL+P) prints your sketch. **Preferences** (CTRL+COMMA) lets you tinker with settings of your Arduino IDE. **Quit** (CTRL+Q) closes the IDE and your sketch.

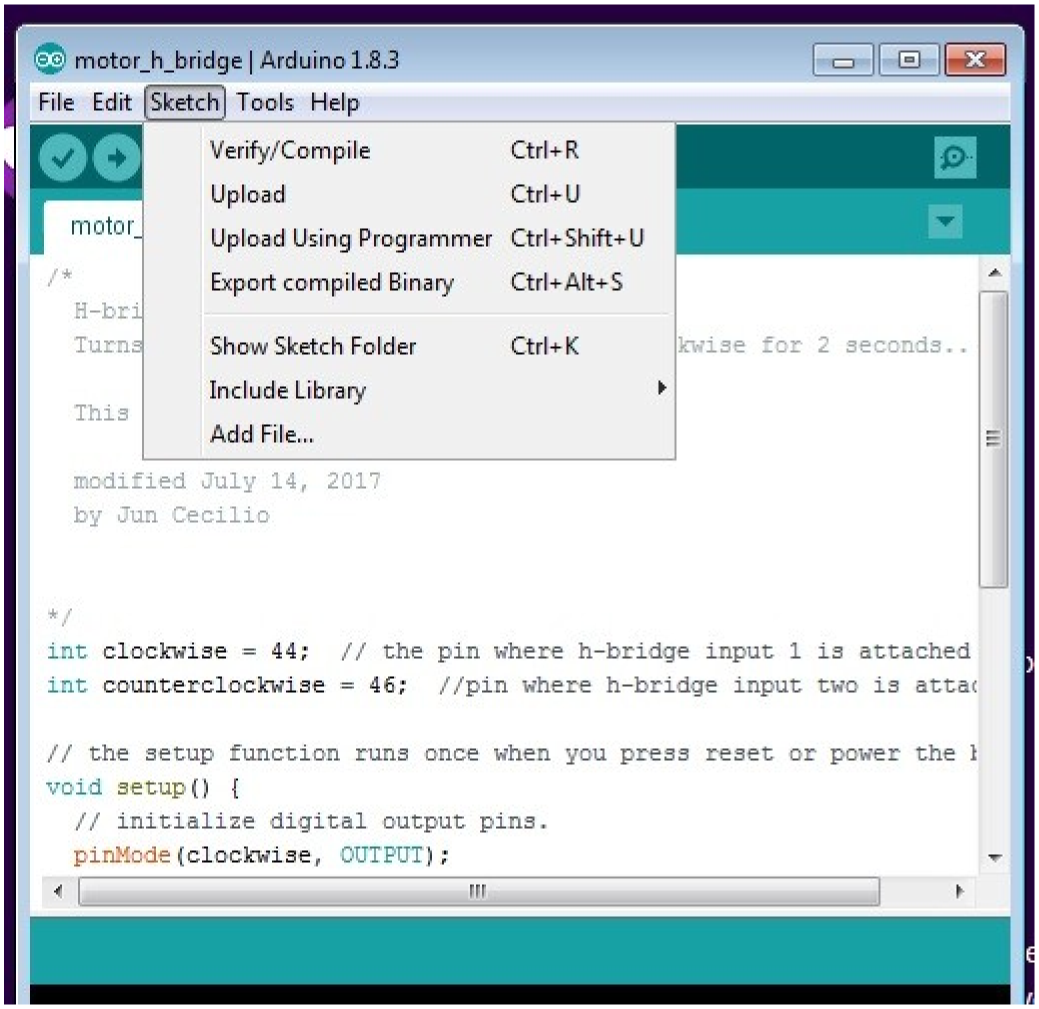
Clicking the **Edit** menu pulls down several submenus in it.



Code formatted for the Arduino forum has been copied to the clipboard.

Undo (CTRL+Z) lets you undo any error in typing or editing you may have made. If you change your mind and wants to restore what you encoded, **Redo** (CTRL+Y) is what you click. When you want to transfer sections of your codes or even entire codes to another sketch or other applications, you use the **Cut** (CTRL+X) submenu. Just highlight first the sections you want to transfer. However if you just want to copy the codes, use the **Copy** (CTRL+C) submenu. If you subscribe to the Arduino forum, and you want to post your code, use **Copy to Forum** (CTRL+SHIFT+C). If you want to post to a website, **Copy as HTML** (CTRL+ALT+C) is what you use. When you are already inside the other Arduino sketch screen or another application where you want to place your code use **Paste** (CTRL+V) and you will see the codes appear. If it is a different application (not another Arduino sketch), better to use CTRL+V. **Select All** (CTRL + A) highlights all areas of code for copying or transferring or even deleting. **Go to line** (CTRL+L) puts your cursor to the specific line of code you want to go to. **Comment/Uncomment** (CTRL+SLASH) lets you convert a line in your code into comment. Or when you intend to make a comment, this automatically places double slash. More on this when you go to the coding/programming section of this ebook. **Increase Indent** (TAB)/**Decrease Indent** (SHIFT+TAB) lets you indent or unindent while coding your sketch. **Find** (CTRL+F) helps you look for particular letters or texts in your code which you may want to replace. **Find Next** (CTRL+G)

Next to click is the **Sketch** menu and see something like the one below.


motor_h_bridge | Arduino 1.8.3
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
motor_

```
/*
  H-bri
  Turns                                               kwise for 2 seconds..

  This

  modified July 14, 2017
  by Jun Cecilio


*/
int clockwise = 44;  // the pin where h-bridge input 1 is attached
int counterclockwise = 46;  //pin where h-bridge input two is attac

// the setup function runs once when you press reset or power the b
void setup() {
  // initialize digital output pins.
  pinMode(clockwise, OUTPUT);
```

Verify/Compile (CTRL+R) checks the syntax of your code and compiles it. If there are errors you will be prompted to correct it. **Upload** (CTRL+U) will upload your current sketch to your ARDUINO board.

Below the menus you will see some icons. These are short cuts to some menu items previously described. The check symbol verifies your sketch for syntax errors. The right arrow uploads your sketch to the board. The document symbol opens a new template for a new sketch. The up arrow will let you choose and open a previously saved sketch. Last, the down arrow will save your current sketch.

There again!. You are finished with the Arduino IDE interface. Very quick like a ninja right? This prevents you from getting too bored and losing interest. Now to the heart of this book - the project making and programming proper.

Chapter Three: Making Projects

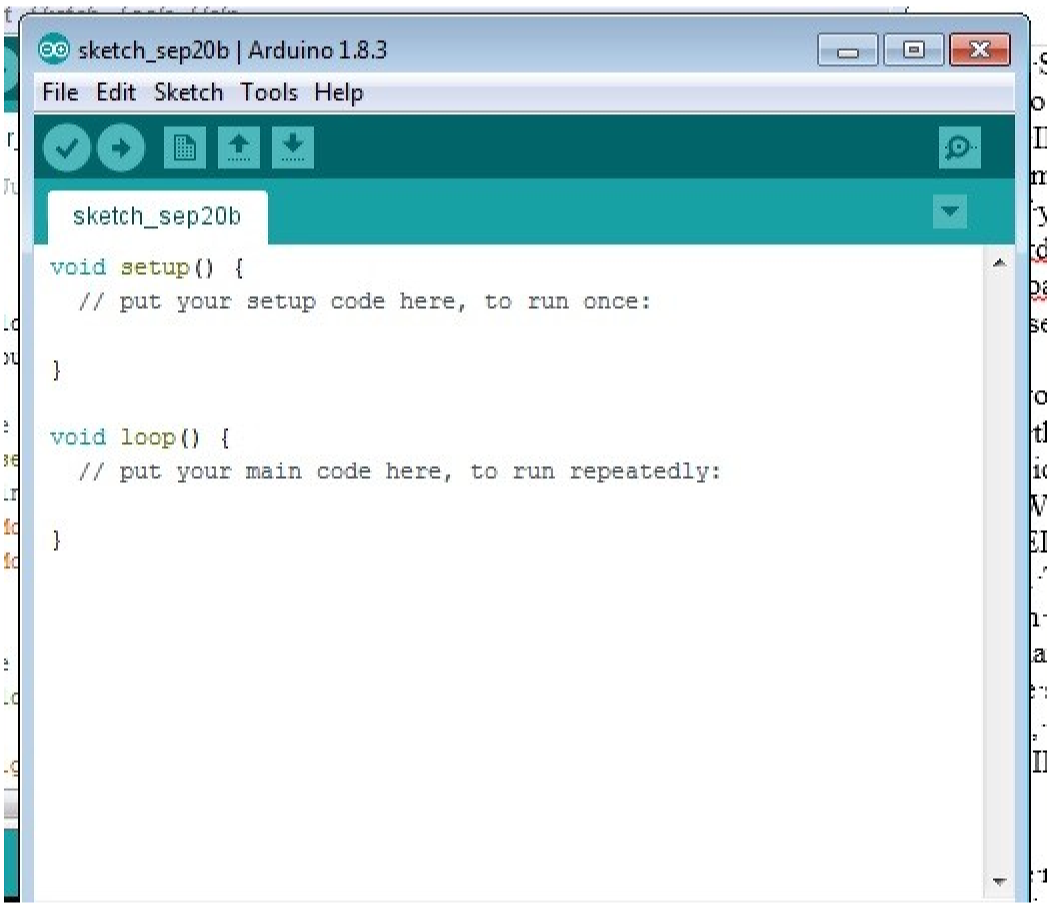
Agility and Strategy. "As the ninja is renowned for his agile and brilliant strategy in the performance of his mission, so now is the time for you to exhibit agility and brilliance in planning and making projects.".

More than at any point in your journey to learning Arduino as a beginner, this is where those ninja character must externalize in you. This is where you put into action your preparations from the two previous chapters. You already have the basic tools. What you need to learn a little bit is the code syntax. And then you can make simple projects which can be building blocks for more complex applications. But first you have to check to make sure that your new Arduino board will work. Launch first the Arduino IDE by clicking its icon in your list of programs. Then, using the USB cable that came with your board, connect it to your PC USB port. The Arduino IDE should automatically detect your board. If not, go to TOOLS menu and click the SERIAL MONITOR submenu. The IDE will try to check if there is a device connected to one of the serial COM port of your PC or laptop. You will see a message on the lower portion of the IDE if it does not detect your board. You may need to click the BOARD submenu in the TOOLS menu and manually select the particular board you are using (e.g. Arduino Genuino UNO). If you bought an Arduino board clone, make sure the installed bootloader program is the official Arduino bootloader for your particular board. Some clone board suppliers install their own bootloader program incompatible with the Arduino IDE. If that is the case you may have to use an Arduino device programmer to install the proper bootloader.

If everything went fine upon first connecting your board to the PC, you will see an LED on the board blink a few times and then settle. Now you are ready to upload the first sketch to your Arduino board. Click the following in sequence: File>Examples>01Basics>Blink. A new window will open with the Blink sketch on it. Click Sketch>Upload. Wait a few moments while the sketch is first verified/compiled. Then you will see an on board LED blink a few times. After that you will see a message on the IDE that says "Done uploading". There you are. You just uploaded a sketch to your board. You will now see a blinking built in LED in your board.

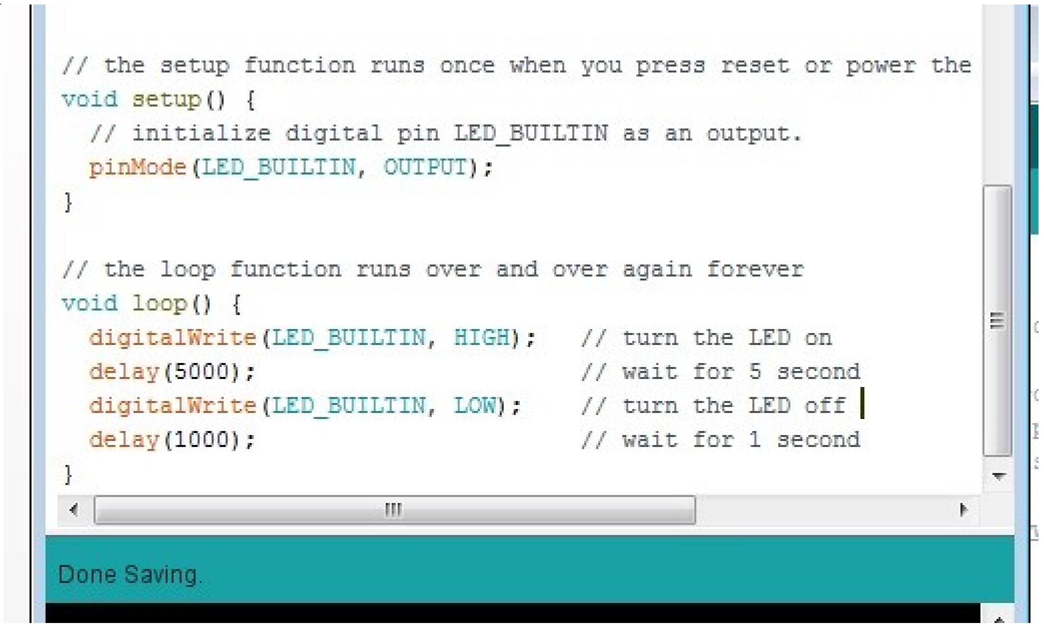
Now you are ready to tackle sketches. Always remember that an Arduino is a microcontroller. Its purpose is to control other devices. As such the simple sketches you will learn here and upload to your Arduino are meant to control external devices (sensors, actuators, signal light).

First, to open a new window to type your sketch, click in sequence File>New or use CTRL+N. You will see a new window pop up like the one below.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

That is a template for making your own sketch. The set up section is where you state your initializations like which input/output pins you will use for control. The loop section is where you place your logical sequences that will loop any number of times you make it in your sketch. Now type the codes in the picture below in the void set up and void loop sections of your template.



```
// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
  delay(5000);                       // wait for 5 second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
  delay(1000);                       // wait for 1 second
}
```

Done Saving.

Being a beginner, you will need some explanation of the above codes. The double slash above void setup and void loop lets you write one line comments which serves as reminders about the purpose of the code. It is a good practice to make plenty of comments throughout your codes so that when you revisit an old sketch, you will be reminded on how you developed it and the purpose of a specific line(s) of code(s).

Now back to void setup. Below it you will see a comment that the onboard built in LED will be used as output. The next line is the actual code. **pinMode** tells how the pin (built in LED) will be used- in this case as an output. In the void set up section, notice also the curly brackets right after void set up() and after the pinMode code line. Make sure every time that all your lines of code for void set up falls inside the curly brackets. That is just how Arduino software syntax is. When you forget this, during verification/compiling you will receive error message that can assist you to pinpoint this error. Also, the lines of code under void set up must be indented relative to void set up (). Then finish each line of code with a semicolon. That is a requirement for Arduino syntax. Otherwise there will be a compiling error message about this. For this example, there is only one line of code under void set up.

Next is the void loop section. As mentioned before, your codes here will loop over and over again. There is a comment above the void loop reminding this. This time you have four lines of codes under void loop (). All of them indented relative to the void loop () line. Notice that each line of code also ends with a semicolon. **digitalWrite** command means you are sending logic high (high voltage) to pin LED_BUILTIN. This will light up the on board LED. In the next line of code, delay sets the length of time the LED will be on. In this case 5000 milliseconds or 5 seconds. Milliseconds is one of the timing ways allowed in Arduino software. There is a different way that you would learn as you become more competent in coding in Arduino. The next digitalWrite command sends logic low (low voltage, 0volts) to on board LED and will turn it off. The next line **delay(1000)** gives a command of 1000 milliseconds for the LED off time after which the loop will start again to the first digitalWrite command line. And so on. Notice that at each line there is a comment after each code. It is a good practice to place as many comments within your codes, not necessarily at every line of code, but at important points along your program to make it understandable even to those who did not do the coding. Even you as the coder may not fully remember every logic or tricks you incorporated in your codes as time goes on. You will be very thankful to yourself once you revisit old codes you made that you would want to recycle in other applications you are going to make.

If your Arduino board is not yet connected to the PC, connect it now via USB cable. Click the check icon in the IDE to verify your code. Then click the right arrow to upload the sketch to your Arduino. Afterwards you will see a message on the message bar in the lower portion of the IDE saying successful upload. Then you will see the LED blink but this time the on time is longer than the off time. Practice changing the on and off time relative to each other in the sketch then upload to your Arduino after each change.

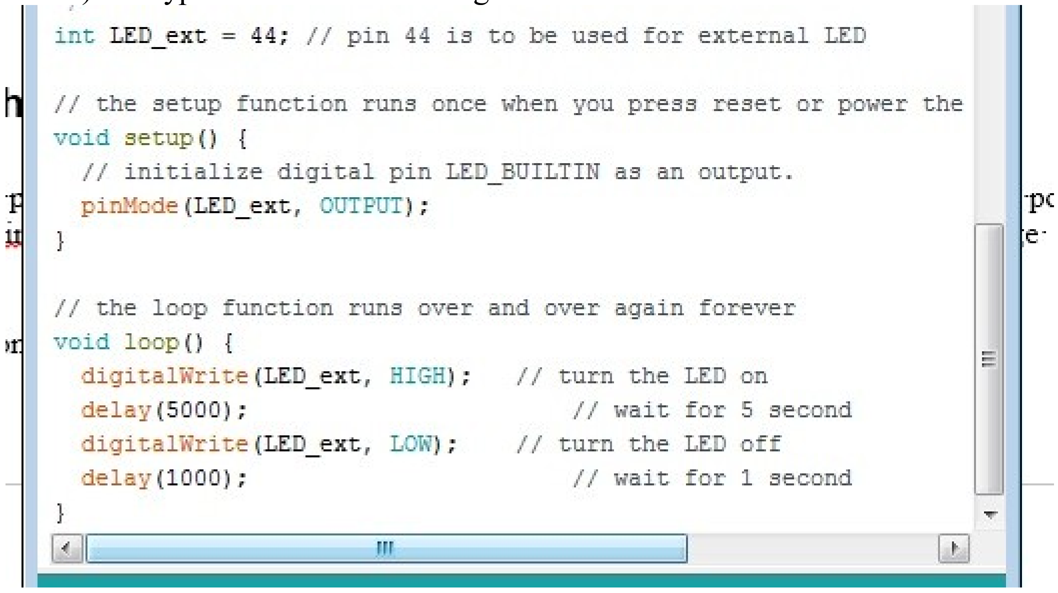
You have made your first sketch now. Congratulations! Now save your file first. Click File>Save As then type any filename you want for your first sketch. Click the Save button. In the next chapter, you will start controlling external devices via the digital input/output port.

###

Chapter Four: More Projects

Ruthless and Unorthodox. "As a ninja is ruthless and unorthodox in executing his missions, so you must be in executing projects."

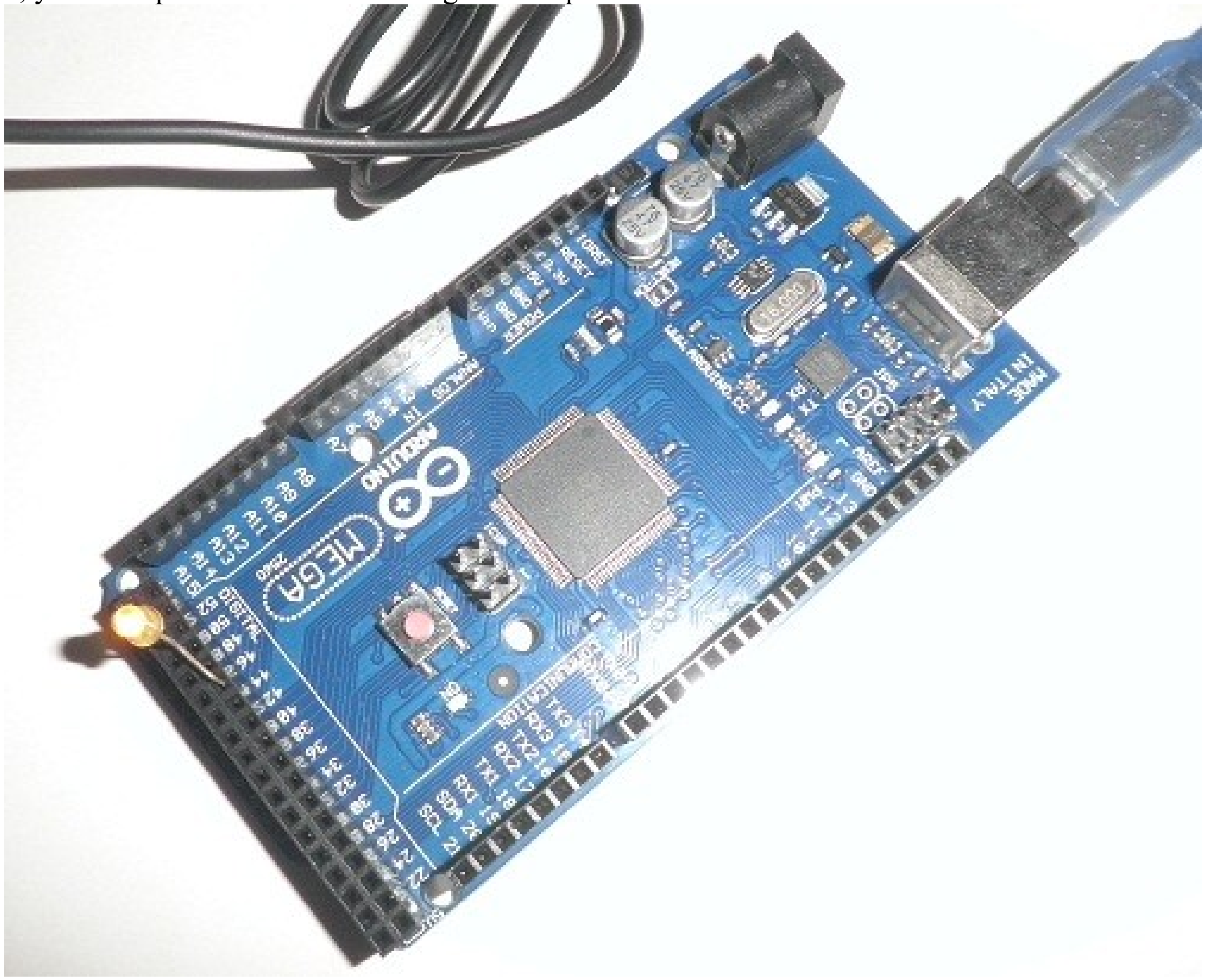
In the future, as you gain experience and skill, visualize yourself creating unusual projects and execute your ideas without hesitation. For this next project, you will start controlling an external device via the digital input/output port of your Arduino. Open a new sketch window first (CTRL+N) and type the codes in the image below.



```
int LED_ext = 44; // pin 44 is to be used for external LED

// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_ext, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_ext, HIGH);   // turn the LED on
  delay(5000);                       // wait for 5 second
  digitalWrite(LED_ext, LOW);    // turn the LED off
  delay(1000);                       // wait for 1 second
}
```

You would notice that the codes are very similar to that of your first sketch. What is different is that there is an initialization line of code above **void set up**. The line "int LED_ext = 44; " means pin 44 of the digital input/output port will be used to control an external LED. In this case pin 44 is part of the input/output port of Arduino 2560. Examine your Arduino board. If it is not Atmega2560, look for the block terminal port with printing of "digital input/output". Choose a terminal number you want to use as your output terminal and reflect that number on your sketch by replacing terminal 44 in the code. Click File>Save As and name your file "External LED". Click the verify icon to compile your sketch. Connect again your Arduino to your PC. Click the upload icon to load the sketch to your board. After that, disconnect first your board from the PC.

To check if your sketch works, you will need an LED. The super bright type of any color is recommended here but any other type is ok. The USB port can easily drive an LED. Whatever LED you use, you will find that one leg pin is longer than the other. The longer leg is the anode which should be connected to the signal source. The shorter leg (or cathode) is connected to ground. If there is no ground (GND) pin in your Arduino digital port, it will be difficult for the LED to reach both the output pin and the ground pin in the power section. You have to use two jumper wires to make the connections to your board. Any wires which you can wrap around the LED legs and insert to the output pin of the board will do. One wire you wrap around the anode of the LED. The other end you insert into the terminal number you will use (44 for Atmega 2560). The second wire you wrap around the cathode and the other end you insert into the ground terminal of the power section in your board. If there is a GND in the digital port of your board, your set up should look something like the picture below.

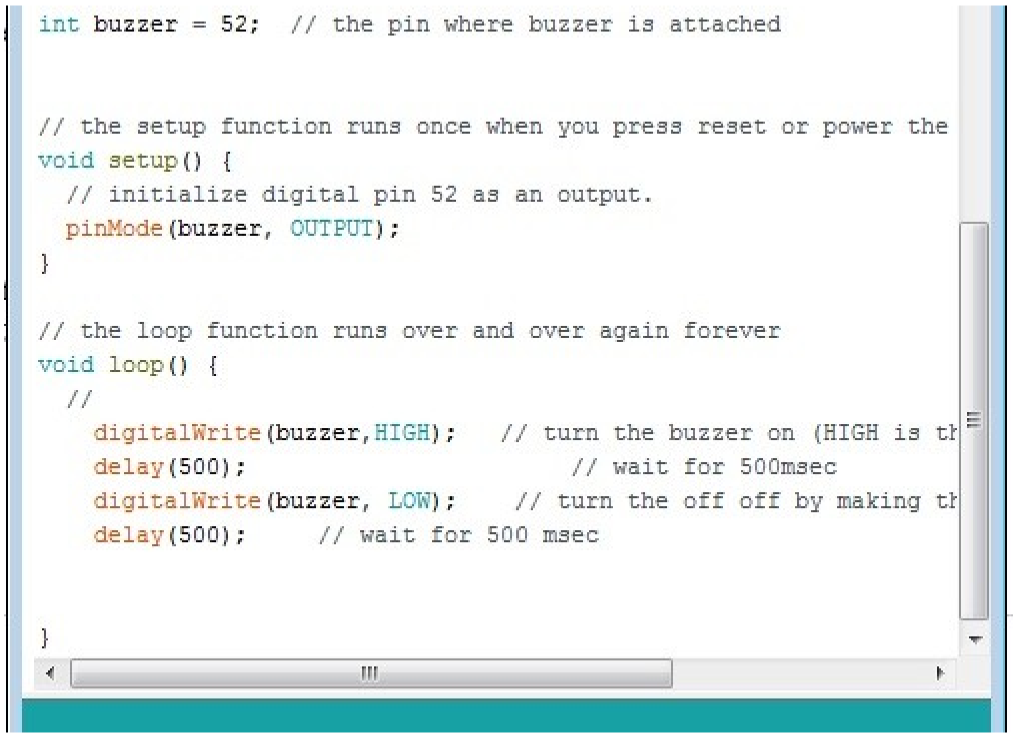


In the case of the Arduino 2560 board, there is a ground (GND) pin in the digital port itself. It is more convenient to use as it is nearer and no need to use jumper wires for the LED. You just have to bend the legs to fit in the port.

Now reconnect your board to the PC. As soon as your board is powered up by the PC USB port, the LED should light up for 5 seconds then turn off for 1 second repeatedly. Congratulations again! For the rest of this chapter you will be making sketches to control other types of devices via the digital port.

For the next project, you will control a piezoelectric buzzer. You need to obtain a piezo buzzer similar to any of the devices pictured below.

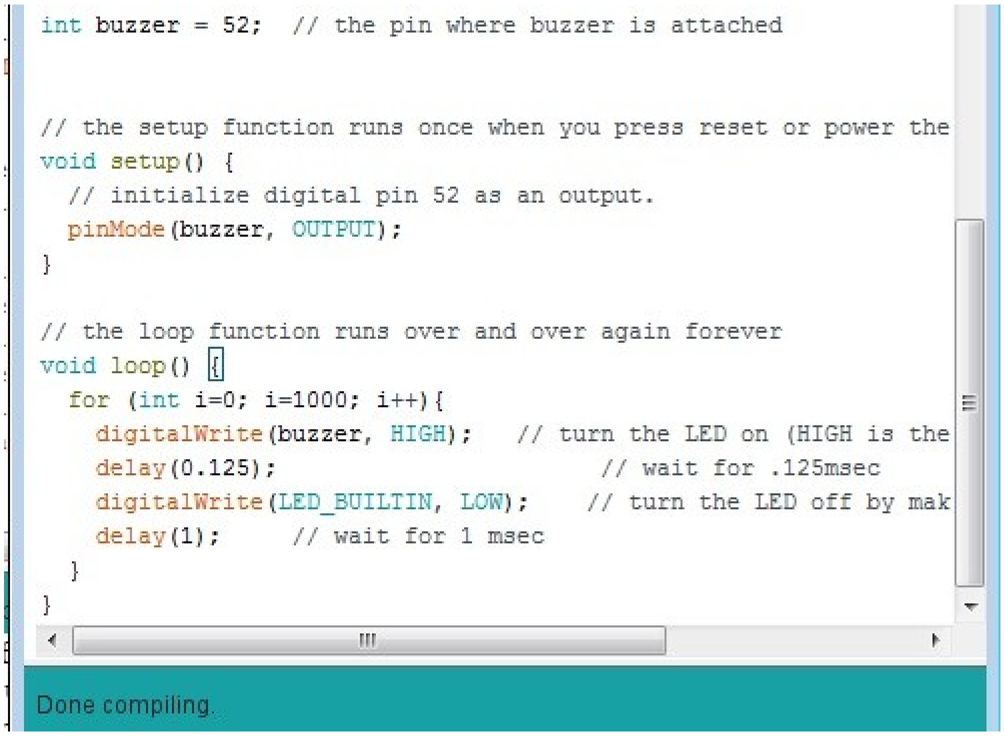
The one with the black casing is the preferred one since it emits stronger sound than the flat disc. These items can be purchased at any electronics shop or online. Now open again a new sketch to type the codes in the image below.



```
int buzzer = 52;  // the pin where buzzer is attached


// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital pin 52 as an output.
  pinMode(buzzer, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  //
    digitalWrite(buzzer,HIGH);   // turn the buzzer on (HIGH is th
    delay(500);                         // wait for 500msec
    digitalWrite(buzzer, LOW);    // turn the off off by making th
    delay(500);     // wait for 500 msec


}
```

The codes are self-explanatory now. It is similar enough to the previous sketch. A different pin for output is being used (52 instead of 44). You can try using a different output pin in your particular Arduino. Select any number in your digital input/output port and type it in your sketch. Verify and then save your sketch first. Connect your Arduino to the PC and upload the sketch to the board. Disconnect the Arduino after uploading. Now insert one of the wires of the piezo buzzer to the pin you chose in your sketch. The other wire of the device you insert to the ground pin of your board (whether on the digital port or power section). Reconnect your board to the PC. Now you will hear a tic toc or beeping sound as the buzzer is alternately powered and relaxed by the Arduino. When you become more proficient at programming in Arduino, you can actually make various tones or even music on the buzzer using your Arduino.

Next is a slightly more complex variation of the previous sketch with application on a buzzer again. Type the codes in a new sketch from the image below. Verify it and then save.



```
int buzzer = 52;  // the pin where buzzer is attached


// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital pin 52 as an output.
  pinMode(buzzer, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  for (int i=0; i=1000; i++){
    digitalWrite(buzzer, HIGH);   // turn the LED on (HIGH is the
    delay(0.125);                     // wait for .125msec
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by mak
    delay(1);     // wait for 1 msec
  }
}
```

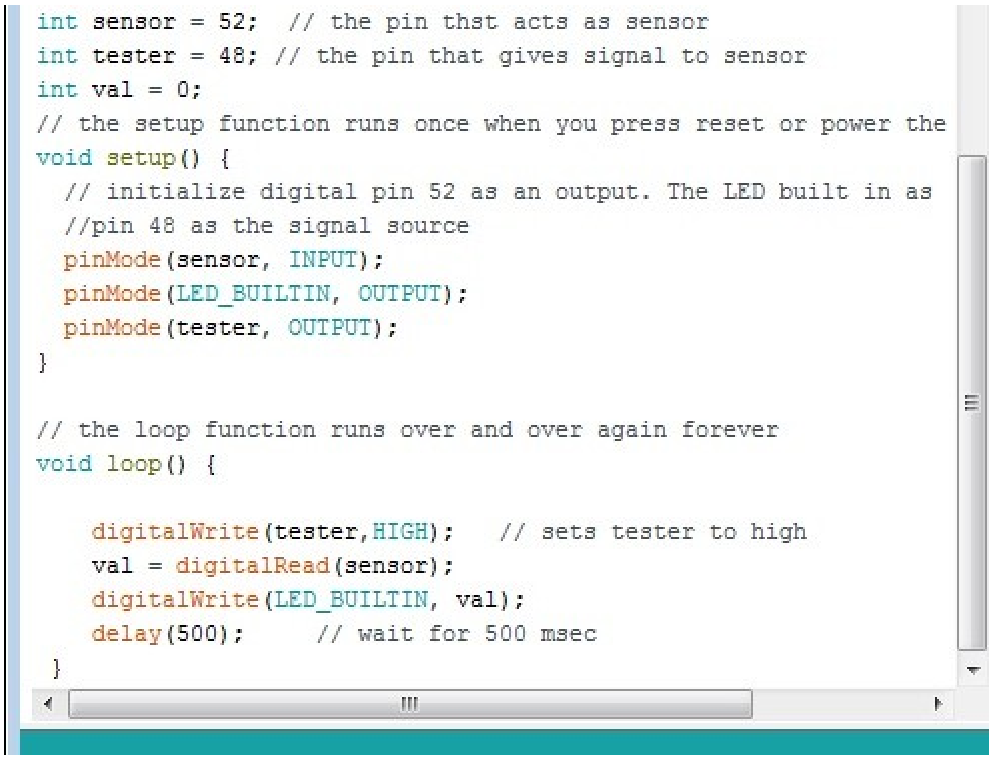
Done compiling.

Pin 52 of Arduino 2560 will still be used in this example. Again, use the pin number in your Arduino you used in the previous buzzer example. The void set up section is the same as before. The void loop section is now more challenging. A FOR loop statement is incorporated in the codes. Before the explanation on how the program works, examine the curly brackets first. There is the set of curly brackets after the void loop and at the last line of code. There is another set of curly brackets after the **for (int i=0...)** and at the second to the last line of code. The sub statements under the for loop must be enclosed by curly brackets. The same goes for any loop statement you use when you make more complex sketches. Notice also that the **for** is indented relative to the void loop line. The statements under the "for" condition is in turn indented relative to "for". You can nest several loop statements but do not forget the rules on indentation and curly brackets just discussed here. In the last chapter you will be pointed to some resources on how and where to place curly brackets and the syntax requirements for various type of logical loops.

Now to explain the codes in the sketch. The four statements under the "for" loop are executed 1000 times. There is an initial count of 0 (i=0) and incremented (i++) up to 1000 (i=1000). After that the whole loop is repeated again and again.

Connect now your Arduino board with the buzzer as previously connected to it. Upload the sketch to your board. When the upload is finished you will hear a steady buzzing sound coming from it. That indicates successful uploading of your sketch.

For the last several example sketch, all of them used the output port. For the next example, you will learn the use of the input port. Type the codes in the image below in a new sketch, and then verify and save with filename SENSOR.



```
int sensor = 52;  // the pin thst acts as sensor
int tester = 48; // the pin that gives signal to sensor
int val = 0;
// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital pin 52 as an output. The LED built in as
  //pin 48 as the signal source
  pinMode(sensor, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(tester, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {

    digitalWrite(tester,HIGH);   // sets tester to high
    val = digitalRead(sensor);
    digitalWrite(LED_BUILTIN, val);
    delay(500);     // wait for 500 msec
 }
```

You will need two pins in the digital port of your Arduino. For this example, pins 52 and 48 of the Atmega2560. Just choose two pins in your Arduino digital port if you use a different board. One will be used as the sensor and the other as the tester. See the initializations above the **void set up** in the codes above. The built in LED on the board will be used as the indicator of the sensor status. A variable "val" is initialized to "0" or low voltage.

In the **void set up** section, the sensor pin (52) is designated as the input. The tester pin (48) is designated as output. The LED_BUILTIN is also designated as an output.

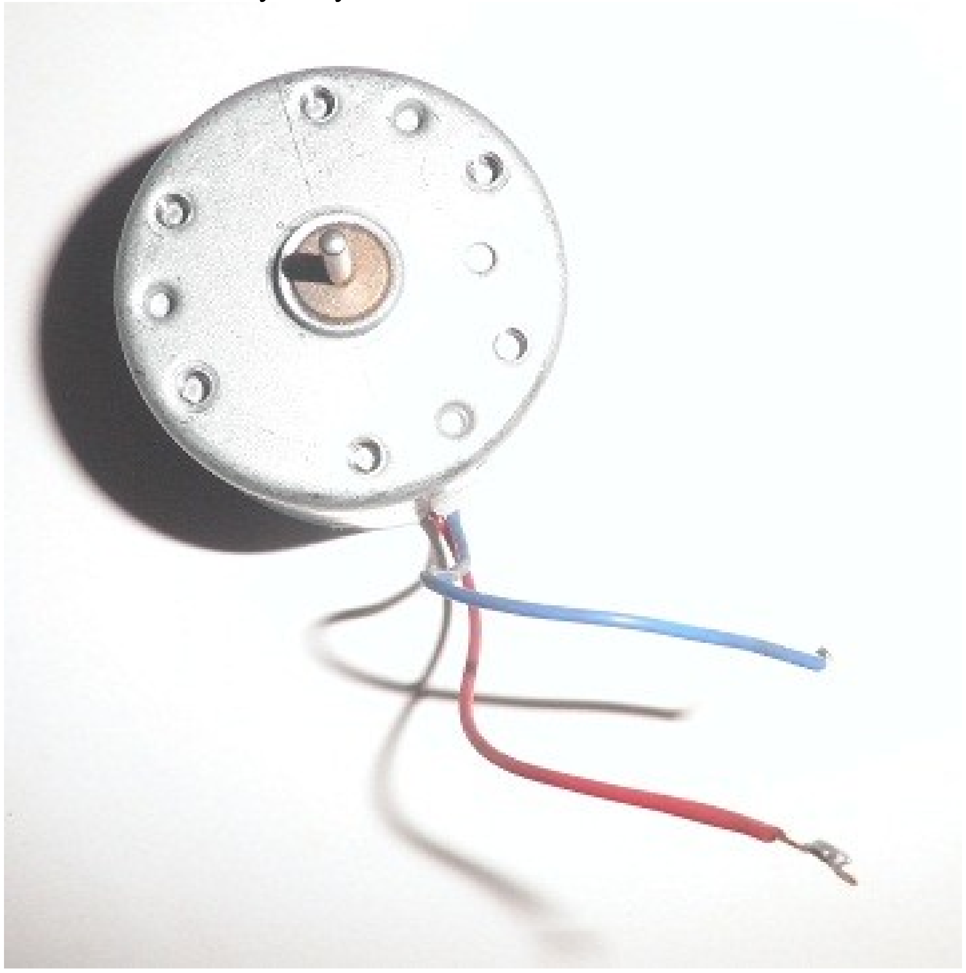
In the **void loop** section, the tester pin is brought high (using digitalWrite). Using digitalRead, the value, "val", of the sensor pin is read. On the third line of code, whatever the "val" of the sensor is written on the on-board LED (digitalWrite). The last line is just a delay, and then the loop repeats itself.

Upload the SENSOR sketch to your Arduino board. Disconnect the Arduino afterwards.

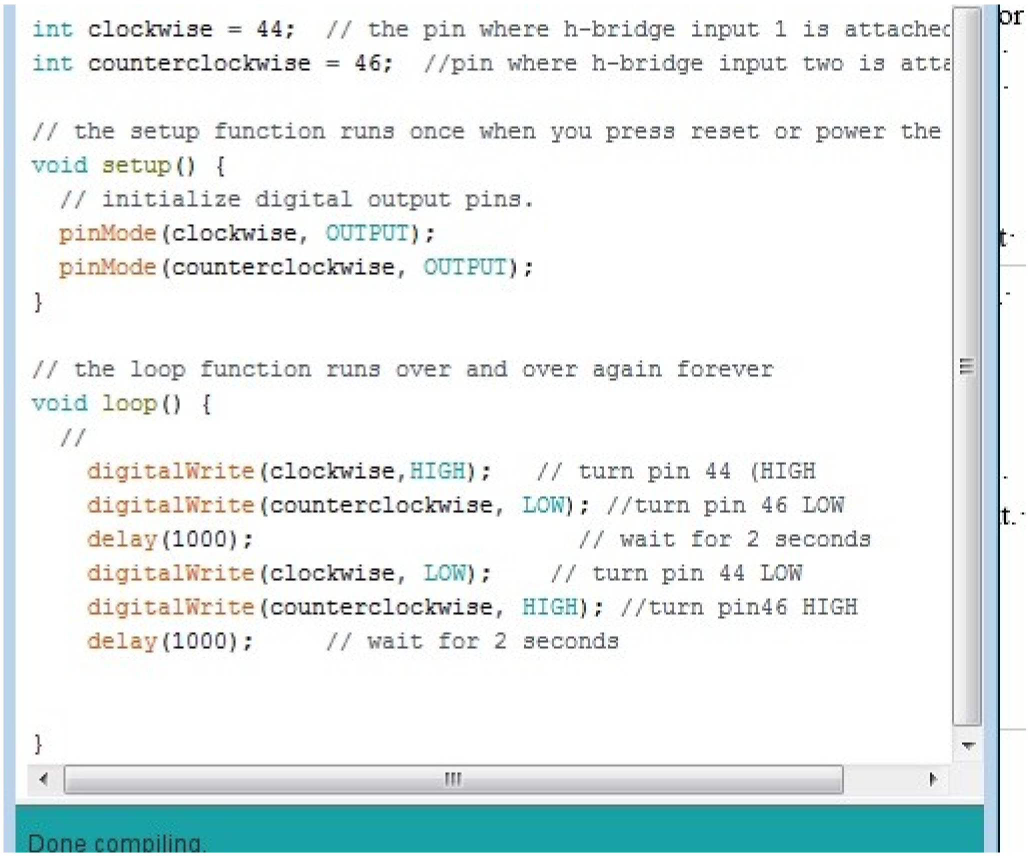
To test if the sketch/program works, you need two pieces of jumper wires. One wire you connect the ends to pins 48 and 52 (or whatever pin numbers you use in your Arduino). The other wire you insert also to pin 48 (tester pin) or whatever tester pin number you designated in your board. You must use a thin enough wire so that it can fit to pin 48 even with two wires in it. The other end of that wire you just leave hanging for the meantime.

To see if the sketch/program works, connect again your Arduino to your PC. As soon as the board is powered up, you should see the built in LED light up steadily. This is because the sensor pin, which is jumpered to tester pin is detecting a high voltage state, the tester pin being written high in the void loop section of the sketch. Now try and connect the free end of the second wire from the tester to the ground (GND) pin in the power section of the board. As soon as you do this, the built in LED will turn off. This is because you connected the tester pin to low voltage state (0volts). The tester being connected to the sensor pin, the sensor will sense the low value state. This will trigger the sketch to 'digitalWrite' a low value to the onboard LED which turns it off. As soon as you remove the connection to the ground pin in the power section, the LED will light up again.

Now you are ready for the last example in this book. This time you will control a small bidirectional motor. A note of caution first. The digital output port can only supply a small current, 20mA or even less. If the motor you choose to drive requires greater than this, you would need an H-bridge set up. More on the H-bridge circuit in the last chapter. For now, try to choose as small motor as possible. A vibrator motor from cellphones is ideal for this experiment. The type of motor pictured below is also suitable for testing without any load on it. An H-bridge circuit will be needed if you try to run it with a load.



Copy the codes in a new sketch from the image below.



```
int clockwise = 44;  // the pin where h-bridge input 1 is attachec
int counterclockwise = 46;  //pin where h-bridge input two is atta

// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital output pins.
  pinMode(clockwise, OUTPUT);
  pinMode(counterclockwise, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  //
    digitalWrite(clockwise,HIGH);   // turn pin 44 (HIGH
    digitalWrite(counterclockwise, LOW); //turn pin 46 LOW
    delay(1000);                       // wait for 2 seconds
    digitalWrite(clockwise, LOW);    // turn pin 44 LOW
    digitalWrite(counterclockwise, HIGH); //turn pin46 HIGH
    delay(1000);     // wait for 2 seconds


}
```

Done compiling.

Verify then save with filename MOTOR.

In the initialization prior to void set up, pins 44 and 48 of the digital port (Arduino 2560) will be used to control the motor. Choose your digital pins if you have a different board. What is important is you have two pins for clockwise and counterclockwise motor shaft rotation.

In the void set up codes, both pins will be designated as OUTPUT.

In the void loop section, first the clockwise pins is set high while counterclockwise pin is set low. Next a delay of 1 second is allowed. After that the clockwise pin is set low and counterclockwise is set high. There is a delay of 1 second again and then the loop repeats endlessly.

To prepare your motor for testing, solder wires to its two terminals if it does not have wires already. Insert the wires to the chosen pins in your board (44 and 46 for Atmega 2560).

Connect your Arduino to the PC and then upload the sketch to the board. As soon as the sketch is loaded, your motor should turn clockwise (or counterclockwise) for 1 second then counterclockwise (or clockwise) for 1 second. This will repeat over and over again.

At last you have finished the beginner's ninja-quick tutorial on Arduino. The last chapter will try to connect it all together and suggest the next step for you, the beginner, to tackle.

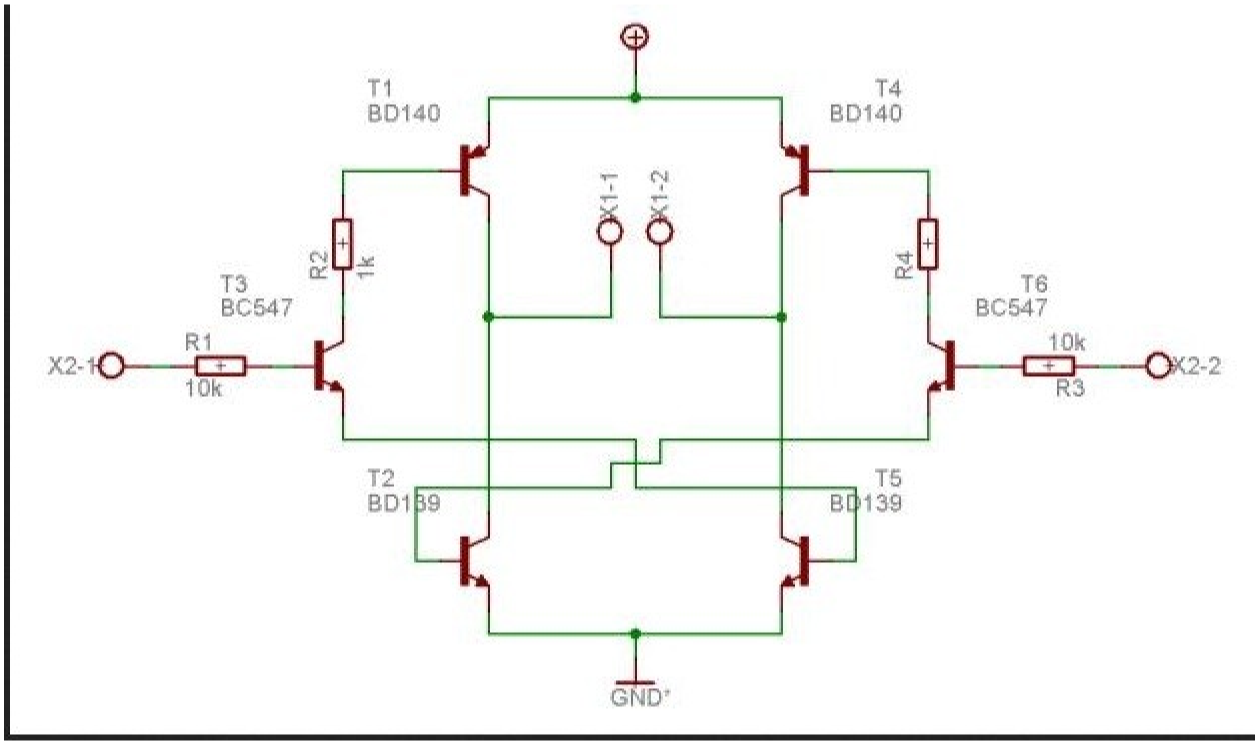
Chapter Five: Putting It All Together and Next Step

Calm and Ingenious. "The ninja is both calm in spirit and ingenious in the use of his skills. This is now the part where you must also be calm in spirit as you find ingenious ways to put into action your learnings in this book."

Now that you have finished all the lesson and example sketches, what next? Before you tackle any project on your own, it is important to recap what you have learned so far. Basically you have been equipped with the foundation needed to use your Arduino microcontroller to control other devices. Just like a ninja, you have the knowledge and skill to go on missions now. Be brave like a ninja and do projects on your own. A good approach would be to tackle projects with increasing complexity. Start with the suggested project in the next paragraphs.

So what have you learned so far? You have actually sort of mastered the use of the digital port of the Arduino. Depending on your board, you would have more or less number of pins compared to the Arduino Atmega 2560 used as an example here. You now know how to send signals via output pins. Conversely you can detect/read input signal via the same port. You can already do a lot of things using those two concepts. You also know of delay timing. You need to judiciously put delays between commands for the external device being controlled to have time to react. You can use LEDs attached to the digital port as signals when a certain command is being executed. And much more. Just use your creativity like a ninja.

As a suggested follow up project, you can do an Arduino-controllled robot car or carbot. In this project, it is best if you can obtain a pre-built robot car platform, where you will only need to connect the Arduino controller board, If you have the know-how to hack an old RC car, it is a good alternative. If you have to assemble your platform from scratch, you will also need an H-bridge circuit to drive the motors. The circuit looks like the diagram below.



The motor connection is made at the open ends near the middle of the diagram (x1-1, x1-2). The Arduino port pins are connected to the open wires at the sides (x2-1, x2-2). You need a separate power source for this. See the + sign at the top and GND at the bottom where the batteries are connected. This diagram is from several you can find over the internet. Wikipedia has an article on how the H bridge operates.

Once you have the carbot platform ready, you can program your Arduino to make a set movement. Example: forward for 1 second, turn right 1 second, turn right again for 1 second. It could be as simple as that. Or add a sensor using optical or infrared LED/sensor pair. Then you can make it stop when there is a barrier in front. Or even follow a black line or white line on the floor (line following). You can pretty much do all the suggested carbot abilities using digitalWrite and digitalRead commands only.

If you need more complex commands like other logical loops (e.g. if-else, case statements), you can look at the other preinstalled sketches in your Arduino files. Click File>Examples and check the sketches listed there to see how the commands are implemented. Or you can go to the official Arduino website and check the References there.

As suggested before, start with simpler sketches with simpler commands. Tackle bigger projects as you gain experience and skill. Remember, if you want to use other features of your Arduino board (e.g. PWM, RX/TX serial communication), you first have to study the basic concepts involved (what is PWM, serial communication protocol basics). This is so that you will understand the example codes and the syntax involved.

It is all up to you now. Like a ninja, you have to hone and improve your skill/knowledge by constant practice and and doing projects (going on missions). Good luck on your road to being an Arduino ninja master.

The End

About the Author

Jun Cecilio currently trades in the stock market using personally developed computer programs to identify profitable trades. In his spare time he tinkers with projects of interest to him mostly in the following areas: renewable energy/energy harvesting, electronics, hobby robotics. This is his first attempt at publishing an ebook on microcontrollers. Some of his projects can be viewed at http://www.pinoyhobbytech.weebly.com.

Connect with Jun Cecilio

Thank you from the bottom of my heart for reading my book. For any feedback, I can be reached via email at fceciliojr@yahoo.com. Or if you have spare time, visit my website at http://www.pinoyhobbytech.weebly.com