STEVE M. ECKERT

ArduiUno PROGRAMMING

Programmable Circuit Boards and Coding Programs of Your Choice to Run on Your Computer

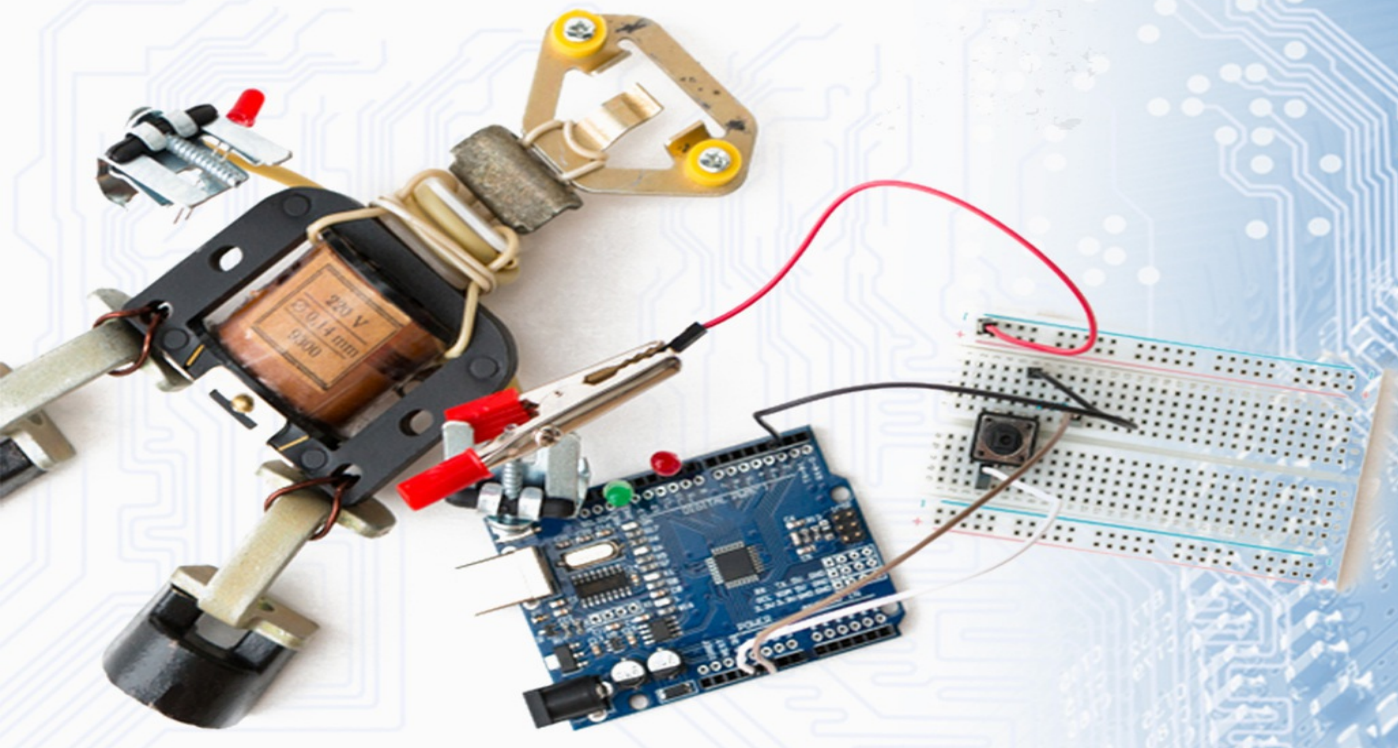

TABLE OF CONTENTS

CHAPTER ONE: YOUR FIRST INTRODUCTION TO ARDUINO

Wow! Your constant ramblings finally led you to this place, to Arduino? That's pretty awesome! This is a sure sign that you are an interesting fellow. Welcome to Arduino, the handy tool of automation. This book is assuming you don't know jack air about Arduino, so we are going to start from the basics of the basics.

Before we proceed, what is Arduino?

Arduino is a small board. A microcontroller board. It contains a universal serial bus (USB) plug (you would be surprised at the huge number of people in the world who don't know the full meaning of USB!). This USB plug is connected to your computer or any other electronics such as speakers, microphones, relays, sensors, etc. The source of the power for the Arduino could be through an external power supply, a 9volts battery or through the USB connection to the computer or electronics. The Arduino can be controlled directly from the control or programmed by the computer and then disconnected to be used independently. A sensor could be attached to the Arduino board for example, and then the Arduino is controlled through the sensor.

Have you ever seen a remotely controlled garage door before? I mean, the driver gets to the door and presses a button on his remote, and the garage door opens. That is pretty awesome and you know it. So, what happens when the driver pushes the button in his remote control? The remote control sends a signal to the sensor in the Arduino, the sensor now sends an electrical signal to the pins in the Arduino board. These pins now transmit the electric signals to the motors which control the door. Then the garage door opens. Arduino is like an interface between the mechanical/electrometrical parts of a device, and its electronic controls. The electronic controls send instructions to the

Arduino board, and the Arduino board sends the instructions to the mechanical parts.

This example of the garage door given above is the typical mode of operation of Arduino. We will learn more about how Arduino works in subsequent sections. But for the meantime, I’m sure you have seen that Arduino is not difficult. At least it’s mode of operation is quite straightforward.

We have said before that the Arduino is a microcontroller board. Something that is quite awesome about Arduino is the fact that the board design is open source. What this means is that anyone can make boards that are compatible with Arduino. As long as they can make the boards anyways.

Did I hear you ask: if Arduino is a board, then what do I mean by boards that are compatible with Arduino? Does it mean that there is something else called Arduino which the boards need to be compatible with? We will talk about it in due time. Arduino is not just the board; Arduino is also the software which controls the board. This means that any other person who comes up and designs a microcontroller board must design it in such a way that this board be compatible with the Arduino software.

Who this book is for?

This book is for the beginner. It is the basics of the basics. For the beginner who has never come across Arduino before. It is for the beginner who has never come across programming before. In fact, it is for the beginner who has little to zero foundation in science. This book is a foundation book for the beginner and it is the springboard from which the beginner begins to get a grasp of Arduino. Inasmuch as this book is for the beginner, this does not mean that the advanced learner and the professional cannot find use with this book. Because it guides the reader again through the basic concepts of Arduino, the professional might find it useful as a refresher material for some concepts which he may find that he has become rusty at.

The bottom line is that to use this book, you do not a previous experience in programming. You do not also need technical experience. This book will build you up from scratch, or brush you up, depending on your needs.

In going through this book, depending on what you want to do with Arduino, you could purchase some equipment or not. If you just want to have an understanding of what Arduino can do, you do not really need any equipment, just curl up on your sofa and enjoy the book. I promise to make it as interesting as possible. If, however, you want to read this book as a foundation course for Arduino, I advise that you buy the following equipment while you are it: some lengths of solid core wire, a digital multimeter (go for the cheap ones) and of course, an Arduino Uno board. We will dwell on the Arduino Uno board in due course.

On using this book

As previously pointed out, this book is a beginner course on Arduino, and the contents point to that fact. The introductory chapters will tell you what you need to know about Arduino. It will answer the question of: what is Arduino? It will also tell you what Arduino is used for and its features. It will tell you how Arduino came about, and how it has developed along the years. It will answer the question of: who can learn Arduino? It will give you a brief introduction into Arduino hardware and all the components of the hardware. This first chapter tells you what Arduino can do and the different types of programmable circuit boards there are. Then it will tell you what makes Arduino superior to other programmable circuit boards. After this, we will talk about the Arduino Integrated Development Environment (IDE), explaining what it is and how it works hand in hand with the Arduino hardware. We will then talk about getting started with Arduino, how we can conduct simple practical experiments with our Arduino board: we will install the software, power it up and then we will talk about the different tools and menus in the Arduino IDE.

After the introductory parts of the book, will talk about the basics of the Arduino language. This is identical to the C++ programming language. This is because the C++ programing language is the language Arduino was built from. This will also give you a brief introduction to the world of programming in general.

CHAPTER TWO: THE THINGS YOU SHOULD KNOW ABOUT ARDUINO

A brief discussion on Arduino and the internet of things (IoT)

Arduino is an electronic platform that also happens to be an open source platform. The hardware and software parts are easy to use, so you have nothing to fear. Arduino boards (which we will talk about later) are able to read inputs. Inputs like light from a sensor (as we described above), or a push from a button. It could be a message from your SMS, a twitter or WhatsApp message etc. When it reads those inputs, it turns them into an output. The output could be switching on a light bulb, sending power to a servo motor, or right about anything which can be powered by electricity. You could command your Arduino to publish your post online, or tweet something on your handle, reply an SMS or make a call. This is the beauty of Arduino; it can be adapted to suit every one of your purposes. All these are done by commanding your Arduino board via set of instructions sent to your microcontroller. You send these instructions using a special language designed for Arduino; the Arduino programming language and the Arduino software. As the years go by, Arduino has become the center of many projects from the simple everyday tools to the complex engineering and scientific projects. The fact that Arduino is open source is its greatest strength because people from all walks of life; engineers, students, programmers, and just about every person who has an interest in automation has gathered around this platform to contribute their quota to its ever growing knowledge. This is what makes Arduino so robust. The fact that it is open source has made its knowledge accessible to about everyone who needs it.

What is internet of things? If you have had a little exposition to technology, you may have heard the word internet of things. We will give you a little idea of what this thing is. Internet of things is a concept that has come to stay. Internet of thing is not something we rigidly define; it is something that can be shown. Think of your phone which may contain a GPS tracker, a projector remote controller, a door opener and light bulb switch. Notice how you control a plethora of processes with just a single device. Well, this is basically the idea of internet of things. We can go ahead to define internet of things as a concept that makes it possible for everyday things that have been embedded with electronics, software and sensors to be

connected to the internet, thus making it possible for them to collect and share data. Internet of thing have made it possible for you to completely control your house with your phone. For example, when your home system is connected to your phone, immediately you come home from work, your phone automatically connects to your home system and your door will open. Thus, you don’t need a key. Another example is the burglar alarm which notifies its owner when there is a forceful entrance that breaches the security of the house, you also see this in fire alarms which send a text message to a user’s phone. When we talk about internet of things, it goes beyond the home use we have just outlined, it sweeps through every aspect of our lives. Take a patient who is in the hospital for example, let us assume that this patient is on life support and is connected to a heart monitor. When there is any irregularity in the heart rate of the patient, a component connected to the heart monitor can be programmed so that it notifies the doctor and the caregiver at once. The concept of internet of things is also seen in engineering where huge CNC machines are controlled by just a laptop located in another room. As we said before, internet of things has come to stay and the best thing we can do is to tap into this vast resource by learning about it as much as we can. Now, Arduino is a very important guy when it comes to internet of things, this is because of what Arduino is capable of doing. It is the little programmable circuit board that does all the magic. In a subsequent section, we will talk about programmable circuit board. However, let us first talk about how Arduino came about and how it has become a major player when it comes to internet of things.

A brief history of Arduino

Arduino was the brainchild of the Ivrea Interaction institute. In 2005, Massimo Banzi and David Cuartielles (building upon Hernando Barragan's work, the creator of Wiring) created Arduino, a programmable design that is easy to use. They designed it for interactive art projects. At the Ivrea Interactive Design Institute in Italy, the software which was based on Barragan's Wiring was created by David Mellis. A little while later, Gianluca Martino and Tom Igoe joined them in the project and the five of them are the original founders of Arduino. Their aim was to create a device which was simple and easily connectible to different things such as motors, sensors, relays etc. thy wanted it to be easy to program, they also wanted it to be cheap. This was because they knew students and artists were not always wealthy. They selected their microcontroller and designed a circuit board that had easily handled connections and then wrote the program for the microcontroller and then put them together in a simple Integrated Development Environment (IDE). This was how the Arduino was born.

Since after this, Arduino has grown in many directions, beyond what the original developers could even dream of. Different versions of it kept coming out, with some of them becoming smaller and smaller, much smaller than the original board. Some of them also came out larger than the original version. However, each of these boards had a specific purpose for which it was designed for. However, as much as they looked different in size, they had one common denominator, which was the fact that they were serviced by the same Arduino Integrated Development Environment (IDE).

Designed to be used for fast prototyping. The aim of its development was to be an introductory tool for students who have no foundation in electronics and software programming. However, when Arduino was introduced to the general public, it began to be adapted to suit new needs and challenges. Since Arduino boards are completely open source, it gives users the power to independently build them and adapt them to every specific and unique need. What is more? The software is also completely open source, and thus it keeps growing and becoming more robust as the years go by, through the constant inputs from users all around the globe.

What is Arduino used for?

From the foregoing, you may have begun to catch a glimpse of what Arduino can be used for. You must have begun to see the infinite potential of this miracle interface. The nature of Arduino has made it possible for it to be adapted to every form of automation possible. Starting from the simple automation of switching on and off your light bulb to the control of the robotic arms of different equipment. Arduino finds use from the most basic of home appliances to the most complicated of technical equipment. Arduino is what you see in automated doors, both the ones which use proximity and motion sensors and the ones controlled by remote controls. They all Arduino have Arduino in the core of their performance. The light bulbs which you see in movies which switch on when they detect a human presence or when the door opens or closes, they are controlled by Arduino. The smoke sensors which set off alarms when smoke is detected; the alarms are controlled by Arduino. When the smoke sensor sends signals to the Arduino, the Arduino via a series of processes described in the introductory part of this book, sends a current to the pin which sets off the alarm. The fire alarm which causes water to be sprayed automatically when it senses a fire through its heat and smoke detectors; what do you think controls them? you guessed right! Arduino. Think of any situation where automation is involved and I will show you where Arduino lies at the heart of it.

The Arduino can also be viewed as a computer, the kind of computer which your thermostat at home is; this is because you can program it so that when some kinds of inputs come in, it will perform some certain actions as a result of those inputs. The Arduino is a microcontroller platform, and basically, what we do with them is to connect them to sensors (or connect sensors to them) such that when something happens, the sensor detects it and then some certain events will be triggered. Thus, you can have your Arduino connect with all sorts of sensors such as temperature sensors, light sensors, proximity sensors, and every other imaginable type of sensor. You can also connect your Arduino to things like LED lights, motors, servo motors as we pointed out above. You can have Arduino connect to every imaginable electrical and electronic device. You could look at your Arduino as an advanced form of your thermostat which detects changes in temperature and

acts accordingly. Either raising the turning on the heat when the temperature drops below a certain level, or turning on the air conditioner when the temperature rises above a certain degree.

So, while we are it, may I congratulate you once more for your interest in Arduino. I assure you, your interest is not misplaced in the slightest.

Who can learn Arduino?

From the brief history outlined above, you would have seen that Arduino is friendly and accessible to all, and that everyone can learn Arduino. The very nature of Arduino has made it easy for everyone, even people with very little technical knowledge, to easily understand and adapt to it. The question you should be asking is: why should I learn Arduino?

The first reason why you should learn Arduino is the potential it unlocks within you. You wouldn't know the extent at which you can tinker and goof around with techy stuff until you learn Arduino. What makes Arduino even more powerful is that there is almost no limit to what you can make with it. In this case, you are just restricted by your own imagination. Arduino unlocks your magic fingers when you learn it. Imagine this scenario where you get yourself a model toy car, driven by a battery. Then in your exuberance to test your Arduino skills, you find yourself programming the car such that it moves on its own and when it senses barriers, it changes direction. This is just a tip of the iceberg of what you can do with Arduino. Did I tell you that when you learn Arduino, you now have the power within your fingertips to build a robot? Who knows, your robot might begin the robot uprising!

Well, we have talked about all the techy goofy stuff you could do with Arduino. That is assuming you do have found nothing else more meaningful you could do with it. Well, there are plenty of very meaningful things you could do with it. For instance, leaning Arduino could provide you with that soft opening you would need to break into the world of programming. Most people find programming very difficult to get into. This is because of its rather unfriendly and sometimes forbidding nature. Arduino, with its very nature to unlock your goofy and imaginative nature is such a joy to learn. You would find yourself enjoying your learning of Arduino so much that you wouldn't know that you have become familiar with the world of programming. When you finish your training on Arduino, go back to that python programming book or that Java programming book, or that C programming book and find that it has become less difficult than when you left it in frustration a few weeks ago.

Besides being a great introduction to the world of software programming,

Arduino is also a great intro to the world of electronics. This is a quite obvious fact because when you are tinkering with Arduino, you find yourself becoming more and more familiar with all those electronic components which you would be coming across over and over again in the course of your Arduino education.

Another top-notch reason why you should probably try out Arduino is that it is actually a very cheap hobby to go into. Unlike mountain climbing and skydiving. You could just be at the safety of your room and order all those cheap contents and get a-tinkering! Getting a complete Arduino starter kit would not cost you up to seventy dollars. Unlike other hobbies I know.

Programmable circuit boards and microcontrollers

Programmable circuit boards are circuit boards that are programmable. Your Arduino is a good example of a programmable circuit board. A programmable circuit board contains microcontrollers that can be replaced when they are damaged (we will talk about microcontrollers in due course). Examples of programmable circuit boards are:

The ESP32: this board incorporates some features not found in the basic Arduino board, the ESP32 contains a Wi-Fi and a Bluetooth module, among other things. Arduino boards do not contain these things, this is why Arduino has an allowance for shields, we will talk about shields later on, but before then, think of the shield as an enhancer. For instance, when your Arduino project requires a Wi-Fi connection, you can connect a Wi-Fi shield to it. Let's just say that the ESP32 microcontroller eliminates the need for a Wi-Fi shield.

The Raspberry Pi: this is more than a programmable board because it is technically a computer. The Arduino board is programmed with the C++ programming language while the Raspberry Pi was programmed with the Python programming language. We will talk about programming languages in the latter parts of this book.

and of course your Arduino. There are plenty of programmable circuit boards out there, but the aim of this paragraph is to give the reader an idea of what Arduino is.

Your Arduino is powered by a microcontroller. Everything in the board is geared towards providing power to the board and making the board to communicate with your computer.

What is a microcontroller? A microcontroller is like very small computer. A computer on a chip. It possesses all the characteristics a computer should possess, and it has all the components of the early computers and even more. A computer has a processor, a random access memory (RAM) for temporary data storage and read only memory. A microcontroller has a few kilobytes of random access memory (RAM), it has a few kilobytes of erasable

programmable read only memory (EPROM) and it has pins. Input and output pins, that connect the microcontroller to the other parts of the electronics.

The inputs read both digital and analog. The digital inputs read are such like checking if the switch is on or off and the analog inputs are such like the voltage in a particular pin. This means that different types of sensor can be connected light, temperature, sound etc.

Just as the inputs can be both analog and digital, the outputs can also be both analog and digital. Thus, a pin can be set to on or off. On is 0volts and off is 5volts. You can use this to switch a light emitting diode on or off. It is the output that can be used to control such devices require higher power such as motors (like the servo motor we saw in the introductory part of this book). Also, it is possible to set the analog output voltage, you set the output voltage of the pin to a particular value, this allows you to control the speed of the motor as you see fit.

Microcontrollers are used in a huge number of systems. One device may have tens or hundreds of microcontrollers to handle individual tasks. For instance, a car might have plenty of microcontrollers that control different functions of the system inside the car. A microcontroller may control the braking system, while another may control the fuel injection, another may control the suspension control. These microcontrollers would communicate with each other. Some may still communicate with a more advanced computer within the car while others may communicate only with the other microcontrollers. With their input and output systems, they send and receive data and use their processors to process data and perform their tasks.

The elements of a microcontroller include the following core components:

The processor can be explained to be the brain of the microcontroller. It is the processor that responds to- and processes various instructions that control its functions. These functions include performing some basic calculations, basic logic and input-output operations.

The memory of the microcontroller is used to hold the data that the processor receives. It is then that the processor will work on the data it has received. There are two types of memory in the microcontroller. The first one is the program memory which is used to store long term information about

the commands that the processor carries out. The program memory is the permanent memory storage and is non-volatile, that is needing no external power source to hold the information stored in it over time. The second type of the memory in the microcontroller is the data memory and it is needed for non-permanent storage. It is only maintained as long there is a power source.

The next part of the microcontroller we have are the input and output peripherals. These input and output devices are the devices that act as the interface between the microprocessor and the external world. The input devices send instructions to the processor who receives them, processes them, and then sends them out to the output devices which then execute instructions that are external to the microcontroller.

Besides the above elements of the microcontroller, we have other supporting accessories in the microcontroller which include: the analog to digital converter (ADC). The analog to digital converter is a circuit, it transforms analog signals to digital, hence the name. The ADC allows the microprocessor to interface with analog devices, example sensors.

Then we have the digital to analog converter (DAC). The DAC does the opposite of what the ADC does. It enables the microprocessor to communicate its output signals to the external analog components.

The microcontroller also has a system bus, which is the wire that connects all the components of the microcontroller to each other. Then there is the serial port which is an input and output port that enables the microcontroller to connect to other external components. It is similar in function to a USB.

There are different types of microcontrollers available out there. You would find such ones as the Intel MCS-51, often called the 8051 microcontroller. It was first developed in 1985. Then there is the AVR microcontroller which was developed in 1996 by Atmel. There is also the programmable interface controller (PIC) developed by Microchip Technology. There are still a couple of other ones as well.

Arduino is not a microcontroller; it is a board. A development board. Boards provide complete access to all the of a microcontroller. This is like saying that with a board, you can unlock the full potentials of a microcontroller. The microcontroller Arduino uses is the Atmel microcontrollers.

Why use Arduino, what makes it better than other platforms?

Arguably, the most important reason for using Arduino is its open source nature. The Arduino platform welcomes continuous innovations into its already robust library. As the platform allows more people into its folds, these people become the future Arduino gurus and thus add their own quota to the already robust platform. It is indeed a cycle of excellence.

Arduino provides boards that are easy to use. Thus the new user is not faced with an insurmountable task in trying to decipher the board.

Then there is the beautiful advantage of Arduino being able to run in all the different types operating systems. It runs in Windows, Linux and Mac.

The libraries within the Arduino make it very easy to write Arduino code efficiently. We will talk about the Arduino IDE and the Arduino libraries in later sections. The Arduino platform is very accessible.

There are a large number of examples and projects which the user can learn with. There is never a lack of materials in the Arduino platform. Because the boards are open source, anyone who can build clone boards that perform similar functions to Arduino are allowed to do so. This makes Arduino boards and other Arduino compatible boards to be very cheap and easily obtainable.

In summary; Arduino has a marked advantage over others systems because it is cheap, able to cross-platform, has a simple and clear programming environment and has an open source hardware.

There are many other advantages that Arduino has over the other platforms, but as we keep on progressing in our knowledge of these boards, the more they become more apparent to us.

CHAPTER THREE: A BRIEF INTRO INTO THE ARDUINO HARDWARE

The different types of Arduino boards

Remember we said that Arduino is open source, which means that anybody can design the Arduino board. This is the advantage of the Arduino, as we pointed out, because it gives you, the user a freedom of choice. You can select the Arduino board which best satisfies what you want to do. Now, as this is the advantage of being open source, it is also its disadvantage. This is because as anyone can build an Arduino board, they will come up with different designs for the different boards. This will pose a problem to the user because the board you end up selecting may not support all the things you need from it. For instance; you may need your Arduino board to be able to connect to Wi-Fi, or have a specific rate of power consumption, or need an additional number of sensors that need to be connected to it. So, let us talk about the more popular types of Arduino boards out there (there are thousands of different Arduino boards by the way).

The Arduino Uno (R3)

The Arduino Uno is a good introduction into the world of Arduino. It is called the Uno which is the Italian one for one, because it was chosen to commemorate the release of the software (the Arduino integrated development environment, IDE) It consists of fourteen digital input and output pins. Six of the pins can be used as pulse width modulation (PWM), six analog inputs reset button and one power jack, a USB connection and others. The Arduino Uno contains everything required to maintain the microcontroller. All you need to do is plug it to your system with the help of your USB cable and give it power. The beauty of the Arduino Uno is that you need not worry too much about damaging, so it affords you the freedom of tinkering about and experimenting with it. The Arduino board, of course, is the most expensive type of Arduino board, this is not surprising, seeing that the board is original board, and as branded goods usually go, they are usually costlier than non-branded ones. This would also imply that when you buy the Arduino Uno board, you may have to pay as much as twice the price for it than for the other types of boards.

Figure 1: the Arduino Uno

The Arduino Mega

This is like the bigger version of the Arduino Uno. It has a lot of input-output pins, thus making it a good candidates for projects requiring plenty of I/O pins. The arduino Mega is shown below. You can also get the arduino mega board at a cheaper price than the arduino Uno board.

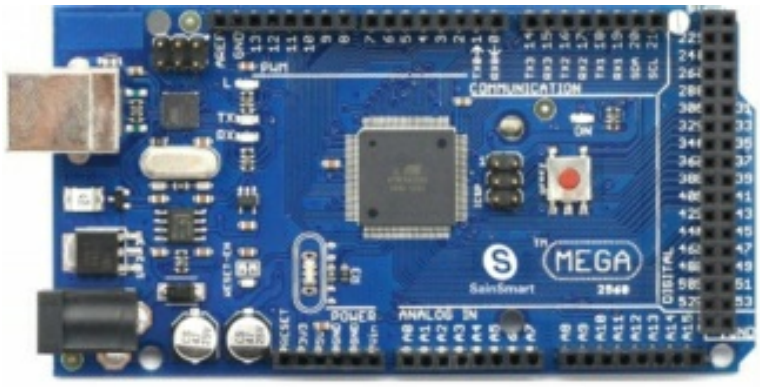

Figure 2: the Arduino Mega

The LilyPad Arduino board

The LilyPad Arduino board is adapted to wearable technology. This is a testimony to how Arduino has grown and adapted to about every need of modern technology. It can be sewn into clothing materials with the use of conductive threads. The LilyPad Arduino has a lot of details about it which we would not want to go into details because not only is it both beyond the scope of the readers of the intended readers of this book, it can also be easily gotten from the internet. A picture is shown below.

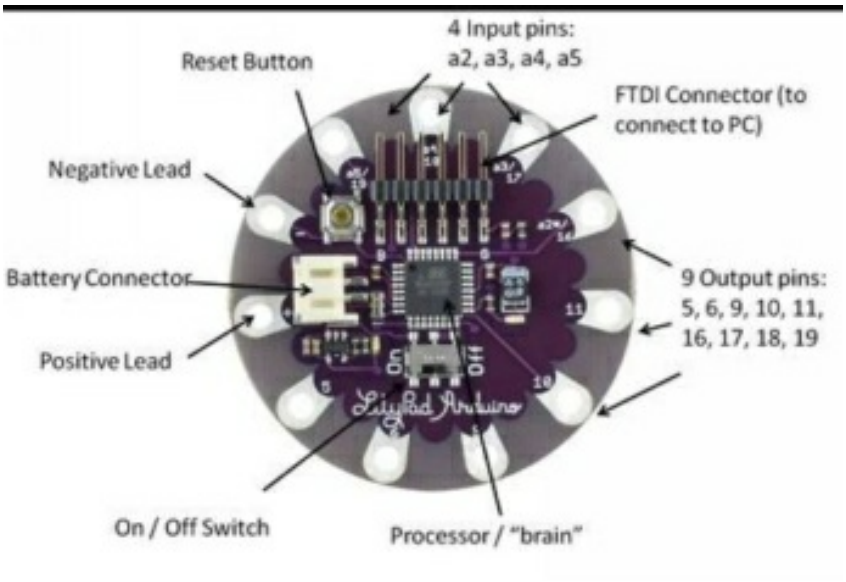

Figure 3: the LilyPad Arduino Board

The RedBoard Arduino Board

This is another type of the Arduino board family that was created with some modifications such as the type of FTDI chip that was used. FTDI is a company based in Scotland that produces USB. The RedBoard Arduino board is also an easy to use board. Take a look at the figure below.

Figure 4: the Arduino RedBoard

The arduino Leonardo board

This is one of the simplest to use arduino boards. It uses just one microcontroller together with the USB. It is cheap and easily obtained. The arduino Leonardo is shown below.

Figure 5: the Arduino Leonardo board

There are plenty of other Arduino boards including the clone boards not produced by Arduino inc. such boards include names like: Roboduino, Freeduino, Seeeduino (seriously, three e's?). These boards are produced to be cheaper than the original Arduinos.

Then there are other boards that are produced as an improvement to the original Arduino in some certain manner. There are plenty of them, but some of the popular ones include: Chipkit, this one has been known to be a high speed board which is compatible with Arduino. Then there is the Femtoduino, which is a small variant of the Arduino. Then there is also the Ruggeduino and the Teensy.

While getting your arduino boards or any of its variants, you should always take functionality into consideration. This means that before you buy, you must ask yourself what you want for the project you are embarking on. Then you begin to search for it. There are thousands of variants of the Arduino, and the chances are that you will find what you are looking for.

Introduction to the Arduino Board

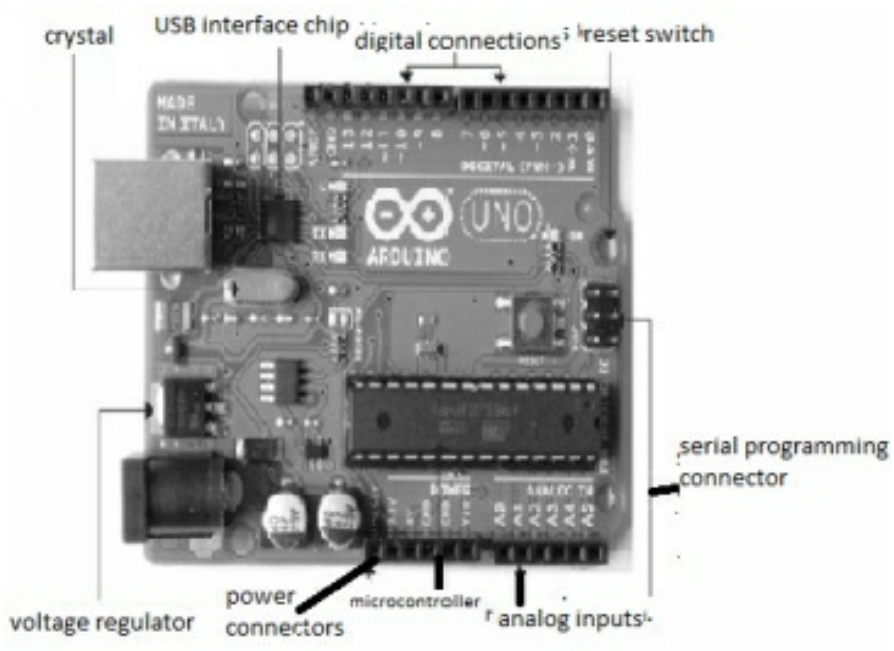

Figure 6: the Arduino board (Uno)

Take a long and hard look at the Arduino board above. This is how your lessons into the world of Arduino begins. The Arduino board is the center of all your Arduino devices.

Take note of the labeling as well. We will now explain the parts.

Power connectors

This like the gateway through which the entire board is activated. If you take a close look at the board, you will see some embedded circuits. The electricity flows through these embedded circuits to the connected parts. It is the electricity that makes it possible for these parts to perform their functions. Functions like receiving signals, analyzing these signals and data conversion. For the board to function very well, there must be a smooth power flow such that the system is always kept activated.

The sources of electricity supply can be placed through USB port or barrel jack. The connection distributes electricity around the board. The Arduino USB port is the same USB port found in computers and other USB devices.

They are the same size. There are some Arduino boards which do not have USB ports. People who like to use USB ports as their power gateway may find this a problem. So, be careful to check the specifications including the power supply source. USB ports are not used power supply ports; they are also used to transfer code from the computer to the Arduino board. It is the Arduino codes that you write in the Arduino IDE that you push to the Arduino board. The Arduino's USB port functions just like the ports in other devices, for transferring files.

Now, when you take a close look at the power connectors, you will see a labeling of 3.3 volts and 5 volts. What does this mean? It means that if you connect a wire to the 3.3-volt connector or to the 5-volt connector, it would transmit a voltage of 3,3 volts or 5 volts respectively. The five volt pins are the more commonly used power connectors.

The next power source the Arduino board has is the barrel jack. The barrel jack or power jack is your typical external power source. It is called the barrel jack due to the barrel-like shape that are embedded in an open board. It works with an adapter. Three metal prongs that conduct electricity are installed with the barrel jack.

The digital connections

The digital pins are used as input/output devices. They are used for reading and interpreting digital signals to digital outputs. For instance, when you push your button, it is an analog input, then when the LED bulb switches on, you can say that the analog input has been transmitted to digital output. The digital pin headers are labeled with a bunch of numbers from 0-13 right to left, which actually means that there are fourteen pins in all. The pin headers give you access to the pins on the chip located opposite to it on the board. What these pins help you do is: they can either act as inputs in which case, they can read the levels of voltage off something, or they can act like outputs, in which case, they can apply voltage to something. For instance, they can apply 5 volts or 9 volts as required. So, it is safe to say that the digital pins are input-output devices. That is why they are called I/O pins. The digital pins; 3, 5, 6, 9, 10 and 11 can use pulse width modulation (PWM). This simply means that they can actually adjust the amount of voltage they apply between 0 and 5 volts. So, with these pins, you can connect resistors into those holes or connect LEDs into the holes and you can

be sure that it will make electrical contact with the electrical components.

Now, let us talk about the pins: 0 and 1. Pick up your Arduino UNO board and look at the pins 0 and 1, you will see that they have a label; RX and TX. RX stands for “receive” and TX stands for “transmit.” When you are communicating with your computer or with another device, through your Arduino, it is these two pins that are being used. So, it is advised that you do not use the transmit and receive pins too much especially if you are working on a project which involves a lot of communications. Because it might affect how your program works. To go along with the transmit and receive pins, there is also the transmit and receive LEDs which are embedded on the board. These LED lights will blink when you are sending or receiving data. So if you have used an Arduino before, or seen them in use, you will notice them blinking when a sketch is being uploaded to the board (the Arduino codes you write are called sketches). This is because information is being sent to the Arduino. These lights are good because they are an aid for detecting issues (troubleshooting) in the Arduino. Take for instance, you are loading a sketch into the board and the lights for RX are not blinking. This is a sure sign that what you are doing is not working. The information you are trying to send to the Arduino board, the Arduino board is not receiving that information. If on the other hand, you think your Arduino board should be transmitting information, and the TX lights on the board is not blinking, you can be sure that the board is not transmitting the information it is supposed to.

The digital connections can also have more than a single connector; it is based on the model.

Analog connections

These are what the Arduino uses to transfer data signals from the analog sensor. In the Arduino board, you can find where the analog pins are by looking for the place labeled “analog in” this is a short form of “analog input.” It is here that the analog signals are converted to digital ones. The digital signals are what we see on the displays. If you want to build a temperature sensor for instance, which you want to transmit output in digital form. Let us say you have a thermometer which as we know, reads analog signals. The digital connector is like the interface between the analog and the digital display.

The analog pins are marked A0 to A5 in the Arduino Uno board. This

brings the total number of analog pins to five. The analog pins as we explained above help convert analog signals to digital signals. Here is how it does that: the analog pins give you access to a thing called the analog to digital converter. The analog to digital converter is on the microcontroller. What it allows you to do is to take analog signals and convert them to digital signals. For example, (just as in the previous example), let us say you have a radio with a volume knob and you turn up the volume to ten or say eleven. You know in that the analog knob, there are an infinite number of small steps between ten and eleven for instance (for example, the volume can be turned to 10.1, 10.11, 10.2… and so on), but you cannot get this in the digital setting. The digital setting is divided into a number of small discrete steps which are not infinite. This would mean, that between ten and eleven, there are no infinite intervals. What the analog to digital converter does is to take those infinite variations in the analog signals and it digitizes them into discrete or small steps. Most of the sensors you will use will provide their information in analog form, so this analog to digital converter is very useful. Also, take note that those analog pins could also be used just like the digital pin headers which we discussed in the previous sections. It is up to you how you want to use them.

The number of pins the Arduino has depends on the model. The analog pins are usually located on the lower right corner if your view of the box is with power jack at the left lower side.

The reset switch (reset button)

The function of the reset button is just what the name is, it resets the Arduino. This means that when you press the reset button, the Arduino will start all over again from the beginning of the program. This does not mean that it will erase anything off the board, what it actually does is a kind of reboot, but very fast. Now, when you move towards the power connectors, you will see another reset pin beside them (it is boldly labeled “RESET”). If you apply 0 volts to the pin, you will reset the whole board. This is another way by which you can reset the board as well.

The Arduino pins

The Arduino Uno board has fourteen digital pins and these pins can be used as either input pins or output pins. This is set by using some inbuilt functions within the Arduino programming language. Below is a list of some

of the Arduino pins and their some of their functions.

The serial pins 0 (Rx) and 1 (Tx): The Rx and Tx pins are used to receive and transmit serial data.

The external interrupt pins: These pins can be configured in such a way that they can trigger an interrupt on a change in value (we will talk about interrupts later on).

The PWM PINS 3, 5,6,9 and 11: These pins provide pulse width modulation using a function in the Arduino.

SPI pins 10 (SS), 11 (MOSI), 12(MISO) and 13 (SCK): These pins are used for serial peripheral interface (SPI) communications.

In-built LED pin 13: this pin is what controls the LED, when the pin 13 is on HIGH, the LED is turned on, when it is LOW, the LED is off. It is connected with a built in LED.

Besides the fourteen digital pins, there also six analog input pins. Arduino Uno also has some other pins. They are:

AREF: This pin is used to provide a reference voltage for analog inputs using a built in function.

Reset pin: when this pin is made low, it resets the microcontroller.

The ground pins

You will find the ground pins on both ends of the Arduino board. They are clearly labeled “GND.” There are three of them in total. You will find two at the bottom close to the 3.3 and 5 volts’ pins, then you will the one at the top close to the digital pin 13. The ground pins allow you to access the lowest voltage on the board. We will not begin to go into the details of how they work, but just bear in mind that they are very important and we will be using them quite often as we get along in Arduino lessons.

The power-on LED

This one comes up when you send power to the Arduino board. Whatever way you send in the power, be it through a USB, or through a computer, through an external battery or any other way you send in power to the Arduino board. It is that power-on LED that indicates that power has been sent into the Arduino board.

The battery pack

If you want to power the Arduino via an external battery connected to it, you do this through the battery pack.

Arduino accessories

Arduino shields

Shields are components that you can directly connect to your Arduino board so as to add a functionality to the board. Now, if you connect something like a sensor shield to your Arduino board, it enables you to connect a great number of sensors to your Arduino board. If you connect a Wi-Fi shield to your Arduino board, it adds the Wi-Fi functionality to the Arduino board. Then, there is also a motor, or a servo shield, so that if you want to control a servo or motor, you can actually do that through the motor or servo shield. There is also the GPS field which can actually enable you to track the location of your Arduino device. You can also get the battery shield. The beauty of shields is that they are also stackable, which means that after you have added a motor shield for instance, you can also add a Wi-Fi shield over the motor shield, and even a GPS shield over the Wi-Fi shield and so on.

As you have seen how Arduino shields can be used to stack over each other so that you create a fully functioning system that does all you need from it. Now the question is: at what cost? Imagine you want to design a system that needs to control a servo, be able to connect to a Wi-Fi, have an extra battery connection, have a GPS that you can track and so forth. Now you stack all the shields together to do all you want. Now, it may do what you want but you are left with an ugly, unsightly and unwieldy stack. Now, you may not want the final product which you will deliver to the customer to be like this, so when you have ensured that whatever you are building works, you now design a board that contains all the components you need without any other extra components. This will introduce us to the term called prototyping. Arduino is said to be a prototyping board because it allows us to test for functionality with its different shields available. After testing the functionality of your design, you can now build a board that is specific for your purpose. For example, you may end up designing a new board which only contains a servo control port, a Wi-Fi port, a GPS port and a battery connection, without having any other extra parts. This is another advantage of Arduino; it enables the user to prototype.

Modules

We have talked about how shields add functionality to the Arduino board. Now let us imagine that you have a toy car which you are designing. Now you want this car to be autonomously driven. You know that these proximity sensors need to be at one part of the autonomous vehicle. So, this means that the proximity sensor shields will be at another part of the autonomous vehicle, and the servo motor shield will be on another side, and the Arduino board will be at another part of the vehicle. So you are at a loss on how you can make things work. This is where modules come in. modules make it possible for different shields to be able to connect to the adapter from different locations with the help of cables. For instance, the proximity sensor which may be located at the front and back of the car, can be connected to the Arduino board which may be located at the center of the vehicle via these modules and the cables. So, we can say that these modules act like adapters, but from a distance.

The LED lights

Let us say you are designing a temperature sensor for a system you are using which you need to keep at a particular temperature range. So you attach the temperature sensor to the system. Now, let us say that you need this system to indicate to you when the temperature is exceeding a particular range. One of the ways you can do this is by attaching LED lights to your system. You can make the red lights to indicate that the temperature is exceeding the safe range, and the green lights to show that the temperature is at a safe range. This is one of the uses of the LED lights in Arduino and electronics. They are used to indicate the states of different components of the system, or even the state of the system as a whole. So that if a condition is not met, a red LED might come on and begin to blink until the condition is met, at which it then turns green.

The breadboard

Also called the solderless breadboard. This is a component in which the wiring components are pushed into it. Thus the solderless breadboard provides a way of connecting electronics without having to solder them.

This is the major function of the breadboard; it helps us to make connections in our projects without having to solder. Now, there is nothing wrong with soldering, just that it kind of attaches a permanence to what we are doing. Now, you want to make sure that all the components of your

project are working before you begin to solder. That is where the breadboard comes in. It gives you the freedom to test and discard connections before making the final connections and soldering, without having to worry about soldering. So you can basically use the breadboard to connect everything you want to; like the servo motor, the LED lights, the sensors. Basically anything.

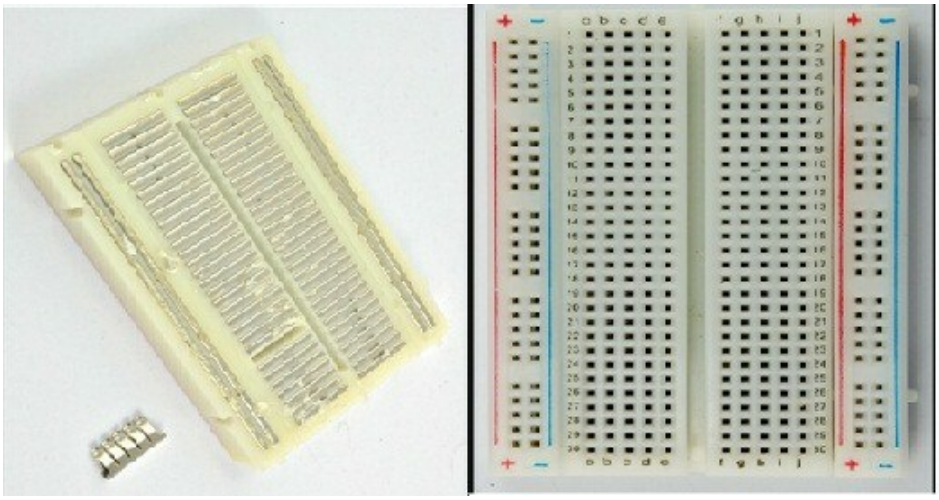

Figure 7: the breadboard

Now, let us talk about the breadboard's features. The first figure shows the typical breadboard and the connections. If you look closely at the breadboard, you will see that it is numbered and that the connections are in groups. The first column from the left is labeled at the top, with a positive (+) and a negative sign (-). Now, look at the second column. The second column is numbered by row from 1-30. The third row is numbered in the same way. Notice that the second column is numbered from a-e and the third column is numbered from f-j. the final column is also labeled on top as positive (+) and negative (-).

Now, haven seen the physical features of the breadboard, let us talk about how it is wired. Notice that in the second figure above, the breadboard has been opened. Now, each row connects anything placed on it. For example, row 15 will connect together any two components which are connected on row 15 (if in the first middle row, from a-e only). This means that if you connect a servo motor on row 15 and an LED light on row 15 as well, you have basically connected the servo motor and the LED light together. This also means that if you connect a temperature sensor on row 20, it is not

connected to the row 15. This is for the two middle columns labeled a-e and f-j. another interesting thing you can do with the breadboard is that you can connect individual columns, and they begin to function as one column. For example, if you connect row 15 with row 20 for instance with the help of a jumper cable, you have, in essence, made the rows 15 and 20 a single row. This means that when you connect the servo motor and LED light on the row 15, and connect a jumper cable from row 15 to row 20 (where the temperature sensor is connected), you have connected the servo motor, the LED light and the temperature sensor together.

The same cannot be said for the two extreme columns in the left and right sides of the breadboard however. Their connections do not run horizontally, but vertically. This means that when you connect a component on the column labeled (+), any other component you connect along that column will be connected to the first component. A look at the figure above should make this apparent.

The servo motor

You may will probably come across the servo motor in the course of your tinkering with Arduino, so we might as well introduce them now, so that you become familiar with them. the servo motor is a kind of a very precise type of electric motor which you can control to move in a specific way and to a specific point. The servo motor is no longer a novel invention, but its use spans across every aspect of technology. It finds use in almost every part of technology where some things are required to be in motion. These servo motors are great because they are highly efficient. They are also very powerful. These important features are what makes them very good candidates for operating systems that are remotely controlled or radio controlled. For example, remote-controlled or radio-controlled toy cars, toy robots, toy airplanes, etc. Servo motors also find applications in industrial processes, in-line manufacturing, robotics, pharmaceutics and even in food services. So, how do these highly efficient and important machines work?

When you open a servo motor, you will see a direct current (DC) motor, a potentiometer and a control circuit. At this point, you may need to find out what these terms stand for, if you have not come across them. a quick google search will give you directions. Gears are used to attach the motor to the control wheels, and as the motor turns, the potentiometer’s resistance

changes. This means that the control circuit can actually control and regulate the movement and direction of the motor, because of these constant changes in resistance. When the motor is at the position which it is supposed to be, power is cut, so that the motor stops when it gets to that position. Also, the amount of power (and consequently, the speed) transmitted to the servo is determined by the distance of the shaft from its desired position. This means that the further away the motor is from its desired position, the faster it will turn, and the closer it is from its position, the slower it will turn. In the field of automation, this is called proportional control. This also means that the motor will just give as much energy as needed to accomplish a movement. It does not go too fast when it is close, or too slowly when it is fast. This is what makes it very efficient.

So, how are servos able to be so efficient in accomplishing these speed moderations? Servos are controlled through the transmission of electrical pulses of variable widths through the wires. This is called pulse width modulation (PWM). The pulse width modulation makes it possible to have a maximum pulse, a minimum pulse and a rate of repetition. Now, the servo motor has only the potential to turn 90 degrees in either direction. This brings the total movement to 180 degrees. The motor can be said to be in the neutral position when the distance it can turn in either direction is the same. Now, this is where the awesome thing in the servo motor occurs: when the pulse width modulation is sent to the servo motor, it determines the exact position of the motor and thus the distance the motor requires to move to get to the final position. Think of it as a bat’s echolocation. You know that the bat does not see the walls of the cave, the cave is also filled with obstacles. So what does the bat do? It sends out high pitched sounds that hit the walls of the cave and other obstacles and then bounce back to it as echoes. So the bat knows that there are obstacles in so and so place, and thus; successfully avoids them. This is kind of like what the PWM does. The PWM determines the position of the shaft and then the rotor turns to the required position based on the length of time the pulse takes as it travels through the control wire. Also, after 20 milliseconds, the servo motor expects to see a pulse. This means that the length of the pulse will then determine the extent to which the motor turns. In essence, a pulse that is shorter than 1.5 milliseconds will make the shaft to move to position zero degrees, while a pulse of 1.5 milliseconds will move the shaft to the position of 90 degrees, and a pulse of length greater than 1.5 milliseconds will move the shaft to a position of 180 degrees. This

feature is what makes the servo motor indispensable in control. This principle is applied in robotic arms and in self-driving vehicles and so on.

Figure 8: the servo motor

The LCD screen

If you are reading this through your computer or through your phone, you are actually staring at an LCD screen. The acronym LCD actually stands for liquid crystal display, which is basically a definition of what the LCD screen does. It utilizes the two states of matter; the solid and the liquid states to make a display. The LCD makes use of a liquid crystal to display an image. The technology of liquid crystal display involves display on very thin display screen, far much thinner than other display technologies like cathode ray tube technology. They are found every in television screens, cell phone screens, computer screens and so on.

So, how do LCDs work? The working principle of the LCD is a little complicated, but let us look at it this way. The liquid crystal display works through the principle of polarization of light. You may have heard of the double slit experiment in high school physics. Where light is passed through a polarizer. Now, you may have heard that light is both a particle and a wave, so a light wave which vibrates in all directions is known as non-polarized light, now when this light passes through a polarizer, the polarizer, depending on its orientation, accepts only light particles which conforms to its orientation, and then absorbs the ones which do not. Think of the polarizer as a flat board with many horizontal slits. Now, imagine light to be a flat arrow with different orientations, the light particles which are horizontal (like the slits) will be allowed to pass through the slits, since they have the same orientation as the slits. Light particles that have a vertical orientation will not

be allowed to pass. Now, imagine that there are two slits, stacked against each other; a horizontal slit over a vertical slit. The horizontally oriented particles will pass through the horizontal slits, but when they get to the vertical slits, they will be absorbed. Now, an LCD is designed such that there is a liquid molecule between the differently oriented polarizers. This liquid molecule is twisted and thus orients the light in such a way that it can pass through the second slit which is differently oriented. This means that the light can only pass through the second polarizer if the last layer of the twisted molecule has oriented the light in such a way that it is now in the same plane as the second filter.

Now, when a voltage is passed through this liquid molecule, it causes it to untwist. When it untwists, light passes through it without the direction of this light being affected. This makes the light to strike the differently oriented polarizer and thus get absorbed. This means that the twisting and untwisting of the liquid crystal can be used to control the absorbance and admittance of light. The parts of the polarizer which has absorbed light is darker than the surrounding parts of the polarizer which has admitted light. Thus, the number “1” can be printed on a screen by darkening the screen strategically to produce a shape like one. Think of the LCD as a shadow producer. When light shines on an object, the object absorbs light and produces a shadow behind it. The LCD works roughly like this, only that it controls the light that will be shone, so that it can produce different objects.

So, lest we forget why we went on this long winded explanation on how the LCD works; it is used in Arduino as display. Have you been to a gas station, and you saw the gas meter display, well that is Arduino and LCD at work. The LCD is used in Arduino for purposes like temperature display, heat, and even in proximity or distance sensors. They help the human interact with the Arduino, so the human can read off on a screen what the Arduino wants the human to know.

CHAPTER FOUR: THE ARDUINO INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Well, now that we have talked about the hardware and all the different components of the Arduino, there is something else we may need to talk about, and that is the Arduino integrated development environment (IDE). Now, as you know, if you have taken a look at the Arduino board, it reveals nothing. Even though we have talked about the general components of the board, it still has not told us anything about how we can get the board to do whatever we want it to do. It is only when we have talked to the integrated development environment, then can we make the Arduino do what we want it to do. Think of yourself as a king, and the IDE as the messenger of the king. Assuming your Arduino board is your army, it is when the messenger has sent the king's message to the army that the army can do what the king wants.

Before we jump into what the Arduino integrated development environment is all about, we will first talk about what an integrated development environment is, in programming, in general.

The IDE is a piece of software that enables you to write codes, now you can think of an IDE along the lines of a Photoshop which makes editing photos much easier, or Autodesk which makes engineering designs very much easier. An IDE makes writing a code very much easier as well. The IDE performs three basic actions, it acts as a text editor because it can be used to edit text (correct spelling, punctuations, grammar, etc.), it acts a debugger because it can be used to correct the codes you have written, and it acts as a compiler because it can actually be used to read and translate code from high level language to low level language. This basically means that the compiler translates your written codes to zeros and ones which is the language the computer can understand. This is called machine code.

So, let’s talk about the Arduino IDE. Well, the first thing you will need to do is to download and install the IDE. I mean, we can’t talk about the IDE if you have not downloaded it, now can we?

Installing the software

So, first go to the Arduino website: http://arduino.cc/ you will be directed to a page that looks like this

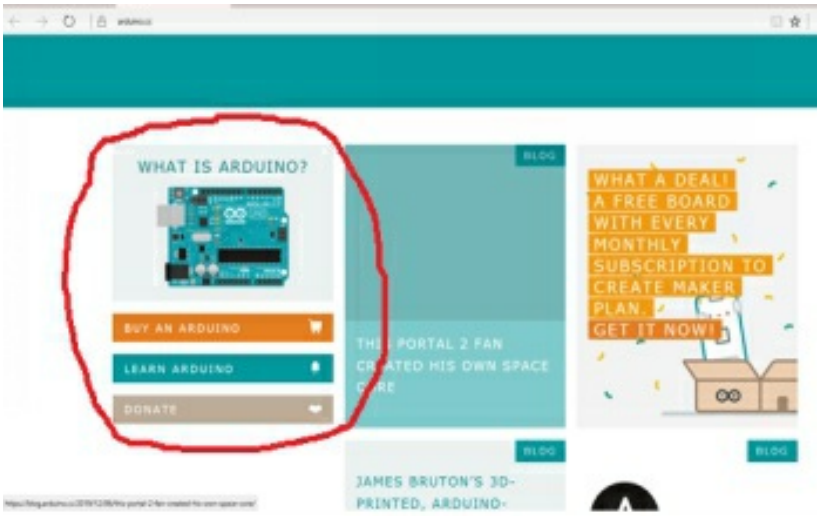

Figure 9: this is the page where the link will direct you to

Now, click on the circled time and it will direct you to this page.

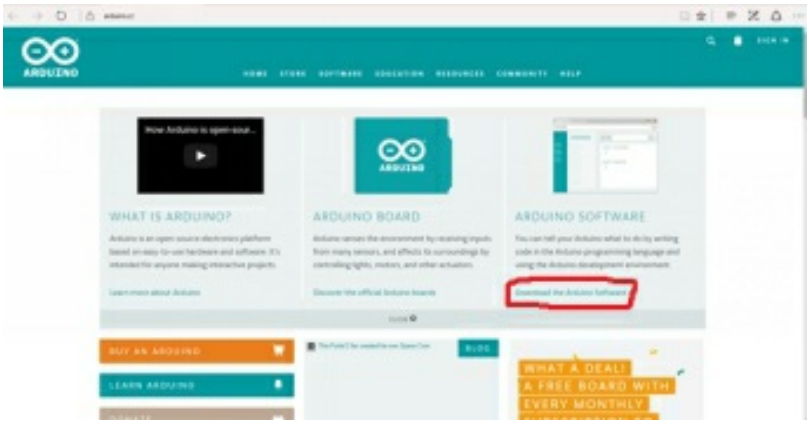

Figure 10: click on the circled link

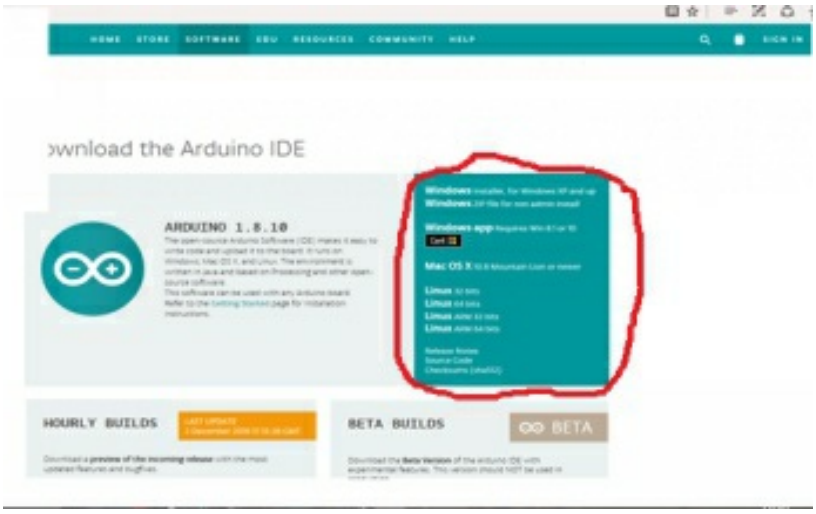

Figure 11: choose your PC

From there, choose the type of operating system you are using. Also, if you are using Linux, indicate if you are using a 32-bit operating system or a 64-bit operating system.

After this, it becomes easy, now, just click on the download file and you are good.

When you have finished downloading the file, you will probably end up with a compressed or zipped file. Just deal with the zipped file the way zipped files are dealt with (you unzip them with a suitable software). Next is to install the file. Just click on the unzipped file and follow the prompts. After this, you are good to go!

Now, let’s get going on the Arduino IDE.

The different tools in the Arduino IDE

So, now that you have installed your Arduino, open it and you will see the blank page which is your Arduino. We will begin our journey through the Arduino IDE on this IDE. Let us take a quick look at it and we will be on

Figure 12: the Arduino interface

So, this is basically how the Arduino interface looks like.

The file menu

Now, the first menu you will come across is the file menu. This is basically the file menu in other programs you have come across. You will see stuff like: “New” “Open” “Open Recent” “Sketchbook” “Examples” “Close” “Save” “Save as” “Page Setup” “Print” “Preferences” “Quit”

The “New” option works just like the way it does in every other software. It is used to open a new file. In this case, a new Arduino file.

The “Open” option also works just like every other “Open” used to open a previously saved file. Which in our case, would be a previously saved Arduino file. Arduino files are called sketches. However, if you are new to Arduino, and if this is the first time you are downloading Arduino in your computer system, I doubt if there will be any previously saved files.

Next in the list is the “Open Recent” option; just like in every other software, it is used to open recently saved files. sketches in our case.

Next in line is the “Sketchbook” as we know, Arduino files are called sketches. So the “Sketchbook” option is where all sketches are stored, every project folder you create in your Arduino is stored in the “Sketchbook.”

Next in the list is the “Example” option. This shows you example codes for different basic operations. I advise you to go take a look at those code snippets, they will help you to begin to get a feel of how the Arduino environment. There, you will see plenty of codes, and if you are already exposed to programming languages, it should be easier for you to follow, and thus set you off on Arduino journey. This option is also important because as a professional working with Arduino, it will make your work faster because you already have model codes available. All you need to is adjust the code or edit one or two things in the code. There are code examples for all kinds of systems, stepper motors, servo motors, Wi-Fi, robot controls and so on. Also as a learner, you can go to the example codes to know how the codes for what you are designing should look like, thus from there, you can write your own code.

The next option in the list is the “Close” option, and just like in every other software, it is used to close the program. However, you could just as easily close the program with normal shortcut at the top right corner of the screen.

The next option in the list is the “Save” option. This requires no explanation I’m sure, but for the sake of emphasis, the “Save” option is used to save files into the system.

You will also see the “Save As” option which gives you the option of saving your previously saved file in another location, and to make some other minor changes on how you saved your file initially.

The next option you will see is the “Page Setup” option. This gives you the option of making some changes in the configuration of your page. You could adjust the left, right, top and bottom of the margins of your page. Also, you can adjust the paper size and the orientation of the page as well.

After the “Page Setup” option, you will see the “Print” option. This is also not new to you, if you want to print your sketches, this is where you do it.

Now, after this option, we meet the “Preferences” option. Now, let us talk about this option. Now that you are new to Arduino, you may not yet know how the software works, and thus what you may prefer and what you may not. But as you keep advancing in the software, some things may start becoming apparent to you. However, this is where you make changes on your language, edit your font size preferences. This is also where you adjust the interface scale of your Arduino interface. Its default is usually at 100 percent, but you can adjust it after unchecking the “automatic” checkbox. Then there is the “Display line numbers” checkbox, I advise that you check this box as a beginner so that you can follow your code easily as well as know the line you at. There is the “Enable code folding” checkbox, this gives you the option of collapsing and expanding your lines of code at will. This is very handy when you have plenty of lines of code. You just click on the (-) sign and the code will shrink down to the main loops. The next checkbox which is unchecked by default is the “Verify code after upload” this will help you verify your code before you upload it to the Arduino. The next checkbox which may or may not be checked by default is the “Check for updates” checkbox. This helps your software to check for updates and inform you of such updates. There are many other options in the preferences window, such as the “Use accessibility features” “Use external editor” “Save when verifying or uploading” then there is the “Network” tab under the “Preferences” options. but they will become apparent to you as you keep using the software.

The final option you will meet under the “File” menu is the “Quit” option which works basically like the “Close” option. This option is used to close or quit the program.

The Edit menu

Under the edit menu, you will of course see some options which you are already familiar with. You will also some options which are unique to the Arduino IDE. The options include: “Undo” “Redo” “Cut” “Copy” “Copy for forum” “Copy as HTML” “Paste” “Select all” “Go to line” Comment/uncomment” “Increase indent” “Decrease Indent” “Increase font size” “Decrease font size” “Find” “Find Next” “Find Previous”

The “Undo” option works just like every other undo option, used to go back one step or to undo a code you have written, or to undo a command you have given while working on the software.

The “Redo” option also works just like the redo option in every other software. It is used to reverse an undo command.

The “Cut” option also works like every other cut option you know. It is used to move a snippet of code from one location to another location. You could use it to cut a code from Arduino and paste in your notepad or anywhere else for that matter.

The next command in the edit menu after the “Cut” command, is the “Copy” command. Just like every other copy command, it is used to copy a block of Arduino code from one location to another while the original code remains intact.

The next command is the “Copy for forum.” Now, let us talk about this command. As we have initially pointed out, Arduino is an open source system, so this means that many people would want to upload their codes for the different designs they have completed. They would want to upload these codes to different forums like GitHub and others. This is a good opportunity for the reader to learn the importance of these forums. It is nearly impossible that as a beginner, there is a problem you will face while trying to learn a programming language that another person has not faced. There is no question which you will have as a beginner which someone else has not asked, and which has not been answered, so, these online forums are your friends. If you have any question or have run into a problem, do well to consult these forums.

Now, back to the matter. Why the “Copy for forum” command? This command is set up because most online forums will not allow “live codes” into their forums. Most administrators disable live codes before they are allowed into their forums. Now, from the context, you would have understood “live code” to mean executable codes. Codes are actually capable of running. So, administrators disable these codes for the reason of security. You wouldn’t want your site to be hacked by a live code that you allowed into your forum, now would you? Now, the problem with disabling code is that it kind of renders the code impotent. In that when a person copies the code to run in their system, the code wouldn’t run. Not surprising since the code has been disabled is it? Now, this is when the “Copy for forum” command comes in to save the day. This command copies the code in such a way that when it is pasted in the forum, the forum does not recognize it as a

live code, rather the forum sees it as text file. So, the forum reads it as a standard text, but it actually is code. So, when you copy this code and paste it in your IDE, it resurrects again and becomes code! Think of it as a material that takes on different states at different states. Or think of it as a material that enables the transformers to change their shapes in different locations.

The next option is “Copy as HTML” this one works just like the “Copy for forum” it is also used to transform code to a form that is not flagged and disabled by online forums.

The next command in the edit menu is that the “Paste” command. This, as you know, is used to paste copied or cut snippets to a new location.

The next command is the “Select all” command. It works just like the “Select all” command in about every other software you know. It is used to select all the sketches at once.

The next option is the “Go to line” is the command that is used to move the cursor to the exact line you want it to go to. This command works well when you have activated your line numbering. Then you can send the cursor to wherever line you want it to go.

The next command you will meet is the “comment/uncomment” it is used to comment or uncomment the line of code where your cursor is, at the particular moment. Now, you may or may not know that as your compiler is executing your code line by line, whenever it gets to a line of code that has been commented, it ignores it and jumps to the next line. Commenting your code is important, not just for other people who would want to read and understand your code, but also for you, the programmer. This is because sometimes, your own code would be strange to you when you return to it after many weeks or months or even days. It is only through the comments you make that the sketches you make can be meaningful to you, no matter how long you have left it. If you are new to the programming world, a comment is usually made with a double backward slash (something like this: //)

Together with the comment command, is the uncomment command as well, the uncomment command is used to remove a comment in the line where the cursor is, at the moment.

After the comment/uncomment tab is the “Increase indent” command. It is

used to increase the indentation in a line.

The next command after the “Increase indent” command is the “Decrease indent” command, it is of course used to reduce the indentation in the line where your cursor is.

After the indenting commands, you will meet the “Find” command. Just like in other word document software, it is used to find and replace a word in the Arduino sketch. For instance, you could find and replace the word “INPUT” with “OUTPUT” and it will be effected immediately. Now, you could replace all of them at once using the “Replace all” command in the “Find and replace” dialogue box, or you could replace them one after another.

The “Find next” command is what you use to find and replace a word one after another. Whenever you click on the command, it finds the next word which you have already indicated in the “find and replace” dialogue box.

The “Find previous” command is used to do just the opposite of the “Find next” command. It highlights the previous word which corresponds to the word indicated in the “Find and replace” tab.

The Sketch menu

The sketch menu involves all the actions that are involved in implementing your sketches. Remember we said that in Arduino, your codes are called sketches. The sketch menu has the following commands: “Verify/Compile” “Upload” “Upload using programmer” “Export compiled binary” “Show sketch folder” “Include library” and “Add file”

The Verify/Compile command works like this: when you want to upload codes to the Arduino, it will verify the codes before it is uploaded. So, what if you just want to verify and compile your Arduino sketches without having to upload to any Arduino? Maybe, you don’t have your Arduino board handy, or you haven’t gotten an Arduino board, but you just want to keep in practice, how do you know that your Arduino skills are still intact? This is where the Verify/Compile command comes in handy. You can run your sketches with the command to check if the code is correct or if there are issues you may need to resolve. If your code is correct it will show you a message: “Done compiling.” If, on the other hand, you have an error in your code, say you forgot to put in a semicolon at the end of your code, it will show you a

message: “Expected ‘;’.”

So, the next command after the verify/compile command, is the “Upload” command. This is basically where you upload your code to the Arduino board. Then there is the “Upload using programmer” but that’s some complicated stuff you don’t need to worry about for the time being. Then there is the “Export compiled binary” another complicated stuff you don’t need to worry about now.

The next command after the upload commands is the “Show the sketch folder” command. When you click on it, it redirects you to the folder where the sketch is saved.

The next command is the “Include library” command. Here, you can add any library you want to use in your sketch. But hold up, what is a library? Well look at a library this way: when you are looking for a book someone wrote, besides buying the book, you look for it in a library. A library in programming means about the same thing. When someone writes a code, and they think they would probably be needing the code in the future, the save it in the programs library so that they can call it at will. Thus, a library is a place where reusable codes are stored. When you click on the “Include library” command, you will see a lot of default libraries already installed inside your Arduino IDE. You will see libraries for robot control, mouse, Wi-Fi, etc. However, what if there is another library that you need which is not included in the default libraries? What will you do? Remember that Arduino is an open source platform. So, there are plenty of updates by different people who keep making all sorts of designs and uploading them to the Arduino platforms. So all you need to do is go and download the library you need. So, how do you download these libraries from the platforms? Well it is actually simpler than you think, all you need to do is just make a google search and you are good to go. What you should be worrying about is how to import the library into the Arduino environment. Well, thank heavens it is simple as well. Click on the “Add .ZIP Library” option and it will redirect you to your file manager. From there, you can find the file.

The next command you will encounter is the “Add file” command. It is used to import a previously saved file.

The Tools menu

Now, just as the sketch menu helps along with anything concerning uploading your code, the tools menu contains the tools which you can use to make your code read better, compile better, and run better. It also contains commands that can be used to transform your code to different forms for easy transfer. The tools menu contains commands like: “Auto format” “Archive sketch” “Fix encoding and reload” “Manage libraries” “Serial monitor” “Serial plotter” “WiFi101/WiFiNINA Firmware updater” “Board: ‘Arduino/Genuino Uno’” “Port” “Get board info” “Programmer:l” “Burn bootloader.” Now, let us talk about each of these commands in turn.

When you see the “Auto format” command, what comes to mind? It is used to clean up your code. Remember that it is possible to finish writing a code and storing it, this is where the auto format comes in handy. It cleans up your code and makes it human readable. Remember the verify/compile command in the sketch menu? The auto format command does basically the same thing with the verify/compile command. But the advantage of the auto format command is that it actually cleans up your code by itself, it does not just point out errors the verify/compile command, it actually goes through your code, straightening things out. I advise you to use this command after writing your Arduino sketches. This is because the little errors (such as omitting a semicolon, putting a semicolon where it is not supposed to be, or forgetting to put your curly braces) are the most difficult to track. The auto format command conveniently takes care of this for you.

The next command is the “Archive sketch” command. This command is used to save your code in a ZIP format so that you can send your code to other people through email and stuff.

After the archiving command, you will meet the “Fix encoding and reload” command. This command is used to restore the sketch to its initial form before the user begins to make changes to the code.

The next command which is “Manage libraries” is another method of installing your downloaded libraries into your Arduino IDE so that you can make use of it.

After the manage libraries command, the next command you will encounter is the “Serial monitor” command. The serial monitor is used to read information from the Arduino. For instance, you are working on a temperature project, so you put in analogue sensor in your Arduino and you

want to read the temperature which the analogue sensor has measured. You can do this by using the serial monitor. So the serial monitor is the command that enables you to read analogue signals in their digital forms.

The next command after the serial monitor is the “Serial plotter” this goes a step further than the serial monitor, because it does not just display the temperature value, it actually plots a chart of the temperatures and display it for the user to look at. So, as the temperature goes up or down, the chart goes up or down. So, the serial plotter is quite valuable for stuff that involve continuously changing conditions. conditions like; humidity, temperature, etc. can be easily handled by the serial plotter which produces a visualization of the changing conditions.

The next command is the “WiFi101/WiFiNINA Firmware updater” which is used to update and checking the Wi-Fi firmware of your Arduino board. This command is one of the tools required to update your board.

The next command you will meet along the line is the “Board: ‘Arduino/Genuino Uno’” command. When you click on this command, you will marvel at the huge quantity of boards that are at your disposal. You will see boards like Arduino/Genuino Uno, Arduino Nano, Arduino Leonardo, Arduino Mini and so on. Now, these names are the names of the different boards which the Arduino software is compatible with. Remember that Arduino is open source, thus, it is possible for different boards designed by different people to connect to the software.

The next command you go to is the “Port.” This is where you go to ensure that the Arduino you plugged in is actually working. Go to the port, click on the board which you have plugged in to verify that the board is plugged. This is important because sometimes when you keep plugging in and unplugging your USB, the Arduino will get confused and hence, when you try to upload your software, it will fail. It fails because it doesn’t just know the port which you want it to upload it to. So when you click on the presently plugged port, it verifies to the software the port you want to upload to.

The next tool is the “Get board info command” it is used to get the details of the board. When you click on it, it shows you the specifications of the board you have plugged in. so when you get the information on the board plugged in, you can then go to the “Board: ‘Arduino/Genuino Uno’” command to select the board you need.

The next command which is the “Programmer” is used to choose the built in function compatible with the board you are using.

The next command which is the “Burn bootloader” command is a small program that manages the actions that are involved in uploading and starting the Arduino codes (sketches) you have made.

The Help menu

The help menu is what will help you find your way around Arduino. Just as this book is giving you some pointers, you may not be opportune to test out everything being taught to you right now, but as the days go by, and you become more familiar with the Arduino software and the Arduino board, you will find yourself running into minor questions you will need to be answered for you. Besides books, tutorial videos and online forums, another resource that you have at your disposal is the help menu. The help menu compiles your frequently asked questions and answer them for you. You will get the guidance on getting started, the Arduino environment, on troubleshooting, references, frequently asked questions, etc. The help menu is your go-to guy for most of the challenges you will face while using the Arduino software.

Finally, there are other symbols in the Arduino IDE, clicking on any of the symbols will show you what the symbol is used for. The checkmark symbol is the verify option, it is used to verify and compile your code. The right pointing arrow is the upload option, and it is used to upload code from the IDE to the board. The writing pad icon is used to open a new file. The upward pointing arrow is used to open previously saved files. The downward pointing arrow is used to save a sketch in your computer.

CHAPTER FIVE: THE PROGRAMMING LANGUAGES AND ARDUINO PROGRAMMING LANGUAGE

If you have a little knowledge of software, you would probably have heard of programming languages. You would have heard a person say: “I am a programmer” or “he is a programmer” you would have also heard things like: “I want to write code.” When you hear stuff like that, you will know that they are talking about programming. So, the question now is: what is this programming everyone is talking about? This section of the book will answer this question for you. It will not just answer the question of what programming is, it will also tell you what programming languages are, and what programming language was used for Arduino. So, sit back, relax, and let us begin.

Programming: what is it?

Well, the simplest definition or description of programming is that it is a way to instruct the computer to do different tasks. Well, this may sound shallow in the beginning, because you may say to yourself: “but plotting a graph with a computer is an instruction?” Well, technically yes, but let us pick out the definition term by term.

First of all, programming is an instruction to the computer. What this means is that you provide the computer with some sets of instructions, but these instructions are written in a language that the computer can understand. This is called syntax (we will talk about syntax and the Arduino syntax later on). Now, these instructions can come in various types: it could be the addition of two or more numbers, it could be rounding a number up or down. Now, just as we humans have different languages such as English, French,

German, Italian, Spanish, Portuguese, Swahili, Afrikaans, Mandarin, etc. the computer also has its own language which it understands if you s. Now, the beauty of the computer is that you do not have to teach it a new language (well technically you don't, but you have to install the software for the language though) the way you would teach a human being. The computer can work with any language you give to it.

Secondly, the computer does different tasks. Now, the computer does different tasks from the instructions given to it. These tasks range from simple ones like adding different numbers, rounding a number up or down, to complex ones that would require a series of instructions in a particular sequence. These more complex ones could be stuff like calculating simple interest when you have been provided with principal, time and rate. It could be tasks like calculating compound interest or the return on tax. These tasks require a series of steps and they are more complicated than just adding two numbers, so these instructions cannot be expressed in the same way you expressed the addition of two or more numbers. What this explanation drives at is that programming is a way to give computers instructions to do a particular task. It should not be mistaken for other activities which involve providing an input to the computer and expecting an output. You could say that programming is a way to instruct a computer to do a particular task whenever it is commanded to do so. So, when you write a code for a computer to add two numbers, whenever you want to add two numbers, the program will always add two numbers for you.

Programming languages: what they are

Remember the analogy we gave about the computer programming language being just like the human languages. Well, the analogy still holds. When you are speaking to someone, you don't use a language that the person cannot understand. For example, this book is written in English, I cannot expect a non-English speaker to understand this book unless it is translated to the language they can understand. Thus, just as English is a language, and French is another language, that is just how the computer has different languages which it understands. And just as I cannot speak French to a non-French speaker, and I cannot speak English to a non-English speaker, this is how I cannot write Java syntax for a C program or Python syntax to a C++ program.

Well, this brings us to the question: what is syntax? Well, syntax is simply the way which the symbols in a computer language are structured. It is the rule or set of rules which govern how symbols are arranged in a computer language. Just as human language has spelling and grammar rules, syntax of a programming language is the rules of spelling and grammar which the user has to obey before the computer will understand. However, unlike human beings, the computer is unforgiving in its rules. This means that while you can speak wrong grammar and your listener understands and fills in the gaps, the computer does not know what you are saying when you break the rules of syntax.

The different programming languages

In the previous section, we said that you cannot write a Java syntax to a C program. Well, you may have suspected that Java and C may be a programming language. Well, your suspicions are correct. There are hundreds of programming languages out there, and more are being developed as the days go by. Some of these are more suited for some specific types of tasks than others. Some of the popular programming languages these days are Python, Java, JavaScript, C, C++, Ruby, etc. these languages can be used for different things like building web pages, data visualization, artificial intelligence and machine learning, robotics and automation.

The language used in building the Arduino software is the C++ programming language, this means that the Arduino syntax is the same as the syntax for C++. So, if you already have a knowledge of the C or C++ language syntax, good for you, but if you don’t know, well, you will soon know all you need to know about the Arduino syntax.

A note on the data types

When you want to think of data types, think of it as the different ways the program stores data. You could also say that a data type is the type a data comes in. What this simply means is that the program accepts data in different forms such as integers, fractions and characters like letters. It can also be seen as the storage format which has a range of values it can contain. Before our Arduino stores data in variables, a specific data type must be assigned to it. The popular data types in Arduino include:

String( )

When something has been classified as a string, it means that it is seen as a quote, hence, it is usually enclosed in a double quotation mark. For instance, when you write a code like Serial.print(“hello world”), the “hello world” is a string (don’t bother yourself with the other elements of the syntax, it will become clear to you in due time). Also, if you write a function like this: a = String(14), 14 becomes a string because it has been defined as a string.

The array data type is a data type that can be accessed via index numbers. This is because the data type has been given some sort of labeling. mySensVals[0] == 2. In this syntax, the “mySenseVals” has been given an index of 2.

The bool holds one or two values: true or false. Each of the two values occupies a byte of memory space. So, this is what is used to check whether a statement is true or false.

A byte is a memory storage unit and it stores an unsigned number of 8 bits, between 0 and 255.

The char data type is used to store characters. A character is at least 8 bits in size.

The float is a data type for storing floating point numbers which are numbers that contain decimals.

The int. data type is used to store integers.

The long data type is used for the storage of large numbers.

Other data types include: Boolean, double, short, size_t, string, unsigned char, unsigned int, unsigned long, void, word.

An overview of the syntax used for Arduino

We talked about the integrated development environment and the different tools in the previous sections, now, we will talk about the syntax of Arduino and how it is used together with the tools. The Arduino IDE is made to be as simple for the user as possible. For example, once you open your program, the IDE comes up. You may not understand the stress this saves you, until you have used other IDEs. We have also said that Arduino uses the C++ programming language. So, the basic concepts of every other programming language are what you still use in the Arduino.

Now, you may not know this now, but the language which you use to code to your computer is what we call human readable language. This is because your codes can actually be read and understood by another human being or by yourself. Now, one of the things the IDE does is to take the human readable language, and translate it to a code which is called the machine readable code. This is the one the computer can actually understand. So the IDE is like a translator who translates whatever you say to the computer. This is called compiling.

The user does not know the processes the computer undergoes while compiling, so he/she does not feel any stress involved in compiling all the user does is write the code, press a button and the computer compiles. If there is an error, the computer indicates by displaying an error message. Sometimes, the error message is very clear; like telling you that you omitted a semicolon in a certain line. Sometimes, it is not so clear.

Now, let us take a look at the syntaxes used in Arduino and what they do. Consider this model code.

[code]

```
/*
  AnalogReadSerial

  Reads an analog input on pin 0, prints the result to the Serial Monitor.
  Graphical representation is available using Serial Plotter (Tools > Serial Plotter menu).
```

```
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/AnalogReadSerial
*/

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

[/code]

This is the code that is written for reading analog signals. Now, let us talk about the syntaxes and what they imply.

The semicolon (;)

A semicolon must always follow the end of every statement used in Arduino programming. Take the code below as an instance

```
int LEDpin = 9;
```

In the statement, an integer variable is being assigned a value. Take note of the semicolon at the end of the statement. The semicolon is used to tell the compiler that the chunk of code you are working on is finished, and that you want to move to another chunk of code. What a semicolon does for an Arduino code is what the period does in a sentence. It is used to show a

complete statement.

The double backslash used in single line comments //

Try typing a double backslash and notice how everything after that backslash grays out in that line.

A comment is what you use to make a code understandable. It is like labeling a diagram. It is the comments that help you or another to understand a code many days and months after writing it. Remember how we talked about the importance of commenting your code in the previous sections. A code is considered well-written if it is well commented. Take a look at the comment below.

//this is the pin on the Arduino that the LED is plugged into int LEDpin = 9

If you have commented on your code, you have not just helped yourself remember where to put in the LED, you have also helped another person to where to plug in the LED. The backslash makes the comment to be ignored by the compiler. This means that the compiler jumps every comment it comes along. Thus, you can write whatever you want to write on a comment without worrying about the compiler.

There is also a provision for a multiple line comment. The symbol used for the multi-line is the backslash and star (/*). Once you make a multiple line comment, you can write as many lines of comment as you like, until you close it with another backslash and star. The example of a double line comment is shown below.

/*the multiple line comment is opened with a single backslash which is then followed by an asterisk, so that whatever statement you make will be grayed out all through until you close the statement with another backslash like this*/

Consider comments to be the footnotes in a writing.

Curly braces {}

Curly braces are used when you want to include further instructions you want a function to further carry out. The order of using curly braces is that there should be an opening curly brace and a closing curly brace. This signifies the beginning and ending of the instructions. If you included the opening curly bracket only, and did not include the closing curly bracket, the

compiler will throw out an error code. An example of a statement that contains a curly brace is shown.

```
void loop () {      //thus curly bracket opens here.

        //a program is written here

    }       //this is where the curly bracket closes.
```

Reminder: make sure you close every opened curly bracket.

Functions

Let us talk about something more technical than we have been talking about before: functions. Functions are pieces of codes which used so many times that special keywords have been dedicated to them. Think of functions as command words you give to a dog. Whenever it hears a particular word like “sit!” or “wait!” the dog either sits down or waits for you. You basically do the same thing while writing a function. Writing a function is also like speaking a magic word. Once you speak these magic words, the things the magic word is capable of begins to happen. The following sets of instructions below could be called functions.

Washing car

1. Bring a bucket
2. Fill the bucket with water
3. Add a washing chemical
4. Get the car
5. Lather the car
6. Wash the car
7. Rinse the car
8. Dry the car
9. Put away your washing materials

This set of simple instructions involved in washing a dog can be brought together into a single function that we could call WashingCar. This means that every time you want these instructions to be carried out, you simply type WashingCar and all these instructions will be swiftly carried out.

In the Arduino software, there are some functions that have been built into the IDE so that whenever you type in these functions, the name of the function will show in orange in the IDE. For instance, the function pinMode() is a common function which can be used to show the mode of an Arduino pin.

You may be wondering what the parentheses that follow the function is used for. The reason is this; many functions need an argument to run. An argument is the information the function makes use of when it runs. For instance, in the WashingCar function the arguments included could be the brand of the car and the type of washing detergent used, it could be the water temperature and the soap size.

```
pinMode(13, OUTPUT);
```

```
//Sets the mode of the Arduino pin
```

The argument 13, inside the bracket, refers to the pin 13 and the argument OUTPUT is the mode you want the pin to operate. When you key in these arguments, this is called passing. It actually means passing the necessary information to the functions. Some functions that are used do not require arguments, but the opening and closing parenthesis is still included, regardless, even though it is empty.

```
millis();
```

```
//Brings out the length of time in milliseconds that the Arduino has been running
```

If you type in the word “OUTPUT” in your Arduino IDE, it will turn blue automatically. Some frequently used keywords usually turn blue to help the user identify them.

You can easily make your functions in Arduino and get the Arduino to give them colors for you. However, there are two functions used in nearly every Arduino program.

Void setup()

As the name implies, the function setup() is used to set up the Arduino board. It is the function of the Arduino to execute all the commands within the curly braces of setup() just once. The normal things that are done with setup include stuff like setting the modes of the pins.

The void which precedes the function setup(), what does it do? The void implies that the function does not return information. Some functions, however, return information. For instance, the WashingCar function, it could return the number of buckets that will be needed to wash the car. The function analogRead() returns any integer value between 0-1023. These

things may sound new to you, but don’t worry, as you progress in your Arduino knowledge, they will become more and more apparent to you.

In summary, these are what can be said about setup(): it runs only once, it needs to be the function in the Arduino sketch, also, the function must be preceded and succeeded by curly braces. This means that there will be a curly brace opening and closing the setup function.

An example code of a setup() function. The code below is used to set the data rate in bits per seconds

```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
```

The void loop()

As the name implies, in the void loop, all the codes between the curly braces are repeated in a loop; over and over again. Forever. It is in the loop() function that bulk of your program will be. Since the function loop() returns nothing, just like the setup() function, it is preceded by the word void.

```
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

Whatever code you put in the void loop will execute over and over again. However, this does not work the way you would assume it will work. This is because even though the code seems to be repeating the same thing over and over again, what is actually happening is that after each complete execution of the code, there is a trigger which makes it begin to run again. This means that if you have written a code for a temperature sensor which should monitor the temperature and kick of the fan when the temperature reaches a certain level. If you are looking at the execution of the program, as long as the

temperature is below that threshold, it will be triggered to repeat over and over again. An observer will simply see a code that keeps repeating itself. However, each new execution is unique by its own standards and is set off by a trigger. The observer will notice the difference only when the temperature exceeds the benchmark and the fan is kicked off. This means that as the code loops over and over again, not every piece of the code is executed at every iteration of the loop, some are only executed when a certain condition has been met.

There are other functions, though not as popular as the two functions mentioned above.

Digital input/output

The digitalRead() function reads the value from a particular digital pin. Either HIGH or LOW. The syntax looks like this: (digitalRead(pin)). The pin within the parentheses is the Arduino pin number which you want it to read. However, the digitalRead() function cannot return HIGH or LOW if it has not yet been connected to anything. Also, the analog pins can also be used as digital pins and they are referred to as A0. A1 etc. just as we discussed in the previous chapters.

The digitalWrite() function sends a HIGH OR LOW value to a digital pin. If the pin was configured with the pinMode() function to be an OUTPUT, its voltage will be set to the value that corresponds to what it reads. That is, the voltage is 5V (or 3.3 volts for 3.3-volt boards) for HIGH and 0 volts for LOW. If on the other hand, the pin is configured to be an INPUT, the internal resistance (internal pullup) on the input pin will be enabled (HIGH) or disabled (LOW). The syntax of the digitalWrite() looks like this:

digitalWrite(pin, value)

The parameters, pin and value; the pin refers to the pin number in the Arduino you are referring to. The value refers to whether it is HIGH or LOW.

Take a look at this code below which makes the digital pin 12 an OUTPUT and switches it by making alternations between HIGH and LOW at a pace of one second.

```
void setup() {
    pinMode(12,        OUTPUT);      //this sets the digital pin 13 as output
```

```
}
void loop() {
        digitalWrite(12,          HIGH);          // sets the digital pin 12 on
        delay(1000);            // waits for 1000 milliseconds or one second
        digitalWrite(12,          LOW);          // sets the digital pin 12 to off
      delay(1000);                  // waits for 1000 milliseconds or one second
}
```

pinMode()

You would probably have seen the pinMode() function mentioned in the previous section and you may wonder what it is about. Well, the pinMode() function is used to configure a specified pin to behave as either an input or an output. The syntax looks like this:

pinMode(pin, mode)

The parameter pin is the Arduino pin number you want to set the mode of.

The parameter mode is INPUT, OUTPUT, or INPUT_PULLUP

Analog input/output functions

The analogRead() function

This function reads the value from an analog pin that has been specified for it. In Arduino boards, there are analog to digital converters. What this implies that input voltages between 0 and 5 volts or 3.3 volts (depending on the operating voltage) will be mapped into integer values between 0 and 1023. In simple terms, it is this function that enables the analog to digital converter to read off analog voltages and present them to the viewer in whatever unit of the parameter being measured is. This implies that a temperature sensor sends in analog values in terms of voltage, it is this function that works with the analog to digital converter to translate it to a human readable form.

The analogReference function

This one is used to configure the reference voltage used for a particular analog input. Now, there are two default reference voltages in Arduino, there is the 3.3 volts and the 5 volts. The analogReference() function is what you

can use to recalibrate the readings by reconfiguring these reference voltages. The syntax looks like this:

analogReference(type)

However, while making these changes in configuration, take note of the following: make sure not to use a voltage less than 0 volts or a voltage above 5 volts for external reference voltage on the AREF pin before you call the analogRead() function. This may possibly spoil the microcontroller.

The analogWrite() function

This function writes the analog value, that is, the pulse width modulation wave, to a pin. It is used to light a LED, and to vary the brightness of the LED as well. When the analogWrite() function has been called, the pin will generate a steady rectangular wave cycle until it is called again. The syntax of the analogWrite() function is shown below:

analogWrite(pin, value)

the “pin” inside the parentheses signifies the Arduino pin to be written into. The allowed data types are the int data types. This means that it only allows integers as input values.

The “value” refers to the duty cycle; between 0 and 255. The allowed data types are the int data types as well.

Look at the example code below

```
int ledPin = 9;                // the LED is connected to the digital pin number 9
int analogPin = 3;             // the potentiometer is connected to the analog pin number 3
int val = 0;                   // variable where the read value will be stored
void setup()      {
        pinMode(ledPin, OUTPUT};       // this sets the pin to be an output
}
Void loop()       {
        val = analogRead(analogPin);     // this will read the input pin
        analogWrite(ledPin, val / 4);  // analogRead values go from 0 to 1023, analogWrite values go from 0 to 255 which is roughly 1023/4
}
```

The Zero, Due, and MKR

The analogReadResolution() function

This function is what sets the size (in bits) of the value returned by the analogRead() function. This function is used to set the range (or resolution) of whatever parameter you want your Arduino to read. It is found in the analog API for Arduino Due, Zero and MKR family. So, if you are using Arduino Uno, you may not encounter it, since the resolution in Arduino Uno is set with lines of code.

The analogWriteResolution()

Just like the analogueReadResolution() function is an extension of the analogRead() function, the analogWriteResolution function is an extension of the analogWrite() function. This function is used to set the range of the analogWrite() function.

The advanced I/O

The tone() function.

The tone() function generates a square wave of a specified frequency on a pin. A duration for which the waveform will last can be specified, if not, the wave continues until it is terminated by calling the noTone() function (we will talk about the noTone() function next). The pin can also be connected to a speaker or a piezo buzzer so that it plays tones. Only a single tone can be generated at a particular time. The syntax is shown below

Tone(pin, frequency)

Tone(pin, frequency, duration)

The parameter pin is the particular Arduino pin where you want the tone to be generated.

The parameter “frequency” is the frequency of the tone, measured in hertz. The data type that is allowed is the int. data type.

The parameter “duration” is the amount of time the tone takes in milliseconds.

noTone() function

This function terminates the production of a square wave which is being

set off by the tone() function. It, however, has no effect when there is no tone being generated. The syntax is as shown below

noTone(pin)

The “pin” in the parentheses is the Arduino pin on which you want the tone to stop generating. A simple hack in using the function to generate different pitches on the multiple pins: you should call the noTone() function on one pin and then call the tone() function on the following pin.

The pulseIn() function

This function reads a pulse (which is either HIGH or LOW) on a pin. For instance, if the value is at HIGH, the pulseIn() function starts timing while it waits for the pin to go from LOW to HIGH, then it stops timing when the pin goes to LOW. This function returns the length of a pulse in microseconds or if a complete pulse was not received, it returns a zero (0). Take a look at the syntax below

pulseIn(pin, value)

pulseIn(pin, value, timeout)

The parameter “pin” is the Arduino pin number on which you want the function to read the pulse. The data types allowed are the int. Remember that this means that only integers are allowed.

The parameter “value” is the type of pulse the function should read either HIGH or LOW. The allowed data types are int. as well.

The parameter “timeout” is the duration to wait (in microseconds) to wait for the pulse to start. The default timeout is 1 million microseconds.

The pulseInLong() function.

This function is more adept at handling longer pulses than the pulseIn() function. This function reads a pulse (HIGH or LOW) on a pin. If the read value is HIGH, the pulseInlong() function starts timing when the pin goes from LOW to HIGH, then, when the pin goes back to LOW, it stops timing. Then, it returns the duration of the pulse in microseconds and if it does not receive a complete pulse, it gives up and returns to zero.

Since this function has been designed for longer pulses, it may show errors while reading shorter pulses. It works well on pulses from lengths of 10

microseconds to three minutes. The syntax for this function is as shown below

pulseInLong(pin, value)

pulseInLong(pin, value, timeout)

the meaning of the parameters:

the parameter “pin” refers to the number designation of the Arduino pin on which the pulse will be read. The data types that this function allows is the int. data types.

The parameter “value” is the type of pulse that is read. Is it HIGH or LOW? The data types that are allowed are the int. data types as well.

The parameter “timeout” is the duration of time in microseconds to wait before the pulse begins. The default time is one second which is one million microseconds.

The shiftIn() function.

Notice how all the functions in Arduino work just as their names imply. This function also works just as the name implies. It shifts a byte of data in, one bit at a time. It starts from the most significant bit or from the least significant bit (the rightmost or leftmost bit). Each of the bits is written in a data pin after which a clock pin is taken high then low (pulsed) to show that the bit is now available. The syntax of this function looks like this:

byte incoming = shiftIn(dataPin, clockPin, bitOrder)

The parameters explained:

The parameter “dataPin” is the particular pin where each bit will be input. The allowed data type is int.

the parameter “clockPin” the pin that you set to high and low to signal a read.

The parameter “bitOrder” is the order of which the bits are shifted in; either the MSBFIRST (most significant but first) or LSBFIRST (least significant but first).

The shiftOut() function.

Just as the previous function shifts in a byte of data one bit at a time, the

shiftOut() shifts out a byte of data one bit at a time. And just like the shiftIn() function, it does this either from the most significant or from the least significant bit (the rightmost or leftmost bit). Each of the bits is written in a data pin after which a clock pin is taken high then low (pulsed) to show that the bit is now available. The syntax for this function as shown below

```
shiftOut(dataPin, clockPin, bitOrder, value)
```

The parameters explained:

The parameter “dataPin” is the on which each bit will be output. The data type that is allowed is the int. data type.

The “clockPin” parameter is the particular pin you set on high then low to signal a read when you have set the dataPin to the correct value. The allowed data type is the int. data type.

The “bitOrder” parameter is where you determine which order the bits will be shifted out. It could either be MSBFIRST (most significant bit first) or LSBFIRST (least significant bit first). The allowed data type is the byte.

The time functions

The delay( ) function.

This function is used to pause the program for the amount of time (in milliseconds) specified as a parameter. 1000 milliseconds are in one second.

The syntax looks like this:

```
delay(ms)
```

The parameter “ms” is how many milliseconds to pause. An example code shows how this function is used:

```
int ledPin = 13;  // signifying that the LED is connected to digital pin number 13

void setup()      {

        pinMode(ledPin,          OUTPUT);        // this sets the digital pin as an output

}

void loop()       {

    digitalWrite(ledPin,  HIGH);              // this sets the LED to on

    delay(1000);           // waits for 1000 milliseconds or 1 second
```

```
    digitalWrite(ledPin,  LOW);              // sets the LED to off

      delay(1000);          // waits for 1000 milliseconds or 1 second
}
```

From the code above, each delay function included requires you to specify the delay time (which is in milliseconds).

The delayMicroseconds() function

Just as the delay () function works in milliseconds, the delayMicroseconds() function works in microseconds. It pauses the program for the specified amount of time in microseconds. Since there are one thousand microseconds in one millisecond, there are one million microseconds in one second. The longer the delay duration, the lesser the accuracy. Thus, the highest value to produce an accurate delay is 16383. This will probably change in the future. However, if you need a delay longer than a few thousand microseconds, make use of the delay () function. The syntax of the delay function is shown below

```
DelayMicroseconds(us)
```

The parameter “us” is how many microseconds to pause.

You should probably use the delay () function to set delays from ranges of 3 microseconds and up. If you set it at lower values, the accuracy is not guaranteed.

The micros ( ) function

This function returns the number of microseconds the Arduino board has run since it began running the current program. After approximately seventy minutes, the number will go back to zero. On Arduino boards of 16 MHz, the micros( ) function has a resolution of four microseconds (this means that it returns a time value that is always in multiples of four). On Arduino boards of 8 MHz (the LilyPad for example), the micros () function has a resolution of 8 microseconds. The syntax of the microse() function is shown below

```
time = micros ( )
```

Take a look at the block of code below. That is an example of the micros () function.

```
unsigned long time
```

```
        serial.begins (9600);
}
Void loop ()      {
        Serial.print("Time:   ");
        time = micros ();
            Serial.print(time);          // this prints the length of time that has passed the program started
            running.
             delay (1000);     // wait for one second, so that you will not send huge amounts of data.
    }
```

Take note that there are a thousand milliseconds in a second and thousand microseconds in a millisecond. It is important to bear this in mind while inputting values, to prevent mixing them up.

The millis( ) function

Just as the micros( ) function returns the time the program has run, in microseconds, the milllis( ) function does the same thing, but in milliseconds. It returns the time that has passed since the Arduino board started running in milliseconds. The syntax is as shown below:

time = milllis( )

the sample code below shows how the millis( ) function is used. A quick observation shows that it is almost identical to the micros() function.

```
unsigned long time
        serial.begins (9600);
}
Void loop ()      {
        Serial.print("Time:   ");
        time = millis ();

            Serial.print(time);          // this prints the length of time that has passed the program started
            running.
            delay (1000);     // wait for one second, so that you will not send huge amounts of data.
}
```

The math functions

The abs( ) function

This function calculates the absolute value of any number. Take a look at the syntax below

```
abs(x)
```

The parameter “x” is the number that the user inputs.

The sample code below shows how the abs( ) function is used

```
abs(a);
a++;
```

The constrain() function

Just like the name implies, the constrain() places a constraint on the range of a number to br allowed. The syntax is as shown below

```
constrain(x,  a,  b)
```

The parameter “x” is the number to constrain. The data type allowed are all data types.

The parameter “a” is the lower end of the range and it also allows all data types.

The parameter “b” is the upper end of the range and it also allows all data types.

Look at this example code below

```
sensVal = constrain(sensval,  10,  150); // this sets a limit to the range of the sensor values.
```

The map( ) function

If you have had any exposition to high school math, you would probably have heard of mapping. This is what this function does. It maps a number from one range to another. This means that low values get mapped to low values in another corresponding range, high values get mapped to high values and values in between get mapped to their corresponding in between values. The map() function only deals with integers and thus will not generate fractions or irrational numbers. This means that fractions are truncated or

rounded off to whole numbers. The syntax of the map() function is as shown below

map(value, fromLow, fromHigh, toLow, toHigh)

y = map(x, 50, 50, 1);

The function could also be used in mapping negative numbers as well:

y = map(x, 1, 50, 50, -100);

The parameter value is the number you want to map.

The parameter fromLow is the lower bound of the current range of the value.

The parameter fromHigh is the upper bound of the current range of the value.

The parameter toLow is the lower bound of the target range of the value.

The parameter toHigh is the upper bound of the target range of the value.

Take a look at this example code below

```
// map an analog value to 8 bits (0 to 255)
void setup( )  { }
void loop( )  {
      int val = analogRead(0);
      val = map(val, 0, 1023, 0, 255);
      analogWrite(9, val);
}
```

A simple note of warning: it has already been stated that this function suppresses fractions and tends to round them off. So, if you are working on a project which requires precise calculations (such as getting a voltage to the accuracy of three decimal places), it may be better if you do not use this function. You could make use of manual calculations which should give you accurate results up to three decimal places.

The max( ) function

The max( ) function, as the name implies is used to calculate the maximum of two numbers. It is used to make a relative comparison of two numbers.

The syntax of the function is as follows

```
max(x, y)
```

The parameter “x” is the first number. The data types allowed are any data type.

The second parameter “y” is the second number. The allowed data types is also any data type.

The max( ) returns the larger of the two compared numbers.

The min( ) function

Just as the max( ) function calculates the maximum of two numbers, the min( ) function calculates the minimum of two number under comparison. The syntax of the min( ) function is as shown below

```
min(x, y)
```

The parameter x is the first number and the data types allowed are any data type.

Also, the second parameter “y” is the second number and the data types allowed are any data type as well. The min( ) function returns the smaller of the two numbers under comparison.

The pow( ) function.

The pow( ) function is what calculates the value of a number raised to a particular power. The pow( ) is used to raise a number to a particular power. It can also be used to raise a number to a fractional power. The pow( ) function is used in generating exponential mapping of values and curves. The syntax for the function is shown below

```
Pow(base,  exponent)
```

The parameter “base” is the number you want to raise to a particular power. It allows the float data types. Consider the float data type to be in the set of real numbers. This simply means that the float allows for real numbers such as fractions. Just as the int. data type allows for only integers.

The parameter “exponent” is the power to which you raised the base. The allowed data type is also float. Look at the sample code below

```
z = pow(x,      y)
```

The sq( ) function

The sq( ) function calculates the square of a number. This means that the sq( ) function performs an operation which multiplies a number by itself once. The syntax of the sq( ) function is as shown below

```
Sq( x )
```

The parameter “x” is the number. It allows any data type. The function returns the square of a number that has been input.

The sqrt( ) function

The sqrt( ) function calculates the square root of a number. The syntax of the sqrt( ) function is as shown below

```
sqrt( x )
```

The parameter “x” is the number which we want to find the square root. The allowed data types is any data type.

The trigonometric functions

The cos( ) function

The cos( ) function calculates the cosine of an angle in radians. The result will always lie between -1 and +1. The syntax is as shown below

```
cos( rad )
```

The parameter “rad” is the angle in radians. The allowed data type is float. Take note that you should always input the value of the angle expressed in radians. The function returns the cos of the angle in radians.

The sin( ) function

The sin( ) function calculates the sin of an angle expressed in radians. The result will also be between -1 and +1. The syntax is as shown below

```
sin( rad )
```

The parameter “rad” is the value of the angle you want to calculate its sine, expressed in radians. The sin( ) function returns the sine of the value of the angle expressed in radians, the allowed data types are the float data types.

The tan( ) function

The tan( ) function is used to calculate the tangent of an angle expressed in radians. The result of this calculation is between negative infinity and positive infinity. The syntax of the function is as shown below

```
tan( rad )
```

The parameter “rad” as usual, is the value of the angle you want to find the tangent of, expressed in radians. The allowed data type is the float data types. The function returns the tangent of the value of the angle expressed in radians.

The character functions

The isAlpha( ) function

The isAlpha( ) is used to analyze and determine if the character is a letter. It returns True is the character (thisChar) contains a letter. The syntax is as shown below

```
isAlpha(thisChar)
```

The parameter thisChar is the variable under analysis. The allowable data type is the char data type. This means that the function only allows characters. The function returns True if the character is an alphabet (if thisChar is an alpha). Take a look at the example code below

```
if (isAlpha(myChar)       {this line of code tests if myChar is a letter
        serial.printIn( “the character is a letter”);
}
else     {
        serial.printIn( “this character is not a letter” );
}
```

The isAlphanumeric( ) function

This function is what analyzes and determines if a character is a letter or number (that is: char is alphanumeric). The function returns True if thisChar has a letter or number. The syntax is as show below

```
isAlphaNumeric( thisChar )
```

the parameter thisChar is the variable under analysis. The data type

allowed is the char data type. The function returns True if thisChar is alphanumeric.

Take a look at the example code below

```
if (isAlphaNumeric(myChar)      {this line of code tests if myChar is a letter or a number
        serial.printIn( “the character is alphanumeric”);
}
else    {
        serial.printIn( “this character is not alphanumeric” );
}
```

The isAscii( ) function

This function is what analyses if a character is an Ascii character. It returns True if the character is an Ascii character (that is, if thisChar is Ascii). The syntax is as shown below

isAscii(thisChar)

The parameter “thisChar” is a variable and it allows only the data type char. The function returns a True if thisChar is Ascii. Take a look at the example code below

```
if (isAlpha(myChar)      {this line of code tests if myChar is an Ascii character
        serial.printIn( “the character is a Ascii”);
}
else    {
        serial.printIn( “this character is not an Ascii character” );
}
```

The isControl( ) function

The isControl( ) function is used to analyze and determine if a character is a control character. A control character is a character that is not printed. They do not occupy a printing position on display and are the opposite of a printable character. The syntax is as shown below

```
isControl ( thisChar )
```

The parameter “thisChar” is a variable and the allowed data type is the

char data type. The function returns a True if the character is a control character. Take a look at the sample code below:

```
if (isControl(myChar)     {this line of code tests if myChar is a control character
        serial.printIn( “the character is a control character”);
}
else     {
          serial.printIn( “this character is not a control character” );
}
```

The isDigit( ) function

This function analyses and determines if a character is a number (a digit). The function returns True if the character is a number (if thisChar is a digit). The syntax is as shown below:

```
isDigit( thisChar )
```

The parameter “thisChar” is the variable and the allowed data type is the char. If thisChar is a number, the function returns True. Take a look at the example code below

```
if (isDigit(myChar)        {this line of code tests if myChar is a number
        serial.printIn( “the character is a digit”);
}
else     {
        serial.printIn( “this character is not a digit” );
}
```

The isGraph( ) function

This function analyses and determines if a character is printable with some content (take an empty space for instance, it is printable, but it has no content). This function returns True if the character (thisChar) is printable. The syntax is as shown below

```
isGraph( thischar )
```

The parameter thisChar is a variable and the allowed data type is the char data type. if thisChar is printable The function returns True. Take a look at the example code below

```
if (isGraph(myChar)       {this line of code tests if myChar is a printable character, yet not an empty space
        serial.printIn( “the character is a printable character”);
}
else      {
        serial.printIn( “this character is not a printable character” );
}
```

The isHexadecimalDigit( )

This function analyses and determines if a function is a hexadecimal digit (that is: A-F, 0-9). The function returns true if the character contains a hexadecimal digit (if thisChar is hexadecimal). The syntax is as shown below

```
IsHexadecimalDDigit( thisChar )
```

The parameter “thisChar” is a variable and the allowed data type is the char data type. It returns True if the character is hexadecimal. Take a look at the example code below

```
if (isHexadecimalDigit( myChar) {this line of code tests if myChar is a hexadecimal digit
          serial.printIn( “the character is a hexadecimal digit”);
}
else      {
          serial.printIn( “this character is not a hexadecimal digit” );
}
```

The isLowerCase( ) function

This function is used to analyze if a character is lower case letter. The function returns True if the character contains a lower case letter. The syntax is as shown below

```
isLowerCase( thisChar )
```

The parameter “thisChar” is a variable and the data types it allows is the char data type. The function returns True if thisChar is a letter in the lower case. Take a look at the example code below

```
if (isLowerCase ( myChar)          {this line of code tests if myChar is a letter in the lower case
          serial.printIn( “the character is a lower case letter”);
```

```
}
else    {
        serial.printIn( “this character is not a lower case letter” );
}
```

The isPrintable( ) function

This function analyses and determines if a character is printable (this means as long as the character produces an output, even if it is a blank space). The function returns True if the character is printable. The syntax of the code is as shown below

```
isPrintable( thisChar )
```

The parameter thisChar is a variable and the data types it allows is the char data type. It also returns a true if the character is printable. Take a look at the sample code below

```
if (isPrintable( myChar ) {this line of code tests if myChar is a character that is printable
      serial.printIn( “the character is a printable character”);
}
else    {
      serial.printIn( “this character is not a printable character” );
}
```

The isPunct( ) function

The isPunct( ) function is used to analyse and determine if a character is a punctuation (a comma, a period, a semicolon, a colon and so on). The function returns true if the character is a punctuation. Take a look at the syntax below

```
isPunct( thisChar )
```

The parameter “thisChar” is a variable and the data type that is allowed is the char data type. The function isPunct( ) returns a True if the character is a punctuation. Take a look at the sample code below

```
if (isPunct( myChar )    {this line of code tests if myChar is a punctuation character
     serial.printIn( “the character is a punctuation character”);
}
```

```
else     {
      serial.printIn( “this character is not a punctuation character” );
}
```

The isSpace( ) function

The isSpace( ) analyses and determines if a character is a white space character. The function returns True if the argument is a space, newline (“\n”), form feed (“\f”), carriage return(“\r”), vertical tab (“\v”) or horizontal tab (“\t”). The syntax of the function is as shown below

```
IsSpace( thisChar )
```

The parameter “thisChar” is a variable and the data type it allows is the char data type. It returns True if the character is a white space character. The sample code is as shown below

```
if (isSpace( myChar )     {this line of code tests if myChar is a white space character
       serial.printIn( “the character is a white space character”);
}
else     {
       serial.printIn( “this character is not a white space character” );
}
```

The isUpperCase( ) function

This function analyses and determines whether a character (that is, a letter) is in the upper case. If the character is in the upper case, it returns True. Take a look at the syntax below

```
isUpperCase( thisChar )
```

The parameter “thisChar” is a variable and it only allows the char data types. The function returns True if the character is upper case. Take a look at the example code below

```
if (isUpperCase ( myChar )        {this line of code tests if myChar is a character that is upper case
         serial.printIn( “the character is an upper case character”);
}
else     {
              serial.printIn( “this character is not an upper case character” );
```

```
}
```

The isWhitespace( ) function

This function analyses and determines if the character is a space character. It returns a true if a character is a horizontal tab (“\t”). The syntax is as shown below

```
isWhitespace( thisChar )
```

The parameter “thisChar” is a variable and the data type it allows is the char data type. The function returns a True if the character is a space character. Take a look at this sample code below

```
if (isWhitespace ( myChar )        {this line of code tests if myChar is a space character
        serial.printIn( “the character is a space character”);
}
else      {
         serial.printIn( “this character is not a space character” );
}
```

The random number functions

The random( ) function

The random function generates numbers that are seemingly random. The syntax is as shown below

```
random(max)
random(min, max)
```

The parameter “min” is the lower bound of the random value and the parameter “max” is the upper bound of the random value. A random number returns a number between the min and the max. The function accepts long data type. Take a look at the example code below

```
long randNumber
void setup( )     {
        Serial.begin(9600);
        randomSeed(analogRead(0));
}
```

```
void loop( )      {
        // print a random number between 0 and 299
        randNumber = random( 300 );
        Serial.printIn(randNumber);
        // instruction to print a random number between 10 and 19
        randNumber = random(10, 20);
        serial.printIn(randomNumber);
        delay(50);
}
```

The randomSeed( ) function

The randomSeed( ) functions resets the random number generator and causes it to begin at a random point. The randomSeed( ) function is used in situations where you want the starting points of a random sequence to be different. The function initializes random number generator with a random input. The syntax of the function is as shown below

```
randomSeed(seed)
```

The parameter “seed” is the number that will reset the pseudo-random sequence. It allows unsigned long data types. Take a look at the example code to see how the function is used.

```
long randNumber
void setup( )     {
        Serial.begin(9600);
        randomSeed(analogRead(0));
}
void loop( )      {
        randNumber = random( 300 );
        Serial.printIn(randNumber);
        delay(50);
}
```

Bits and bytes functions

The bit( ) function

This function computes the value of a particular bit (bit 0 is 1 and bit 1 is 2, bit 2 is 4 and bit 4 is 16 and so on). The syntax of the function is as shown below

```
bit(n)
```

The parameter “n” is the bit which you want to compute its value.

The bitClear( ) function

This function clears (or replaces with a zero) a bit of a numeric variable. The syntax is as shown below

```
bitClear(x,        n)
```

the parameter “x” is the number which you want to clear its bit.

The parameter “n” indicates the bit to be cleared. Starting from zero for the rightmost or least significant bit.

The bitRead( ) function

The function reads a bit of a number. The syntax is shown below

```
bitRead(x, n)
```

The parameter “x” is the number from which you want to read its bit.

The parameter “n” is the bit to be read. It starts at zero for the rightmost or least significant bit.

The bitSet( ) function

This function writes a 1 to (or sets) a bit of a numeric variable. The syntax is as shown below

```
bitSet(x,          n)
```

The parameter “x” is the numeric variable you want to set its bit

The parameter “n” is the bit to be set starting from zero for the rightmost or least significant bit.

The bitWrite( ) function

This function writes a bit of a numeric variable. The syntax is as shown below

bitWrite(x, n, b)

The parameter “x” is the numeric variable you want to set its bit.

The parameter “n” is the bit to be written starting from the rightmost or least significant bit.

The parameter “b” is the value you want to write to the bit (either 0 or 1).

The highByte( ) function

This function extracts the leftmost (higher order) byte of a word or second least byte of a larger data type. The syntax is as shown below

highByte(x)

The parameter x is the value of any type.

The lowByte( ) function

This function extracts the rightmost (lower order) byte of a variable. A word for example. The syntax is as shown below

lowByte(x)

the parameter x is a value of any type.

The external interrupts functions

The attachInterrupt( ) function

This function is used to create an external interrupt (an interrupt is an input signal which indicates that an even needs urgent attention. first parameter to be included in the attachInterrupt( ) function is an interrupt number. The syntax is as shown below

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode). This is the recommended syntax

attachInterrupt(interrupt, ISR, mode). This syntax is not recommended.

attchInterrupt(pin, ISR, mode). This syntax is not recommended, also, it does not work on all Arduino boards

The parameter “interrupt” is the number of interrupts and the data type allowed is the int. data type.

The parameter “pin” is the pin number of the Arduino board.

The parameter “ISR” is the interrupt service routine to call when the interrupt occurs. This function must not take any parameters and must return

nothing. The ISR function is also called the interrupt service routine function.

The parameter “mode” is what defines when the interrupt should be triggered. There are four constants which have been predefined as valid values.

LOW: this is to set off the interrupt when the pin is low

CHANGE: this is to set off the interrupt whenever the value of the pin changes.

RISING: this will be set off when the pin rises from low to high

FALLING: this will be set off when the pin falls from high to low.

Some boards such as the DUE, MRK1000 and Zero boards allow:

HIGH: this will be set off when the pin is high.

Take a look at the sample code below

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}
void blink() {
  state = !state;
}
```

Interrupt Numbers,

The detachInterrupt( ) function

This function does the opposite of what the previous function does, it turns

off the interrupt. Consider the syntax below

detachInterrupt(digitalPinToInterrupt(pin)). This is the recommended syntax

detachInterrupt(interrupt). This syntax is not recommended.

detachInterrupt(pin). This syntax is not recommended, also, it does not work on all Arduino boards

The parameter “interrupt” is the number of interrupts needed to be disabled.

The parameter “pin” is the pin number of the Arduino to disable.

The Communication functions

The serial functions

This function is used for communicating between the Arduino board and other devices. A serial port is found in all Arduino boards. The serial function has the following functions.

The if(Serial) function is what indicates if the Serial function is ready. The syntax is if(Serial)

The available( ) function is used to obtain the number of bytes available for reading from a serial port. The syntax is serial.available( ) and it returns the number of bytes that are available to read.

The availableForWrite( ) function is used to obtain the number of bytes or characters available for writing into the serial buffer without blocking the operation (write). The syntax is Serial.availableFor Write( ).

Other serial functions are begin( ), end( ), find( ), findUntil( ), flush( ), parseFloat( ), parseInt( ), peek( ), print( ), printIn( ), read( ), readBytes( ), readBytesUntil( ), readString( ), readStringUntil( ), setTimeout( ), write( ), serialEvent( ).

The stream functions

The stream function is not a directly called function, it is however invoked whenever a function that relies on it is used. In Arduino, it defines the reading functions. The libraries that rely on the stream function include: Serial, Wire, Ethernet and SD. The functions that are related to the stream function are: available( ), read( ), flush( ), find( ), findUntil( ), peek( ), readBytes( ), readBytesUntil( ), readString( ), readStringUntil( ), parseInt( ), parseFloat( ), setTimeout( ).

The USB functions

The keyboard functions.

The keyboard functions are Keyboard.begin( ), Keyboard.end( ), Keyboard.press( ), Keyboard.print( ), Keyboard.printIn( ), Keyboard.release( ), Keyoard.releaseAll( ), Keyboard.write( ).

The mouse functions.

The mouse functions are: Mouse,begin( ), Mouse.click( ), Mouse.end( ), Mouse.move( ), Mouse.press( ), Mouse.release( ), Mouse.isPressed( ).

These functions which we have taken a lot of time to discuss are not the only functions in Arduino. The fact that Arduino is open source has also led to its dynamism. Good ideas from spirited members of different platforms are usually incorporated and added to the Arduino libraries. New functions are also included regularly into the newer versions that have been released. Also, the functions mention here have just been explained in passing so that the reader can become acquainted to the basic jobs these functions can do. As you keep advancing in your studies in Arduino, these basic functions of these functions (pun intended) that have been explained will prove very valuable when you refer back to them.

CHAPTER SIX: CONCLUSIONS

In the beginning of this book, it was made very clear to the reader that this book is a purely introductory course into Arduino. It has accomplished its goal of gently introducing the reader into the subject of Arduino. While reading this book, you may not have to pick up any Arduino board to practice, but the concepts which this book will introduce you to will make it very easy for you to pick up a board and practice, if you want to practice. This book serves as a general introduction to Arduino. It is also an introduction for people who have no initial exposure to electronics, or to technology in general. In reading this book, you have learnt all you need to learn so that you can become better acquainted with the general concepts in Arduino. You can hold an intelligent conversation with anyone on the subject of Arduino, the subject will no longer be an obscure one to you, and you now have the capacity to grasp more complicated concepts which will be introduced to. Thus, if you want to pursue a long-term study in Arduino, this book will afford you the entryway to enter the world of mainstream Arduino.

However, this book does not claim to have exposed every introductory concept satisfactorily. This would imply that there may some concepts which you may require some more explanations before you grasp them. So, it is advised that as you are reading this book, keep your internet close by, so that you search online, any concept which feel has not been well explained.

You have indeed come a long way to have stumbled upon Arduino, this book will be a soft landing for you as you begin to explore the world of Arduino. Good luck and all the best as you enlighten yourself with the knowledge of Arduino.